**BILKENT UNIVERSITY**

*Department of Computer Engineering*

*Bilkent 06800 Ankara Turkey*


**SPRING 2021**

# CS 461 ARTIFICIAL INTELLIGENCE

# TERM PROJECT REPORT

**Group: WATSON**

Mehmet Berk Şahin

Mehmet Alper Genç

Ege Hakan Karaağaç

Fırat Yönak

Balaj Saleem

**TABLE OF CONTENTS**

**List of Figures**



**List of Tables**

## 1.0 INTRODUCTION

The New York Times 5x5 Mini Crossword Puzzle was first created by Matt Hural and today, it is daily updated by Joel Fagliano. Every puzzle has 5 down and 5 across clues except Saturday's puzzle, which is the hardest one and out of the scope of this paper. Monday's puzzle is the easiest and difficulty increases each day. The purpose of this project is designing an Artificial Intelligence (AI) to solve the puzzle by using the given clues just like human can do so.

## 2.0 PROBLEM DEFINITION

Puzzle consists of 25 boxes, which are white and black (see Appendix A). Letters can only be written into white boxes and each number corresponds to a clue. The problem is putting correct letters into empty boxes by answering clues correctly. Although it sounds trivial, it can easily get complicated due to diversity of clues. They can require fact-based answers whereas they may require ambiguous answers, which can be hardly recognized by AI, as well. For example, "Who is the president of US?" is a fact-based answer and its answer can easily be found through a web search. However, if the clue is "Nice one", then to solve it, one needs to understand "Nice" is French city and the word "one" in French means "une", which is the answer. This is a challenging problem for AI because AI does not perceive the semantic. It solves the puzzle dozens or hundreds of times and selects the solution that best fits the constraints [1] which will be mentioned in the subsequent section. However, human can use pattern recognition, decision making and lexical memory access, which makes the problem easier [1].

## 3.0 PROPOSED SOLUTION

To solve the puzzle, reasoning with constraints and depth-first search were implemented as AI algorithms in this project and to explain the AI algorithm's implementation, these two concepts should be explained.

### 3.1 Reasoning with Constraints

Constraint Satisfaction Problem (CSP) for crossword puzzle includes across and down domains consisting of possible letters. Although the objective is to solve 5x5 puzzle grid, for the sake of simplicity, consider the puzzle grid in the figure (see Appendix B). For better understanding, it is

better to go through an example. Let $D_{d1}$ be the first down clue and its scope is $x_1, x_5$ and $x_7$. Then, $D_{d1} = [(x_1, x_5, x_7), \{(a, b, c), (d, e, f)\}]$ means $x_1, x_5, x_7$ can be either $x_1 = a, x_5 = b, x_7 = c$ or $x_1 = d, x_5 = e, x_7 = f$ [2] and $d_1$ is word where $d_1 \in D_{d1}$. Using that approach, consider the following example:

Let the program receives the words from somewhere (e.g., internet) according to clues and creates following domains

$$D_{d1} = [(x_1, x_5, x_7), \{(C, A, T), (D, O, G)\}] \quad (1)$$
$$D_{d2} = [(x_3, x_6, x_9, x_{11}), \{(D, E, A, L), (V, E, A, L)\}] \quad (2)$$
$$D_{a1} = [(x_1, x_2, x_3, x_4), \{(D, R, U, G), (D, O, V, E), (F, O, U, R)\}] \quad (3)$$
$$D_{a2} = [(x_7, x_8, x_9, x_{10}), \{(G, O, A, T), (T, O, A, D)\}] \quad (4)$$

Then, constraints are created according to puzzle's shape.

$$C_1: a_1[x_1] = d_1[x_1] \quad (5)$$
$$C_2: a_1[x_3] = d_2[x_3] \quad (6)$$
$$C_3: a_2[x_7] = d_1[x_7] \quad (7)$$
$$C_4: a_2[x_9] = d_2[x_9] \quad (8)$$

Therefore, there are 10 arcs, which are

$$< a_1, C_1 >, < d_1, C_1 >$$
$$< a_1, C_2 >, < d_2, C_2 >$$
$$< a_2, C_3 >, < d_1, C_3 >$$
$$< a_2, C_4 >, < d_2, C_4 >$$

Then, program starts to check consistency of the arcs.

Is $< a_1, C_1 >$ arc consistent? No. "FOUR" does not satisfy the constraint so it is removed.

$$D_{a1} = [(x_1, x_2, x_3, x_4), \{(D, R, U, G), (D, O, V, E)\}] \quad (9)$$

Is $< a_1, C_2 >$ arc consistent? No. "DRUG" does not satisfy the constraint, so it is removed.

$$D_{a1} = [(x_1, x_2, x_3, x_4), \{ (D, O, V, E)\}] \qquad (10)$$

Is $< a_2, C_3 >$ arc consistent? Yes.

Is $< a_2, C_4 >$ arc consistent? Yes.

Is $< d_1, C_1 >$ arc consistent? No. "CAT" does not satisfy the constraint, so it is removed.

$$D_{d1} = [(x_1, x_5, x_7), \{(D, O, G)\}] \qquad (11)$$

Is $< d_1, C_3 >$ arc consistent? Yes.

Is $< d_2, C_2 >$ arc consistent? No. "DEAL" does not satisfy the constraint, so it is removed.

$$D_{d2} = [(x_3, x_6, x_9, x_{11}), \{(V, E, A, L)\}] \qquad (12)$$

Is $< d_2, C_4 >$ arc consistent? Yes.

$1^{st}$cycle is completed. Then, one needs to check the arc's consistency where there was a domain change.

Is $< a_1, C_1 >$ arc consistent? ($D_{d1}$has changed) Yes.

Is $< a_2, C_3 >$ arc consistent? ($D_{d1}$has changed) No. "TOAD" does not satisfy the constraint, so it is removed.

$$D_{a2} = [(x_7, x_8, x_9, x_{10}), \{(G, O, A, T)\}] \qquad (13)$$

Is $< a_1, C_2 >$ arc consistent? ($D_{d2}$ has changed) Yes.

Is $< a_2, C_4 >$ arc consistent? ($D_{d2}$ has changed) Yes.

Is $< d_1, C_1 >$ arc consistent? ($D_{a1}$ has changed) Yes.

Is $< d_2, C_2 >$ arc consistent? ($D_{a1}$has changed) Yes.

$2^{nd}$ cycle is completed.

Is $< d_1, C_3 >$ arc consistent? ($D_{a2}$ has changed) Yes.

Is $< d_2, C_4 >$ arc consistent? ($D_{a2}$has changed) Yes.

Since there is no change, algorithm is done. Final domains are

$$D_{d1} = [(x_1, x_5, x_7), \{(D, O, G)\}] \quad (14)$$

$$D_{d2} = [(x_3, x_6, x_9, x_{11}), \{(V, E, A, L)\}] \quad (15)$$

$$D_{a1} = [(x_1, x_2, x_3, x_4), \{(D, O, V, E)\}] \quad (16)$$

$$D_{a2} = [(x_7, x_8, x_9, x_{10}), \{(G, O, A, T)\}] \quad (17)$$

Then, letters in these domains are put into corresponding positions (see Appendix C). Simply, algorithm checks consistency of all arcs and if there is no change in domains after a cycle, it stops.

### 3.2 Depth-First Search

Depth-First Search is a blind search technique. It is called blind because it works with no information about the search space except the knowledge of the goal state. To conduct a depth-first search, one needs to do the following (see Appendix D):

- One element queue consisting only root node is created.
- Until the first path in the queue ends at the goal node or there is no path in queue do the following:
    - Remove first path and create new paths by extending it to all the neighbors of the terminal node.
    - Reject all new paths with loops (e.g., 3-5-2-5)
    - If any, add new paths to the end of the queue.
- If the goal node is found, announce success, else announce failure.

This technique is preferable if partial paths which does not end with goal node are never too long.

## 4.0 IMPLEMENTATION

Explanation for the implementation of the project can be divided into two: data structures and solving the puzzle.

### 4.1 Data Structures

To make the code efficient, three classes were constructed: word class, constraint class, tile class and crossword class.

#### 4.1.1 Word Class

Word class represents words in the crossword, and it has the following attributes (see Appendix M)

- word_length: Number of letters in a word.
- crossword_x, crossword_y: Location of the first letter.
- number: Clue number of the word.
- clue: Clue as a string.
- domain: List of possible answers with the same length.
- domainChanged: Keeps domain's status (whether it was changed in constraint check).
- direction: Direction for constraint check.
  ($e.g., a_1[x_1]$ is compared with $d_1[x_1]$ or vice versa )

#### 4.1.2 Constraint Class

Constraint Class represents arcs explained in section 3.1.

- word1, word2: Two words that will be compared as strings.
- word1index, word2index: Index of the letter what will be compared.

#### 4.1.3 Tile Class

Tile Class represents one square box in the puzzle grid.

- letter: Letter in a tile as a string if tile is black then it is ".".

- number: Number of the square.

### 4.1.4 Crossword Class

Crossword Class represents the entire 5x5 crossword (see Appendix M). To reduce the complexity of the algorithm, combination of python list and dictionary structures were used. Since puzzle's structure is 5x5, two-dimensional list of tile objects was constructed and for across and down clues, two dictionaries were used. Dictionaries' keys are clue numbers as integers, and corresponding values are clues as strings. Additionally, two dictionaries, whose keys are numbers and values are word objects, were constructed to easily access the information of word. This class has the following attributes

- tile_grid: Two-dimensional list of tiles.
- across_clues/down_clues: Two dictionary for across and down clues.
- across_word_objects/down_word_objects: Two dictionary for across and down words.
- constraints: list of constraint objects.
- step: list of single stepping.

These classes and their combinations increase the flexibility of the program and make the program more maintainable.

## 4.2 Solving The Puzzle

Solving the puzzle consists of two steps: elimination through constraint check and finding the correct puzzle.

### 4.2.1 Domain Initialization and Reduction With Constraints

First, clues, shape of the puzzle grid and relevant information are taken from New York Times website (see Appendix E). Then, using the shape of the puzzle, constraints are created (see Appendix F). Answers of the clues are searched through Wikipedia and all of the words found in first 500 search results are taken. While doing this, words are being eliminated by using the length information of the word object. Because of puzzle's shape, each clue can only take specific length of words so large number of words are eliminated and domains for each across

and down clue are created (see Appendix G). After that constraints were checked by applying the algorithm mentioned in section 3.1. Mostly, 23 constraints are created from puzzle grid so large number of words are eliminated (see Appendix H).

### 4.2.2 Finding The Puzzle

After the constraint elimination is done; domains have much lower words (see Appendix I) than before. The problem is how to decide which one is correct. To achieve this, depth-first search is implemented. Goal state is the solution which includes words from every non-empty domain and these words should satisfy the relevant constraints. Note that goal state may not always exists. Sometimes number of words found by constraints may be lower than number of non-empty domains. In that case, every solution to the crossword is searched exhaustively and the solution with maximum number of words is taken as a correct solution. Depth-first search algorithm works as explained in section 3.2.

## 5.0 RESULTS AND ANALYSIS

To test AI's performance, 20 unique mini crossword examples were extracted from New York Times website and saved into a file. Then, these examples were taken from that file and tested. To see the AI algorithm's performance properly, two cases were analyzed. In the first case, algorithm uses the words, which may include or may not include correct answers, taken from Wikipedia. In the second case, it was assumed that domains include correct words after the extraction from Wikipedia, so answers were added to the domains before AI algorithm starts doing its job. Results of both cases can be seen in Appendix J and K. Additionally, to see the results clearly, one can check the table (see Appendix L). It seems that AI performs poorly in the absence of data but performs very well in the presence of data. The reason for its poor performance for the first case was expected because the only knowledge for the correctness of the words is words' constraints except their lengths. Since if any domain does not include the answer, that may result in eliminating the correct answers because their constraints may not be satisfied by wrong answers, which is the case in general.

Does the poor performance in the first case make the AI bad or useless? No, it does not. By definition, AI is the study of making computers do intelligent things which people can do, such as think and make decisions. These intelligent things can be done by an important ability of humankind which connects humans to the present. It is recalling [3]. Thanks to recalling, retrieval of words from the semantic structures are achieved; therefore, humans are able to answer questions properly [3]. Hence, it would not be illogical to expect such a poor result in the absence of knowledge because just like AI, humans would not be able to answer as well [3].

## 6.0 CONCLUSION AND RECOMMENDATIONS

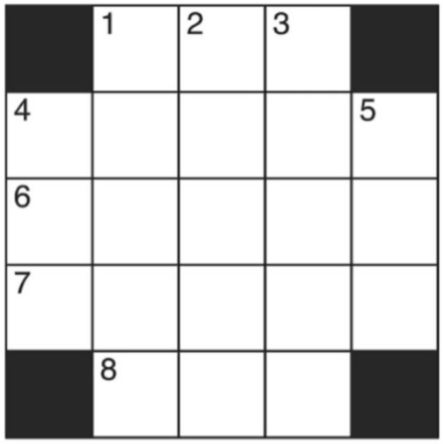
This project's purpose was to make AI to solve crossword puzzle and it was achieved. In the absence of correct answers, algorithm works poorly but, in the presence, it is almost perfect. To increase the performance, retrieval of words should be done more efficiently. If the number of words extracted from web is increased too much, after the constraint check, there may be more than one answers remained in domains and that introduces the chance factor to find the solution. Despite the large number, resources do not include informal words such as slangs, so it is highly unlikely to find all correct words. Another option is checking for synonyms of the words to find correct answer, but this is very time consuming. Lastly, more resources can be utilized to look for words. Only four resources (Wikipedia, wordnet, Merriam-Webster dictionary, Theasarus and Encylopedia) were allowed to be used and it was found that Wikipedia is the most general source, namely that the words found in other resources can be found in Wikipedia, but some words found in Wikipedia cannot be found in the other resources, so only Wikipedia was used as a resource. Doing that, the largest possible resource was used in searching for words and program becomes time efficient.

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

**Word Count: 2000**

## APPENDICES

## Appendix A – New York Times Mini Crossword Puzzle Example




**ACROSS**

1 "Downton Abbey" airer
4 U.C.L.A. athlete
6 Ruffles potato chip feature
7 "You ___ to know better"
8 Word with the longest entry in the O.E.D.

**DOWN**

1 Car that's popular among progressives
2 Barely move
3 One of the senses
4 Many a fraternity member, in modern slang
5 Brooklyn baller

Figure 1: Sample New York Times Mini Crossword Puzzle

## Appendix B – A Crossword Puzzle

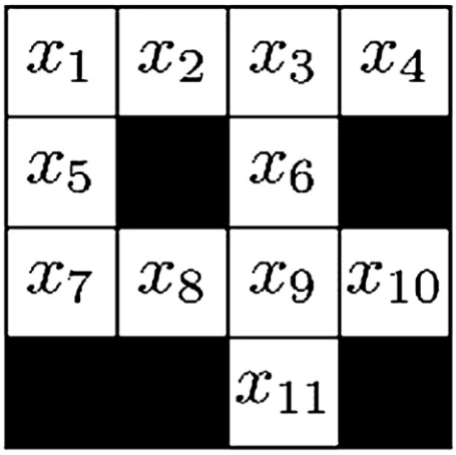



Figure 2: A Crossword Puzzle [2]

## Appendix C – Solution of The Crossword

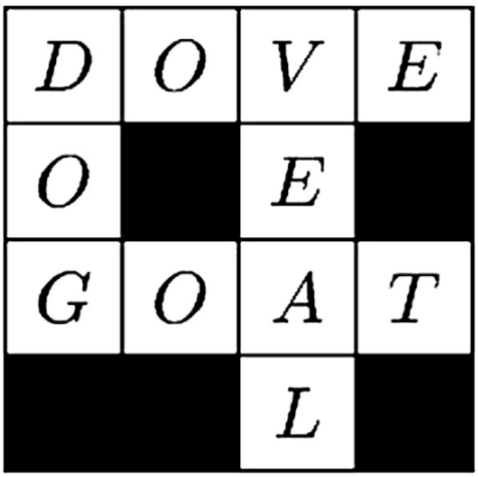



*Figure 3 : Solution of the Crossword [2]*

## Appendix D – Depth-First Search

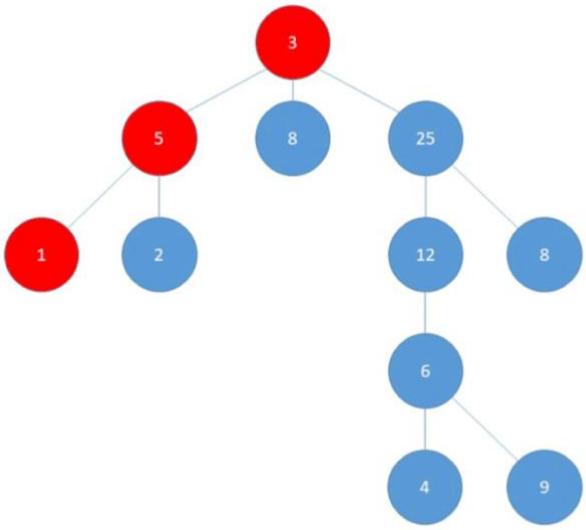



*Figure 4: Example of Depth-First Search*

## Appendix E – Extraction of Information from Puzzle

```
Clues have been received successfully.
{1: 'A-mile-a-minute speed', 6: 'In the loop', 7: 'Computer screen dot', 8: 'Surprising thing that frigatebirds can do while flying', 9: 'Acid, by another name'}
{1: 'Drains of energy', 2: '"___ Survive" (disco hit)', 3: 'Horizontal lines on graphs', 4: 'Covered in forest', 5: 'Site with crowdsourced reviews'}
ACROSS WORDS:
{'word_length': 5, 'crossword_y': 0, 'crossword_x': 0, 'number': 1, 'clue': 'A-mile-a-minute speed', 'domain': [], 'domainChanged': True, 'direction': 'across'}
{'word_length': 5, 'crossword_y': 1, 'crossword_x': 0, 'number': 6, 'clue': 'In the loop', 'domain': [], 'domainChanged': True, 'direction': 'across'}
{'word_length': 5, 'crossword_y': 2, 'crossword_x': 0, 'number': 7, 'clue': 'Computer screen dot', 'domain': [], 'domainChanged': True, 'direction': 'across'}
{'word_length': 5, 'crossword_y': 3, 'crossword_x': 0, 'number': 8, 'clue': 'Surprising thing that frigatebirds can do while flying', 'domain': [], 'domainChanged': True, 'direction': 'across'}
{'word_length': 3, 'crossword_y': 4, 'crossword_x': 1, 'number': 9, 'clue': 'Acid, by another name', 'domain': [], 'domainChanged': True, 'direction': 'across'}
DOWN WORDS:
{'word_length': 4, 'crossword_y': 0, 'crossword_x': 0, 'number': 1, 'clue': 'Drains of energy', 'domain': [], 'domainChanged': True, 'direction': 'down'}
{'word_length': 5, 'crossword_y': 0, 'crossword_x': 1, 'number': 2, 'clue': '"___ Survive" (disco hit)', 'domain': [], 'domainChanged': True, 'direction': 'down'}
{'word_length': 5, 'crossword_y': 0, 'crossword_x': 2, 'number': 3, 'clue': 'Horizontal lines on graphs', 'domain': [], 'domainChanged': True, 'direction': 'down'}
{'word_length': 5, 'crossword_y': 0, 'crossword_x': 3, 'number': 4, 'clue': 'Covered in forest', 'domain': [], 'domainChanged': True, 'direction': 'down'}
{'word_length': 4, 'crossword_y': 0, 'crossword_x': 4, 'number': 5, 'clue': 'Site with crowdsourced reviews', 'domain': [], 'domainChanged': True, 'direction': 'down'}
```

