Bilkent University

Department of Computer Engineering

CS 461 – Artificial Intelligence

"Term Project Report / WORD++"

Hamza PEHLİVAN

Ahmet Canberk BAYKAL

Meryem Binnaz EFE

Zeynep Hande ARPAKUŞ

27.12.2019

CONTENT

1. Introduction 2

2. Architecture 3

2.1. WordNet 3

2.1.1. Finding Causes 4

2.1.2. Finding Entailments 4

2.1.3. Finding Substance Holonyms 5

2.1.4. Finding Substance Meronyms 5

2.1.5. Finding Part Holonyms 6

2.1.6. Finding Part Meronyms 6

2.1.7. Finding Attributes 6

2.1.8. Finding Also Sees 7

2.1.9. Finding Antonyms 7

2.1.10. Finding Synonyms 7

2.1.11. Finding Direct Hyponyms 8

2.1.12. Finding Direct Hypernyms 8

2.1.13. Finding Coordinate Terms 8

2.1.14. Finding Senses 9

2.2. Stanford CoreNLP API and Google Search 9

2.2.1. Lemmatization 9

2.2.1. Part of Speech 9

2.2.2. Named Entity Recognition and Google Search 10

2.3. Paraphrasing Tool 10

3. Examples 10

4. References 21

5. Appendix 23

1. Introduction

In this project, we'll write a program called WORD++. We'll use the Java programming language. Our program will input the clues of a 5x5 New York Times mini puzzle by Joel Fagliano and its official solution. In this project, we will change nothing but the clues. The original puzzle and its solution will be kept the same. The project will have a 5x5 puzzle area and original clues. The program will provide to see the official solution as on the original website. However, the main point is to change original clues with different clues that refer to the same solution. In AI part of the project, through natural language processing and some other techniques, we will find different clues without changing the original solutions.

During the project, we inspired techniques used in the article written by Michael Littman, Greg Keim, and Noam Shazeer.[1]

2. Architecture

To generate new clues for the crossword, for each answer, we implemented the following strategy.

Firstly, we call WordNets' methods to find a sensible explanation. In this part, fourteen different methods are used and the order in Chapter 2.1. is followed.

If the WordNet methods cannot find any sensible result, the second step is to use Stanford CoreNLP. In Stanford CoreNLP, it is searched that whether the answer is a special name. If it is a special name, the answer is searched in Google.

The last step is to use paraphrasing technique. If the answer is found in neither WordNet nor Google search, the clue of the answer is written in the paraphrasing tool.

2.1. WordNet

WordNet is an English database where the words are associated together according to their semantic relationships. WordNet associates not only the meaning

[1] "A probabilistic approach to solving crossword puzzles ...." 14 Jan. 2010, https://www.sciencedirect.com/science/article/pii/S000437020100114X. Accessed: 26 Dec. 2019.

of the words but also links their different senses. It is publicly available for downloading and using in specific applications such as natural language processing projects.[2]

In WordNet, synonyms are linked together to form synsets which are unordered sets. There are brief definitions of each of these synsets along with a couple of example sentences to demonstrate how these words are used. Then, each of these synsets are also linked to each other according to their relationships and closeness.

There are a lot of word relationships in WordNet. We have implemented functions to obtain the words that are related to our answers with these relationships. These functions are explained in the following subsections:

2.1.1. Finding Causes

One of the important methods of WordNet is *getCause()*. This method works for verbs. When it is run for a verb it returns that verbs' results.
Example:

- A result of the verb "ignite" is "burn".
- A result of the verb "hurt" is "smart".

To generate a new clue by using this method, we add "Result of ..." to the beginning of the word which is returned by *getCause()* method.

2.1.2. Finding Entailments

getEntailment() method is used to get entailment of a word. Entailment means implication. It returns what the given word implies.
Example:

- Entailment of "look" is "see".
- Entailment of "buy" is "pay".

To generate a new clue by using this method, we add "It implies ..." to the beginning of the word which is returned by *getEntailment()* method.

[2] "Introduction to WordNet: An On-line Lexical Database - Index of." https://wordnetcode.princeton.edu/5papers.pdf. Accessed: 27 Dec. 2019.

2.1.3. Finding Substance Holonyms

"Holonym is a term that denotes a whole, a part of which is denoted by a second term."[3] *In the WordNet library, there are several methods to find such relationships between two words. In this project, two of them are used.*

Substance holonym returns the things that are made of the substance this synset represents.[4] To find substance holonym of a word, in WordNet library, there is *getSubstanceHolonym()* method. As a parameter, it takes a word in synset format and it returns substance holonym of this word.

Examples:

- A substance holonym of paper is page.
- Substance holonyms of copper is brass, bronze, bornite etc.
- A substance holonym of hydrogen is carbon.
- Substance holonyms of carbon are coal, petroleum, crude oil etc.

To generate a new clue by using this method, we add "A substance of ..." to the beginning of the word which is returned by *getSubstanceHolonym()* method.

2.1.4. Finding Substance Meronyms

Meronymy is a part to whole relationship. It is the exact opposite relationship of holonymy. Therefore, meronym is a word that is a part or a member of a whole. A word is a substance meronym of another word if it represents something that is made from the other word. To obtain a substance meronym of a word, the function *getSubstanceMeronym()* is provided by WordNet library. The input parameters to this function is a word in synset format.

Examples:

- A substance meronym of rubber is wheel.
- A substance meronym of water is hydrogen.

To generate clues from substance meronyms, we create a sentence that has the structure: "It consists of …" where the space is the word that our function returns.

---

[3] *"holonym - Wiktionary." https://en.wiktionary.org/wiki/holonym. Accessed: 26 Dec. 2019.*

[4] "NounSynset." https://mind.cs.byu.edu/projects/DARCI/source_code/source_code/LanguageAnalysis/doc/edu/smu/tspell/wordnet/NounSynset.html. Accessed: 26 Dec. 2019.

2.1.5. Finding Part Holonyms

Part holonym returns the holonyms (whole that includes this part) of this type. In other words, part holonym represents the part-and-whole relationship between two words.
To find the part holonym of a word, in WordNet library, there is *getPartHolonym()* method. As a parameter, it takes a word in synset format and it returns part holonym of this word.
Examples:

- A part holonym of fuselage is airplane.
- A part holonym of arm is human body.

To generate a new clue by using this method, we add “A part of ...” to the beginning of the word which is returned by *getPartHolonym()* method.

2.1.6. Finding Part Meronyms

Part meronymy is the relationship of being a part of a whole. Hence, a part meronym of a word is basically a part of that word. In WordNet library, the function *getPartMeronym()* returns the part meronyms of a word.
Examples:

- A part meronym of a car is tire
- A part meronym of body is arm

To generate clues from part meronyms, we create a sentence that has the structure: “It has parts like .... and … “ where the spaces are the words returned by our function.

2.1.7. Finding Attributes

Another important method in WordNet is *getAttributes()* method. “This method returns adjectives that describes states associated with this noun synset's concept.”
Examples:

- Attributes of “seriousness” are “serious” and “frivolous”.

To generate a new clue by using this method, we add “It can be ...” to the beginning of the word which is returned by this method.

2.1.8. Finding Also Sees

getAlsoSees() is another method of WordNet. This method is used for getting the words which are related to the given word. This method "returns alternate/equivalent version of a sense"[5].

Examples:

- For the word "more", the method returns "much, many, more than" words.

Because this method also returns the word itself, when we choose a clue from the array which is returned by this method, we check it and eliminate the same word with we gave.

2.1.9. Finding Antonyms

While defining a word, its antonyms can be useful. For this purpose, there is *getAntonyms()* method which returns the antonyms (words with the opposite meaning), if any, associated with a word form in this synset.
Examples:

- Antonym of "left" is "right".

To generate a new clue by using this method, we add "Antonym of ..." to the beginning of the word which is returned by this method. Because antonyms are common and we wanted to show different features of the software, we allowed the software to produce maximum two antonym relationships.

2.1.10. Finding Synonyms

In puzzle clues, synonym words are used commonly. To get these words, we used *getSynonym()* method of WordNet. Synonym means "a word or phrase that has the same or nearly the same meaning as another word or phrase in the same language"[6].
Example:

---

[5] "WordNet framework improvements for NLP: Defining abstraction and scalability layers" http://www.inesc-id.pt/publications/8113/pdf. Accessed: 26 Dec 2019
[6] "SYNONYM" https://dictionary.cambridge.org/tr/s%C3%B6zl%C3%BCk/ingilizce/synonym. Accessed: 26 Dec. 2019

- Synonym of “small” is “tiny”
- Synonym of “smart” is “clever”

To generate a new clue by using this method, we add “Synonym of ...” to the beginning of the word which is returned by *getSynonym().* Synoyms can be created by software is restricted because synonyms are quite common.

2.1.11. Finding Direct Hyponyms

Hyponym is a word whose meaning is included in the meaning of another word.[7] For this purpose, there is *getDirectHyponyms()* method which gets the immediate parents of synset.

Examples:

- Direct hyponym of “fox” is “canine”

To generate a new clue by using this method, we add “A kind of ….” to the beginning of the word. Because almost every word has a parent, maximum number of using hyponym is one per puzzle.

2.1.12. Finding Direct Hypernyms

Hypernym and hyponyms are antonyms. Hypernym is a word whose meaning includes a group of other word.[8] It means children of a word. For this purpose, there is *getDirectHypernyms()* method which gets the immediate children of synset.

Examples:

- Direct hypernym of “chat” is “small talk”

To generate a new clue by using this method, we add “... is a kind of it.” to the end of the word which is returned by this method. Like hyponyms, almost every word has a child. Therefore, maximum number of using hypernym relationship is one.

2.1.13. Finding Coordinate Terms

The words with the same hypernym are called coordinate terms. *getCoordinateTerm()* method is used to find the coordinate terms of the given word.

Example:

---

[7] "HYPONYM" https://dictionary.cambridge.org/tr/s%C3%B6zl%C3%BCk/ingilizce/hyponym. Accessed: 26 Dec. 2019.

[8] "HYPERNYM" https://dictionary.cambridge.org/tr/s%C3%B6zl%C3%BCk/ingilizce/hypernym. Accessed: 26 Dec. 2019.

- Coordinate term of “tree” is “melilotus”.
- Coordinate term of “book” is “volume”.

2.1.14. Finding Senses

The most general method is *getSenses()*. Therefore, we use this method lastly. It finds many words related to the parameter word and it returns these words according to the order of relevance.
Example:

- Senses of “idea” are “thought” and “mind”.

Because the most relevant words can contain the actual word, only one relevant word among them which does not contain the parameter is chosen as a new clue.

