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

0 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 and save preferences.

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.

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

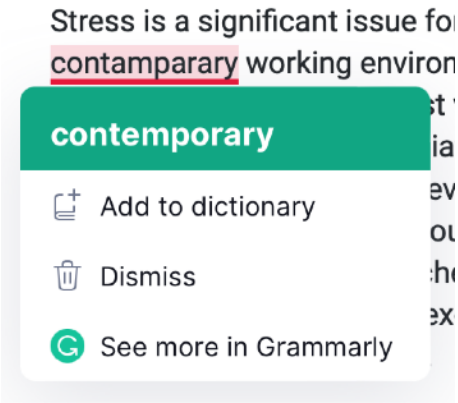
1.4.1 Existing and similar solutions

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.



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").

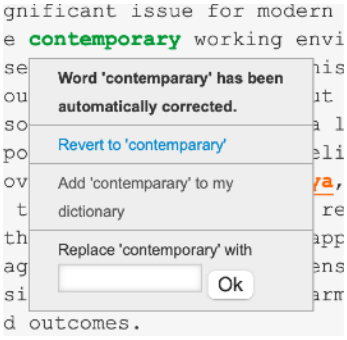
To summarise, Grammarly offers some more advanced features (such as formality and structure checking) but it does lack on some of the basic features (such as preference saving). Also, there can be some compatibility issues when attempting to use, and Grammarly doesn't have the option to define words.

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.



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

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.

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.

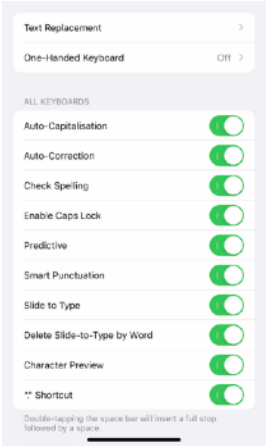
Users can save their dictionary by creating an account and logging in. It can also automatically correct some incorrectly spelt words, if the initial spelling was close enough to the correct word.

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.



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.

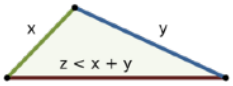
None of the spell checkers I have looked at allow the user to define words, view the dictionary or change features like the font colour.

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.

The Levenshtein distance is also sometimes known as edit

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

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]

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 .

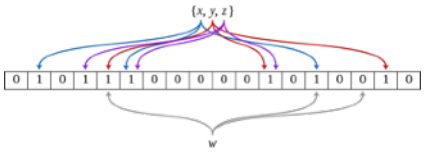
There are some bounds for a Levenshtein distance

- 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 an effective application. Blue cells represent cells that have been edited from the last stage. Green cells represent cells that have been checked.

- 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

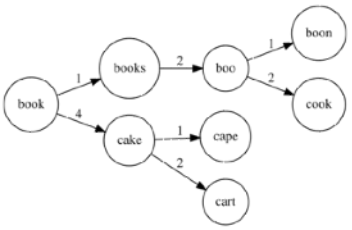
```
BloomFilter[0] = 1
BloomFilter[4] = 1
BloomFilter[8] = 1
```

- 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.

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.



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

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.

Pseudocode representation

```
function hamming distance (word1, word2)
        distance ← 0

        for each letter in word1
                if letter of word1 is not letter of word2
                        increase distance by 1
        return distance
end function
```

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**0**1**0** and 1**1**1**1** 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 a have a few more personalisation features.

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 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

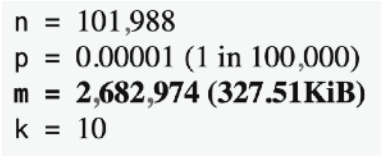
Bloom filters can be used to efficiently store and lookup elements in a set. This is because instead of storing the actual elements a Bloom filter stores an array of ones and zeros. To lookup an element it is uniquely hashed multiple times. If all of the indexes at the results of the hashes are one, 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. In the final implementation of the Bloom filter, I will use a larger bit array for better accuracy.



