1. Introduction

This project required us to come up with an intelligent program to solve the New York Times daily crossword mini-puzzle. The mini-puzzle is basically a 5x5 crossword where you have to fill in words both across and down the grid. Our approach to solving this problem was that of using the constraint satisfaction algorithm. Constraint Satisfaction is a process of finding the solution to a set of variables which satisfy several different constraints. In our case, these constraints were the number of search hits and the length of the current word being searched for. As for the implementation, firstly we obtain the puzzle from New York Times's (NYT) website. After that we search all of the hints in Google, take the top 3 search hits, disregard all of the words which do not match the length of the required word and try to find the word in the remaining word data. Once found, we output that word to the screen.

2. New York Times Crossword

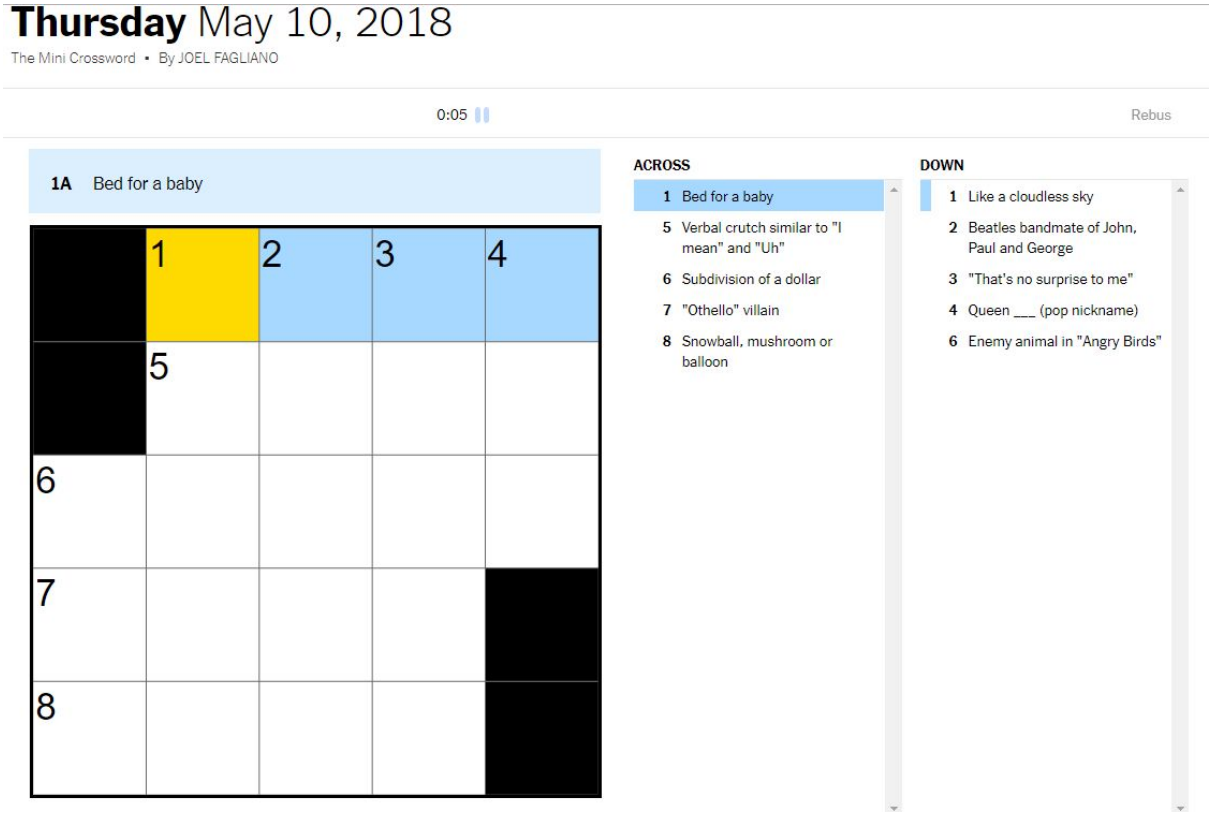

Figure 1: NYT Website

Above it is possible to see the mini-puzzle crossword by Joel Fagliano which is posted on NYT website. This particular mini-puzzle is for Thursday, May 10th, 2018. We can conclude that, everyday new, different, maybe way harder mini-puzzle is posted on the website. The picture above clearly visualizes the way our program GUI output should look like. Namely, the GUI output of our program should have the exact grid with the same numbers on the

cells and colors of them (f.e black or white). Additionally, the output should clearly display the across and down hints. There are always ten hints, five of which is across and other half is down.

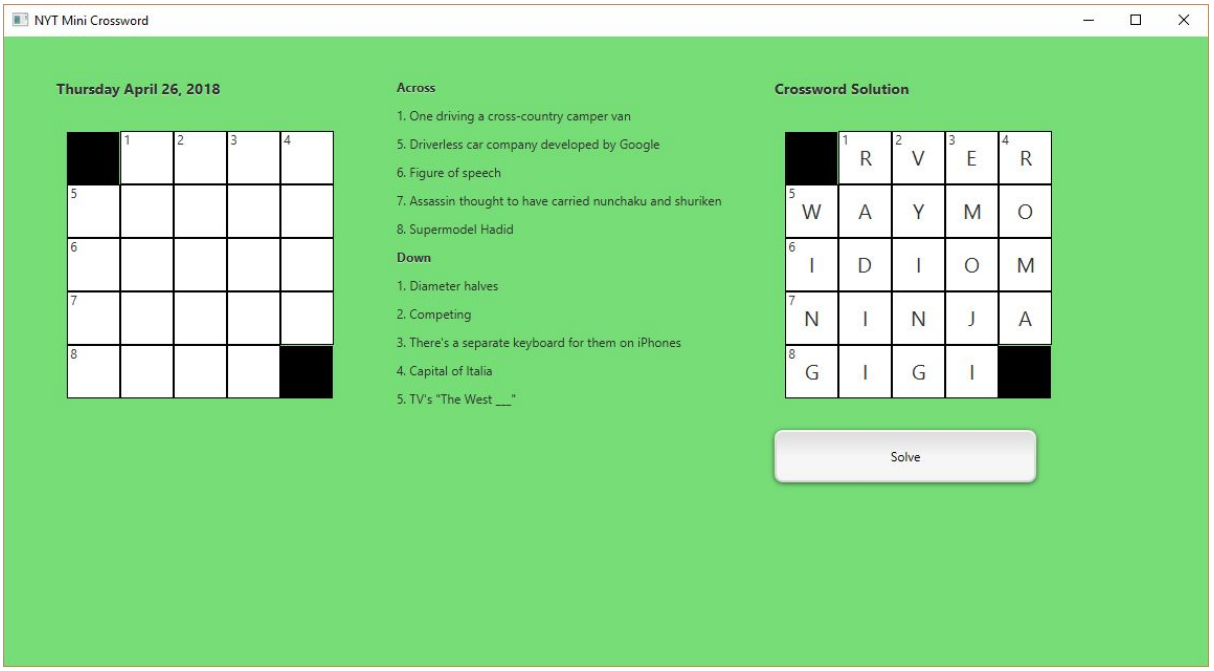

Figure 2: 26th April non-solved Output

The picture above is the illustration of Alfred's GUI output. The left side of the output represents the exact same crossword type as the one posted on NYT website (in this case, the one which was posted on 26th April). That is to say, the numbers on the cells, colors of the cells and the hints for each column are explicitly same as the posted crossword. How are we sure that they are same? Visible "Crossword Solution" on the right side of the output is the mini-puzzle which is identical to the one posted on the website. The solve button under the solution puzzle solves the puzzle box on the left and outputs the results. The tracing of the solving process is not shown on the GUI output but on the console. Further, the detailed information about how Alfred gets the identical puzzle and how it solves the puzzle will be provided.

3. Overview of Alfred

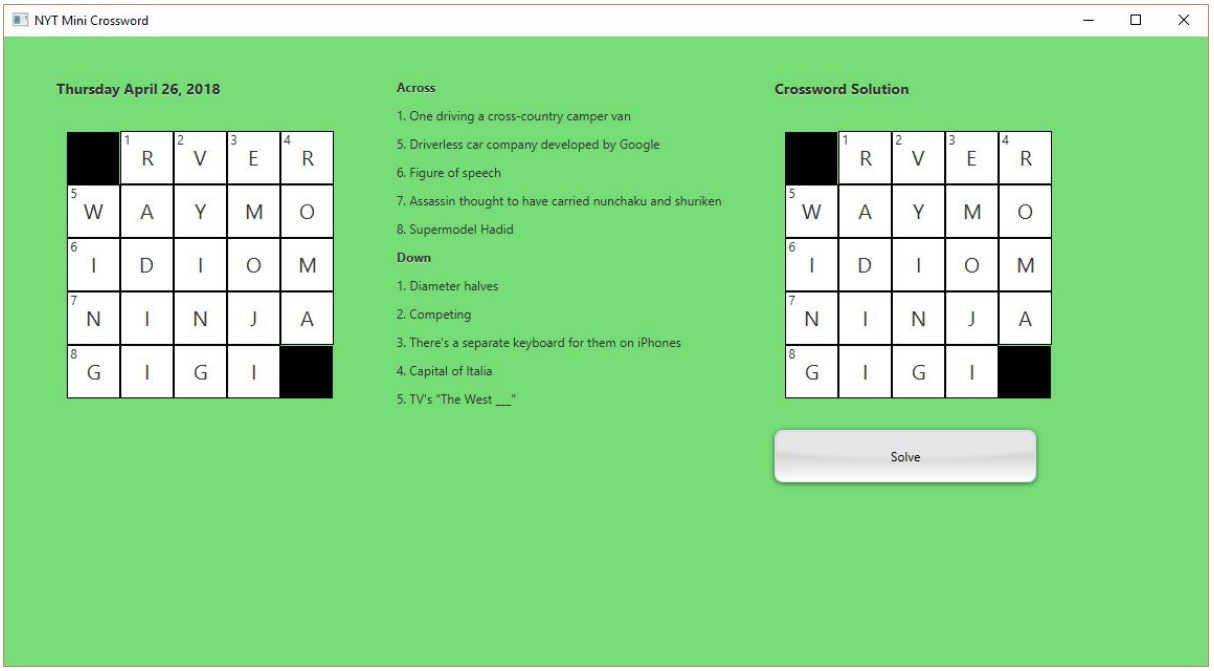

Figure 3: 26th April solved Output

The picture above is the 26th April, 2018's solved mini-puzzle.In this section, we will look at successful and not successful outputs, the reasons of success and the reasons of a failure, methodologies of the solutions.

