**Ayhan Okuyan** ayhan.okuyan[at]ug.bilkent.edu.tr

**Barış Akçin** baris.akcin[at]ug.bilkent.edu.tr

**Berkan Özdamar** berkan.ozdamar[at]ug.bilkent.edu.tr

**Deniz Doğanay** deniz.doganay[at]ug.bilkent.edu.tr

**Mustafa Bay** mustafa.bay[at]ug.bilkent.edu.tr

**CS 461 – Term Project report swaplıano**



Table of Contents

[Introduction 2](#_Toc28354958)

[Working Principle of Algorithm 2](#_Toc28354959)

[Retrieving the Clues 2](#_Toc28354960)

[Preprocessing 3](#_Toc28354961)

[Final Adjustments and Precautions 4](#_Toc28354962)

[Dictionaries Used 6](#_Toc28354963)

[Dictionary.com 6](#_Toc28354964)

[Oxford Dictionary 7](#_Toc28354965)

[Merriam-Webster 7](#_Toc28354966)

[Wordnet 7](#_Toc28354967)

[Urban Dictionary 7](#_Toc28354968)

[Sample Crossword Puzzles with Generated Clues 8](#_Toc28354969)

[Conclusion 11](#_Toc28354970)

[References 11](#_Toc28354971)

[Word Count 12](#_Toc28354972)

[Appendix 13](#_Toc28354973)

# Introduction

Artificial Intelligence, AI, is referred as the intelligence that is adopted by machines. This is a roof term for all intelligent behavior demonstrated by machines, meaning showing actions that look to be done by an intelligent being. Hence, in this project, we have implemented an AI algorithm in order to obtain a puzzle from the NYTimes webpage [1] and then to create alternate versions of the clues that are presented there. We have followed a deterministic approach and not a learning-based approach throughout the project, meaning we have used an algorithm that would obtain the clues from sources and not generated them using Natural Language Processing. The project consists of two main stages. One is the obtain the crossword puzzle from the website and create a UI that resembles the one in the webpage. Second is to retrieve the answers from the puzzle structure and generate clues. For this project, we have chosen to use the Python (3) programming language for its flexibility and its already existing APIs that would ease the programming load.

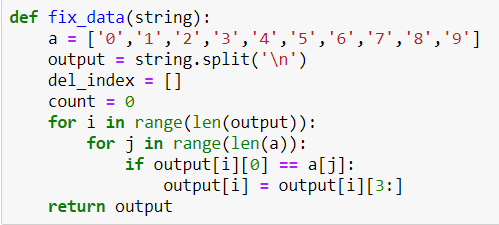
Through the first part, we have retrieved the clues using Selenium API and its webdriver capabilities, preprocessed the data and retrieved the structural form of the puzzle. Then we have created the user interface via the tkinter package of Python. Further, we have obtained various definitions of the word from different sources again using Selenium and then ranked them for quality of the sources which are determined by ourselves to make the algorithm work in resemblance with the clues that are presented by Fagliano in the second part. This report presents the second part of the project, working principles of the AI algorithm created and the sources used in detail and not the first part.

# Working Principle of Algorithm

In the algorithm, different clues are retrieved from various dictionaries. We have chosen to use dictionaries since the we decided that the definitions of the words to be the most reliable clues that we could generate in a deterministic environment. After retrieving the clues, a preprocessing step is applied for each dictionary in order to eliminate unwanted clues. After the preprocessing step, we assigned weights to different dictionaries in order to give some dictionaries priority over the others in terms of their quality. Finally, we are randomly select a clue from the list of possible clues. Each step will be analyzed below in detail.

## Retrieving the Clues

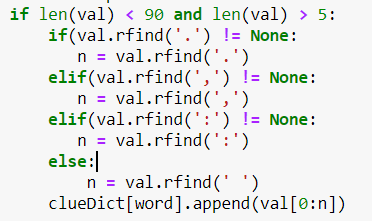
While retrieving the clues, a feature selection is applied to the clues. Since the structure of definition of each word differs in dictionaries, a “fix\_data” function is implemented which is given below.



**Figure 1 –** “fix\_data” Method

The problem we faced while retrieving clues is sometimes rather than retrieving a single clue, the algorithm obtained multiple clues at once and appended all those clues into a single clue. This is the reason behind the implementation of this function. Function “fix\_data” takes each clue as an input and then checks the splits in the string done by the newline character (“\n”) in the clue and separates it according to these newline characters. Additionally, it also eliminates the numbers at the beginning of each clue if it has a newline character in it, since in some websites the bullet numbers used for definitions were given in the same string with the clue. By using this function, we were able to separate each string which had multiple clues in it. Also, getting rid of the numbers is a method used in the Rigutini et.al. [2].

In Fagliano’s puzzle, the clues are usually between 3-5 words long. Since our objective is to imitate his behavior in preparing the clues, we have eliminated the long clues. This is done by limiting the length of each clue obtained. We limited the length of clue as minimum 5 characters and maximum 90 characters. These limits were determined by trial and error, by observing the character sizes given in the clues. If a clue exceeds the limit, algorithm retrieves the clue, but it doesn’t store the string as a clue. The algorithm we have used to limit the clue length is given below.



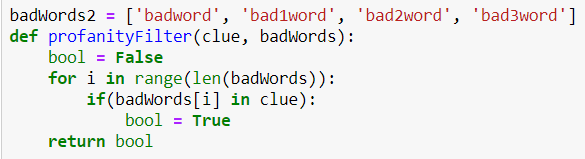
**Figure 2 –** Limiting Clues

As it can be seen from Figure 2, algorithm takes strings between 5 and 90 in terms of their lengths.

Another problem we faced is that sometimes algorithm couldn’t find clues less than 90 or more than 5 characters. In order to solve that problem, we have taken the strings which had more than 90 characters, but we have split them from a 90-character limit. This may cause last words of the clues divided or some meaningless word(s) at the end. In order to solve this problem, we have looked for the last indexes of dot, comma, colon and space respectively. If it finds dot, then it will cut the clue from the dot which would mean a complete sentence. If it can’t find the dot, then it looks for comma. This implementation helps us to obtain meaningful clues when we cannot find any clues in our nominal range.

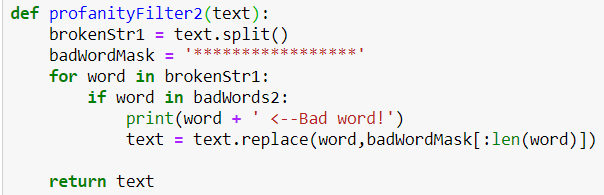
## Preprocessing

In the preprocessing step of the algorithm, profanity is checked by using two different functions “profanityFilter” and “profanityFilter2” whom their purposes differ. Profanity is checked after the clues from given dictionaries are retrieved. The reason behind having two different profanity filters is the possibility of eliminating all clues. If all clues are eliminated by “profanityFilter”, then algorithm uses “profanityFilter2” which changes the words that are inappropriate with ‘\*\*\*’. The two “profanityFilter” functions are given below and explained in detail.



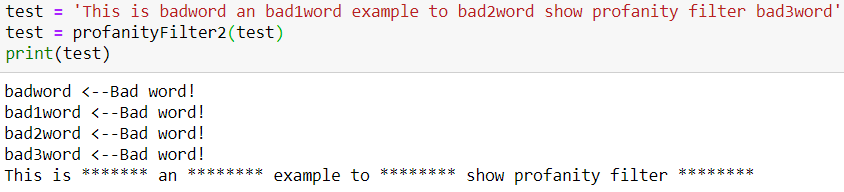
**Figure 3** – “profanityFilter” Function

badWords2 list consists of profanity words, but for the sake of this report we changed it into sample bad words in order to show the working principle. Function “profanityFilter” checks the inappropriate words and if there is one, it removes the clue. This will improve the clues by eliminating clues which has profanity in it.



**Figure 4 – “**profanityFilter2” Function

In the profanityFilter2 function, the algorithm takes a clue as an input and then checks the profanity inside the clue. Then, it converts the profanity word to ‘\*\*\*’. An example is given below;



**Figure 5 –** Sample Output for profanityFilter2 Function

As it can be seen from Image 5, profanityFilter2 detects badword, bad1word, bad2word and bad3word. Afterwards, it changes the test sentence by placing profanities to strings consisted of ‘\*’ with the same length as the original word. Profanity filters help us to improve the clues according to the purpose of the crossword puzzle by Joel Fagliano.

## Final Adjustments and Precautions

After the preprocessing step, we are checking the number of generated clues if we have enough clues at hand. If any word exist that doesn’t have any clue, then we are separating the words by taking the first letter and then the rest or and by taking the last letter and then the rest. The underlying reason behind this is we observed that some words in Fagliano’s puzzles are given with “a/an” prefixes or “s” suffix for singularity and plurality. This method is shown below.