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

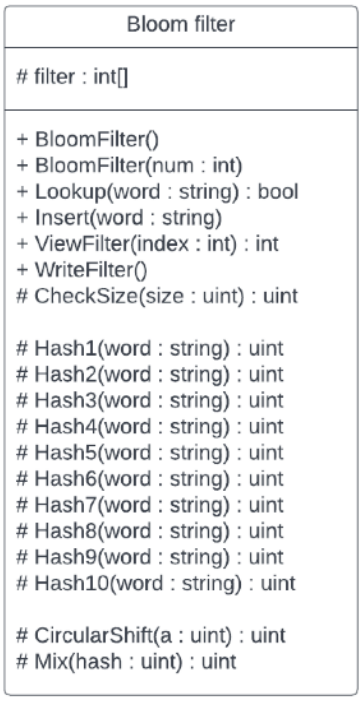
This method is used by all the hash functions to reduce the size of the hash to one that it less than the filter size. It works by using an if statement nested inside a do loop (lines 80-90) to test if the value is greater than the filter size. If it is, then. It performs a modulus function on the hash. 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 0 - 4,294,967,295). 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 4) [21]
- sdbm (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 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, then I will type out the recorded transcript.

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 from a-z; search in the dictionary; filter it by words I've added; and define words 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.

These objectives should solve the problem that there isn't currently an efficient and easy to use spell checker, that allows for personalisation by the user.

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:

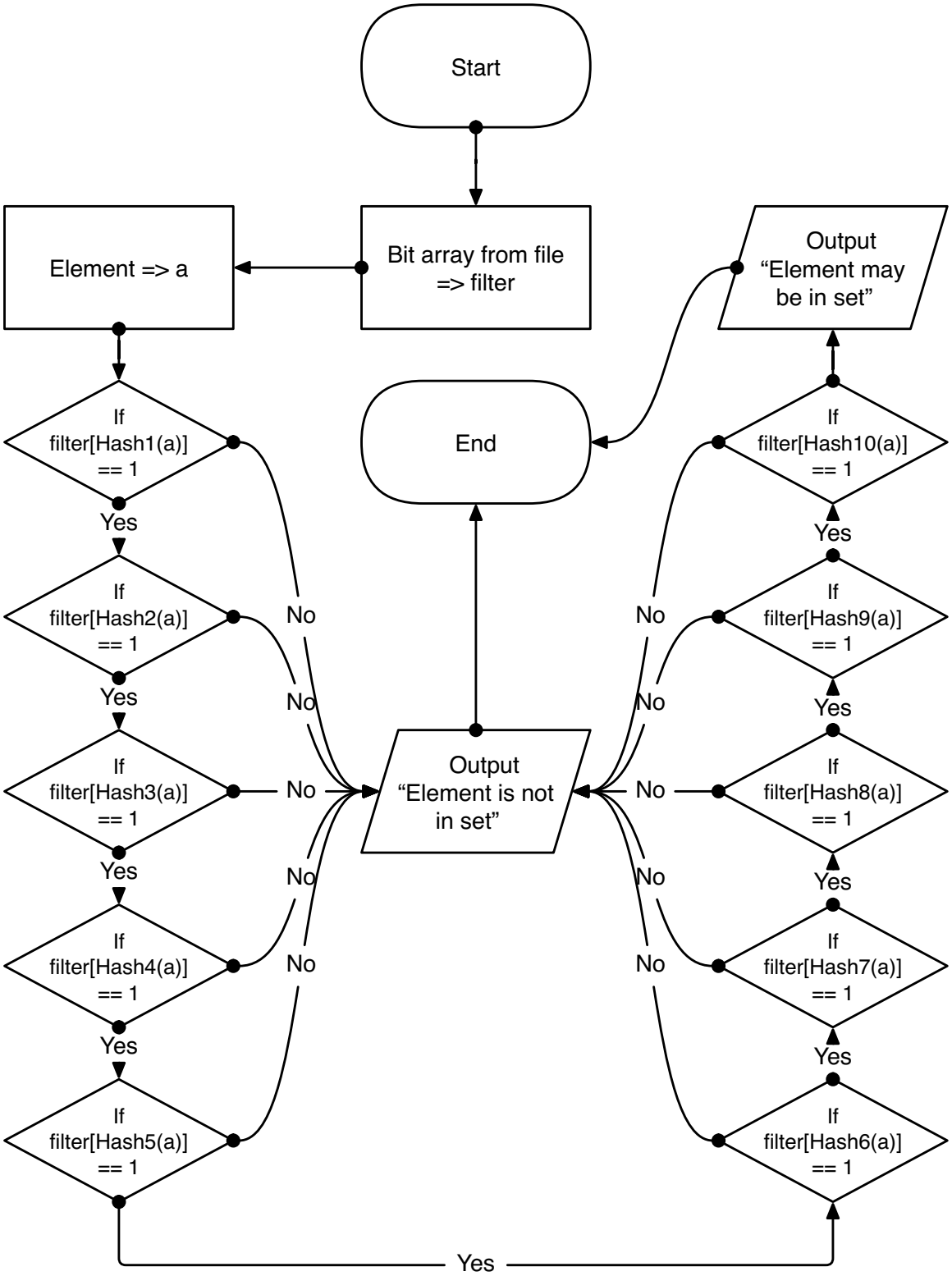
   i. Change the incorrect word to one that has been recommended by the algorithm, and undo the formatting applied to the incorrect word.
   ii. Ignore the error and undo the formatting applied to the incorrect word.
   iii. Add the word to the dictionary and undo the formatting applied to the incorrect 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 user should be able to increase the criteria for 'closest match' and request more close words.
   e. 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 spaces before and/or after punctuation.
   b. The program should automatically fix any grammar errors, if enabled in the settings page.
5. **Change the status of features:**
   a. The program should allow the user to toggle there status of all features.
   b. The status of features should be saved, so they can be reapplied each time the user starts the program.
6. **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.
7. **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 change the acronym to the full meaning when found in text.

