Final Project Report: Spell Checker

Frank Anderson

# Introduction

I developed a spell checker to read plain text files and provide the user with suggestions of correctly spelled words which are similar to the misspelled word. The program writes the chosen suggestion instead of the misspelled word to a new file.

To find misspelled words, the program reads a dictionary from a plain text file. Then, as the program iterates through the file of the user’s choice word by word, it compares each word to those in the dictionary text file. If they are present the program writes the word to the output file and continues to the next word. If the word is not in the dictionary, the program tries a possessive and plural version, and then asks the user to choose from a list of suggestions.

To create the list of suggestions, the program uses the Fuzzy Wuzzy library to compare the given word to all the words in the dictionary. Fuzzy Wuzzy produces a ratio which is a value from 0 to 100, 100 being the words are the same. I then have the program display the three most similar values to the user to choose from. Alternatively, the user can choose to write something of their own or to use the original word. I tried writing my own version of fuzzy search where I gave each word a difference score based on how many letters were different from the original word, but the method returned too many results.

I tried to filter options where the only different letters were only those which were close to the letter which was originally tried (for example: “sord” and “qord” but not “bord” would be valid replacements for “word” because “s” and “q” are adjacent to “w” but “b” is not) but this limited the results too much. I also noticed many errors were based on not knowing how to spell words, not just typing the wrong letters. Hence, I decided to use Fuzzy Wuzzy since its results were so much more effective and I was running low on time.

The results are entirely dependent on the dictionary used, so I spent a lot of time at the start of the project testing paragraphs from a variety of sources on a variety of dictionaries I found. After testing, I decided it would be best to combine dictionaries to be able to search more words. I used the “words.txt” list from <https://github.com/dwyl/english-words> and “contractions.txt” from <https://gist.github.com/J3RN/ed7b420a6ea1d5bd6d06>. The results also depend on how many other words are more similar to what you were trying to type, so more obscure words receive more accurate suggestions because very few words are similar to them.

# Application Design

The system is comprised of a main class, GrammarChecker, and two helper classes, Dictionary and FileManager.

GrammarChecker is the class that iterates over each word in the file and controls the logic for what to do if words are spelled correctly or not, and what to do when the user wants to use a different word. It contains an instance of the FileManager and the Dictionary. Most of this logic is handled in the initializer because it makes using the checker very user friendly; all the user must do is initialize a new GrammarChecker instance. The GrammarChecker has helper methods to print the options found, ask the user what choice they want to select, and ask the user for a custom word.

Dictionary is the class which handles reading the dictionary files and the convenience methods for strings. It stores the list of words as an instance variable after reading them in during the initializer. The clean\_input function removes special characters from strings to make them searchable. The word\_in function checks if a given word is in the dictionary or if its plural or possessive forms are. Finally, the find\_similar function finds words similar to the given word. The GrammarChecker class has an instance of the Dictionary and accesses it as needed.

Finally, the FileManager class handles the file path for the file to read and write to. When the FileManager is initialized, it runs validate\_filepath where it asks the user for a file name and checks that the file path is valid. When an instance of GrammarChecker is created, an instance of FileManager is created and the path is validated before the rest of GrammarChecker can be run.

Diagram

Description automatically generated