## Appendix F – Constraint Generation

```
CONSTRAINTS:
a1[0] = d1[0]
a1[1] = d2[0]
a1[2] = d3[0]
a1[3] = d4[0]
a1[4] = d5[0]
a6[0] = d1[1]
a6[1] = d2[1]
a6[2] = d3[1]
a6[3] = d4[1]
a6[4] = d5[1]
a7[0] = d1[2]
a7[1] = d2[2]
a7[2] = d3[2]
a7[3] = d4[2]
a7[4] = d5[2]
a8[0] = d1[3]
a8[1] = d2[3]
a8[2] = d3[3]
a8[3] = d4[3]
a8[4] = d5[3]
a9[0] = d2[4]
a9[1] = d3[4]
a9[2] = d4[4]
```

Figure 6: Constraint Generation

## Appendix G – Domain Generation

```
Crawler Made...
Domain received for a1
['speed', 'miles', 'first', 'known', 'speak', 'rapid', 'extra', 'today', 'metre', 'usage', 'equal', 'times', 'takes', 'place', 'areas', 'texas', 'limit', 'andor', 'types', 'track', 'short', 'gar
Domain received for a6
['loops', 'black', 'thick', 'chris', 'which', 'union', 'forms', 'edwin', 'built', 'coney', 'other', 'kinds', 'music', 'sound', 'short', 'refer', 'areas', 'radio', 'north', 'train', 'shunt', 'kn
Domain received for a7
['image', 'space', 'pitch', 'video', 'color', 'later', 'tandy', 'using', 'sizes', 'there', 'three', 'basic', 'kinds', 'fonts', 'front', 'asked', 'stare', 'cross', 'lines', 'angle', 'known', 'wh
Domain received for a8
['thing', 'while', 'comes', 'title', 'blows', 'sparm', 'sides', 'their', 'tying', 'other', 'cinet', 'could', 'ruddy', 'above', 'valid', 'files', 'about', 'gulls', 'seize', 'there', 'doubt', 'ch
Domain received for a9
['ion', 'was', 'the', 'but', 'asp', 'all', 'and', 'has', 'jns', 'pro', 'oil', 'for', 'its', 'new', 'jim', 'who', 'not', 'nok', 'fat', 'are', 'neo', 'rap', 'act', 'can']
Domain received for d1
['been', 'with', 'flow', 'zero', 'also', 'head', 'otec', 'uses', 'deep', 'heat', 'publ', 'aera', 'used', 'both', 'fuel', 'dibl', 'this', 'seek', 'from', 'then', 'form', 'good', 'down', 'when',
Domain received for d2
['disco', 'known', 'never', 'first', 'award', 'stone', 'songs', 'daily', 'their', 'chris', 'metro', 'verse', 'hints', 'dance', 'panic', 'hopes', 'marie', 'album', 'train', 'which', 'bside', 'he
Domain received for d3
['graph', 'lines', 'point', 'given', 'cycle', 'class', 'april', 'march', 'maint', 'theil', 'nodes', 'other', 'units', 'chart', 'bound', 'feint', 'paper', 'which', 'pixel', 'final', 'types', 'sc
Domain received for d4
['cover', 'miles', 'state', 'globe', 'moist', 'cloud', 'level', 'these', 'south', 'large', 'trees', 'refer', 'parts', 'areas', 'which', 'about', 'yukon', 'india', 'along', 'coast', 'norte', 'hu
Domain received for d5
['with', 'site', 'play', 'from', 'many', 'lead', 'food', 'lets', 'tmdb', 'they', 'scam', 'spam', 'also', 'what', 'user', 'have', 'life', 'film', 'into', 'that', 'data', 'list', 'more', 'like']
```

Figure 7: Domain Generation

## Appendix H – Sample Constraint Elimination

```
entered consistency check
miles is removed from across 1
known is removed from across 1
rapid is removed from across 1
metre is removed from across 1
equal is removed from across 1
exact is removed from across 1
reach is removed from across 1
means is removed from across 1
first is removed from across 1
extra is removed from across 1
times is removed from across 1
limit is removed from across 1
```

Figure 8: Sample Constraint Elimination

## Appendix I – Domains After Constraint Elimination

```
a1 domain: ['sixty']
Number of Words:  0
a6 domain: ['aware']
Number of Words:  0
a7 domain: ['pixel']
Number of Words:  0
a8 domain: ['sleep']
Number of Words:  0
a9 domain: ['are', 'lsd']
Number of Words:  1
d1 domain: ['heat', 'this', 'saps']
Number of Words:  2
d2 domain: ['iwill']
Number of Words:  0
d3 domain: ['']
Number of Words:  0
d4 domain: ['norte', 'treed']
Number of Words:  1
d5 domain: ['list', 'yelp']
Number of Words:  1
-================ STEPS ==================-
Displaying application UI.
UI window closed. Program will now exit.
```