3.1 Successful and Not Successful Outputs

Briefly to say, in order for an output to be a successful output, the puzzle box on the left has to be the identical twin of the puzzle on the right after the "solve" button is pressed. Conversely, if the puzzles are not the same then the output is considered as a failure. In other words, the successful puzzle is the puzzle which can get all ten words correctly. There is no in between. For illustration, in the picture above all following ten words: RVER, WAYMO, IDIOM, NINJA, GIGI, WING, RADII, VYING, EMOJI, ROMA are the correct words that fit the puzzle perfectly. When it comes to the a failure output, it simply does not solve the puzzle. It looks as a non-pressed solve button version of the output.

3.2 Reasons of Successful and Not Successful Outputs

The main and essential reason for the success is the ability of Alfred to find all ten words correctly. The way Alfred does that will be explained much more detailed in the class diagram explanation. However, before that, we would like to introduce you to Constraint Satisfaction Algorithm that Alfred uses for finding ten correct words out of all the words scraped from the net.

When Alfred first runs, the program scrapes the html of NYT website, stores it to the txt files named as "26AprilPuzzleCode.txt". Later, to find the solution, Alfred presses the reveal button on the website and scrapes the letters from the new html. It stores those letters line by line to "26AprilPuzzleAnswers.txt". This is how new puzzles are stored. When it comes to

the usage of old puzzles, Alfred simply reads the old txt files. After reading the txt file, it creates the GUI and outputs it. At this point, only the right puzzle box is filled because it is a solution box. When a user presses the solve button, Alfred starts “Googling”. It searches each hint in Google and scrapes all the words from the top three result websites. For each hint there is one corresponding ArrayList that stores the scraped words. Later, to make things easy, Alfred cleans the ArrayLists. Namely, for each hint there is a word length (for example, hint number 1 across on the picture above has a length of 4 letters) and Alfred removes the words which are longer than that specific length. Then, Alfred reduces the duplicates. Eventually, for each hint the clear and precise ArrayList remains. After getting all word lists, by using Constraint Satisfaction Algorithm (CSA) Alfred gets rid of unnecessary words. Finally, only ten words (one for each list) remains, which Alfred assumes are correct. It prints the words on the left puzzle box.
As you can guess, in the non-successful output’s final running process some of the lists remain empty due to lack of the correct word. Instead of a correct word, the list contains either null or nothing. Note that one lacking word is enough to have an unsuccessful output.

```
Candidates of 1. Across after clean up:
[RVER]
Candidates of 5. Across after clean up:
[WAYMO]
Candidates of 6. Across after clean up:
[IDIOM]
Candidates of 7. Across after clean up:
[NINJA]
Candidates of 8. Across after clean up:
[GIGI]
Candidates of 1. Down after clean up:
[RADII]
Candidates of 2. Down after clean up:
[VYING]
Candidates of 3. Down after clean up:
[EMOJI]
Candidates of 4. Down after clean up:
[ROMA]
Candidates of 5. Down after clean up:
[WING]
```

Figure 4: 26th Puzzle Successful Output

```
Candidates of 1. Across after clean up:
[null]
Candidates of 4. Across after clean up:
[null]
Candidates of 6. Across after clean up:
[null]
Candidates of 7. Across after clean up:
[null]
Candidates of 8. Across after clean up:
[]
Candidates of 1. Down after clean up:
[null]
Candidates of 2. Down after clean up:
[null]
Candidates of 3. Down after clean up:
[null]
Candidates of 4. Down after clean up:
[null]
Candidates of 5. Down after clean up:
[null]
```

Figure 5: 25th April non-successful Puzzle

3.3 Solution for Failures

Having failures in the program, however, is a solvable issue. There are mainly three solutions for this, which can be implemented to the code in the future: searching more websites for each hint and storing already existed hints and related words.

3.3.1 Searching More Websites for Each Hint

In this solution the number of the top websites that is scraped for each hint will increase from 3 to 10. This will ensure that Alfred got more words and more probability to solve the puzzle.

3.3.2 Storing Already Existed Hints and Related Words

In this solution not only the answers of the puzzles but also the hints with the corresponding word will be stored in a text file. This way, the efficiency of a program will increase. It will be faster and the possibility to find a word (for example, if the hints match) will be faster.

4. AI in Alfred

There are three main AI tools and elements used in Alfred: JSoup, Selenium and CSA.

4.1 JSoup Library

JSoup is a Java Library that scrapes html from the websites. Using this library we managed to scrape the html of the NYT website and store it in the text file.

4.2 Selenium Library

Selenium is a Java Library that artificially navigates in the internet. Using this library we managed to search hints, click buttons (reveal button on NYT website) or go back on websites to scrape the words later.

4.3 CSA

CSA, as mentioned before, is the main algorithm that Alfred uses for finding the correct words. Let's look deeper at how it works. As you know the logic of any crossword, there is always at least one letter in a word that is common with only one words any one letter. For example, in 26th March puzzle output the words RVER and RADII has one common letter R. We call them constraints. Namely, if two words have a common letter, we call it a constraint satisfaction. So, Alfred takes the first word from first hint's candidate list (ArrayList). Then, for each letter of that word, Alfred finds the intersection word. For example, in word RVER Alfred takes the first 'R', finds that it has an intersection with the first R of a word RADII which is a first down hint word (it continues until RVER is finished). During this intersection process, if none of the words in the first down hint's candidate list satisfies RVER's R, then the word RVER is eliminated. It continues until all lists have only one word remained. The advantage of CSA is that, it is not guessing the results by chance, but it uses solid algorithm to eliminate unnecessary words. This is why, Alfred can find all ten words very easily.

