BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 461 - ARTIFICIAL INTELLIGENCE

TERM PROJECT REPORT

BIFKE - SOLVING NY TIMES CROSSWORD PUZZLE

Members: A. Fırat Uyar 21602930
İpeksu Tutsak 21501551
Koral Yıldıran 21000292
Muhammed Burak Görmüş 21602797
Salih Efe Boyacı 21802629

Section: 2
Instructor: Prof. Dr. Varol Akman

25.12.2020

TABLE OF CONTENT

1.0 INTRODUCTION

People have different approaches to solve crossword puzzles such as looking up a dictionary or guessing them. Even if some solve them entertainingly, some are not interested or hate them. The difficulty level of finding the solutions might dissuade people from solving them. Some puzzles challenge people because understanding the clue or finding the correct answer might be rough. Furthermore, since there might be more than one possible answer, finding the appropriate one is a challenging task. This is because all answers affect others. Therefore, our term project is trying to make the work of the people, who solve the NY Times crossword puzzle created by Joel Fagliano, easier using Artificial Intelligence [1]. This crossword puzzle contains a 5x5 grid, and our program is trying to give correct answers corresponding to each clue shown in the grid. Our program is written in Python due to the extensive availability of sources to generate answers.

2.0 PROBLEM APPROACH

As said earlier, the purpose of the project is to find correct answers for the NY Times crossword puzzle. Solving a crossword puzzle is a two-step process generating a candidate answers list corresponding to each clue and choosing the most suitable one among them [2]. In order to create a candidate answers list, we first conducted research on the sources that we can use. After that, we determined Wordnet, Wikipedia, the word2vec project, and PyDictionary as our sources. We, then, discussed the possible ways to examine the clues and make a relation between the sources that we found and the clues. As the second part of our project, we needed to choose an answer among candidates. Therefore, we tried different algorithms to keep the correct answer in the candidate list while determining the solution.

3.0 RESOURCES

As mentioned earlier, we used 4 different sources to create a candidate list for each clue. These sources are, namely, Wordnet, word2vector project, Wikipedia, and PyDictionary. The reason why we decided to use Wordnet is that it allows us to obtain extensive information about words such as synonyms, hypernyms, and hyponyms. Even though it allows us to reach broad information, it is also observed that it sometimes gives irrelevant candidate answers. For our second source, we used the word2vector project provided by Google. This source

allows us to create a candidate answer list based on the vector value of the given clue. The third source used is Wikipedia, and we observed that it is the one that allows us to obtain the most correct answers if the clue is related to proper nouns. For the last source, we decided to employ the PyDictionary so that we can reach the definitions, synonyms and antonyms of the words in the given clue.

3.1 WordNet

Wordnet is one of the sources that can be used in natural language processing. The reason why it is so commonly used is that it allows us to reach different information about the word that we are trying to study. Synonyms, hypernyms, and hyponyms are some of the aspects that we can reach about the word [3]. Hypernyms are basically the superwords, and hyponyms are the subnames that provide semantic relations between words [3]. Further, using the NLTK module allows us to use different built-in functions such as pos tag and word tokenize. In our code, we firstly tokenize the clue. After that, we are checking for each word whether it is in the stopwords list. Subsequently, we are finding the synonyms, hypernyms, and hyponyms of each word in the clue. Basically, the function that we wrote for Wordnet gives us a list containing the synonyms, hypernyms, and hyponyms of each word in the clue. This list also provides the source of data which is Wordnet and the base value for each candidate's answers which is equal to one. However, it should be noted that WordNet lacks “pop culture references” and “not popular phrases” [4].

3.2 Word2Vector

One of the interesting methods that can be used in natural language processing is to use vector representation of the words provided by Google word2vec project. The pre-trained model that we use contains 3 million phrases and words represented by 300 dimensional vectors [5]. By using word2vec, it is possible to obtain a number of answers closest to the given word. The distance is determined by the cosine similarity of the word vectors [5]. For example, if the difference of the vectors Paris and France is summed by the vector Italy, it should give the answer Rome [6]. In our project, we employed this source with a similar approach that we followed for previous sources. Firstly, we tokenize the clue, and remove the words if they are in the stopword list. After that, we are summing the vector values of the words left and reaching a vector value again. Lastly, we are finding the most similar 20

candidate answers to the final summed vector value. These 20 candidate answers are also checked whether they only contain letters because it is observed that some candidates may not contain alphabetical characters. In summary, the function that we wrote for Google word2vec source, returns a list that contains the candidate answers which are closest to the vector value calculated. Also, this function provides the source of data and the base value for each candidate answers which is equal to one.

3.3 Wikipedia

One of the built-in modules in the Python programming language that allows us to reach information about words is the Wikipedia module. Therefore, we decided to use it as our third source. In the function that we wrote for Wikipedia source, we are using the search and summary functions. The search function takes the clue and tokenizes the clue. After tokenizing, we are removing the words if they are in the stopwords list. Subsequently, we are searching Wikipedia for each word through the search function and we are storing the words that appear after that search. The second part of the code runs if there is a string in quotation marks. The reason is that we observed that the string in quotation marks that appears in clues are mostly proper nouns. We are taking this string in quotation marks, and using the summary function. After obtaining the summary about this string, we are adding the words appearing in the first sentence of the summary. In short, the code that we wrote for Wikipedia returns a list of words that appears when we search the words. Also, it provides the source and base value information like previously used functions.

3.4. PyDictionary

Solving the mini crossword puzzle, the function that searches for the given clue of the puzzle in PyDictionary was created. The dictionary database module in an offline manner, which provides the build files, was used. PyDictionary provides meaning, antonyms, synonyms of the given words. Moreover, from the NLTK module (Natural Language Toolkit) nltk.tokenize was called in order to tokenize the clue, and nltk.corpus was used for removing commonly used words such as "a", "an", "the". Furthermore, isalpha() was used for understanding whether all characters in the string are alphabets. Whether a word is a noun or a verb, the dictionary is able to provide the definition for both noun and verb. However, we used it only for nouns because of the more accurate answers rate. Basically, the function

takes each word of the given clue and the number of letters that are required for the answer blank. After getting the inputs, it returns 3 attributes in a list which are the appropriate answers, the source name as a PyDict, and the power value as 1. If any values are not found from the PyDictionary database, then the empty list will be returned.

4.0 IMPLEMENTATION STEPS

4.1 Scraping & Parsing the Puzzle

To get today's puzzle from the New York Times website an automated browser is used with the selenium module. When the program starts it looks for today's puzzle file if not found it opens up a browser and goes to the New York Times Crossword. In sequential order, it searches for the required buttons to click using a search string such as the "OK" button when the site loads, or the reveal puzzle button. After revealing the puzzle with the same method it gets up/down clues, black squares on the grid, and where the clue numbers are in the grid. Then it writes down them to a dictionary to be saved to file with today's date by a module called Pickle.

4.2 Graphical User Interface (GUI)

Making GUIs with python could be frustrating when using Qt or wx. For this reason, we chose to use Tkinter to draw GUIs. For grid squares and puzzle grid, separate classes were made to avoid drawing 25 squares sometimes with clue numbers for each puzzle. There is today's puzzle grid and answer in the left, across and down clues in the middle, and our solution to the puzzle in the right. When the program is started it looks for today's puzzle file, if not found it uses our scraping module to create one, then our algorithm solves it. These happen when the GUI is initialized but not shown, lastly we show the GUI.

4.3 Candidate Answers List Generation

As mentioned in section 3, lists of candidate answers for each clue is generated by calling the Wordnet, word2Vector, Wikipedia and PyDictionary functions. After calling these functions, the outputs of these functions are combined together in a list for each clue by using get candidates function before starting our algorithm.

4.4 Algorithm For Choosing The Answer

For the sake of our algorithm, we firstly filter the candidate answers, and then implement the search algorithm between the filtered candidate answers. To filter the candidate answers, we determine the rank for each candidate according to the number of intersections of this candidate with other candidate answers. For example, if the candidate answer "Saudi" matches with other candidates twice, its ranking is 2. Since the same candidate answers, sometimes, appear twice in a candidate list coming from different sources, they are combined together under the so-called 'Intersected' source. After that, the ranking for each candidate is completed. Subsequently, we find the best three ranking values for each clue and filter the candidates that match with these best three ranking values.

For our search algorithm, we take the filtered candidate answers as input. Then, the number of occurrences of letters for each blank in the grid is calculated. After that, we ranked the candidate answers according to this calculation. For example, if the blank u in "aura" has a value of 3 because of u appearing from other candidates a third time, the value of aura is updated with 3. Therefore, our heuristic information is based on the number of letters in the blanks. Calculating the ranking value for each candidate, the worst candidates that have a lower ranking value than others in the candidate list are deleted. This implementation continues until the number of candidate answers for each clue drops to one.

5.0 CONCLUSION

Solving a crossword puzzle using Artificial Intelligence requires serious work because it is composed of two steps. First is to create candidate answers lists, and the second is to choose the most suitable one from the candidate answers list [2]. We observed that even though the correct answer is in the candidate answers list, it sometimes is deleted due to the search algorithm. Therefore, the ranking of candidate answers and search algorithms should be improved to increase the efficiency of crossword puzzle solving tasks. For example, using A* algorithm could be useful as a heuristic search technique; however, choosing to use A* depends on the complexity of the task [7]. The other way to increase the efficiency depends on the sources. Therefore, using a Web search for the clues may also improve the results [7].

Word Count: 2120

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 BIFKE.

REFERENCES

[1] “The New York Times Mini Crossword,” *The New York Times*. [Online]. Available: https://www.nytimes.com/crosswords/game/mini. [Accessed: 25-Dec-2020].

[2] A. Thomas and S. S., “Towards a Semantic Approach for Candidate Answer Generation in Solving Crossword Puzzles,” *Procedia Computer Science*, vol. 171, pp. 2310–2315, 2020.

[3] Z. Gong, C. W. Cheang, and U. L. Hou, “Web Query Expansion by WordNet,” *Lecture Notes in Computer Science Database and Expert Systems Applications*, pp. 166–175, 2005.