Figure 9: Domains After Constraint Elimination

## Appendix J – Snapshots of Puzzle Results Without Adding Correct Answers into the Domains

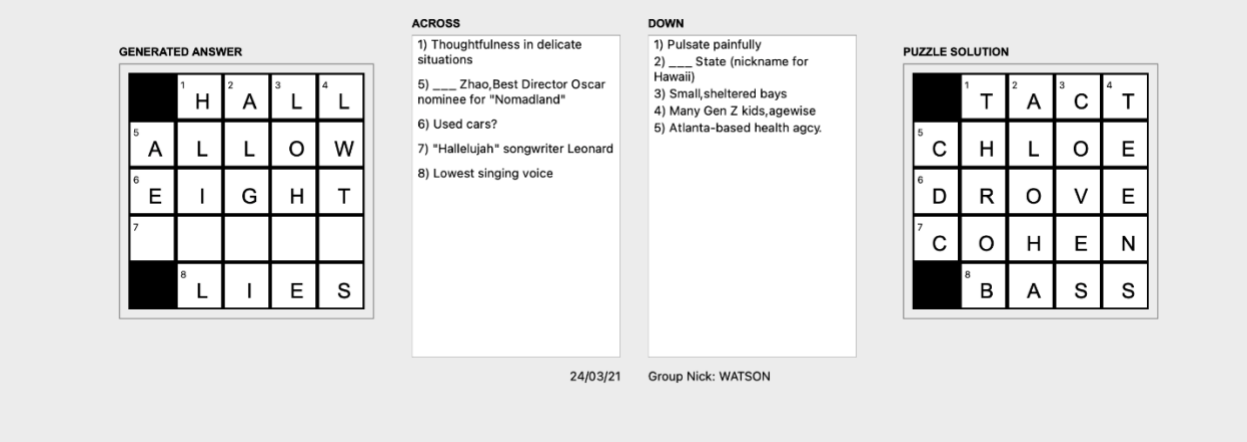



Figure 10: Result for 24/03/2021

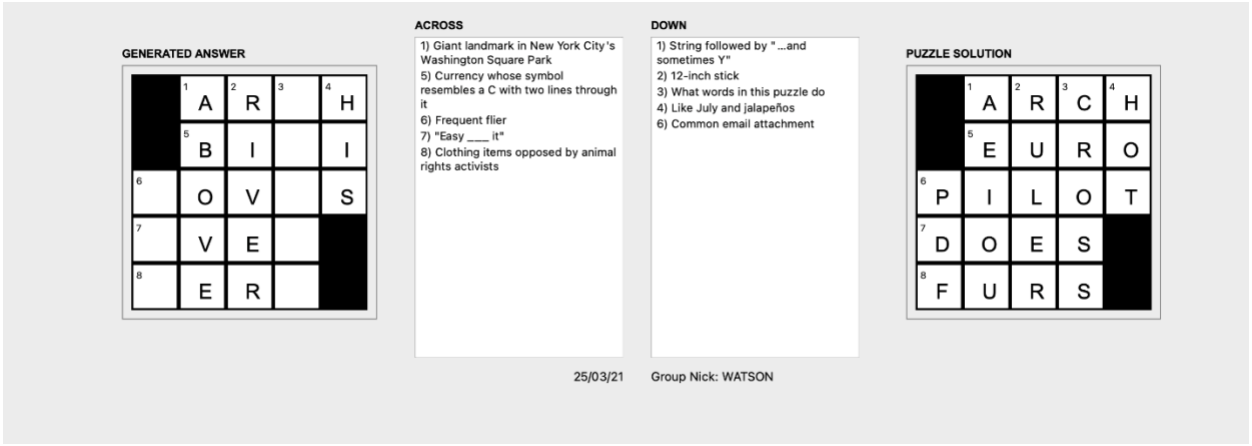



Figure 11: Result for 25/03/2021

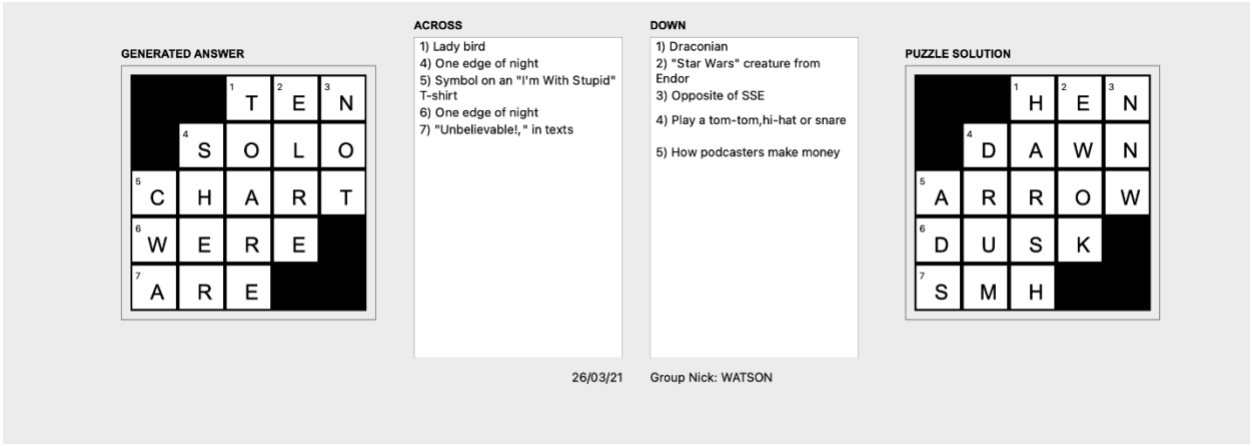



Figure 12: Result for 26/03/2021

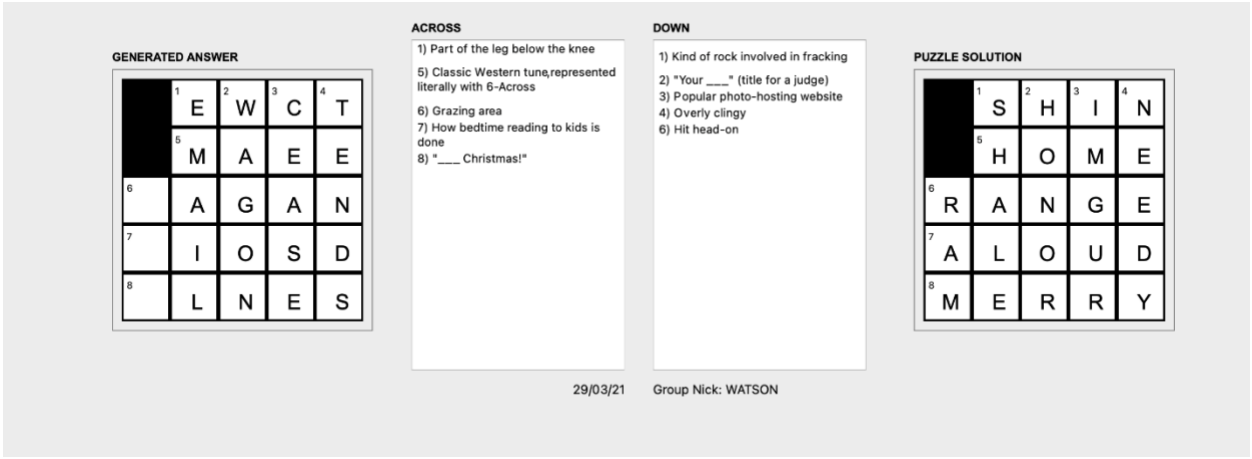



Figure 13: Result for 29/03/2021

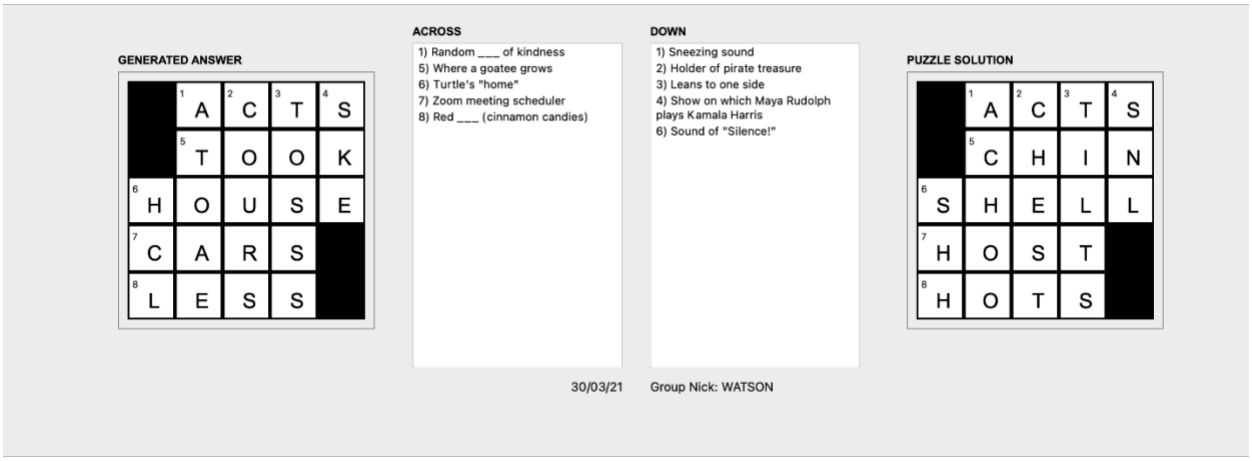



Figure 14: Result for 30/03/2021

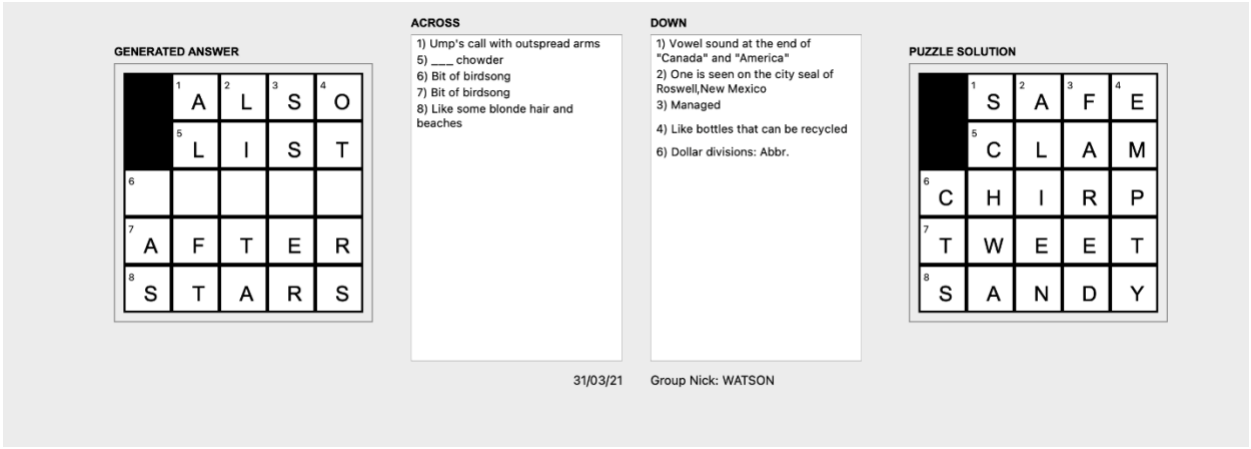



Figure 15: Result for 31/03/2021

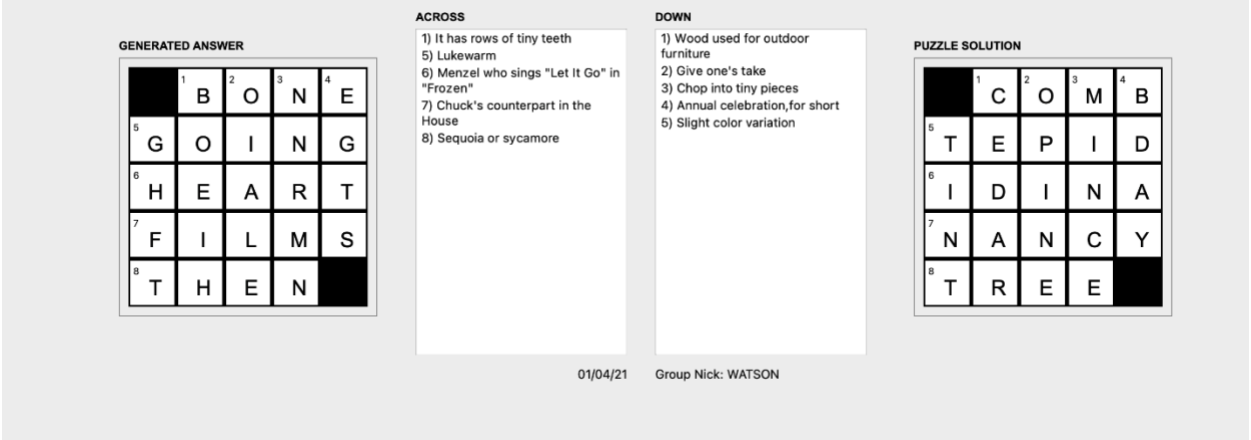



Figure 16: Result for 01/04/2021

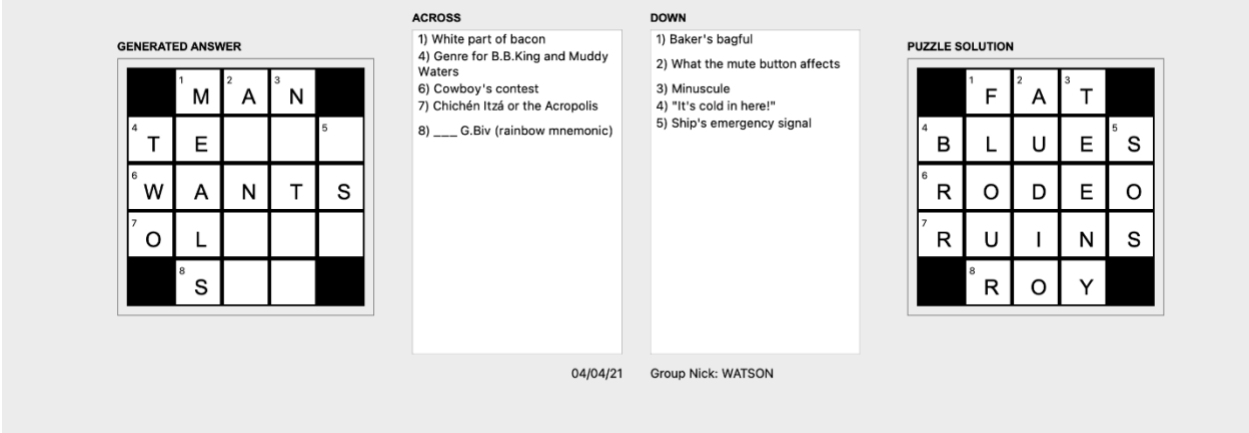



Figure 17: Result for 04/04/2021

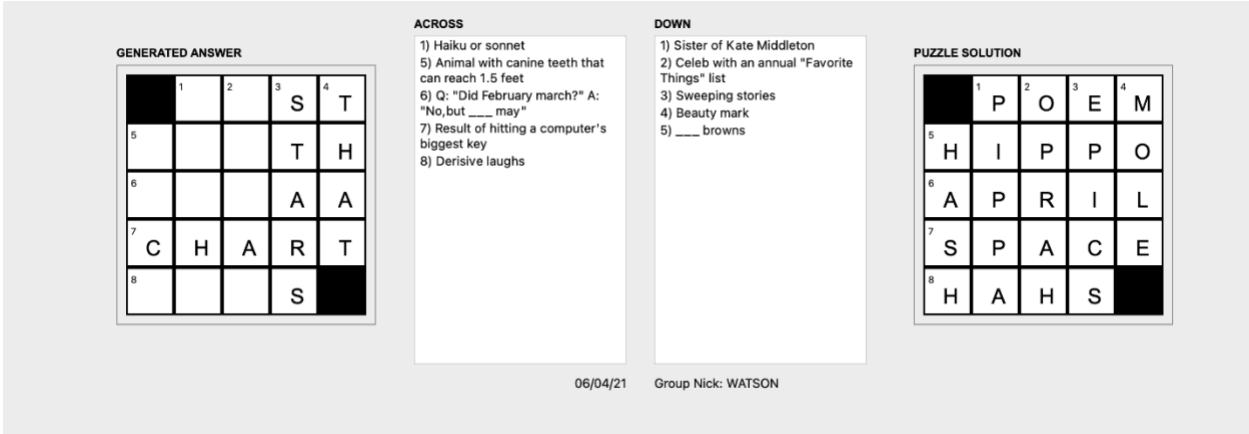



Figure 18: Result for 06/04/2021

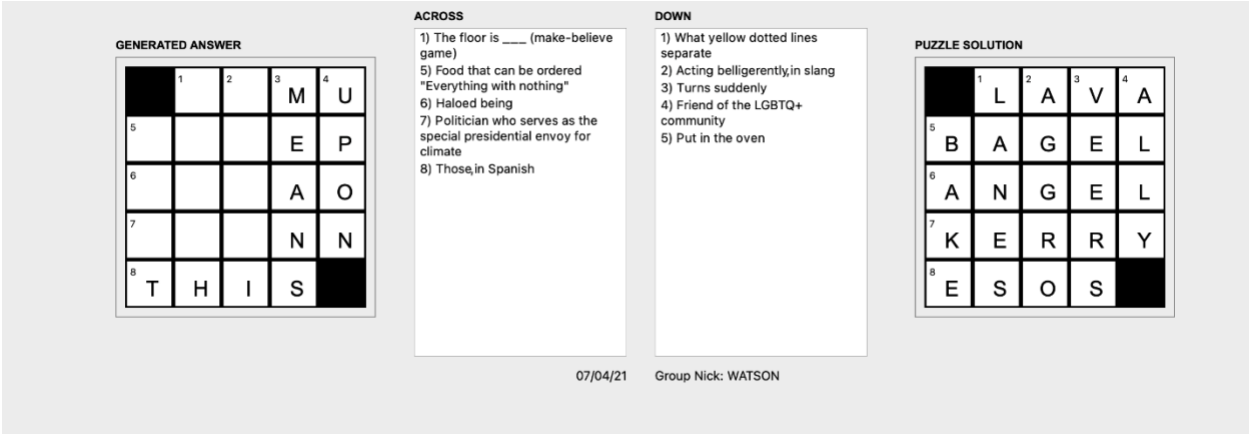



Figure 19: Result for 07/04/2021

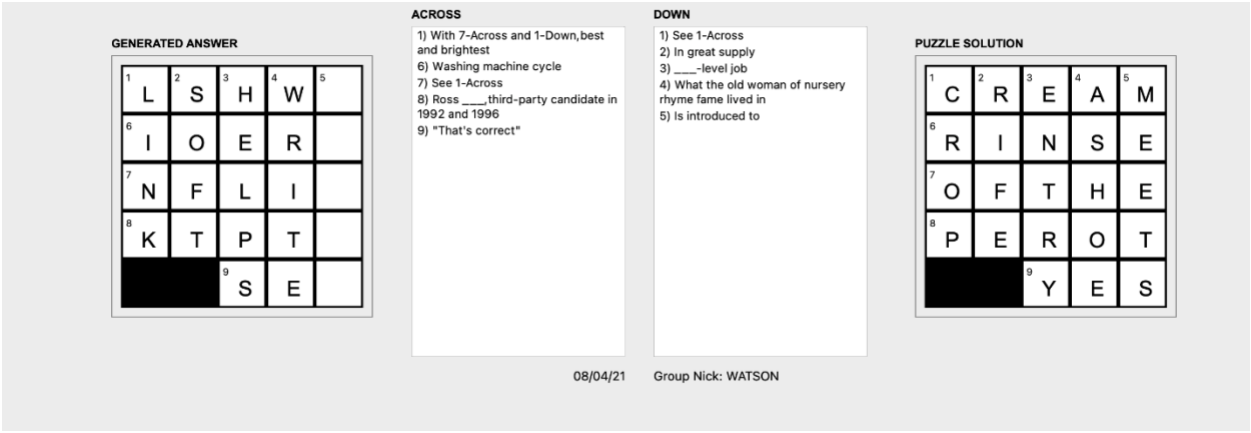



Figure 20: Result for 08/04/2021

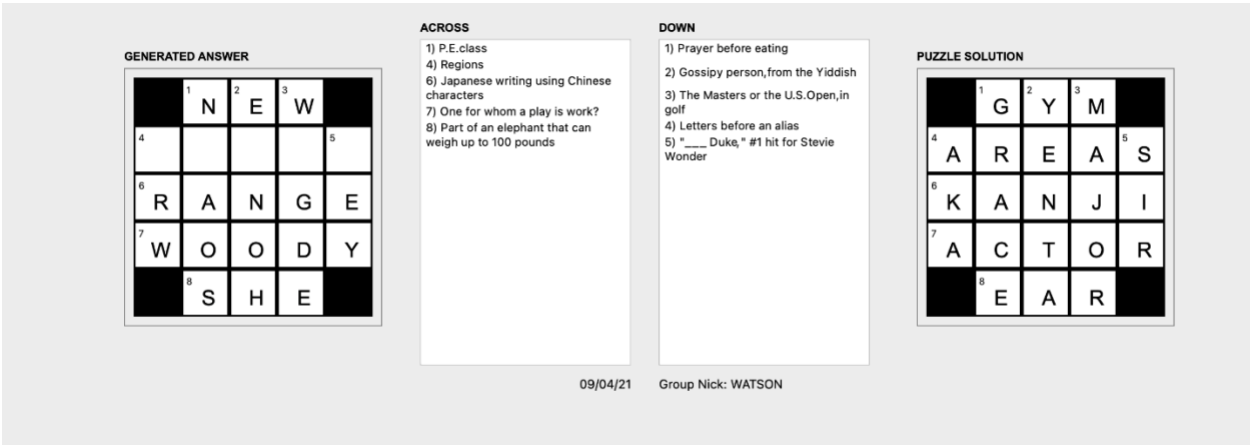



Figure 21: Result for 09/04/2021

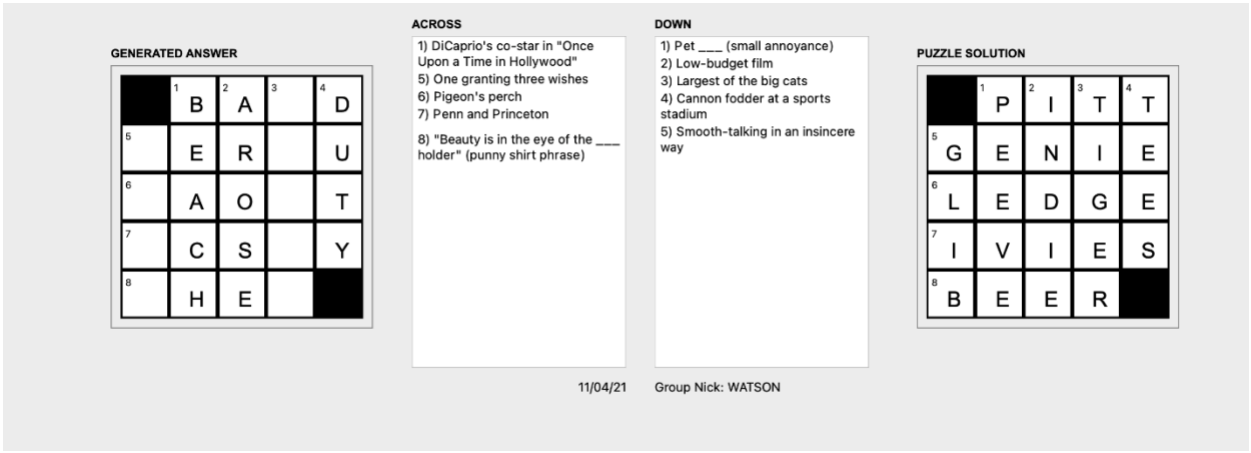



Figure 22: Result for 11/04/2021

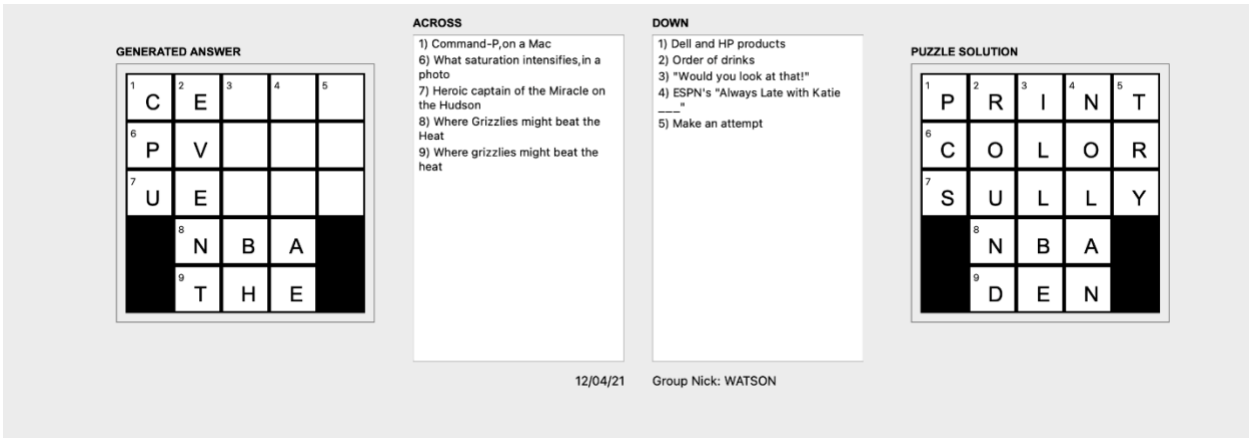



Figure 23: Result for 12/04/2021

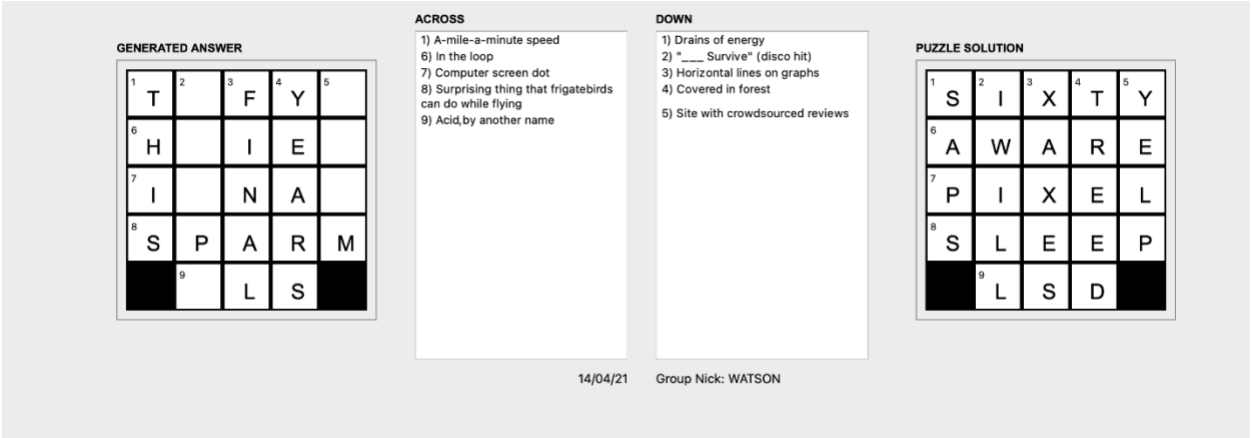



Figure 24: Result for 14/04/2021

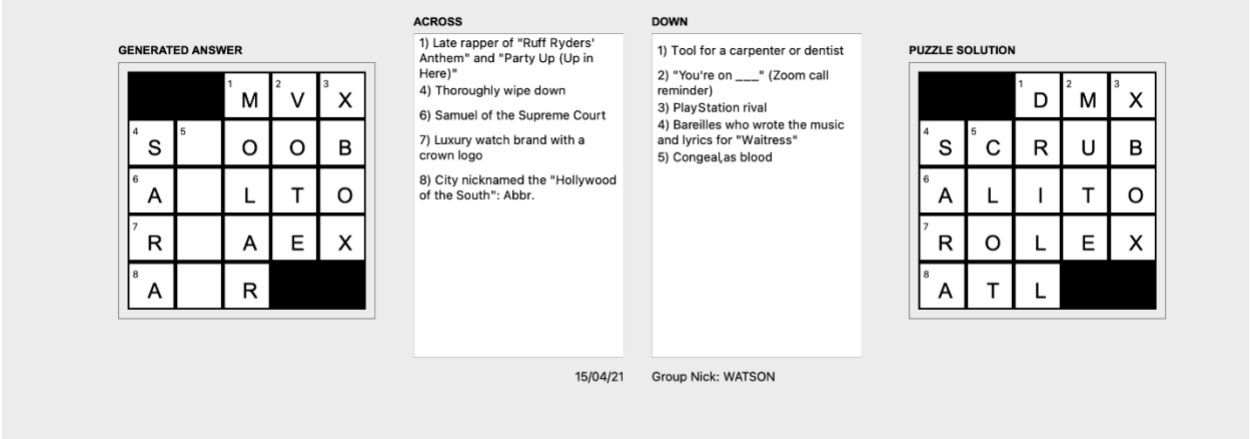



Figure 25: Result for 15/04/2021

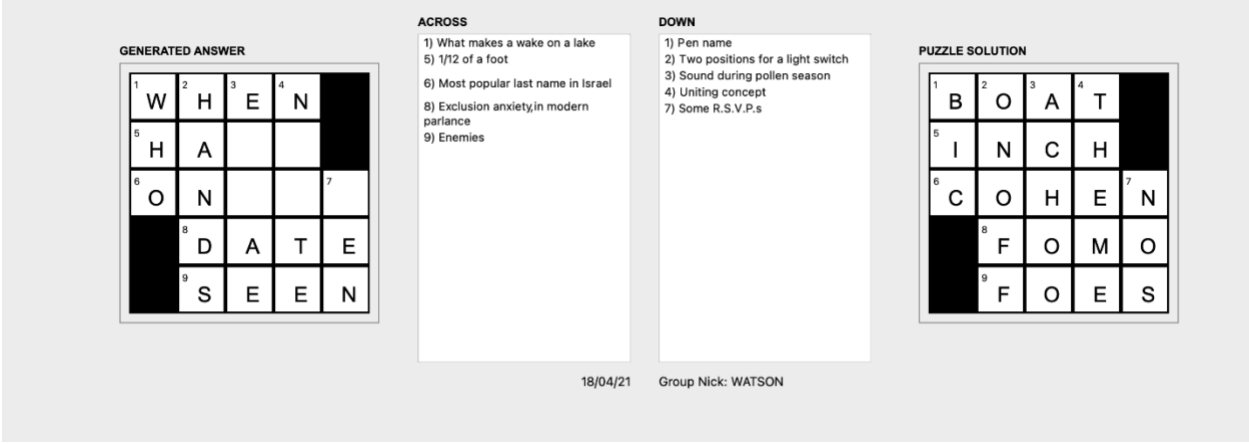



Figure 26: Result for 18/04/2021

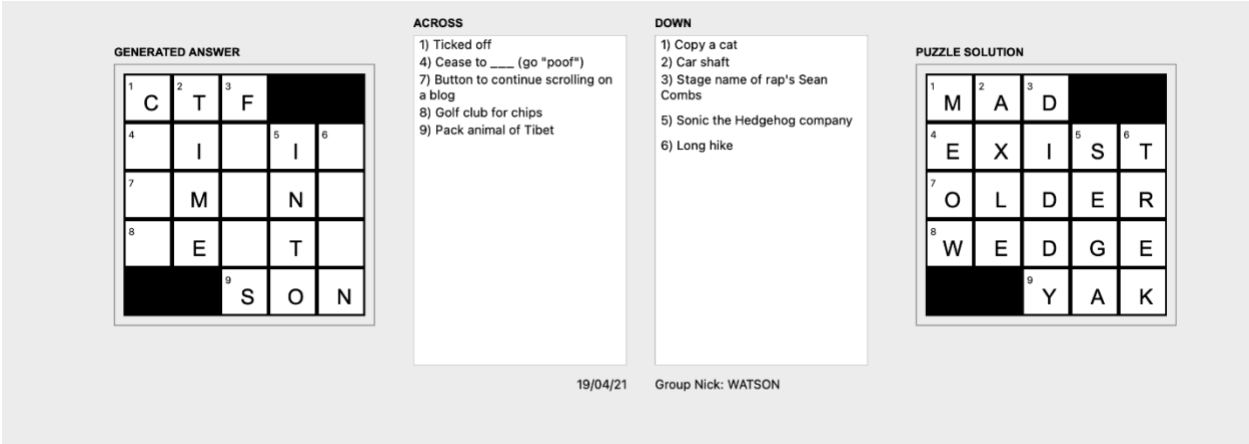



Figure 27: Result for 19/04/2021

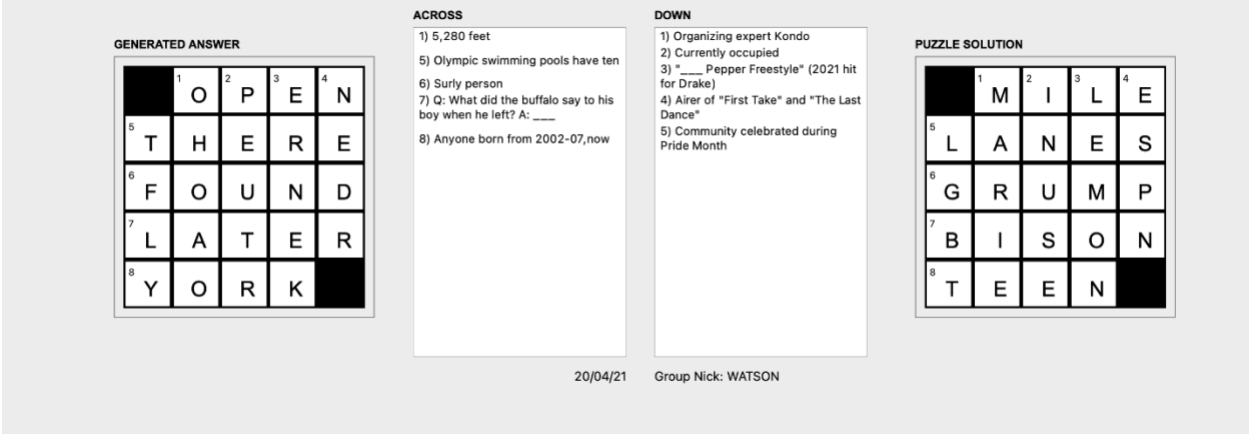



*Figure 28: Result for 20/04/2021*

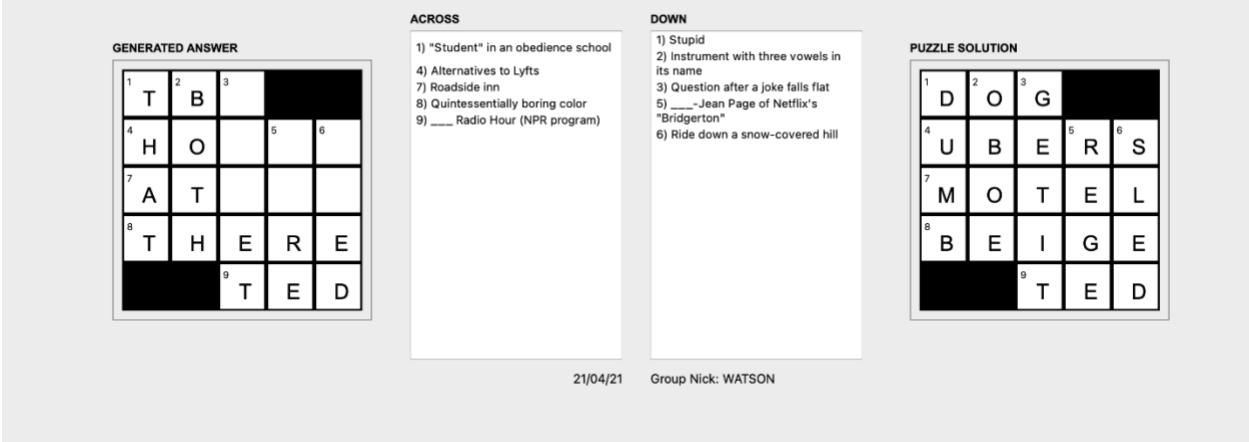



Figure 29: Result for 21/04/2021

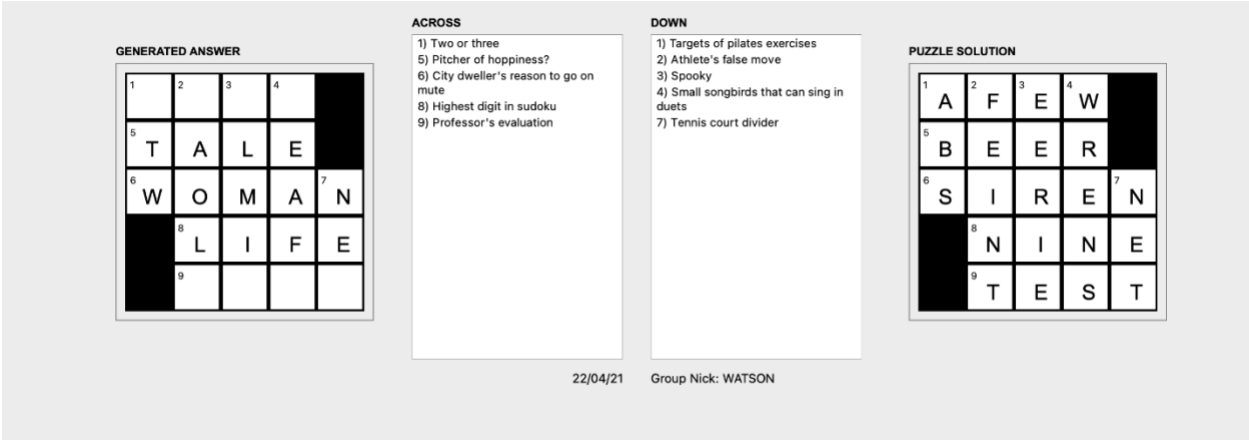



Figure 30: Result for 22/04/2021

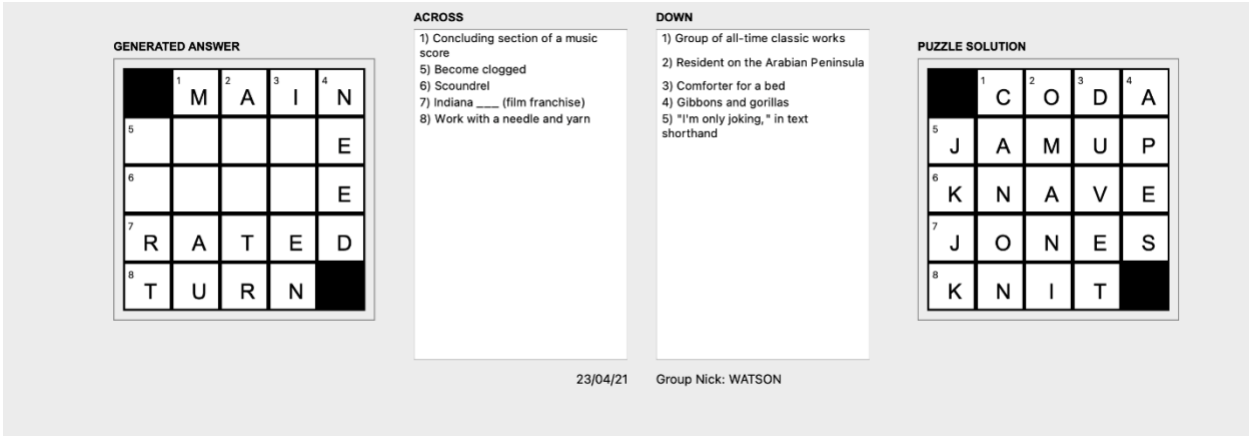



Figure 31: Result for 23/04/2021

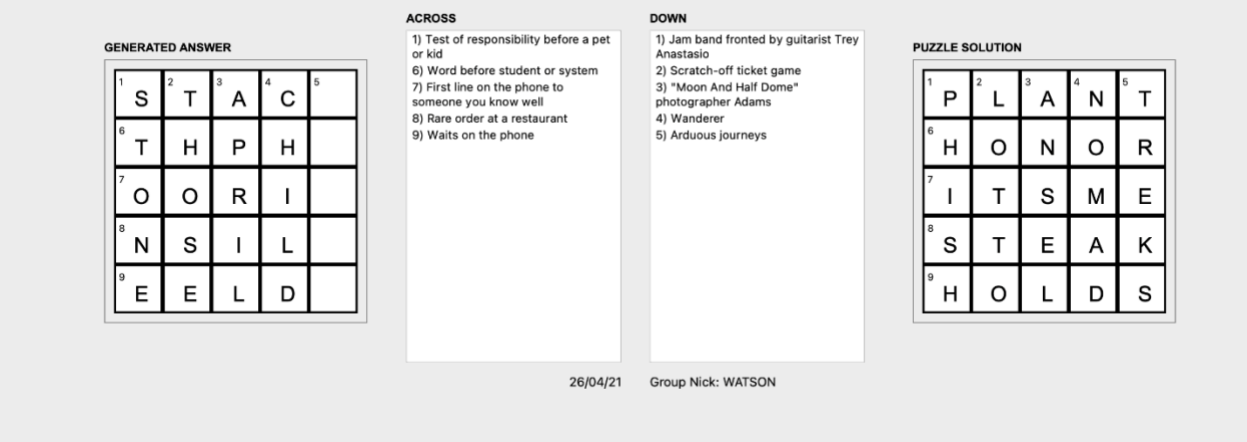



Figure 32: Result for 26/04/2021

## Appendix K – Snapshots of Puzzle Results with Adding Correct Answers into the Domains

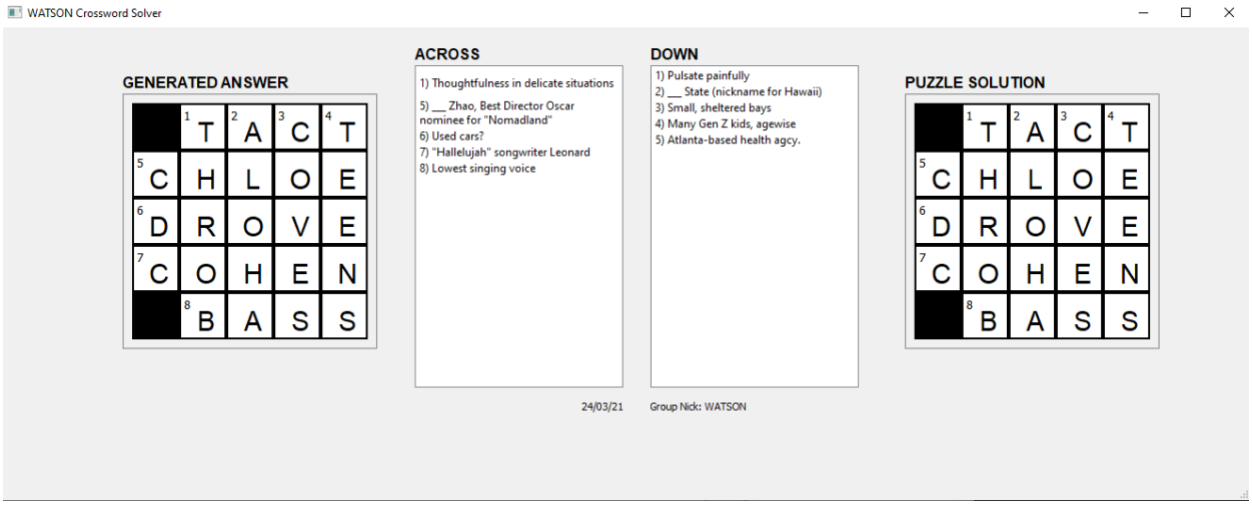



Figure 33: Result for 24/03/2021

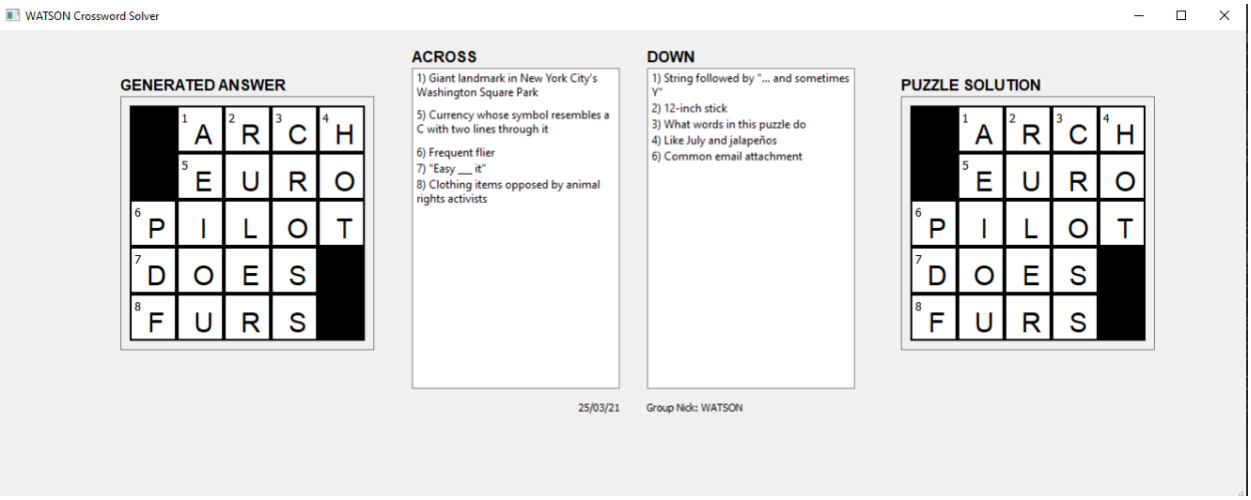



Figure 34: Result for 25/03/2021

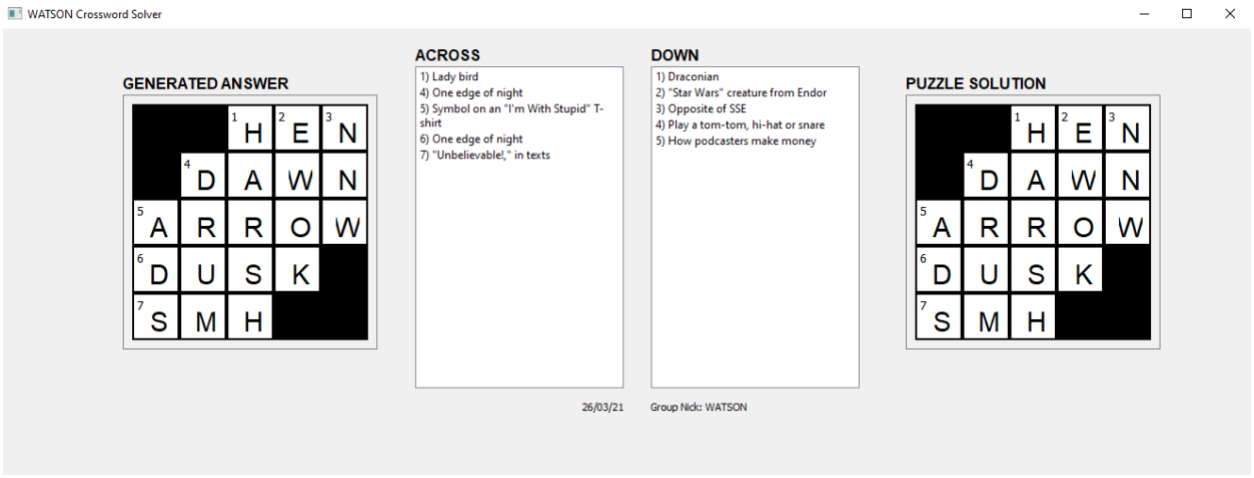



Figure 35: Result for 26/03/2021

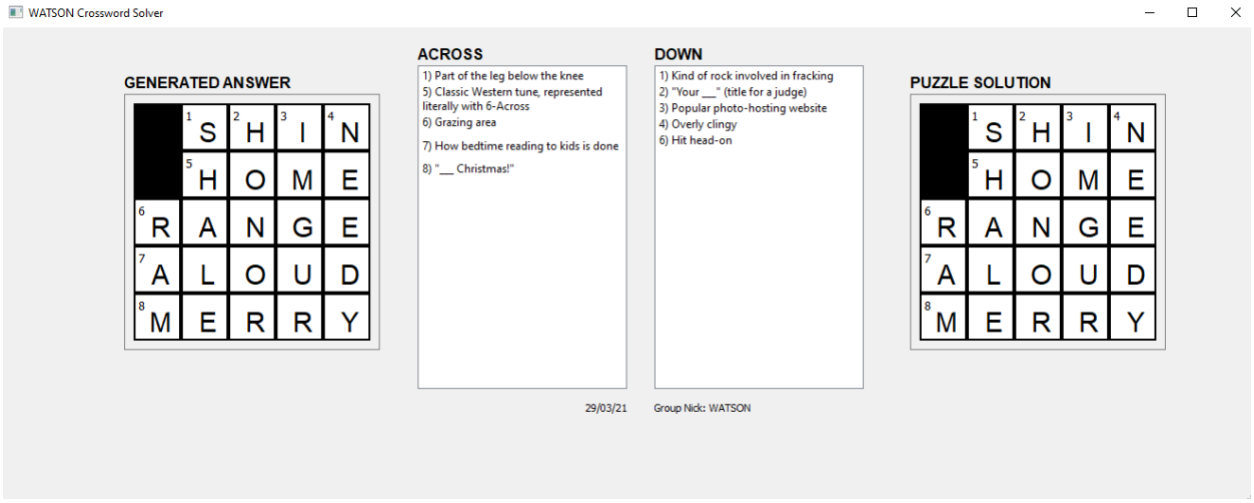



Figure 36: Result for 29/03/2021

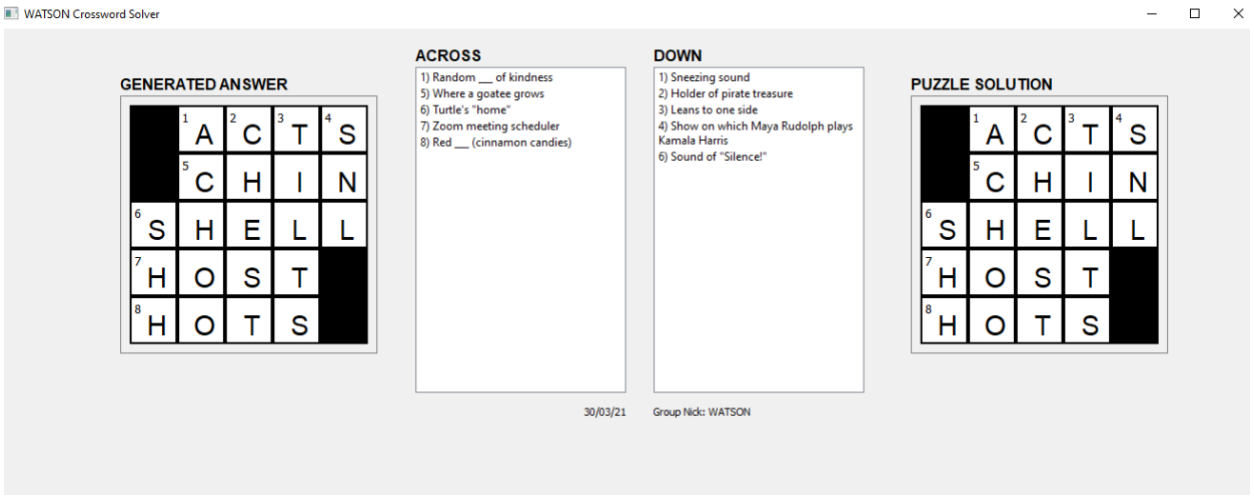



Figure 37: Result for 30/03/2021

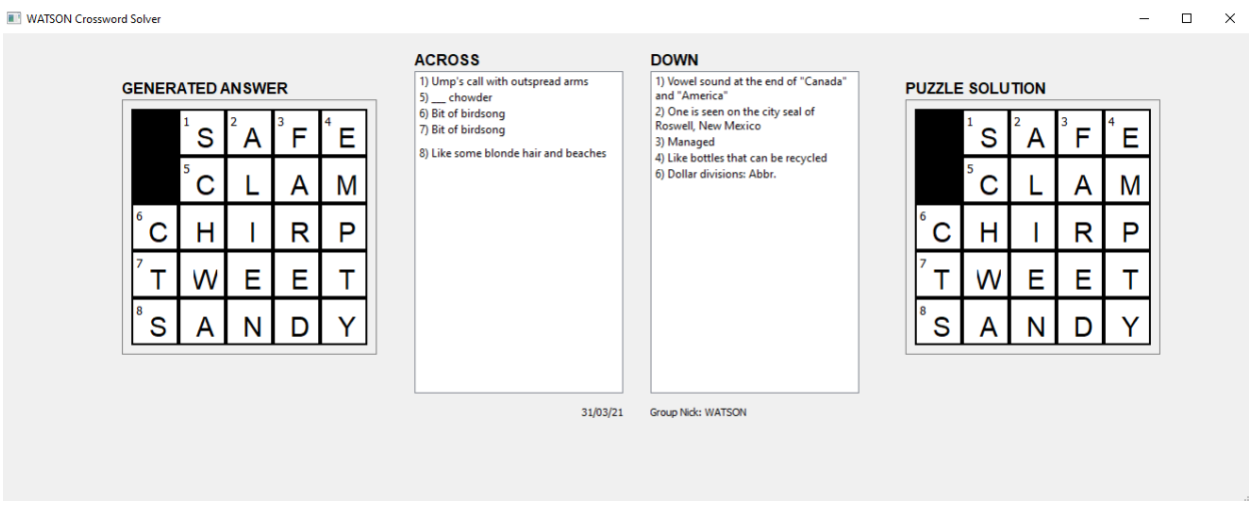



Figure 38: Result for 31/03/2021

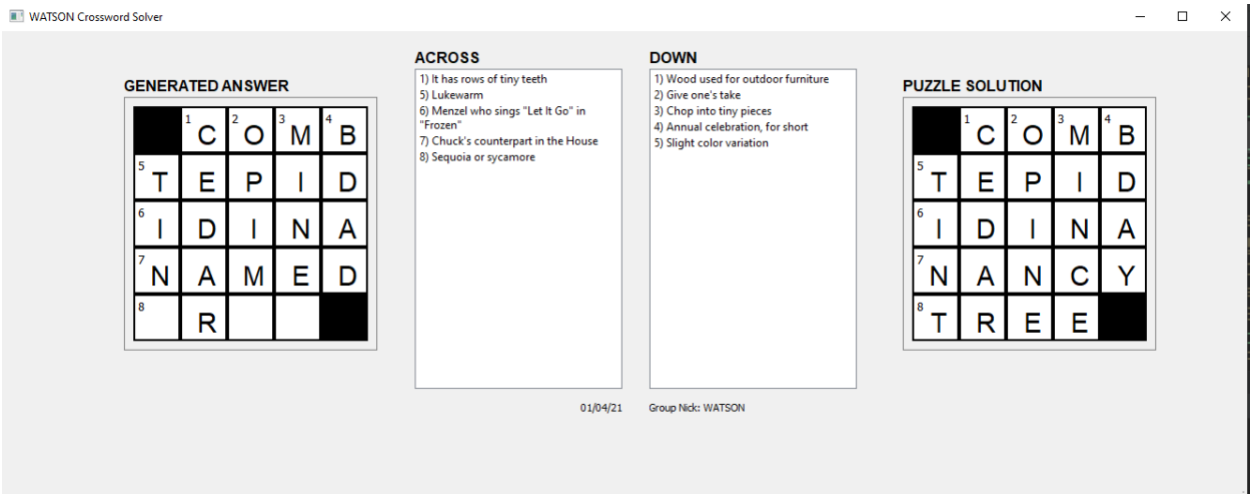



Figure 39: Result for 01/04/2021

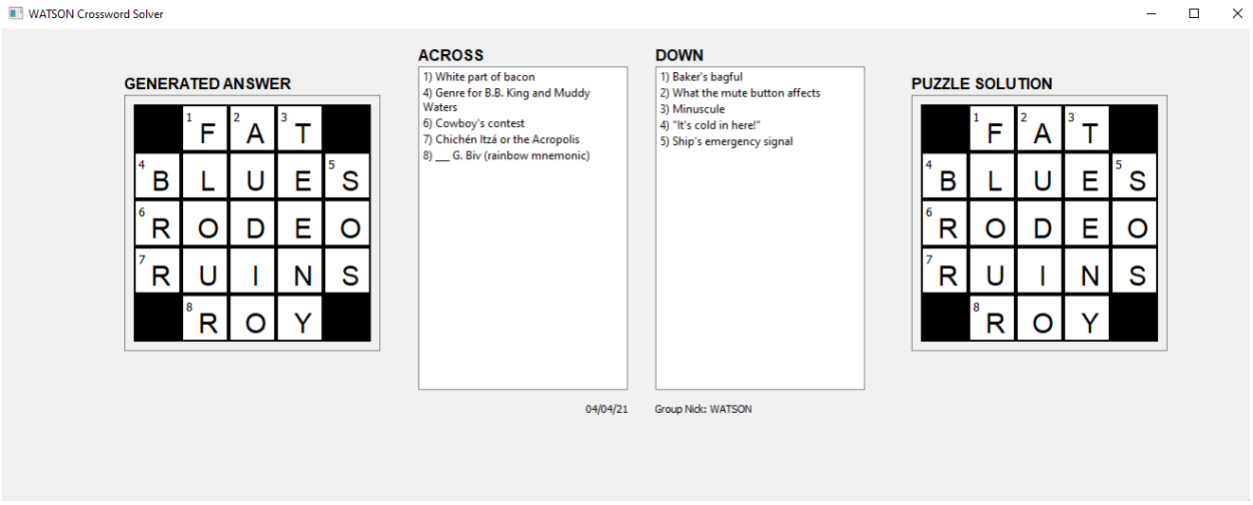



Figure 40: Result for 04/04/2021

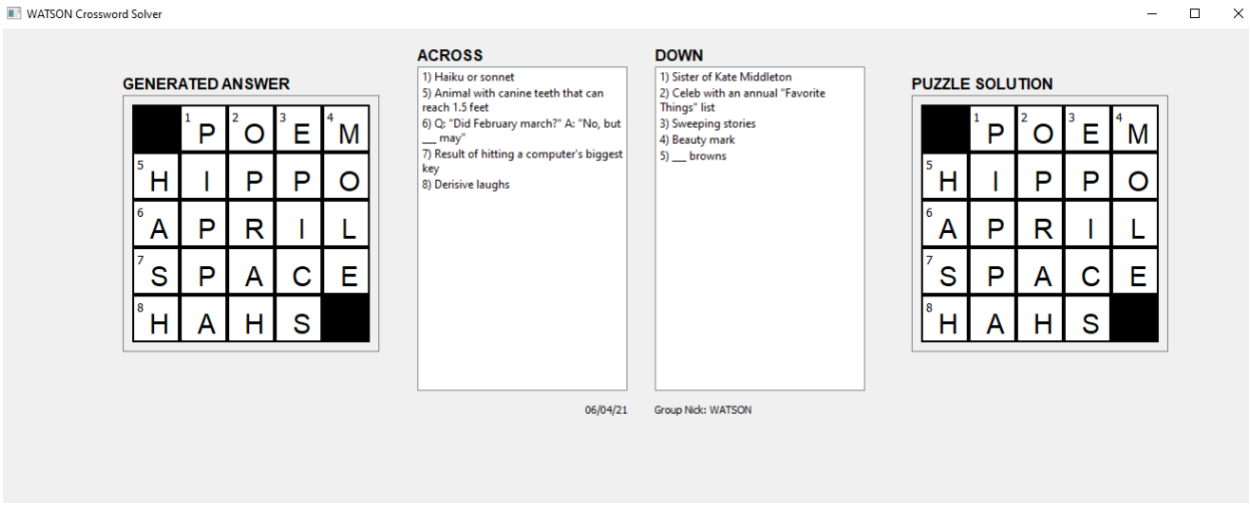



Figure 41: Result for 06/04/2021

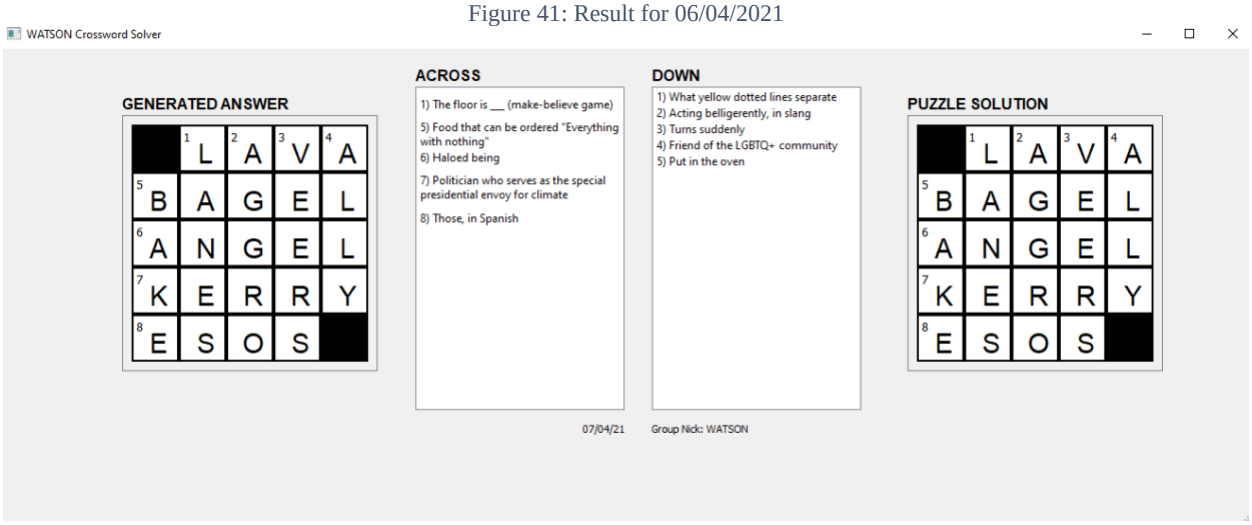



Figure 42: Result for 07/04/2021

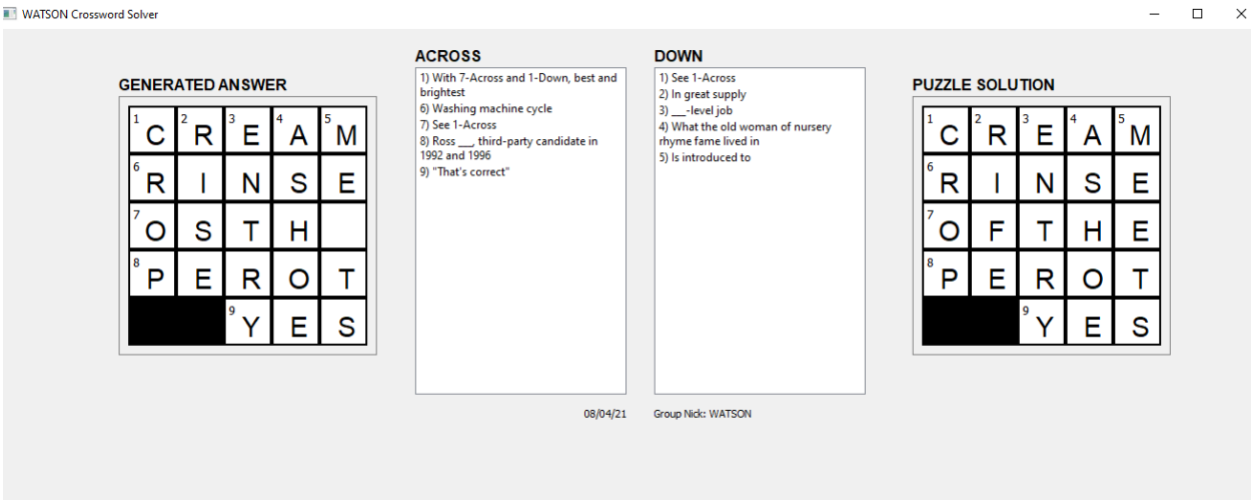



Figure 43: Result for 08/04/2021

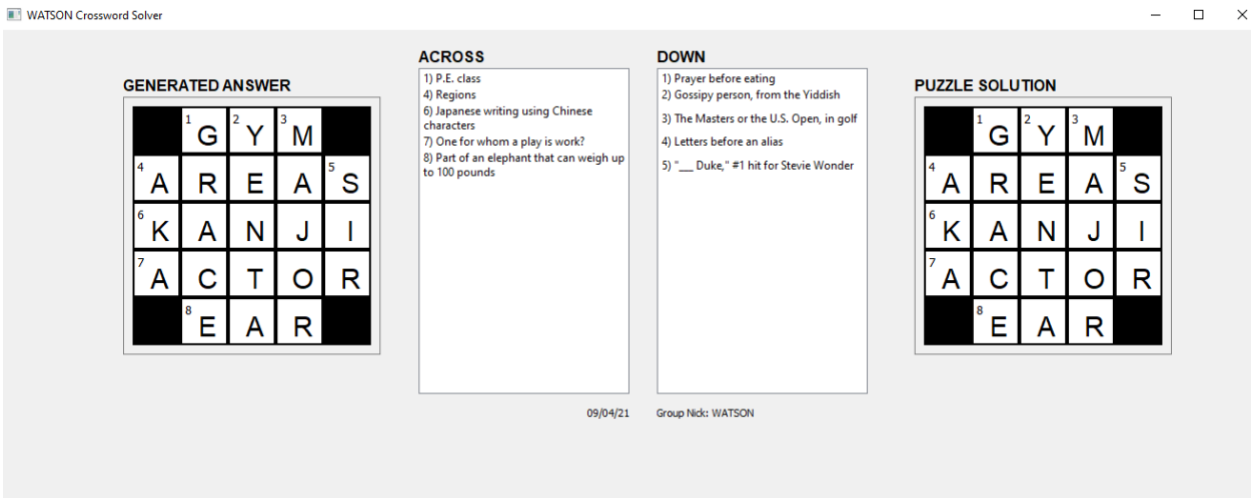



Figure 44: Result for 09/04/2021

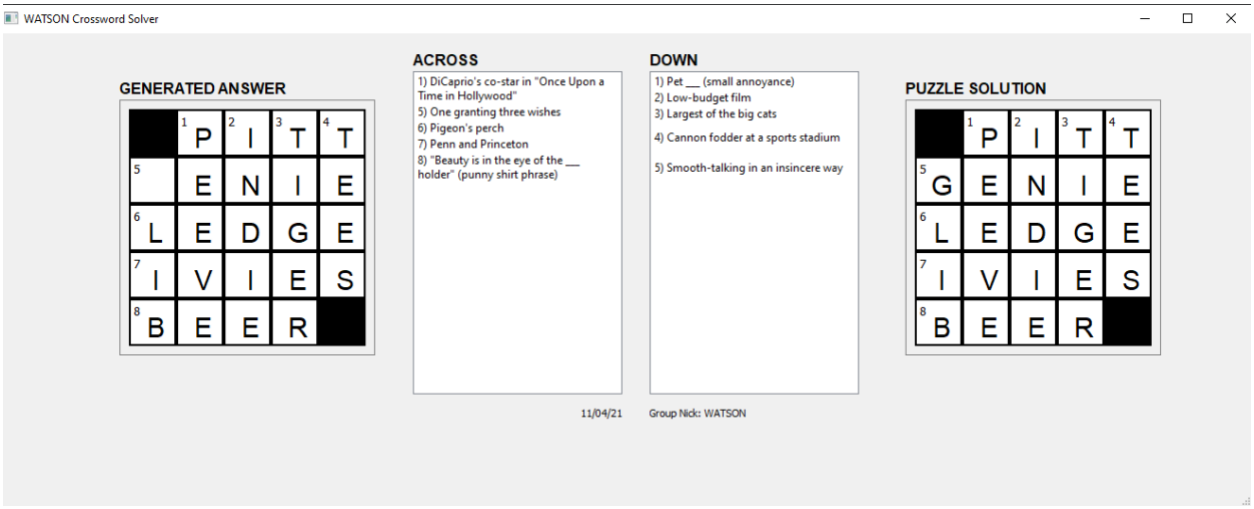



Figure 45: Result for 11/04/2021

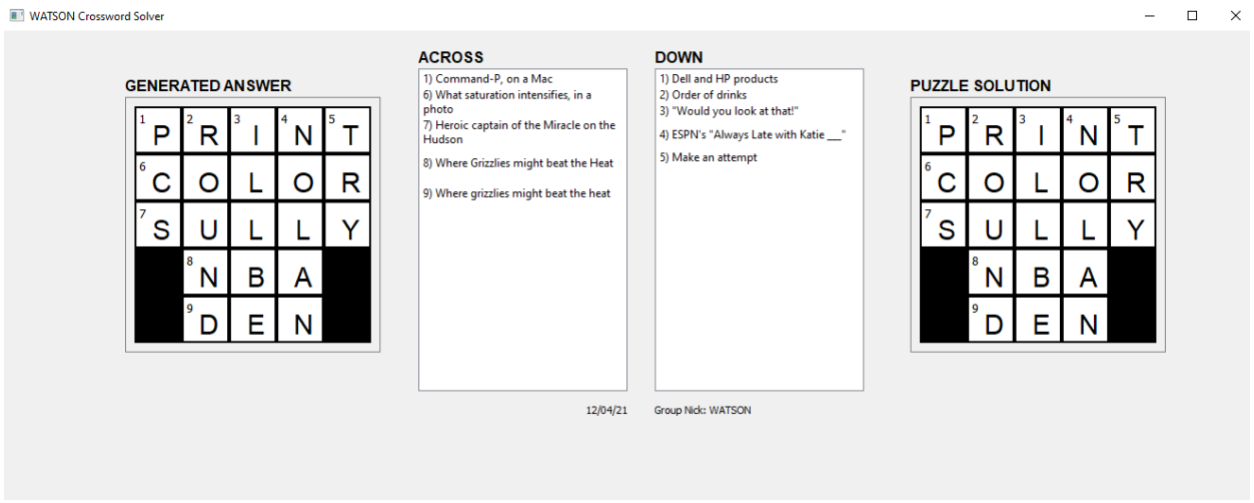



Figure 46: Result for 12/04/2021

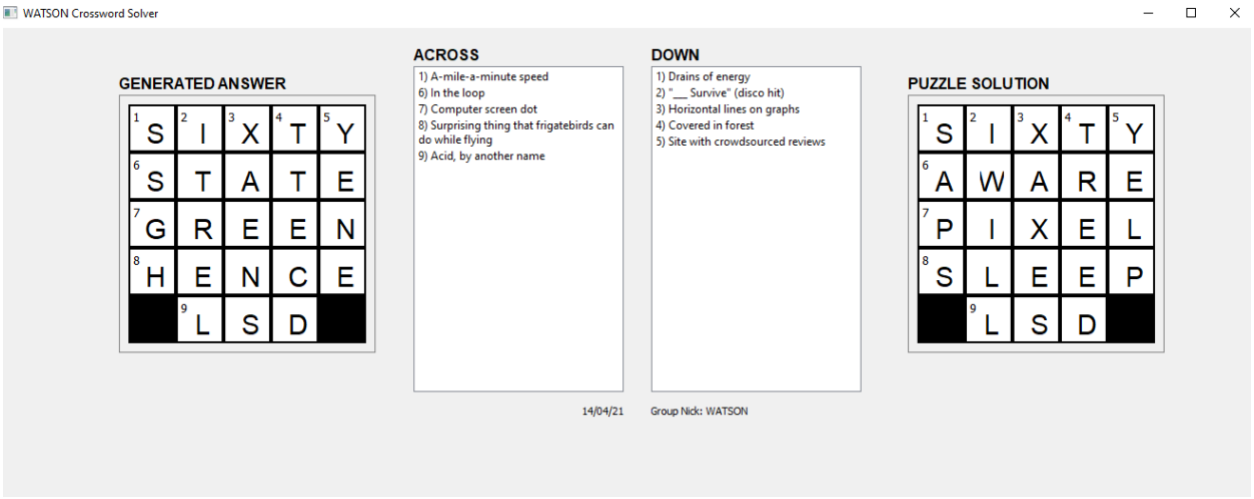



Figure 47: Result for 14/04/2021

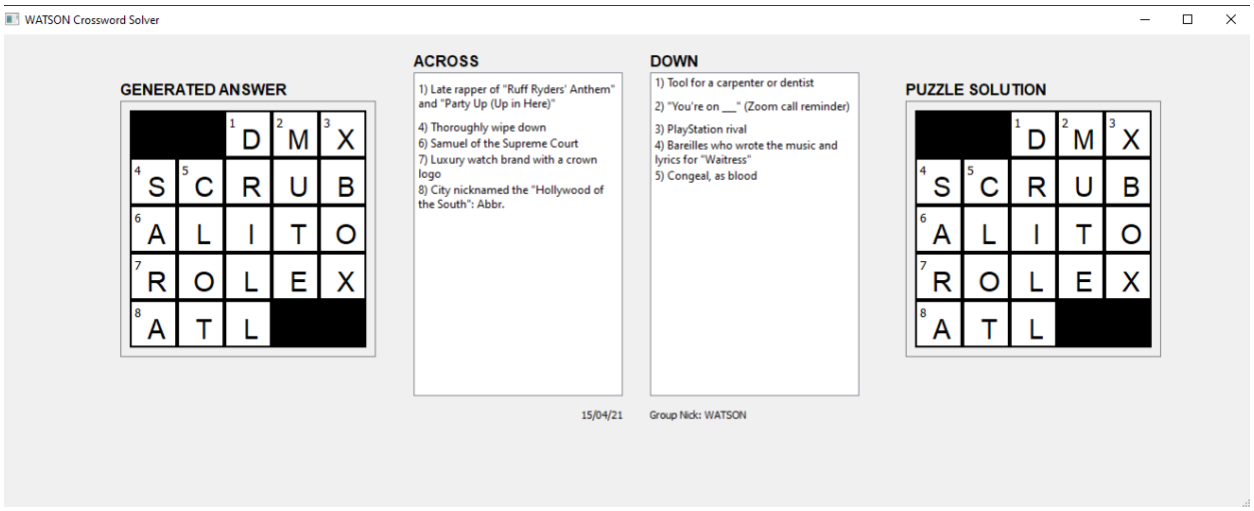



Figure 48: Result for 15/04/2021

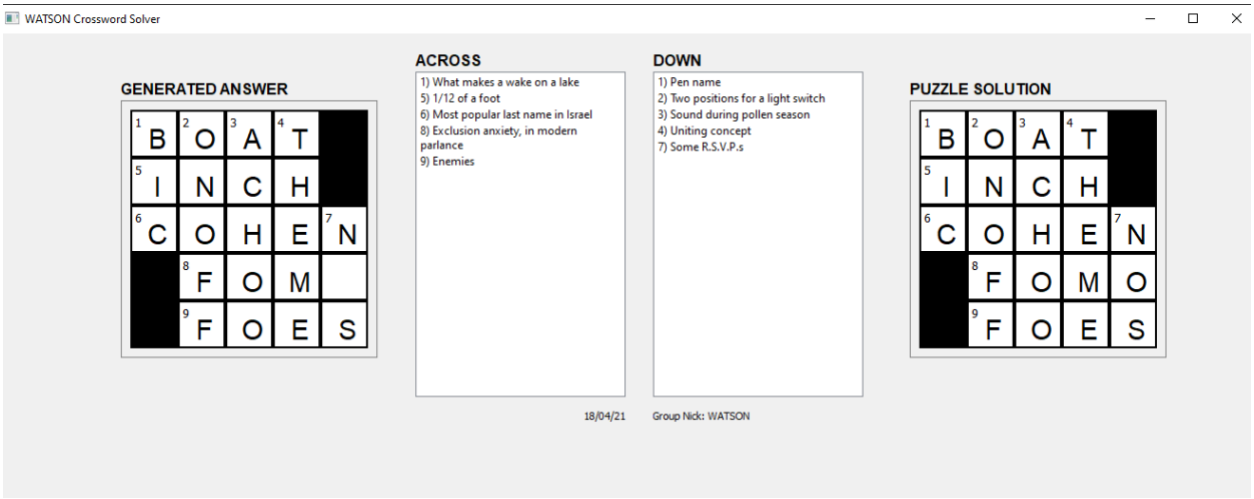



Figure 49: Result for 18/04/2021

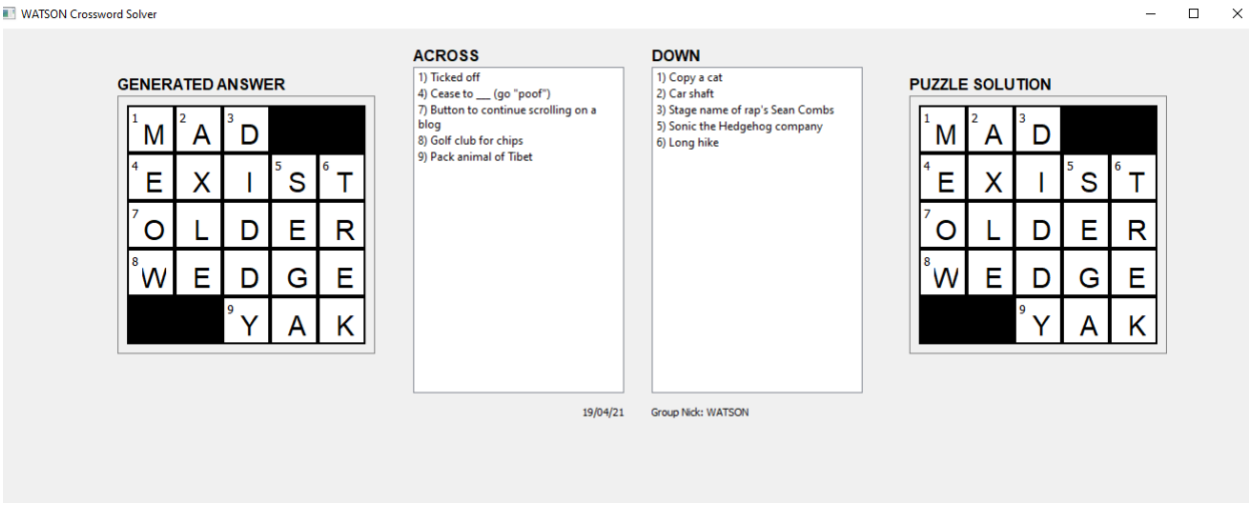



Figure 50: Result for 19/04/2021

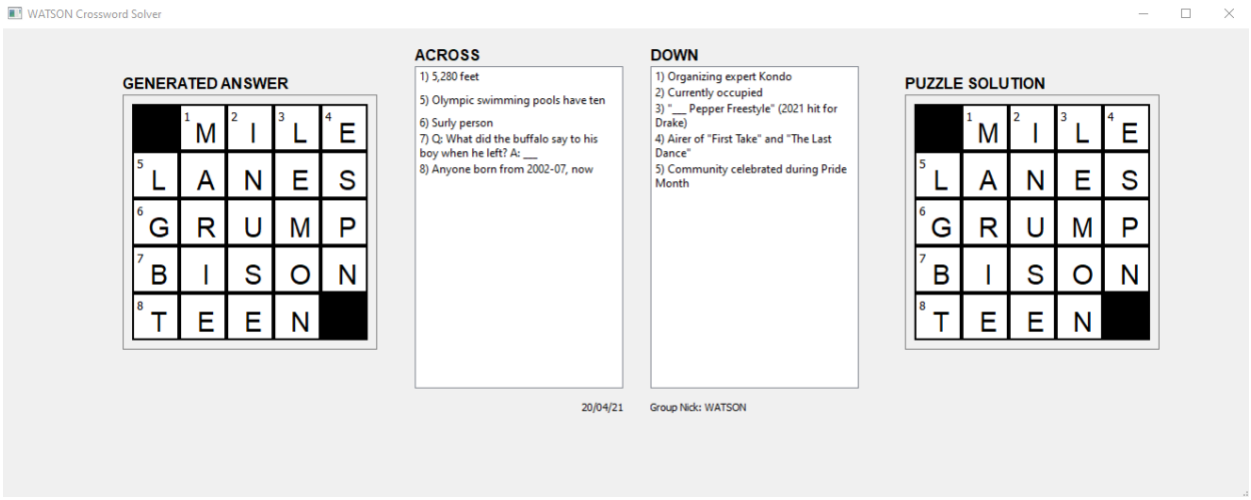



Figure 51: Result for 20/04/2021

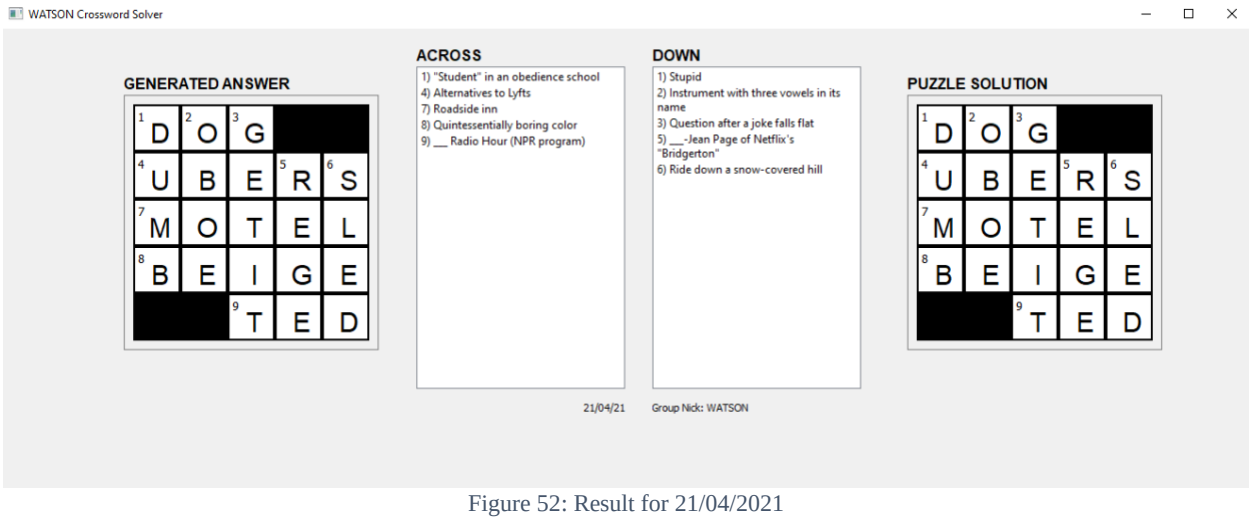



Figure 52: Result for 21/04/2021

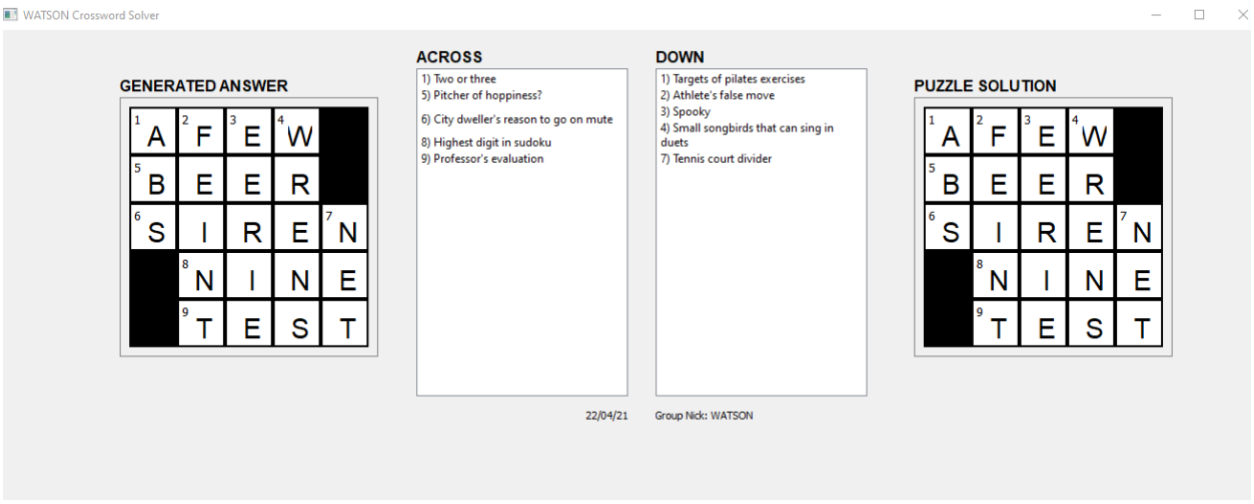



Figure 53: Result for 22/04/2021

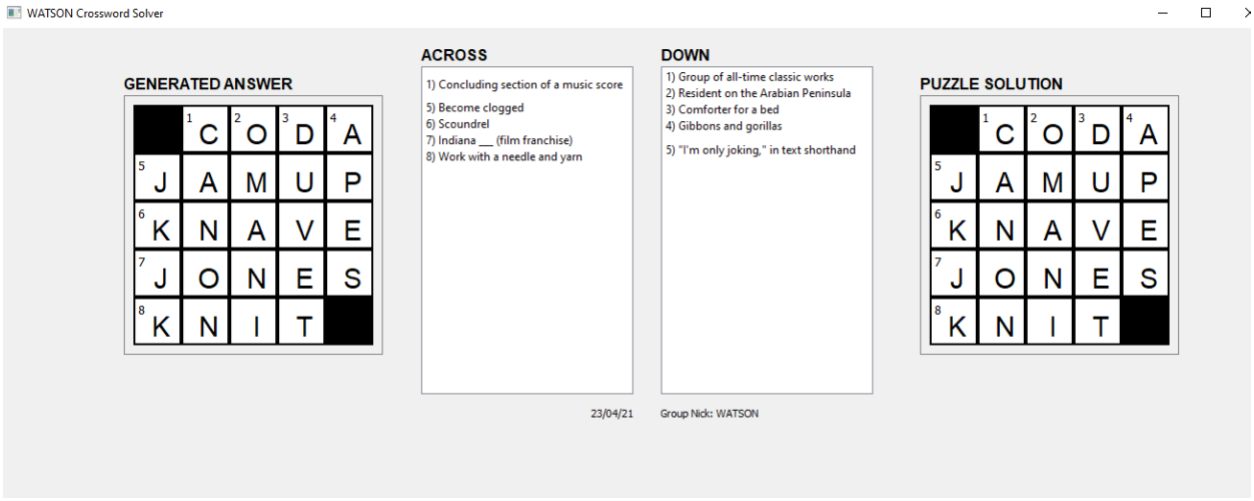



Figure 54: Result for 23/04/2021

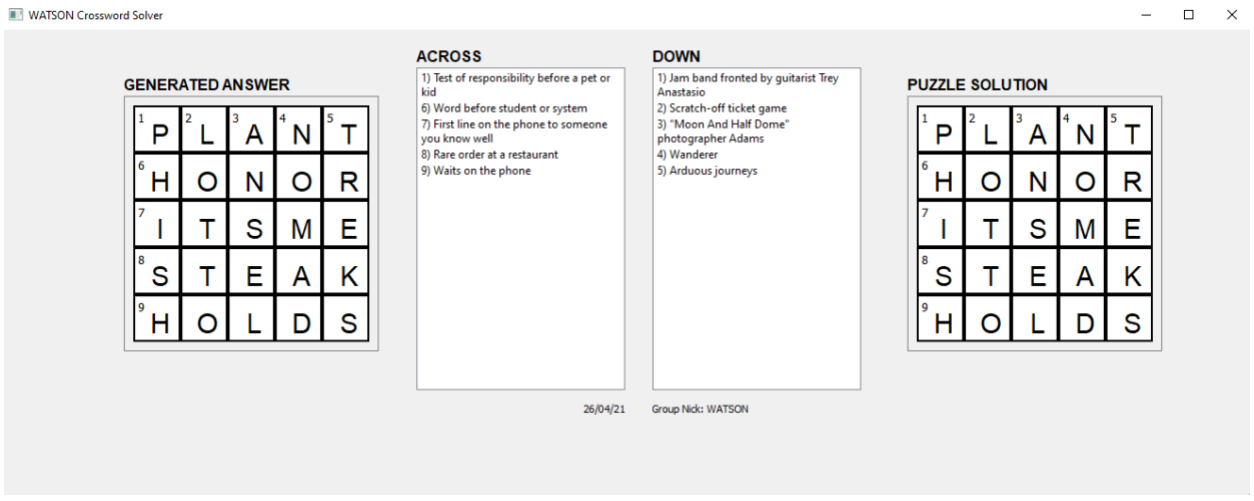



Figure 55: Result for 26/04/2021

## Appendix L – Analysis Table for Puzzle Results

| Puzzle Date | Correct Answer Amount without Adding Correct Words into the Domain | Correctness Percentage | Correct Answer Amount with Adding Correct Words into the Domain | Correctness Percentage |
|---|---|---|---|---|
| 24/03/2021 | 0 | 0% | 10 | 100% |
| 25/03/2021 | 0 | 0% | 10 | 100% |
| 26/03/2021 | 0 | 0% | 10 | 100% |
| 29/03/2021 | 0 | 0% | 10 | 100% |
| 30/03/2021 | 1 | 10% | 10 | 100% |
| 31/03/2021 | 0 | 0% | 10 | 100% |
| 01/04/2021 | 0 | 0% | 4 | 40% |
| 04/04/2021 | 0 | 0% | 10 | 100% |
| 06/04/2021 | 0 | 0% | 10 | 100% |