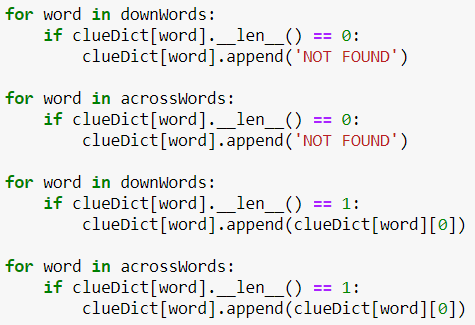
**Figure 6-** Rest of the Word and Last Character

As it can be seen from Figure 6, if there are no clues for the specific word, we are dividing the word into two parts and then search for the word without last character. We are trying to retrieve clue(s) for the rest of the word if it is possible. Afterwards we added the last character of the word to the clue. Also, in the Eli Pincus & David Traum’s paper [3] they used a method which like our methos and they put specific same words after the finding clue but we just add last character of our word. This helps us to increase the probability to find the clues for all words. For example, clue for word ‘SNL’ could not be found directly. This method divides the word in ‘SN’ and ‘S’. Then, it searches dictionaries for ‘SN’ word and finds the clue as ‘Say Nothing’. Finally, it converts the clue as ‘Say Nothing + L’. This also enables us to create more “tricky” clues instead of conventional clues.



**Figure 7-** First Character and Rest of the Word

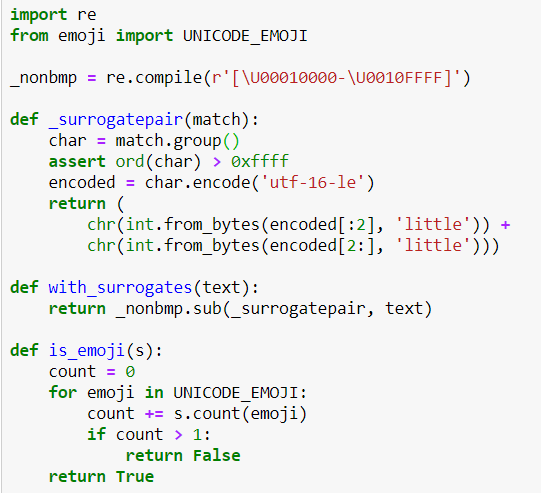
As it can be observed from Figure 7, we are also checking for the word which is divided in the first character and the rest of the word. Rest of the word’s clue is retrieved if possible and then first character is added to the beginning of the clue. For example, clue word ‘AKIND’ could not be found directly with our prior trials. This method divides the word into ‘A’ and ‘KIND’. Then, it searches dictionaries for ‘KIND’ word and finds the clue as ‘of a good or benevolent nature or disposition, as a person’. Finally, it converts the clue as ‘A + of a good or benevolent nature or disposition, as a person’. Joel Fagliano is not using these kinds of clues in the crossword puzzle as we observed, hence in the project, we are using five different dictionaries in order to avoid these word plays as much as possible in order to accomplish crossword puzzle’s purpose. Additionally, in order to eliminate any possible errors due to empty clue list and one clue existing for a word we implemented an algorithm which is given below.



**Figure 8-** Eliminating Errors

After retrieving clue process is done, algorithm seen in Image 8 finds empty clues and append them ‘NOT FOUND’ in order to show that our algorithm couldn’t find any clues for that specific word.

Finally, we faced with another problem related with printing clues. We are using tkinter package to create UI in our code and printing emoji to generated puzzle creates encoding error, so we needed to modify the emojis in order to print them on the puzzle we generated. These functions search for an emoji in the retrieved clue and converts its Unicode to JavaScript to print it. The code is given below.



**Figure 9-** Emoji Functions

# Dictionaries Used

Used dictionaries will be analyzed in subheadings by explaining the reason behind using that specific dictionary and how we manage it are discussed throughout this section.

## Dictionary.com

Dictionary.com is the world’s leading digital dictionary. Dictionary.com’s main, proprietary source is the Random House Unabridged Dictionary, which is continually updated by Dictionary.com’s team of experienced lexicographers and supplemented with trusted, established sources including American Heritage and Harper Collins to support a range of language needs.

Since Dictionary.com’s definitions are similar to hardcopy dictionaries, we wanted to get our first definitions from dictionary.com to ensure that most of the clues obtained first are always nearly meaningful and good clues. However, crossword puzzles should be tricky, and these clues are always straightforward, which was done with other dictionaries. Also, in this dictionary we couldn’t recover some clues like famous people’s names or most of the words from other languages. As an example, when we searched the word “Howie” in this dictionary, we couldn’t find any results for Howie Mandel. And when we searched “Poulet”, which is chicken in French, definition of this word cannot be found. Also, dictionary.com has no API, or an offline version, so Selenium library is used to acquire those definitions. We get only the text description of the words by using Selenium.

## Oxford Dictionary

Since all the dictionaries differ from each other, we wanted to use some other online dictionary sites to verify our word definition. For that, another trusted dictionary site is Oxford Dictionary. Like Dictionary.com, Oxford Dictionary couldn’t find famous people’s names and words from other languages. However, we used Oxford Dictionary because it adds variety to our clue pool. Thanks to this variety we can improve our clues because one of the strategy of choosing clue of our program is looking through the length of the clue and when we cannot find a proper clue from dictionary.com, we can find a proper clue for our program from the Oxford Dictionary.

## Merriam-Webster

In order to further strengthen our clue base, we have used a third dictionary, which is Merriam-Webster. Like we did with dictionary.com and Oxford Dictionary, we get the dictionary meanings of the of the words from Merriam-Webster. In this dictionary we used same algorithm with Dictionary.com and Oxford using Selenium.