[4] A. R. Jobin, A. G. Menon, A. Sekhar, and V. Damodaran, “Key to crossword solving : NLP,” *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017.

[5] “Google Code Archive - Long-term storage for Google Code Project Hosting.,” *Google*. [Online]. Available: https://code.google.com/archive/p/word2vec/. [Accessed: 24-Dec-2020].

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” Sep. 2013.

[7] G. Angelini, M. Ernandes, and M. Gori, “Webcrow: A Web-Based Crosswords Solver,” *Lecture Notes in Computer Science Intelligent Technologies for Interactive Entertainment*, pp. 295–298, 2005.

APPENDIX

Appendix A: Source Code

nycCrosswordScraper.py

```
from selenium import webdriver as wd
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.expected_conditions import
presence_of_element_located
import pickle
import time
import sys
from datetime import datetime

url = 'https://www.nytimes.com/crosswords/game/mini'
chrome_driver_path = '/Users/muhammedburakgormus/Desktop/chromedriver'
webdriver = None

def open_browser(silent=False):
        print("Open Browser")

        global webdriver
        chrome_options = Options()
        if silent:
        chrome_options.add_argument('--headless')

        webdriver = wd.Chrome(executable_path=chrome_driver_path,
options=chrome_options)
        # webdriver = wd.Chrome(options=chrome_options)

        # Init Scraper
        wait = WebDriverWait(webdriver, 2)
def open_url():
        print("Load URL")

        global webdriver
        webdriver.get(url)
def close_browser():
        print("Close Browser")
```

```
        global webdriver
        webdriver.close()


across_clues = []
down_clues = []
grid = []
answer_grid = []


def scrape():
        global webdriver
        global across_clues, down_clues, grid, answer_grid

        answer_grid = ["" for a in range(25)]
        grid = ["" for a in range(25)]

        print("Press OK Button")

        ok_button = []
        # Show Puzzle
        while len(ok_button) == 0:
        ok_button = webdriver.find_elements_by_xpath("//button[@aria-label='OK']")

        while len(ok_button) == 1:
        try:
        ok_button[0].click()
        except:
        pass
        ok_button = webdriver.find_elements_by_xpath("//button[@aria-label='OK']")

        print("Get clues")

        # Scrape Clues
        across = webdriver.find_elements_by_xpath("//div[h3[contains(@class,
'ClueList-title')]]/ol")[0].text.split("\n")
        down = webdriver.find_elements_by_xpath("//div[h3[contains(@class,
'ClueList-title')]]/ol")[1].text.split("\n")

        i = 0
        while i < len(across) - 1:
        # print(across[i], across[i + 1])
```

```
        across_clues.append((int(across[i]), across[i + 1]))
        i += 2
        i = 0
        while i < len(down) - 1:
        # print(down[i], down[i + 1])
        down_clues.append((int(down[i]), down[i + 1]))
        i += 2

        print("Across Clues:", across_clues)
        print("Down Clues:", down_clues)

        print("Reveal puzzle")

        # Scrape Answers
        # Show Answers
        reveal_button =
webdriver.find_elements_by_xpath("//button[@aria-label='reveal']")[0]
        reveal_button.click()
        revealPuzzleButton =
webdriver.find_elements_by_xpath("//button[@aria-label='reveal']/following-sibling::ul/li/a
[text()='Puzzle']")[0]
        revealPuzzleButton.click()
        reveal_confirm_button =
webdriver.find_elements_by_xpath("//button[@aria-label='Reveal']")[0]
        reveal_confirm_button.click()
        cross_button = webdriver.find_elements_by_xpath("//span[contains(@class,
'ModalBody')]")[0]
        cross_button.click()

        print("Save grid squares and answers")

        # Parse Answers
        cells = webdriver.find_elements_by_xpath("//*[local-name() = 'g' and
@data-group='cells']//*[local-name() = 'g']")

        for i in range(len(cells)):

        start_element = cells[i].find_elements_by_xpath(".//*[local-name() = 'text' and
@text-anchor='start']")
        if len(start_element) > 0:
        grid[i] = int(start_element[0].text)
```

```
            middle_element = cells[i].find_elements_by_xpath(".//*[local-name() = 'text' and
@text-anchor='middle']")
            if len(middle_element) > 0:
            char = middle_element[0].text
            answer_grid[i] = char
            else:
            answer_grid[i] = "-"
            grid[i] = "-"

            print("Grid", grid)
            print("Answer Grid", answer_grid)

            time.sleep(10)


def save(filename="default"):
            if filename == "default":
            now = datetime.now()
            filename = now.strftime("%d-%m-%Y.save")

            print("Saving answers, clues and grid to", filename)

            # Save for later use
            saveDict = {"clues_across": across_clues, "clues_down": down_clues, "grid": grid,
"grid_answers": answer_grid}
            with open(filename, "wb") as saveFile:
            pickle.dump(saveDict, saveFile)
            return filename


def main(step_callback=None):

            open_browser()
            open_url()
            scrape()
            save_file = save()
            close_browser()
            return save_file


if __name__ == '__main__':
            main()
```

Puzzle_reader.py

```
import pickle
class puzzle:
  def __init__(self, filename):
    with open(filename, "rb") as saveFile:
      self.data = pickle.load(saveFile)
    self.clues_across = self.data["clues_across"]
    self.clues_down = self.data["clues_down"]
    self.grid = self.data["grid"]
    self.grid_answers = self.data["grid_answers"]
    self.across_answer_lengths = []
    self.down_answer_lengths = []
    self.grid_2D = []
    self.intersections = [[[] for n in range(5)] for i in range(5)]
    # short example
    # [[[('Across', 1, 0), ('Down', 1, 0)], [('Across', 1, 1), ('Down', 2, 0)]]]
    # 5 by 5 array inside there is intersecting words (clue direction, clue number, character
index starting from 0)
    # empty if it is black square

    for i in range(len(self.grid)):
      if(i % 5 == 0):
        self.grid_2D.append([])
      self.grid_2D[i//5].append(self.grid[i])

    for x in range(5):
      for y in range(5):
        if(self.grid_2D[y][x] == "-"):
          continue
        for xn in range(x+1):
```

```
            if(type(self.grid_2D[y][xn]) == type(int(0))):
                #direction, clue_num, nth character
                self.intersections[y][x].append(("Across", self.grid_2D[y][xn], x-xn))
                break

        for yn in range(y+1):
            if(type(self.grid_2D[yn][x]) == type(int(0))):
                self.intersections[y][x].append(("Down", self.grid_2D[yn][x], y-yn))
                break

for x in range(5):
    for y in range(5):
        if(type(self.grid_2D[y][x]) == type(int(0))):
            for clue in self.clues_across:
                if(clue[0] == self.grid_2D[y][x]):
                    lenght = 0
                    for i in range(5):
                        if(x+i > 4):
                            break
                        if(self.grid_2D[y][x+i] == "-"):
                            break
                        lenght += 1
                    self.across_answer_lengths.append((self.grid_2D[y][x], lenght))
                    break

            for clue in self.clues_down:
                if(clue[0] == self.grid_2D[y][x]):
                    lenght = 0
                    for i in range(5):
                        if(y+i > 4):
                            break
                        if(self.grid_2D[y+i][x] == "-"):
```

```
                    break
                lenght += 1
            self.down_answer_lengths.append((self.grid_2D[y][x], lenght))
            break
```

