

## **Description and motivation**

The project I developed is a spell check program, aiming to simulate the spell check functionality found in programs like Microsoft Word while adhering to Object-Oriented Programming (OOP) principles. The program utilizes Java Swing to create a window where users can input their text. As the user types, the program identifies potential spelling errors by employing a minimum-edit-distance algorithm. Upon detection, it generates a list of suggested replacement words. The program then generates buttons with these suggested words. Users can simply click on the appropriate button to replace the misspelled word in their text with the correct replacement.

## **Algorithm**

Two algorithms are employed in the project.

The first algorithm focuses on determining the validity of a word. Initially, the program incorporates a pre-existing English dictionary, from which all valid words are extracted. These words are then sorted based on their length and arranged in alphabetical order. This organization enables efficient word identification without the need to search through the entire English language. Instead, the program utilizes an indexing system that references the word's length and then its position in the alphabet, allowing for streamlined searches within the data structure.

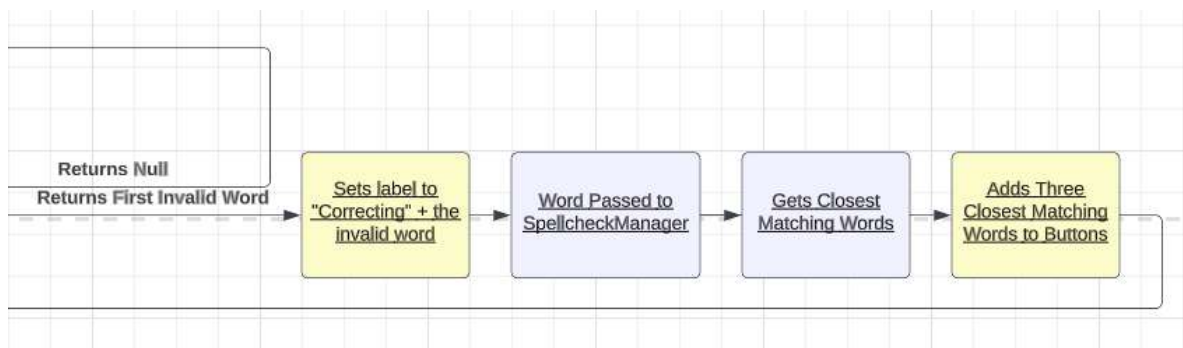
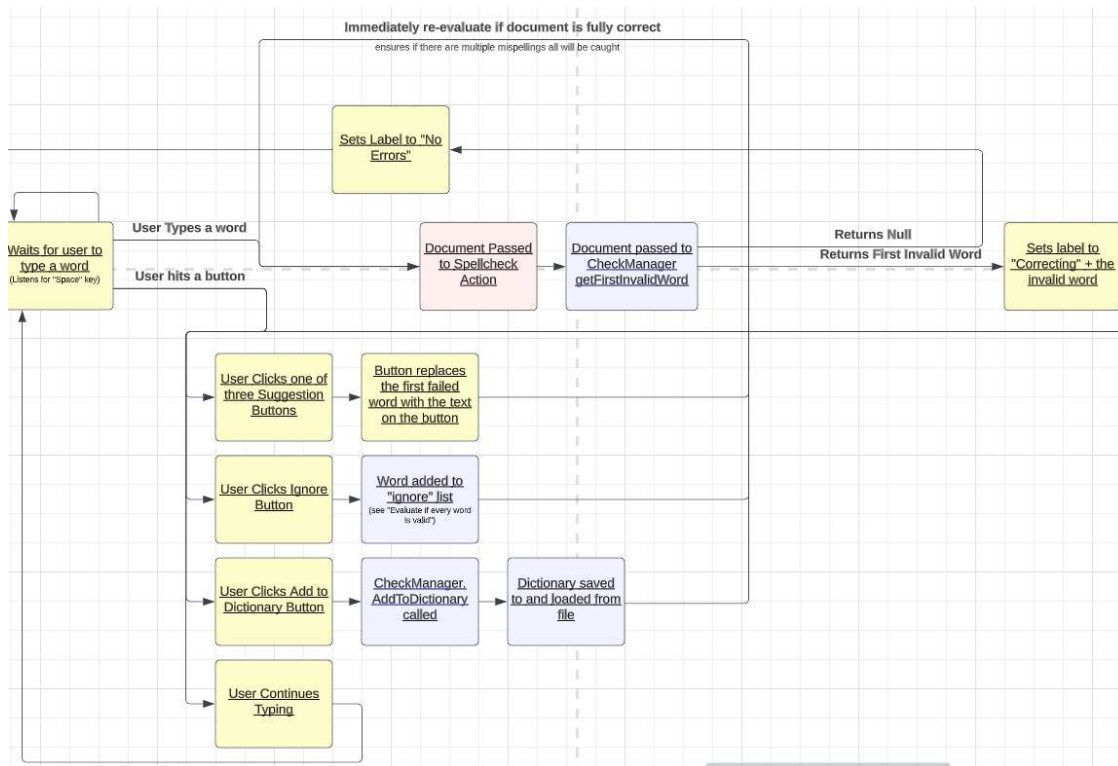
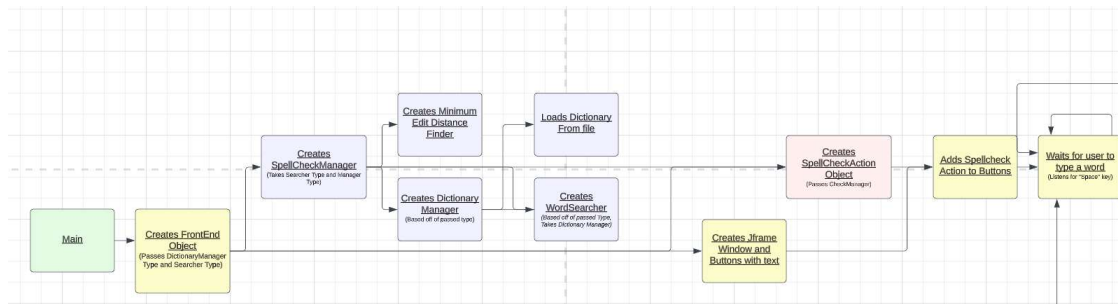
The second algorithm revolves around calculating the minimum edit distance between two words. This is accomplished through the utilization of a minimum edit distance table. The table, with dimensions  $m \times n$  (where  $n$  represents the length of word1 and  $m$  represents the length of word2), is populated by evaluating the number of changes required to transform one word into another. The table is filled following a specific rule: if letter  $i$  of word1 matches letter  $j$  of word2, cell  $(i, j)$  is set to the value of the cell located above and to the left  $(i-1, j-1)$ . Otherwise, it is set to the minimum value among the three cells located above, to the left, and diagonally left-up  $(i-1, j; i-1, j-1; i, j-1)$ , incremented by one. Ultimately, the bottom-right cell of the table contains the minimum edit distance between the two words. This approach facilitates the sorting of words based on their "proximity" to a given word.

The algorithm for generating suggested words begins by considering the length of the word being corrected and searches within an expanding scope. Initially, it identifies if there are at least three words with the same length and a minimum edit distance of 1 from the input word. If not, it examines words with lengths one greater and one smaller than the input word. If this search does not yield a minimum of three results, the algorithm proceeds to explore words that possess a minimum edit distance of 2 from the original word. This process continues until three matching words are obtained.