2.2. Stanford CoreNLP API and Google Search

Stanford CoreNLP API is a natural language processing tool[9] which helps finding base form and part of speech of a word. Also, it is used to find special names and special place names like cities, countries and locations.

2.2.1. Lemmatization

We used lemmatization to avoid showing actual answer within the clue. For instance, if the answer is “eggs”, WordNet process can produce the sense “egg” which is not desirable for a puzzle. Because the base form of “eggs” is “egg”, produced clues including “egg” are eliminated.

2.2.1. Part of Speech

We use part of speech feature to be consistent with answer. For example, if the answer is “cells”, WordNet process can produce “A part of organism”. The problem is generated clue does not include anything about being plural. After detecting if given answer is plural, we added “(plural)” to end of the generated clue. So, the generated clue becomes “A part of organism (plural)”.

[9] "Proceedings of 52nd Annual Meeting of the Association for ...." https://www.aclweb.org/anthology/volumes/P14-5/. Accessed: 26 Dec. 2019.

2.2.2. Named Entity Recognition and Google Search

Famous places and people are indispensable for word puzzles. However, it is hard to find them in WordNet. For this problem we detect if the given clue is a famous person or place using Stanford CoreNLP API. After detecting it, the software performs a Google search. For example, if the answer is “Yoda”, Stanford CoreNLP API decides that “Yoda” is a special name and Google search starts. After searching in Google, the clue “____ Star Wars character” is produced.

2.3. Paraphrasing Tool

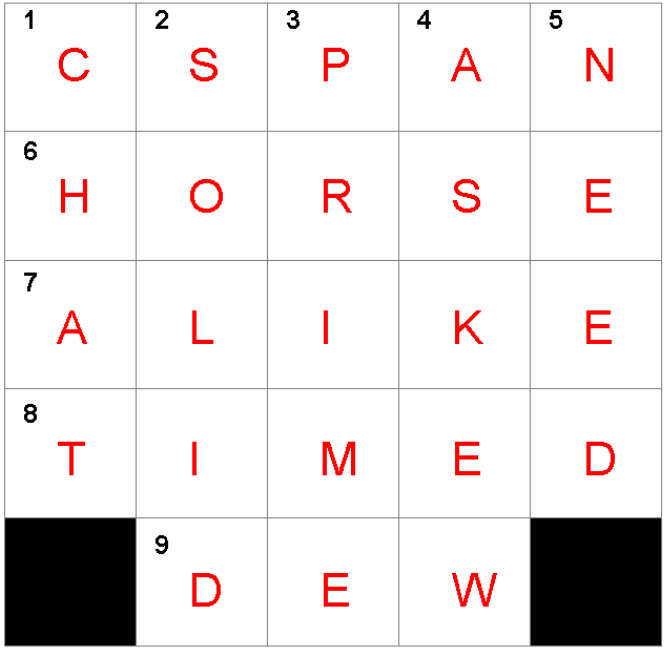
If the none of the above methods works for an answer, that is, the answer is tricky or a famous place-person couldn’t find by Google search, the software uses an online paraphrasing tool to rewrite given clue. For example, if the answer is “psst” and original clue is “Hey, you! Over here!”, online paraphrasing tool converts it to “Hi there, you! Over right here!”[10]

3. Examples

There are ten examples in this section. Each of them is put to a different page so that reader can follow them easily.

[10] "Paraphrasing tool - Code Beautify." https://codebeautify.org/paraphrasing-tool. Accessed: 26 Dec. 2019.

WORD++



Tuesday November 19, 2019 WORD++

ACROSS

1. Airer of the impeachment inquiry
6. Chess knight, essentially
7. Cut from the same cloth
8. Like Sporcle quizzes
9. Morning moisture

DOWN

1. Convo
2. Not hollow
3. Like 2003, 2011 and 2017
4. Slightly off-kilter
5. Have to have

NEW ACROSS

1. Airer of the impeachment inquiry
6. It has parts like encolure and withers and gaskin.
7. Antonym of dissimilar
8. Synoym of regular
9. A kind of condensation

NEW DOWN

1. Small talk is a kind of it
2. Antonym of liquid
3. Synoym of first
4. Awry
5. Demand

WORD++

Thursday November 21, 2019 WORD++

ACROSS

1. Big success
4. Philosopher who believed that the physical world is only a shadow of true reality
6. Poet with an epic legacy
7. Idiot
8. Pig's home

DOWN

1. Muscle on the back of your leg, slangily
2. Big-ticket ___ (expensive purchase)
3. British conservative
4. Degrees held by less than 2% of Americans
5. No-good guy

NEW ACROSS

1. Antonym of miss
4. A kind of philosopher
6. Solo blast is a kind of it
7. Synoym of artificial
8. Hordeolum

NEW DOWN

1. Synoym of dramatic
2. A part of list
3. An American who favored the British side during the American Revolution
4. Ph.D. (plural)
5. Clod

WORD++

1 F	2 R	3 O	4 S	5 T
6 L	A	T	H	E
7 U	S	H	E	R
8 S	T	E	E	R
9 H	A	R	R	Y

Friday November 22, 2019 WORD++

ACROSS

1. "Stopping by Woods on a Snowy Evening" poet
6. Wood-crafting tool
7. Wedding seater
8. Member of a cattle herd
9. Styles of music

DOWN

1. Five cards of the same suit
2. Dreadlocks wearer, informally
3. "On the ___ hand ..."
4. Dangerously steep
5. Fabric used for bath towels

NEW ACROSS

1. A kind of ice
6. It has parts like carriage and chuck and handwheel.
7. Marshal is a kind of it
8. Tip
9. Harass

NEW DOWN

1. Result of run, flow, feed and course.
2. Follower of Rastafarianism
3. Antonym of same
4. Synoym of complete
5. English actress (1847-1928)

WORD++

■	■	1 F	2 O	3 X
4 S	5 P	E	R	M
6 K	O	R	E	A
7 E	U	R	O	S
8 W	R	Y	■	■

Sunday November 24, 2019 WORD++

ACROSS

1. Animal whose babies are known as "kits"
4. What fuses with an ovum
6. Peninsula divided at the 38th parallel
7. Currency symbols that can be typed using Ctrl+Alt+E
8. Like a knowing smirk

DOWN

1. One way to get to Staten Island
2. Best-selling cookie brand in the U.S.
3. Present day, for short
4. Distort
5. Rain heavily

NEW ACROSS

1. A kind of canine
4. A part of male reproductive system
6. A part of Asia
7. The basic monetary unit of most members of the European Union (introduced in 1999)
8. Synoym of humorous

NEW DOWN

1. A boat that transports people or vehicles across a body of water and operates on a regular schedule
2. Chocolate cookie with white cream filling
3. Christmas (plural)
4. Antonym of align
5. Result of run, flow, feed and course.

WORD++

1 P	2 S	3 S	4 T	
5 T	I	T	H	6 E
7 S	N	A	R	L
8 D	A	N	E	S
	9 I	D	E	A

Tuesday November 26, 2019 WORD++

ACROSS

1. "Hey, you! Over here!"
5. Give 10% to the church
7. Hair tangle
8. "Great" dogs
9. Inkling

DOWN

1. Condition for a returning combat vet
2. Mount where Moses received the Ten Commandments
3. Roadside produce seller
4. Number of bones in the human ear
6. Queen in "Frozen 2"

NEW ACROSS

1. "hi there, you! Over right here!"
5. A kind of levy
7. Antonym of disentangle
8. Zealander is a kind of it
9. Thought

NEW DOWN

1. It has parts like survivor guilt.
2. A part of Arabian Desert
3. A part of ballpark
4. Synoym of cardinal
6. _____
Fictional character

WORD++

■	■	1 C	2 E	3 O
4 T	5 H	I	G	H
6 W	I	N	G	S
7 A	N	D	S	O
8 S	T	Y	■	■

Thursday November 28, 2019 WORD++

ACROSS

1. Top of a corporate ladder
4. Certain piece of turkey
6. Some pieces of turkey
7. "Therefore ..."
8. Totally messy room

DOWN

1. ___ Lou Who, character in "How the Grinch Stole Christmas"
2. Food sold in sixes and dozens
3. Very
4. "___ the night before Christmas ..."
5. Crossword puzzle clue, for example

NEW ACROSS

1. A kind of corporate executive
4. A part of leg
6. Buzz is a kind of it (plural)
7. "consequently ..."
8. Hordeolum

NEW DOWN

1. _____ Crawford
American model
2. It consists of protein. (plural)
3. Very
4. The World Academy of Sciences
5. Intimation

WORD++

1 A	2 D	3 A	4 M	5 S
6 M	E	T	A	L
7 A	L	L	M	E
8 S	T	A	M	P
9 H	A	S	A	T

Sunday December 1, 2019 WORD++

ACROSS

1. America's first vice president
6. Almost any element that ends in -ium
7. "I can take care of that myself"
8. Purchase that might be Forever
9. Vigorously attacks

DOWN

1. Justin ___, first sitting Republican to support Trump's impeachment
2. Airline with a Sky magazine
3. Book of maps
4. "___ Mia!" (Abba musical)
5. Took a nap

NEW ACROSS

1. A part of Cascades
6. Synoym of aluminiferous
7. "i can cope with that myself"
8. A part of battery
9. Vigorously assaults

NEW DOWN

1. Justin _____
U.S. Representative
2. A kind of geological formation
3. Book of maps
4. Udder is a kind of it
5. Antonym of wake

WORD++

■	1 E	2 W	3 W	■
4 A	L	O	H	5 A
6 B	L	O	O	D
7 C	E	L	L	S
■	8 N	F	L	■

Wednesday December 4, 2019 WORD++

ACROSS

1. "Gross!"
4. Honolulu hello
6. With 7-Across, your body has produced millions of them while you've been reading this clue
7. See 6-Across
8. Org. with three Thanksgiving Day games

DOWN

1. Talk show host who voices Pixar's Dory
2. Virginia who wrote "Mrs. Dalloway"
3. "___ Pull Santa's Sleigh Tonight?" (children's book)
4. Epitome of simplicity
5. What're found on the sides of many buses

NEW ACROSS

1. "Gross!"
4. A kind of acknowledgment
6. It consists of serum.
7. A part of organism (plural)
8. Org. With 3 Thanksgiving Day games

NEW DOWN

1. _____ DeGeneres
American comedian
2. English author whose work used such techniques as stream of consciousness and the interior monologue
3. "___ Pull Santa's Sleigh this night?" (kid's ebook)
4. Rudiments
5. Direct mail is a kind of it