5. Class Diagram Description

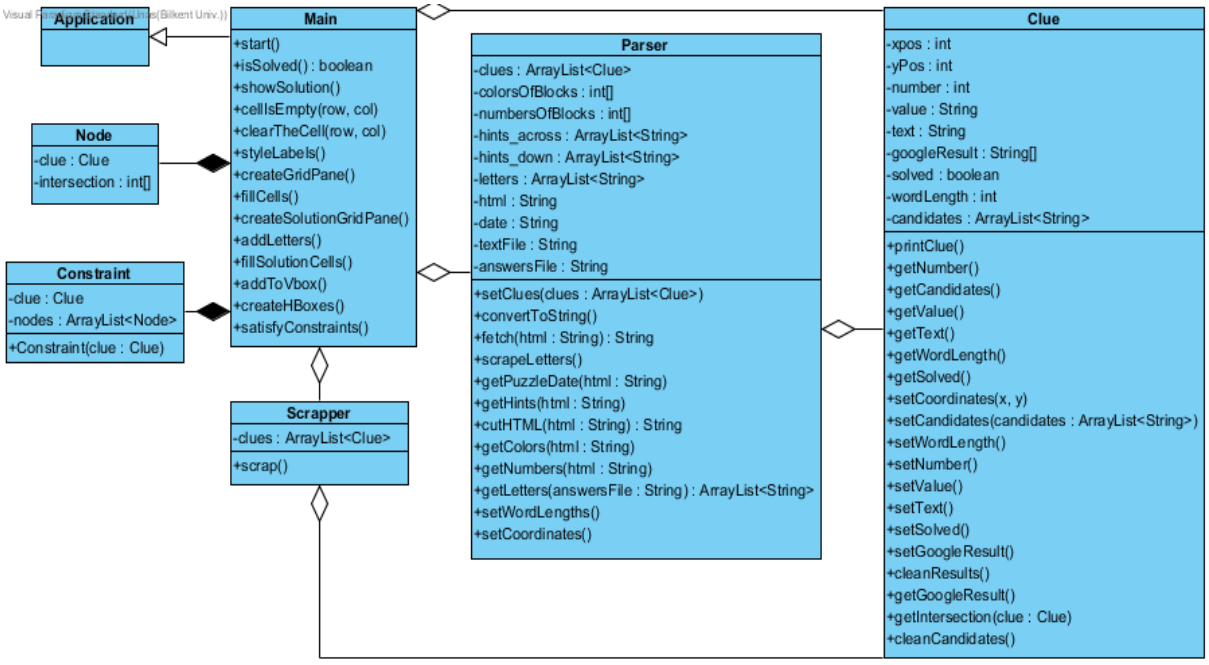

Figure 6: Class Diagram

5.1 Class Parser

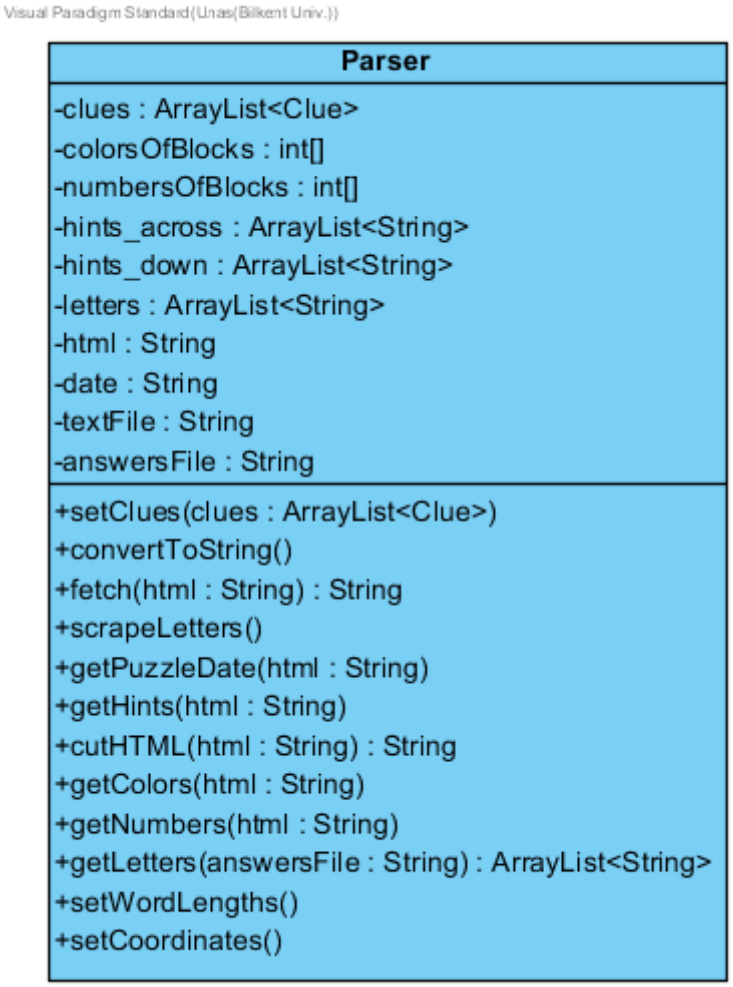

Figure 7: Parser Class Diagram

Description

Parser class was designed first and foremost. To begin working on the project, information was required from the New York Times website about the puzzle. This is where the Parser Class steps in.
The main purpose of the Class is to go to the nytimes website and download all the html code of the mini crossword page and store it in a String. Afterwards, this html String is parsed and the required data is extracted which is used to set-up the puzzle in question.

Attributes

- clues : ArrayList<Clue>

An ArrayList of Clue objects. Each hint in the puzzle corresponds to a single Clue object. These objects are explained in detail in the Clue Class description.

- colorsOfBlocks : int[]
  This integer array holds 25 integer values. Each corresponding to a single cell of the puzzle grid. Each of these value will be either 0 or 1, which will map to the grid cell being a white cell or a black cell.

- numbersOfBlocks : int[]
  This attribute is very similar to the colorsOfBlocks array explained above. The only difference is that numberOfBlocks stores the number that a particular block would have written on it signalling the corresponding clue.

- hints_across : ArrayList<String>
  This arraylist is used to store all the across type hints as String objects.

- hints_down : ArrayList<String>
  This arraylist is used to store all the down type hints as String objects.

- Letters : ArrayList<String>
  This arraylist is used to store all the solutions of the hints as String objects.

- html : String
  This String object stores the complete html code, extracted from the webpage.

- date : String
  This object stores the publishing date of the puzzle. This is used to keep a record of old puzzles.

- textFile : String
  Stores the name of the textFile that is used to store the html data. This is done so that the puzzle can be stored and used again at a later time.

- answerFile : String
  This object is used to store the name of the file that is used to store the answers of a particular puzzle. This is also done for puzzle reuse purposes.

Operations

- fetch(html : String) : String
  This method takes a String input which is the website address to the nytimes mini crossword puzzle. This method does the major work in this class. Firstly, the method uses Jsoup library functions to connect to the webpage given as the input and then parse the html content and extract particular blocks of html data and store them in a

String which is returned from the function. This method also uses helper functions to further parse this String object to extract the information required. These functions are: getHints(), getColors(), getNumbers(), scrapeLetters(), setWordLengths() and setCoordinates. These are further explained below.

- convertToString()
  This method is an alternate to the fetch method and is invoked when the program is required to solve an older puzzle which has already been saved.

- scrapeLetters()
  This function is used to extract the solution letters from the web page html. It again opens the nytimes crossword page and uses the chrome webdriver to navigate and click the button that reveals the answers to the puzzle. Afterwards the html is parsed again and the blocks holding the answers are extracted. These answers are then stored into a text file.

- getPuzzleDate(html : String)
  This method parses the html data and extracts the puzzle date.

- getHints(html : String)
  This method parses the html data and extracts all the hints. Then for each hint a Clue object is created and stored into the clues arraylist. For each clue object, apart from the clue text, clue value (across/down) is also stored and the clue number is also stored.

- cutHTML(html : String) : String
  This method slices the htnl String and return a String that contains only the data that is concerned with the grid blocks,

- getColors(html : String)
  This method extracts the colors of the boxes of the grid from the html data. These are stored in the colorsOfBlocks attribute as explained earlier.

- getNumbers(html : String)
  This method extracts the numbers of the boxes of the grid from the html data. These are stored in the numbersOfBlocks attribute as explained earlier.

- setWordLengths()
  This method calculates word lengths of the answers. The lengths are calculated using the starting positions of the words and calculating how many blocks between the start block of the word and end of the grid or a black block.

- setCoordinates()
  This method is used to set Clue coordinates. This is done using the numbersOfBlocks array. The index of the number corresponds to the position of the

clue holding that number. Using that the x and y coordinates are calculated of the Clue and stored in the Clue object properties.

5.2 Class Scrapper

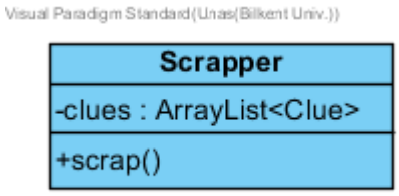

Figure 8: Scrapper Class Diagram

Description

Scrapper class scrapes the words from the websites using Selenium Library. It also stores the candidate words to the lists for each hint.

Operations

- scrap() : void
  This method scrapes the words from the websites and then stores them to each hint's list.