## **Workflow of your system (a flowchart that tells the story of starting to finishing one execution of your program)**

To view my workflow, follow this link and look for the colored diagram. It is also screenshotted below

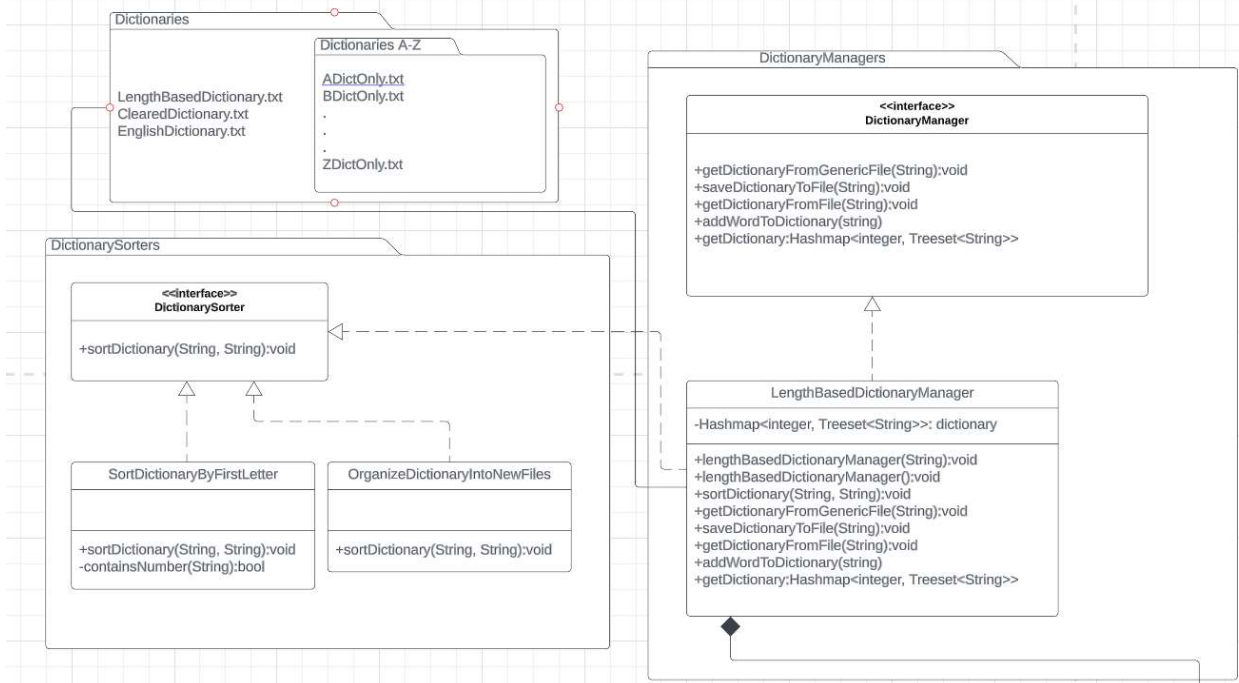
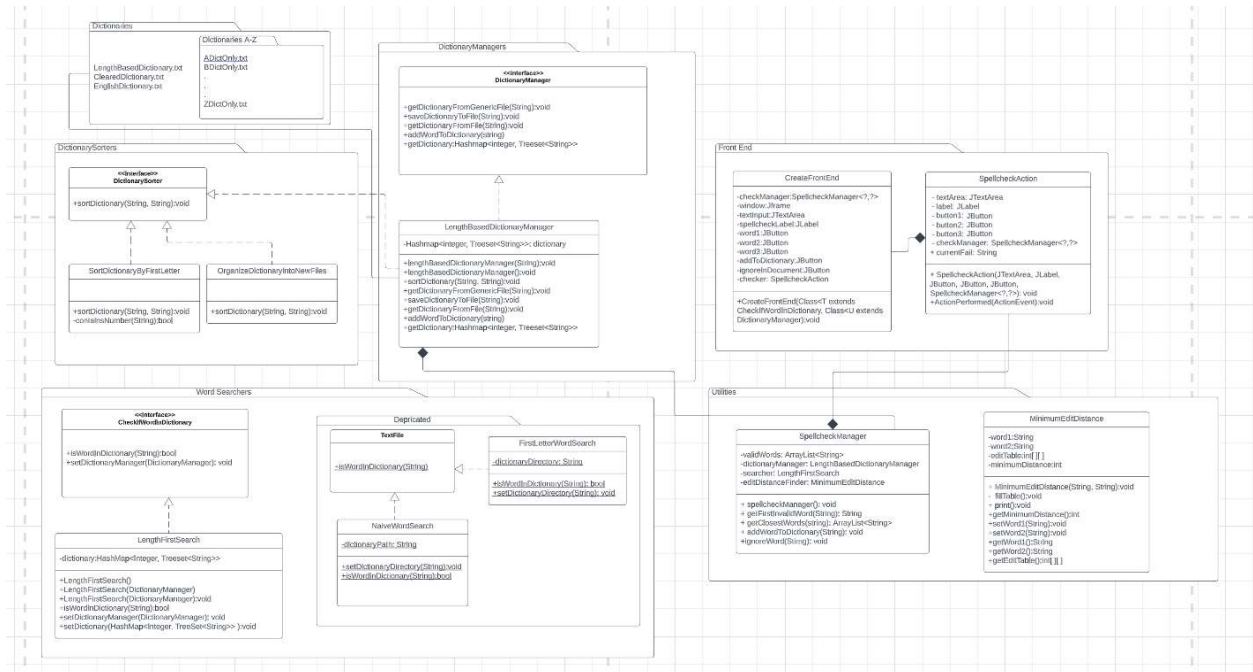
[https://lucid.app/lucidchart/170c3b23-be8a-4fd7-8dbf-a220d3a2d81d/edit?viewport\\_loc=-3022%2C204%2C6529%2C3104%2C0\\_0&invitationId=inv\\_08540b17-afe8-4489-a756-d7c6ea014e59](https://lucid.app/lucidchart/170c3b23-be8a-4fd7-8dbf-a220d3a2d81d/edit?viewport_loc=-3022%2C204%2C6529%2C3104%2C0_0&invitationId=inv_08540b17-afe8-4489-a756-d7c6ea014e59)