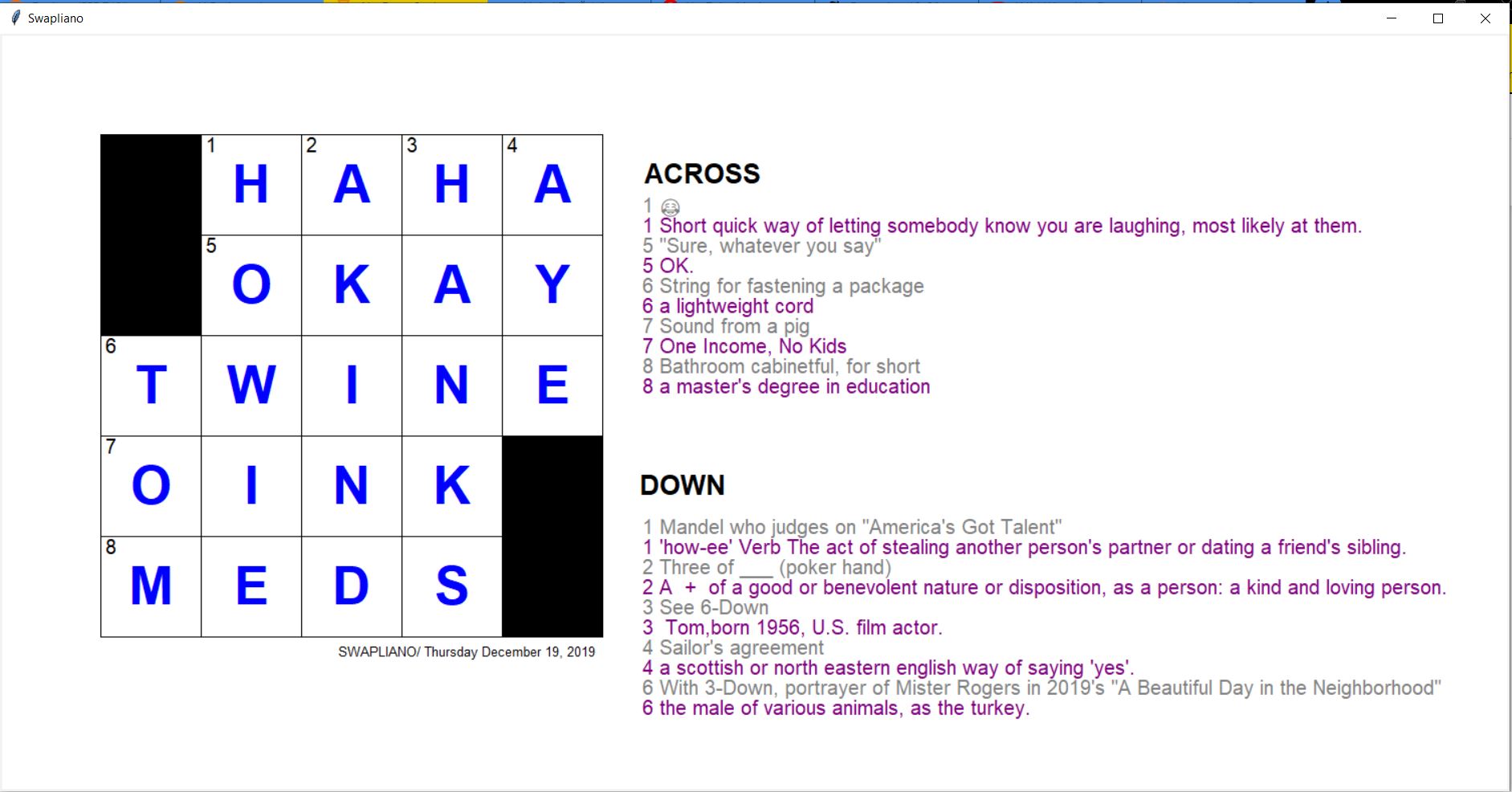
## Wordnet

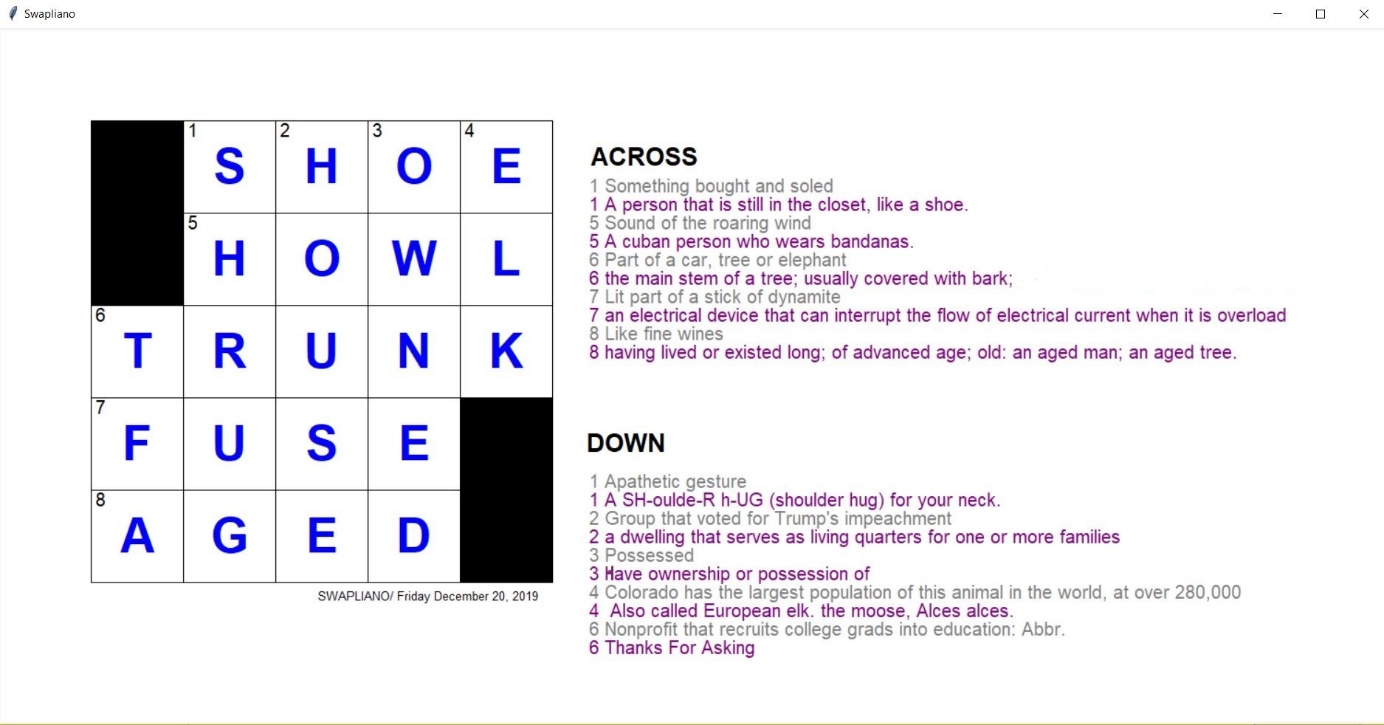
Wordnet is a lexical database of English. Wordnet contains synonyms for nouns, verbs and adjectives. With the help of Wordnet, semantic and lexical relations from words are easily found. Using this property, we were able to collect definitions which is not hardcopy definition but semantically related. With those definitions, we were able to manipulate our clues. In this case, we used Wordnet’s offline library. Also, in Aoife Aherne and Carl Vogel’s paper [4] they stated that they were using wordnet because it has open sources code and offline mode.