| 07/04/2021 | 0 | 0% | 10 | 100% |
| 08/04/2021 | 0 | 0% | 7 | 70% |
| 09/04/2021 | 0 | 0% | 10 | 100% |
| 11/04/2021 | 0 | 0% | 8 | 80% |
| 12/04/2021 | 1 | 10% | 10 | 100% |
| 14/04/2021 | 0 | 0% | 2 | 20% |
| 15/04/2021 | 2 | 20% | 10 | 100% |
| 18/04/2021 | 0 | 0% | 8 | 80% |
| 19/04/2021 | 0 | 0% | 10 | 100% |
| 20/04/2021 | 0 | 0% | 10 | 100% |
| 21/04/2021 | 1 | 10% | 10 | 100% |
| 22/04/2021 | 0 | 0% | 10 | 100% |
| 23/04/2021 | 0 | 0% | 10 | 100% |
| 26/04/2021 | 0 | 0% | 10 | 100% |

Table 1: Analysis Table of Puzzle Results with Adding Correct Answers into the Domains in Compared with Puzzle Results without Adding Correct Answers into Domains

## Appendix M – Crossword Source Code

```
from typing import List, Dict
from WikiPediaScrapper import Crawler
from Tile import Tile
import copy

class Word:
    def __init__(self, x, y, num, word_length, clue, direction):
        self.word_length = word_length
        self.crossword_y = y
        self.crossword_x = x
        self.number = num
        self.clue = clue
        self.domain = []
        self.domainChanged = True
        self.direction = direction



class Constraint:
    def __init__(self, w1, w1i, w2, w2i):
        self.word1 = w1
        self.word1index = w1i
        self.word2 = w2
        self.word2index = w2i


class Crossword:
    def __init__(self, tile_grid: List[List[Tile]], across_clues: Dict[int,
str], down_clues: Dict[int, str]):
        self.tile_grid = tile_grid
        self.across_clues = across_clues
        self.down_clues = down_clues
        self.across_word_objects: Dict[int, Word] = dict()
        self.down_word_objects: Dict[int, Word] = dict()
        self.constraints: List[Constraint] = []
        self.steps = []

    def find_domains(self):
        cw = Crawler()
        for item in self.across_word_objects.values():
            item.domain = cw.scrape(item.clue, item.word_length)
            print("Domain received for a", item.number, sep="")
            print(item.domain)

        for item in self.down_word_objects.values():
            item.domain = cw.scrape(item.clue, item.word_length)
            print("Domain received for d", item.number, sep="")
            print(item.domain)

    def find_words(self):
        across_word_objects = dict()
        down_word_objects = dict()
        for i in range(len(self.tile_grid)):
```

```
            for j in range(len(self.tile_grid[0])):
                if self.tile_grid[i][j].number is not None:
                    if ((i == 0 or self.tile_grid[i - 1][j].black)
                            and i != len(self.tile_grid) - 1
                            and not self.tile_grid[i + 1][j].black):
                        word_length = self.find_down_word_length(j, i)
                        down_word_objects[self.tile_grid[i][j].number] =
Word(j, i, self.tile_grid[i][j].number,

word_length, self.down_clues[

self.tile_grid[i][j].number], "down")

                    if ((j == 0 or self.tile_grid[i][j - 1].black)
                            and j != len(self.tile_grid[0]) - 1
                            and not self.tile_grid[i][j + 1].black):
                        word_length = self.find_across_word_length(j, i)
                        across_word_objects[self.tile_grid[i][j].number] =
Word(j, i, self.tile_grid[i][j].number,

word_length, self.across_clues[

self.tile_grid[i][j].number], "across")

        print("ACROSS WORDS:")
        for item in across_word_objects.keys():
            print(across_word_objects[item].__dict__)

        print("DOWN WORDS:")
        for item in down_word_objects.keys():
            print(down_word_objects[item].__dict__)

        self.across_word_objects = across_word_objects
        self.down_word_objects = down_word_objects

    def find_constraints(self):
        if len(self.across_word_objects) == 0:
            print("Please find the words of the puzzle first.")

        search_points = [(item, item.crossword_x, item.crossword_y) for item
in self.across_word_objects.values()]
        constraints: List[Constraint] = []

        for (word, x, y) in search_points:
            length = word.word_length
            # for each across word, examine each block from left to right.
            for i in range(x, x + length):
                # for each block, go up until you reach a block with a
number. once that is found, add a constraint.
                consnum = None