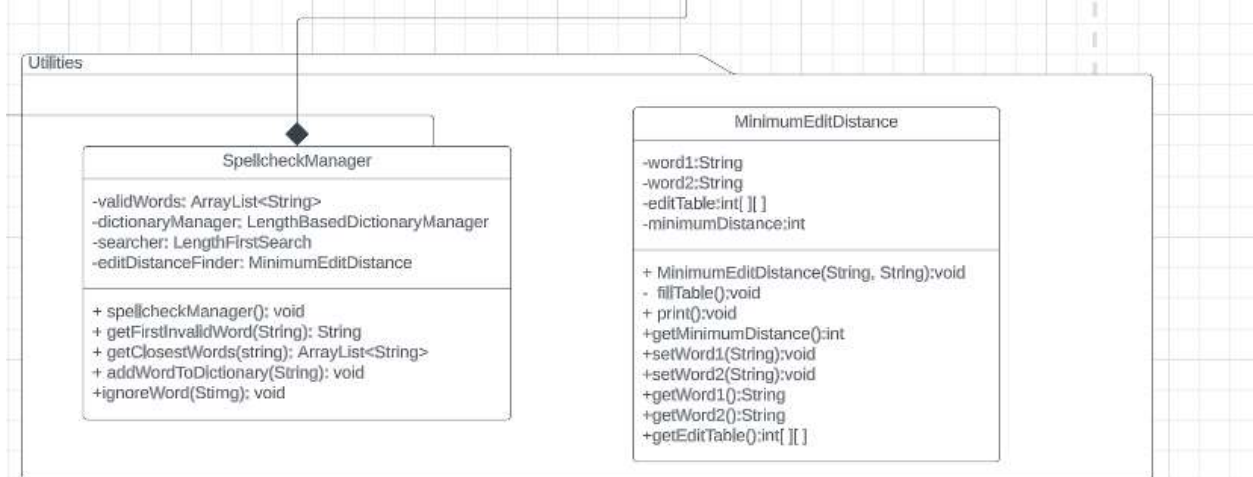
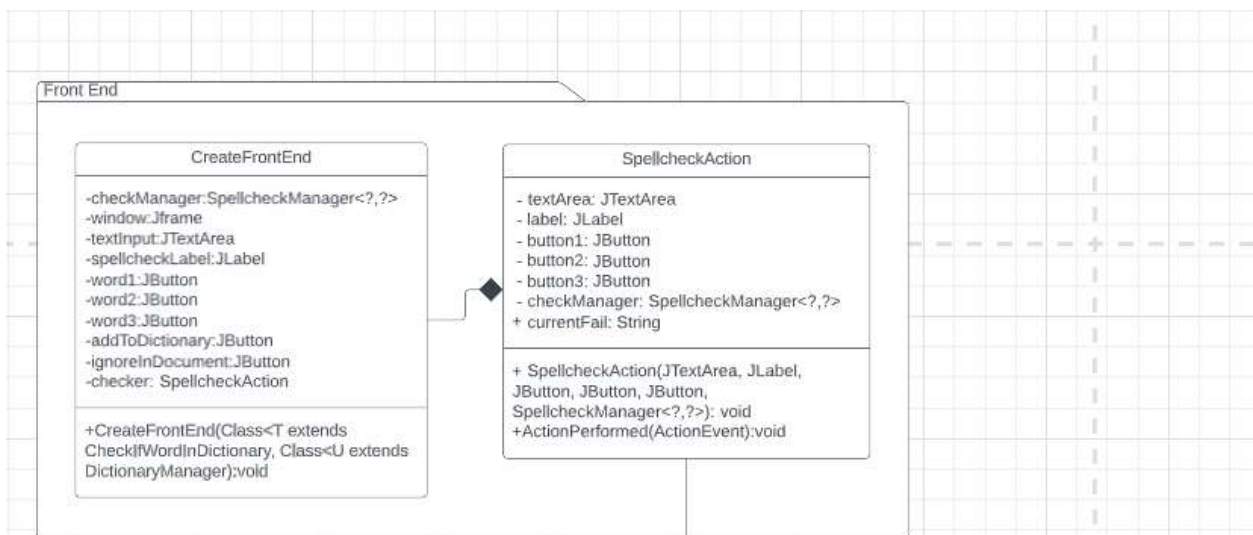