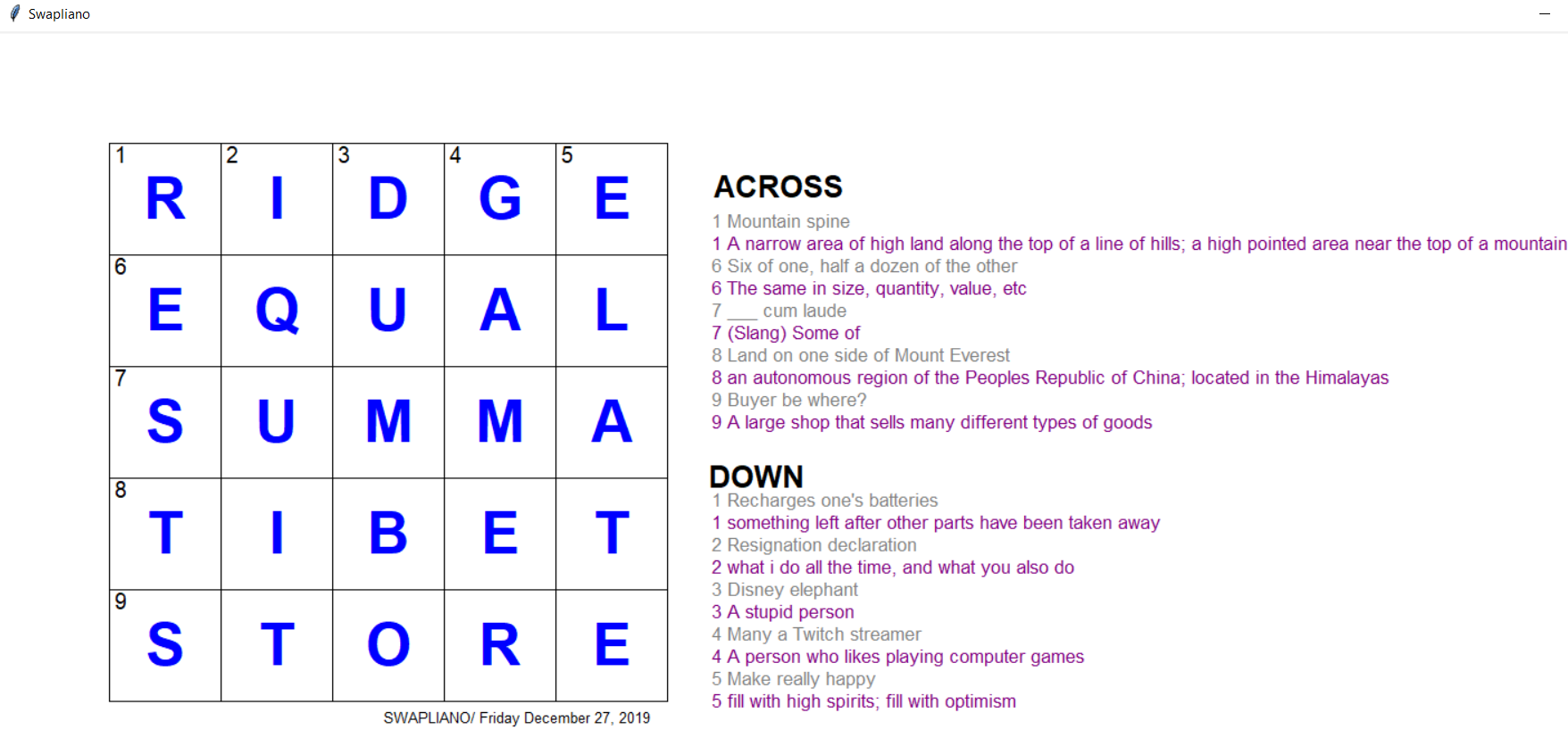
## Urban Dictionary

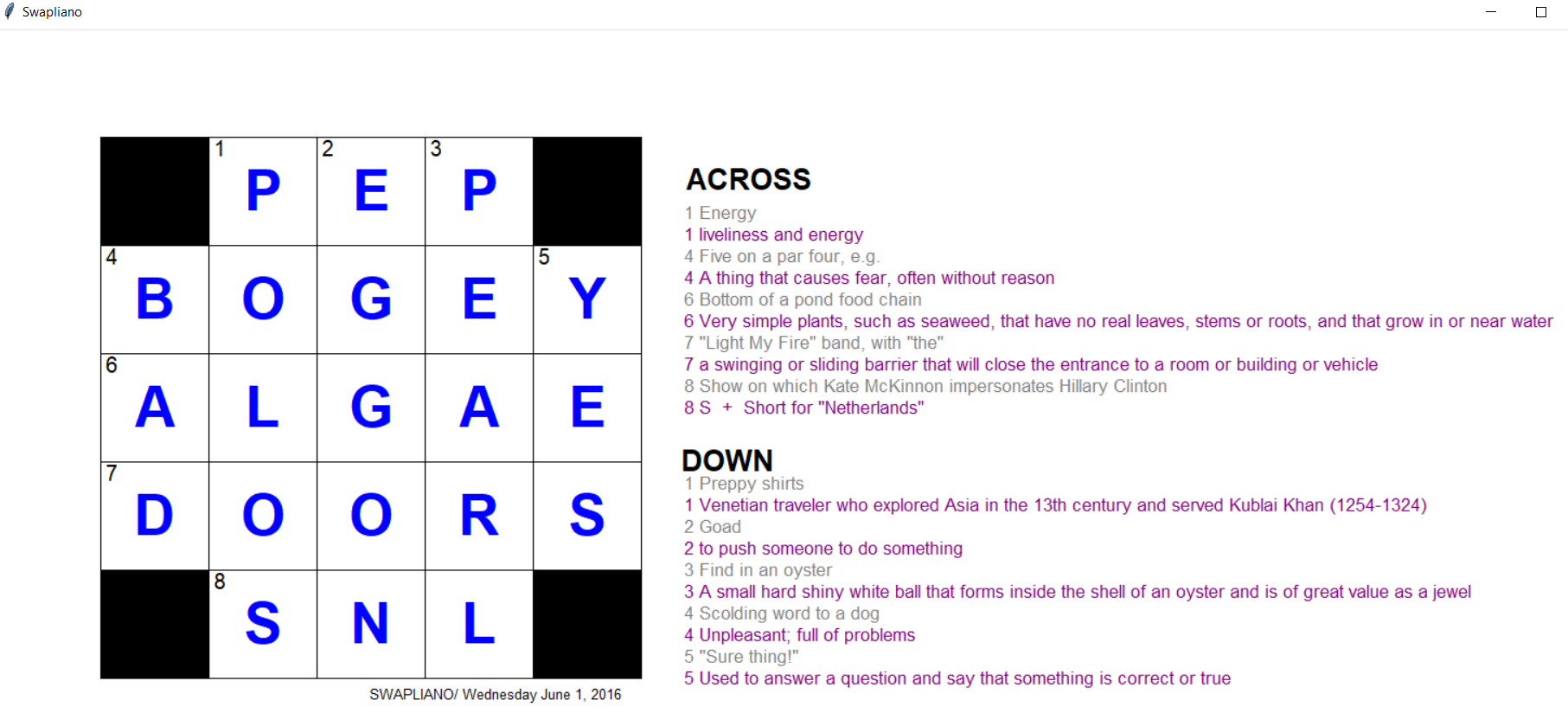
Urban Dictionary contains almost all sorts of words and provides some informal definitions to them. We have used Urban Dictionary as our last option. If we could not find any data from the previously described dictionaries, we have used Urban Dictionary to get a definition. But the problem with the Urban Dictionary is that, definitions available on that site contains lots of swearing and cuss words. Since we do not want that kind of strong language in our clues, we have used the profanityFilter to eliminate them. We also use Selenium here, with the addition of the filters.

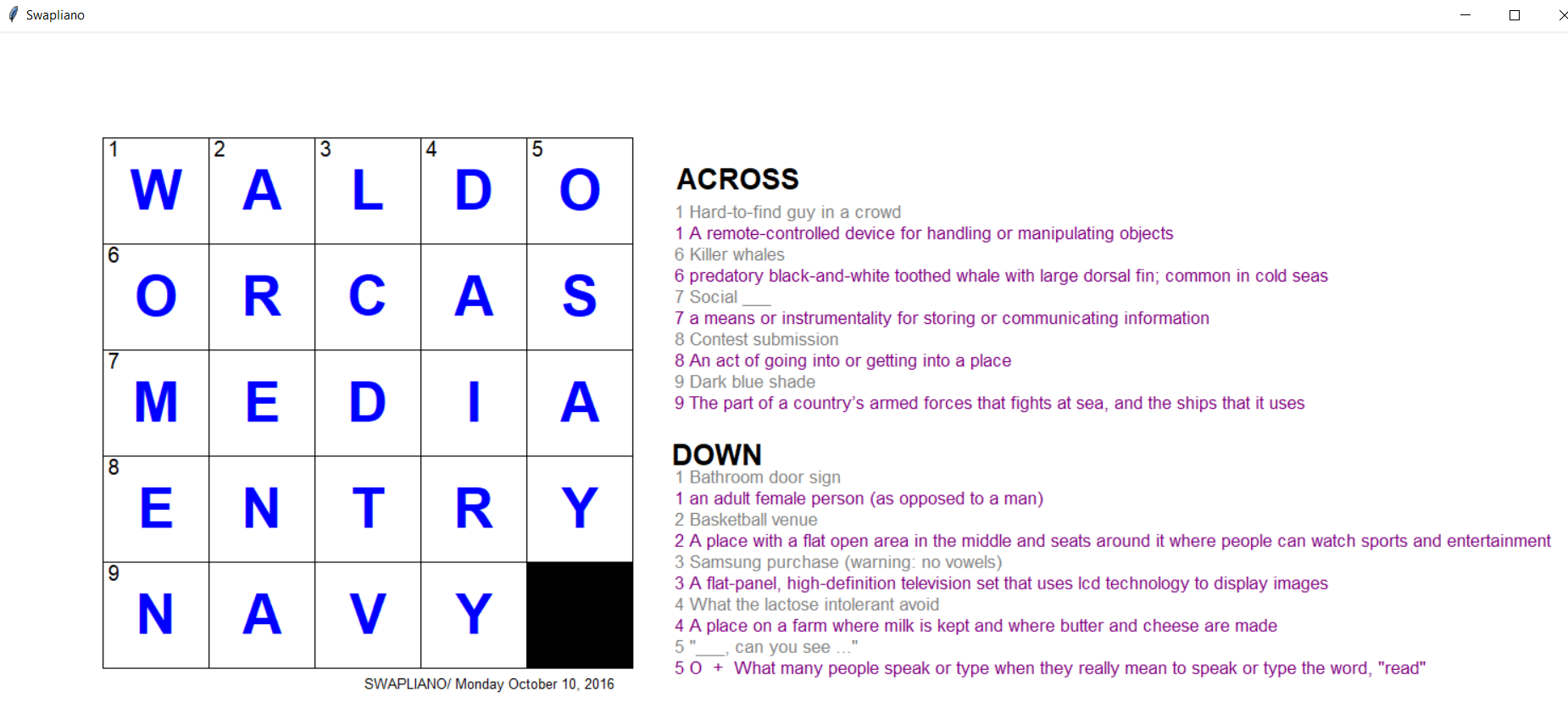
# Sample Crossword Puzzles with Generated Clues