                j = 0
                for j in range(y, -1, -1):
                    if self.tile_grid[j][i].black:
                        j += 1
                        break
                    if self.tile_grid[j][i].number is not None:
                        consnum = self.tile_grid[j][i].number
```

```
                if consnum is not None:
                    new_cons = Constraint(word, i - x,
self.down_word_objects[consnum], y - j)
                    constraints.append(new_cons)

        print("CONSTRAINTS:")
        for item in constraints:
            print("a", item.word1.number, "[", item.word1index, "] = d",
item.word2.number, "[", item.word2index, "]",
                  sep="")

        self.constraints = constraints

    def consistency_check(self):
        print("entered consistency check")
        change = True
        while change:
            #print("loop 1")
            change = False
            self.constraints.sort(key=lambda x: x.word1.number)
            for constraint in self.constraints:
                #if constraint.word1.domainChanged:
                result = self.check_for_constraint(constraint, 0)
                if result:
                    change = True
            self.constraints.sort(key=lambda x: x.word2.number)
            for constraint in self.constraints:
                #if constraint.word2.domainChanged:
                result = self.check_for_constraint(constraint, 1)
                if result:
                    change = True
        for item in self.across_word_objects.values():
            print("a", item.number, " domain: ", item.domain, sep="")
            print("Number of Words: ", len(item.domain))
        for item in self.down_word_objects.values():
            print("d", item.number, " domain: ", item.domain, sep="")
            print("Number of Words: ", len(item.domain))

    def check_for_constraint(self, constraint, side):
        if side == 0:
            changeWord = constraint.word1
            changeWordIndex = constraint.word1index
            checkWord = constraint.word2
            checkWordIndex = constraint.word2index
        else:
            changeWord = constraint.word2
            changeWordIndex = constraint.word2index
            checkWord = constraint.word1
            checkWordIndex = constraint.word1index

        if len(checkWord.domain) == 0:
            return False

        domainChanged = False

        for word in changeWord.domain:
```

```
            #print("checking word")
            found = False
            for check in checkWord.domain:
                if word[changeWordIndex] == check[checkWordIndex]:
                    found = True
                    break

            if not found:
                print(word, "is removed from", changeWord.direction,
changeWord.number)
                self.steps.append(str(word) + " is removed from " +
str(changeWord.direction) + str(changeWord.number))
                changeWord.domain.remove(word)
                domainChanged = True

        changeWord.domainChanged = domainChanged

        return domainChanged

    def constraint_dfs(self):
        precedence_list = [*self.across_word_objects.values()] +
[*self.down_word_objects.values()]
        nonemptycount = 0
        for item in precedence_list:
            if(len(item.domain) > 0):
                nonemptycount += 1

        current_max = 0
        current_max_count = 0
        queue = [(dict(), precedence_list[0])]
        # queue entry structure = (word choices = dict(word, choice), current
word)

        while len(queue) > 0:
            word_choices, current_word = queue.pop(0)

            error = False
            # if any of the current word choices conflict a constraint,
disregard this solution.
            for constraint in self.constraints:
                if constraint.word1 in word_choices.keys() and
constraint.word2 in word_choices.keys():
                    if word_choices[constraint.word1][constraint.word1index]
!= word_choices[constraint.word2][constraint.word2index]:
                        error = True
                        break

            if(error):
                continue


            if len(word_choices) == nonemptycount:
                current_max = word_choices
                break
            elif len(word_choices) > current_max_count:
                current_max = word_choices
                current_max_count = len(word_choices)