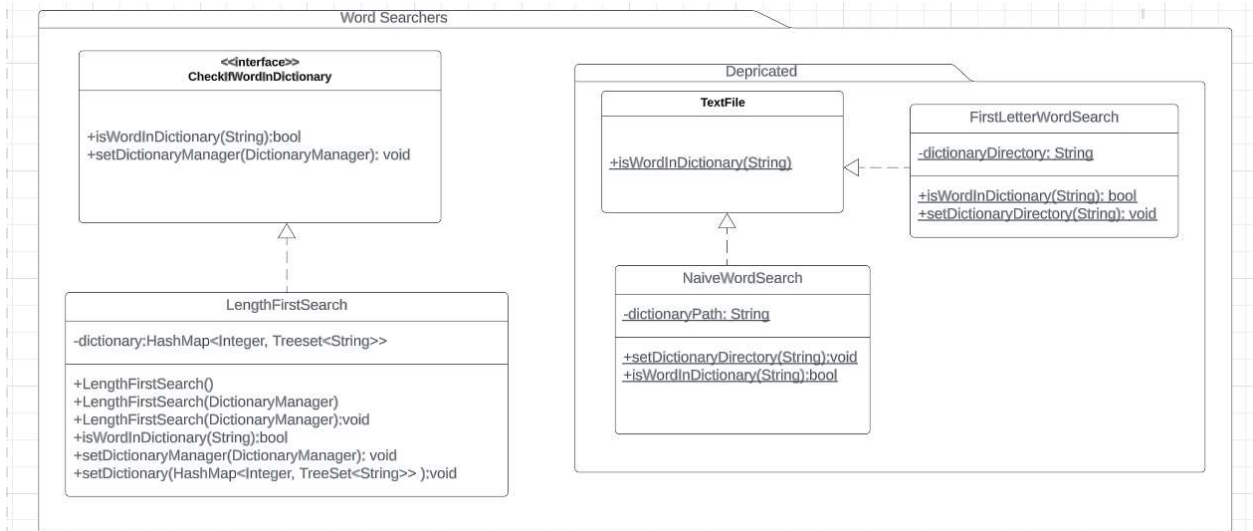