WORD++

1 A	2 H	3 A		
4 B	A	B	5 Y	
6 S	N	O	O	7 P
	8 D	U	D	E
		9 T	A	G

Sunday December 8, 2019 WORD++

ACROSS

1. "Now I get it!"
4. With 5-Down, much-buzzed-about character on Disney Plus's "The Mandalorian"
6. Ask overly invasive questions
8. Bro
9. Graffiti signature

DOWN

1. Muscles exercised when doing a Russian twist
2. Round of applause
3. Give or take
5. See 4-Across
7. Coat rack part

NEW ACROSS

1. "Now i am getting it!"
4. A kind of child
6. Eavesdropper is a kind of it
8. Fellow
9. Ticket

NEW DOWN

1. Acrylonitrile-butadiene-styrene
2. A part of arm
3. Synoym of active
5. _____ Star Wars character
7. A part of stringed instrument

WORD++

1 R	2 B	3 G		
4 A	L	E	5 C	6 K
7 M	U	C	H	O
	8 S	K	I	N
	9 H	O	N	G

Friday November 29, 2019 WORD++

ACROSS

1. "Notorious" Supreme Court justice
4. Smart ___
7. A lot, in Mexico
8. Inedible part of an onion
9. With 6-Down, site of high-profile democracy protests

DOWN

1. Ewe's mate
2. Get red in the face
3. Creature with sticky toe pads
5. Body part that rests on a violin
6. See 9-Across

NEW ACROSS

1. "notorious" supreme courtroom justice
4. Clever ___
7. Lots, in Mexico
8. It consists of melanin.
9. _____ Kong
Chinese special administrative region

NEW DOWN

1. It has parts like buffer and buffer storage and buffer store.
2. A kind of good health
3. Ptychozoon homalocephalum is a kind of it
5. A part of face
6. See nine-across

4. References

Littman, Michael L., et al. “A Probabilistic Approach to Solving Crossword Puzzles.” *Artificial Intelligence*, Elsevier, 28 Dec. 2001, https://www.sciencedirect.com/science/article/pii/S000437020100114X.

Miller, George A., et al. *Introduction to WordNet: An On-Line Lexical Database*. Aug. 1993, https://wordnetcode.princeton.edu/5papers.pdf.

“Holonym.” *Wiktionary*, https://en.wiktionary.org/wiki/holonym.

NounSynset, https://mind.cs.byu.edu/projects/DARCI/source_code/source_code/LanguageAnalysis/doc/edu/smu/tspell/wordnet/NounSynset.html.

Fialho, Pedro, et al. *WordNet framework improvements for NLP: Defining abstraction and scalability layers.* 06 Sep. 2011, http://www.inesc-id.pt/publications/8113/pdf

“SYNONYM.” *SYNONYM| Cambridge İngilizce Sözlüğü'ndeki Anlamı*, https://dictionary.cambridge.org/tr/s%C3%B6zl%C3%BCk/ingilizce/synonym.

“HYPONYM.” *HYPONYM | Cambridge İngilizce Sözlüğü'ndeki Anlamı*, https://dictionary.cambridge.org/tr/sözlük/ingilizce/hyponym.

"HYPERNYM." *HYPERNYM | Cambridge İngilizce Sözlüğü'ndeki Anlamı*, https://dictionary.cambridge.org/tr/s%C3%B6zl%C3%BCk/ingilizce/hypernym.

Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations - ACL Anthology, https://www.aclweb.org/anthology/volumes/P14-5/.

Jimmy. "Paraphrasing Tool, Best Tool to Rewrite the Content." *Paraphrasing Tool, Best Tool to Rewrite the Content*, https://codebeautify.org/paraphrasing-tool.