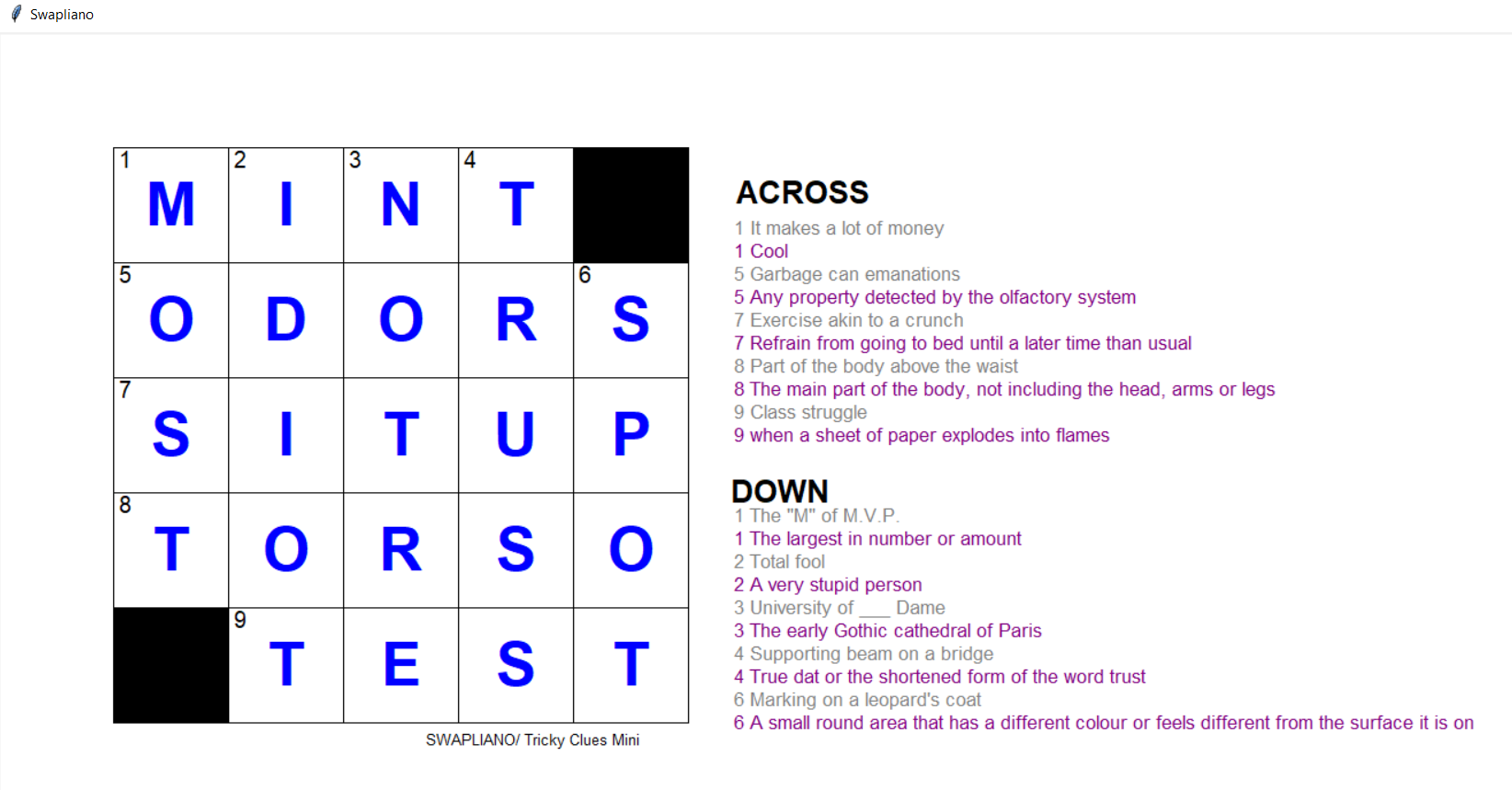




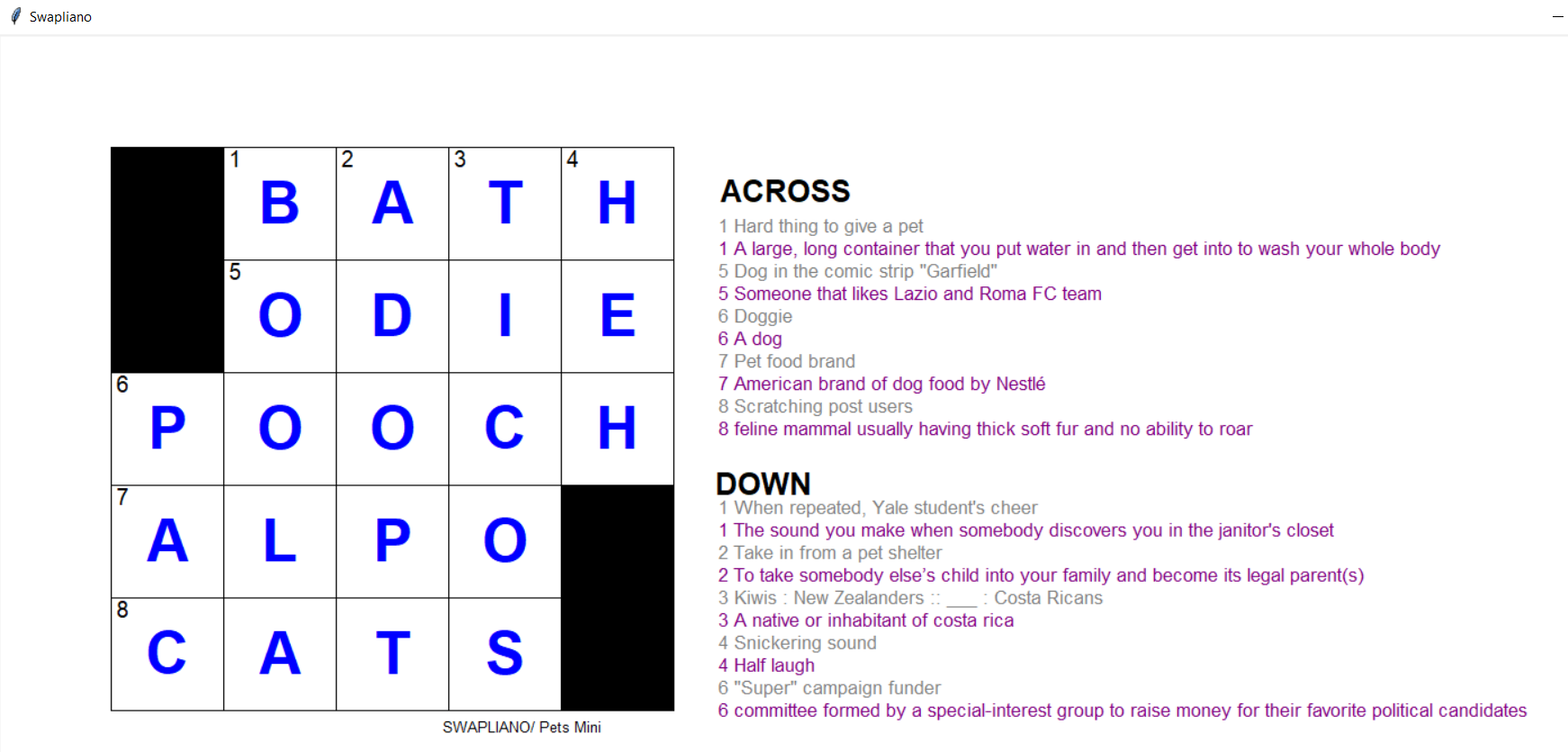












# Conclusion

To conclude, we have attempted to generate alternative clues for Joel Fagliano’s Mini Crosswords Game in New York Times. Keeping the newly generated clues in Joel Fagliano’s style was our biggest problem. However, we have implemented specific functions in order to fix the generated clues and make them fit into his style. Some keywords were hard to come up with a suitable clue, such as plural words and names of people which are known by an ordinary US citizen, but not so well known for a non-American. In those cases, the clues we generated from Urban Dictionary have eased the work for us. However, other than those cases, we have tried not to use it as much as we can, since the site contains heavy language. In the end, we have come up with a close to perfect program, which takes answers from Joel Fagliano’s game, stores every answer, searches different sources to generate new clues, prioritizes the generated clues in the order of reliability of the sources, and prints that specific day’s puzzle in its revealed form and original clues; with our new clues, each of which are below to their relative original clue. Overall, it was a challenging project that required hard work and we believe that it was successful.

# References

[1] J. Fagliano, “The New York Times Mini Crossword,” *The New York Times*. [Online]. Available: https://www.nytimes.com/crosswords/game/mini. [Accessed: 26-Dec-2019].

[2] Rigutini, L., Diligenti, M., Maggini, M. and Gori, M. (2019). Automatic Generation of Crossword Puzzles. [online] researchgate. Available at: https://www.researchgate.net/publication/229589133\_Automatic\_Generation\_of\_Crossword\_Puzzles [Accessed 26 Dec. 2019].

[3] Pincus, E. and Traum, D. (2019). Towards Automatic Identification of Effective Clues for Team Word-Guessing Games. [online] Aclweb.org. Available at: https://www.aclweb.org/anthology/L16-1435.pdf [Accessed 27 Dec. 2019].

[4] Aherne, A. and Vogel, C. (2019). Wordnet Enhanced Automatic Crossword Generation. [online] Tara.tcd.ie. Available at: http://www.tara.tcd.ie/bitstream/handle/2262/32246/wordnet+enhanced+automatic.pdf;jsessionid=7407BF11B8B26C313B375ED05D6FFA9B?sequence=1 [Accessed 26 Dec. 2019].

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

# Word Count

This article consists of 2438 words.