```

```
            if precedence_list.index(current_word) == len(precedence_list) -
1:
                continue

            # if the word's domain is empty, just skip it.
            if len(current_word.domain) == 0:
                currentIndex = precedence_list.index(current_word)
                queue.insert(0, (word_choices, precedence_list[currentIndex +
1]))
                continue

            # for each word in current_word's domain, add it to the
dictionary as a word choice
            for word_string in current_word.domain:
                currentIndex = precedence_list.index(current_word)
                new_dict = copy.deepcopy(word_choices)
                new_dict[current_word] = word_string
                queue.insert(0, (new_dict, precedence_list[currentIndex +
1]))


        answer_grid = copy.deepcopy(self.tile_grid)
        for i in range(5):
            for j in range(5):
                if not answer_grid[i][j].black:
                    answer_grid[i][j].letter = ""

        for word in current_max.keys():
            if word.direction == "across":
                self.fill_word_across(word, current_max[word], answer_grid)
            else:
                self.fill_word_down(word, current_max[word], answer_grid)

        return answer_grid, self.steps

    def find_across_word_length(self, x, y):
        length = 0
        for i in range(x, len(self.tile_grid[0])):
            if not self.tile_grid[y][i].black:
                length += 1
            else:
                return length

        return length

    def find_down_word_length(self, x, y):
        length = 0
        for i in range(y, len(self.tile_grid)):
            if not self.tile_grid[i][x].black:
                length += 1
            else:
                return length

        return length

    def fill_word_across(self, word_obj: Word, word_str: str, answer_grid:
```

```
List[List[Tile]]):
        start_x = word_obj.crossword_x
        start_y = word_obj.crossword_y

        for x in range(start_x, start_x + word_obj.word_length):
            answer_grid[start_y][x].letter = word_str[x - start_x].upper()

    def fill_word_down(self, word_obj: Word, word_str: str, answer_grid:
List[List[Tile]]):
        start_x = word_obj.crossword_x
        start_y = word_obj.crossword_y

        for y in range(start_y, start_y + word_obj.word_length):
            answer_grid[y][start_x].letter = word_str[y - start_y].upper()
```

## Appendix N – Source Code of Tile Class

```
class Tile:
    # The Tile class is the object representation of one of the squares in
the crossword puzzle.
    # It has two attributes: number, which represents the number of the
square, or None if there is none.
    # letter represents the correct solution of the tile, or '.' if it is a
black tile.
    def __init__(self, letter=".", number=None):
        self.number = number
        self.letter = letter
        self.black = number is None and letter == "."
```

## Appendix P – Source Code of Wikipedia Search

```
from urllib.request import urlopen as uReq
# from bs4 import BeautifulSoup as soup
#
# metro_url = 'https://metro-online.pk/category/frozen-food/frozen-ready-to-
cook'
# user_agent = 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.7)
Gecko/2009021910 Firefox/3.0.7'
# headers = {'User-Agent':user_agent}
# uClient = uReq(metro_url)
# uClient.add_header=headers
# metro_html = uClient.read()
# uClient.close()
# page_soup = soup(metro_html, 'html.parser')
# print(page_soup.h1)
#
#


# import urllib.request
# from bs4 import BeautifulSoup as soup
# user_agent = 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.7)
Gecko/2009021910 Firefox/3.0.7'
#
# url = "https://metro-online.pk/category/frozen-food/frozen-ready-to-cook"
```

```
# headers={'User-Agent':user_agent,}
#
# request = urllib.request.Request(url,None,headers) #The assembled request
# response = urllib.request.urlopen(request)
# data = response.read() # The data u need
# response.close()
# page_soup = soup(data, 'html.parser')
# items = page_soup.findAll("div",{"class":"productdivinner"})
# print(items)

import time
import csv
import threading
import re
import numpy as np

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC


class Crawler:
    url = ""

    driver = webdriver

    def __init__(self,