6. Test Cases

6.1 Monday March 5, 2018 Puzzle

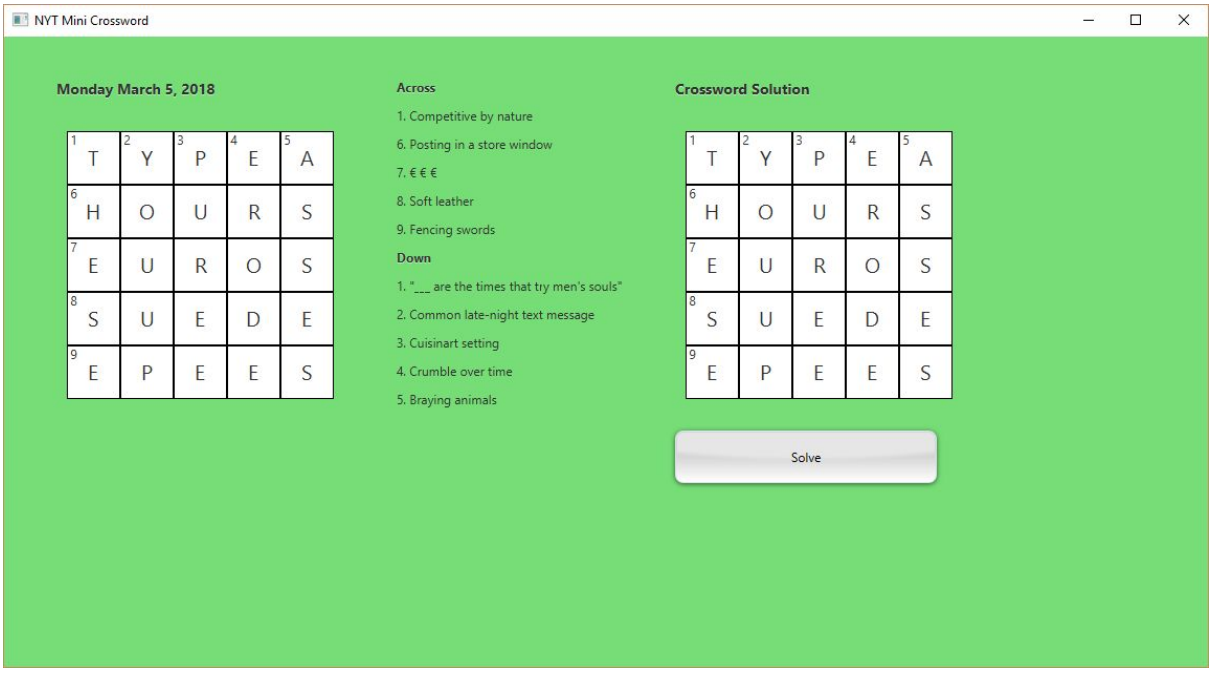

Figure 9: 5th March Puzzle

6.2 Tuesday March 6, 2018 Puzzle

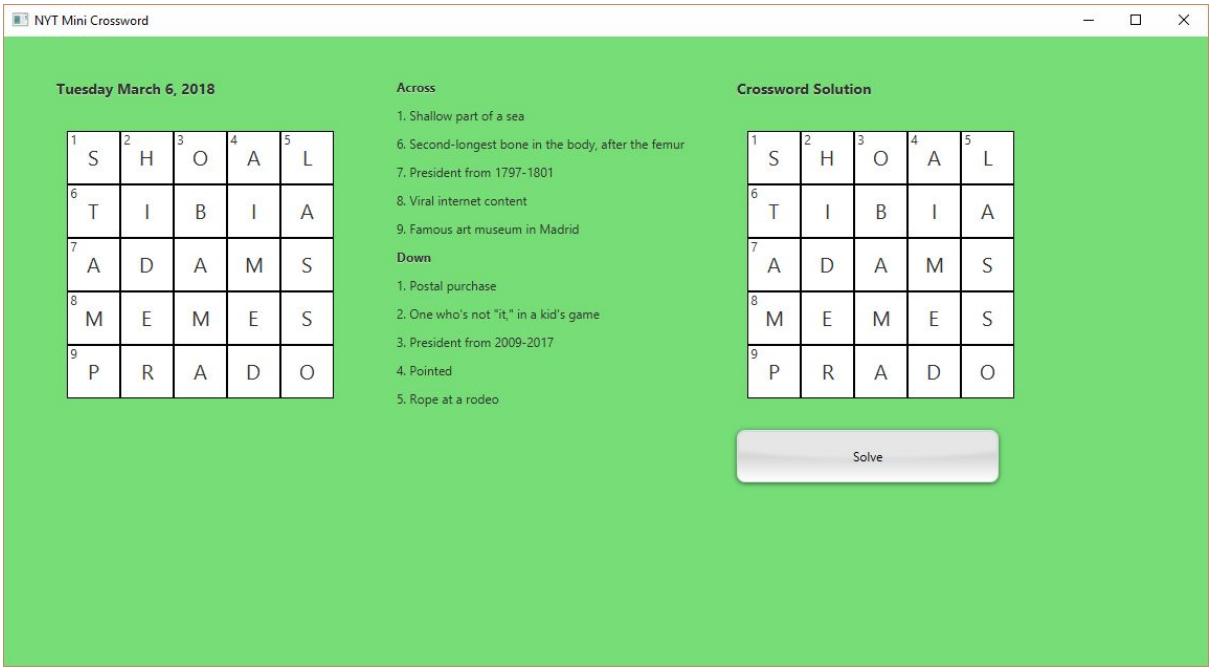

Figure 10: 6th March Puzzle

6.3 Monday April 23, 2018 Puzzle

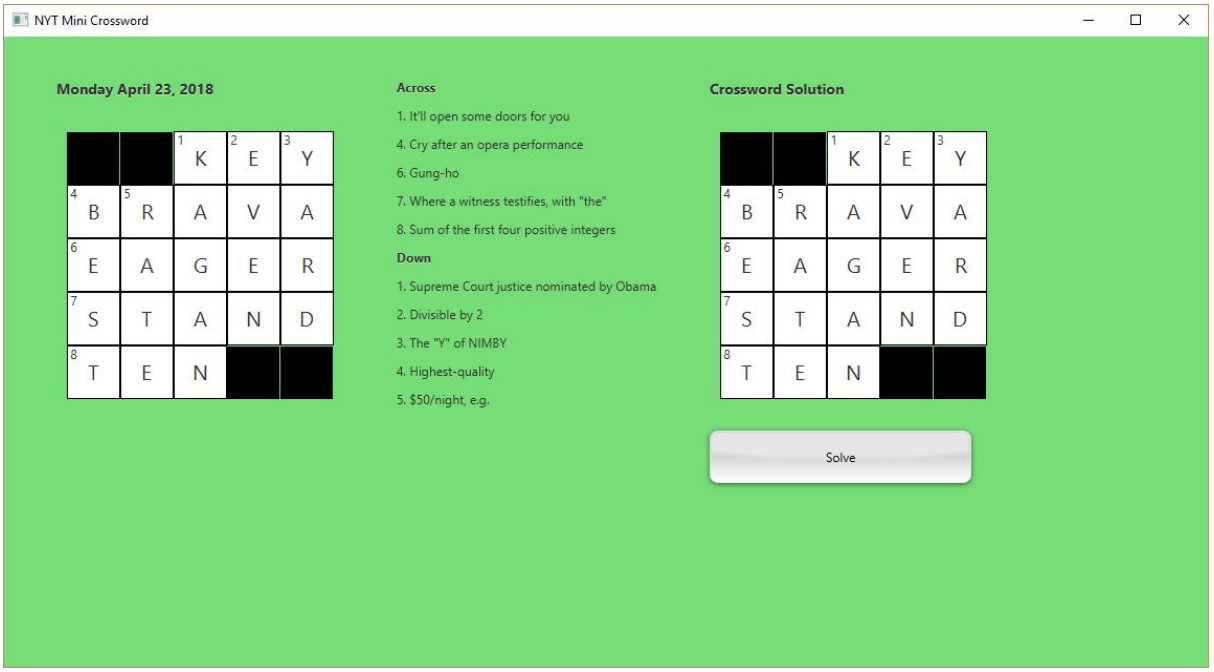

Figure 11: 23th AprilPuzzle

6.4 Tuesday April 24, 2018 Puzzle

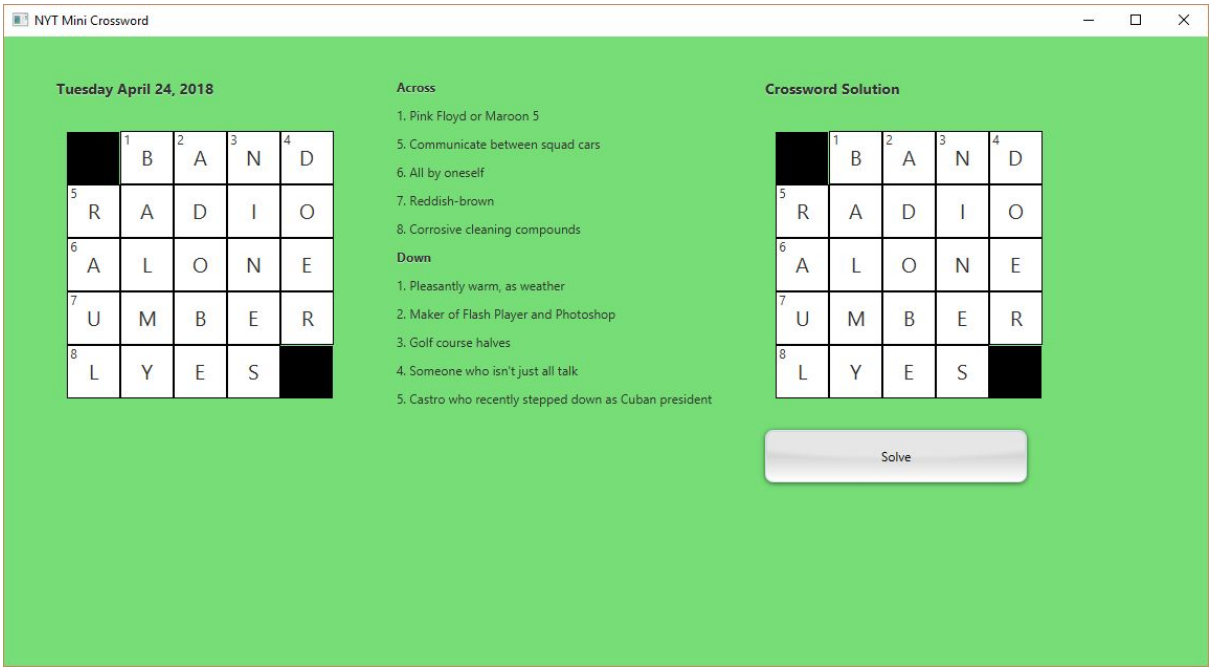

Figure 12: 24th April Puzzle

6.5 Thursday April 26, 2018 Puzzle

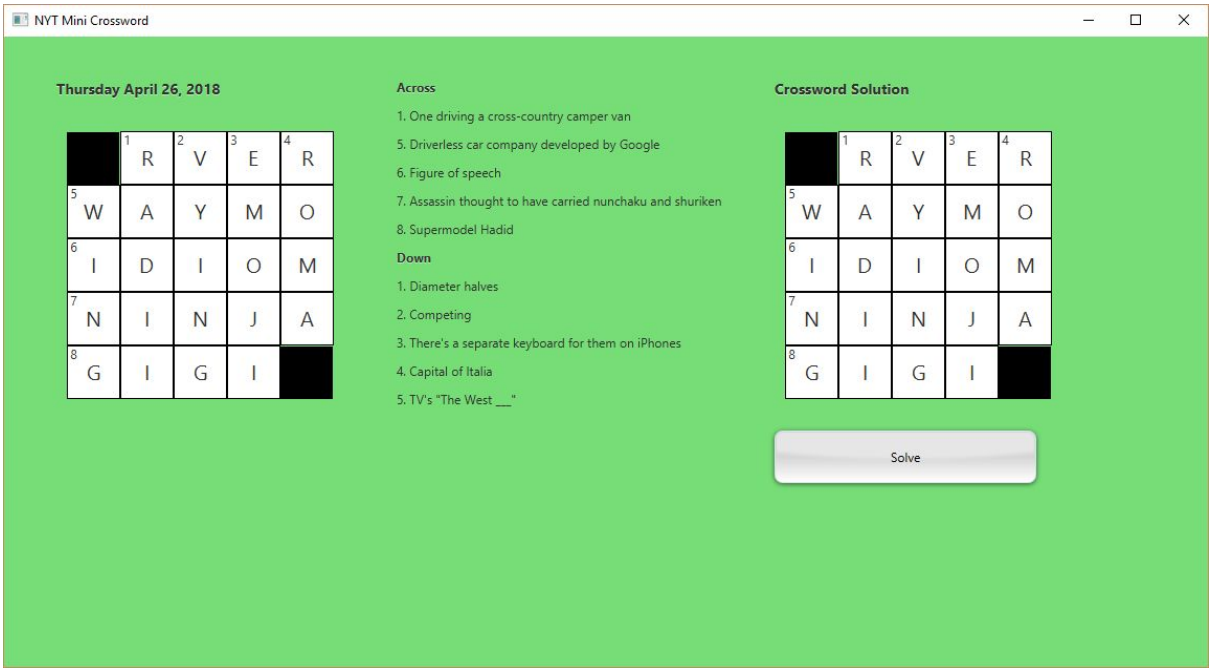

Figure 13: 26th AprilPuzzle

6.6 Thursday May 10, 2018 Puzzle

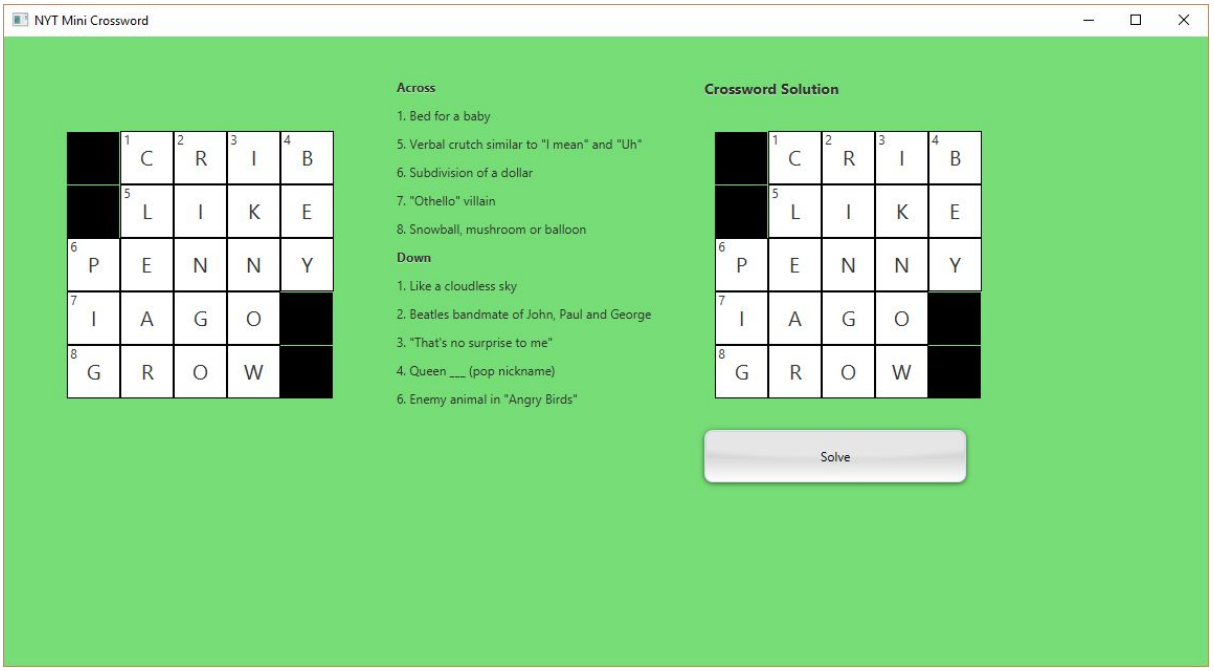

Figure 14: 10th May Puzzle

7. Conclusion

This project reports work done in partial fulfillment of the requirements for the course CS 461 - Artificial Intelligence. The software developed in this project is, to a large extent, original (with borrowed code clearly identified). It was written solely by the members of the project group and is not simultaneously being submitted to another course.