5. Appendix

```
// ACCESSING NEWYORK TIMES AND GETTING DATA
package accessToNewYorkTimes;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;

import java.util.List;

public class Informations {

    final int MAX_SIZE = 5;
    private boolean [][] blackCells;
    private String [] acrossClues;
    private String [] downClues;
    private int [][] littleNumbers;
    private String [][] solution;
    private String date;
    private int [] acrossClueNumbers;
    private int [] downClueNumbers;

    public Informations() throws InterruptedException {

        blackCells = new boolean [MAX_SIZE][MAX_SIZE];
        acrossClues = new String [ MAX_SIZE];
        downClues = new String [MAX_SIZE];
        littleNumbers = new int [ MAX_SIZE][ MAX_SIZE];
        solution = new String [MAX_SIZE][MAX_SIZE];
        date = "";
        acrossClueNumbers = new int [MAX_SIZE];
        downClueNumbers = new int [MAX_SIZE];
        getInformationsfromWeb();
    }

    public boolean [][] getBlackCells() {
        return blackCells;
    }
    public String printBlackCells() {
        String data = "";
        for ( int i = 0; i < MAX_SIZE; i++) {
            for ( int j = 0; j < MAX_SIZE; j++) {
                data = data + blackCells[i][j] + " ";
            }
            data = data + "\n";
        }
        return data;
    }

    public String [] getAcrossClues () {
        return acrossClues;
    }
    public String printAcrossClues () {
        String data = "";
        for ( int i = 0; i < MAX_SIZE; i++) {
```

```
            data = data + acrossClues[i] + "\n";
        }
        return data;
    }
    public String [] getDownClues() {
        return downClues;
    }
    public String printDownClues () {
        String data = "";
        for ( int i = 0; i < MAX_SIZE; i++) {
            data = data + downClues[i] + "\n";
        }
        return data;
    }

    public int [][] getLittleNumbers() {
        return littleNumbers;
    }
    public String printlittleNumbers() {
        String data = "";
        for ( int i = 0; i < MAX_SIZE; i++) {
            for ( int j = 0; j < MAX_SIZE; j++) {
                data = data + littleNumbers[i][j] + " ";
            }
            data = data + "\n";
        }
        return data;
    }

    public String [][] getSolution() {
        return solution;
    }
    public String printSolution() {
        String data = "";
        for ( int i = 0; i < MAX_SIZE; i++) {
            for ( int j = 0; j < MAX_SIZE; j++) {
                data = data + solution[i][j] + " ";
            }
            data = data + "\n";
        }
        return data;
    }

    public String getDate() {
        return date;
    }
    public int [] getAcrossClueNumbers () {
        return acrossClueNumbers;
    }
    public String printAcrossClueNumbers () {
        String data = "";
        for ( int i = 0; i < MAX_SIZE; i++) {
            data = data + acrossClueNumbers[i] + " ";
        }
        return data;
    }

    public int [] getDownClueNumbers () {
        return downClueNumbers;
```

```
    }
    public String printDownClueNumber () {
        String data = "";
        for ( int i = 0; i < MAX_SIZE; i++) {
            data = data + downClueNumbers[i] + " ";
        }
        return data;
    }

    private void getInformationsfromWeb() throws InterruptedException {
        System.setProperty("webdriver.chrome.driver",
"C:\\Users\\pehli\\Desktop\\chromedriver.exe");

        WebDriver driver = new ChromeDriver();

        driver.get("https://www.nytimes.com/crosswords/game/mini");  // Go to URL

        driver.manage().window().maximize();  // Maximize Screen

        WebElement element;

        element = driver.findElement(By.className("buttons-modalButton--1REsR"));
// Click on OK 'Ready to Start?'
        element.click();

        Actions act = new Actions ( driver);

        element = driver.findElement(By.xpath("//button[text()='reveal']"));
// Click on Reveal
        element.click();

        /// Wait time
        Thread.sleep(1000);

        List<WebElement> elements =
driver.findElements(By.xpath("//a[text()='Puzzle']")); // Click on Puzzle
By.xpath("//a[text()='Puzzle']")
        WebElement curr = elements.get(1);
        act.moveToElement(curr).click().build().perform();


        /// Wait time
        Thread.sleep(1000);

        elements = driver.findElements(By.className("buttons-modalButton--
1REsR"));   // Click on Reveal 'Are you sure to reveal?'
        curr = elements.get(1);
        curr.click();

        /// Wait time
        Thread.sleep(1000);

        act.moveByOffset(0, 0).click().build().perform();     // Click on Exit

        /// Wait time
        Thread.sleep(1000);

        //Take elements
```

```
        extractDate( driver);
        extractClues(driver);
        extractPuzzleContent(driver);

        driver.close();

        driver.quit();
    }

    private  void extractClues (WebDriver driver) {

        List <WebElement> elements = driver.findElements(By.className("Clue-text--
3lZl7"));
        List <WebElement> numbers = driver.findElements(By.className("Clue-label--
2IdMY"));

        for (int i = 0; i < MAX_SIZE; i++) {
                acrossClues[i] = elements.get(i).getText();
                acrossClueNumbers[i] = Integer.parseInt(numbers.get(i).getText());

        }
        for ( int i = MAX_SIZE; i < 2*MAX_SIZE; i++) {
            downClues [ i-MAX_SIZE] = elements.get(i).getText();
            downClueNumbers [i-MAX_SIZE] =
Integer.parseInt(numbers.get(i).getText());
        }
    }

/* private void extractSolution(WebDriver driver) {
      List <WebElement> elements = driver.findElements(By.xpath("//"));
   }*/

    private void extractPuzzleContent( WebDriver driver) {

        List <WebElement> elements = driver.findElements(By.tagName("g"));
        for (int i = 0; i < 4; i++)
            elements.remove(0);
        elements.remove(25);



        int i = 0, j = 0;
        int count = 0;
        for ( WebElement e: elements) {
            List <WebElement>  children = e.findElements(By.tagName("rect"));
            List <WebElement>  siblings = e.findElements(By.tagName("text"));

            for (WebElement child: children) {
                if ( child.getAttribute("class").equals("Cell-block--1oNaD Cell-
nested--x0A1y") )
                    blackCells [i][j++] = true;
                else {
                    if (siblings.size() != 0) {
                        if ( siblings.size() == 2) {
                            solution[i][j] = siblings.get(0).getText();
                        }
                        else {
                            littleNumbers [i][j] =
Integer.parseInt(siblings.get(0).getText());
```

```
                            solution [i][j] = siblings.get(2).getText();
                        }
                    }

                    blackCells [i][j++] = false;
                }
            }
            if ( j == MAX_SIZE) {
                j = 0;
                i++;
            }
        }
    }

    private void extractDate ( WebDriver driver) {
        WebElement element = driver.findElement(By.className("PuzzleDetails-date--
1HNzj"));
        date = element.getText();
    }
}
```

```
// USING PREVIOUSLY GENERATED DATA
package accessToNewYorkTimes;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class PuzzleDatabase {
    int MAX_SIZE = 5;
    public boolean [][] blackCells;
    public String [] acrossClues;
    public String [] downClues;
    public int [][] littleNumbers;
    public String [][] solution;
    public String date;
    public int [] acrossClueNumbers;
    public int [] downClueNumbers;
    public PuzzleDatabase () {
        blackCells = new boolean [MAX_SIZE][MAX_SIZE];
        acrossClues = new String [ MAX_SIZE];
        downClues = new String [MAX_SIZE];
        littleNumbers = new int [ MAX_SIZE][ MAX_SIZE];
        solution = new String [MAX_SIZE][MAX_SIZE];
        date = "";
        acrossClueNumbers = new int [MAX_SIZE];
        downClueNumbers = new int [MAX_SIZE];
        File file = new
File("C:\\Users\\pehli\\Desktop\\Java\\deneme\\Puzzle19.txt");
        int count = 0;
        try {
            Scanner scanner = new Scanner(file);
            //scanner.useDelimiter("\n");
            scanner.useDelimiter( " ");
            while ( scanner.hasNextInt() && count < 5) {
```

```
    acrossClueNumbers[count]  = scanner.nextInt();
    count++;
}
count = 0;
scanner.useDelimiter("\n");
scanner.next();

while ( scanner.hasNext() && count < 5) {
   acrossClues [count] = scanner.next();
   count++;
}
count = 0;
int countTwo = 0;
scanner.next();
scanner.useDelimiter( " ");
while (scanner.hasNext() && countTwo < 5 ) {
    String s = scanner.next();
    if ( count == 0)
        s = s.substring( 1);
  blackCells [countTwo ][ count] =  s.equals( "true");
    count++;
  if ( count == 5) {
      count = 0;
      countTwo++;
  }
}
scanner.useDelimiter( "\n");
scanner.next();
scanner.next();
scanner.useDelimiter( " ");
count = 0;
countTwo = 0;
while ( scanner.hasNext() && count < 5) {
    String s = scanner.next();
    if ( count == 0)
        s = s.substring(1);

    downClueNumbers [ count] = Integer.parseInt(s);
    count++;
}
count = 0;
scanner.nextLine();
scanner.useDelimiter( "\n");
while ( scanner.hasNext() && count < 5) {
    downClues [count] = scanner.next();
    count++;
}
count = 0;
scanner.next();
scanner.useDelimiter( " ");

while ( scanner.hasNext() && countTwo < 5 ) {
    String s = scanner.next();
    if ( count == 0 )
        s = s.substring( 1);
    littleNumbers [countTwo ][ count] = Integer.parseInt( s);
    count++;
    if ( count == 5) {
        count = 0;
```

```
                countTwo++;
            }
        }
        count = 0;
        countTwo = 0;
        scanner.nextLine();
        scanner.nextLine();
        while ( scanner.hasNext() && countTwo < 5) {
            String s = scanner.next();
            if ( count == 0  && countTwo != 0)
                s = s.substring( 1);
            solution [countTwo ][ count] = s.equals( "null") ? null : s;
            count++;
            if ( count == 5) {
                count = 0;
                countTwo++;
            }
        }
        scanner.useDelimiter( "\n");
        while ( scanner.hasNext() )
            date = scanner.next();
        scanner.close();
       /* for ( int i = 0; i < 5; i++)
            System.out.println( acrossClueNumbers[i]);*/
       for ( int i = 0; i < 5 ; i++) {
            for ( int j = 0; j < 5;j++)
                System.out.print(blackCells [i][j] + " ");
           System.out.println();
        }

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

public boolean [][] getBlackCells() {
    return blackCells;
}
public String printBlackCells() {
    String data = "";
    for ( int i = 0; i < MAX_SIZE; i++) {
        for ( int j = 0; j < MAX_SIZE; j++) {
            data = data + blackCells[i][j] + " ";
        }
        data = data + "\n";
    }
    return data;
}

public String [] getAcrossClues () {
    return acrossClues;
}
public String printAcrossClues () {
    String data = "";
    for ( int i = 0; i < MAX_SIZE; i++) {
        data = data + acrossClues[i] + "\n";
    }
    return data;
}
```

```
public String [] getDownClues() {
    return downClues;
}
public String printDownClues () {
    String data = "";
    for ( int i = 0; i < MAX_SIZE; i++) {
        data = data + downClues[i] + "\n";
    }
    return data;
}

public int [][] getLittleNumbers() {
    return littleNumbers;
}
public String printlittleNumbers() {
    String data = "";
    for ( int i = 0; i < MAX_SIZE; i++) {
        for ( int j = 0; j < MAX_SIZE; j++) {
            data = data + littleNumbers[i][j] + " ";
        }
        data = data + "\n";
    }
    return data;
}

public String [][] getSolution() {
    return solution;
}
public String printSolution() {
    String data = "";
    for ( int i = 0; i < MAX_SIZE; i++) {
        for ( int j = 0; j < MAX_SIZE; j++) {
            data = data + solution[i][j] + " ";
        }
        data = data + "\n";
    }
    return data;
}

public String getDate() {
    return date;
}
public int [] getAcrossClueNumbers () {
    return acrossClueNumbers;
}
public String printAcrossClueNumbers () {
    String data = "";
    for ( int i = 0; i < MAX_SIZE; i++) {
        data = data + acrossClueNumbers[i] + " ";
    }
    return data;
}

public int [] getDownClueNumbers () {
    return downClueNumbers;
}
public String printDownClueNumber () {
    String data = "";
    for ( int i = 0; i < MAX_SIZE; i++) {
```

```
            data = data + downClueNumbers[i] + " ";
        }
        return data;
    }
}
```

```
//CREATING PIPELINE FOR STANFORD CORENLP
package analyzer;

import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import org.apache.log4j.BasicConfigurator;

import java.util.Properties;

public class Pipeline {

    private static Properties properties;
    // tokenize, ssplit, pos, lemma, ner
    private static String propertyNames = "tokenize,ssplit,pos,lemma, ner";
    private static StanfordCoreNLP stanfordCoreNLP;

    private Pipeline () {

    }

    static {
        properties = new Properties();
        properties.setProperty( "annotators", propertyNames);
        properties.setProperty("coref.algorithm", "neural");
    }