---

nycCrosswordSolver.py

```
from wordnet import get_wordnet
from wikisearch import get_wikipedia
from word2vec import get_word2vector
from PytonDictionary import dictionary

def solve(intersection_boxes, across_len, down_len, across_clues,down_clues):

  dict_puzzle = get_candidates(across_clues,across_len,down_clues,down_len)
  ranked_puzzle_dict =
ranking_candidates(intersection_boxes,across_clues,across_len,down_clues,down_len,dict_
puzzle)
  removed_dict = checking_occurences(ranked_puzzle_dict)
  filtered_puzzle_dict = select__best_candidates_for_puzzle(removed_dict)
  grid_letters =
get_grid_letters(intersection_boxes,across_clues,across_len,down_clues,down_len,filtered_
puzzle_dict)
  ranked_sol =
point_rank(filtered_puzzle_dict,grid_letters,intersection_boxes,across_clues,down_clues)
  final =
one_candidate(ranked_sol,grid_letters,intersection_boxes,across_clues,across_len,down_cl
ues,down_len)
  #print(final)
  return final

##########################################################################
```

```
#########################################################################
def get_candidates(across_clues,across_len,down_clues,down_len):
  cand_dict = {}
  for i in down_clues:
    number = i[0]
    clue = i[1]
    for j in down_len:
      if number == j[0]:
        candidate_list = []
        clue_len = j[1]
        try:
          wiki_list = get_wikipedia(clue,clue_len)
          wordnet_list = get_wordnet(clue,clue_len)
          vec_list = get_word2vector(clue,clue_len)
          py_list = dictionary(clue,clue_len)
          [candidate_list.append(i) for i in wiki_list]
          [candidate_list.append(i) for i in wordnet_list]
          [candidate_list.append(i) for i in vec_list]
          [candidate_list.append(i) for i in py_list]
          cand_dict[clue] = candidate_list
        except:
          cand_dict[clue] = []
  for i in across_clues:
    number = i[0]
    clue = i[1]
    for j in across_len:
      if number == j[0]:
        candidate_list = []
        clue_len = j[1]
        try:
          wiki_list = get_wikipedia(clue,clue_len)
          wordnet_list = get_wordnet(clue,clue_len)
```

```
            vec_list = get_word2vector(clue,clue_len)
            py_list = dictionary(clue,clue_len)
            [candidate_list.append(i) for i in wiki_list]
            [candidate_list.append(i) for i in wordnet_list]
            [candidate_list.append(i) for i in vec_list]
            [candidate_list.append(i) for i in py_list]
            cand_dict[clue] = candidate_list
        except:
            cand_dict[clue] = []
  return cand_dict
##########################################################################
##########################################################################
def
ranking_candidates(intersection_boxes,across_clues,across_len,down_clues,down_len,dict_
puzzle):
  for i in intersection_boxes:
    for j in i:
      if len(j) != 0:
        first_el = j[0]
        second_el = j[1]
        type_first = first_el[0]
        type_second = second_el[0]
        number_first = first_el[1]
        number_second = second_el[1]
        letter_first_index = first_el[2]
        letter_second_index = second_el[2]
        if type_first == "Across":
          for k in across_clues:
            if k[0] == number_first:
              clue_across = [k][0][1]
              clue_across_answers = dict_puzzle[clue_across]
        if type_second == "Down":
```

```
            for u in down_clues:
                if u[0] == number_second:
                    clue_down = [u][0][1]
                    clue_down_answers = dict_puzzle[clue_down]
        for across_answers in clue_across_answers:
            for down_answers in clue_down_answers:
                if across_answers[0][letter_first_index] ==
down_answers[0][letter_second_index]:
                    new_across_val = across_answers[2] + 1
                    across_answers[2] = new_across_val
                    down_answers[2] = down_answers[2] + 1
  return dict_puzzle
#############################################################################
#############################################################################
def checking_occurences(puzzle_dict):
  #checking whether the same word appears from different sources;
  #if it is the case, summing the ranks up and making it one element
  for i in puzzle_dict:
    words = []
    for j in puzzle_dict[i]:
      if j[0] not in words:
        words.append(j[0])
      else:
        puzzle_dict[i].remove(j)
        for k in puzzle_dict:
          for m in puzzle_dict[k]:
            if m[0] == j[0]:
              puzzle_dict[k].append([j[0],"intersected",j[2] * 2])
              puzzle_dict[k].remove(m)
  return puzzle_dict
#############################################################################
#############################################################################
```

```
def select_best_candidates(candidate_list):
  ranking_list = []
  best_candidates = []
  for clues in candidate_list:
     ranking_list.append(clues[2])
  ranking_list.sort(reverse=True)

  best_3_scores = ranking_list[0:3]
  for clues in candidate_list:
     if clues[2] in best_3_scores:
        best_candidates.append(clues)
  return best_candidates

def select__best_candidates_for_puzzle(dict_puzzle):
  for clues in dict_puzzle:
     best_clues = select_best_candidates(dict_puzzle[clues])
     dict_puzzle.update({clues:best_clues})
  return dict_puzzle
#############################################################################
#############################################################################
def
get_grid_letters(intersection_boxes,across_clues,across_len,down_clues,down_len,dict_puz
zle):
  grid_letters_dict = {}
  counter = 0
  for i in intersection_boxes:
     letters = {}
     for j in i:
        counter = counter + 1
        if j:
           first_el = j[0]
```

```
        second_el = j[1]
        type_first = first_el[0]
        type_second = second_el[0]
        number_first = first_el[1]
        number_second = second_el[1]
        letter_first_index = first_el[2]
        letter_second_index = second_el[2]
        if type_first == "Across":
            for k in across_clues:
                if k[0] == number_first:
                    clue_across = [k][0][1]
                    clue_across_answers = dict_puzzle[clue_across]
        if type_second == "Down":
            for u in down_clues:
                if u[0] == number_second:
                    clue_down = [u][0][1]
                    clue_down_answers = dict_puzzle[clue_down]
        for across_answers in clue_across_answers:
            if across_answers[0][letter_first_index] not in letters:
                letters.update({ across_answers[0][letter_first_index]:1})
            else:
                new_val = letters[across_answers[0][letter_first_index]] + 1
                letters.update({ across_answers[0][letter_first_index]:new_val})
        for down_answers in clue_down_answers:
            if down_answers[0][letter_second_index] not in letters:
                letters.update({ down_answers[0][letter_second_index]:1})
            else:
                new_val = letters[down_answers[0][letter_second_index]] + 1
                letters.update({ down_answers[0][letter_second_index]:new_val})
        grid_letters_dict.update({counter:letters})
    else:
        grid_letters_dict.update({counter:[]})
```

```
  return grid_letters_dict
##############################################################################
##############################################################################
def point_rank(dict_puzzle,grid_letters,intersection_boxes,across_clues,down_clues):
  last_point_dict ={}
  counter= 0
  for i in intersection_boxes:
    for j in i:
      counter = counter + 1
      if j:
        first_el = j[0]
        second_el = j[1]
        type_first = first_el[0]
        type_second = second_el[0]
        number_first = first_el[1]
        number_second = second_el[1]
        letter_first_index = first_el[2]
        letter_second_index = second_el[2]
        if type_first == "Across":
          for k in across_clues:
            if k[0] == number_first:
              clue_across = [k][0][1]
              clue_across_answers = dict_puzzle[clue_across]
        if type_second == "Down":
          for u in down_clues:
            if u[0] == number_second:
              clue_down = [u][0][1]
              clue_down_answers = dict_puzzle[clue_down]
        for across_answers in clue_across_answers:
          point = grid_letters[counter][across_answers[0][letter_first_index]]
          if across_answers[0] not in last_point_dict:
            last_point_dict.update({across_answers[0]:point})
```

```
                else:
                    x = last_point_dict[across_answers[0]] + point
                    last_point_dict.update({across_answers[0]:x})

            for down_answers in clue_down_answers:
                point = grid_letters[counter][down_answers[0][letter_second_index]]
                if down_answers[0] not in last_point_dict:
                    last_point_dict.update({down_answers[0]:point})
                else:
                    x = last_point_dict[down_answers[0]] + point
                    last_point_dict.update({down_answers[0]:x})

    for i in dict_puzzle:
        for j in dict_puzzle[i]:
            for k in last_point_dict:
                if k == j[0]:
                    j[2] = last_point_dict[k]

    return dict_puzzle
##############################################################################
##############################################################################
def
one_candidate(puzzle_dict,grid_letters,intersection_boxes,across_clues,across_len,down_cl
ues,down_len):
    counter_list = []
    for i in puzzle_dict:
        counter_list.append(len(puzzle_dict[i]))
    for i in puzzle_dict:
        rank_val = []
        if len(puzzle_dict[i])>1:
            for j in puzzle_dict[i]:
                rank_val.append(j[2])
```

```
        min_val = min(rank_val)
        for k in puzzle_dict[i]:
            if k[2] == min_val:
                puzzle_dict[i].remove(k)
        new_dict= puzzle_dict
        grid_letters =
get_grid_letters(intersection_boxes,across_clues,across_len,down_clues,down_len,new_dic
t)
        ranked_new_dict =
point_rank(new_dict,grid_letters,intersection_boxes,across_clues,down_clues)

one_candidate(ranked_new_dict,grid_letters,intersection_boxes,across_clues,across_len,d
own_clues,down_len)
  return puzzle_dict
```