8. Appendix

8.1 Main Class

```
public class Main extends Application {

    final int WIDTH = 1150;
    final int HEIGHT = 600;

    final int CELLWIDTH = 50;
    final int CELLHEIGHT = 50;
    Stage window;
    Scene scene1, scene2;
    Button button1;
    Label date, solutionLabel;
```

```
BorderPane border;
ArrayList<String> clueList;
ArrayList<Label> labelList;
ArrayList<Rectangle> cellList, solutionCellList;
ArrayList<TextArea> textList;
Label[] numLabelList, solutionNumLabelList;
Label[] letterList;
int[] blockColors;
Parser parser;
Button solution;
Label across, down, trace;
GridPane gridPane, solutionGridPane;
VBox cluePane, crossword, solutionCrossword;
HBox mainLayout;
SolutionBox box;
Rectangle traceBox;
Scrapper scrapper;
ArrayList<Clue> clues;
public TextArea test;
public String text = "";
String answerFile = "10MayPuzzleAnswers";

ArrayList<Constraint> constraints;

public static void main(String[] args) throws IOException {
    launch(args);
}

@Override
public void start(Stage primaryStage) throws IOException {
    parser = new Parser();
    clues = parser.clues;
    scrapper = new Scrapper(clues, this);

    //scrapper.scrap();


    box = new SolutionBox();
    window = primaryStage;
    border = new BorderPane();

    StackPane layout = new StackPane();

    // add background image
```

```
        Image image = new
Image(Paths.get("C:/Users/User/Desktop/School/cs461/Project/backgrou
nd.jpg").toUri().toString(), true);
        layout.setBackground(new Background(new
BackgroundImage(image, BackgroundRepeat.NO_REPEAT,
BackgroundRepeat.NO_REPEAT, BackgroundPosition.CENTER,
BackgroundSize.DEFAULT)));

        // initialize
        solutionLabel = new Label("Crossword Solution");
        trace = new Label("Trace Box");
        letterList = new Label[25];

        for (int i = 0; i < letterList.length; i++) {
            letterList[i] = null;
        }

        constraints = new ArrayList<>();
        clueList = new ArrayList<>();
        clueList = parser.getAllHints();
        labelList = new ArrayList<>();
        cellList = new ArrayList<>();
        solutionCellList = new ArrayList<>();
        textList = new ArrayList<>();
        blockColors = parser.getColorsOfBlocks();
        numLabelList = new Label[25];
        solutionNumLabelList = new Label[25];
        test = new TextArea();
        test.setPrefHeight(180);
        test.setPrefWidth(400);
        traceBox = new Rectangle();
        traceBox.setHeight(180);
        traceBox.setWidth(400);
        traceBox.setFill(Color.WHITE);
        traceBox.setStrokeWidth(3.0);
        traceBox.setStroke(Color.BLACK);

        for (int i = 0; i < 10; i++) {
            labelList.add(new Label(clueList.get(i)));
        }

        // style labels
        styleLabels();

        // Create Cells
        fillCells();
```

```
        fillSolutionCells();

        // creating grid and adding cells
        createGridPane();
        createSolutionGridPane();

        // create HBoxes
        // adding clues to VBox
        addToVbox();
        createHBoxes();

        border.setCenter(mainLayout);
        border.setPadding(new Insets(40, 50, 10, 50));

        layout.getChildren().add(border);

        // add style
        String style =
this.getClass().getResource("style.css").toExternalForm();
        scene1 = new Scene(layout, WIDTH, HEIGHT);
        scene1.getStylesheets().add(style);

        window.setScene(scene1);
        window.setTitle("NYT Mini Crossword");
        window.show();
    }

    public boolean isSolved() {
        boolean solved = true;
        for (Clue clue : clues) {
            if (clue.getCandidates().size() > 1)
                solved = false;
        }
        return solved;
    }

    public boolean checkAcrossClues() {
        boolean passAcross = false;
        for (int i = 0; i < 5; i++) {
            if (clues.get(i).getCandidates().size() == 1 &&
clues.get(i).getCandidates().get(0) == null) {
                passAcross = true;
                break;
            }
        }
        return passAcross;
```

```
    }

    public boolean checkDownClues() {
        boolean passDown = false;
        for (int j = 5; j < clues.size(); j++) {
            if (clues.get(j).getCandidates().size() == 1 &&
clues.get(j).getCandidates().get(0) == null) {
                passDown = true;
            }
            break;
        }
        return passDown;
    }

    public void showSolution() throws IOException {
        scrapper.scrap();
        for (Clue clue : clues) {
            constraints.add(new Constraint(clue));
        }
        while (!isSolved()) {
            satisfyConstraints();
            for (Clue clue : clues) {
                clue.cleanCandidates();
                System.out.println("Candidates of " +
clue.getNumber() + ". " + clue.getValue() + " after clean up:");

System.out.println(clue.getCandidates().toString());
            }
        }


        for (int i = 0; i < clues.size(); i++) {
            if (clues.get(i).getValue().equals("Across") &&
!checkAcrossClues()) {
                int y = clues.get(i).yPos;
                int x = clues.get(i).xPos;
                Label[] first = new
Label[clues.get(i).getWordLength()];
                for (int j = 0; j < first.length; j++) {
                    first[j] = new
Label(clues.get(i).getLetters().get(j));
                    first[j].setFont(new Font(20));
                    if (cellIsEmpty(x, y)) {
                        letterList[5 * y + x] = first[j];
                        gridPane.add(letterList[5 * y + x], x, y);
```

```
                        gridPane.setHalignment(letterList[5 * y +
x], HPos.CENTER);
                        //letterList[5*y + x] = first[j];
                    } else {
                        clearTheCell(x, y);
                        letterList[5 * y + x] = first[j];
                        gridPane.add(letterList[5 * y + x], x, y);
                        gridPane.setHalignment(letterList[5 * y +
x], HPos.CENTER);
                        //letterList[5*y + x] = first[j];
                    }
                    x++;
                }
            } else if (clues.get(i).getValue().equals("Down") &&
!checkDownClues()) {
                int y = clues.get(i).yPos;
                int x = clues.get(i).xPos;
                Label[] first = new
Label[clues.get(i).getWordLength()];
                for (int j = 0; j < first.length; j++) {
                    first[j] = new
Label(clues.get(i).getLetters().get(j));
                    first[j].setFont(new Font(20));
                    if (cellIsEmpty(x, y)) {
                        letterList[5 * y + x] = first[j];
                        gridPane.add(letterList[5 * y + x], x, y);
                        gridPane.setHalignment(letterList[5 * y +
x], HPos.CENTER);

                    } else {
                        clearTheCell(x, y);
                        letterList[5 * y + x] = first[j];
                        gridPane.add(letterList[5 * y + x], x, y);
                        gridPane.setHalignment(letterList[5 * y +
x], HPos.CENTER);

                    }
                    y++;
                }
            }
        }
    }

    public boolean cellIsEmpty(int row, int col) { // works fine
        return letterList[5 * col + row] == null;
    }
```

```
    public void clearTheCell(int row, int col) {
        letterList[5 * col + row].setText(" ");
    }

    public void styleLabels() throws IOException {
        // date label
        date = new Label(parser.getPuzzleDate());
        date.setId("date-text");
        solutionLabel.setId("date-text");
        trace.setId("date-text");

        // solution button
        solution = new Button("Solve");
        solution.setId("green");
        solution.setPrefSize(250, 50);
        solution.setOnAction(e -> {
            try {
                showSolution();
            } catch (FileNotFoundException e1) {
                e1.printStackTrace();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        });
        solution.setOnMouseEntered(new EventHandler<MouseEvent>
                () {

            @Override
            public void handle(MouseEvent t) {
                solution.setStyle("-fx-background-color:#c3c4c4;");
            }
        });
        solution.setOnMouseExited(new EventHandler<MouseEvent>
                () {

            @Override
            public void handle(MouseEvent t) {
                solution.setStyle("-fx-background-color:\n" +
                        "        #dae7f3,\n" +
                        "        linear-gradient(#d6d6d6 50%, white
100%),\n" +
                        "        radial-gradient(center 50% -40%,
radius 200%, #e6e6e6 45%, rgba(230,230,230,0) 50%);");
            }
        });
```

```
        // across and down labels
        across = new Label("Across");
        across.setId("across_and_down");
        down = new Label("Down");
        down.setId("across_and_down");
    }

    public void createGridPane() {
        gridPane = new GridPane();
        gridPane.setPadding(new Insets(10, 10, 10, 10));
        gridPane.setVgap(0);
        gridPane.setHgap(0);

        int row = -1, col = 0;
        for (int i = 0; i < 25; i++) {
            if (col % 5 == 0) {
                col = 0;