    public static StanfordCoreNLP getPipeline() {
        if ( stanfordCoreNLP == null) {
            BasicConfigurator.configure();
            stanfordCoreNLP = new StanfordCoreNLP( properties);
        }
        return stanfordCoreNLP;
    }
}
```

```
//RELATIONSIP FINDER IN WORDNET
package analyzer;

import net.sf.extjwnl.JWNLException;
import net.sf.extjwnl.data.PointerUtils;
import net.sf.extjwnl.data.Synset;
import net.sf.extjwnl.data.list.PointerTargetNode;
import net.sf.extjwnl.data.list.PointerTargetNodeList;

public class RelationshipExplorer {
    private RelationshipExplorer() {}
    public static PointerTargetNodeList getAntonyms(Synset synset) {
        try {
            return PointerUtils.getAntonyms( synset);
        }catch (JWNLException e){
            return null;
```

```
    }
}
public static PointerTargetNodeList getAttributes (Synset synset) {
    try {
        return PointerUtils.getAttributes( synset);
    } catch (JWNLException e) {
        return null;
    }
}
public static PointerTargetNodeList findChildren(Synset synset) {
    try {
        return PointerUtils.getDirectHyponyms( synset);
    }catch (JWNLException e) {
        return  null;
    }

}
public static PointerTargetNodeList findParents(Synset synset) {
    try {
        return PointerUtils.getDirectHypernyms(synset);
    }
    catch ( JWNLException e) {
        return null;
    }
}
public static PointerTargetNodeList getSynonym(Synset synset) {
    try {
        return PointerUtils.getSynonyms(synset);
    } catch (JWNLException e) {
        return null;
    }

}

// I added:


public static PointerTargetNodeList getMemberHolonyms(Synset synset) {
    try {
        return PointerUtils.getMemberHolonyms(synset);
    } catch (JWNLException e) {
        return null;
    }
}

public static PointerTargetNodeList getPartHolonyms(Synset synset) {
    try {
        return PointerUtils.getPartHolonyms(synset);
    } catch (JWNLException e) {
        return null;
    }
}


public static PointerTargetNodeList getSubstanceHolonyms(Synset synset) {
    try {
        return PointerUtils.getSubstanceHolonyms(synset);
    } catch (JWNLException e) {
        return null;
```

```
    }
}

public static PointerTargetNodeList getAlsoSee(Synset synset) {
    try {
        return PointerUtils.getAlsoSees(synset);
    } catch (JWNLException e) {
        return null;
    }

}
public static PointerTargetNodeList getCause(Synset synset) {
    try {
        return PointerUtils.getCauses(synset);
    } catch (JWNLException e) {
        return null;
    }

}
public static PointerTargetNodeList getCoordinateTerm(Synset synset) {
    try {
        return PointerUtils.getCoordinateTerms(synset);
    } catch (JWNLException e) {
        return null;
    }

}
public static PointerTargetNodeList getEntailment(Synset synset) {
    try {
        return PointerUtils.getEntailments(synset);
    } catch (JWNLException e) {
        return null;
    }

}
public static PointerTargetNodeList getParticiple(Synset synset) {
    try {
        return PointerUtils.getParticipleOf(synset);
    } catch (JWNLException e) {
        return null;
    }

}

/*
    A car is a member of a traffic jam.
    Car --> member meronym of traffic jam.
    Bir bütünün üyeleri
     */
public static PointerTargetNodeList getMemberMeronyms (Synset synset) {
    try {
        return PointerUtils.getMemberMeronyms( synset);
    } catch (JWNLException e) {
        return null;
    }
}
/*
A tire is a part of a car
tire --> part meronym of car
```

```
    Bir bütünün parçaları
     */
    public static PointerTargetNodeList getPartMeronyms (Synset synset) {
        try {
            return PointerUtils.getPartMeronyms( synset);
        } catch (JWNLException e) {
            return null;
        }
    }
    /*
    A wheel is made from rubber
    rubber --> substance meronym of wheel
    Neyden yapıldığını döndürüyor
     */
    public static PointerTargetNodeList getSubstanceMeronyms (Synset synset) {
        try {
            return PointerUtils.getSubstanceMeronyms( synset);
        } catch (JWNLException e) {
            return null;
        }
    }

}
```

```
// STANFORD CORENLP PROCESSOR
package analyzer;

import edu.stanford.nlp.ling.CoreAnnotations;
import edu.stanford.nlp.ling.CoreLabel;
import edu.stanford.nlp.pipeline.CoreDocument;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;

import java.util.List;

public class StanfordNLPpos {
    private String PLURAL = "NNS";
    private String PAST = "VBD";
    private String VERB_PARTICIPLE = "VBN";
    private String BASE_VERB = "VB";
    private String ADJECTIVE = "JJ";
    private String PERSON = "PERSON";
    private String COUNTRY = "COUNTRY";
    private String CITY = "CITY";
    private String LOCATION = "LOCATION";
    private String answer;
    public boolean isPlural = false;
    public boolean isPast = false;
    public boolean isPerson = false;
    public boolean isPlace = false;
    public String lemma = "";
    public StanfordNLPpos (String answer) {
        this.answer = answer;
        getPOS();
        getEntity();
    }

    public void getPOS () {
        StanfordCoreNLP stanfordCoreNLP = Pipeline.getPipeline();
```

```
        CoreDocument coreDocument = new CoreDocument( answer);

        stanfordCoreNLP.annotate( coreDocument);

        List<CoreLabel> coreLabelList = coreDocument.tokens();
        if ( coreLabelList.size() > 0) {
            String pos =
coreLabelList.get(0).get(CoreAnnotations.PartOfSpeechAnnotation.class);
            if (pos.equals(PLURAL)  ) {
                CoreDocument coreDocument1 = new CoreDocument( answer + " TO");
                stanfordCoreNLP.annotate( coreDocument1);
                List<CoreLabel> coreLabelList1 = coreDocument1.tokens();
                if ( coreLabelList1.size() > 0) {
                    String pos1 =
coreLabelList1.get(0).get(CoreAnnotations.PartOfSpeechAnnotation.class);
                    if ( pos1.equals( VERB_PARTICIPLE) || pos1.equals( ADJECTIVE)
|| pos1.equals( BASE_VERB))
                        isPlural = false;
                    else
                        isPlural = true;
                }
            } else if (pos.equals(PAST)) {
                isPast = true;
            }
        }
    }
    public void getEntity() {
        StanfordCoreNLP stanfordCoreNLP = Pipeline.getPipeline();
        CoreDocument coreDocument = new CoreDocument( answer);

        stanfordCoreNLP.annotate( coreDocument);
        List<CoreLabel> coreLabelList = coreDocument.tokens();
        if ( coreLabelList.size() > 0) {
            String ner =
coreLabelList.get(0).get(CoreAnnotations.NamedEntityTagAnnotation.class);
            if (ner.equals(PERSON))
                isPerson = true;
            if (ner.equals(COUNTRY) || ner.equals(CITY) || ner.equals(LOCATION))
                isPlace = true;
        }
    }
    public String getLemma () {
        StanfordCoreNLP stanfordCoreNLP = Pipeline.getPipeline();
        CoreDocument coreDocument = new CoreDocument( answer);
        stanfordCoreNLP.annotate( coreDocument);
        return coreDocument.tokens().get(0).lemma();
    }
}
```

```
// CLASS FOR EXTRACTING SOME FEATURES
package analyzer;

import java.util.ArrayList;

public class StringAnalyzer {

    private StringAnalyzer () {}

    public static String [] getWords (String givenString) {
```

```
        String WORD_KEY = "Words: ";
        String END_OF_WORD_KEY = " --";

        int start =  givenString.indexOf(WORD_KEY);
        int end = givenString.indexOf(END_OF_WORD_KEY);
        String extract = givenString.substring( start + WORD_KEY.length() , end);
        String [] words = extract.split( ", ");
        return words;
    }
    public static String getDescription( String givenString) {
        String BEGIN = "(";
        String END = ")";

        int start = givenString.indexOf( BEGIN);
        int end = givenString.lastIndexOf( END);
        String extract = givenString.substring( start+1, end);
        if ( extract.contains( "; \"")) {
            int index = extract.indexOf( "; \"");
            extract = extract.substring( 0,index);
        }

        if ( extract.contains( ";")) {
            extract = extract.substring( 0, extract.indexOf( ";"));
        }

        return extract;
    }
    public static ArrayList<String> getExampleSentences (String givenString) {
        ArrayList <String> sentences = new ArrayList<>();
        return sentences;
    }
}
```

```
// CLASS FOR ANALYZING A WORD IN WORNDET
package analyzer;

import net.sf.extjwnl.JWNLException;
import net.sf.extjwnl.data.*;
import net.sf.extjwnl.dictionary.Dictionary;

import java.util.*;

public class WordAnalyzer {

    private Dictionary dictionary;
    private String word;
    private IndexWordSet wordSet;
    private boolean isValid;
    private IndexWord indexWord;

    public WordAnalyzer(Dictionary dictionary, String word) {
        this.dictionary = dictionary;
        this.word = word;
        isValid = true;
        createWordSet();
    }

    private void createWordSet ()   {
        try {
```

```
            wordSet = dictionary.lookupAllIndexWords(word);
            if ( wordSet.getIndexWordCollection().size() == 0)
                isValid = false;
        }catch (JWNLException e) {
            e.printStackTrace();
        }
    }

    public Set<POS> getPOSSet () {
        return wordSet.getValidPOSSet();
    }

    public IndexWord getIndexWord(POS pos) {
        try {
            return dictionary.lookupIndexWord( pos, word);
        } catch (JWNLException e) {
            return null;
        }
    }

    public boolean isValid() {
        return isValid;
    }
}
```

```
//CLASS FOR ACCESSING PARAPHRASING TOOL
package clueFactory;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class ParaphraseFactory {
    private WebDriver driver;
    private String generatedClue;
    private String clue;
   public ParaphraseFactory (WebDriver driver, String clue) {
       this.driver = driver;
       this.clue = clue;
        goToWebsite();
        enterClue();
        getClue();
    }
    private void goToWebsite() {
       // Driver is being initialized for the first time
        if ( driver == null) {
            driver = new ChromeDriver();
            driver.get("https://codebeautify.org/paraphrasing-tool");  // Go to
URL
            driver.manage().window().maximize();  // Maximize Screen

        }
        // Refresh the existing page
        else {
            driver.navigate().refresh();
        }
    }
```

```
    private void enterClue() {
       sleep( 1000);
        WebElement textArea = driver.findElement(By.className("ace_text-input"));
        int i = 3;
        textArea.sendKeys(clue);
        sleep(2000);
        while ( i > 0 ) {
            WebElement button = driver.findElement(By.id("defaultaction"));
            button.click();
            sleep(5000);
            generatedClue = driver.findElement(By.id("mainRightDiv")).getText();
            generatedClue = generatedClue.substring(9);
            sleep(2000);
            if ( ! generatedClue.equalsIgnoreCase( clue)) {
                break;
            }
            i--;
        }

        driver.close();
        driver.quit();

    }
    private void sleep(int ms) {
        try {
            Thread.sleep( ms);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public String getGeneratedClue() {
        return generatedClue;
    }
}
```

```
// CLASS FOR GENERATIN RELATIPNSHIPS IN WORDNET
package clueFactory;

import analyzer.Pipeline;
import analyzer.RelationshipExplorer;
import analyzer.StringAnalyzer;
import analyzer.WordAnalyzer;
import edu.stanford.nlp.ling.CoreLabel;
import edu.stanford.nlp.pipeline.CoreDocument;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import net.sf.extjwnl.JWNLException;
import net.sf.extjwnl.data.*;
import net.sf.extjwnl.data.list.PointerTargetNode;
import net.sf.extjwnl.data.list.PointerTargetNodeList;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Set;

public class WordNetFactory {

    private WordAnalyzer wordAnalyzer;
    private String originalSolution;
```

```
    private Set <POS> available;
    private HashMap <String, ArrayList<String>> generatedSolutions;

    public WordNetFactory(Set<POS> available, String originalSolution,
WordAnalyzer wordAnalyzer) throws JWNLException {
        StanfordCoreNLP coreNLP = Pipeline.getPipeline();
        CoreDocument document = new CoreDocument( originalSolution);
        coreNLP.annotate( document);
        List <CoreLabel> lemma = document.tokens();
        originalSolution = lemma.get(0).lemma();
        this.originalSolution = originalSolution;
        this.available = available;
        this.wordAnalyzer = wordAnalyzer;
        generatedSolutions = new HashMap<>();
        generateSolutions();
    }
    private void generateSolutions() throws JWNLException {

        generateParticiple();
        generateCause();
        generateEntailments();
        generateSubstanceHolonyms();
        generateSubstanceMeronyms();
        generatePartHolonyms();
        generatePartMeronyms();
        generateAttributes();
        generateSees();
        generateCoordinateTerm();
        generateAntonyms();
        generateSynonyms();
        generateParents();
        generateChildren();
        generateSenses();