---

nycCrosswordGUI.py

```
import tkinter as tk
from tkinter import filedialog
import tkinter.ttk as ttk
from datetime import datetime
import textwrap
import pickle
import nycCrosswordScraper
import os
import nycCrosswordSolver
import puzzle_reader

class Application(tk.Frame):
  def __init__(self, master=None):
    super().__init__(master)
    self.master = master
```

```
        self.grid()
        self.create_puzzle_widget()
        self.create_other_widgets()
        self.create_solution_widget()

    def create_other_widgets(self):
        # Buttons
        # self.force_scrape_button = tk.Button(self, text="Force Refresh",
command=scrape_new)
        # self.force_scrape_button.grid(column=5, row=1)
        # self.load_answers_button = tk.Button(self, text="Load Answers",
command=browse_files)
        # self.load_answers_button.grid(column=5, row=2)
        # self.load_answers_button = tk.Button(self, text="Solve", command=solve_puzzle)
        # self.load_answers_button.grid(column=5, row=3)
        # Date and group label
        self.date_label = tk.Label(self, text="BIFKE\n" + now.strftime("%H:%M\n%d/%m/%Y"))
        self.date_label.grid(column=6, row=10)
        self.update_time()

        # File name label
        # self.filename_label = tk.Label(self, text=open_file_name)
        # self.filename_label.grid(column=0, row=10)

    def update_time(self):
        now = datetime.now()
        self.date_label.configure(text="BIFKE\n" + now.strftime("%H:%M\n%d/%m/%Y"))
        self.master.after(1000, self.update_time)

    def create_puzzle_widget(self):
        global puzzle
```

```
        self.puzzle_grid = PuzzleGrid(puzzle.get("grid_answers"), puzzle.get("grid"), master=self)

        self.puzzle_grid.grid(column=0, row=0, rowspan=10, columnspan=1)

        space = tk.Label(self, text=" ")
        space.grid(column=1, row=0)

        # Across Clues
        across_label = tk.Label(self, text="Across Clues")
        across_label.grid(column=2, row=0)

        across_clues_string = ""
        across_clues = puzzle.get("clues_across")
        for i in range(len(across_clues)):
            across_clues_string += textwrap.fill(str(across_clues[i][0]) + " " + across_clues[i][1],
25) + "\n\n"

        across_clues_text = tk.Text(self, width=25, height=18)
        across_clues_text.insert(tk.END, across_clues_string)
        across_clues_text.grid(column=2, row=1, rowspan=9)

        space = tk.Label(self, text=" ")
        space.grid(column=3, row=0)

        # Down Clues
        down_label = tk.Label(self, text="Down Clues")
        down_label.grid(column=4, row=0)

        down_clues_string = ""
        down_clues = puzzle.get("clues_down")
        for i in range(len(down_clues)):
```

```
            down_clues_string += textwrap.fill(str(down_clues[i][0]) + " " + down_clues[i][1], 25)+
"\n\n"

        down_clues_text = tk.Text(self, width=25, height=18)
        down_clues_text.insert(tk.END, down_clues_string)
        down_clues_text.grid(column=4, row=1, rowspan=9)

        space = tk.Label(self, text=" ")
        space.grid(column=5, row=0)

    def create_solution_widget(self, solution=["   " for i in range(25)]):
        self.solution_grid = PuzzleGrid(solution, puzzle.get("grid"), master=self)

        self.solution_grid.grid(column=6, row=0, rowspan=10, columnspan=1)

class PuzzleGrid(tk.Frame):
    def __init__(self, grid_answers, grid_info, master=None):
        super().__init__(master)
        self.master = master

        self.grid_answers = grid_answers
        self.grid_info = grid_info

        self.gridObjects = []

        for i in range(5):
            for j in range(5):
                self.columnconfigure(i, weight=1, minsize=50)
                self.rowconfigure(j, weight=1, minsize=50)
                grid_frame = self.GridSquare(self, grid_answer=str(grid_answers[i * 5 + j]),
                                             clue_number=str(grid_info[i * 5 + j]))
                grid_frame.grid(row=i, column=j, sticky=tk.N + tk.S + tk.E + tk.W)
```

```
            self.gridObjects.append(grid_frame)

    class GridSquare(tk.Frame):
        def __init__(self, master=None, grid_answer="   ", clue_number="-"):
            super().__init__(master)
            self.answer = grid_answer
            self.clue_number = clue_number
            self.master = master

            self.configure(bd=1, relief=tk.SOLID)

            self.columnconfigure(0, weight=1, minsize=1)
            self.columnconfigure(1, weight=1, minsize=1.5)
            self.rowconfigure(0, weight=1, minsize=1)
            self.rowconfigure(1, weight=1, minsize=1.5)

            if self.answer != "-" and clue_number != "-":
                self._num_label_frame = tk.Frame(self)
                self._num_label_frame.grid(row=0, column=0, sticky=tk.N + tk.S + tk.E + tk.W)
                self.num_label = tk.Label(self._num_label_frame, text=self.clue_number)
                self.num_label.grid(row=0, column=0)

                self._answer_label_frame = tk.Frame(self)
                self._answer_label_frame.grid(row=1, column=1, sticky=tk.N + tk.S + tk.E + tk.W)
                self.answer_label = tk.Label(self._answer_label_frame, text=self.answer,
font=("Helvetica", 20))
                self.answer_label.grid(row=0, column=0)

                self._empty_label_frame = tk.Frame(self)
                self._empty_label_frame.grid(row=1, column=2)
                self._empty_label = tk.Label(self._empty_label_frame, text="    ")
                self._empty_label.grid(row=0, column=0)
```

```
            else:
                self.configure(bg="black")

        def set_answer(self, answer):
            self.answer_label['text'] = answer

def load_puzzle(filename="default"):
    if filename == "default":
        filename = now.strftime("%d-%m-%Y.save")
    with open(filename, "rb") as file:
        puzzle_info = pickle.load(file)
        return puzzle_info

def reload_puzzle():
    global app, open_file_name, puzzle
    puzzle = load_puzzle(filename=open_file_name)
    app.puzzle_grid.destroy()
    app.create_puzzle_widget()
    app.solution_grid.destroy()
    app.create_solution_widget()

def reload_solution(solution):
    global app
    app.solution_grid.destroy()
    app.create_solution_widget(solution=solution)

    for i in range(25):
        solution_obj = app.solution_grid.gridObjects[i]
        puzzle_obj = app.puzzle_grid.gridObjects[i]
        # print(solution_obj.answer, puzzle_obj.answer)
        # if(solution_obj.answer != puzzle_obj.answer and "-" != puzzle_obj.answer):
            # solution_obj.num_label.configure(bg="red")
```

```
def scrape_new():
  global open_file_name
  open_file_name = nycCrosswordScraper.main()
  app.filename_label.configure(text=open_file_name.split("/")[-1])
  reload_puzzle()

def browse_files():
  global open_file_name, app
  print("Open file dialog")
  f_name = filedialog.askopenfilename(initialdir=os.getcwd(), title="Select a Puzzle File to
Load",
                                      filetypes=(("Puzzle Files", "*.save"), ("all files", "*.*")))
  if os.path.isfile(f_name):
     open_file_name = f_name
  app.filename_label.configure(text=open_file_name.split("/")[-1])
  reload_puzzle()

def solve_puzzle():
  print("Solving Puzzle")
  puzzle_to_solve = puzzle_reader.puzzle(open_file_name)
  solved_puzzle = nycCrosswordSolver.solve(puzzle_to_solve.intersections,
puzzle_to_solve.across_answer_lengths, puzzle_to_solve.down_answer_lengths,
puzzle_to_solve.clues_across, puzzle_to_solve.clues_down)
  # solved_puzzle ={"Did one's civic duty": [], 'In ___ (holding office)': [], 'Part of the electoral
college': [], 'Poorly behaved': [], '6 a.m. to 9 p.m., for N.Y. polls': [], "#2's on presidential
tickets: Abbr.": [], 'Election day enclosure': ['imran'], 'Engaged in battle': ['boris'], 'Specifics,
slangily': [], 'Beats by ___ (headphones brand)': ['pro']}
  solved_grid = []
  for i in range(25):
     if(puzzle_to_solve.grid == "-"):
```

```
            solved_grid.append("-")
        else:
            solved_grid.append("")


clues = puzzle_to_solve.clues_down + puzzle_to_solve.clues_across

def take_confidance(sol):
    try:
        return solved_puzzle[sol[1]][0][2]
    except IndexError as error:
        return 0

clues.sort(key=take_confidance)

for i in range(len(clues)):
    answer = solved_puzzle[clues[i][1]]
    if(answer == []):
        continue
    is_across = clues[i] in puzzle_to_solve.clues_across

    start_index = -1
    for n in range(25):
        if(puzzle_to_solve.grid[n] == clues[i][0]):
            start_index = n
            break

    if(answer != []):
        for n in range(len(answer[0][0])):
            if(is_across):
                solved_grid[start_index+n] = answer[0][0][n].upper()
            else:
```

```
                solved_grid[start_index+5*n] = answer[0][0][n].upper()
        else:
            if(is_across):
                length = puzzle_to_solve.across_answer_lengths[i][1]
                for n in range(length):
                    try:
                        if(solved_grid[start_index+n*5] == ""):
                            solved_grid[start_index+n*5] = "   "
                    except:
                        pass
            else:
                length = puzzle_to_solve.down_answer_lengths[i][1]
                for n in range(length):
                    try:
                        if(solved_grid[start_index+n] == ""):
                            solved_grid[start_index+n] = "   "
                    except:
                        pass

    reload_solution(solved_grid)

now = datetime.now()
open_file_name = now.strftime("14-12-2020.save")
if not os.path.isfile(open_file_name):
    nycCrosswordScraper.main()

puzzle = load_puzzle(open_file_name)

root = tk.Tk()
app = Application(master=root)
solve_puzzle()
app.mainloop()
```

---

wordnet.py

```
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize

def get_hyper_set(syn_set):
  hyper_set = set()
  for i in syn_set:
    listt = i.hypernyms()
    for y in listt:
      for t in y.lemmas():
        hyper_set.add(t.name())
  return hyper_set

def get_hypo_set(syn_set):
  hypo_set = set()
  for i in syn_set:
    listt = i.hyponyms()
    for y in listt:
      for t in y.lemmas():
        hypo_set.add(t.name())
  return hypo_set

def get_wordnet_each_word(word,length):
  #hyponyms/hypernyms/synonyms
  #iki kelimeliler asad_aslkma şeklinde
  #understood !!!
  syn_set = wordnet.synsets(word)
  name_set = set()
  for i in syn_set:
    for j in i.lemmas():
      name_set.add(j.name())
```

```
    hyper_set1 = get_hyper_set(syn_set)
    hypo_set1 = get_hypo_set(syn_set)

    hyper_set2 = set()
    for i in hyper_set1:
        syn_set_for_i = wordnet.synsets(i)
        for k in get_hyper_set(syn_set_for_i):
            hyper_set2.add(k)

    hypo_set2 = set()
    for i in hypo_set1:
        syn_set_for_i = wordnet.synsets(i)
        for k in get_hypo_set(syn_set_for_i):
            hypo_set2.add(k)

    complete_words = set()
    [complete_words.add(i) for i in name_set if len(i)==length]
    [complete_words.add(i) for i in hyper_set1 if len(i)==length]
    #[complete_words.add(i) for i in hyper_set2 if len(i)==length]
    [complete_words.add(i) for i in hypo_set1 if len(i)==length]
    #[complete_words.add(i) for i in hypo_set2 if len(i)==length]

    return complete_words

def get_wordnet(clue,length):
    name_set = set()
    words_list = word_tokenize(clue)
    for i in words_list:
        for j in get_wordnet_each_word(i,length):
            name_set.add(j)
    word_list = list(name_set)
    last_list = []
```

```
  for i in word_list:
     my_tuple = [i.lower(),"Wordnet",1]
     last_list.append(my_tuple)
  return last_list
```

---

word2vect.py

```
from gensim.models import Word2Vec, KeyedVectors
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import nltk

model =
KeyedVectors.load_word2vec_format("GoogleNews-vectors-negative300.bin",binary=True,l
imit=100000,unicode_errors='ignore')

def get_word2vector(clue,length):

  #taking the clue string and length of the answer
  clue = clue.lower()
  words = word_tokenize(clue)
  filtered_words = []
  stop_words = stopwords.words("english")
  stop_words.append('the')
  for i in words:
     if i not in stop_words:
        filtered_words.append(i)
  #print(filtered_words)
  vector_sum = 0
  for word in filtered_words:
     if word in model:
        vector_sum = vector_sum + model[word]
```

```
  top20models = model.most_similar([vector_sum],topn=20)
  word_set = set()
  for candidates in top20models:
     if len(candidates[0]) == length:
        if candidates[0].isalpha():
           word_set.add(candidates[0])
  word_list = list(word_set)
  last_list = []
  for i in word_list:
     my_tuple = [i.lower(),"Vector",1]
     last_list.append(my_tuple)
  return last_list
```