                row += 1;
            }
            GridPane.setConstraints(cellList.get(i), col, row);
            col += 1;
        }

        for (Rectangle rect : cellList) {
            gridPane.getChildren().add(rect);
        }

        // making number labels for blocks
        row = -1;
        col = 0;
        for (int i = 0; i < 25; i++) {
            if (col % 5 == 0) {
                col = 0;
                row += 1;
            }
            if (parser.getNumbersOfBlocks()[i] != 0) {
                numLabelList[i] = new Label(" " +
String.valueOf(parser.getNumbersOfBlocks()[i]));
                GridPane.setConstraints(numLabelList[i], col, row);
                GridPane.setValignment(numLabelList[i], VPos.TOP);
            } else {
                numLabelList[i] = null;
            }
            col += 1;
        }
```

```
        for (int i = 0; i < 25; i++) {
            if (numLabelList[i] != null) {
                gridPane.getChildren().add(numLabelList[i]);
            }
        }

        row = -1;
        col = 0;
        for (int i = 0; i < 25; i++) {
            if (col % 5 == 0) {
                col = 0;
                row += 1;
            }
            col += 1;
        }

        for (int i = 0; i < 25; i++) {
            if (cellList.get(i).getFill() != Color.BLACK) {
                //gridPane.getChildren().add(textList.get(i));
            }
        }
    }

    public void fillCells() {
        int i = 0;
        for (int value : blockColors) {
            // white cells
            if (value == 0) {
                Rectangle temp1 = new Rectangle(CELLWIDTH,
CELLHEIGHT);
                temp1.setFill(Color.WHITE);
                temp1.setStroke(Color.BLACK);
                temp1.setStrokeWidth(1);
                cellList.add(temp1);

            }
            // black cells
            else {
                Rectangle temp2 = new Rectangle(CELLWIDTH,
CELLHEIGHT);
                temp2.setFill(Color.BLACK);
                cellList.add(temp2);

            }
            i += 1;
```

```
        }
    }

    public void createSolutionGridPane() throws IOException {
        solutionGridPane = new GridPane();
        solutionGridPane.setPadding(new Insets(10, 10, 10, 10));
        solutionGridPane.setVgap(0);
        solutionGridPane.setHgap(0);

        int row = -1, col = 0;
        for (int i = 0; i < 25; i++) {
            if (col % 5 == 0) {
                col = 0;
                row += 1;
            }
            GridPane.setConstraints(solutionCellList.get(i), col,
row);
            col += 1;
        }

        for (Rectangle rect : solutionCellList) {
            solutionGridPane.getChildren().add(rect);
        }

        // making number labels for blocks
        row = -1;
        col = 0;
        for (int i = 0; i < 25; i++) {
            if (col % 5 == 0) {
                col = 0;
                row += 1;
            }
            if (parser.getNumbersOfBlocks()[i] != 0) {
                solutionNumLabelList[i] = new Label(" " +
String.valueOf(parser.getNumbersOfBlocks()[i]));
                GridPane.setConstraints(solutionNumLabelList[i],
col, row);
                GridPane.setValignment(solutionNumLabelList[i],
VPos.TOP);
            } else {
                solutionNumLabelList[i] = null;
            }
            col += 1;
        }
        for (int i = 0; i < 25; i++) {
            if (solutionNumLabelList[i] != null) {
```

```
solutionGridPane.getChildren().add(solutionNumLabelList[i]);
            }
        }

        addLetters();
    }

    public void addLetters() throws IOException {
        Label[] chars = new Label[25];
        int row = -1, col = 0;
        for (int i = 0; i < 25; i++) {
            if (col % 5 == 0) {
                col = 0;
                row += 1;
            }
            if (!parser.getLetters().get(i).equals(" ")) {
                chars[i] = new
Label(parser.getLetters(answerFile).get(i));
                chars[i].setFont(new Font(20));
                GridPane.setConstraints(chars[i], col, row);
                GridPane.setHalignment(chars[i], HPos.CENTER);
            } else {
                chars[i] = null;
            }
            col += 1;
        }

        for (int i = 0; i < 25; i++) {
            if (chars[i] != null) {
                solutionGridPane.getChildren().add(chars[i]);
            }
        }
    }

    public void fillSolutionCells() {
        int i = 0;
        for (int value : blockColors) {
            // white cells
            if (value == 0) {
                Rectangle temp1 = new Rectangle(CELLWIDTH,
CELLHEIGHT);
                temp1.setFill(Color.WHITE);
                temp1.setStroke(Color.BLACK);
                temp1.setStrokeWidth(1);
                solutionCellList.add(temp1);
```

```
            }
            // black cells
            else {
                Rectangle temp2 = new Rectangle(CELLWIDTH,
CELLHEIGHT);
                temp2.setFill(Color.BLACK);
                solutionCellList.add(temp2);
            }
            i += 1;
        }
    }

    public void addToVbox() {
        cluePane = new VBox(10);
        cluePane.getChildren().add(across);

        for (int i = 0; i < 10; i++) {
            if (i == 5)
                cluePane.getChildren().add(down);
            cluePane.getChildren().add(labelList.get(i));
        }

        crossword = new VBox(20);
        crossword.getChildren().addAll(date, gridPane);

        solutionCrossword = new VBox(20);
        solutionCrossword.getChildren().addAll(solutionLabel,
solutionGridPane, solution);
    }

    public void createHBoxes() {
        mainLayout = new HBox();
        mainLayout.getChildren().addAll(crossword, cluePane,
solutionCrossword);
        mainLayout.setSpacing(50);
    }

    public void satisfyConstraints() {
        for (int c = 0; c < constraints.size(); c++) {
            Clue clue1 = constraints.get(c).clue;
            for (int n = 0; n < constraints.get(c).nodes.size();
n++) {
                Node node = constraints.get(c).nodes.get(n);
                Clue clue2 = node.clue;
                int[] intersect = node.intersection;
                if (clue1.getValue().equals("Across")) {
```

```
                    System.out.println("intersect[0]: " +
intersect[0] + " intersect[1]: " + intersect[1]);
                    int index1 = intersect[0] - clue1.xPos; // clue1
char position
                    int index2 = intersect[1] - clue2.yPos; // clue2
char position
                    for (int cand1 = 0; cand1 <
clue1.getCandidates().size(); cand1++) {
                        boolean eliminate = true;
                        String candidate1 =
clue1.getCandidates().get(cand1);
                        if (candidate1 == null)
                            continue;
                        for (int cand2 = 0; cand2 <
clue2.getCandidates().size(); cand2++) {
                            String candidate2 =
clue2.getCandidates().get(cand2);
                            if (candidate2 != null) {
                                if (candidate1.charAt(index1) ==
candidate2.charAt(index2)) {
                                    eliminate = false;
                                }
                            }
                        }
                        if (eliminate) {
                            clue1.getCandidates().set(cand1, null);
                        }
                    }
                } else {
                    // clue1 is down
                    int index1 = intersect[1] - clue1.yPos;
                    int index2 = intersect[0] - clue2.xPos;
                    for (int cand1 = 0; cand1 <
clue1.getCandidates().size(); cand1++) {
                        boolean eliminate = true;
                        String candidate1 =
clue1.getCandidates().get(cand1);
                        if (candidate1 == null)
                            continue;
                        for (int cand2 = 0; cand2 <
clue2.getCandidates().size(); cand2++) {
                            String candidate2 =
clue2.getCandidates().get(cand2);
                            if (candidate2 != null) {
                                if (candidate1.charAt(index1) ==
candidate2.charAt(index2)) {
```

```
                                    eliminate = false;
                                }
                            }
                        }
                        if (eliminate) {
                            clue1.getCandidates().set(cand1, null);
                        }
                    }
                }
            }
        }
    }

    private class Node {
        Clue clue;
        int[] intersection;
    }

    private class Constraint {
        Clue clue;
        ArrayList<Node> nodes;

        public Constraint(Clue clue) {
            this.clue = clue;
            nodes = new ArrayList<>();
            for (int i = 0; i < clues.size(); i++) {
                Clue temp = clues.get(i);
                int[] intersection = clue.getIntersection(temp);
                if (intersection != null) {
                    Node node = new Node();
                    node.clue = temp;
                    node.intersection = intersection;
                    nodes.add(node);
                }
            }
        }

    }
}
```