        generateMemberHolonyms();
        generateMemberMeronyms();

    }


    /*
    participle
     */
    private void generateParticiple(){
        generatedSolutions.put("participle", new ArrayList<>());
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            for (Synset sense :senses) {
                PointerTargetNodeList participle =
RelationshipExplorer.getParticiple( sense);
                for (PointerTargetNode node : participle) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words)
                        generatedSolutions.get("participle").add(word);
                }
            }
        }
    }
```

```
    // non causative counterpart of a verb; Example: “burn” is a cause
    //of “ignite”
    private void generateCause(){
        generatedSolutions.put("cause", new ArrayList<>());
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            boolean found = false;
            int i = 0;
            while ( !found && i < senses.size()){
                PointerTargetNodeList cause =
RelationshipExplorer.getCause(senses.get(i));
                for (PointerTargetNode node : cause) {
                    found = true;
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words) {
                        if (! word.toLowerCase().contains(
originalSolution.toLowerCase()))
                        generatedSolutions.get("cause").add(word);
                    }
                }
                i++;
            }
        }
    }


    //An entailment is an implication. For example, looking implies seeing. Buying
implies choosing and paying.
    private void generateEntailments(){
        generatedSolutions.put("entailments", new ArrayList<>());
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            for (Synset sense :senses) {
                PointerTargetNodeList entailments =
RelationshipExplorer.getEntailment( sense);
                for (PointerTargetNode node : entailments) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words)
                        generatedSolutions.get("entailments").add(word);
                }
            }
        }
    }


    /*
    Example: if (input -> copper), output -> [brass, bronze, bornite,  peacock
ore, chalcocite,  copper glance, chalcopyrite,  copper pyrites, cuprite,
malachite]
    Example: if (input -> hydrogen), output -> [water,  H2O]
    Example: if (input -> carbon), [coal, petroleum,  crude oil,  crude,  rock
oil,  fossil oil,  oil, limestone]
    */
    private void generateSubstanceHolonyms() {
        generatedSolutions.put( "substance_holonyms", new ArrayList<>());
```

```
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List<Synset> senses = indexWord.getSenses();
            for ( Synset sense : senses) {
                PointerTargetNodeList substanceholonyms =
RelationshipExplorer.getSubstanceHolonyms( sense);
                for ( PointerTargetNode node : substanceholonyms) {
                    String [] words = StringAnalyzer.getWords( node.toString());
                    for ( String word : words)
                        generatedSolutions.get( "substance_holonyms").add( word);
                }
            }
        }
    }


    private void generateSubstanceMeronyms () {
        generatedSolutions.put("substance_meronyms", new ArrayList<>());

        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord(pos);
            List<Synset> senses = indexWord.getSenses();
            if (senses.size() != 0) {
                PointerTargetNodeList substanceMeronyms =
RelationshipExplorer.getSubstanceMeronyms(senses.get(0));
            for (PointerTargetNode node : substanceMeronyms) {
                String[] words = StringAnalyzer.getWords(node.toString());
                for (String word : words) {
                    if ( !word.toLowerCase().contains(
originalSolution.toLowerCase()))
                    generatedSolutions.get("substance_meronyms").add(word);

                }
            }
        }

        }
    }


    /*
    It just returns part relationship.
    Example: if (input -> arm), then the method returns body because arm is a part
of human body.
    We can modify the solution as "It is a part of .....".
     */
    private void generatePartHolonyms()  {
        generatedSolutions.put( "part_holonyms", new ArrayList<>());

        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List<Synset> senses = indexWord.getSenses();
            for ( Synset sense : senses) {
                PointerTargetNodeList partholonyms =
RelationshipExplorer.getPartHolonyms( sense);
                for ( PointerTargetNode node : partholonyms) {
                    String [] words = StringAnalyzer.getWords( node.toString());
                    for ( String word : words) {
```

```
                        if ( !word.toLowerCase().contains(
originalSolution.toLowerCase()))
                            generatedSolutions.get("part_holonyms").add(word);
                        }
                    }
                }
            }
        }


    private void generatePartMeronyms () {
        generatedSolutions.put("part_meronyms", new ArrayList<>());

        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List<Synset> senses = indexWord.getSenses();
            if ( !senses.isEmpty()) {
                PointerTargetNodeList partMeronyms =
RelationshipExplorer.getPartMeronyms(senses.get(0));
                for (PointerTargetNode node : partMeronyms) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words)
                        generatedSolutions.get("part_meronyms").add(word);
                }
            }

        }
    }


    // POS has to be noun, in order to find the words' attributes.
    private void generateAttributes() {
        generatedSolutions.put( "attributes", new ArrayList<>());

        if ( available.contains( POS.NOUN)) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( POS.NOUN);
            List <Synset> senses = indexWord.getSenses();
            for ( Synset sense: senses) {
                PointerTargetNodeList attributes =
RelationshipExplorer.getAttributes( sense);
                for ( PointerTargetNode node : attributes) {
                    String [] words = StringAnalyzer.getWords( node.toString());
                    for ( String word : words)
                        generatedSolutions.get( "attributes").add( word);
                }
            }
        }
    }


    //Find what words are related to synset
    private void generateSees(){
        generatedSolutions.put("sees", new ArrayList<>());
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            for (Synset sense :senses) {
                PointerTargetNodeList sees = RelationshipExplorer.getAlsoSee(
sense);
```

```
                for (PointerTargetNode node : sees) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words) {
                        if ( !word.toLowerCase().contains(
originalSolution.toLowerCase()))
                        generatedSolutions.get("sees").add(word);
                    }
                }
            }
        }
    }


    //Coordinate terms are nouns or verbs that have the same hypernym .
    private void generateCoordinateTerm(){
        generatedSolutions.put("coordinateTerm", new ArrayList<>());
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            for (Synset sense :senses) {
                PointerTargetNodeList coordinateTerm =
RelationshipExplorer.getCoordinateTerm( sense);
                for (PointerTargetNode node : coordinateTerm) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words)
                        generatedSolutions.get("coordinateTerm").add(word);
                }
            }
        }
    }


    private void generateAntonyms() {
        generatedSolutions.put( "antonyms", new ArrayList<>());
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            for (Synset sense :senses) {
                PointerTargetNodeList antonyms = RelationshipExplorer.getAntonyms(
sense);
                for (PointerTargetNode node : antonyms) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    //    System.out.println( node.getWord().toString());
                    for (String word : words) {
                        if ( !word.toLowerCase().contains(
originalSolution.toLowerCase()))
                        generatedSolutions.get("antonyms").add(word);
                    }
                }


            }
        }
    }


    // Hande
    private void generateSynonyms() {
        generatedSolutions.put( "synonyms", new ArrayList<>());
```

```
        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            for (Synset sense :senses) {
                PointerTargetNodeList synonyms = RelationshipExplorer.getSynonym(
sense);
                for (PointerTargetNode node : synonyms) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words) {
                        if ( !word.toLowerCase().contains(
originalSolution.toLowerCase()))
                        generatedSolutions.get("synonyms").add(word);
                    }
                }
            }
        }
    }


    private void generateParents () {
        generatedSolutions.put("parents", new ArrayList<>());

        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List<Synset> senses = indexWord.getSenses();
            for ( Synset sense : senses) {
                PointerTargetNodeList parents = RelationshipExplorer.findParents(
sense);
                for ( PointerTargetNode node : parents) {
                    String [] words = StringAnalyzer.getWords( node.toString());
                    for ( String word : words)
                        generatedSolutions.get( "parents").add( word);
                }
            }
        }
    }


    private void generateChildren() {
        generatedSolutions.put( "children", new ArrayList<>());

        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List<Synset> senses = indexWord.getSenses();
            for ( Synset sense : senses) {
                PointerTargetNodeList children =
RelationshipExplorer.findChildren( sense);
                for (PointerTargetNode node : children) {
                    String[] words = StringAnalyzer.getWords(node.toString());
                    for (String word : words) {
                        if ( !word.toLowerCase().contains(
originalSolution.toLowerCase()) && !word.contains( "dug"))
                        generatedSolutions.get("children").add(word);
                    }
                }


            }
        }
```

```
    }


    // Hamza
    private void generateSenses () {
        generatedSolutions.put( "senses", new ArrayList<>());
        generatedSolutions.put( "description" , new ArrayList<>());
        for ( POS pos: available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List <Synset> senses = indexWord.getSenses();
            for ( Synset sense: senses)  {
                List <Word> words = sense.getWords();
                for ( Word word: words) {
                    generatedSolutions.get("senses").add( word.getLemma());
                    generatedSolutions.get( "description").add(
StringAnalyzer.getDescription( word.toString()));
                }
            }
        }
    }


    /*
    It just returns member relationship not all part relationship.
    Example: if (input -> student), output -> [teacher-student relation].
Likewise, if (input -> teacher), output is the same.
    Example: if (input -> lawyer), output -> [lawyer-client relation,  attorney-
client relation]
     */
    private void generateMemberHolonyms() {
        generatedSolutions.put( "memberholonyms", new ArrayList<>());

        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List<Synset> senses = indexWord.getSenses();
            for ( Synset sense : senses) {
                PointerTargetNodeList memberholonyms =
RelationshipExplorer.getMemberHolonyms( sense);
                for ( PointerTargetNode node : memberholonyms) {
                    String [] words = StringAnalyzer.getWords( node.toString());
                    for ( String word : words)
                        generatedSolutions.get( "memberholonyms").add( word);
                }
            }
        }
    }


    // Canbo
    private void generateMemberMeronyms () {
        generatedSolutions.put("member meronyms", new ArrayList<>());

        for ( POS pos : available) {
            IndexWord indexWord = wordAnalyzer.getIndexWord( pos);
            List<Synset> senses = indexWord.getSenses();
            for ( Synset sense : senses) {
                PointerTargetNodeList memberMeronyms =
RelationshipExplorer.getMemberMeronyms( sense);
                for ( PointerTargetNode node : memberMeronyms) {
```

```
                    String [] words = StringAnalyzer.getWords( node.toString());
                    for ( String word : words)
                        generatedSolutions.get( "member meronyms").add( word);
                }
            }
        }
    }


    public HashMap<String, ArrayList<String>> getGeneratedSolutions() {
        return generatedSolutions;
    }
}
```

```
//CLASS FOR DRAWING PUZZLE
package gui;

import Main.ClueGenerator;
import accessToNewYorkTimes.Informations;
import accessToNewYorkTimes.PuzzleDatabase;
import net.sf.extjwnl.JWNLException;