---

wikisearch.py

```
import re
import wikipedia
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.corpus import stopwords

def get_wikipedia(clue,length):
  stop_words = stopwords.words("english")
  word_set = set()
  try:
     search = wikipedia.search(clue)
     for elements in search:
        words_list = word_tokenize(elements)
        for i in words_list:
           if len(i) == length:
              if i.isalpha():
                 if i not in stop_words:
                    word_set.add(i)
```

```
  except wikipedia.exceptions.DisambiguationError as e:
    pass
  try:
    if re.findall(r'"([^"]*)"', clue):
      special_str = re.findall(r'"([^"]*)"', clue)
      summary = wikipedia.summary(special_str,sentences = 1)
      word_list_2 = word_tokenize(summary)
      for s in word_list_2:
        if len(s) == length:
          if s.isalpha():
            if s not in stop_words:
              word_set.add(s)
  except wikipedia.exceptions.DisambiguationError as e:
    pass
  word_list = list(word_set)
  last_list = []
  for i in word_list:
    my_tuple = [i.lower(),"Wiki",1]
    last_list.append(my_tuple)
  return last_list
```

PythonDictionary.py

```
from PyDictionary import PyDictionary
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

def dictionary(sentence,length):
  empty_list = []
  dictionary = PyDictionary()
  words = sentence.split()
  sw = stopwords.words('english')
```

```
for i in words:
    if i in sw:
        words.remove(i)
try:
    defi = set()
    for element in words:
        element = element.lower()
        definition = dictionary.meaning(element)
        synonyms = dictionary.synonym(element)
        antonyms = dictionary.antonym(element)
        items = definition["Noun"]
        for elements in items:
            words_list = word_tokenize(elements)
            for i in words_list:
                if len(i) == length:
                    if i.isalpha():
                        if i not in sw:
                            defi.add(i)
        for elements in synonyms:
            words_list = word_tokenize(elements)
            for j in words_list:
                if len(j) == length:
                    if j.isalpha():
                        if j not in sw:
                            defi.add(j)

        for elements in antonyms:
            words_list = word_tokenize(elements)
            for k in words_list:
                if len(k) == length:
                    if k.isalpha():
                        if k not in sw:
```

```
                    defi.add(k)
    except:
        pass
    for i in list(defi):
        listt = [i,"PyDict",1]
        empty_list.append(listt)
    return empty_list
```