# Appendix

#!/usr/bin/env python  
# coding: utf-8  
  
# In[1]:  
  
  
from selenium import webdriver  
from datetime import datetime  
from datetime import date  
import time  
import numpy as np  
from tkinter import Tk, Canvas, BOTH, Label  
  
now = datetime.now()  
driver = webdriver.Chrome('chromedriver/chromedriver.exe')  
  
# https://www.nytimes.com/crosswords/game/special/tricky-clues-mini  
driver.get('https://www.nytimes.com/crosswords/game/special/pets-mini')  
time.sleep(0.2)  
driver.find\_element\_by\_xpath(  
 "/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/div[2]/div[3]/div/article/div[2]/button/div/span").click()  
time.sleep(0.2)  
driver.find\_element\_by\_xpath("/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/div/ul/div[2]/li[2]/button").click()  
time.sleep(0.2)  
driver.find\_element\_by\_xpath(  
 "/html/body/div[1]/div/div/div[4]/div/main/div[2]/div/div/ul/div[2]/li[2]/ul/li[3]/a").click()  
time.sleep(0.2)  
driver.find\_element\_by\_xpath("/html/body/div[1]/div/div[2]/div[2]/article/div[2]/button[2]/div").click()  
time.sleep(0.2)  
driver.find\_element\_by\_xpath("/html/body/div[1]/div/div[2]/div[2]/span").click()  
  
hint\_list = {} # fill later  
keywords = ("Across", "Down")  
  
index = 0  
prev\_no = 0  
  
clues\_of\_across = []  
clues\_of\_down = []  
  
clues = driver.find\_elements\_by\_class\_name("Clue-text--3lZl7")  
numbers = driver.find\_elements\_by\_class\_name("Clue-label--2IdMY")  
  
for number, clue in zip(numbers, clues):  
 no = number.get\_property("textContent")  
 content = clue.get\_property("textContent")  
 if (int(no) < prev\_no):  
 index = 1  
 s = keywords[index] + ":\t" + no + " " + content + "\n"  
 print(s)  
 if index == 0:  
 clues\_of\_across.append([keywords[0], no, content])  
 else:  
 clues\_of\_down.append([keywords[1], no, content])  
 prev\_no = int(no)  
  
reveals = {}  
for i in range(25):  
 reveal = driver.find\_element\_by\_id("cell-id-{i}".format(i=i))  
 reveal\_sibs = reveal.get\_property("parentNode").get\_property("childElementCount")  
 if reveal\_sibs == 1:  
 s = str(i + 1) + ":\tblack" + "\n"  
 reveals[i + 1] = ("black", "", "")  
 elif reveal\_sibs == 3:  
 value = reveal.get\_property("parentNode").get\_property("childNodes")[1].get\_property("textContent")  
 s = str(i + 1) + ":\twhite " + value + "\n"  
  
 reveals[i + 1] = ("white", value, "")  
 elif reveal\_sibs == 4:  
 number = reveal.get\_property("parentNode").get\_property("childNodes")[1].get\_property("textContent")  
 value = reveal.get\_property("parentNode").get\_property("childNodes")[2].get\_property("textContent")  
 s = str(i + 1) + ":\twhite " + value + " " + number + "\n"  
 reveals[i + 1] = ("white", value, number)  
  
print(reveals)  
  
# drawing shape part  
# x = 0  
# root = Tk()  
# canvas = Canvas(root, width=1500, height=750)  
# canvas.configure(bg="white")  
  
# for i in range(5):  
# for j in range(5):  
# x += 1  
# if reveals[x][0] == "black":  
# canvas.create\_rectangle(100 \* (j + 1), 100 \* (i + 1), 100 + (100 \* (j + 1)), 100 + (100 \* (i + 1)), fill="black")  
# elif reveals[x][2] != "":  
# canvas.create\_rectangle(100 \* (j + 1), 100 \* (i + 1), 100 + (100 \* (j + 1)), 100 + (100 \* (i + 1)), fill="white")  
# canvas.create\_text((100 \* (j + 1)) + 50, (100 \* (i + 1)) + 50, fill="blue", font="Arial 40 bold", text=reveals[x][1][0])  
# canvas.create\_text((100 \* (j + 1)) + 10, (100 \* (i + 1)) + 10, fill="black", font="Arial 15 ", text=reveals[x][2])  
# else:  
# canvas.create\_rectangle(100 \* (j + 1), 100 \* (i + 1), 100 + (100 \* (j + 1)), 100 + (100 \* (i + 1)), fill="white")  
# canvas.create\_text((100 \* (j + 1)) + 50, (100 \* (i + 1)) + 50, fill="blue", font="Arial 40 bold",text=reveals[x][1][0])  
  
# #canvas.create\_text(700, 140, fill="black", font="Arial 20 bold", text="ACROSS")  
# #  
# #for i in range(len(clues\_of\_across)):  
# # canvas.create\_text(640, 170 + (i\*20), fill="gray", font="Arial 15", text=clues\_of\_across[i][1] + " " + clues\_of\_across[i][2], anchor="w")  
# #  
# #canvas.create\_text(680, 170 + (len(clues\_of\_across)+2)\*20, fill="black", font="Arial 20 bold", text="DOWN")  
# #  
# #for i in range(len(clues\_of\_down)):  
# # canvas.create\_text(640, 170 + (len(clues\_of\_across)+4)\*20 + (i\*20), fill="gray", font="Arial 15", text=clues\_of\_down[i][1] + " " + clues\_of\_down[i][2], anchor="w")  
# #  
# #  
date = driver.find\_element\_by\_class\_name("PuzzleDetails-date--1HNzj").get\_attribute("textContent")  
# canvas.create\_text(465, 615, fill="black", font="Arial 10", text="SWAPLIANO/ " + date)  
# canvas.pack(fill=BOTH, expand=1)  
  
# root.title("Swapliano")  
  
downWords = []  
acrossWords = []  
aIndex = []  
dIndex = []  
  
for i in range(5):  
 aWord = ''  
 isFirst = True  
 for k in range(5):  
 (c1, l1, n1) = reveals[i \* 5 + k + 1]  
 aWord += l1[0:1]  
 if isFirst and n1 != '':  
 aIndex.append(n1)  
 isFirst = False  
 acrossWords.append(aWord)  
  
for i in range(5):  
 dWord = ''  
 isFirst = True  
 for k in range(5):  
 (c2, l2, n2) = reveals[k \* 5 + i + 1]  
 dWord += l2[0:1]  
 if isFirst and n2 != '':  
 dIndex.append(n2)  
 isFirst = False  
 downWords.append(dWord)  
print(downWords)  
print(acrossWords)  
aSort = np.argsort(np.asarray(aIndex).astype(int))  
dSort = np.argsort(np.asarray(dIndex).astype(int))  
print(aSort)  
print(dSort)  
  
downWords = np.asarray(downWords)[dSort]  
acrossWords = np.asarray(acrossWords)[aSort]  
  
print(downWords)  
print(acrossWords)  
  
  
# In[2]:  
  
  
def fix\_data(string):  
 a = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']  
 output = string.split('\n')  
 for i in range(len(output)):  
 for j in range(len(a)):  
 if output[i][0] == a[j]:  
 output[i] = output[i][3:]  
 return output  
  
  
from collections import defaultdict  
  
clueDict = defaultdict(list)  
  
for word in acrossWords:  
 clueDict[word] = []  
 driver.get('https://www.oxfordlearnersdictionaries.com/definition/english/' + str.lower(word) + '?q=' + str.lower(word))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '//\*[@id="' + str.lower(word) + '\_def\_1"]').get\_property(  
 'textContent')  
 except:  
 break  
 temp = temp.capitalize()  
 clueDict[word].append(temp)  
  
  
for word in downWords:  
 clueDict[word] = []  
 driver.get(  
 'https://www.oxfordlearnersdictionaries.com/definition/english/' + str.lower(word) + '?q=' + str.lower(word))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '//\*[@id="' + str.lower(word) + '\_def\_1"]').get\_property(  
 'textContent')  
 except:  
 break  
 temp = temp.capitalize()  
 clueDict[word].append(temp)  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
  
for word in acrossWords:  
 driver.get('https://www.merriam-webster.com/dictionary/' + str.lower(word))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '//\*[@id="dictionary-entry-1"]/div[2]/div[1]/span[1]/div/span[2]/span').get\_property(  
 'textContent')  
 except:  
 break  
 temp = temp.capitalize()  
 clueDict[word].append(temp)  
  
  
for word in downWords:  
 driver.get('https://www.merriam-webster.com/dictionary/' + str.lower(word))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '//\*[@id="dictionary-entry-1"]/div[2]/div[1]/span[1]/div/span[2]/span').get\_property(  
 'textContent')  
 except:  
 break  
 temp = temp.capitalize()  
 clueDict[word].append(temp)  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
  
from nltk.corpus import wordnet  
  
for word in downWords:  
 syns = wordnet.synsets(word)  
 if syns:  
 clueDict[word].append(syns[0].definition())  
for word in acrossWords:  
 syns = wordnet.synsets(word)  
 if syns:  
 clueDict[word].append(syns[0].definition())  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 driver.get('https://www.dictionary.com/browse/' + str.lower(word))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[1]/div/div/div[2]/div/main/section/section/div[1]/section[2]/div/div').get\_property(  
 'textContent')  
 if len(temp) < 90:  
 temp = temp.capitalize()  
 clueDict[word].append(temp)  
 except:  
 break  
  