8.2 Parser Class

```
public class Parser {

    // variables
    ArrayList<Clue> clues;
    ArrayList<String> hints;
```

```
    int[] colorsOfBlocks;
    int[] numbersOfBlocks;
    ArrayList<String> hints_across;
    ArrayList<String> hints_down;
    ArrayList<String> letters;

    String html;
    String date;
    String textFile = "10MayPuzzleCode";
    String answersFile = "10MayPuzzleAnswers";

    public Parser() throws IOException {
        // initialization
        clues = new ArrayList<>();
        hints = new ArrayList<String>();
        hints_across = new ArrayList<String>();
        hints_down = new ArrayList<String>();
        colorsOfBlocks = new int[25];
        numbersOfBlocks = new int[25];
        letters = new ArrayList<String>();

        // use for today's puzzle
        html = fetch("https://www.nytimes.com/crosswords/game/mini");

        // use for old puzzles
        //convertToString();

    }

    // getters and setters
    public void setHTMLString(String html) {
        this.html = html;
    }
    public String getHTMLString() {
        return html;
    }
    public int[] getNumbersOfBlocks() {
        return numbersOfBlocks;
    }
    public int[] getColorsOfBlocks() {
        return colorsOfBlocks;
    }
    public ArrayList<Clue> getClues() { return clues; }
    public void setClues(ArrayList<Clue> clues) { this.clues = clues;
}
    public ArrayList<String> getAllHints() {
```

```
        return hints;
    }
    public ArrayList<String> getLetters() {
        return letters;
    }
    public ArrayList<String> getAcrossHints() {
        for( int i = 0; i < 5; i++) {
            hints_across.add( hints.get(i));
        }
        for( int i = 0; i < 5; i++) {
            System.out.println(hints_across.get(i));
        }
        return hints_across;
    }
    public ArrayList<String> getDownHints() {
        for( int i = 5; i < 10; i++) {
            hints_down.add( hints.get(i));
        }
        for( int i = 0; i < 5; i++) {
            System.out.println(hints_down.get(i));
        }
        return hints_across;
    }
    public void setPuzzleDate( String date) {
        this.date = date;
    }
    public String getPuzzleDate() {
        return date;
    }

    // for using old puzzles
    public void convertToString() throws IOException {
        String entireFileText = new Scanner(new File(textFile +
".txt")).useDelimiter("\\A").next();
        setHTMLString(entireFileText);
        getHints(entireFileText);
        getColors(entireFileText);
        getNumbers(entireFileText);
        getLetters(answersFile);
        getPuzzleDate( entireFileText);
        setWordLengths();
        setCoordinates();
    }
```

```
    // this method fetchs the html code from the website and writes
it to the textfile (FOR TODAY)
    public String fetch(String html) {
        try {
            Document doc = Jsoup.connect(html).get();
            String htmlDocument = doc.toString();
            Elements links = doc.select("link");
            Elements scripts = doc.select("script");
            for (Element element : links) {
                htmlDocument += element.absUrl("href");
            }
            for (Element element : scripts) {
                htmlDocument += element.absUrl("src");
            }

            // save html code in the text file
            String path =
"C:/Users/User/Desktop/School/cs461/Project/" + textFile + ".txt";
            Files.write(Paths.get(path), htmlDocument.getBytes(),
StandardOpenOption.CREATE);

            getHints(htmlDocument);
            getColors(htmlDocument);
            getNumbers(htmlDocument);
            scrapeLetters(); // gets the solution and adds it into
the letters arrayList
            setWordLengths();
            setCoordinates();

            return htmlDocument;

        } catch (IOException e) {
            e.printStackTrace();
            return "";
        }
    }

    // method gets the solution of today's puzzle
    public void scrapeLetters() throws IOException {
        // opens crhome
        System.setProperty("webdriver.chrome.driver",
"C:/Users/User/Desktop/School/chromedriver.exe");
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.nytimes.com/crosswords/game/mini");
// done
```

```
        // presses ok button

driver.findElement(By.xpath("//*[@id=\"root\"]/div/div/div[3]/div/ma
in/div[2]/div/div[2]/div[2]/article/div[2]/button/div/span")).click(
); // done

        // presses reveal button

driver.findElement(By.xpath("//*[@id=\"root\"]/div/div/div[3]/div/ma
in/div[2]/div/div/ul/div[1]/li[2]/button")).click(); // done

        // presses puzzle button

driver.findElement(By.xpath("//*[@id=\"root\"]/div/div/div[3]/div/ma
in/div[2]/div/div/ul/div[1]/li[2]/ul/li[3]/a")).click(); // done

        // presses second reveal button

driver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div[2]/art
icle/div[2]/button[2]/div/span")).click(); // done

        // presses x button

driver.findElement(By.xpath("//*[@id=\"root\"]/div/div[2]/div[2]/div
/a")).click(); // done

        // copy solutions
        List<WebElement> allLinks =
driver.findElements(By.tagName("text"));
        System.out.println("Links count is: "+ allLinks.size());
        String answers = "";
        String path = "C:/Users/User/Desktop/School/cs461/Project/" +
answersFile + ".txt";
        for(WebElement link : allLinks) {
            if( !link.getText().matches("-?\\d+")) {
                answers += link.getText();
                answers += "\n";
            }
        }

        // save answers in the text file
        Files.write(Paths.get(path), answers.getBytes(),
StandardOpenOption.CREATE);
        getLetters(answersFile);

    }
```

```
    public void getPuzzleDate( String html) {
        Document document = Jsoup.parse( html);
        Elements elements =
document.getElementsByClass("PuzzleDetails-date--1HNzj");
        setPuzzleDate( elements.text());
    }

    public void getHints(String html) {
        Document document = Jsoup.parse(html);
        Elements numbers =
document.getElementsByClass("Clue-label--2IdMY");
        Elements clues =
document.getElementsByClass("Clue-text--3lzl7");

        for (int i = 0; i < numbers.size() && i < clues.size(); i++)
{
            hints.add(numbers.get(i).text() + ". " +
clues.get(i).text());
            System.out.println(numbers.get(i).text() + ". " +
clues.get(i).text());
            if(i < 5) {
                Clue temp = new Clue("Across", clues.get(i).text());

temp.setNumber(Integer.parseInt(numbers.get(i).text()));
                this.clues.add(temp);
            }
            else{
                Clue temp = new Clue("Down", clues.get(i).text());

temp.setNumber(Integer.parseInt(numbers.get(i).text()));
                this.clues.add(temp);
            }
        }
    }

    // extract the grid part from the text file
    public String cutHTML(String html) {
        String startIndex = "<g data-group=\"cells\"";
        String endIndex = "<g data-group=\"grid\"";

        int start = html.indexOf(startIndex) + startIndex.length();
        int end = html.indexOf(endIndex) - 1;

        return html.substring(start, end);
    }
```

```java
// i need to get colors for blocks
public void getColors(String html) {
    int size = 25;

    // cutting html only cell parts left
    String newHTML = cutHTML(html);

    for (int i = 0; i < size; i++) {
        int indexStart = newHTML.indexOf("class=\"") + 7;
        int indexEnd = indexStart + 16;
        String color = newHTML.substring(indexStart, indexEnd);

        if (color.equals("Cell-block--1oNa"))
            colorsOfBlocks[i] = 1; //black
        else
            colorsOfBlocks[i] = 0; //white

        newHTML = newHTML.substring(indexEnd);
    }
    for (int i = 0; i < 25; i++)
        System.out.print(colorsOfBlocks[i] + " ");
    System.out.println("end");
}

// i need to get numbers for blocks
public void getNumbers(String html) {
    int size = 25;

    String newHTML = cutHTML(html);
    Document document = Jsoup.parse(newHTML);
    Elements elements = document.getElementsByTag("g");

    for (int i = 0; i < elements.size() && i < size; i++) {
        String text = elements.get(i).text();
        if (!text.isEmpty()) {
            text = text.substring(0, 1);
            if( text.matches("-?\\d+"))
                numbersOfBlocks[i] = Integer.parseInt(text);
            else
                numbersOfBlocks[i] = 0;
        } else
            numbersOfBlocks[i] = 0;
    }
    for (int i = 0; i < 25; i++)
        System.out.print(numbersOfBlocks[i] + " ");
```

```
        System.out.println("end");
    }

    public ArrayList<String> getLetters(String answersFile) throws
IOException { // for old puzzles, scrapes the answers from the
textfile
        //String file = new Scanner(new File(answersFile +
".txt")).useDelimiter("\\A").next();
        BufferedReader reader = new BufferedReader(new
FileReader(answersFile + ".txt"));
        String line = "";
        for( int i = 0; i < 25; i++) {
            if( colorsOfBlocks[i] == 1)
                letters.add(" ");
            else {
                if( (line = reader.readLine()) != null)
                    letters.add(line);
            }
        }
        reader.close();
        return letters;
    }
    // get the number of letters for the answer to a across clue
    public void setWordLengths(){
        for(int x = 0; x < clues.size(); x++){  // for each clue in
the clues list
            int index = clues.get(x).getNumber();   // index = number
of clue
            int i = 0;
            int count = 0;
            while(numbersOfBlocks[i] != index){ // find the position
of index in the numbersOfBlocks list
                i++;
            }
            // Across Clues
            if(clues.get(x).getValue().equals("Across")) {
                do {
                    count++;
                    i++;
                }
                while (i % 5 != 0 && colorsOfBlocks[i] != 1); //
while we are in the same row and block color is not black
                clues.get(x).setWordLength(count);
            }
            // Down Clues
            else{
```

```
                do{
                    i += 5;
                    count++;
                }while( i<25 && colorsOfBlocks[i] != 1); // while
within the grid and block is not black
                clues.get(x).setWordLength(count);
            }
        }
    }

    public void setCoordinates(){
        for(int i = 0; i < clues.size(); i++){ // for each clue
            Clue temp = clues.get(i);            // get the clue
            int number = temp.getNumber();       // get the number of
clue
            int index;
            for(index = 0; index < 25; index++){
                if(numbersOfBlocks[index] == number)
                    break;
            }
            int x = index % 5;
            int y = index / 5;
            clues.get(i).setCoordinates(x, y);
        }
    }
}
```