Appendix B: Test Run

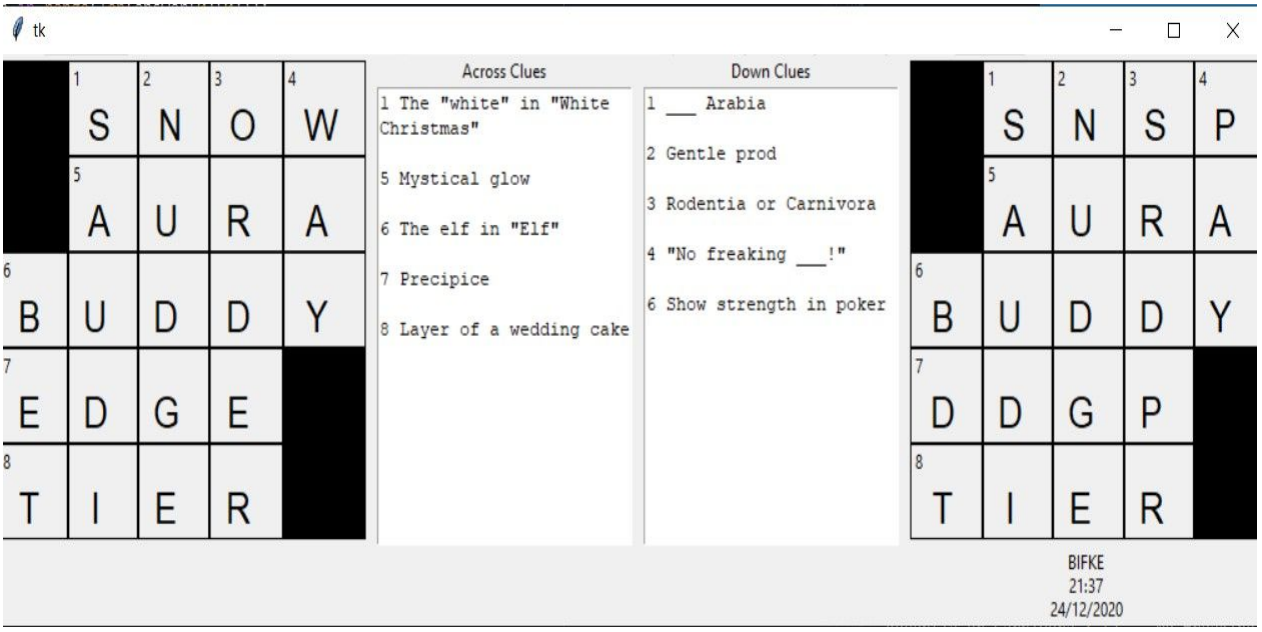

Figure 1: Test run for 24-12-2020

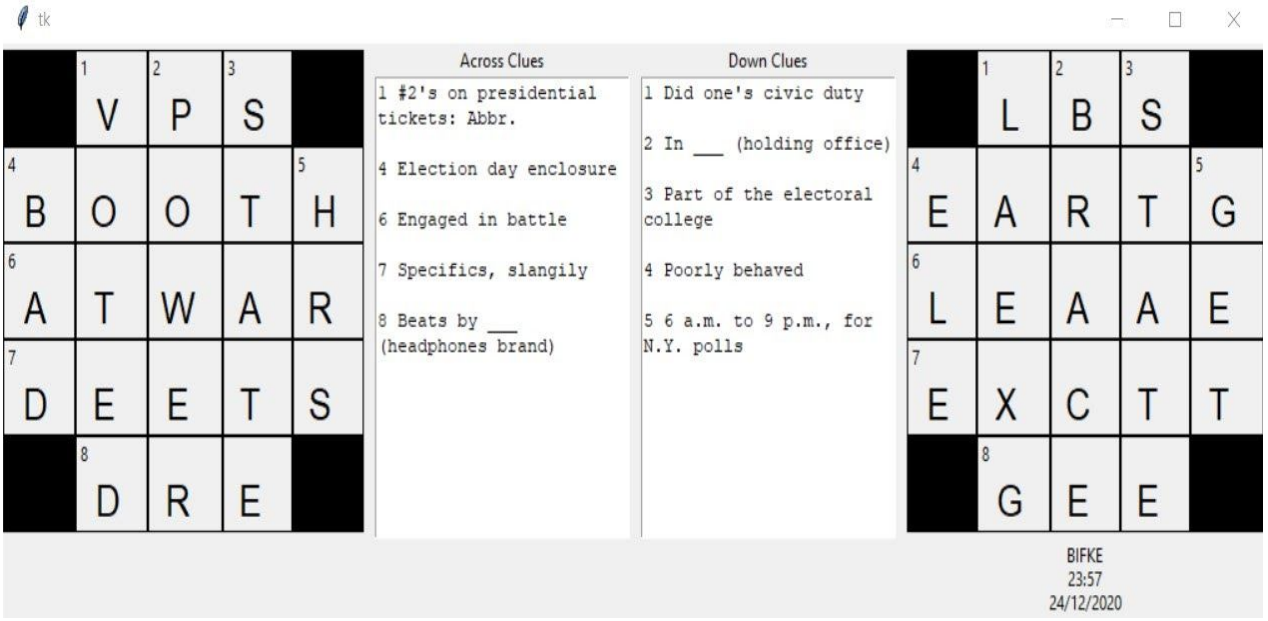

Figure 2: Test run for 03-11-2020

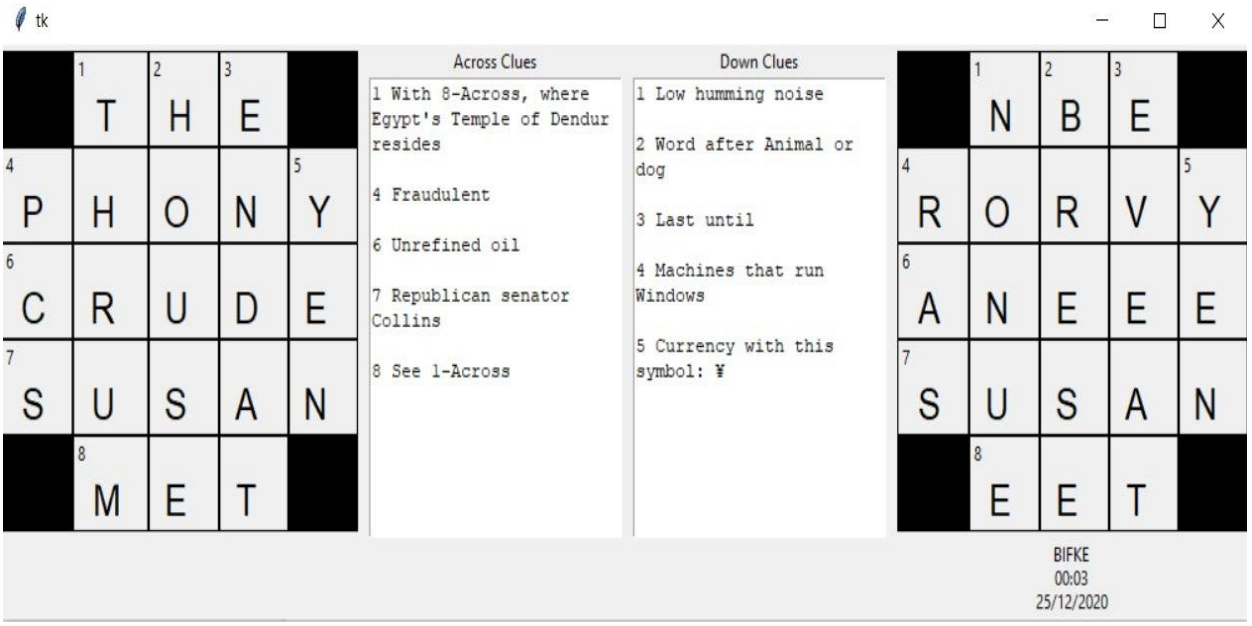

Figure 3: Test run for 04-12-2020

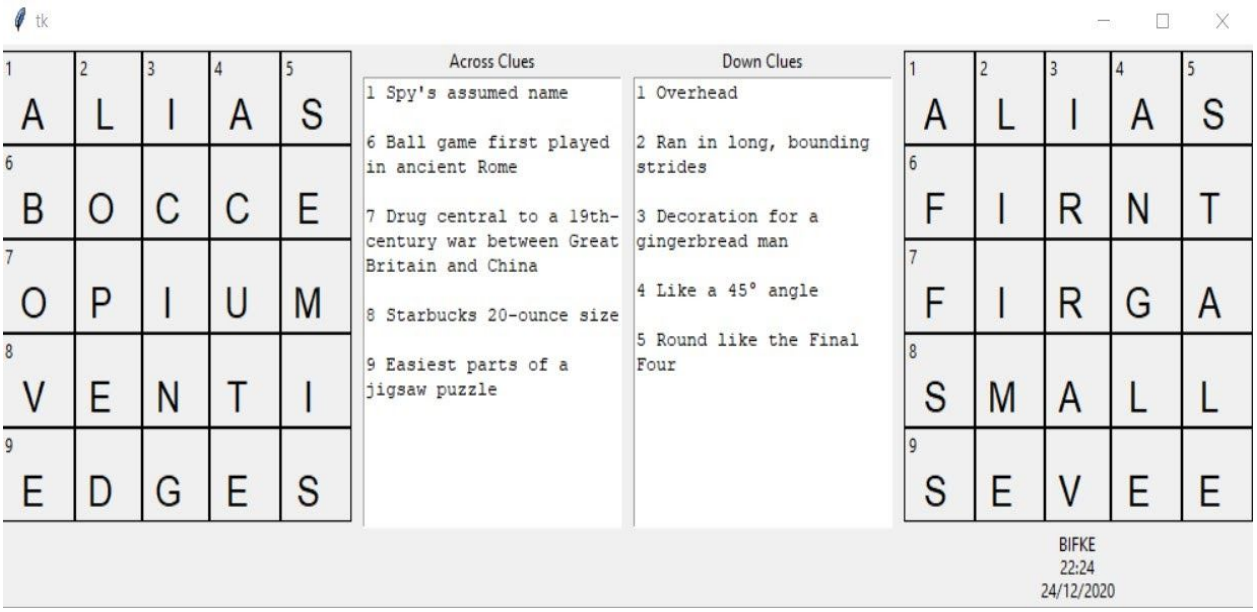

Figure 4: Test run for 30-11-2020

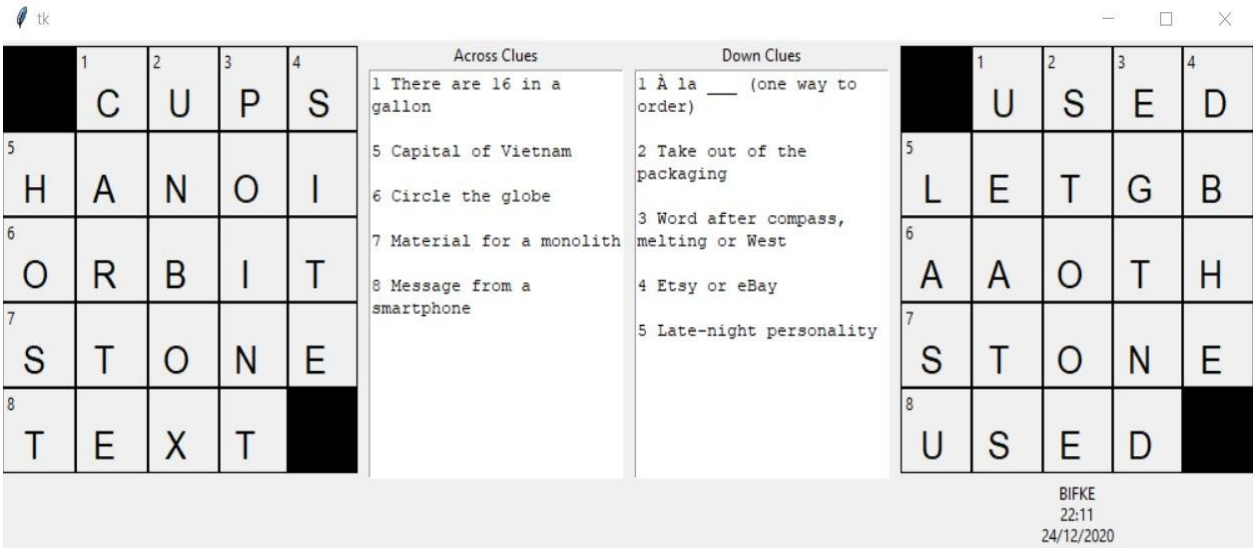

Figure 5: Test run for 16-11-2020