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 Choice of languages used

For most of the program I used C# to build the back-end algorithms and user interface. I decided to use C# because it has the capabilities to build both complex algorithms and user facing code. This would help make the implementation of features across the solution more simple, as no other external libraries, interpreters or tools are required. I have also chose to use C# as the main language because I have been writing programs with C# for two years at college, this means that I have good knowledge of built-in functions and syntax. Finally I have chosen to use C#, because it is the commonly taught language at my college. This means that I have wide access to other knowledge and resources form both students and staff.

For some of the resource setup I needed to use a different language to C#, this is because the program would run for long periods of time and cannot be stopped during runtime. This means that it would need to run on my Raspberry Pi. For these reasons, I chose to use Python as there is an IDE preinstalled on Raspberry Pi OS. Also Python is an extremely high level programming language, this means that it is simple to understand and there are lots of resources online. Furthermore in secondary school I was taught Python syntax for three years, so I have good knowledge of how to write simple programs. None of the Python code will be used in final execution, rather it will only be used for the setup of resources.

If I had more experience, knowledge and time, I would have used Swift to create the spell checker instead. This is because I would have been able to create an app that works on devices running iOS, without the need to download other tools.

2.2 OOP design

All the utilities are combined into a single "Utilities" class, so they can be easily accessed and imported into other namespaces. Apart from the utility that is used to define words, as this is but using multiple classes. Because of this it is its own namespace, but is still organised in the same file as all other utilities.

Each view its own class that is spilt across two files as a partial class. This means that the designer for the view can be split from the logic: so new features can be added easier; and problems can be located in code easier when debugging.

2.3 Resources

2.3.1 Word list

2.3.1.1 Original word list

To setup the bloom filter (used to check if a word is true) a word list will be needed as an input for the initial hashing. I had difficulty finding a suitable word list, as it needed to contain all English words that only contained letters (no numbers or symbols). This is because I was worried that numbers and symbols may crash the program during execution. However eventually I found a GitHub repository[28] that had several word lists, and found one that would work for me.

2.3.1.2 Filtering the word list

Full code to filter the word list can be found on page ** in the appendix.