8.3 Scrapper Class

```
public class Scrapper{
    // Properties
    ArrayList<Clue> clues;
    ArrayList<String> trace;
    Main main;

    // Constructor
    public Scrapper(ArrayList<Clue> clues, Main main)
    {
        this.clues = clues;
        this.main = main;
        trace = new ArrayList<>();
        //scrap();
    }

    // Methods
    // After running this method, every hint will contain the text of
top 5 google hits
```

```
    @SuppressWarnings("all")
    public void scrap() {
        System.out.println("Lets Google The Hints!");
        System.setProperty("webdriver.chrome.driver",
"C:/Users/User/Desktop/School/chromedriver.exe");
        WebDriver driver = new ChromeDriver();     // Using Firefox
        WebDriver driver2 = new ChromeDriver();




        // Process each hint
        for(int i = 0; i < clues.size(); i++)
        {
            ArrayList<String> googlePages  = new ArrayList<String>();
            System.out.println("Googling Hint Number " + (i+1));
            String[] googleResult = new String[3]; // Change this
size to change the number of hits to be visited for each hint
            driver.get("http://www.google.com");
            WebElement element = driver.findElement(By.name("q"));
            CharSequence searchQuery = clues.get(i).getText() + "
crossword clue\n";
            element.sendKeys(searchQuery);
            (new WebDriverWait(driver,
10)).until(ExpectedConditions.presenceOfElementLocated(By.id("result
Stats")));
            List<WebElement> searchResults =
driver.findElements(By.xpath("//*[@id='rso']//h3/a"));

            // these are all the links you want to visit
            for (WebElement webElement : searchResults)
            {
                googlePages.add(webElement.getAttribute("href"));
            }
            // visit those links
            for (int j = 0; j < googleResult.length; j++)
            {
                // Might not have sufficient hits or unusable hits
                try
                {
                    driver2.get(googlePages.get(j));
                    (new WebDriverWait(driver,
10)).until(ExpectedConditions.presenceOfElementLocated(By.tagName("b
ody")));
                    WebElement text =
driver2.findElement(By.tagName("body"));
```

```
                    if( text.getText().equals("Please Sign in")) {

driver2.findElement(By.xpath("/html/body/a")).click();
                        driver2.navigate().back();
                        text =
driver2.findElement(By.tagName("body"));
                    }
                    googleResult[j] = text.getText();
                }
                catch(RuntimeException ex)
                {
                    System.out.println("Unable to retrieve data from
page " + (j+1));
                }
            }
            clues.get(i).setGoogleResult(googleResult);
            clues.get(i).cleanResults(i);
        }
    }
    public ArrayList<String> getTrace() {
        return trace;
    }
}
```

8.4 Clue Class

```
public class Clue {
    // Properties
    int xPos;
    int yPos;
    private int number;
    private String value;
    private String text;
    private String[] googleResult;
    private boolean solved;
    private int wordLength;
    private ArrayList<String> candidates;

    // Constructors
    Clue(String value, String text)
    {
        this.value = value;
        this.text = text;
        googleResult = null;
        solved = false;
        wordLength = 0;
        this.number = -1;
```

```
        xPos = -1;
        yPos = -1;
    }

    Clue()
    {
        value = "";
        text  = "";
        googleResult = null;
        solved = false;
        wordLength = 0;
        number = -1;
        xPos = -1;
        yPos = -1;
    }

    // Methods
    public void printClue(){
        System.out.println("Clue Number: " + number + "\nValue: " +
value + "\nText: " + text + "\nWordLength: " + wordLength +
"\nCoordinates: (" + xPos + "," + yPos + ")\n");
    }

    public void setCoordinates(int x, int y){
        xPos = x;
        yPos = y;
    }

    public ArrayList<String> getCandidates() {
        return candidates;
    }

    public void setCandidates(ArrayList<String> candidates) {
        this.candidates = candidates;
    }

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public String getValue()
    {
```

```
        return value;
    }

    public String getText()
    {
        return text;
    }

    public void setValue(String value)
    {
        this.value = value;
    }

    public void setText(String text)
    {
        this.text = text;
    }

    public void setGoogleResult(String[] result)
    {
        googleResult = result;
    }
    public void cleanResults( int j) {
        System.out.println("For clue number " + (j + 1));
        ArrayList<String> candids = new ArrayList<>();
        String candidate = "";
        for( int i = 0; i < googleResult.length; i++) {
            String[] line = googleResult[i].split("\\s+");
            for( int k = 0; k < line.length; k++) {
                line[k] = line[k].replaceAll("[^\\w]", "");
                line[k] = line[k].toUpperCase();
            }
            for( int m = 0; m < line.length; m++) {
                if( line[m].length() == wordLength &&
line[m].matches("[A-Za-z]+")) {
                    candids.add(line[m]);
                }
            }
        }

        // removing duplicates
        Set<String> candidsWithoutDuplicates = new
LinkedHashSet<String>(candids);
        candids.clear();
        candids.addAll(candidsWithoutDuplicates);
```

```
        //System.out.println("candids size: " + candids.size());
        setCandidates(candids);
        System.out.println("Initial candidates for hint " + (j+1));
        for( int i = 0; i < candids.size(); i++) {
            System.out.println(candids.get(i));
        }
    }

    public String[] getGoogleResult()
    {
        return googleResult;
    }

    public void setSolved(boolean value)
    {
        solved = value;
    }

    public boolean getSolved()
    {
        return solved;
    }

    public void setWordLength(int value)
    {
        wordLength = value;
    }

    public int getWordLength()
    {
        return wordLength;
    }

    public int[] getIntersection(Clue clue){
        if(this.getValue().equals(clue.getValue())){
            return null;  // no intersection
        }

        if(this.getValue() == "Across"){
            int i = this.xPos;
            if(i > clue.xPos)
                return null;
            if(this.wordLength-1 < clue.xPos){
                return null;
            }
            while(i != clue.xPos){
```

```
                i++;
            }
            int[] result = {i, yPos};
            return result;
        }
        else{            // DOWN clue
            int i = this.yPos;
            if(i > clue.yPos)
                return null;
            if(this.xPos > clue.xPos + clue.wordLength -1){
                return null;
            }
            while(i != clue.yPos){
                i++;
            }
            int[] result = {xPos, i};
            return result;
        }
    }
    public ArrayList<String> getLetters() {
        ArrayList<String> word = new ArrayList<>();
        for( int i = 0; i < candidates.get(0).length(); i++) {
            word.add(String.valueOf(candidates.get(0).charAt(i)));
        }
        return word;
    }
    public void cleanCandidates() {
        for( int i = 0; i < candidates.size(); i++) {
            if( candidates.get(i) == null)
                candidates.remove(i);
        }
    }
}
```