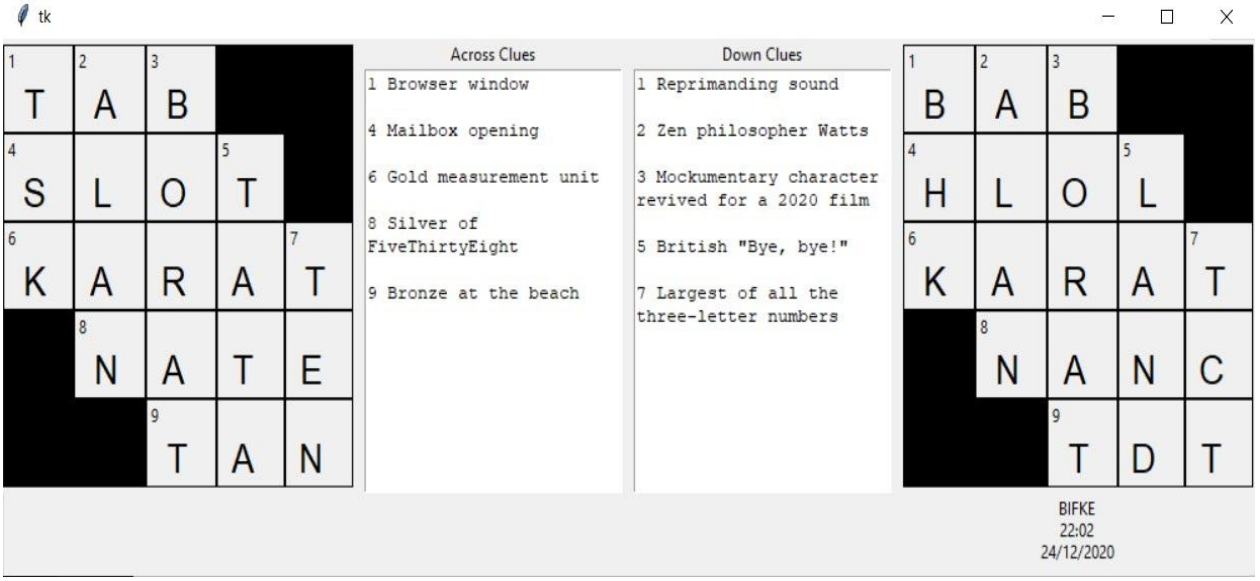

Figure 6: Test run for 09-11-2020

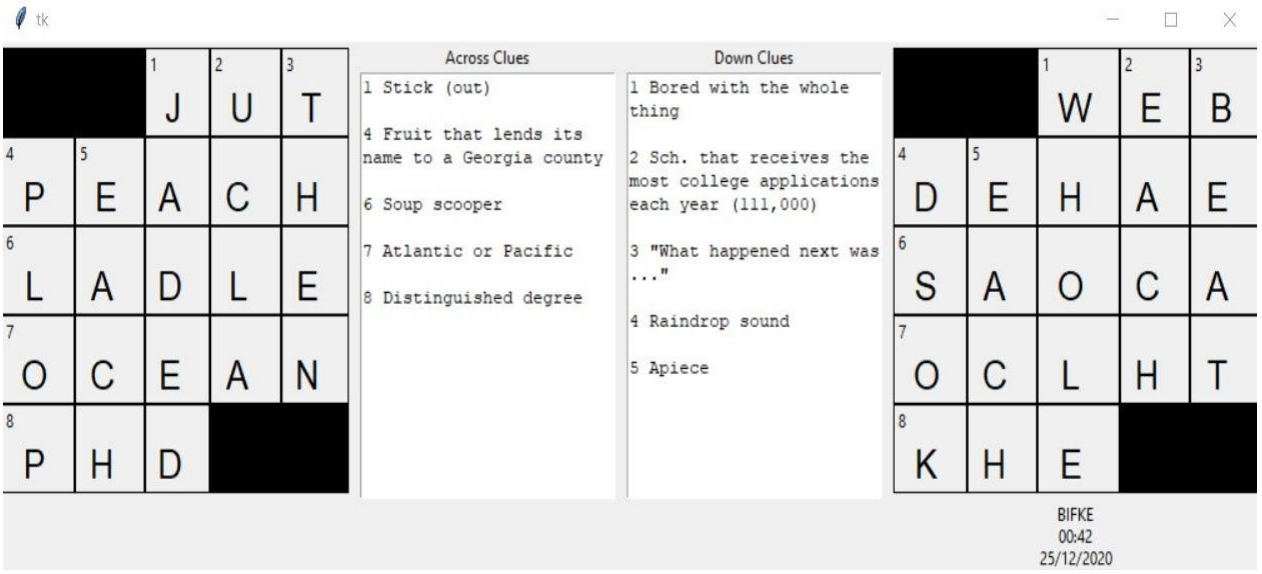

Figure 7: Test run for 13-11-2020

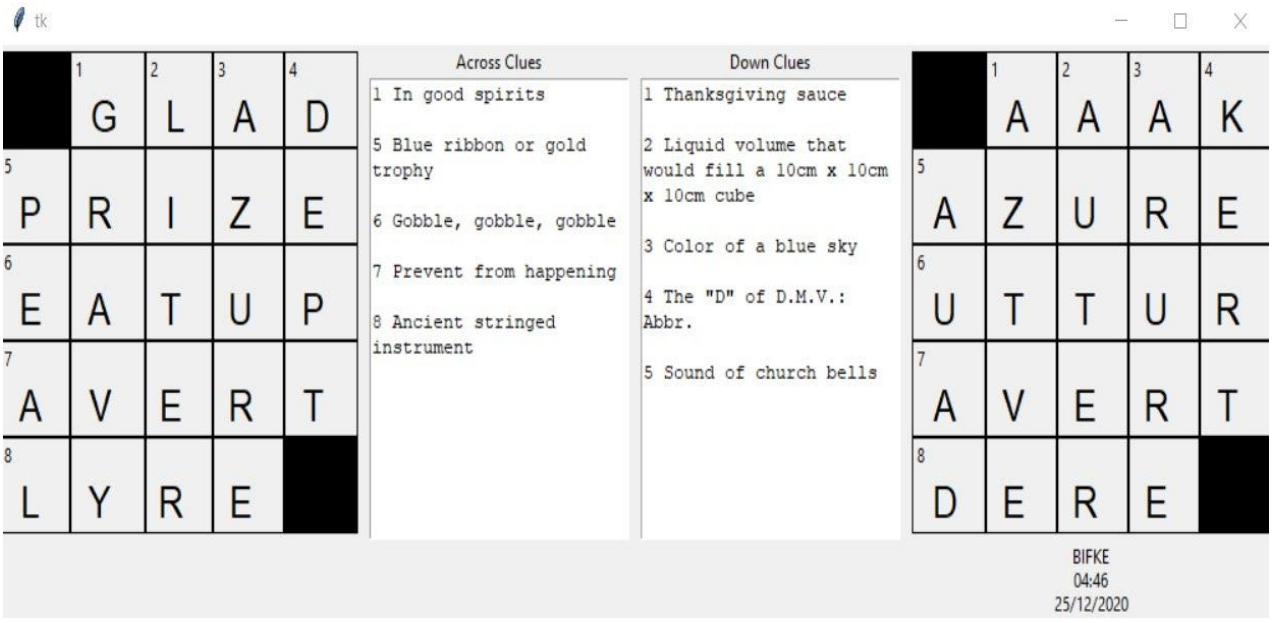

Figure 8: Test run for 26-11-2020

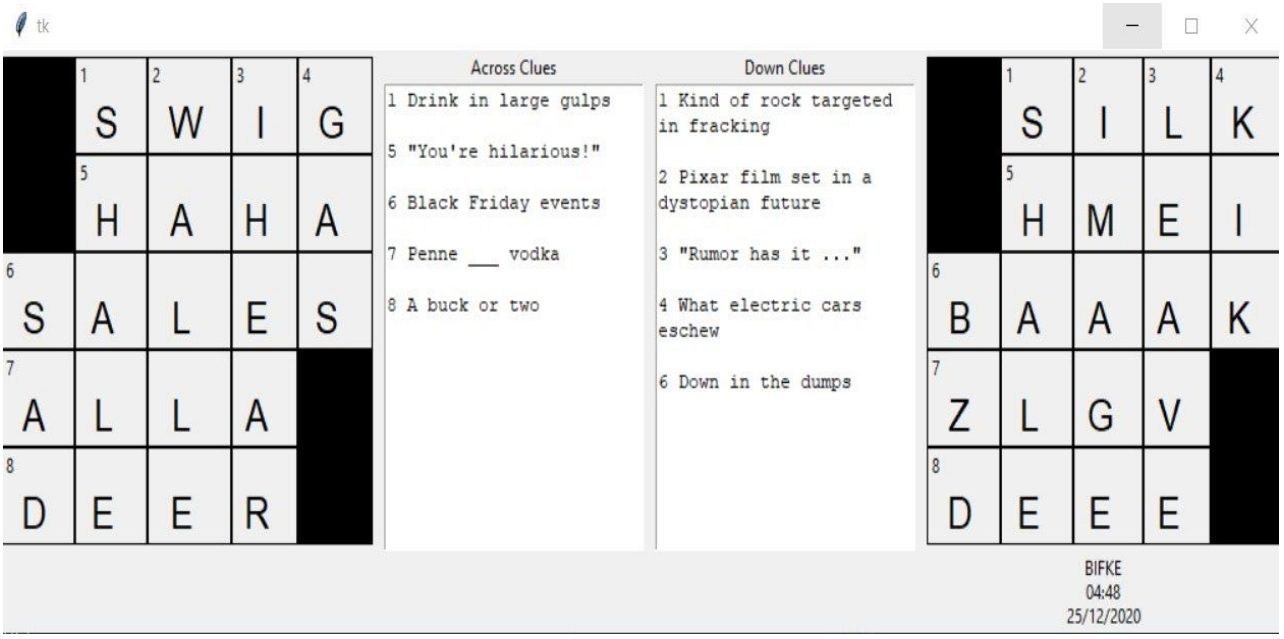

Figure 9: Test run for 29-11-2020

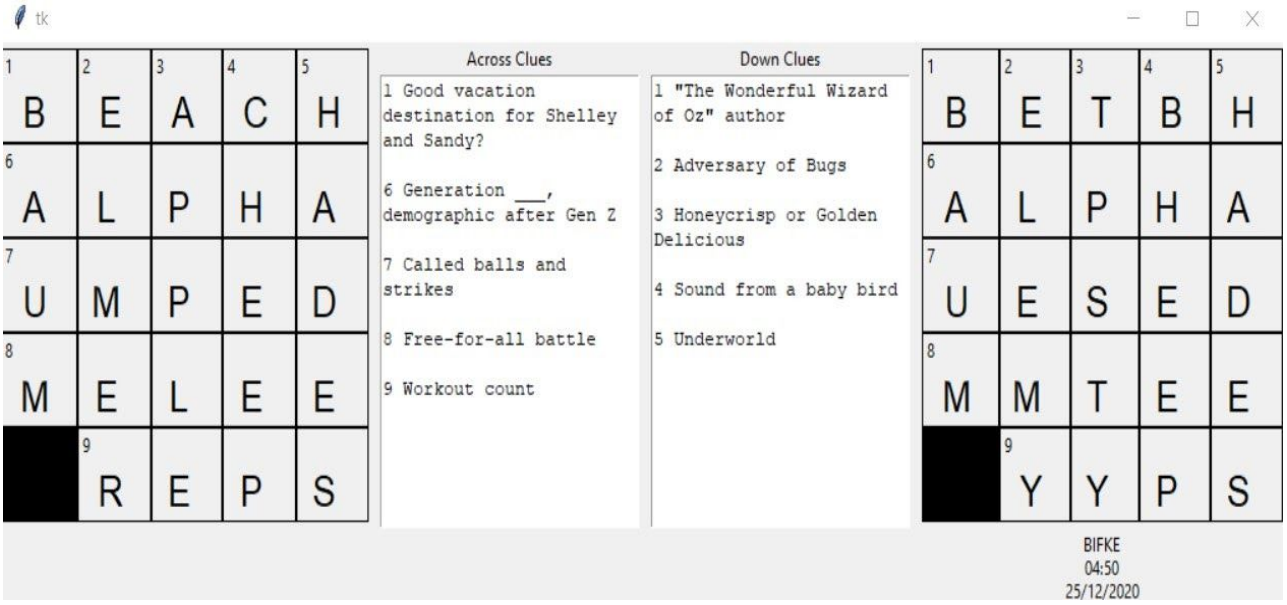

Figure 10: Test run for 11-11-2020