import javax.swing.*;
import java.awt.*;

public class CellsPanel extends JPanel {

    // constant
    private final int CELL_SIZE = 100;

    // property
    private ClueGenerator clueGenerator;    // From this attribute, you can access
new generated clues: getNewAcrossClues(), getNewDownClues()
   // private Informations info;      // From this attribute, you access anything
on the original NYTimes puzzle.
    private PuzzleDatabase database;
    // constructor
    public CellsPanel() throws InterruptedException, JWNLException {
        clueGenerator = new ClueGenerator();
       // info = clueGenerator.getInformations();
        database = new PuzzleDatabase();
    }

    // methods

    public void draw(Graphics g, int x, int y, boolean isBlack) {
        if( isBlack) {
            g.setColor( Color.BLACK);
            g.fillRect(x+50, y+50, CELL_SIZE, CELL_SIZE - 2);
            g.setColor( Color.GRAY);
            g.drawRect( x+50, y+50, CELL_SIZE, CELL_SIZE-2);
        }
        else {
            g.setColor( Color.WHITE);
            g.fillRect(x+50, y+50, CELL_SIZE, CELL_SIZE-2);
            g.setColor( Color.GRAY);
            g.drawRect( x+50, y+50, CELL_SIZE, CELL_SIZE - 2);
        }
```

```
    }

    public void drawLetter( Graphics g, int x, int y, String letter) {
        g.setFont( new Font("default", Font.PLAIN, 36));
        g.setColor( Color.RED);
        g.drawString( letter, x*CELL_SIZE+CELL_SIZE/2+50-10,
y*CELL_SIZE+CELL_SIZE/2+50+10);
    }

    public void drawNumber( Graphics g, int x, int y, int number) {
        g.setFont( new Font("default", Font.BOLD, 20));
        g.setColor( Color.BLACK);
        g.drawString( ""+number, x*CELL_SIZE+CELL_SIZE/7+50,
y*CELL_SIZE+CELL_SIZE/5+50);
    }

    @Override
    public void paintComponent( Graphics g) {
        super.paintComponent( g);
        int x = 0;
        int y = 0;

        boolean [][] blackCells = database.getBlackCells();
        String[][] solutions = database.getSolution();
        int[][] numbers = database.getLittleNumbers();

        for( int i = 0; i < 5; i++) {
            x = 0;
            for( int j = 0; j < 5; j++) {
                draw( g, x, y, blackCells[i][j] );
                if( solutions[i][j] != null) {
                    drawLetter( g, j, i, solutions[i][j]);
                    if( numbers[i][j] != 0)
                        drawNumber( g, j, i, numbers[i][j]);
                }

                x = x + CELL_SIZE;
            }

            y = y + CELL_SIZE -2;
        }
    }


    public ClueGenerator getClueGenerator() {
        return clueGenerator;
    }
    public PuzzleDatabase getInfo() {
        return database;
    }
}
```

```
//CLASS FOR DRAWING PUZZLE
package gui;

import net.sf.extjwnl.JWNLException;

import javax.swing.*;
```

```
import java.awt.*;

public class PuzzlePanel extends JPanel {

    // properties
    private CellsPanel cellsPanel;
    private JTextField date;
    private JTextArea acrossCluesText;
    private JTextArea downCluesText;
    private JTextField groupNickname;

    // TO BE ADDED
    private JTextArea newAcrossCluesText;
    private JTextArea newDownCluesText;


    // constructor
    public PuzzlePanel() throws InterruptedException, JWNLException {
        JPanel cluesPanel = new JPanel();
        JPanel newCluesPanel = new JPanel();
        JPanel dateAndNickPanel = new JPanel();
        cellsPanel = new CellsPanel();

        acrossCluesText = new JTextArea(
                "ACROSS\n" + "\n");


        downCluesText = new JTextArea(
                "DOWN\n" + "\n");

        newAcrossCluesText = new JTextArea(
                "NEW ACROSS\n" + "\n");

        newDownCluesText = new JTextArea(
                "NEW DOWN\n" + "\n");


        writeCluesTexts();

        date = new JTextField( cellsPanel.getInfo().getDate());
        date.setFont(new Font("default", Font.PLAIN, 20));
        date.setEditable( false);

        groupNickname = new JTextField( "WORD++");
        groupNickname.setFont(new Font("default", Font.PLAIN, 20));
        groupNickname.setEditable( false);

       FlowLayout layout = new FlowLayout();
    //    layout.setVgap(70);
     //   layout.setHgap(50);
        dateAndNickPanel.setLayout( layout);
        dateAndNickPanel.setBackground(Color.WHITE);
        dateAndNickPanel.add( date);
        dateAndNickPanel.add( groupNickname);


        FlowLayout layout2 = new FlowLayout();
         layout2.setVgap(80);
        layout2.setHgap(50);
```

```
        cluesPanel.setLayout ( layout2);
        cluesPanel.setBackground(Color.WHITE);
        cluesPanel.add( acrossCluesText);
        cluesPanel.add( downCluesText);

        FlowLayout layout3 = new FlowLayout();
        layout3.setVgap(120);
        layout3.setHgap(50);
        newCluesPanel.setLayout( layout3);
        newCluesPanel.add( newAcrossCluesText);
        newCluesPanel.add( newDownCluesText);
        newCluesPanel.setBackground( Color.WHITE);


        cellsPanel.setBackground(Color.WHITE);
        JPanel leftPanel = new JPanel();
        cellsPanel.setPreferredSize( new Dimension( 960,360));
        dateAndNickPanel.setPreferredSize( new Dimension(960 ,50));
        leftPanel.add( cellsPanel);
        leftPanel.add( dateAndNickPanel);
        BoxLayout boxLayout = new BoxLayout( leftPanel, BoxLayout.Y_AXIS);
        leftPanel.setLayout( boxLayout);

        JPanel rightPanel = new JPanel();

        rightPanel.setBackground( Color.WHITE);
        cluesPanel.setPreferredSize( new Dimension(960, 250));
        newCluesPanel.setPreferredSize( new Dimension( 500, 540));
        rightPanel.add( cluesPanel);
        rightPanel.add( newCluesPanel);
        BoxLayout bl2 = new BoxLayout( rightPanel, BoxLayout.Y_AXIS);
        rightPanel.setLayout( bl2);

        leftPanel.setPreferredSize( new Dimension(960, 540));
        rightPanel.setPreferredSize( new Dimension( 960,540));
        add( leftPanel);
        add( rightPanel);
        BoxLayout bl3 = new BoxLayout( this, BoxLayout.X_AXIS);
        setLayout( bl3);


      //  add ( dateAndNickPanel, BorderLayout.SOUTH);

    }

    public void writeCluesTexts() {
        String[] acrossClues = cellsPanel.getInfo().getAcrossClues();
        String[] downClues = cellsPanel.getInfo().getDownClues();
        int[] acrossCluesNumbers = cellsPanel.getInfo().getAcrossClueNumbers();
        int[] downCluesNumbers = cellsPanel.getInfo().getDownClueNumbers();


        for( int i = 0; i < acrossClues.length; i++) {
            acrossCluesText.append( acrossCluesNumbers[i] + ". ");
            String [] clues =  acrossClues[i].split( " ");
            int slider = 1;
            for ( String part : clues) {
                acrossCluesText.append( part + " ");
                if ( slider % 8 == 0 && ((slider - clues.length) % 8 != 0)){
```

```
                    acrossCluesText.append( "\n");
                }
                slider++;
            }
            acrossCluesText.append( "\n");
        }

        for( int i = 0; i < downClues.length; i++) {
            downCluesText.append( downCluesNumbers[i] + ". ");
            String [] clues =  downClues[i].split( " ");
            int slider = 1;
            for ( String part : clues) {
                downCluesText.append( part + " ");
                if ( slider % 8 == 0 && ((slider - clues.length) % 8 != 0)){
                    downCluesText.append( "\n");
                }
                slider++;
            }
            downCluesText.append( "\n");
        }

        acrossCluesText.setFont(new Font("default", Font.PLAIN, 15));
        acrossCluesText.setEditable( false);
      //  acrossCluesText.setPreferredSize( new Dimension( 300, 500));
        downCluesText.setFont(new Font("default", Font.PLAIN, 15));
        downCluesText.setEditable( false);
    //    downCluesText.setPreferredSize( new Dimension( 300, 500));

        // NEW CLUES
        String[] newAcrossClues =
cellsPanel.getClueGenerator().getNewAcrossClues();
        String[] newDownClues = cellsPanel.getClueGenerator().getNewDownClues();

        for( int i = 0; i < newAcrossClues.length; i++) {
            newAcrossCluesText.append( acrossCluesNumbers[i] + ". ");
            String [] clues =  newAcrossClues[i].split( " ");
            int slider = 1;
            for ( String part : clues) {
                newAcrossCluesText.append( part + " ");
                if ( slider % 8 == 0 && ((slider - clues.length) % 8 != 0)){
                   newAcrossCluesText.append( "\n");
                }
                slider++;
            }
            newAcrossCluesText.append( "\n");
           // newAcrossCluesText.append(acrossCluesNumbers[i] + ". " +
newAcrossClues[i] + "\n");
        }

        for( int i = 0; i < newDownClues.length; i++) {
            newDownCluesText.append( downCluesNumbers[i] + ". ");
            String [] clues =  newDownClues[i].split( " ");
            int slider = 1;
            for ( String part : clues) {
                newDownCluesText.append( part + " ");
                if ( slider % 8 == 0 && ((slider - clues.length) % 8 != 0)){
                    newDownCluesText.append( "\n");
                }
                slider++;
```

```
            }
            newDownCluesText.append( "\n");
        }

        newAcrossCluesText.setFont(new Font("default", Font.PLAIN, 15));
        newAcrossCluesText.setEditable( false);
        newDownCluesText.setFont(new Font("default", Font.PLAIN, 15));
        newDownCluesText.setEditable( false);

    }
}
```

```
// MAIN CLASS
package Main;

import gui.PuzzlePanel;
import net.sf.extjwnl.JWNLException;

import javax.swing.*;
import java.awt.*;

public class AIProject {

    public static void main(String[] args) throws JWNLException,
InterruptedException {

        PuzzlePanel panel = new PuzzlePanel();

        JFrame frame = new JFrame( "WORD++");
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
        //frame.setState( JFrame.ICONIFIED);

        GraphicsEnvironment env =
GraphicsEnvironment.getLocalGraphicsEnvironment();
        frame.setExtendedState( frame.getExtendedState() | frame.MAXIMIZED_BOTH);

        frame.add( panel);
        frame.setLocationRelativeTo( null);

        frame.setVisible( true);
    }
}
```

```
//CLASS FOR DECIDING WHICH GENRATED CLUE WILL BE PICKED
package Main;

import accessToNewYorkTimes.Informations;
import accessToNewYorkTimes.PuzzleDatabase;
import analyzer.StanfordNLPpos;
import analyzer.WordAnalyzer;
import clueFactory.ParaphraseFactory;
import clueFactory.WordNetFactory;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import net.sf.extjwnl.JWNLException;
import net.sf.extjwnl.data.POS;
import net.sf.extjwnl.dictionary.Dictionary;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
```

```java
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.sql.SQLOutput;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;
import java.util.Set;

public class ClueGenerator {

    //private Informations informations;
    PuzzleDatabase puzzleDatabase;

    // These two arrays include solution of the clues in the "word" format.
    private String[] acrossWords;
    private String[] downWords;

    // These two arrays hold new generated clues.
    private String[] newAcrossClues;
    private String[] newDownClues;
    boolean isParent = false;
    boolean isChild = false;
    int antonymCount = 0;
    int synoymCount = 0;
    int aPartCount = 0;


    public ClueGenerator() throws InterruptedException, JWNLException {
        //informations = new Informations();
        puzzleDatabase = new PuzzleDatabase();

        acrossWords = new String[puzzleDatabase.getAcrossClues().length];
        downWords = new String[puzzleDatabase.getDownClues().length];

        takeWords();

        newAcrossClues = searchClues( acrossWords, true);
        newDownClues = searchClues( downWords, false);

    }

    /*
    This method finds solution of every clue (as a word) by using letters in the
original solution.
     */
    public void takeWords() {
        int[] acrossClueNumbers = puzzleDatabase.getAcrossClueNumbers();
        int[] downClueNumbers = puzzleDatabase.getDownClueNumbers();
        int[][] littleNumbers = puzzleDatabase.getLittleNumbers();
        String[][] solution = puzzleDatabase.getSolution();

        for( int i = 0; i < 5; i++){
            for( int j = 0; j < 5; j++){

                // For across words
```

```
                for( int k = 0; k < acrossClueNumbers.length; k++){
                    if( littleNumbers[i][j] == acrossClueNumbers[k]){
                        String str = "";
                        for( int m = j; m < solution[i].length && solution[i][m]
!= null; m++){
                            str += solution[i][m];
                        }
                        acrossWords[k] = str;
                    }
                }

                // For down words
                for( int k = 0; k < downClueNumbers.length; k++){
                    if( littleNumbers[i][j] == downClueNumbers[k]){
                        String str = "";
                        for( int m = i; m < solution.length && solution[m][j] !=
null; m++){
                            str += solution[m][j];
                        }
                        downWords[k] = str;
                    }