## Code and System details: UML

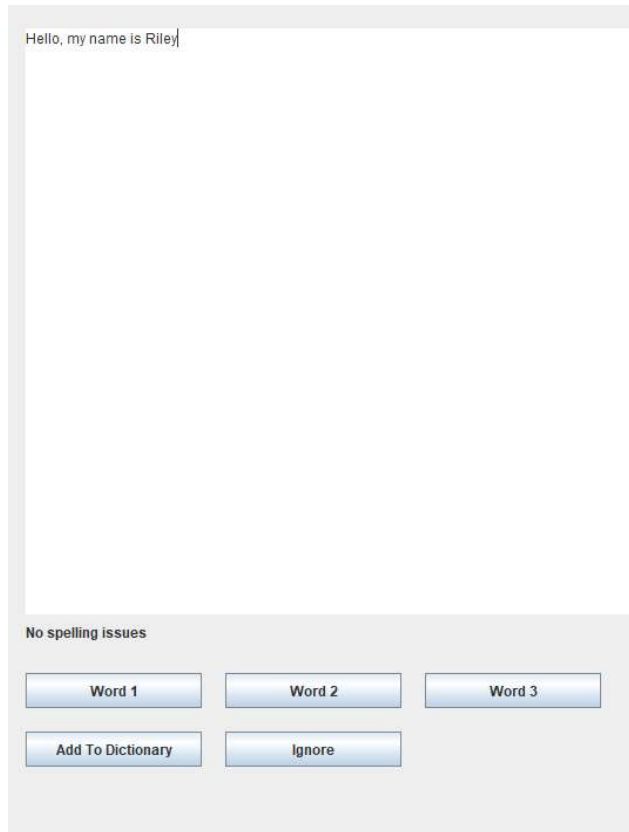
To view my UML diagram, follow the link below. It is also screenshotted below.

[https://lucid.app/lucidchart/170c3b23-be8a-4fd7-8dbf-a220d3a2d81d/edit?viewport\\_loc=-3022%2C204%2C6529%2C3104%2C0\\_0&invitationId=inv\\_08540b17-afe8-4489-a756-d7c6ea014e59](https://lucid.app/lucidchart/170c3b23-be8a-4fd7-8dbf-a220d3a2d81d/edit?viewport_loc=-3022%2C204%2C6529%2C3104%2C0_0&invitationId=inv_08540b17-afe8-4489-a756-d7c6ea014e59)



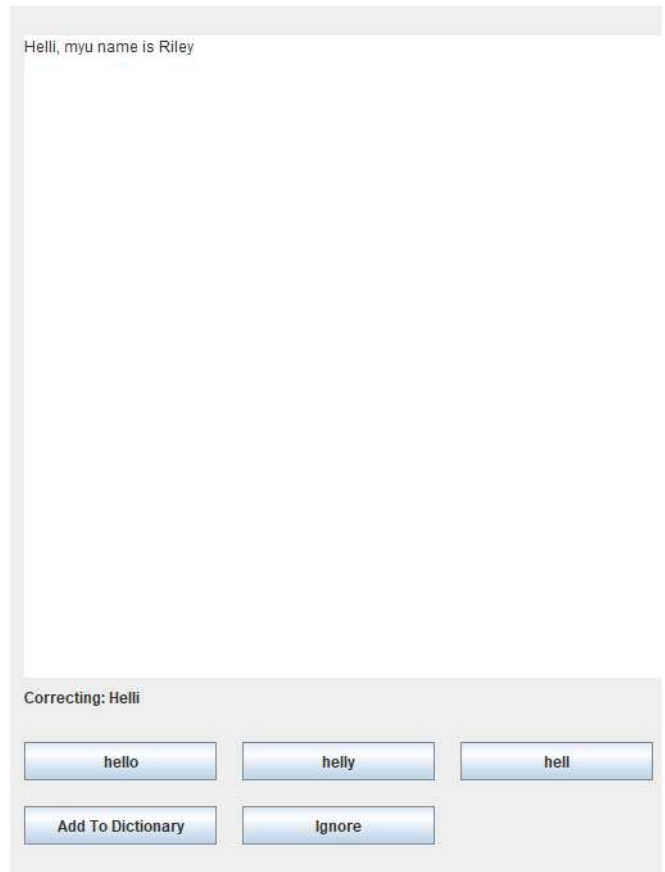


## Sample IO



Sample screenshot of the program interface after receiving correct input. The input field contains the text "Hello, my name is Riley". The output area displays "No spelling issues". Below this, there are three buttons labeled "Word 1", "Word 2", and "Word 3", and two buttons labeled "Add To Dictionary" and "Ignore".

**Program after receiving all correct input**



Sample screenshot of the program interface after detecting misspelling. The input field contains the text "Helli, myu name is Riley". The output area displays "Correcting: Helli". Below this, there are three buttons labeled "hello", "helly", and "hell", and two buttons labeled "Add To Dictionary" and "Ignore".

**Program after detecting misspelling**

## Limitations

The program has several limitations, primarily in determining word validity and providing suggestions.

One limitation relates to the initial wordset used. The program relies on a downloaded dictionary that includes uncommon words but lacks many common words, such as conjugations and verb tenses. As a result, the program may recognize "boat" as a valid word while considering "boats" as invalid. This issue extends to verbs in different tenses compared to those listed in the dictionary. Two potential solutions can address this limitation. Firstly, the program could be provided with a comprehensive list of valid English strings, encompassing more than just dictionary-worthy words. Alternatively, incorporating a Natural Language Processing (NLP) application would enable handling of verb tenses and plurals.

Another limitation pertains to the suggestions provided by the program. Currently, the program identifies words with the minimum edit distance from the user's input. However, if more than three such words are found, the program does not implement any pruning mechanism to determine which suggestions should be displayed. It simply presents the first three words it found. Consequently, this can lead to potentially relevant words not appearing in the suggestions. To mitigate this limitation, improvements can be made by sorting the suggestions

based on their commonality to ensure more accurate and relevant suggestions. Additionally, checking the part of speech of the suggested word could be implemented to ensure its compatibility with the sentence context.

The final limitation involves an oversight in the application of OOP principles. Specifically, the dictionary manager interface requires the `getDictionary` method to return a `HashMap<Integer, ArrayList<String>>` object, limiting the storage options for implementations of this interface. Addressing this limitation would involve making adjustments to the spellchecker, reducing its independence from the dictionary manager, to enable alternative data storage methods.

## **References**

[UML Class Diagram Arrows Guide. Brief guide for choosing right type of... | by Paul Romyancev | Medium](#)  
[\(137\) Minimum Edit Distance Dynamic Programming - YouTube](#)  
[Webster's Unabridged Dictionary by Various - Free Ebook \(gutenberg.org\)](#)