for word in downWords:  
 driver.get('https://www.dictionary.com/browse/' + str.lower(word))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[1]/div/div/div[2]/div/main/section/section/div[1]/section[2]/div/div').get\_property(  
 'textContent')  
 if len(temp) < 90:  
 temp = temp.capitalize()  
 clueDict[word].append(temp)  
 except:  
 break  
  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
  
  
for word in acrossWords:  
 driver.get('https://www.urbandictionary.com/define.php?term=' + str.lower(word))  
 for i in range(1, 5):  
 if i != 2:  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[2]/div[4]/div/div/div[1]/div[1]/div[' + str(i) + ']/div[3]').get\_property(  
 'textContent')  
 temp = fix\_data(temp)  
 for val in temp:  
 if len(val) < 90 and len(val) > 5:  
 if(val.rfind('.') != None):  
 n = val.rfind('.')  
 elif(val.rfind(',') != None):  
 n = val.rfind(',')  
 elif(val.rfind(':') != None):  
 n = val.rfind(':')  
 else:  
 n = val.rfind(' ')  
 clueDict[word].append(val[0:n])  
 except:  
 break  
for word in downWords:  
 driver.get('https://www.urbandictionary.com/define.php?term=' + str.lower(word))  
 for i in range(1, 5):  
 if i != 2:  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[2]/div[4]/div/div/div[1]/div[1]/div[' + str(i) + ']/div[3]').get\_property(  
 'textContent')  
 temp = fix\_data(temp)  
 for val in temp:  
 if len(val) < 90 and len(val) > 5:  
 if(val.rfind('.') != None):  
 n = val.rfind('.')  
 elif(val.rfind(',') != None):  
 n = val.rfind(',')  
 elif(val.rfind(':') != None):  
 n = val.rfind(':')  
 else:  
 n = val.rfind(' ')  
 clueDict[word].append(val[0:n])  
 except:  
 break  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
  
badWords2 = ['badword', 'bad1word', 'bad2word', 'bad3word']  
  
  
def profanityFilter2(clue, badWords):  
 bool = False  
 for i in range(len(badWords)):  
 if (badWords[i] in clue):  
 bool = True  
 return bool  
  
def profanityFilter(text):  
 brokenStr1 = text.split()  
 badWordMask = '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'  
 for word in brokenStr1:  
 if word in badWords2:  
 print(word + ' <--Bad word!')  
 text = text.replace(word,badWordMask[:len(word)])  
  
 return text  
  
  
for word in downWords:  
 bool2 = False  
 for i in range(clueDict[word].\_\_len\_\_()):  
 bool = profanityFilter2(clueDict[word][i], badWords2)  
 if bool == True:  
 print('Word: ' + word + ' index: ' + str(i))  
 print('Clue: ' + clueDict[word][i])  
 temp1 = word  
 temp2 = i  
 bool2 = True  
 if bool2 == True:  
 del clueDict[temp1][temp2]  
for word in acrossWords:  
 bool2 = False  
 for i in range(clueDict[word].\_\_len\_\_()):  
 bool = profanityFilter2(clueDict[word][i], badWords2)  
 if bool == True:  
 print('Word: ' + word + ' index: ' + str(i))  
 print('Clue: ' + clueDict[word][i])  
 temp1 = word  
 temp2 = i  
 bool2 = True  
 if bool2 == True:  
 del clueDict[temp1][temp2]  
  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
  
for word in downWords:  
 if len(clueDict[word]) == 0:  
 first = word[0]  
 print(first)  
 rest = word[1:]  
 driver.get('https://www.urbandictionary.com/define.php?term=' + str.lower(rest))  
 for i in range(1, 5):  
 if i != 2:  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[2]/div[4]/div/div/div[1]/div[1]/div[' + str(i) + ']/div[3]').get\_property(  
 'textContent')  
 temp = fix\_data(temp)  
 for val in temp:  
 if len(val) < 90:  
 clueDict[word].append(first + ' + ' + val)  
 except:  
 break  
  
 driver.get('https://www.dictionary.com/browse/' + str.lower(rest))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[1]/div/div/div[2]/div/main/section/section/div[1]/section[2]/div/div').get\_property(  
 'textContent')  
 if len(temp) < 90:  
 clueDict[word].append(first + ' + ' + temp)  
 except:  
 break  
  
for word in acrossWords:  
 if len(clueDict[word]) == 0:  
 first = word[0]  
 print(first)  
 rest = word[1:]  
 driver.get('https://www.urbandictionary.com/define.php?term=' + str.lower(rest))  
 for i in range(1, 5):  
 if i != 2:  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[2]/div[4]/div/div/div[1]/div[1]/div[' + str(i) + ']/div[3]').get\_property(  
 'textContent')  
 temp = fix\_data(temp)  
 for val in temp:  
 if len(val) < 90:  
 clueDict[word].append(first + ' + ' + val)  
 except:  
 break  
 driver.get('https://www.dictionary.com/browse/' + str.lower(rest))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath(  
 '/html/body/div[1]/div/div/div[2]/div/main/section/section/div[1]/section[2]/div/div').get\_property(  
 'textContent')  
 if len(temp) < 90:  
 clueDict[word].append(first + ' + ' + temp)  
 except:  
 break  
  
for word in downWords:  
   
 if len(clueDict[word]) == 0:  
 last = word[-1]  
 print(last)  
 rest = word[:-1]  
 driver.get('https://www.urbandictionary.com/define.php?term=' + str.lower(rest))  
 for i in range(1,5):  
 if i != 2:  
 try:  
 temp = driver.find\_element\_by\_xpath('/html/body/div[2]/div[4]/div/div/div[1]/div[1]/div[' + str(i) + ']/div[3]').get\_property('textContent')  
 temp = fix\_data(temp)  
 for val in temp:  
 if len(val) < 90:  
 n = val.rfind(' ')  
 clueDict[word].append(val[0:n] + ' + ' + last)  
 except:  
 break  
   
 driver.get('https://www.dictionary.com/browse/' + str.lower(rest))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath('/html/body/div[1]/div/div/div[2]/div/main/section/section/div[1]/section[2]/div/div').get\_property('textContent')  
 if len(temp) < 90:  
 n = temp.rfind(' ')  
 clueDict[word].append(val[0:n])  
 clueDict[word].append(temp[0:n] + ' + ' + last)  
 except:  
 break   
   
for word in acrossWords:  
 if len(clueDict[word]) == 0:  
 last = word[-1]  
 print(last)  
 rest = word[:-1]  
 driver.get('https://www.urbandictionary.com/define.php?term=' + str.lower(rest))  
 for i in range(1,5):  
 if i != 2:  
 try:  
 temp = driver.find\_element\_by\_xpath('/html/body/div[2]/div[4]/div/div/div[1]/div[1]/div[' + str(i) + ']/div[3]').get\_property('textContent')  
 temp = fix\_data(temp)  
 for val in temp:  
 if len(val) < 90:  
 clueDict[word].append(val[0:n] + ' + ' + last)  
 except:  
 break  
 driver.get('https://www.dictionary.com/browse/' + str.lower(rest))  
 for i in range(1):  
 try:  
 temp = driver.find\_element\_by\_xpath('/html/body/div[1]/div/div/div[2]/div/main/section/section/div[1]/section[2]/div/div').get\_property('textContent')  
 if len(temp) < 90:  
 clueDict[word].append(temp[0:n] + ' + ' + last)  
 except:  
 break  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
  
for word in downWords:  
 if clueDict[word].\_\_len\_\_() == 0:  
 clueDict[word].append('NOT FOUND')  
  
for word in acrossWords:  
 if clueDict[word].\_\_len\_\_() == 0:  
 clueDict[word].append('NOT FOUND')  
  
for word in downWords:  
 if clueDict[word].\_\_len\_\_() == 1:  
 clueDict[word].append(clueDict[word][0])  
  
for word in acrossWords:  
 if clueDict[word].\_\_len\_\_() == 1:  
 clueDict[word].append(clueDict[word][0])  
  
  
for word in downWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
for word in acrossWords:  
 print('\*\*\*\*\*')  
 print('WORD IS: ' + word)  
 print(clueDict[word])  
 print('\*\*\*\*\*\*\*\*\*\*')  
  
  
import re  
from emoji import UNICODE\_EMOJI  
  
\_nonbmp = re.compile(r'[\U00010000-\U0010FFFF]')  
  
  
def \_surrogatepair(match):  
 char = match.group()  
 assert ord(char) > 0xffff  
 encoded = char.encode('utf-16-le')  
 return (  
 chr(int.from\_bytes(encoded[:2], 'little')) +  
 chr(int.from\_bytes(encoded[2:], 'little')))  
  
  
def with\_surrogates(text):  
 return \_nonbmp.sub(\_surrogatepair, text)  
  
  
def is\_emoji(s):  
 count = 0  
 for emoji in UNICODE\_EMOJI:  
 count += s.count(emoji)  
 if count > 1:  
 return False  
 return True  
  