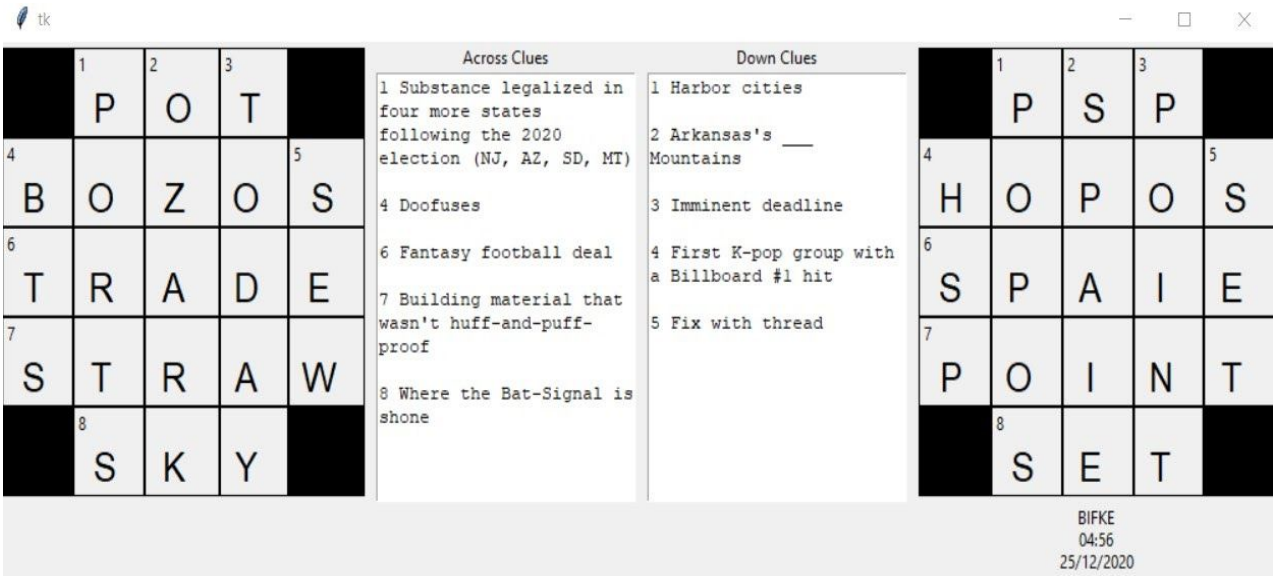

Figure 11: Test run for 18-11-2020

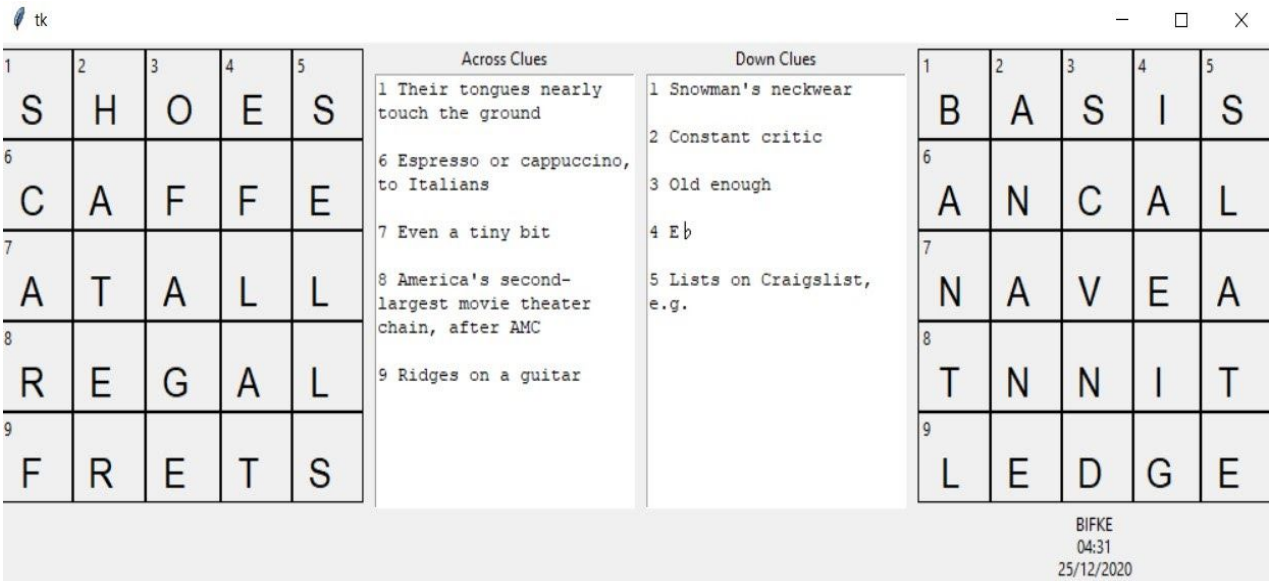

Figure 12: Test run for 05-11-2020

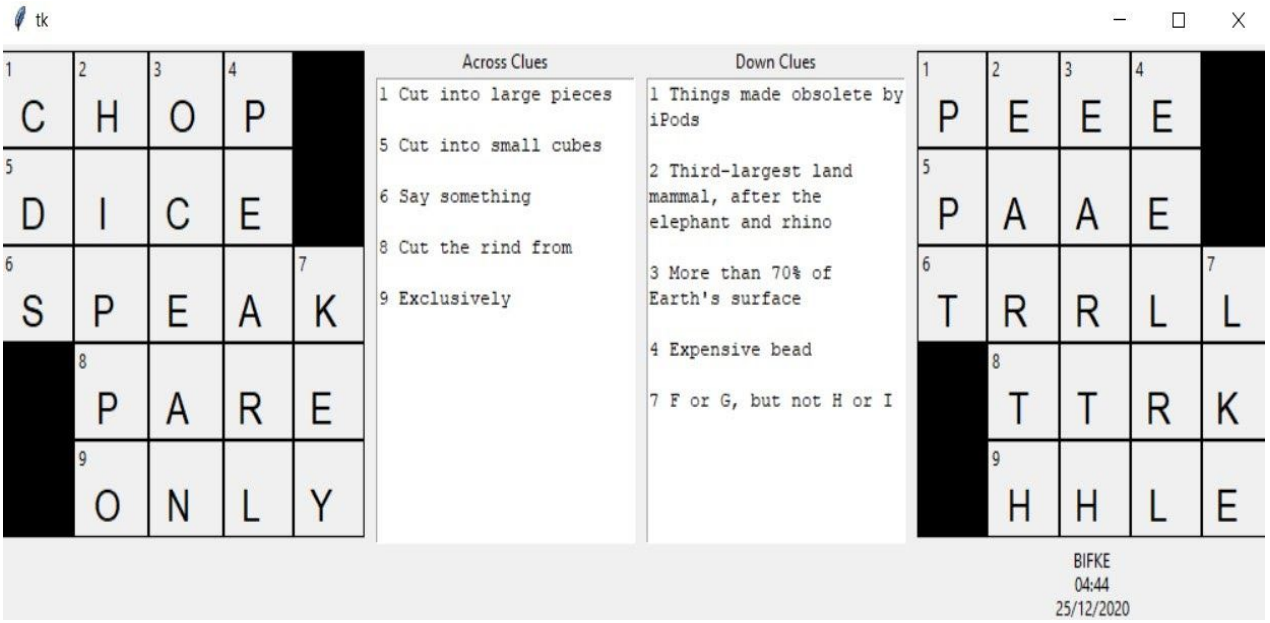

Figure 13: Test run for 25-11-2020

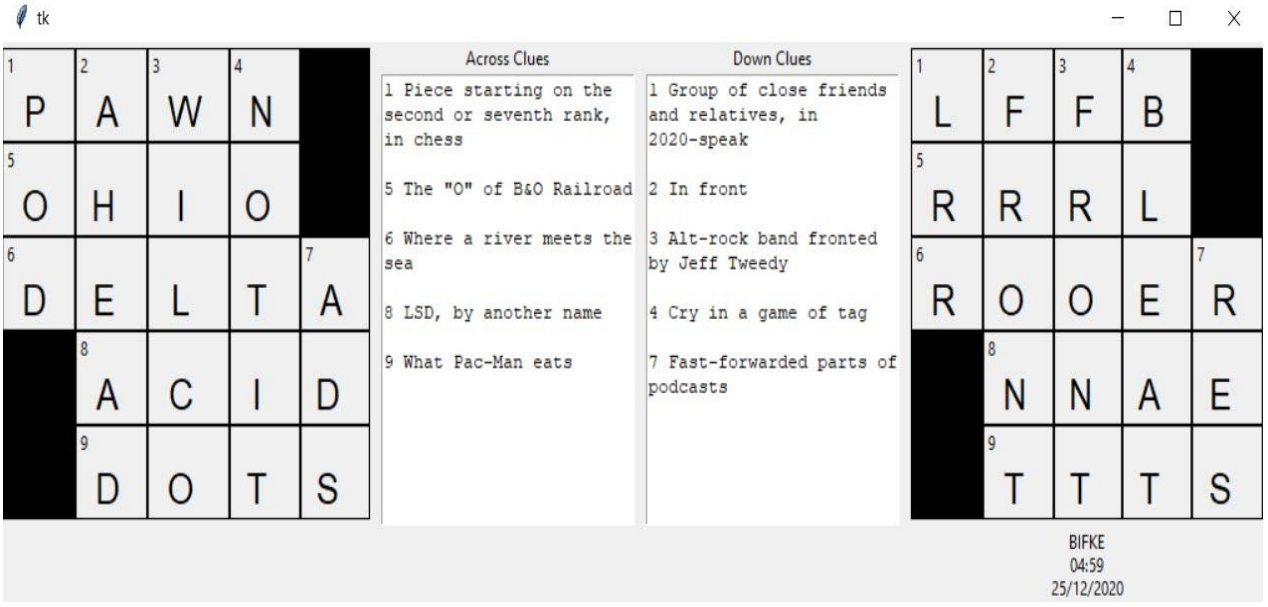

Figure 14: Test run for 10-11-2020

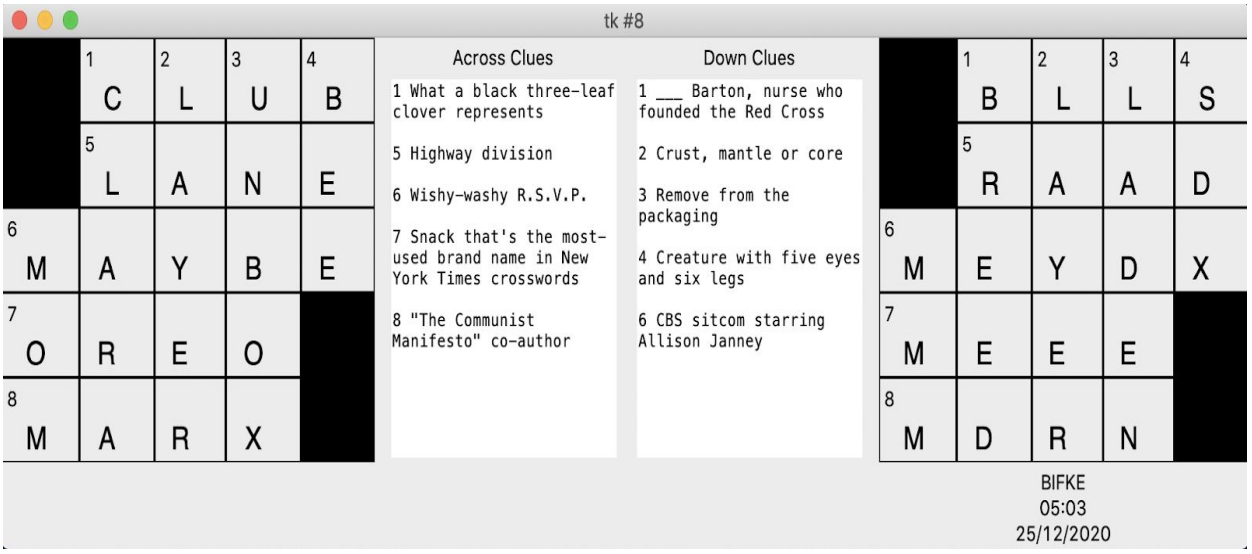

Figure 15: Test run for 17-12-2020

Figure 16: Test run for 25-12-2020