url="https://en.wikipedia.org/w/index.php?title=Special%3ASearch&profile=advanced&fulltext=1&ns0=1&ns1=1&ns2=1&ns3=1&ns4=1&ns5=1&ns6=1&ns7=1&ns8=1&ns9=1&ns10=1&ns11=1&ns12=1&ns13=1&ns14=1&ns15=1&ns100=1&ns101=1&ns108=1&ns109=1&ns118=1&ns119=1&ns446=1&ns447=1&ns710=1&ns711=1&ns828=1&ns829=1&ns2300=1&ns2301=1&ns2302=1&ns2303=1&search="):
        self.driver =
webdriver.Chrome("/Users/berksahin/PycharmProjects/pythonProject6/chromedriver")
        print('Crawler Made...')
        self.url = url
        # self.driver.get(url)
        # self.driver.maximize_window()

    # def scroll_down(self):
    #     """A method for scrolling the page."""
    #
    #     # Get scroll height.
    #     last_height = self.driver.execute_script("return
document.body.scrollHeight")
    #
    #     while True:
    #
    #         # Scroll down to the bottom.
    #         self.driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
    #
    #         # Wait to load the page.
```

```
    #         time.sleep(2)
    #
    #         # Calculate new scroll height and compare with last scroll
height.
    #         new_height = self.driver.execute_script("return
document.body.scrollHeight")
    #
    #         if new_height == last_height:
    #             break
    #
    #         last_height = new_height

    def scrape(self, query='', word_length=5):
        elimination_regex = r'[^A-Za-z]'
        found_words = []
        start_time = time.time()
        self.driver.get(self.url + query)
        try:
            element = WebDriverWait(self.driver, 15).until(
                EC.presence_of_element_located((By.CLASS_NAME,
"searchresults"))
            )
            headings = self.driver.find_elements_by_class_name("searchmatch")
            descriptions =
self.driver.find_elements_by_class_name("searchresult")
        except:
            print('ok done')
            return
        end_time = time.time()
        #print(end_time - start_time)

        for heading in headings:
            split = heading.text.split()
            for word in split:
                filtered_word = re.sub(elimination_regex, '', word).lower()
                if len(filtered_word) is word_length:
                    if filtered_word not in found_words:
                        found_words.append(filtered_word)

        for description in descriptions:
            split = description.text.split()
            for word in split:
                filtered_word = re.sub(elimination_regex, '', word).lower()
                if len(filtered_word) is word_length:
                    if filtered_word not in found_words:
                        found_words.append(filtered_word)

        end_time = time.time()
        #print(end_time - start_time)
        return found_words

    def close(self):
        self.driver.close()


# the list the crawlers contributed to
if __name__ == "__main__":
```

```
    st_time = time.time()
    wikiCrawler = Crawler()
    words = wikiCrawler.scrape(query='This and or but if the that',
word_length=5)
    print(words)
    wikiCrawler.close()
    # TODO
    # write to csv
```

## Appendix Q – Source Code of User Interface Window

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'view.ui'
#
# Created by: PyQt5 UI code generator 5.15.3
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import Qt, QDateTime, QLocale
import math
import locale


class Ui_MainWindow(object):
    def setupUi(self, MainWindow, tile_grid, answer_tile_grid, across_clues,
down_clues):
        self.tile_grid = tile_grid
        self.answer_tile_grid = answer_tile_grid
        self.show_solution = False
        self.across_clues = across_clues
        self.down_clues = down_clues
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1327, 500)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setContextMenuPolicy(QtCore.Qt.NoContextMenu)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setContextMenuPolicy(QtCore.Qt.NoContextMenu)
        self.centralwidget.setObjectName("centralwidget")
        self.frame = QtWidgets.QFrame(self.centralwidget)
        self.frame.setGeometry(QtCore.QRect(960, 70, 270, 270))
        self.frame.setStyleSheet("border: 1px solid gray;")
        self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.frame.setFrameShadow(QtWidgets.QFrame.Raised)
        self.frame.setObjectName("frame")
        self.answer_frame = QtWidgets.QFrame(self.centralwidget)
        self.answer_frame.setGeometry(QtCore.QRect(130, 70, 270, 270))
```

```
        self.answer_frame.setStyleSheet("border: 1px solid gray;")
        self.answer_frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.answer_frame.setFrameShadow(QtWidgets.QFrame.Raised)
        self.answer_frame.setObjectName("answer_frame")
        self.grids = [QtWidgets.QWidget(self.frame) for i in range(25)]
        self.numbers = [QtWidgets.QLabel(self.grids[i]) for i in range(25)]
        self.letters = [QtWidgets.QLabel(self.grids[i]) for i in range(25)]
        self.answer_grids = [QtWidgets.QWidget(self.answer_frame) for i in
range(25)]
        self.answer_numbers = [QtWidgets.QLabel(self.answer_grids[i]) for i
in range(25)]
        self.answer_letters = [QtWidgets.QLabel(self.answer_grids[i]) for i
in range(25)]
        gridSizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        gridSizePolicy.setHorizontalStretch(0)
        gridSizePolicy.setVerticalStretch(0)
        numberSizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        numberSizePolicy.setHorizontalStretch(0)
        numberSizePolicy.setVerticalStretch(0)
        letterSizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        letterSizePolicy.setHorizontalStretch(0)
        letterSizePolicy.setVerticalStretch(0)
        numberFont = QtGui.QFont()
        numberFont.setPointSize(10)
        letterFont = QtGui.QFont()
        letterFont.setFamily("Arial")
        letterFont.setPointSize(18)
        letterFont.setBold(False)
        letterFont.setItalic(False)
        letterFont.setWeight(5)

        for i in range(25):
            self.grids[i].setGeometry(QtCore.QRect(10 + 50 * (i % 5), 10 + 50
* math.floor(i / 5), 50, 50))
            self.grids[i].setSizePolicy(gridSizePolicy)
            self.grids[i].setObjectName("grid" + str(i))
            self.numbers[i].setGeometry(QtCore.QRect(5, 2, 20, 21))
            self.numbers[i].setSizePolicy(numberSizePolicy)
            self.numbers[i].setFont(numberFont)
            self.numbers[i].setStyleSheet("border: none;")
            self.numbers[i].setObjectName("number" + str(i))
            self.letters[i].setGeometry(QtCore.QRect(16, 18, 25, 25))
            self.letters[i].setSizePolicy(letterSizePolicy)
            self.letters[i].setFont(letterFont)
            self.letters[i].setStyleSheet("border: none;\n"
                                          "font: 75 22pt \"Arial\";")
            self.letters[i].setObjectName("letter" + str(i))
            self.letters[i].setAlignment(Qt.AlignCenter | Qt.AlignVCenter)
            self.answer_grids[i].setGeometry(QtCore.QRect(10, 10, 50, 50))
            self.answer_grids[i].setGeometry(QtCore.QRect(10 + 50 * (i % 5),
10 + 50 * math.floor(i / 5), 50, 50))
            self.answer_grids[i].setObjectName("answer_grid" + str(i))
            self.answer_numbers[i].setGeometry(QtCore.QRect(5, 2, 20, 21))
            self.answer_numbers[i].setSizePolicy(numberSizePolicy)
```

```
            self.answer_numbers[i].setFont(numberFont)
            self.answer_numbers[i].setStyleSheet("border: none;")
            self.answer_numbers[i].setObjectName("answer_number" + str(i))
            self.answer_letters[i].setGeometry(QtCore.QRect(16, 18, 25, 25))
            self.answer_letters[i].setSizePolicy(letterSizePolicy)
            self.answer_letters[i].setFont(letterFont)
            self.answer_letters[i].setStyleSheet("border: none;\n"
                                                 "font: 75 22pt \"Arial\";")
            self.answer_letters[i].setObjectName("answer_letter" + str(i))
            self.answer_letters[i].setAlignment(Qt.AlignCenter |
Qt.AlignVCenter)


        self.acrossCluesList = QtWidgets.QListWidget(self.centralwidget)
        self.acrossCluesList.setGeometry(QtCore.QRect(440, 40, 221, 341))
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.acrossCluesList.sizePolicy().hasHeightForWi
dth())
        self.acrossCluesList.setSizePolicy(sizePolicy)
        self.acrossCluesList.setObjectName("acrossCluesList")
        self.acrossCluesList.setWordWrap(True)
        self.downCluesList = QtWidgets.QListWidget(self.centralwidget)
        self.downCluesList.setGeometry(QtCore.QRect(690, 40, 221, 341))
        self.downCluesList.setWordWrap(True)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.downCluesList.sizePolicy().hasHeightForWidt
h())
        self.downCluesList.setSizePolicy(sizePolicy)
        self.downCluesList.setObjectName("downCluesList")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(440, 20, 71, 16))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setTextFormat(QtCore.Qt.PlainText)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(690, 20, 61, 16))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
```

```
        self.label_3.setGeometry(QtCore.QRect(960, 50, 151, 16))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.dateTime = QtWidgets.QLabel(self.centralwidget)
        self.dateTime.setGeometry(QtCore.QRect(490, 390, 171, 20))

self.dateTime.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCor
e.Qt.AlignVCenter)
        self.dateTime.setObjectName("dateTime")
        self.label_5 = QtWidgets.QLabel(self.centralwidget)
        self.label_5.setGeometry(QtCore.QRect(690, 390, 171, 20))
        self.label_5.setObjectName("label_5")
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(130, 50, 181, 16))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.label_4.setFont(font)
        self.label_4.setObjectName("label_4")
        font = QtGui.QFont()
        font.setPointSize(10)
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(24)
        font.setBold(False)
        font.setItalic(False)
        font.setWeight(9)

        MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def fillGrid(self):
        # Fill the grid with the appropriate values.
        _translate = QtCore.QCoreApplication.translate
        i = 0
        for row in self.tile_grid:
            for tile in row:
                if tile.number is None and tile.letter == ".":
                    # This means that the grid must be black. First, paint
the grid black,
                    # then set the strings of letter and number labels to an
empty string.
                    self.grids[i].setStyleSheet("background-color: rgb(0, 0,
0);\n"
                                                "border-color: rgb(0, 0,
```

```
0);\n"
                                                    "border: 2px solid black;")
                    self.grids[i].show()
                    self.numbers[i].setText(_translate("MainWindow", ""))
                    self.letters[i].setText(_translate("MainWindow", ""))
                elif tile.number is None and tile.letter != ".":
                    # This means that the grid is a numberless white grid.
First, paint the grid white,
                    # then leave the number label empty but fill the solution
label with the correct letter.
                    self.grids[i].setStyleSheet("background-color: rgb(255,
255, 255);\n"
                                                    "border-color: rgb(0, 0,
0);\n"
                                                    "border: 2px solid black;")
                    self.numbers[i].setText(_translate("MainWindow", ""))
                    self.letters[i].setText(_translate("MainWindow",
tile.letter))
                else:
                    # This means that the grid is a numbered white grid.
First, paint the grid white,
                    # then fill the number and letter labels with the correct
values.
                    self.grids[i].setStyleSheet("background-color: rgb(255,
255, 255);\n"
                                                    "border-color: rgb(0, 0,
0);\n"
                                                    "border: 2px solid black;")
                    self.numbers[i].setText(_translate("MainWindow",
str(tile.number)))
                    self.letters[i].setText(_translate("MainWindow",
tile.letter))
                i = i + 1

        i = 0
        for row in self.answer_tile_grid:
            for tile in row:
                if tile.number is None and tile.letter == ".":
                    # This means that the grid must be black. First, paint
the grid black,
                    # then set the strings of letter and number labels to an
empty string.
                    self.answer_grids[i].setStyleSheet("background-color:
rgb(0, 0, 0);\n"
                                                    "border-color: rgb(0, 0,
0);\n"
                                                    "border: 2px solid black;")
                    self.answer_grids[i].show()
                    self.answer_numbers[i].setText(_translate("MainWindow",
""))
                    self.answer_letters[i].setText(_translate("MainWindow",
""))
                elif tile.number is None and tile.letter != ".":
                    # This means that the grid is a numberless white grid.
First, paint the grid white,
                    # then leave the number label empty but fill the solution
label with the correct letter.
```

```
                    self.answer_grids[i].setStyleSheet("background-color:
rgb(255, 255, 255);\n"
                                                       "border-color: rgb(0, 0,
0);\n"
                                                       "border: 2px solid black;")
                    self.answer_numbers[i].setText(_translate("MainWindow",
""))
                    self.answer_letters[i].setText(_translate("MainWindow",
tile.letter))
                else:
                    # This means that the grid is a numbered white grid.
First, paint the grid white,
                    # then fill the number and letter labels with the correct
values.
                    self.answer_grids[i].setStyleSheet("background-color:
rgb(255, 255, 255);\n"
                                                       "border-color: rgb(0, 0,
0);\n"
                                                       "border: 2px solid black;")
                    self.answer_numbers[i].setText(_translate("MainWindow",
str(tile.number)))
                    self.answer_letters[i].setText(_translate("MainWindow",
tile.letter))
                i = i + 1

    def fillClueList(self):
        self.acrossCluesList.addItems(self.across_clues)
        self.downCluesList.addItems(self.down_clues)

    def retranslateUi(self, MainWindow):

        now = QDateTime.currentDateTime()
        dt_locale = QLocale(QLocale.English, QLocale.UnitedKingdom)
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "WATSON Crossword
Solver"))
        self.label.setText(_translate("MainWindow", "ACROSS"))
        self.label_2.setText(_translate("MainWindow", "DOWN"))
        self.label_3.setText(_translate("MainWindow", "PUZZLE SOLUTION"))
        self.dateTime.setText(_translate("MainWindow",
dt_locale.toString(now, "d/M/yyyy, hh:mm")))
        self.label_5.setText(_translate("MainWindow", "Group Nick: WATSON"))
        self.label_4.setText(_translate("MainWindow", "GENERATED ANSWER"))

        self.fillGrid()
        self.fillClueList()
```

## Appendix R – Source Code of Testing Past Puzzles

```
import json
import re
from Tile import Tile

SQUARE_GRID_LENGTH = 5
class JSONTileGridParser:
    def __init__(self, filepath):
```

```
        self.filepath = filepath

    def parse(self):
        with open(self.filepath) as file:
            jsonstr = file.read()

            puzzle = json.loads(jsonstr)
            tile_grid = []
            answer_tile_grid = []

            for i in range(SQUARE_GRID_LENGTH):
                new_empty_row = []
                new_row = []
                for j in range(SQUARE_GRID_LENGTH):

                    jsontile = puzzle["grid"][i][j]
                    if jsontile["number"] is None:
                        number = None
                    else:
                        number = int(jsontile["number"])

                    if jsontile["black"]:
                        new_empty_tile = Tile(jsontile["letter"], number)
                    else:
                        new_empty_tile = Tile("0", number)
                    new_tile = Tile(jsontile["letter"], number)
                    new_empty_row.append(new_empty_tile)
                    new_row.append(new_tile)
                tile_grid.append(new_empty_row)
                answer_tile_grid.append(new_row)

            across_clues = dict()
            down_clues = dict()

            for item in puzzle["across_clues"]:
                search = re.search(r"([0-9]+)\) (.+)", item)
                number = int(search.group(1))
                clue = search.group(2)
                across_clues[number] = clue

            for item in puzzle["down_clues"]:
                search = re.search(r"([0-9]+)\) (.+)", item)
                number = int(search.group(1))
                clue = search.group(2)
                down_clues[number] = clue

            return tile_grid, answer_tile_grid, across_clues, down_clues,
puzzle["across_clues"], puzzle["down_clues"]


if __name__ == "__main__":
    js = JSONTileGridParser("2021-04-14.json")

    (tile_grid, answer_tile_grid, across_clues, down_clues) = js.parse()
    print(tile_grid)
    print(answer_tile_grid)
```

```
    print(across_clues)
    print(down_clues)
```

## Appendix S – Source Code of Trace Wiindow

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'TraceWindow.ui'
#
# Created by: PyQt5 UI code generator 5.15.3
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow, traceText):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1321, 900)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.traceText = QtWidgets.QTextEdit(self.centralwidget)
        self.traceText.setGeometry(QtCore.QRect(10, 10, 1301, 841))
        self.traceText.setObjectName("traceText")
        self.traceText.setText(traceText)
        self.traceText.setLineWrapMode(True)
        self.traceText.setReadOnly(True)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1321, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Trace Window"))
```

## Appendix T – Source Code of Main Code

```
# This is the submission of the project group named WATSON for Demo 1. Coded
on Python 3.9.1.
# Selenium and PyQt5 packages are required to run.

# Members:
#           Mehmet Berk Şahin (CONTACT)
#           Balaj Saleem
#           Mehmet Alper Genç
```

```
#           Ege Hakan Karaağaç
#           Fırat Yönak

from PyQt5 import QtWidgets
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.common.exceptions import NoSuchElementException

from Crossword import Crossword
from JSONTileGridParser import JSONTileGridParser
from Tile import Tile
import copy
import view
import window
import sys
import time

# To run this code, please install:
# 1- Google Chrome
# 2- Chromedriver with the same major version number as the Google Chrome
installed.
# Then, update the DRIVER_PATH variable below with the full path of your
chromedriver.exe.
DRIVER_PATH = '/Users/berksahin/PycharmProjects/pythonProject6/chromedriver'
SQUARE_GRID_LENGTH = 5
SINGLE_STEP = True

class ContentWrapper:
    def __init__(self, content):
        self.content = content
        self.index = 0

    def getContent(self):
        if (self.index < len(self.content)):
            self.index = self.index + 1
            out = self.content[0:self.index]
            out = ' '.join(out)
            return out
        else:
            out = ' '.join(self.content)
            return out

    def setContent(self, content):
        self.content = content

    def addContent(self, item):
        self.content.append(item)

if __name__ == "__main__":
    print(len(sys.argv))
    print(sys.argv[0])
    if len(sys.argv) == 2:
        js = JSONTileGridParser(sys.argv[1])
        (tile_grid, answer_tile_grid, acrossCluesDict, downCluesDict,
acrossClues, downClues) = js.parse()
    else:
        options = Options()
```

```
        options.add_experimental_option("prefs",

{"profile.default_content_settings.cookies": 2})  # this disables cookies

        driver = webdriver.Chrome(executable_path=DRIVER_PATH,
options=options)  # load chrome driver from driver path
        print("Connecting to website...")
        driver.get('https://www.nytimes.com/crosswords/game/mini')  # open
mini crossword screen
        time.sleep(5)
        # when opening the mini crossword screen without a login, the user
can see one of 3 things:
        # 1. a blue colored button with the text "OK"
        # 2. a black colored button with the text "Continue" that goes to
signup screen, and a "Play without an account"
        # link
        # 3. the same as option 2, but without the play without an account
link. the code below first tries to find an
        # "OK" button to check if it is option 1, and if found, clicks the
button. if not, tries to find the
        # "play without an account" button and if found, clicks it. if
neither of these are found, the code reloads the
        # driver to try again, until one of the first 2 options is found.
        while True:
            try:
                driver.find_element_by_css_selector("[aria-
label=OK]").click()
                break
            except NoSuchElementException as e:
                try:
                    driver.find_element_by_xpath('//button[text()="Play
without an account"]').click()
                    break
                except NoSuchElementException as e2:
                    print("Neither OK button nor the 'Play without an
account' link have been found. "
                          "Reloading driver to try again...")
                    driver.quit()
                    driver = webdriver.Chrome(executable_path=DRIVER_PATH,
options=options)

driver.get('https://www.nytimes.com/crosswords/game/mini')

        # click the "Reveal" button.
        driver.find_element_by_css_selector("[aria-label=reveal]").click()
        # click the "Puzzle" button.
        driver.find_element_by_link_text("Puzzle").click()
        # finally, click the new "Reveal" button that pops up to reveal the
solution.
        driver.find_element_by_css_selector("[aria-label=Reveal]").click()

        # capture the grid and solution, and put them into a 2D array of Tile
objects.
        print("Receiving puzzle grid and solution...")
        table_cells = driver.find_element_by_id("xwd-
board").find_element_by_css_selector(
            "[data-group=\"cells\"]").find_elements_by_xpath("./*")
```

```
        x = 0
        y = 0
        tile_grid = []
        answer_tile_grid = []
        for cell in table_cells:
            cell_texts = cell.find_elements_by_tag_name("text")
            new_tile = None
            empty_tile = None
            if len(cell_texts) == 0:
                new_tile = Tile()
                empty_tile = Tile()
            elif len(cell_texts) == 2:
                cell_letter = cell_texts[0].text.replace("\n", "")
                new_tile = Tile(cell_letter.rstrip())
                empty_tile = Tile("0")
            elif len(cell_texts) == 4:
                cell_number = cell_texts[0].text[0]
                cell_letter = cell_texts[2].text[0]
                new_tile = Tile(cell_letter.rstrip(),
int(cell_number.rstrip()))
                empty_tile = Tile("0", int(cell_number.rstrip()))
            else:
                print(
                    "Something unexpected occurred while downloading the grid
and solution. The program will now exit.")
                exit(-1)

            if x == 0:
                answer_tile_grid.append([new_tile])
                tile_grid.append([empty_tile])
            else:
                answer_tile_grid[y].append(new_tile)
                tile_grid[y].append(empty_tile)

            x = x + 1
            if x == SQUARE_GRID_LENGTH:
                y = y + 1
                x = 0

        print("The puzzle grid and solution have been received
successfully.")
        # Get the clues
        print("Receiving clues...")
        across_ol = driver.find_element_by_xpath(

"/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/article/section[2]/div[
1]/ol")
        down_ol = driver.find_element_by_xpath(

"/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/article/section[2]/div[
2]/ol")
        acrossClues = []
        acrossCluesDict = dict()
        downClues = []
        downCluesDict = dict()

        # Put the across and down clues into separate lists
```

```
        for child in across_ol.find_elements_by_xpath("*"):
            spans = child.find_elements_by_xpath("*")
            number = spans[0].text
            clue = spans[1].text
            acrossCluesDict[int(number)] = clue
            acrossClues.append(number + ") " + clue)

        for child in down_ol.find_elements_by_xpath("*"):
            spans = child.find_elements_by_xpath("*")
            number = spans[0].text
            clue = spans[1].text
            downCluesDict[int(number)] = clue
            downClues.append(number + ") " + clue)

    print("Clues have been received successfully.")

    print(acrossCluesDict)
    print(downCluesDict)
    cw = Crossword(tile_grid, acrossCluesDict, downCluesDict)
    cw.find_words()
    cw.find_constraints()
    cw.find_domains()
    cw.consistency_check()
    tile_grid, steps = cw.constraint_dfs()
    # Finally, the program creates the user interface with the data gathered.

    print('-================ STEPS ==================-')
    #print(steps)
    outString = '\n'.join(steps)
    #print(outString)


    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    MainWindow2 = QtWidgets.QMainWindow()
    c2 = ContentWrapper(["STEPS: "])
    ui = view.Ui_MainWindow()
    ui2 = window.Ui_MainWindow()
    ui.setupUi(MainWindow, answer_tile_grid, tile_grid, acrossClues,
downClues)
    ui2.setupUi(MainWindow2, "STEPS", c2, ui2.refresh)
    MainWindow.show()
    MainWindow2.show()
    if (SINGLE_STEP):
        for s in steps:
            out = (s)
            c2.addContent(out + '\n')
    else:
        c2.addContent(steps)
    print("Displaying application UI.")
    app.exec_()
    print("UI window closed. Program will now exit.")
```

## BIBLIOGRAPHY

[1] K. Thanasuan and S. T. Mueller, "Crossword expertise as recognitional decision making: an artificial intelligence approach," *Frontiers in Psychology*, vol. 5, 2014. [Online]. Available: https://internal-journal.frontiersin.org/articles/10.3389/fpsyg.2014.01018/full. [Accessed: May 3, 2021].

[2] T. R. Shultz and L. Egri, "Constraint-Satisfaction Models

[3] C. Boyce-Jacino and S. DeDeo, "Opacity, obscurity, and the geometry of question-asking," *Cognition*, vol. 196, p. 104071, 2020.