  
# In[27]:  
  
  
import random  
  
x = 0  
root = Tk()  
canvas = Canvas(root, width=1500, height=750)  
canvas.configure(bg="white")  
  
for i in range(5):  
 for j in range(5):  
 x += 1  
 if reveals[x][0] == "black":  
 canvas.create\_rectangle(100 \* (j + 1), 100 \* (i + 1), 100 + (100 \* (j + 1)), 100 + (100 \* (i + 1)),  
 fill="black")  
 elif reveals[x][2] != "":  
 canvas.create\_rectangle(100 \* (j + 1), 100 \* (i + 1), 100 + (100 \* (j + 1)), 100 + (100 \* (i + 1)),  
 fill="white")  
 canvas.create\_text((100 \* (j + 1)) + 50, (100 \* (i + 1)) + 50, fill="blue", font="Arial 40 bold",  
 text=reveals[x][1][0])  
 canvas.create\_text((100 \* (j + 1)) + 10, (100 \* (i + 1)) + 10, fill="black", font="Arial 15 ",  
 text=reveals[x][2])  
 else:  
 canvas.create\_rectangle(100 \* (j + 1), 100 \* (i + 1), 100 + (100 \* (j + 1)), 100 + (100 \* (i + 1)),  
 fill="white")  
 canvas.create\_text((100 \* (j + 1)) + 50, (100 \* (i + 1)) + 50, fill="blue", font="Arial 40 bold",  
 text=reveals[x][1][0])  
  
canvas.create\_text(465, 615, fill="black", font="Arial 10", text="SWAPLIANO/ " + date)  
canvas.pack(fill=BOTH, expand=1)  
  
root.title("Swapliano")  
  
for i in range(len(clues\_of\_across)):  
 temp = is\_emoji(clues\_of\_across[i][2])  
 if temp == True:  
 new = with\_surrogates(clues\_of\_across[i][2])  
 clues\_of\_across[i][2] = new  
  
for i in range(len(clues\_of\_down)):  
 temp = is\_emoji(clues\_of\_down[i][2])  
 if temp == True:  
 new = with\_surrogates(clues\_of\_down[i][2])  
 clues\_of\_down[i][2] = new  
canvas.create\_text(700, 140, fill="black", font="Arial 20 bold", text="ACROSS")  
cnt=0  
cnt2 = 0  
for i in range(len(clues\_of\_across)):  
 org\_clue = clues\_of\_across[i][2]  
 if len(org\_clue)>120:  
 org\_clue\_div = org\_clue[0:120].rsplit(" ",1)  
 org\_clue\_div[1] = org\_clue[len(org\_clue\_div[0])+1:]  
 print(org\_clue\_div)  
 print(org\_clue\_div[0])  
 print(org\_clue\_div[1])  
 canvas.create\_text(640, 170 + (i\*40 + cnt\*20), fill="gray", font="Arial 13", text=clues\_of\_across[i][1] + " " + org\_clue\_div[0] , anchor="w")  
 cnt+=1  
 canvas.create\_text(640, 170 + (i\*40 + cnt\*20), fill="gray", font="Arial 13", text=" " + org\_clue\_div[1] , anchor="w")  
 else:  
 canvas.create\_text(640, 170 + (i\*40 + cnt\*20), fill="gray", font="Arial 13", text=clues\_of\_across[i][1] + " " + org\_clue , anchor="w")  
  
 gen\_clue = (clueDict[acrossWords[i]][0])  
 gen\_clue = gen\_clue.split('.',1)[0]  
 gen\_clue = gen\_clue.split(':',1)[0]  
  
 if len(gen\_clue)>120:  
 gen\_clue\_div = gen\_clue[0:120].rsplit(" ",1)  
 gen\_clue\_div[1] = gen\_clue[len(gen\_clue\_div[0])+1:]  
 print(gen\_clue\_div)  
 print(gen\_clue\_div[0])  
 print(gen\_clue\_div[1])  
 canvas.create\_text(640, 170 + (i\*40 + cnt\*20+20), fill="purple", font="Arial 13", text=clues\_of\_across[i][1] + " " + gen\_clue\_div[0] , anchor="w")  
 cnt+=1  
 canvas.create\_text(640, 170 + (i\*40 + cnt\*20+20), fill="purple", font="Arial 13", text=" " + gen\_clue\_div[1] , anchor="w")  
 else:  
 canvas.create\_text(640, 170 + (i\*40 + cnt\*20+20), fill="purple", font="Arial 13", text=clues\_of\_across[i][1] + " " + gen\_clue , anchor="w")  
  
canvas.create\_text(680, 200 + (len(clues\_of\_across))\*40, fill="black", font="Arial 20 bold", text="DOWN")  
  
for i in range(len(clues\_of\_down)):  
 org\_clue = clues\_of\_down[i][2]  
 if len(org\_clue)>120:  
 org\_clue\_div = org\_clue[0:120].rsplit(" ",1)  
 org\_clue\_div[1] = org\_clue[len(org\_clue\_div[0])+1:]  
 print(org\_clue\_div)  
 print(org\_clue\_div[0])  
 print(org\_clue\_div[1])  
 canvas.create\_text(640, 50 + (len(clues\_of\_across)+4)\*40 + (i\*40 + cnt\*20), fill="gray", font="Arial 13", text=clues\_of\_down[i][1] + " " + org\_clue\_div[0] , anchor="w")  
 cnt+=1  
 canvas.create\_text(640, 60 + (len(clues\_of\_across)+4)\*40 + (i\*40 + cnt\*20), fill="gray", font="Arial 13", text=" " + org\_clue\_div[1] , anchor="w")  
 else:  
 canvas.create\_text(640, 60 + (len(clues\_of\_across)+4)\*40 + (i\*40 + cnt\*20), fill="gray", font="Arial 13", text=clues\_of\_down[i][1] + " " + org\_clue , anchor="w")  
  
 gen\_clue = (clueDict[downWords[i]][0])  
 gen\_clue = gen\_clue.split('.',1)[0]  
 gen\_clue = gen\_clue.split(':',1)[0]  
 if len(gen\_clue)>120:  
 gen\_clue\_div = gen\_clue[0:120].rsplit(" ",1)  
 gen\_clue\_div[1] = gen\_clue[len(gen\_clue\_div[0])+1:]  
 print(gen\_clue\_div)  
 print(gen\_clue\_div[0])  
 print(gen\_clue\_div[1])  
 canvas.create\_text(640, 60 + (len(clues\_of\_across)+4)\*40 + (i\*40 + cnt\*20+20), fill="purple", font="Arial 13", text=clues\_of\_down[i][1] + " " + gen\_clue\_div[0] , anchor="w")  
 cnt+=1  
 canvas.create\_text(640, 60 + (len(clues\_of\_across)+4)\*40 + (i\*40 + cnt\*20+20), fill="purple", font="Arial 13", text=" " + gen\_clue\_div[1] , anchor="w")  
 else:  
 canvas.create\_text(640, 60 + (len(clues\_of\_across)+4)\*40 + (i\*40 + cnt\*20+20), fill="purple", font="Arial 13", text=clues\_of\_down[i][1] + " " + gen\_clue , anchor="w")  
  
  
root.mainloop()  
'''  
canvas.create\_text(640, 170 + (i \* 40), fill="gray", font="Arial 15",  
 text=clues\_of\_across[i][1] + " " + clues\_of\_across[i][2], anchor="w")  
 canvas.create\_text(640, 170 + (i \* 40) + 20, fill="purple", font="Arial 15",  
 text=clues\_of\_across[i][1] + " " + (clueDict[acrossWords[i]][0]), anchor="w")  
  
canvas.create\_text(680, 170 + (len(clues\_of\_across) + 2) \* 40, fill="black", font="Arial 20 bold", text="DOWN")  
for i in range(len(clues\_of\_down)):  
 canvas.create\_text(640, 130 + (len(clues\_of\_across) + 4) \* 40 + (i \* 40), fill="gray", font="Arial 15",  
 text=clues\_of\_down[i][1] + " " + clues\_of\_down[i][2], anchor="w")  
 canvas.create\_text(640, 130 + (len(clues\_of\_across) + 4) \* 40 + (i \* 40) + 20, fill="purple", font="Arial 15",  
 text=clues\_of\_down[i][1] + " " + (clueDict[downWords[i]][0]), anchor="w")  
  
root.mainloop()  
'''  
# SIMIARITY Ã–LÃ‡EÄÄ°  
from difflib import SequenceMatcher  
  
  
def similar(a, b):  
 return SequenceMatcher(None, a, b).ratio()  
  
  
# In[ ]: