Spell checker

Report on the creation and effectiveness of my spell checker

Subject: AQA A-Level Computer Science (7517)

Component: NEA – Computing practical project (7517/C)

Name: Lewis Drake

Candidate number: 0631

Centre name: Barton Peveril Sixth Form College

Centre number: 58231

Due date: February 2023

Table of contents

1 Analysis

1.1 Statement of problem

The problem I aim to solve is how current spell checkers lack the ability to allow users to customise recommendations, save preferences and set character limits.

This is a problem because many names, places and acronyms are often not recognised by spell checkers, and therefore can be marked as incorrectly spelt. This means that many subject specific phrases will appear wrong when writing essays, reports or presentations for an assignment. Furthermore current spell checkers don’t allow the user to add new words to the allowed word list, this means that users are unable to save their preferences for later use.

When writing personal statements and CVs there is often a character limit and this is defined by the organisation or employer, and the text needs to be entered onto a specific webpage. However these webpages don’t have spell checkers, which means that users have no way of knowing if their text is written with correct spelling and grammar. However these webpages do have a character limit that stops the user from entering any more text when reached. I am to solve this problem by building both of these features into one application.

People who face these problems often waste time by switching applications or reapplying settings, instead of having one solution that solves all their problems.

1.2 Background

Spell checkers are programs that check for misspellings and incorrect words in a text. Spell checkers can be built into specific applications or they can be separate applications that the user enters text into. Some spell checkers also allow users to check grammar and offer recommendations on how the text can be improved.

Currently a lot of documents are written electronically using digital word processors. This means that spelling and grammar errors could be identified and corrected quickly using software.

1.3 End user

The program is being created for all users who type text and use specific acronyms, names or places. Specifically I will be aiming my program at university students as they type long essays which need to be written in formal English with correct spelling. Furthermore, many university subjects use specific words, names and acronyms which current spell checkers don’t recognise.

I have chosen Gabi as my end user who is currently at University doing a Script Writing Masters at Bournemouth University. Another reason I have chosen Gabi is because she meets my requirements of someone who will use the program:
- She often writes long essays, reports and scripts for her course.
- She spends a lot of time reapplying settings that have not been saved from previous use.
- She spends a lot of time correcting the spellings of names, places and acronyms that have been automatically changed to an incorrect word.

Gabi has also used other spell checkers before, so she knows what needs to be in the program, and what she wants in the spell checker.

1.4 Initial research

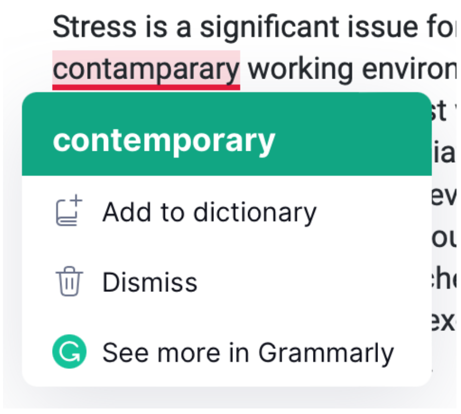
1.4.1 Existing and similar solutions

There are currently many different current solutions that come in different forms, some are separate applications or webpages, and others are built into applications as an extra feature. The main feature of all the following solutions is spell checking, and some of them have more advanced features (such as artificial intelligence to sense formality). However none of the current solutions allow for customisation by the user.

1.4.1.1 Grammarly[1]

Grammarly is an online extension that allows checks for errors in spelling and grammar. It has an intuitive user interface allowing the user to quickly view if there are any errors in their writing, and then a helpful dropdown menu that gives the user multiple options of what to do with the incorrect spelling.

Grammarly also uses artificial intelligence and uses the context of the writing to recommend alternative words which would help improve the formality or structure. This is something that I will not be able to achieve in my spell checker as Grammarly uses a complex AI algorithm to do this.


Stress is a significant issue fo
contamparary working environ
contemporary
Add to dictionary
Dismiss
See more in Grammarly

Example of Grammarly offering a recommendation on an incorrectly spelt word.

Furthermore Grammarly also offered a premium version which can offer the user more advanced recommendations, such as rephrasing sentences and enhancing word choices. It also informs the user of how their writing may come across to other readers.

However Grammarly doesn't work in text documents and is only available as a web extension on certain browsers. It also appears to be available as an API, because many online applications use Grammarly for spell checking. This means that users may face some compatibility issues when downloading and using, which can be a major problem for some users. Also, during my testing Grammarly often didn't save preferences that I changed and I had to re-apply them each use.

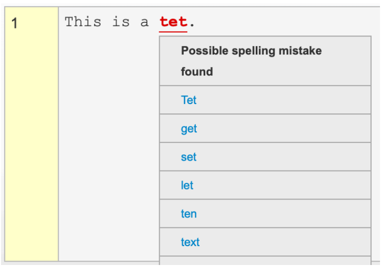
Grammarly does allow users to dismiss an incorrect spelling or add it to the dictionary, but it doesn't allow the user to view this dictionary or delete words from it. This means that words can't be removed if added on accident or were only needed for a certain document or piece of writing. The dismiss spelling is a helpful feature that I plan to implement on my program as it will allow the user to ignore the incorrect spelling, without adding it to the dictionary.

Grammarly also doesn't define incorrect spellings to users, which is something I aim to do. This means that users could be unaware when they use words that have similar sounds and spellings, but different meanings (such as "knight" and "night").

To summarise, Grammarly offers some more advanced features (such as formality and structure checking) but it does lack on some of the basic features (such as preference saving). Also, there can be some compatibility issues when attempting to use, and Grammarly doesn't have the option to define words.

1.4.1.2 Online-spellcheck[2]

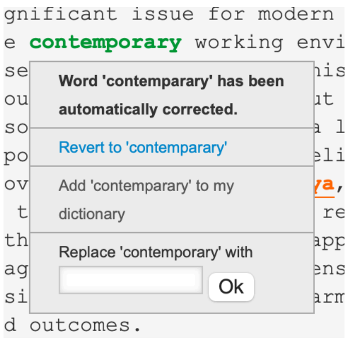
Online-spellcheck is an online application that allows users to enter text and check the spelling. The interface isn't as well built as Grammarly, but still provides functionality and allows the user to access all features. This spell checker is only available as an online application and cannot be downloaded for global use across an operating system or in its own app.



Example of online-spellcheck providing alternative words for an incorrectly spelt word.

Online-spellcheck allows users to upload a file in a variety of different formats (these include: DOC/DOCX, PDF, TXT and RTF). This is unique to this spell checker and isn't something that I feel is necessary for mine. Online-spellcheck can also identify text from images using optical character recognition (OCR), which is something that I will not be able to implement as it is too complex.

This spell checker has clear formatting for words that have been spelt incorrectly (red), automatically changed (green), changed by the user (blue) and errors unknown by the algorithm (orange). This is helpful for the user as they can easily see any errors in their work and know where to go to fix errors.



```
gnificant issue for modern
e contemporary working envi
```

Word 'contemparary' has been automatically corrected.

Revert to 'contemparary'

Add 'contemparary' to my dictionary

Replace 'contemporary' with

Ok

```
d outcomes.
```

Example of online-spellcheck automatically correcting the spelling of a word.

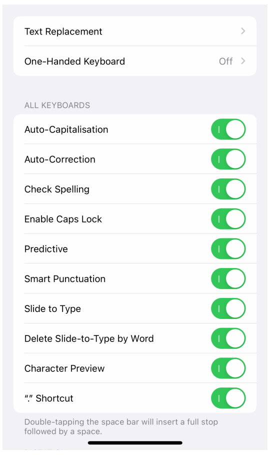
Online-spellcheck doesn't allow users to ignore/dismiss words that have been spelt incorrectly. However it does allow users to add words to the dictionary, which means that the word will be identified as correctly spelt for the rest of the document. But users can't view or remove words from the dictionary, at any time. Users can save their dictionary by creating an account and logging in. It can also automatically correct some incorrectly spelt words, if the initial spelling was close enough to the correct word.

The program doesn't allow users to change the font size, font colour or error colours.
This means that some users won't be able to use the program because they can't read the text. This is something I aim to solve with my program, by allowing users to personally the settings and be able to save them.

To summarise, online-spellcheck does allow for the user to personalise the spell checker to an extent (by allowing users add words to the dictionary), but it doesn't have as many advanced features as Grammarly (such as formality checking). Online-spellcheck also has good formatting by changing the font colour, which is something that I might use in my program.

1.4.1.3 iOS spell checker

iPhones have a built in spell checker that works globally across the device. This enables the user easy access to a spell checkers it will always be available no matter what application they are using. The spell checker is enabled by default on new devices. The spell checker also checks for grammar errors in text, and auto-capitalises letters if needed.



Text Replacement	>
One-Handed Keyboard	Off >

ALL KEYBOARDS

Auto-Capitalisation	On
Auto-Correction	On
Check Spelling	On
Enable Caps Lock	On
Predictive	On
Smart Punctuation	On
Slide to Type	On
Delete Slide-to-Type by Word	On
Character Preview	On
"." Shortcut	On

Double-tapping the space bar will insert a full stop followed by a space.

The setting page for the spell checker, showing the customisation offers for the user.

The spell checker allows for some customisation from the user from the settings app on the device. It allows the user to toggle the status of features and add text replacement. It allows the user to add words to the dictionary by "learning the spelling", however it doesn't let the user view or remove words from the dictionary.

iOS spell checker also has a "text replacement" feature. This is where users can define acronyms along with their full meaning, and replace the acronym with the full meaning when found. Such as omw, which expands to on my way. This is a useful feature that I may try to implement in my program to save the user time when writing.

iOS spell checker provides the user with simple, clear formatting when a word is spelt incorrectly, by underlining the word in a number of different colours (depending on the error). Apple also allows for customisation of accessibility features in the settings, but these are applied globally and not specific to the spell checker.

To summarise, the spell checker provides a simple, easy to use system that is implemented globally across the device. This makes it easily accessible and means that it is commonly used. It allows for some basic customisation by allowing the user to enable/disable features, and change accessibility features (such as font size). The spell checker doesn't have any advanced features like Grammarly, but does all the basics efficiently to a high standard.

1.4.1.4 Summary

To summarise, there are some current solutions that vary on levels of complexity, but all achieve the same goal of checking the selling of words and recommending alternative ones. Some pride a simpler way than other to do this.

Online-spellcheck is the least complex as it is only a web application and doesn't allow the user to customise accessibility features. It also provides a simple way of showing errors and allowing the user to correct errors. It uses a simple user interface to achieve this.

iOS spell checker uses the simplest way to present errors and changes to users. It also changes words automatically most frequently. It is slightly more complex than Online-spellcheck due to the interface, but not as complex as Grammarly due to advanced algorithms.

Grammarly is the most complex of the spell checkers as it uses algorithms to detect formality and help the user achieve different tones in their writing. It also uses a well-built interface to suggest changes and show errors.

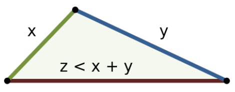
None of the spell checkers I have looked at allow the user to define words, view the dictionary or change features like the font colour/size.

1.4.2 Potential algorithms and data structures

1.4.2.1 Definitions

Triangle inequality[3]

- States that for any triangle, the sum of two sides must be greater than or equal to the length of the remaining side: (where XYZ represent sides of a triangle as per the image on the left)
  - $d(x) + d(y) \geq d(z)$
  - $x + y \geq z$
- In a triangle inequality, there is a special case if the area is equal to zero:
  - $(d(x) + d(y) = 0) \equiv (area = 0)$
- For all right angle triangles, this is a consequence of Pythagorean's theory.
- For all other triangles, this is a consequence of the cosine laws.



Example of the triangle inequality.
Sourced from Wikipedia[4]

Metric[5]

- A function that returns a distance between two items (in a set).
- A metric includes the topology of a set, but not all topologies can be generated from a metric.
- There are different types of metric and they all define their own rules.

String metric[6]

- An integer that measures the similarity between two strings.
- A string metric must meet the triangle inequality and be greater than zero.

Metric space[7]

- A non-empty set with a metric.
- The metric is a function that defines the concept of distance between two items in a set.
- A metric space must meet the following criteria: (where ABC are points)
  - The distance between two distinct points must always be greater than or equal to zero:
    - $d(a, b) \geq 0$
  - The distance between two points is only zero if the points are the same:
    - $(d(a, b) = 0) \equiv (a = b)$
  - The distance between two points is the same either way:
    - $d(a, b) \equiv d(b, a)$
  - The distance between two points is less than or equal to the distance between he two points via any third point:
    - $d(a, b) \leq d(a, c, b)$

1.4.2.2 Levenshtein distance[8]

A Levenshtein distance is a string metric used to measure the difference between two strings. The Levenshtein distance measures the number of insertions, deletions and substitutions of single characters that are required to translate one string to another.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}\big(\operatorname{tail}(a), b\big) \\ \operatorname{lev}\big(a, \operatorname{tail}(b)\big) \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) \end{cases} & \text{otherwise,} \end{cases}$$

The Levenshtein distance between two strings (a, b) is given by the above. Where the tail of a string is all the string but the first character. Sourced from Wikipedia[9]

The Levenshtein distance is also sometimes known as edit distance, as it refers to the number of character edits needed to transform one string to another. The Levenshtein distance is often used as a base for other algorithms and data structures.

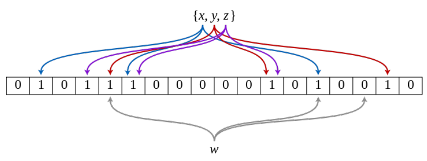
Example:
- To find the Levenshtein distance between "**kitten**" and "**sitting**":
  - Substitution of "**s**" for "**k**" - "**sitten**".
  - Substitution of "**i**" for "**e**" - "**sittin**".
  - Insertion of "**g**" at the end - "**sitting**".
- Therefore results in a Levenshtein distance of three .

There are some bounds for a Levenshtein distance:
- A levenshetin distance must be at least the difference of the sizes of the two strings, for example:
  - The difference between the strings "**hello**" and "**world**" is one, so the Levenshtein distance must be at least one.
- A Levenshtein distance must be at most the length of the longer string.
- A Levenshtein distance can only be zero if the strings are exactly equal, as there are no edits needed to make the strings equal.
- If the strings are the same length, then the Hamming distance is an upper bound of the Levenshtein distance. The Hamming distance is similar to the Levenshtein distance, but for strings of equal length (and only accounts for the number of substitutions).

Unlike a Hamming distance, a Levenshtein distance also takes into account the number of insertions or deletions required to match one string to another. This means that it can work on strings of different lengths, making it much more effective for a spell checker.

1.4.2.3 Bloom filter[10]

A bloom filter is a probabilistic data structure that is used to test whether an item is a part of a set. A probabilistic data structure is one that uses a randomised algorithm or hash function to store and lookup data. Being a probabilistic data structure means that there might be some false positive results, this means that it could return that an element is in the set but it's actually not. An empty bloom filter is a bit array, where all the bits are set to zero.



Example of a bloom filter representing the set {x, y, z} which may be present in the filter, as all the bits for each element is set to one.
{w} is not present in the filter, as one of the bits is set to zero.
Sourced from Wikipedia[11]

A bloom filter only supports two operations, insert and lookup. To insert an item into the array, feed it to each of the hashing algorithms. Set the bits at each of these indexes to one. To query for an item, feed it to each of the hashing algorithms. If any of the bits at these indexes are set to zero, then the item is definitely not present (because it would have been set to one during insertion). If the bits at all of these index are set to one, then the item might be present in the filter. Because the bits could have been set to one when inserting other items, this could be a false positive (returns that the value is present, when its not). There is no way to distinguish between real and false positives.

Items cannot be deleted from a bloom filter, because the deletion may also remove the bit of other items that are in the filter. This is not a problem though for my application, as the dictionary doesn't change very often, and another system can be used for words that may need to be deleted.

The hash functions in a bloom filter must be independent and (should) uniformly distributed. This means that all the hash functions must be different. They should also distribute the true bits (bits set to one) evenly across the bit array. They should also be as fast a possible, as multiple hash functions will be used for each insertion and lookup. Cryptographic hash functions are not required, so time can be saved by using non-cryptographic functions.

To reduce the probability of false positives, two things can be changed: the number of unique hash functions used; and/or the number of bits in the bit array. However, using more hash functions would reduce efficiency of the insertion and lookup functions. The probability of false positive rate can be approximated by: (where k is the number of hash functions, n is the number of expected items to be added to the filter, and m is the size of the bit array used)

- $(1 - e^{-kn/m})^k$

Interesting properties:

- Unlike a hash table, a Bloom filter with a fixed size can represent a set with an arbitrarily large number of elements, as it doesn't store the item itself.
- When elements are added the false positive rate increases steadily, until all queries return a positive result.
- A Bloom filter will never return a false negative. This is where an element exists, but the algorithm returns as if it doesn't exist.
- It is not possible to delete elements from a Bloom filter, as it could result in other elements also being deleted.
- Medium uses bloom filters for recommending posts to users, by filtering posts which have already been seen by the user.

Example:

In this example a bit array of size eleven and three hash functions will be used. All hash function outputs will be random and for illustrative purposes only. More hash functions and a larger bit array will be required for my application. Blue cells represent cells that have been edited from the last stage.

- When the bloom filter is created, a bit array is created and all the bits are set to zero.

0	0	0	0	0	0	0	0	0	0	0

- To insert the word "Hello" it is hashed, and the indexes at the outputs are set to one.

```
Hash1("Hello") % 11 = 1
Hash2("Hello") % 11 = 4
Hash3("Hello") % 11 = 7
```

0	1	0	0	1	0	0	1	0	0	0

- To insert the word "World" it is hashed, and the indexes at the outputs are set to one.

```
Hash1("World") % 11 = 0
Hash2("World") % 11 = 4
Hash3("World") % 11 = 8
```

1	1	0	0	1	0	0	1	1	0	0

- To check if the word "Test" is present in the bloom filter it is hashed, and all the indexes at the outputs are checked.

```
Hash1("Test") % 11 = 0
Hash2("Test") % 11 = 3
Hash3("Test") % 11 = 6
```

1	1	0	0	1	0	0	1	1	0	0

```
BloomFilter[0] = 1
BloomFilter[3] = 0
```

```
BloomFilter[6] = 0
```

- Because not all the bits are set to one, the word "Test" is definitely not present.

- To check if the word "Filter" is present in the bloom filter it is hashed, and all the indexes at the outputs are checked.

```
Hash1("Filter") % 11 = 0
Hash2("Filter") % 11 = 4
Hash3("Filter") % 11 = 8
```

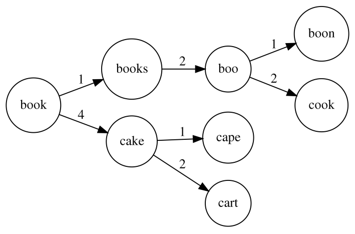
1	1	0	0	1	0	0	1	1	0	0

```
BloomFilter[0] = 1
BloomFilter[4] = 1
BloomFilter[8] = 1
```

- Because all the bits are set to one, the world "Filter" may be present.
- However we know it is not present as it was never added, this is a false positive.

1.4.2.4 BK-tree (Bukhard Keller tree)[12]

BK-trees are data structures used to quickly find near matches in a string. A BK-tree is a tree based data structure which means it can be implemented using an adjacency list or matrix.



Example of a BK-tree.
Sourced from Wikipedia[13]

BK-trees need a way to compare strings, this can be done in many ways but the most common method is a Levenshtein distance. As long as the function forms a metric space, it is acceptable for use in a BK-tree.

BK-trees are fairly fast when compared to other string "fuzzy search" algorithms. This is ideal for my use as a spell checker, because I will be using the tree for every word that is spelt incorrectly. A disadvantage of a BK-tree is the need to create a new tree for every incorrect word, this could take a lot of tine and delay the program significantly. However this could be solved by creating all the BK-tress initially and storing them separately to the main file.

A BK-tree is built so that:

- For each node in the tree, each of its edges are distinct.
- For each node in the tree, each child node has the same Levenshtein distance as the root does from its parent node. For example, using the image shown on the right:
  - "Books" has a distance of one from "Book".
  - Therefore each child node of "Books" has a distance of one from "Book":
    - "Boo" has a distance of two from "Books", but a distance of one from "Book".

To insert an item into a BK-tree:

- Find the distance between the item and the root node.
- Find the edge from the root node that has this distance.
  - If it doesn't exist:
    - Create a new edge with this distance.
    - Insert the item as a new node at with this edge on the tree.
  - If it does exist:
    - Find the node at the end of this edge and treat it as the new root node.
    - Repeat steps two onwards until the item has been inserted.

1.4.2.5 Hamming distance[14]

A hamming distance is a metric that measures the number of positions that are different between two sequences of equal length. A hamming distance can be calculated for sequences of any type (such as strings, integers or bits).

A hamming distance can only measure the difference between two sequences of equal length. This means that it is not entirely effective to find similar words for a spell checker. This is because users may input incorrect words with extra letters or missing characters, which would cause the hamming distance to be incalculable. Furthermore a hamming distance only takes into account the number of substitutions required to translate one sequence to another. Whereas some other metrics also consider the number of deletions and insertions required. Which is more suited for a spell checker.

Algorithmic representation:

```
def HammingDistance(string1, string2):
      distance = 0
      for n in range(len(string1)):
            if string1[n] != string2[n]:
                  dist_counter += 1
      return distance
```

In this representation, string1 and string2 are the sequences being passed in, and that the final hamming distance will be calculated from. The initial hamming distance is set to zero. The algorithm loops for the length of the strings. If the nth character of string1 is not equal to the nth character of string2, then the hamming distance is incremented by one.

- The hamming distance is the number of positions that are different in two strings of equal length.
- A hamming distance can be applied to any symbol (or sequence of symbols), such as characters, numbers or bits.
- For example:
  - "Ka**thr**in" and "Ka**rol**in" have a hamming distance of three.
  - 123**4567**890 and 123**7654**890 have a hamming distance of four.
  - 1**010** and 1**111** have a hamming distance of two.

1.4.2.6 Summary

From my research I found that there are many solutions to each of my problems, however I only researched a few that would suit my application well.

Levenshetin distance

A Levenshtein distance is the number of insertions, deletions and substitutions required to translate one string to another. It can be sued on strings of different lengths and can compare all characters. The Levenshtein distance is an integer, and an algorithm could disregard all distances greater than a given number to only provide items within a certain range. The Levenshtein distance isn't very effective on its own, as it is mainly used as a basis for other algorithms and data structures.

Bloom filter

A Bloom Filter is used to store elements in an array in an efficient way. Searching for elements in a bloom filter is often quicker than a binary or linear search. To search for elements a value is hashed and if all the values at the following indexes are one then the elements is probably in the filter. A Bloom Filter will only return, definitely not in the filter, or possibly in the filter.

BK-tree

A BK-tree is used to find close matches to elements using a tree. It works by using a Hamming or Levenshtein distance to create a tree with each element being its own node. From the root node, all other nodes connected to on elf the primary nodes will have the dame distance as the distance from root to primary node. The tree can therefore be used to find all elements within a close enough match.

Hamming distance

A Hamming distance is like simplified Levenshtein distance. It only measures the number of substitutions, whereas a Levenshtein distance insertions, deletions and substitutions. They distance of both of them does relate to the number of changes needed to translate one string to another. I problem won't use Hamming distances, and will most likely use the Levenshtein distance instead.

1.4.3 First interview

From this interview I aim to ask Gabi questions that will help me formulate the key objectives of my project. I hope to understand what Gabi sees as a priority and how she would like the features to look and work. I will also aim to ask Gabi how she feel I can make the program intuitive and user friendly for her use. I will interview Gabi by talking to her in person and recording a transcript of the interview, then I will type out the recorded transcript.

1.4.3.1 Transcript of frist interview

What spell checker do you currently use, and why?

I currently use the spell checker that is built into whatever app I am using. So when typing documents I use the spell checker built into Google Docs, and when on my iPhone I use Apple's spell checker. I mainly use these for convenience, as they are already in what I am using.

What do you like about this current spell checker?

I like how quickly they pick up on spelling errors and show the error in a clear way. I also like how they sometimes automatically change the word. On my iPhone the spelling error is shown by a red line and an automatic change is shown by a blue line.

What don't you like about this current spell checker?

When the checker finds a word that it doesn't know there is only options to dismiss the error, instead of being able to add it to the list of allowed words. Also, Apple's spell checker doesn't have the option to define words that it recommends for alternative spellings. They also don't allow acronyms or abbreviations of words, which is something I feel can be easily fixed.

Is it important to you for a spell checker to allow abbreviations and acronyms, or just something you "feel can be easily fixed"?

It is important to me to an extent, as it will allow me to save time when writing my scripts and when messaging my friends.

Would you prefer a brief definition, or multiple in-depth definitions for words that are spelt incorrectly?

Brief definition.

Why?

Because when I am writing I don't want to stop for long amounts of time to read a definition as it can break my concentration. Also the screen size on an iPhone isn't too big, so a large definition would take up a lot of the screen space and would stop me from using other features.

Are there any specific accessibility features you feel are important for a spell checker, or any application in general?

I think it would be nice if the user is able to change the font colour, as I know people who are colour blind and can struggle reading text that is certain colours. Also it would be nice if the user is able to change the font size. But these are not necessities, and wouldn't stop me using a specific app or feature.

Do you feel that personalisation of the app is important, why?

Yes. Because it can save me a lot of time by not having to change settings each time I use an app. Currently the apps I use for spell checkers have good personalisation but it could be improved by allowing me to add my own words and more.

Do you have any further comments?
I would like to be able to set my own character or word limit. This is because when writing essays I often have a word limit that I must stick to. Google Docs has a word count, but I can't add a limit and can write as much as I want. But there are other apps that have a word limit and won't allow me to exceed a set amount of words or characters, but don't have a spell checker. So combining these apps would be a good extra feature.

1.4.3.2 First interview analysis

From my first interview I learn that Gabi currently uses the spell checkers built into Google Docs and Apple's own devices, these are used mainly for convenience as they are already in the application that she is using. She likes how fast theses current options are and how they show errors with a red line. She also likes how Apple's spell checker automatically changes some incorrect spellings, if her attempt was close enough. However she doesn't like how the current spell checkers present her with a small number of options for incorrect words. Also she doesn't like how they don't define words that have been recommended for an incorrect spelling.

Gabi feels that it is important for a spell checker to allow acronyms and abbreviations as it will allow her to save time when writing and messaging friends. She also feels that a brief definition for words is better rather than multiple long definitions. This is because when on her iPhone long definitions will take up a lot of space, however this won't be a problem for my program as it will be an application for a computer. Also brief definitions will allow for easier comparisons between words.

Gabi also feels that personalisation is important for a program and would like to be able to change some settings that would help with accessibility. This could include changing the font size, font colour or background colour. However she doesn't feel this is a priority of the program, and would still use the program without it. Finally, Gabi would like to be able to set word or character limits to the program, and have the program restrict her from typing when the limit has been reached.

1.4.4 Key components

The program should be able to:
- Check the spelling of words
- Recommend alternative words for those that have been spelt incorrectly
- Define words with a simple, brief definition
- Allow user to personalise accessibility features (font size, font colour or background colour)
- Allow user to set character or word limits and prohibit typing when reached

1.5 Further research

1.5.1 Prototype

My prototype will model the Bloom filter that will be used in the final technical solution. The Bloom filter will be used to identify words that have been spelt incorrectly, and aren't in the dictionary. The full prototype will also be used in the final solution as the algorithm used to identify incorrect words.
(See appendix for full code for the Bloom filter)

1.5.1.1 Description

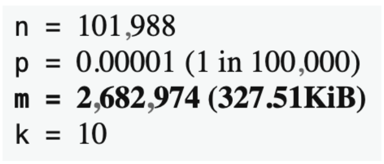
Bloom filters can be used to efficiently store and lookup elements in a set. This is because instead of storing the actual elements a Bloom filter stores an array of ones and zeros. To lookup an element it is uniquely hashed multiple times. If all of the indexes at the results of the hashes are one, then the element may be in the set. If any of the indexes at the results of the hashes are zero, then the elements is definitely not in the set.

1.5.1.2 Design

For the Bloom filter I decided to use a class, this allows me to have code for the filter and methods that will be used only for the Bloom filter together. It also allows me to control the accessibility of methods, so they can only be used by other classes when needed.

Array

I used an online calculator[15] to determine the size of the Bloom filter needed. I opted for a false probability rate of 1E-5 (0.00001) this means that every 1 in 100,000 lookups should reruns a face positive, I felt this was an acceptable rate as the average book has one hundred thousand words. From this calculator, I determined that the array should have a length of 2,682,974 (as shown in the image on the right). Because the array is going to be so big I decided to use a bit array rather than an integer or string array. This is because each index can only store a one or a zero, therefore the size of each index is small so the overall size of the array is smaller than that of an integer or string array. Furthermore in my array I only need each index to be able to store two possible values, so a bit array is the perfect solution.



```
n = 101,988
p = 0.00001 (1 in 100,000)
m = 2,682,974 (327.51KiB)
k = 10
```

Optimal size of bit array in the Bloom filter as calculated
Sourced from Bloom Filter Calculator[16]

Constructors

The Bloom Filter class will have two constructors.

The first constructor is the general one that will be used to convert the filter hat is stored in a text file to an array that will be used for the lookup and insert functions. Lines 21-24 contain a for loop that converts each individual character from the string array (read from the file) to an integer. However it had to minus forty eight from the initial integer conversion, because the character is converted as an ASCII value. It can be accessed by passing in no parameters.

The second constructor is used to reset the filter, it is used for initial use and testing of the filter. It uses a for loop (lines 31-34) to set each index in the filter to zero. It can be accessed by passing in any integer as a parameter.

Lookup

The lookup method works by using an if statement (lines 40-43) to test if all the indexes of the filter are set to one. These indexes are gained by using each of the separate hash functions to get ten indexes. If all these are set to one, then the method returns true. If any of them are set to zero then, the method return false.

Insert

The insert method gets ten integer values by hashing the word with each of the unique hash functions. It then sets the index at each value in the filter to one.

View filter

This method is mainly used for testing, as it will return the value at any given index of the filter. This value can then be compared to the text file for the expected value.

Write filter

This method is used for initial setup of the Bloom filter. It converts the full integer array to a string array and then to a string (lines 70-71), so it can be written to a text file.

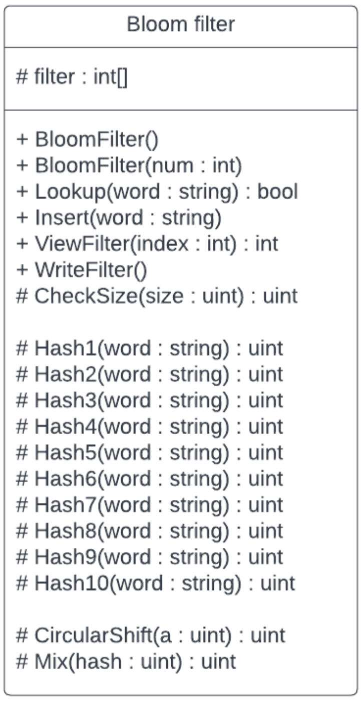
Check size

This method is used by all the hash functions to reduce the size of the hash to one that it less than the filter size. It works by using an if statement nested inside a do loop (lines 80-90) to test if the value is greater than the filter size. If it is, then. It performs a modulus function on the hash. If it isn't then it breaks from the do loop. This method is used to ensure there will be no overflow errors when inserting elements into the filter.

Hashing

A Bloom filter works by hashing the element a certain number of times with unique hash functions. I chose to use ten hash functions as it will help reduce the fall positivity rate, without having an infinitely big array. I found a list of hash functions[17] and chose the ten most appropriate functions to use in my Bloom filter class. Each hash function is written as its own method in the Bloom filter class, and all have an accessibility of protected. I opted to use protected rather than private, because it allows for expansion if I ever had to inherit from this class. Each hash function has its own local variable named "hash", this variable is an unsigned integer which allows it to store an unsigned thirty two bit integer (this means it can store numbers up to 4,294,967,296). Hash functions used:

- Pearson hashing (Hash 1) [18]
- Hashing by cyclic polynomial (Buzhash) (Hash 2) [19]
- Fowler-Noll-Vo (FNV-0) hash (Hash 3) [20]
- djb2 hash (Hash 4) [21]
- sdbm hash (Hash 5) [22]
- PJW hash function (Hash 6) [23]
- Fast-Hash (Hash 7) [24]
- Rabin fingerprint (Hash 8) [25]
- Fletcher-32 (Hash 9) [26]
- CRC32 (Hash 10) [27]


Bloom filter
# filter : int[]
+ BloomFilter()
+ BloomFilter(num : int)
+ Lookup(word : string) : bool
+ Insert(word : string)
+ ViewFilter(index : int) : int
+ WriteFilter()
# CheckSize(size : uint) : uint
# Hash1(word : string) : uint
# Hash2(word : string) : uint
# Hash3(word : string) : uint
# Hash4(word : string) : uint
# Hash5(word : string) : uint
# Hash6(word : string) : uint
# Hash7(word : string) : uint
# Hash8(word : string) : uint
# Hash9(word : string) : uint
# Hash10(word : string) : uint
# CircularShift(a : uint) : uint
# Mix(hash : uint) : uint

Bloom filter class diagram.

Circular shift

The circular shift helper is used to perform a binary shift on an input. It works by rotating the bits one to the left and pushing the most significant bit to the least significant position. Example a shift of 0110 results in 1100. This helper method is used in the Buzhash function.

1.5.1.3 Testing

I will only test the lookup method on the Bloom filter, as it is the only one that will be used frequently in general user the final solution. To test this I will generate five random words, and five random sequences of letters (each of a random length). I will then use a for loop to enter each of the test strings and record the output of each the results.

Objective	Test number	Test input	Expected output	Actual output	Result
Test the lookup function on the Bloom filter.	1	hand	True	True	Pass
	2	fax	True	True	Pass
	3	corsswalk	True	True	Pass
	4	wording	True	True	Pass
	5	thaw	True	True	Pass
	6	zhgrs	False	False	Pass
	7	mok	False	False	Pass
	8	wb	False	False	Pass
	9	n	False	True	Fail
	10	elnfoasyvw	False	False	Pass

Out of the ten tests conducted, there was only one failure. I think this was a failure because the input was a single character. To further test this I will test each single character as an input and record the result.

Objective	Test number	Test input	Expected output	Actual output	Result
Test single character lookups on the Bloom filter.	1	a	True	True	Pass
	2	b	False	True	Fail
	3	c	False	True	Fail
	4	d	False	True	Fail
	5	e	False	True	Fail
	6	f	False	True	Fail
	7	g	False	True	Fail
	8	h	False	True	Fail
	9	i	True	True	Pass
	10	j	False	True	Fail
	11	k	False	True	Fail
	12	l	False	True	Fail
	13	m	False	True	Fail
	14	n	False	True	Fail
	15	o	False	True	Fail
	16	p	False	True	Fail
	17	q	False	True	Fail
	18	r	False	False	Pass
	19	s	False	True	Fail
	20	t	False	False	Fail
	21	u	False	True	Fail
	22	v	False	True	Fail
	23	w	False	True	Fail
	24	x	False	True	Fail
	25	y	False	True	Fail
	26	z	False	False	Pass

The same problem occurred with one character strings, so extra verification and work will need to be done for the final solution. Only "a" and "i" should return a valid result.

1.5.1.4 Evaluation

After testing the Bloom filter lookup function I need to reconsider how single character inputs will be handled for the final worked solution. However the Bloom filter works quickly and accurately for strings of length greater than one. The Bloom filter works quickly, so it will be appropriate fr use in the final worked solution. I didn't test the probability of false positives, and my test doesn't use enough tests to get a definite false positive rate from. But the expected probability of false positives is 0.00001 (0.001% or every 1 in 100,000 lookups).

1.5.2 Second interview

From this interview I hope to get Gabi's opinion my current objectives. I will aim to modify my objectives so that the program will be able to better meet Gabi's wants and needs. I will interview Gabi by talking to her in person and recording a transcript of the interview.

1.5.2.1 Transcript of second interview

How do you feel about the current prototype?
I feel it works well and should be good in the final app due to the speed it operates at. I think the tests you carried out were good, but you do need to think about how to solve the problem with words that are one better long.

What do you suggest I do with words that are one character long?
Maybe you could take these words separately and not sue the lookup function. Instead just return false for all these that aren't real words.

How do you feel about the current objectives?
I feel the current objectives will create a good app with good functionality and I think that all of them are possible. But I would highlight the importance of personalisation and make sure that is a priority objective.

Would you add or change any if the current objectives?
I would just add more requirements under the dictionary section. Such as: sort form a-z; search in the dictionary; filter it by words I've added; and drink words there as well if the user asks.

1.5.2.2 Second interview analysis

Gabi felt that the prototype was good at identifying words and would be good for use in the final technical solution because it was also efficient and could loop a word quickly. However she did highlight the important problem that was occurring with one character words, but suggested a solution. So for this problem I will use an if statement to check the length of the word and return false if the word is only one charter (and not "a" or "I").

Also, Gabi felt that the current objectives will solve the problem well and all of them are possible for my skill level. She did add some extra pints that need to be refined in section seven of the objectives. Such as being able to define words and having them sorted from a-z.

1.6 Objectives

Where the numbers 1-9 represent a main feature to implement and each sub-bullet represents a section of the main feature that will allow it to work intuitively and effectively.

1. **Interface:**
   a. The interface should present the user with a simple welcome page that allows the user to select from a list of options, or enter text to begin the spell checking. The options on the welcome page should be clear and allow the user to access all aspects of the program intuitively.
   b. The interface should mark words that have been spelt incorrectly for the user with an appropriate message, or by changing the formatting of the word.

   c. The interface should provide the user with options for words that have been spelt incorrectly with an appropriate message, and give the user the choice to:
      i. Change the incorrect word to one that has been recommended by the algorithm, and undo the formatting applied to the incorrect word.
      ii. Ignore the error and undo the formatting applied to the incorrect word.
      iii. Add the word to the dictionary and undo the formatting applied to the incorrect word.
   d. The user should be able to personalise the interface by having the ability to change the font size and colour.

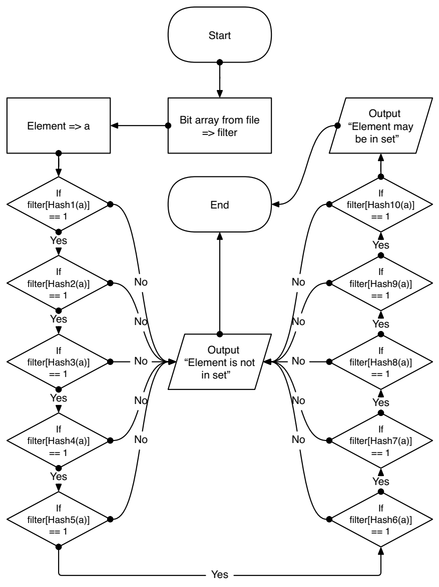
2. **Spell check:**
   a. The program should check all words entered by the user against a list of allowed words and identify any that do not match, as incorrectly spelt.
   b. These incorrect words should be identified with an appropriate message or formatting.
3. **Alternative word recommender:**
   a. The program should recommend alternative words to those that have been identified as an incorrect spelling.
   b. The program should return only the words that are the closest match, instead of returning all words that contain any match. The criteria for 'closest match' should be defined when the algorithm starts, and should be changed depending on how many alternative words have been returned.
   c. The program should allow the user to choose between the recommended words, and then replace the incorrect word with the selected one. Undoing any formatting that has been applied to the incorrect word.
   d. The program should allow the user to define all words that have been recommended, with a brief definition.
4. **Grammar check: (extension)**
   a. The program should check for basic grammar errors, such as missing full stops and capital letters.
   b. The program should display grammar errors in an appropriate way for the user, with an appropriate message or by formatting the errors in a unique way.
   c. The program should mark any errors in an appropriate way and give the user a list of options to 'fix' the error.
5. **Alternative grammar recommender: (extension)**
   a. The program should recommend alternative grammar to any errors that have been identified in the text.
   b. The program should give a suitable reason why the alternative should be used and allow the user to apply the change, undoing any formatting that has been applied to the error.
6. **Change the status of features:**
   a. The program should allow the user to enable and disable all features.
   b. The status of features should be saved, so they can be reapplied each time the user starts the program.
7. **Personalise the dictionary:**
   a. The program should allow the user to add words that have been identified as incorrectly spelt to the dictionary.
   b. The program should allow the user to view the dictionary:
      i. The dictionary should be sorted alphabetically from a-z.
      ii. The user should be able to filter the dictionary by default words and added words.
      iii. The user should be able to search for words in the dictionary and add them if they are not currently in the dictionary.
      iv. The program should define words in the dictionary if requested by the user in a suitable way.
      v. The user should be able to remove words from the dictionary when requested.
8. **Set word and character limits:**
   a. The program should allow the user to set word and character limits, and the program should display a suitable message and stop the user from typing when the limit has been reached.
   b. The user should be able to personalise the limit.
   c. The user should be able to choose whether the limit is based off words or characters.
   d. If a character limit has been chosen, the user should be able to choose whether spaces are included or not.
9. **Replace acronyms:**
   a. The program should allow the user to define acronyms and their full meaning. For example: omw - on my way.
   b. The program should automatically change the acronym to the full meaning when found in text.

c. The program should allow for an easy way for the user to ignore the acronym and revert back to the original shortened version.

1.7 Model

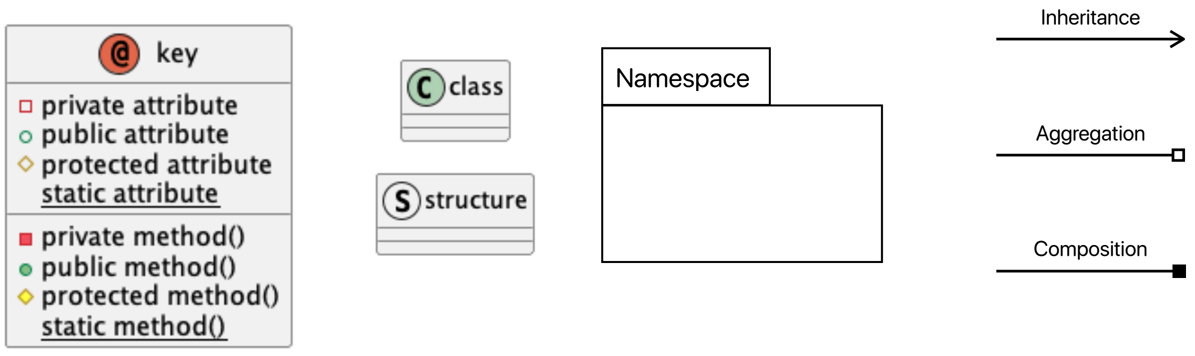
I decided to create a flowchart that shows how a lookup function will work in the Bloom filter. The flowchart shows:

- The array being stored to the variable named “filter”.
- Then the element (word to be searched for) being stored to the variable named “a”.
- A series of if statements that compare the value of the index at each hash function to one.
  - If the if statement returns yes, then the program will continue to the next if statement.
  - If the statement returns no, then the program will output “Element is not in set”.
  - If the final if statement returns yes, then the program will output “Element may be in the set”.
- The program ends.

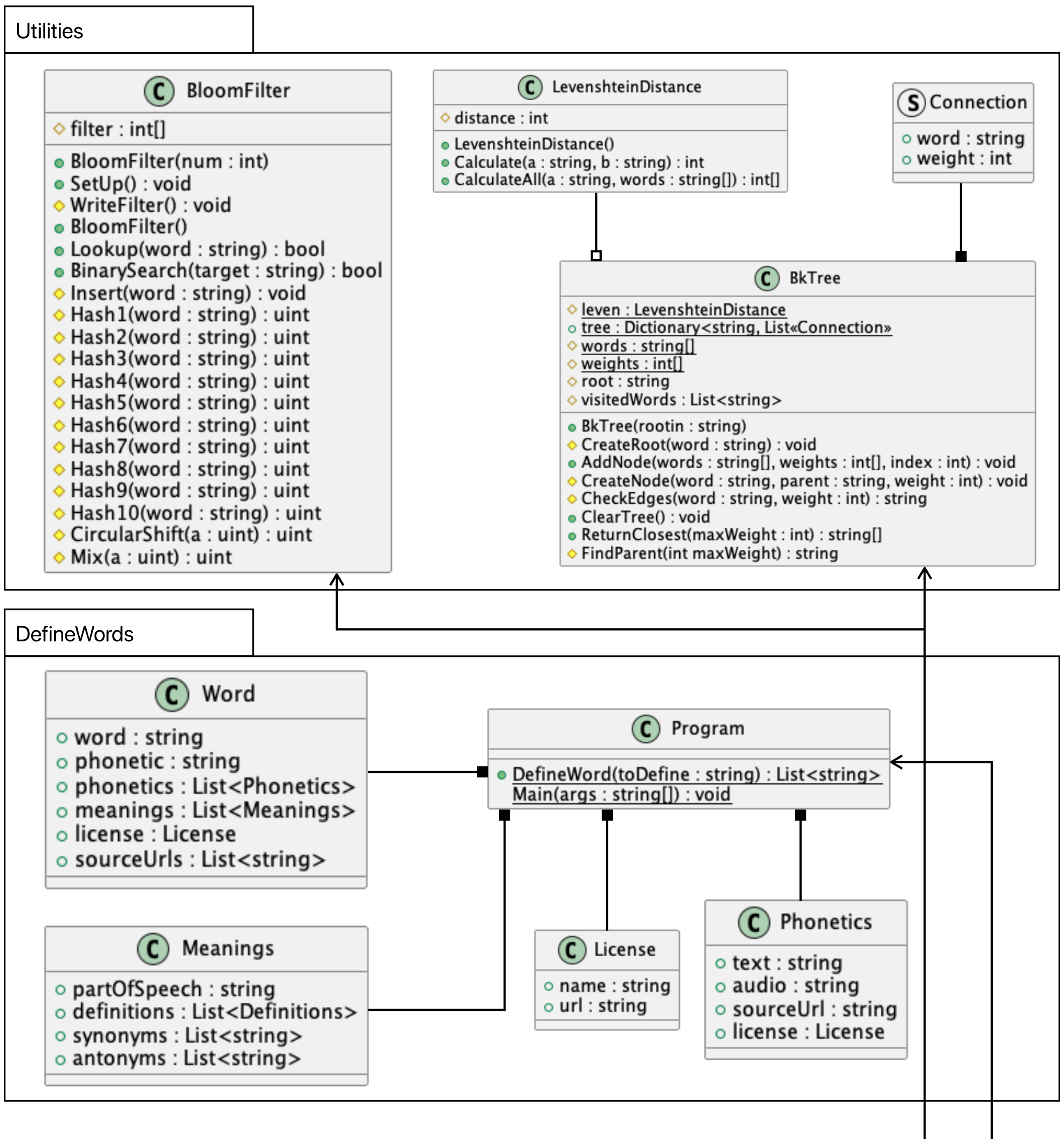


2 Design

2.1 Class diagrams

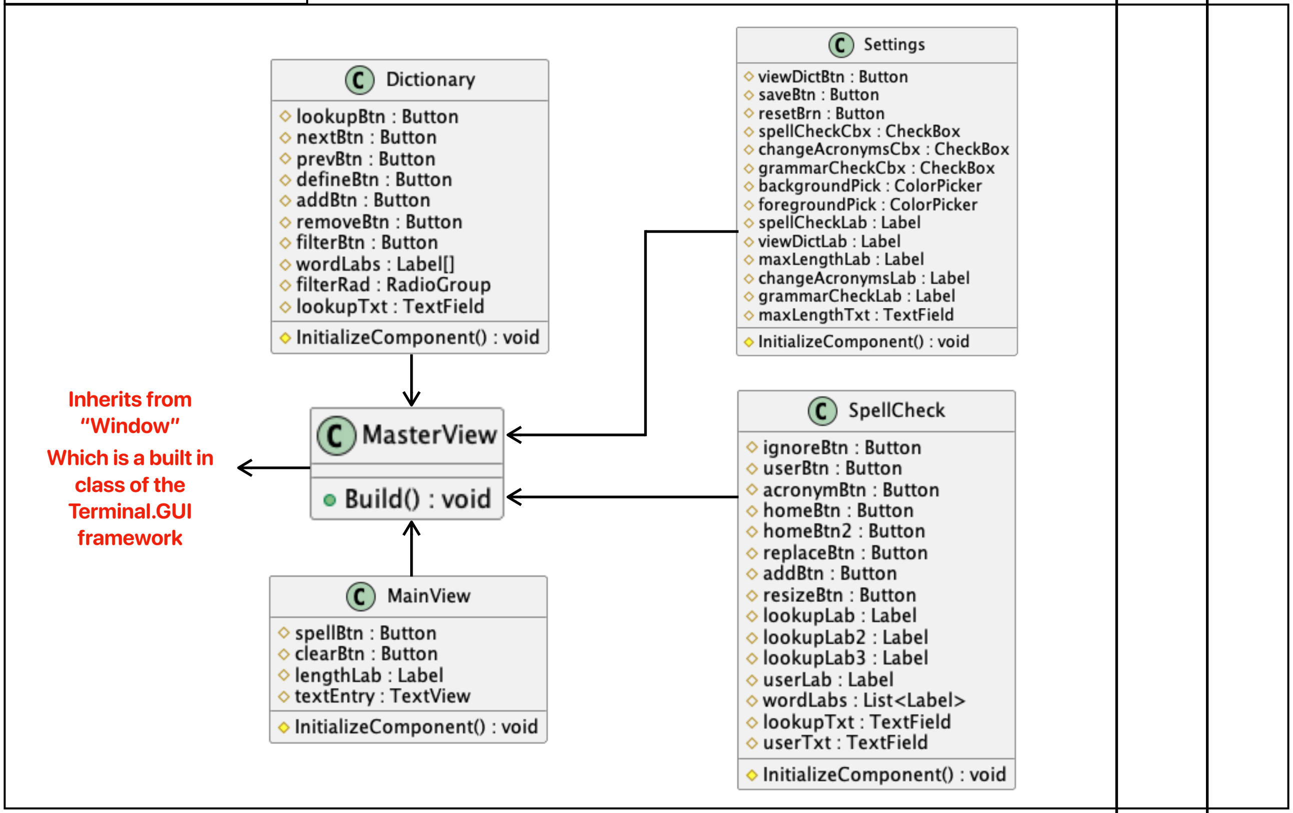


2.1.1 Utilities


Utilities
BloomFilter
filter : int[]
BloomFilter(num : int)
SetUp() : void
WriteFilter() : void
BloomFilter()
Lookup(word : string) : bool
BinarySearch(target : string) : bool
Insert(word : string) : void
Hash1(word : string) : uint
Hash2(word : string) : uint
Hash3(word : string) : uint
Hash4(word : string) : uint
Hash5(word : string) : uint
Hash6(word : string) : uint
Hash7(word : string) : uint
Hash8(word : string) : uint
Hash9(word : string) : uint
Hash10(word : string) : uint
CircularShift(a : uint) : uint
Mix(a : uint) : uint
LevenshteinDistance
distance : int
LevenshteinDistance()
Calculate(a : string, b : string) : int
CalculateAll(a : string, words : string[]) : int[]
Connection
word : string
weight : int
BkTree
leven : LevenshteinDistance
tree : Dictionary<string, List«Connection»
words : string[]
weights : int[]
root : string
visitedWords : List<string>
BkTree(rootin : string)
CreateRoot(word : string) : void
AddNode(words : string[], weights : int[], index : int) : void
CreateNode(word : string, parent : string, weight : int) : void
CheckEdges(word : string, weight : int) : string
ClearTree() : void
ReturnClosest(maxWeight : int) : string[]
FindParent(int maxWeight) : string
DefineWords
Word
word : string
phonetic : string
phonetics : List<Phonetics>
meanings : List<Meanings>
license : License
sourceUrls : List<string>
Program
DefineWord(toDefine : string) : List<string>
Main(args : string[]) : void
Meanings
partOfSpeech : string
definitions : List<Definitions>
synonyms : List<string>
antonyms : List<string>
License
name : string
url : string
Phonetics
text : string
audio : string
sourceUrl : string
license : License

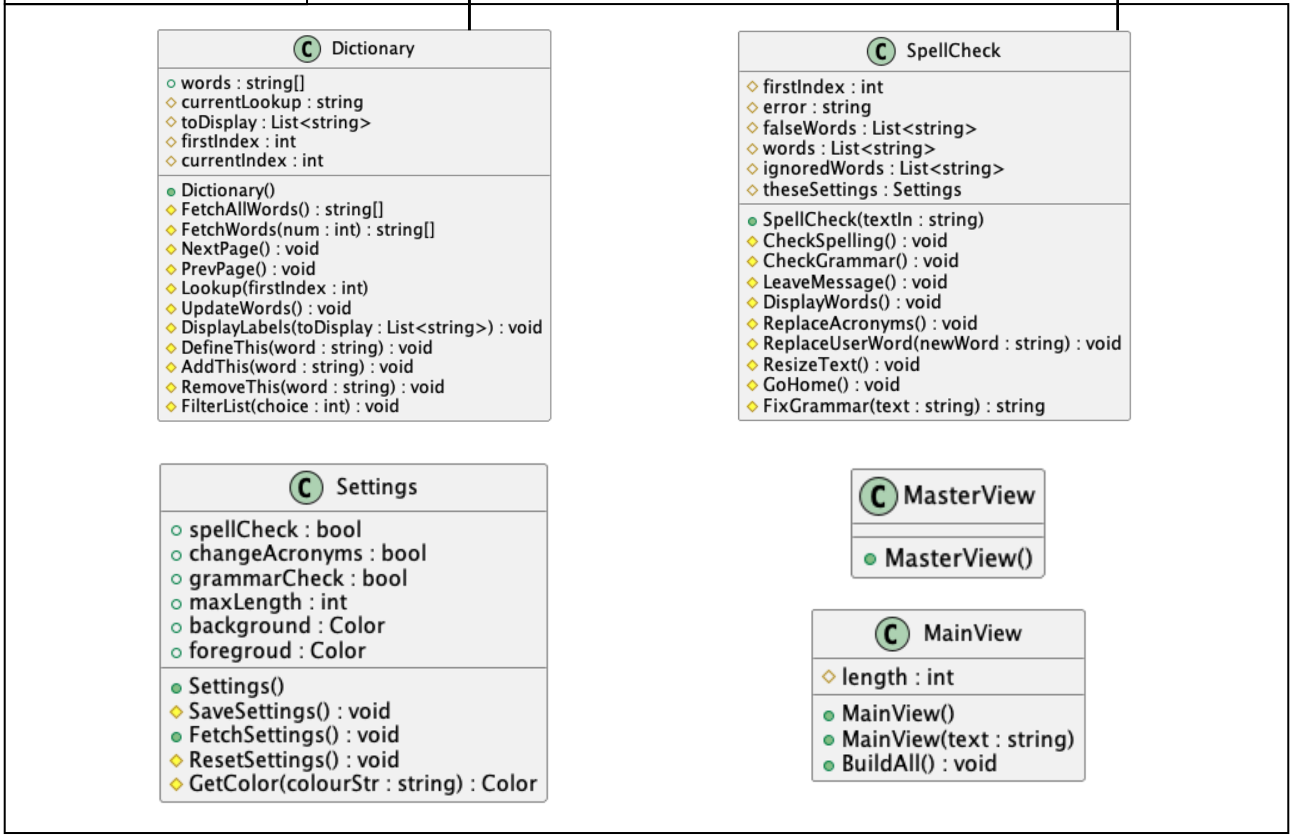
2.1.2 View designers

FINAL



2.1.3 Views

FINAL



3 Testing

4 Evaluation

5 Technical solution

5.1 Table of contents

Master view designer	Code starts	
	Base class	
	Inheritance	

5.3.2 Main view designer

Main view designer	Code starts	
	Inheritance	

5.3.3 Dictionary view designer

Dictionary view designer	Code starts	
	Inheritance	

5.3.4 Settings view designer

Dictionary view designer	Code starts	
	Inheritance	

5.3.5 Spell check view designer

Spell check view designer	Code starts	
	Inheritance	

5.4 Views

5.4.1 Master view

Master view	Code starts	

5.4.2 Main view

Main view	Code starts	
	Ad hoc polymorphism	

5.4.3 Dictionary view

Dictionary view	Code starts	
	Single-dimensional array	
	Reading from text files	
	Writing to a text file	

5.4.4 Settings view

Settings view	Code starts	
	Writing to a text file	
	Reading from text files	

5.4.5 Spell check view

Spell check view	Code starts	

	Lists	
	String manipulation	
5.5 Program.cs		
Main cs file	Code starts	
5.6 FINAL.csproj		
csproj file	Code starts	
5.7 Filter word lists		
Filter word lists	Code starts	
	Reading from a text file	
	Single-dimensional arrays	
	Parameterised API call	
	Writing to text files	

5.2 Utilities

```
/////////////////
// Utilities.cs //
/////////////////

// install dependencies
using System.Text;
using System.Net;
using Newtonsoft.Json;

// suppress warnings
# pragma warning disable

namespace Utilities
{
```

5.2.1 Bk tree

```
    ////////////
    // BKTree //
    ////////////

    class BkTree
    {
        protected static LevenshteinDistance leven = new LevenshteinDistance();
        // each node has its own word, and a list of connections to store edge weights to children
        public static Dictionary<string, List<Connection>> tree = new Dictionary<string,
List<Connection>>();
        protected static string[] words = File.ReadAllLines("Resources/TrueWords.txt");
        protected static int[] weights = new int[words.Length];
        protected string root;
        protected List<string> visitedWords = new List<string>();

        public BkTree(string rootIn)
        {
            tree.Clear();

            root = rootIn;

            weights = leven.CalculateAll(rootIn, words);

            // create the tree where every word is a node
            for (int a = 0; a < words.Length; a++)
            {
                if (tree.Count == 0)
                {
                    CreateRoot(rootIn);
                }
                else
                {
                    if (words[a] != root)
                    {
                        AddNode(words, weights, a);
                    }
                }
            }
        }

        protected void CreateRoot(string word)
        {
            Connection thisConnection = new Connection();
            thisConnection.word = word;
            // weight from root -> root = i (therefore 0)
            thisConnection.weight = 0;

            List<Connection> connections = new List<Connection>();
            connections.Add(thisConnection);

            tree.Add(word, connections);
        }

        public void AddNode(string[] words, int[] weights, int index)
        {
            Connection thisConnection = new Connection();
            thisConnection.word = words[index];
            thisConnection.weight = weights[index];

            CreateNode(thisConnection.word, root, thisConnection.weight);
        }

        protected void CreateNode(string word, string parent, int weight)
        {
            // checks if an edge from the root node already exists with the same weight
            // if it doesn't, then create node as nornmal from the root
            // if it does, then use that node (that has the same weight) as the new parent
            // due to the nature of BK-trees every edge from a parent must have a distinct weight
            string? exists = CheckEdges(parent, weight);
            if (exists is not null)
            {
                // but weight from the parent might be different to the weight from the node, so
recalculate
                int newWeight = leven.Calculate(parent, word);
                CreateNode(word, exists, newWeight);
            }
            else
            {
                Connection thisConnection = new Connection();
                thisConnection.word = word;
                thisConnection.weight = weight;

                if (!tree.ContainsKey(thisConnection.word))
                {
                    tree.Add(thisConnection.word, new List<Connection>());
                    tree[parent].Add(thisConnection);
                }
            }
        }

        // search each edge from a specified parent to see if a particular weight already exists
        protected string CheckEdges(string word, int weight)
        {
            List<int> edges = new List<int>();

            for (int a = 0; a < tree[word].Count; a++)
            {
                edges.Add(tree[word][a].weight);
            }

            for (int a = 0; a < edges.Count; a++)
            {
                if (edges[a] == weight)
                {
                    return tree[word][a].word;
                }
            }

            return null;
        }

        public void ClearTree()
        {
            for (int a = 0; a < tree.Count; a++)
            {
                tree.Remove(words[a]);
            }

            tree.Clear();
        }

        // input 1 when ReturnClosest is called in program
        // public List<string> ReturnClosest(int maxWeight)
        public string[] ReturnClosest(int maxWeight)
        {
            // used as a dummy to store the index of last word used as parent
            Connection dummyConnection = new Connection();
            dummyConnection.word = "";
            dummyConnection.weight = 0;

            string parent = FindParent(maxWeight);
            List<Connection> closeWords = tree[parent];

            while (closeWords.Count < 15)
            {
                if (!(dummyConnection.weight < closeWords.Count))
                {
                    break;
                }

                parent = closeWords[dummyConnection.weight].word;
                closeWords.AddRange(tree[parent]);
                dummyConnection.weight++;
            }

            string[] wordsOut = new string[closeWords.Count];

            for (int a = 0; a < closeWords.Count; a++)
            {
                wordsOut[a] = closeWords[a].word;
            }

            return wordsOut;
        }

        protected string FindParent(int maxWeight)
        {
            bool exists = false;

            while (!exists)
            {
                for (int a = 0; a < tree[root].Count; a++)
                {
                    if (tree[root][a].weight < maxWeight && tree[root][a].word != root)
                    {
                        return tree[root][a].word;
                    }
                }

                if (!exists)
                {
                    maxWeight++;
                }
            }

            return null;
        }
    }
```

5.2.2 Bloom filter

```
/////////////////
// BloomFilter //
/////////////////

class BloomFilter
{
    protected int[] filter = new int[2682975];

    // FIRST TIME USE
    //
    // constructor used to reset all indexes in the filter to 0
    // then setup the filter to the text file
    public BloomFilter(int num)
    {
        for (int a = 0; a < filter.Length; a++)
        {
            filter[a] = 0;
        }

        SetUp();
    }

    // reads the list of words from text file
    // inserts each word into filter
    // write the filter to a text file
    public void SetUp()
    {
        string[] words = File.ReadAllLines("Resources/TrueWords.txt");

        foreach (var a in words)
```

```
            {
                Insert(a);
            }

            WriteFilter();
        }

        // write the filter to a text file
        protected void WriteFilter()
        {
            string filename = "Resources/Filter.txt";
            string[] stringArray = filter.Select(x => x.ToString()).ToArray();
            string result = String.Concat(stringArray);

            File.WriteAllText(filename, result);
        }

        // GENERAL USE
        //
        // read the filter from a text file and store as an int array
        public BloomFilter()
        {
            string filename = "Resources/Filter.txt";
            string text = File.ReadAllText(filename);

            for (uint a = 0; a < filter.Length; a++)
            {
                filter[a] = text[(int)a] - 48;
            }
        }

        public bool Lookup(string word)
        {
            // there are only two one letter words that are in the set - "a" and "i"
            // however all one letter words return true, so check all one letter words
            if (word.Length == 1)
            {
                return (word == "a" || word == "i");
            }
            else
            {
                // if all the indexes are 1, then the item may be in the set
                // if any of the indexes are 0, then the item is definitely not in the set
                if (filter[Hash1(word)] == 1 && filter[Hash2(word)] == 1 && filter[Hash3(word)] == 1
&& filter[Hash4(word)] == 1 && filter[Hash5(word)] == 1 && filter[Hash6(word)] == 1 &&
filter[Hash7(word)] == 1 && filter[Hash8(word)] == 1 && filter[Hash9(word)] == 1 &&
filter[Hash10(word)] == 1)
                {
                    return true;
                }
                // if any of the indexes are 0, then do a binary search to check the list of user-
added words
                else
                {
                    return BinarySearch(word);
                }
            }
        }

        public bool BinarySearch(string target)
        {
            target = target.ToLower();
            string[] words = File.ReadAllLines("Resources/AddedWords.txt");
            int min = 0;
            int max = words.Length;
            int mid;

            while (min < max)
            {
                mid = (min + max) / 2;

                if (words[mid] == target)
                {
                    return true;
                }
                else if (words[mid].CompareTo(target) < 0)
                {
                    min = mid + 1;
                }
                else
                {
                    max = mid - 1;
                }
            }

            return false;
        }

        protected void Insert(string word)
        {
            filter[Hash1(word)] = 1;
            filter[Hash2(word)] = 1;
            filter[Hash3(word)] = 1;
            filter[Hash4(word)] = 1;
            filter[Hash5(word)] = 1;
            filter[Hash6(word)] = 1;
            filter[Hash7(word)] = 1;
            filter[Hash8(word)] = 1;
            filter[Hash9(word)] = 1;
            filter[Hash10(word)] = 1;

            WriteFilter();
        }

        // Pearson Hashing
        protected uint Hash1(string word)
        {
            uint hash = 0;
            byte[] bytes = Encoding.UTF8.GetBytes(word);
            byte[] nums = { 114, 177, 249, 4, 222, 117, 190, 121, 130, 78, 53, 196, 255, 208, 5,
116, 221, 27, 144, 41, 252, 33, 170, 231, 62, 89, 235, 111, 174, 57, 105, 132, 204, 205, 151, 135,
90, 211, 37, 36, 66, 164, 40, 253, 108, 153, 98, 156, 67, 214, 35, 6, 38, 42, 162, 148, 28, 18, 254,
79, 61, 155, 3, 25, 184, 189, 152, 143, 84, 216, 87, 44, 75, 138, 191, 158, 243, 230, 1, 242, 91,
113, 26, 171, 245, 197, 22, 68, 187, 161, 218, 246, 97, 16, 234, 193, 73, 125, 101, 80, 226, 195,
139, 49, 9, 212, 224, 63, 72, 13, 100, 233, 104, 163, 207, 247, 137, 199, 136, 160, 203, 141, 250,
71, 200, 167, 129, 32, 19, 145, 238, 43, 142, 237, 198, 64, 76, 103, 182, 149, 2, 74, 107, 124, 88,
54, 157, 159, 51, 52, 102, 201, 7, 77, 180, 110, 109, 228, 85, 99, 11, 239, 169, 12, 8, 209, 165,
168, 248, 34, 82, 112, 140, 56, 120, 185, 55, 58, 31, 179, 47, 213, 86, 206, 194, 69, 127, 147, 123,
20, 219, 166, 29, 223, 220, 83, 70, 225, 188, 60, 21, 251, 240, 10, 119, 122, 23, 131, 96, 178, 227,
126, 173, 14, 17, 176, 192, 15, 46, 65, 215, 134, 232, 115, 106, 181, 175, 48, 202, 154, 150, 81,
50, 183, 39, 229, 92, 24, 217, 45, 172, 95, 128, 93, 133, 244, 210, 186, 118, 59, 30, 241, 146, 236,
94 };

            foreach (var a in bytes)
            {
                hash = nums[((hash ^ a) % (uint)nums.Length)];
            }

            hash %= (uint)filter.Length;
            return hash;
        }

        // Hashing by cyclic polynomial (Buzhash)
        protected uint Hash2(string word)
        {
            uint hash = 1;

            foreach (char a in word)
            {
                hash = CircularShift(hash) ^ CircularShift((byte)a) ^ CircularShift((uint)(byte)a +
1);
            }

            hash %= (uint)filter.Length;
            return hash;
        }

        // Fowler-Noll-Vo (FNV-0) hash
        protected uint Hash3(string word)
        {
            uint hash = 0;
            byte[] bytes = Encoding.UTF8.GetBytes(word);

            foreach (var a in bytes)
            {
                hash = (hash * 16777619) ^ a;
            }

            hash %= (uint)filter.Length;
            return hash;
        }

        // dijb2
        protected uint Hash4(string word)
        {
            // randomly generated number
            uint hash = 5381;
            byte[] bytes = Encoding.UTF8.GetBytes(word);

            foreach (var b in bytes)
            {
                hash = ((hash << 5) + hash) + 33;
            }

            hash %= (uint)filter.Length;
            return hash;
        }

        // sdbm
        protected uint Hash5(string word)
        {
            uint hash = 0;
            byte[] bytes = Encoding.UTF8.GetBytes(word);

            foreach (var a in bytes)
            {
                hash = 65599 + (hash << 6) + (hash << 16) - hash;
            }

            hash %= (uint)filter.Length;
            return hash;
        }

        // PJW hash function
        protected uint Hash6(string word)
        {
            uint hash = 0;
            uint bits = (sizeof(uint) * 8);
            uint max = (uint)(0xFFFFFFFF) << (int)(bits - (bits / 8));

            for (int a = 0; a < word.Length; a++)
            {
                hash = hash << (int)(bits / 8) + word[a];

                if (max != 0)
                {
                    hash = hash ^ (max >> (int)bits * 3 / 4) & (~max);
                }
            }

            hash %= (uint)filter.Length;
            return hash;
        }

        // Fast-Hash
        protected uint Hash7(string word)
        {
            // randomly generated number
            uint a = unchecked((uint)0x880355f21e6d1965);
            uint hash = 144 ^ ((uint)word.Length * a);
            uint b = 0;

            for (uint c = 0; c < word.Length; c++)
            {
                hash = (hash ^ Mix(c)) * a;
            }

            int d = word.Length & 7;
            d %= word.Length;
            int e = (d - 1) * 8;
            if (e < 0)
            {
                e = 0;
            }
```

```
            b ^= (uint)word[d] << d;

            hash ^= Mix(b) * a;
            hash = Mix(hash);

            hash %= (uint)filter.Length;
            return (uint)hash;
        }

        // Rabin Fingerprint
        protected uint Hash8(string word)
        {
            uint hash = word[0];
            uint length = (uint)word.Length;

            for (int a = 1; a < length; a++)
            {
                hash += (uint)Math.Pow(word[a] * length, a);
            }

            hash %= (uint)filter.Length;
            return hash;
        }

        // Fletcher-32
        protected uint Hash9(string word)
        {
            uint a = 0;
            uint b = 0;

            for (int c = 0; c < word.Length; c++)
            {
                a = (a + word[c] % (uint)0xffff);
                b = (a + b) % (uint)0xffff;
            }

            uint hash = (b << 16) | a;

            hash %= (uint)filter.Length;
            return hash;
        }

        // CRC32
        protected uint Hash10(string word)
        {
            // randomly generated number
            uint hash = 0xffffffff;
            uint index;
            byte[] nums = { 114, 177, 249, 4, 222, 117, 190, 121, 130, 78, 53, 196, 255, 208, 5,
116, 221, 27, 144, 41, 252, 33, 170, 231, 62, 89, 235, 111, 174, 57, 105, 132, 204, 205, 151, 135,
90, 211, 37, 36, 66, 164, 40, 253, 108, 153, 98, 156, 67, 214, 35, 6, 38, 42, 162, 148, 28, 18, 254,
79, 61, 155, 3, 25, 184, 189, 152, 143, 84, 216, 87, 44, 75, 138, 191, 158, 243, 230, 1, 242, 91,
113, 26, 171, 245, 197, 22, 68, 187, 161, 218, 246, 97, 16, 234, 193, 73, 125, 101, 80, 226, 195,
139, 49, 9, 212, 224, 63, 72, 13, 100, 233, 104, 163, 207, 247, 137, 199, 136, 160, 203, 141, 250,
71, 200, 167, 129, 32, 19, 145, 238, 43, 142, 237, 198, 64, 76, 103, 182, 149, 2, 74, 107, 124, 88,
54, 157, 159, 51, 52, 102, 201, 7, 77, 180, 110, 109, 228, 85, 99, 11, 239, 169, 12, 8, 209, 165,
168, 248, 34, 82, 112, 140, 56, 120, 185, 55, 58, 31, 179, 47, 213, 86, 206, 194, 69, 127, 147, 123,
20, 219, 166, 29, 223, 220, 83, 70, 225, 188, 60, 21, 251, 240, 10, 119, 122, 23, 131, 96, 178, 227,
126, 173, 14, 17, 176, 192, 15, 46, 65, 215, 134, 232, 115, 106, 181, 175, 48, 202, 154, 150, 81,
50, 183, 39, 229, 92, 24, 217, 45, 172, 95, 128, 93, 133, 244, 210, 186, 118, 59, 30, 241, 146, 236,
94, 0 };

            byte[] bytes = Encoding.UTF8.GetBytes(word);

            foreach (var b in bytes)
            {
                index = ((hash ^ b) & 0xff) % (uint)nums.Length;
                hash = (hash >> 8) ^ nums[index];
            }

            hash ^= 0xffffffff;

            hash %= (uint)filter.Length;
            return hash;
        }

        // helpers
        // for hash2
        protected uint CircularShift(uint a)
        {
            uint b = a << 1 | a >> 31;
            return b;
        }

        // for hash7
        protected uint Mix(uint hash)
        {
            long a = (long)hash;
            a ^= a >> 23;
            // randomly generated number
            a *= 0x2127599bf4325c37;
            a ^= a >> 47;

            hash = (uint)a;
            return (uint)hash;
        }
    }
```

5.2.3 Connection

```
    ////////////////
    // Connection //
    ////////////////

    public struct Connection
    {
        // node
        public string word;
        // edge to parent
        public int weight;
    }
```

5.2.4 Levenshtein distance

```
    /////////////////
    // Levenshtein //
    /////////////////

    class LevenshteinDistance
    {
        protected int distance;

        public LevenshteinDistance()
        {
            distance = 0;
        }

        public int Calculate(string a, string b)
        {
            int cost;

            // if a or b has no length, return the other one
            if (a.Length == 0)
            {
                return b.Length;
            }
            else if (b.Length == 0)
            {
                return a.Length;
            }

            // row and column sizes are (word.Length + 1) because row/column 1 store index values
            int[,] matrix = new int[a.Length + 1, b.Length + 1];

            // store index values in row/coilumn 1, example
            // 0 1 ...
            // 1
            // 2
            // ...
            for (int c = 0; c < matrix.GetLength(0); c++)
            {
                matrix[c, 0] = c;
                for (int d = 0; d < matrix.GetLength(1); d++)
                {
                    matrix[0, d] = d;
                }
            }

            for (int c = 1; c < matrix.GetLength(0); c++)
            {
                for (int d = 1; d < matrix.GetLength(1); d++)
                {
                    // if the values at the index are the same then no change is needed
                    if (a[c - 1] == b[d - 1])
                    {
                        cost = 0;
                    }
                    // if not then a change (substitution, deletion or addition) is needed
                    else
                    {
                        cost = 1;
                    }

                    int value1 = matrix[c - 1, d] + 1;
                    int value2 = matrix[c, d - 1] + 1;
                    int value3 = matrix[c - 1, d - 1] + cost;
                    int[] values = { value1, value2, value3 };
                    matrix[c, d] = values.Min();
                }
            }

            // final cost is stored in the bottom right of the matrix
            distance = matrix[a.Length, b.Length];
            return distance;
        }

        public int[] CalculateAll(string a, string[] words)
        {
            int[] distances = new int[words.Length];

            for (int b = 0; b < words.Length; b++)
            {
                distances[b] = Calculate(a, words[b]);
            }

            return distances;
        }
    }
}
```

5.2.5 Define words

```
/////////////////
// DefineWords //
/////////////////

namespace DefineWords
{
    // create template to store JSON data retruned by API call
    class Word
    {
        public string? word { get; set; }
        public string? phonetic { get; set; }
        public List<Phonetics>? phonetics { get; set; }
        public List<Meanings>? meanings { get; set; }
        public License? license { get; set; }
        public List<string>? sourceUrls { get; set; }
    }

    class Phonetics
    {
        public string? text { get; set; }
        public string? audio { get; set; }
        public string? sourceUrl { get; set; }
        public License? license { get; set; }
    }

    class License
    {
        public string? name { get; set; }
        public string? url { get; set; }
    }

    class Meanings
    {
        public string? partOfSpeech { get; set; }
        public List<Definitions>? definitions { get; set; }
        public List<string>? synonyms { get; set; }
        public List<string>? antonyms { get; set; }
    }

    class Definitions
    {
        public string? definition { get; set; }
        public List<string>? synonyms { get; set; }
        public List<string>? antonyms { get; set; }
        public string? example { get; set; }
    }

    class Program
    {
        public static List<string> DefineWord(string toDefine)
        {
            WebRequest request = WebRequest.Create("https://api.dictionaryapi.dev/api/v2/entries/
en/" + toDefine);
            HttpWebResponse response = (HttpWebResponse)request.GetResponse();
            Stream dataStream = response.GetResponseStream();

            StreamReader reader = new StreamReader(dataStream);
            string responseFromServer = reader.ReadToEnd();
            IEnumerable<Word>? iword =
JsonConvert.DeserializeObject<IEnumerable<Word>>(responseFromServer);
            Word[] aword = new List<Word>(iword).ToArray();
            Word word = aword[0];

            // makes sure there is only one definition for each word type
            Dictionary<string, string> definitions = new Dictionary<string, string>()
            {
                {"noun", "null"},
                {"pronoun", "null"},
                {"verb", "null"},
                {"adjective", "null"},
                {"adverb", "null"},
                {"preposition", "null"},
                {"conjunction", "null"},
                {"interjection", "null"},
            };

            // if the definition stored for the word type is null, then store the definiton
            for (int a = 0; a < word.meanings.Count; a++)
            {
                if (definitions[word.meanings[a].partOfSpeech] == "null")
                {
                    definitions[word.meanings[a].partOfSpeech] =
word.meanings[a].definitions[0].definition;
                }
            }

            // convert values in dictionary to list
            List<string> finals = new List<string>();
            foreach (KeyValuePair<string, string> kvPair in definitions)
            {
                if (kvPair.Value != "null")
                {
                    string key = char.ToUpper(kvPair.Key[0]) + kvPair.Key.Substring(1);
                    finals.Add(key + " - " + kvPair.Value);
                }
            }

            return finals;
        }

        static void Main(string[] args)
        {
            string word = Console.ReadLine();
            List<string> definitions = DefineWord(word);
            foreach (var definition in definitions)
            {
                Console.WriteLine(definition);
            }
        }
    }
}
```

5.3 View designers

5.3.1 Master view designer

```
///////////////////////////////////////////
// View designers/MasterView.Designer.cs //
///////////////////////////////////////////

// install dependencies
using Terminal.Gui;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class MasterView : Window
    {
        public void Build()
        {
            Title = "Spell Checker " + this;

            var menuBar = new MenuBar(new MenuBarItem[] {
                new MenuBarItem("_File", new MenuItem[] {
                    new MenuItem("_Quit", "", () => {
                        Application.Shutdown();
                    }),

                    new MenuItem("_Settings", "", () => {
                        Application.Run(new Settings());
                    }),

                    new MenuItem("_Main View", "", () => {
                        Application.Run(new MainView());
                    }),

                    new MenuItem("_Dictionary", "", () => {
                        Application.Run(new Dictionary());
                    })
                })
            });
            Add(menuBar);
        }
    }
}
```

5.3.2 Main view designer

```
/////////////////////////////////////////
// View designers/MainView.Designer.cs //
/////////////////////////////////////////

// install dependencies
using Terminal.Gui;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class MainView : MasterView
    {
        protected Button spellBtn = new Button();
        protected Button clearBtn = new Button();

        protected Label lengthLab = new Label();

        protected TextView textEntry = new TextView();
        protected void InitializeComponent()
        {
            //
            textEntry.X = 1;
            textEntry.Y = 2;
            textEntry.Width = 76;
            textEntry.Height = 5;
            textEntry.WordWrap = true;

            Add(textEntry);

            //
            spellBtn.Text = "Spell check";
            spellBtn.X = 1;
            spellBtn.Y = 20;

            clearBtn.Text = "Clear text";
            clearBtn.X = spellBtn.X + spellBtn.Text.Length + 6;
            clearBtn.Y = spellBtn.Y;

            Add(spellBtn, clearBtn);
        }
    }
}
```

5.3.3 Dictionary view designer

```
///////////////////////////////////////////
// View designers/Dictionary.Designer.cs //
///////////////////////////////////////////

// install dependencies
using Terminal.Gui;
using NStack;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class Dictionary : MasterView
    {
        protected Button lookupBtn = new Button();
        protected Button nextBtn = new Button();
        protected Button prevBtn = new Button();
        protected Button defineBtn = new Button();
        protected Button addBtn = new Button();
        protected Button removeBtn = new Button();
        protected Button filterBtn = new Button();

        protected Label[] wordLabs = new Label[101988];

        protected RadioGroup filterRad = new RadioGroup();

        protected TextField lookupTxt = new TextField();

        protected void InitializeComponent()
        {
            // build lookup elements
            lookupTxt.X = 50;
            lookupTxt.Y = 4;
            lookupTxt.Width = 14;

            lookupBtn.Text = "Lookup";
            lookupBtn.X = lookupTxt.X;
            lookupBtn.Y = lookupTxt.Y + 2;

            Add(lookupBtn, lookupTxt);

            // build navigation elements
            nextBtn.Text = "Next page";
            nextBtn.X = 64;
            nextBtn.Y = 21;

            prevBtn.Text = "Prev page";
            prevBtn.X = nextBtn.X - 14;
            prevBtn.Y = nextBtn.Y;

            Add(prevBtn, nextBtn);

            // build functional buttons
            defineBtn.X = lookupTxt.X;
            defineBtn.Y = lookupBtn.Y + 2;
            defineBtn.Text = "Define word";

            addBtn.X = lookupTxt.X;
            addBtn.Y = defineBtn.Y + 2;
            addBtn.Text = "Add word";

            removeBtn.X = lookupTxt.X;
            removeBtn.Y = addBtn.Y + 2;
            removeBtn.Text = "Remove word";

            Add(defineBtn, addBtn, removeBtn);

            // build fitering elements
            filterRad.X = lookupTxt.X;
            filterRad.Y = removeBtn.Y + 2;
            filterRad.RadioLabels = new ustring[] { "All words", "Default words", "Added words" };

            filterBtn.X = lookupTxt.X;
            filterBtn.Y = filterRad.Y + 4;
            filterBtn.Text = "Filter list";

            Add(filterRad, filterBtn);
        }
    }
}
```

5.3.4 Settings view designer

```
/////////////////////////////////////////
// View designers/Settings.Designer.cs //
/////////////////////////////////////////

// install dependencies
using Terminal.Gui;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class Settings : MasterView
    {
        protected Button viewDictBtn = new Button();
        protected Button saveBtn = new Button();
        protected Button resetBtn = new Button();

        protected CheckBox spellCheckCbx = new CheckBox();
        protected CheckBox changeAcronymsCbx = new CheckBox();
        protected CheckBox grammarCheckCbx = new CheckBox();

        protected ColorPicker backgroundPick = new ColorPicker();
        protected ColorPicker foregroundPick = new ColorPicker();

        protected Label spellCheckLab = new Label();
        protected Label viewDictLab = new Label();
        protected Label maxLengthLab = new Label();
        protected Label changeAcryonymsLab = new Label();
        protected Label grammarCheckLab = new Label();

        protected TextField maxLengthTxt = new TextField();

        protected void InitializeComponent()
        {
            // build labels
            spellCheckLab.Text = "Spell check";
            spellCheckLab.X = 2;
            spellCheckLab.Y = 2;

            viewDictLab.Text = "View dictionary";
            viewDictLab.X = spellCheckLab.X;
            viewDictLab.Y = spellCheckLab.Y + 2;

            maxLengthLab.Text = "Maximum length                For no max length, enter \"0\"";
            maxLengthLab.X = spellCheckLab.X;
            maxLengthLab.Y = viewDictLab.Y + 2;

            changeAcryonymsLab.Text = "Change acronyms";
            changeAcryonymsLab.X = spellCheckLab.X;
            changeAcryonymsLab.Y = maxLengthLab.Y + 2;

            grammarCheckLab.Text = "Grammar check";
            grammarCheckLab.X = spellCheckLab.X;
            grammarCheckLab.Y = changeAcryonymsLab.Y + 2;

            Add(spellCheckLab, viewDictLab, maxLengthLab, changeAcryonymsLab, grammarCheckLab);

            // build check boxes
            spellCheckCbx.X = spellCheckLab.X + 20;
            spellCheckCbx.Y = spellCheckLab.Y;
            spellCheckCbx.Checked = spellCheck;

            changeAcronymsCbx.X = spellCheckCbx.X;
            changeAcronymsCbx.Y = changeAcryonymsLab.Y;
            changeAcronymsCbx.Checked = changeAcronyms;

            grammarCheckCbx.X = spellCheckCbx.X;
            grammarCheckCbx.Y = grammarCheckLab.Y;
            grammarCheckCbx.Checked = grammarCheck;

            Add(spellCheckCbx, changeAcronymsCbx, grammarCheckCbx);

            // build buttons
            viewDictBtn.Text = "View";
            viewDictBtn.X = spellCheckCbx.X;
            viewDictBtn.Y = viewDictLab.Y;

            saveBtn.Text = "_Save";
            saveBtn.X = maxLengthLab.X;
            saveBtn.Y = grammarCheckLab.Y + 2;
            saveBtn.IsDefault = true;

            resetBtn.X = maxLengthLab.X;
            resetBtn.Y = saveBtn.Y + 2;
            resetBtn.Text = "Reset to default settings";

            Add(viewDictBtn, saveBtn, resetBtn);

            // build text fields
            maxLengthTxt.X = spellCheckCbx.X;
            maxLengthTxt.Y = maxLengthLab.Y;
            maxLengthTxt.Width = 8;
            maxLengthTxt.Text = maxLength.ToString();

            Add(maxLengthTxt);

            // build color elements
            backgroundPick.X = maxLengthLab.X;
            backgroundPick.Y = resetBtn.Y + 2;
            backgroundPick.Text = "Background colour";
            backgroundPick.SelectedColor = background;

            foregroundPick.X = 38;
            foregroundPick.Y = backgroundPick.Y;
            foregroundPick.Text = "Foreground colour";
            foregroundPick.SelectedColor = foreground;

            Add(backgroundPick, foregroundPick);
        }
    }
}
```

5.3.5 Spell check view designer

```
///////////////////////////////////////////
// View designers/SpellCheck.Designer.cs //
///////////////////////////////////////////
```

```
// install dependencies
using Terminal.Gui;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class SpellCheck : MasterView
    {
        protected Button ignoreBtn = new Button();
        protected Button userBtn = new Button();
        protected Button acronymBtn = new Button();
        protected Button homeBtn = new Button();
        protected Button homeBtn2 = new Button();
        protected Button replaceBtn = new Button();
        protected Button addBtn = new Button();
        protected Button resizeBtn = new Button();

        protected Label lookupLab = new Label();
        protected Label lookupLab2 = new Label();
        protected Label lookupLab3 = new Label();
        protected Label userLab = new Label();
        protected List<Label> wordLabs = new List<Label>();

        protected TextField lookupTxt = new TextField();
        protected TextField userTxt = new TextField();

        protected void InitializeComponent()
        {
            // build lookup elements
            lookupLab.Text = "Enter word from left side";
            lookupLab.X = 50;
            lookupLab.Y = 2;

            lookupLab2.Text = "in the field below.";
            lookupLab2.X = lookupLab.X;
            lookupLab2.Y = lookupLab.Y + 1;

            lookupLab3.Text = "Then handle error";
            lookupLab3.X = lookupLab.X;
            lookupLab3.Y = lookupLab2.Y + 1;

            lookupTxt.X = lookupLab.X;
            lookupTxt.Y = lookupLab3.Y + 1;
            lookupTxt.Width = 14;

            Add(lookupLab, lookupLab2, lookupLab3, lookupTxt, lookupTxt);

            //
            ignoreBtn.X = lookupLab.X;
            ignoreBtn.Y = lookupTxt.Y + 2;
            ignoreBtn.Text = "Ignore error";

            Add(ignoreBtn);

            //
            resizeBtn.Text = "Resize text to " + theseSettings.maxLength;
            resizeBtn.X = lookupLab.X;
            resizeBtn.Y = ignoreBtn.Y + 2;

            Add(resizeBtn);

            // build change error elements
            userLab.Text = "Enter new word";
            userLab.X = lookupLab.X;
            userLab.Y = resizeBtn.Y + 2;

            userTxt.X = lookupLab.X;
            userTxt.Y = userLab.Y + 1;
            userTxt.Width = 14;

            userBtn.Text = "Replace";
            userBtn.X = lookupLab.X;
            userBtn.Y = userTxt.Y + 1;

            replaceBtn.Text = "Recommend words";
            replaceBtn.X = lookupLab.X;
            replaceBtn.Y = userBtn.Y + 2;

            Add(userLab, userTxt, userBtn, replaceBtn);

            //
            acronymBtn.Text = "Replace acronyms";
            acronymBtn.X = lookupLab.X;
            acronymBtn.Y = replaceBtn.Y + 2;

            Add(acronymBtn);

            //
            addBtn.Text = "Learn spelling";
            addBtn.X = lookupLab.X;
            addBtn.Y = acronymBtn.Y + 2;

            Add(addBtn);

            //
            homeBtn.Text = "Home";
            homeBtn.X = lookupLab.X;
            homeBtn.Y = addBtn.Y + 2;

            Add(homeBtn);
        }
    }
}
```

5.4 Views

5.4.1 Master view

```
/////////////////////////
// Views/MasterView.cs //
/////////////////////////

// install dependencies

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class MasterView
    {
        public MasterView()
        {
            Build();
        }
    }
}
```

5.4.2 Main view

```
///////////////////////
// Views/MainView.cs //
///////////////////////

// install dependencies
using Terminal.Gui;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class MainView
    {
        protected int length = 0;

        public MainView()
        {
            BuildAll();
        }
        // used when coming from the spell check view
        // so the text field will have all the corrected words
        public MainView(string text)
        {
            BuildAll();

            textEntry.Text = text;
        }

        public void BuildAll()
        {

            InitializeComponent();

            spellBtn.Clicked += () =>
            {
                string text = textEntry.Text.ToString();

                if (text is null)
                {
                    MessageBox.Query("Spell check complete", "No incorrect words", "OK");
                }
                else
                {
                    Application.Run(new SpellCheck(text));
                }
            };

            clearBtn.Clicked += () =>
            {
                textEntry.Text = "";

                MessageBox.Query("DONE", "DONE", "DONE");
                Utilities.BloomFilter bloom = new Utilities.BloomFilter(1);
                MessageBox.Query("DONE", "DONE", "DONE");
            };
        }
    }
}
```

5.4.3 Dictionary view

```
/////////////////////////
// Views/Dictionary.cs //
/////////////////////////

// install dependencies
using Terminal.Gui;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class Dictionary
    {
        public string[] words;
        protected string? currentLookup;
        protected List<string> toDisplay = new List<string>();

        protected int firstIndex;
        protected int currentFilter;


        public Dictionary()
        {
            words = FetchAllWords();
            firstIndex = 0;
            currentFilter = filterRad.SelectedItem;

            UpdateWords();

            InitializeComponent();

            lookupBtn.Clicked += () =>
            {
                Lookup(0);
            };

            nextBtn.Clicked += () =>
            {
                NextPage();
            };

            prevBtn.Clicked += () =>
            {
                PrevPage();
            };

            defineBtn.Clicked += () =>
            {
                currentLookup = lookupTxt.Text.ToString();

                if (currentLookup is not null)
                {
                    DefineThis(currentLookup);
                }
                else
                {
                    MessageBox.ErrorQuery("Error", "There is no word to define\nEnter a value in the
lookup field", "Ok");
                }
            };

            addBtn.Clicked += () =>
            {
                currentLookup = lookupTxt.Text.ToString();

                if (currentLookup is not null && !currentLookup.Contains(" "))
                {
                    AddThis(currentLookup);
                    UpdateWords();
                }
                else
                {
                    MessageBox.ErrorQuery("Error", "There is no word to add to the dictionary\nEnter
a value in the lookup field", "Ok");
                }
            };

            removeBtn.Clicked += () =>
            {
                currentLookup = lookupTxt.Text.ToString();
                if (currentLookup is not null)
                {
                    RemoveThis(currentLookup);
                    UpdateWords();
                }
                else
                {
                    MessageBox.ErrorQuery("Error", "There is no word to remove fomr the
dictionary\nEnter a value in the lookup field", "Ok");
                }
            };

            filterBtn.Clicked += () =>
            {
                currentFilter = filterRad.SelectedItem;
                UpdateWords();
            };
        }

        protected string[] FetchAllWords()
        {
            string defPath = "Resources/TrueWords.txt";
            string addedPath = "Resources/AddedWords.txt";

            List<string> words = File.ReadAllLines(defPath).ToList();
            List<string> addedWords = File.ReadAllLines(addedPath).ToList();
            words.AddRange(addedWords);
            words.Sort();
            string[] values = words.ToArray();

            return values;
        }
        protected string[] FetchWords(int num)
        {
            string path;

            if (num == 1)
            {
                path = "Resources/TrueWords.txt";
            }
            else
            {
                path = "Resources/AddedWords.txt";
            }

            words = File.ReadAllLines(path);
            return words;
        }

        protected void NextPage()
        {
            firstIndex += 20;
            if ((toDisplay is null && firstIndex > words.Length) || (toDisplay is not null &&
firstIndex > toDisplay.Count))
            {
                firstIndex -= 20;
            }

            Lookup(firstIndex);
        }

        protected void PrevPage()
        {
            if (firstIndex != 0)
            {
                firstIndex = firstIndex - 20;
            }

            Lookup(firstIndex);
        }

        protected void Lookup(int firstIndexIn)
        {
            firstIndex = firstIndexIn;
            currentLookup = lookupTxt.Text.ToString();

            UpdateWords();
        }

        protected void UpdateWords()
        {
            if (currentFilter == 0)
            {
                FetchAllWords();
            }
            else
            {
                FetchWords(currentFilter);
            }

            foreach (var a in wordLabs)
            {
                Remove(a);
            }
            toDisplay.Clear();

            if (currentLookup is null)
            {
                toDisplay = words.ToList();
            }

            else
            {
                for (int a = 0; a < words.Length; a++)
                {
                    if (words[a].StartsWith(currentLookup))
                    {
                        toDisplay.Add(words[a]);
                    }
                }
            }

            DisplayLabels(toDisplay);
        }

        protected void DisplayLabels(List<string> toDisplay)
        {
            int yPos = 2;
            int count = 0;
            int thisIndex = firstIndex;
            int lastIndex = firstIndex + 20;

            while (thisIndex < lastIndex)
            {
                if (thisIndex < toDisplay.Count)
                {
                    wordLabs[count] = new Label();
                    wordLabs[count].X = 2;
                    wordLabs[count].Y = yPos;
                    wordLabs[count].Text = toDisplay[thisIndex];

                    Add(wordLabs[count]);
                }
                else
                {
                    break;
                }

                count++;
                yPos++;
                thisIndex++;
            }
        }

        protected void DefineThis(string word)
        {
            try
            {
                List<string> definitions = DefineWords.Program.DefineWord(word);

                string definitionstring = "";
                for (int a = 0; a < definitions.Count; a++)
                {
                    definitionstring += definitions[a];
                    definitionstring += "\n";
                }

                MessageBox.Query("Definitions of " + word, definitionstring, "Ok");
            }
            catch (Exception e)
            {
                MessageBox.ErrorQuery("Error", "Word could not be defined\n" + e.Message, "Ok");
            }
        }

        protected void AddThis(string word)
        {
            string path = "Resources/AddedWords.txt";
            string[] words = File.ReadAllLines(path);

            if (!words.Contains(word))
            {
                List<string> newWords = new List<string>();

                newWords = words.ToList();
                newWords.Add(word);
                newWords.Sort();

                string[] toWrite = newWords.ToArray();
                File.WriteAllLines(path, toWrite);
            }

            MessageBox.Query("Word added", word + " has been added to the dictionary", "Ok");
        }

        protected void RemoveThis(string word)
        {
            string path = "Resources/AddedWords.txt";
            string[] words = File.ReadAllLines(path);

            if (words.Contains(word))
            {
                List<string> newWords = new List<string>();

                newWords = words.ToList();
                newWords.Remove(word);
                newWords.Sort();

                string[] toWrite = newWords.ToArray();
                File.WriteAllLines(path, toWrite);
            }

            MessageBox.Query("Word removed", word + " has been removed from the dictionary", "Ok");
        }

        protected void FilterList(int choice)
        {
            if (choice == 1)
            {
                string path = "Resources/TrueWords.txt";
                string[] words = File.ReadAllLines(path);
            }
            if (choice == 2)
            {
                string path = "Resources/AddedWords.txt";
                string[] words = File.ReadAllLines(path);
            }

            UpdateWords();
        }
    }
}
```

5.4.4 Settings view

```
///////////////////////
// Views/Settings.cs //
///////////////////////

// install dependencies
using Terminal.Gui;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class Settings
    {
        public bool spellCheck;
        public bool changeAcronyms;
        public bool grammarCheck;

        public int maxLength;

        public Color background;
        public Color foreground;

        public Settings()
        {
            FetchSettings();

            InitializeComponent();

            saveBtn.Clicked += () =>
            {
                SaveSettings();
            };
            viewDictBtn.Clicked += () =>
            {
                Application.Run(new Dictionary());
            };
            resetBtn.Clicked += () =>
            {
                ResetSettings();
            };
        }

        protected void SaveSettings()
        {
            spellCheck = spellCheckCbx.Checked;
            changeAcronyms = changeAcronymsCbx.Checked;
            grammarCheck = grammarCheckCbx.Checked;

            if (!int.TryParse(maxLengthTxt.Text.ToString(), out maxLength))
            {
                MessageBox.ErrorQuery("Error", "The input for max length must be a whole number\nFor
no max length, enter 0", "Ok");

                return;
            }

            string settings = "Spell " + spellCheck + "\n"
                + "Lengt " + maxLength + "\n"
                + "Acron " + changeAcronyms + "\n"
                + "Grama " + grammarCheck + "\n"
                + "Foreg " + foregroundPick.SelectedColor + "\n"
                + "Backg " + backgroundPick.SelectedColor + "\n"
                ;

            string path = "Resources/Settings.txt";
            File.WriteAllText(path, settings);

            FetchSettings();
            MessageBox.Query("Saved", "Settings saved successfully", "Ok");
        }

        public void FetchSettings()
        {
            string path = "Resources/Settings.txt";
            string[] settings = File.ReadAllLines(path);

            spellCheck = bool.Parse(settings[0].Substring(6));
            maxLength = int.Parse(settings[1].Substring(6));
            changeAcronyms = bool.Parse(settings[2].Substring(6));
            grammarCheck = bool.Parse(settings[3].Substring(6));
            foreground = GetColour(settings[4].Substring(6));
            background = GetColour(settings[5].Substring(6));

            Colors.Base.Normal = Application.Driver.MakeAttribute(foreground, background);
        }

        protected void ResetSettings()
        {
            spellCheck = true;
            maxLength = 0;
            changeAcronyms = true;
            grammarCheck = false;
            foreground = Color.Gray;
            background = Color.Blue;

            string settings = "Spell " + spellCheck + "\n"
                + "Lengt " + maxLength + "\n"
                + "Acron " + changeAcronyms + "\n"
                + "Grama " + grammarCheck + "\n"
                + "Foreg " + foreground + "\n"
                + "Backg " + background + "\n"
                ;

            string path = "Resources/Settings.txt";
            File.WriteAllText(path, settings);

            FetchSettings();
            MessageBox.Query("Reset", "Settings reset to default successfully", "Ok");
        }

        protected Color GetColour(string colourStr)
        {
            switch (colourStr)
            {
                case "Black":
                    return Color.Black;
                case "Blue":
                    return Color.Blue;
                case "BrightBlue":
                    return Color.BrightBlue;
                case "BrightCyan":
                    return Color.BrightCyan;
                case "BrightGreen":
                    return Color.BrightGreen;
                case "BrightMagenta":
                    return Color.BrightMagenta;
                case "BrightRed":
                    return Color.BrightRed;
                case "BrightYellow":
                    return Color.BrightYellow;
                case "Brown":
                    return Color.Brown;
                case "Cyan":
                    return Color.Cyan;
                case "DarkGray":
                    return Color.DarkGray;
                case "Gray":
                    return Color.Gray;
                case "Green":
                    return Color.Green;
                case "Magenta":
                    return Color.Magenta;
                case "Red":
                    return Color.Red;
                case "White":
                    return Color.White;
            }

            return Color.Black;
        }
    }
}
```

5.4.5 Spell check view

```
/////////////////////////
// Views/SpellCheck.cs //
/////////////////////////

// install dependencies
using Terminal.Gui;
using Utilities;
using System.Text.RegularExpressions;
using NStack;

// suppress warnings
# pragma warning disable

namespace Views
{
    public partial class SpellCheck
    {
        protected int firstIndex;

        protected string? error;
        protected List<string> falseWords = new List<string>();
        protected List<string> words = new List<string>();
        protected List<string> ignoredWords = new List<string>();

        protected Settings theseSettings = new Settings();

        public SpellCheck(string textIn)
        {
            words.Clear();
            falseWords.Clear();
            words = textIn.Split(' ').ToList();

            InitializeComponent();
            DisplayWords();

            if (falseWords.Count == 0)
            {
                LeaveMessage();
            }
            else if (falseWords[0] == "")
            {
                LeaveMessage();
            }

            ignoreBtn.Clicked += () =>
            {
                error = lookupTxt.Text.ToString();

                if (error is not null)
                {
                    ignoredWords.Add(error);

                    MessageBox.Query("Ignored", error + " has been ignored as an erorr", "Ok");

                    DisplayWords();
                }
            };

            acronymBtn.Clicked += () =>
            {
                Settings theseSettings = new Settings();

                if (theseSettings.changeAcronyms)
                {
                    ReplaceAcronyms();

                    MessageBox.Query("Acronyms replaced", "All acronyms have been replaced", "Ok");

                    DisplayWords();
                }
            };

            homeBtn.Clicked += () =>
            {
                GoHome();
            };

            homeBtn2.Clicked += () =>
            {
                GoHome();
            };

            userBtn.Clicked += () =>
            {
                error = lookupTxt.Text.ToString();
                string newWord = userTxt.Text.ToString();

                if (newWord is not null)
                {
                    ReplaceUserWord(newWord);
                }

                MessageBox.Query("Replaced", error + " has been replaced to " + newWord, "Ok");

                DisplayWords();
            };

            replaceBtn.Clicked += () =>
            {
                error = lookupTxt.Text.ToString();
                string newWord = "";
                int maxWeight = 1;
                bool loop = true;
                string text;

                if (error is not null && falseWords.Contains(error))
                {
                    BkTree bktree = new BkTree(error);

                    do
                    {
                        string[] stringButtons = bktree.ReturnClosest(maxWeight);
                        ustring[] buttons = new ustring[stringButtons.Length + 1];
                        text = "";

                        for (int a = 0; a < stringButtons.Length; a++)
                        {
                            buttons[a] = (a + 1).ToString();
                            text += (a + 1) + ": " + stringButtons[a] + "    ";
                        }

                        text += "\nPress escape to exit";
                        buttons[buttons.Length - 1] = "More words";

                        int choice = MessageBox.Query("Recommended words: " + buttons.Length, text, 
buttons);

                        if (choice == -1)
                        {
                            return;
                        }
                        else if (choice == buttons.Length - 1)
                        {
                            maxWeight++;
                        }
                        else
                        {
                            newWord = stringButtons[choice];
                            words[words.IndexOf(error)] = newWord;
                            falseWords.Remove(error);
                            loop = false;

                            MessageBox.Query("Replaced", error + " has been replaced to " + newWord, 
"Ok");
                        }
                    } while (loop);
                }
                else if (!falseWords.Contains(error))
                {
                    MessageBox.ErrorQuery("Error", error + " is not in the false words list\nMake 
sure it is spelt correctly and in the list", "Ok");
                }

                DisplayWords();
            };

            addBtn.Clicked += () =>
            {
                error = lookupTxt.Text.ToString();

                using (StreamWriter writer = new StreamWriter("Resources/AddedWords.txt"))
                {
                    writer.Write(error + "\n");
                }

                MessageBox.Query("Spelling learnt", "The spelling of " + error + " has been learnt", 
"Ok");

                DisplayWords();
            };

            resizeBtn.Clicked += () =>
            {
                ResizeText();

                MessageBox.Query("Text resized", "Text has been resized to " + 
theseSettings.maxLength, "Ok");

                DisplayWords();
            };
        }

        protected void CheckSpelling()
        {
            Utilities.BloomFilter bloom = new Utilities.BloomFilter();
            falseWords.Clear();

            // check all words in the bloom filter
            if (words.Count != 0)
            {
                for (int a = 0; a < words.Count; a++)
                {
                    if (!bloom.Lookup(words[a]))
                    {
```

```
                    falseWords.Add(words[a]);
                }
            }
        }

        if (falseWords.Count != 0)
        {
            // check all wrong words in the text file of added words
            foreach (var a in falseWords)
            {
                if (bloom.BinarySearch(a))
                {
                    falseWords.Remove(a);
                }
            }

            // check all wrong words againt the array of ignored words
            for (int a = 0; a < falseWords.Count; a++)
            {
                for (int b = 0; b < ignoredWords.Count; b++)
                {
                    if (falseWords[a] == ignoredWords[b])
                    {
                        falseWords.Remove(ignoredWords[b]);
                    }
                }
            }
        }
    }

    protected void CheckGrammar()
    {
        int a = 0;

        while (a < words.Count)
        {
            string word = words[a];

            Match match = Regex.Match(word, @"[^\w\s]");

            // if a word contains a special character
            if (match.Success && word.Length != 1 && match.Value != "'")
            {
                string specialChar = word[match.Index].ToString();
                string part1 = "";
                string part2 = "";
                // checks either side of the special character to ensure it is not null
                //  then stores the values to new strings
                if (word[0] != char.Parse(specialChar))
                {
                    part1 = word.Substring(0, match.Index);
                }
                if (word.IndexOf(specialChar) != (word.Length - 1))
                {
                    part2 = word.Substring(match.Index + 1);
                }

                // creates a sub list
                List<string> postWords = new List<string>();
                if (match.Index != (words.Count - 1))
                {
                    postWords = words.GetRange(a + 1, words.Count - a - 1);
                }
                // removes all words after & including special character
                words.RemoveRange(a, words.Count - a);
                // adds the parts containing & including special character
                if (part1 is not null)
                {
                    words.Add(part1);
                }
                words.Add(specialChar);
                if (part2 is not null)
                {
                    words.Add(part2);
                }
                // concats both lists together
                words.AddRange(postWords);
                a += 2;
            }

            a++;
        }

        falseWords.RemoveAll(item => string.IsNullOrWhiteSpace(item));

        for (int b = 0; b < falseWords.Count; b++)
        {
            Match match = Regex.Match(falseWords[b], @"[^\w]");

            if (match.Success)
            {
                falseWords.Remove(falseWords[b]);
            }
        }
    }

    protected void LeaveMessage()
    {
        RemoveAll();

        Label message1 = new Label();
        message1.Text = "Spell check complete";
        message1.X = Pos.Center();
        message1.Y = Pos.Center() - 1;

        Label message2 = new Label();
        message2.Text = "No errors found".PadLeft(message1.Text.ToString().Length / 2);
        message2.X = message1.X;
        message2.Y = message1.Y + 1;

        homeBtn.Text = "Home";
        homeBtn.X = message1.X;
        homeBtn.Y = message2.Y + 2;

        Add(message1, message2, homeBtn);
    }

    protected void DisplayWords()
    {
        // move any special characters to own index
        CheckGrammar();
        // find false words
        if (theseSettings.spellCheck)
        {
            CheckSpelling();
            CheckGrammar();
        }

        if (falseWords.Count == 0)
        {
            LeaveMessage();
        }
        else if (falseWords[0] == "")
        {
            LeaveMessage();
        }

        int xPos = 2;
        int yPos = 2;
        int thisIndex = firstIndex;

        foreach (var a in wordLabs)
        {
            Remove(a);
        }
        wordLabs.Clear();

        for (int a = 0; a < 41; a++)
        {
            if (thisIndex < falseWords.Count && ignoredWords.Contains(falseWords[thisIndex]) ==
false)
            {
                wordLabs.Add(new Label());
                wordLabs[a] = new Label();
                wordLabs[a].X = xPos;
                wordLabs[a].Y = yPos;
                wordLabs[a].Text = falseWords[thisIndex];

                Add(wordLabs[a]);

                yPos++;
                thisIndex++;
            }

            if (yPos == 22)
            {
                xPos = 30;
                yPos = 2;
            }
        }
    }

    protected void ReplaceAcronyms()
    {
        string path = "Resources/Acronyms.txt";
        string[] acronyms = File.ReadAllLines(path);
        string[] shortAcr = new string[acronyms.Length];
        string[] fullAcr = new string[acronyms.Length];

        for (int a = 0; a < acronyms.Length; a++)
        {
            string[] parts = acronyms[a].Split('*', 2);
            shortAcr[a] = parts[0];
            fullAcr[a] = parts[1];
        }

        for (int a = 0; a < falseWords.Count; a++)
        {
            for (int b = 0; b < acronyms.Length; b++)
            {
                if (falseWords[a] == shortAcr[b])
                {
                    falseWords[a] = fullAcr[b];

                    int index = words.IndexOf(shortAcr[b]);
                    words[index] = falseWords[a];
                }
            }
        }
    }

    protected void ReplaceUserWord(string newWord)
    {
        int index = falseWords.IndexOf(error);

        if (index != -1)
        {
            falseWords[index] = newWord;
            words[index] = newWord;
        }
    }

    protected void ResizeText()
    {
        string text = "";

        foreach (var a in words)
        {
            text += a + " ";
        }

        if (theseSettings.maxLength != 0)
        {
            text = text.Substring(0, theseSettings.maxLength);
        }

        words = text.Split(' ').ToList();
    }

    protected void GoHome()
    {
        string text = "";
        foreach (var a in words)
        {
            text += a + " ";
        }

        text = FixGrammar(text);

        Application.Run(new MainView(text));
    }

    protected string FixGrammar(string text)
    {
        int speechCount = 0;

        for (int a = 0; a < text.Length; a++)
        {
            switch (text[a])
            {
                // need character before and space after
                case '.':
                case ',':
                case '!':
                case '?':
                case ':':
                case ';':
                case ')':

                    // before
                    if (a != 0)
                    {
                        if (char.IsWhiteSpace(text[a - 1]))
                        {
                            text = text.Remove(a - 1, 1);
                        }
                    }

                    // after
                    if (a != text.Length - 1)
                    {
                        if (!char.IsWhiteSpace(text[a + 1]))
                        {
                            string tempText = text.Substring(a + 1);
                            text = text.Substring(0, a + 1);
                            text += " ";
                            text += tempText;
                        }
                    }

                    break;

                // need character before and after
                case '/':

                    // before
                    if (a != 0)
                    {
                        if (char.IsWhiteSpace(text[a - 1]))
                        {
                            text = text.Remove(a - 1, 1);
                        }
                    }

                    // after
                    if (a != text.Length - 1)
                    {
                        if (char.IsWhiteSpace(text[a]))
                        {
                            text = text.Remove(a, 1);
                        }
                    }

                    break;

                // need space before and character after
                case '(':

                    // before
                    if (a != 0)
```

```
                        {
                            if (char.IsWhiteSpace(text[a - 1]))
                            {
                                string tempText = text.Substring(a);
                                text = text.Substring(0, a - 1);
                                text += " ";
                                text += tempText;
                            }
                        }

                        // after
                        if (a != text.Length - 1)
                        {
                            if (char.IsWhiteSpace(text[a + 1]))
                            {
                                text = text.Remove(a + 1, 1);
                            }
                        }

                        break;
                    }
                }

                return text;
            }
        }
}
```

5.5 Program.cs

```
////////////////
// Program.cs //
////////////////

// install dependencies
using Views;
using Terminal.Gui;

// suppress warnings
# pragma warning disable

Application.Init();

try
{
    // used to fetch settings from saved file
    Settings thisSettings = new Settings();

    Application.Run(new MainView());
}
finally
{
    Application.Shutdown();
}
```

5.6 FINAL.csproj

```
<!--//////////////-->
<!-- FINAL.csproj -->
<!--//////////////-->

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Newtonsoft.Json" Version="13.0.1" />
    <PackageReference Include="Terminal.Gui" Version="1.*" />
  </ItemGroup>

</Project>
```

5.7 Filter word lists

```python
from fileinput import filename
from urllib import response
import requests
import datetime
from datetime import datetime

now = datetime.now()
print(now)
print("Start")

trueWords = []
falseWords = []
checkWords = []

readName = "/home/pi/Desktop/words_alpha.txt"
f = open(readName, "r")

now = datetime.now()
print(now)
print("Test")

for a in f:
    try:
        word = a

        URL = "https://api.dictionaryapi.dev/api/v2/entries/en/" + word
        response = requests.get(URL)
        statusCode = response.status_code

        #check the status code
        if statusCode < 400:
            #OK - assume the word is a true word
            trueWords.append(a)
        elif statusCode == 404:
            #Not Found - assume the word is a false word
            falseWords.append(a)
        else:
            #all other status codes
            #check manually later/retry
            checkWords.append(a)
    except:
        now = datetime.now()
        print(now)
        print("An error occurred! - ", word)
        checkWords.append(a)
f.close()

now = datetime.now()
print(now)
print("Write")

f = open("/home/pi/Desktop/TrueWords.txt", "w")
for a in trueWords:
    f.write(a)
f.close()

f = open("/home/pi/Desktop/CheckWords.txt", "w")
for a in checkWords:
    f.write(a)
f.close()

f = open("/home/pi/Desktop/FalseWords.txt", "w")
for a in falseWords:
    f.write(a)
f.close()
```

6 References

Ref. no.	Page no.	Reference name	Reference source
1	5	Grammarly	https://www.grammarly.com
2	5	Online-spellcheck	https://www.online-spellcheck.com
3	7	Triangle inequality definition	https://en.wikipedia.org/wiki/Triangle_inequality
4	7	Example of the triangle inequality	https://en.wikipedia.org/wiki/File:TriangleInequality.svg
5	7	Metric definition	https://en.wikipedia.org/wiki/Metric_(mathematics)
6	7	String metric definition	https://en.wikipedia.org/wiki/String_metric
7	7	Metric space definition	https://en.wikipedia.org/wiki/Metric_space
8	8	Levenshtein distance	https://en.wikipedia.org/wiki/Levenshtein_distance
9	8	Equation for the Levenshtein distance between two strings	https://wikimedia.org/api/rest_v1/media/math/render/svg/6224efffbe9a4e01afbddeeb900bfd1b3350b335
10	8	Bloom filter	https://en.wikipedia.org/wiki/Bloom_filter
11	8	Example of a Bloom Filter	https://en.wikipedia.org/wiki/File:Bloom_filter.svg
12	10	BK-tree	https://en.wikipedia.org/wiki/BK-tree
13	10	Example of a BK-tree	https://commons.wikimedia.org/wiki/File:Bk_tree.svg#/media/File:Bk_tree.svg
14	10	Hamming distance	https://en.wikipedia.org/wiki/Hamming_distance
15	12	Bloom filter calculator	https://hur.st/bloomfilter/?n=101988&p=1.0E-5&m=&k=10
16	12	Optimal size of the array in the Bloom filter as calculated	https://hur.st/bloomfilter/?n=101988&p=1.0E-5&m=&k=10
17	13	List of hash functions	https://en.wikipedia.org/wiki/List_of_hash_functions
18	13	Pearson hashing	https://en.wikipedia.org/wiki/Pearson_hashing
19	13	Hashing by cyclic polynomial (Buzhash)	https://en.wikipedia.org/wiki/Rolling_hash#Cyclic_polynomial
20	13	Fowler-Noll-Vo (FNV-0) hash	https://en.wikipedia.org/wiki/Fowler–Noll–Vo_hash_function

21	13	djb2 hash	http://www.cse.yorku.ca/~oz/hash.html
22	13	sdbm hash	http://www.cse.yorku.ca/~oz/hash.html
23	13	PJW hash function	https://en.wikipedia.org/wiki/PJW_hash_function
24	13	Fast-Hash	https://en.wikipedia.org/wiki/Jenkins_hash_function
25	13	Rabin fingerprint	https://en.wikipedia.org/wiki/Rabin_fingerprint
26	13	Fletcher-32 hash	https://en.wikipedia.org/wiki/Fletcher%27s_checksum
27	13	CRC32 hash	https://en.wikipedia.org/wiki/CRC-32