                }

            }
        }
    }

    public String[] searchClues( String[] answers, boolean isAcross) throws
JWNLException {
        String[] newClues = new String[ answers.length];
        System.out.println(answers.length);
        Dictionary dictionary = null;
        try {
            dictionary = Dictionary.getDefaultResourceInstance();
        } catch (JWNLException e) {
            e.printStackTrace();
        }


        for( int i = 0; i < answers.length; i++) {
            Scanner scan = new Scanner( System.in);
            System.out.println( "Answer is : " + answers [i]);
            System.out.println( "Press a key to continue");
            scan.next();
            boolean isDescription = false;
            WordAnalyzer analyzer = new WordAnalyzer(dictionary, answers[i]);
            if (analyzer.isValid()) {
                System.out.println( "Searching the answer in WordNet");
                Set<POS> available = analyzer.getPOSSet();
                WordNetFactory wordNetFactory = new WordNetFactory(available,
answers[i], analyzer);
                HashMap<String, ArrayList<String>> entry =
wordNetFactory.getGeneratedSolutions();

                if (entry.get("participle").size() != 0) {
                    System.out.println( "Participle is found");
                    newClues[i] = "" + entry.get("participle").get(0) + "
participle";
                } else if (entry.get("cause").size() != 0) {
```

```
                    System.out.println( "Cause is found");
                    String clue = "Result of ";
                    for ( int j = 0; j < entry.get("cause").size(); j++) {
                        if (j == entry.get("cause").size() - 1) {
                            clue += entry.get("cause").get(j) + ".";
                        } else {
                            String add;
                            if ( j == entry.get("cause").size() - 2)
                                add = " and ";
                            else
                                add = ", ";
                            clue += entry.get("cause").get(j) + add;
                        }
                    }
                    newClues [ i] = clue;
                } /*else if (entry.get("entailments").size() != 0) {
                    System.out.println( "Entailment is found");
                    String clue = "It implies ";
                    newClues[i] = clue + entry.get("entailments").get(0);
                } */else if (entry.get("substance_holonyms").size() != 0) {
                    System.out.println( "Substance holonym is found");
                    String clue = "A substance of ";
                    newClues[i] = clue + entry.get("substance_holonyms").get(0);

                } else if (entry.get("substance_meronyms").size() != 0) {
                    System.out.println( "Substance meronym is found");
                    String clue = "It consists of ";
                    for ( int j = 0; j < entry.get("substance_meronyms").size();
j++) {
                        if (j == entry.get("substance_meronyms").size() - 1) {
                            clue += entry.get("substance_meronyms").get(j) + ".";
                        } else {
                            clue += entry.get("substance_meronyms").get(j) + " and
";
                        }
                    }
                    newClues[i] = clue;
                } else if (entry.get("part_holonyms").size() != 0 && aPartCount <
2) {
                    System.out.println( "Part holonym is found");
                    String clue = "A part of ";
                    newClues[i] = clue + entry.get("part_holonyms").get(0);
                    aPartCount++;

                } else if (entry.get("part_meronyms").size() != 0) {
                    System.out.println( "Part Meronym is found");
                    String clue = "It has parts like ";
                    int count = 3;
                    for ( int j = 0; j < entry.get("part_meronyms").size() && j <
count; j++) {
                        if
(!(entry.get("part_meronyms").get(j).toLowerCase().contains(
answers[i].toLowerCase())) ) {
                            if (j == entry.get("part_meronyms").size() - 1 || j ==
count -1) {
                                clue += entry.get("part_meronyms").get(j) + ".";
                            } else {
                                if
(!(entry.get("part_meronyms").get(j).toLowerCase().contains(answers[i].toLowerCase
```

```
()))))
                                    clue += entry.get("part_meronyms").get(j) + "
and ";
                            }
                        }
                        else
                            count++;

                    }
                    newClues[i] = clue;

                }
                else if (entry.get("attributes").size() != 0) {
                    System.out.println( "Attribute is found");
                    String clue = "It can be ";
                    for ( int j = 0; j < entry.get("attributes").size(); j++) {
                        if ( j == entry.get("attributes").size() - 1) {
                            clue += entry.get("attributes").get(j) + ".";
                        }
                        else {
                            clue += entry.get("attributes").get(j) + " or ";
                        }
                    }
                    newClues[i] = clue;

            /*     } else if (entry.get("sees").size() != 0) {
                    System.out.println( "Also see is found ");
                    newClues[i] =  entry.get("sees").get(0);

                } else if (entry.get("coordinateTerm").size() != 0) {
                    newClues[i] = "" + entry.get("coordinateTerm");
                    System.out.println(answers[i] + " (coordinateTerm) :" +
entry.get("coordinateTerm"));
*/
                } else if (entry.get("antonyms").size() != 0 && antonymCount < 2)
{
                    System.out.println( "Antonym is found");
                    String clue = "Antonym of " + entry.get( "antonyms").get(0);
                    newClues[i] = clue;
                    antonymCount++;

                } else if (entry.get("synonyms").size() != 0 && synoymCount < 2) {
                    System.out.println( "Synonym is found");
                    String clue = "Synoym of " + entry.get("synonyms").get(0);
                    newClues[i] = clue;
                    synoymCount++;

                } else if (entry.get("parents").size() != 0 && !isParent) {
                    newClues[i] = "A kind of " + entry.get("parents").get(0);
                    System.out.println(answers[i] + " (parents) :" +
entry.get("parents"));
                    isParent = true;

                } else if (entry.get("children").size() != 0 && !isChild) {
                    System.out.println( "A child is found");
                    newClues[i] = "" + entry.get("children").get(0)+ " is a kind
of it";
                    isChild = true;
```

```
                } else if (entry.get("senses").size() != 0) {
                    System.out.println( "Senses are being searched");
                    boolean found = false;
                    for (int j = 0; j<  entry.get("senses").size(); j++) {
                        String clue = entry.get( "senses").get( j);

                        if ( !clue.equalsIgnoreCase( answers[i]) &&
!(clue.toLowerCase().contains( answers[i].toLowerCase()) )
                                && !(answers[i].toLowerCase().contains(
clue.toLowerCase())) )  {
                            newClues [i] = clue;
                            System.out.println( "A sense is found");
                            found = true;
                            break;
                        }
                    }
                    String clue = "";
                    if (!found) {
                        System.out.println( "Description is found.");
                        clue = entry.get("description").get(0);
                        newClues[i] = clue;
                        isDescription = true;
                    }
                }
                if (!isDescription) {
                    StanfordNLPpos nlPpos = new StanfordNLPpos(answers[i]);
                    if (nlPpos.isPlural)
                        newClues[i] += " (plural)";
                    if (nlPpos.isPast)
                        newClues[i] += " (past form)";
                }
                newClues[i] = newClues [i].substring(0,1).toUpperCase() +
newClues[i].substring( 1);
            }
            else {
                System.out.println( "Answer is not found in WordNet. Checking
StanfordCoreNLP");
                boolean found = false;
                System.setProperty("webdriver.chrome.driver",
"C:\\Users\\pehli\\Desktop\\chromedriver.exe");
                StanfordNLPpos nlPpos = new StanfordNLPpos(
answers[i].toUpperCase());

             if ( nlPpos.isPerson) {
                 System.out.println( "Answer is a Person. Checking Google");
                    String clue = "";
                 ChromeOptions options = new ChromeOptions();
                 options.addArguments("--lang=en-GB");
                 WebDriver driver = new ChromeDriver(options);
                    driver.get( "https://www.google.com");
                    driver.manage().window().maximize();
                    WebElement browser = driver.findElement( By.name("q"));
                    browser.sendKeys( answers[i] + "\n");
                 //   browser.submit();
                 WebElement myDynamicElement = (new WebDriverWait(driver, 10))

.until(ExpectedConditions.presenceOfElementLocated(By.id("resultStats")));
                 try {
```

```
                    Thread.sleep( 4000);

                WebElement job = driver.findElement(By.className("SPZz6b"));
                       clue = job.getText() ;
                      answers[i] =  answers[i].toLowerCase();
                    String first =  answers[i].substring(0,1).toUpperCase();
                    first = first + answers[i].substring(1);
                    answers[i] = first;
                 clue =  clue.replace( answers[i], "_____");
                   if ( !clue.contains("See results about")) {
                       newClues[i] = clue;
                       found = true;
                   }
                       driver.close();
                       driver.quit();
                   }catch ( Exception e) {
                       found = false;
                       driver.close();
                       driver.quit();
                   }
               }
            else if ( nlPpos.isPlace) {
                System.out.println( "Answer is a Place. Checking Google");
                ChromeOptions options = new ChromeOptions();
                options.addArguments("--lang=en-GB");
                WebDriver driver = new ChromeDriver(options);
                driver.get( "https://www.google.com");
                driver.manage().window().maximize();
                WebElement browser = driver.findElement( By.name("q"));
                browser.sendKeys( answers[i] + "\n");
                //   browser.submit();
                WebElement myDynamicElement = (new WebDriverWait(driver, 10))

.until(ExpectedConditions.presenceOfElementLocated(By.id("resultStats")));

                try {
                    WebElement element = driver.findElement(By.className("wwUB2c
PZPZlf E75vKf"));
                   String clue = element.getText();
                    int index = clue.indexOf( "\n");
                    clue = clue.substring( index);
                    newClues[i] = clue;
                    driver.close();
                    driver.quit();
                    found = true;
                }catch ( Exception e) {
                    driver.close();
                    driver.quit();
                    found = false;
                }
            }
            if (!found) {
                System.out.println("Nothing is found. Paraphrasing the clue");
                System.setProperty("webdriver.chrome.driver",
"C:\\Users\\pehli\\Desktop\\chromedriver.exe");
                WebDriver driver = null;

                ParaphraseFactory paraphraseFactory;
                if (isAcross) {
```

```
                    paraphraseFactory = new ParaphraseFactory(driver,
puzzleDatabase.getAcrossClues()[i]);
                    newClues[i] = paraphraseFactory.getGeneratedClue();
                } else {
                    paraphraseFactory = new ParaphraseFactory(driver,
puzzleDatabase.getDownClues()[i]);
                    newClues[i] = paraphraseFactory.getGeneratedClue();
                }

             }
            }
            System.out.println( "----------------------------------------------
--------------------------");
        }

        return newClues;
    }

  /*  public Informations getInformations() {
        return informations;
    }*/

    public String[] getAcrossWords() {
        return acrossWords;
    }

    public String[] getDownWords() {
        return downWords;
    }

    public String[] getNewAcrossClues() {
        return newAcrossClues;
    }

    public String[] getNewDownClues() {
        return newDownClues;
    }
}
```

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 WORD++.

Word Count: 2079