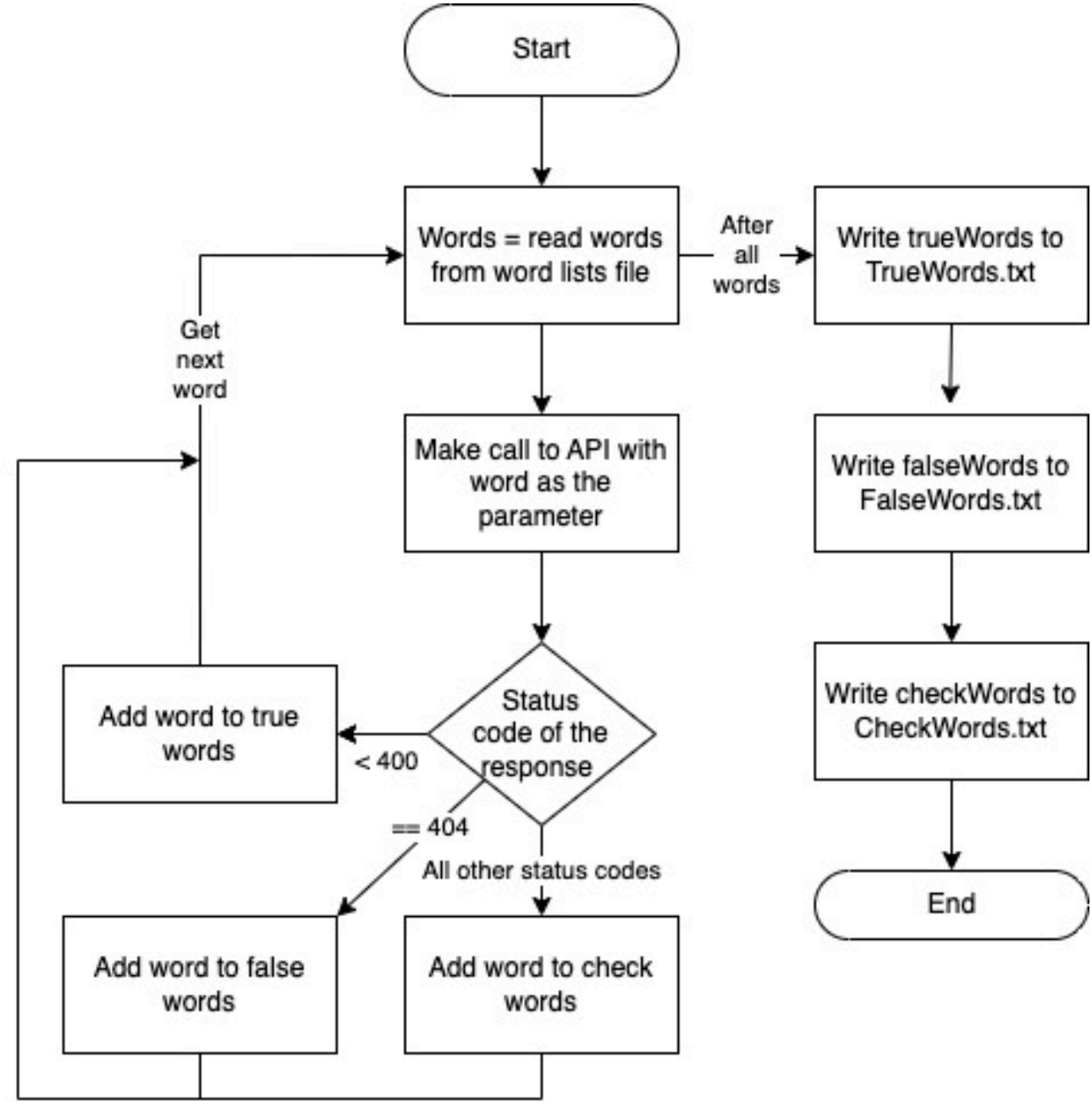
The word list I found did have some words that were not real English words, because of this I had to filter all the false words from the list. To do this I wrote a simple Python program that would make an API call to the dictionary I will use later. The status code of this request is then looked at:

- A status code less than four hundred (400):

- The request was successful
- The word is true

- A status code of four hundred and four (404):
  - The result was unsuccessful
  - The word is false
- All other status code:
  - There was an error making the request
  - The word should be checked manually

Flowchart to filter the word list



Pseudocode to filter the world list

```
list trueWords ← empty list
list falseWords ← empty list
list checkWords ← empty list

readName ← “home/pi/Desktop/words_alpha.txt”
words ← items in file readName

for each word in words
        url ← “https://api.dictionaryapi.dev/api/v2/entries/en/“ + word
        response ← API call to url
        statusCode ← status code of response

        if statusCode is less than 400
                add word to trueWords

        else if statusCode is 404
                add word to falseWords

        else
                add word to checkWords

        end if
end for

writeName ← “home/pi/Desktop/TrueWords.txt”
write trueWords to file writeName

writeName ← “home/pi/Desktop/FalseWords.txt”
write falseWords to file writeName

writeName ← “home/pi/Desktop/CheckWords.txt”
write checkWords to file writeName
```

2.3.2 Dictionary API

To define words I will use an API to get the full definition of a word. The API I have decided to use[29] returns a full JSON file that contains multiple definitions and origins for a given input. I decided to use this API instead of others because it is free and allows for unlimited calls, also during my testing it returned true values for all words I used as test values. The JSON file returned is really long and the basic definitions are nested in the file. This means that I will need to serialise the JSON file into a format that can be used in C#.

Example of JSON data returned from API call (https://api.dictionaryapi.dev/api/v2/entries/en/hello)

Full example JSON data can be found in the appendix[**].

So to get the first definition from this API call:

- `word.meanings[0].definitions[0].definition`
- Which would output "\"Hello!\" or an equivalent greeting."

2.3.3 Graphical user interface library

I wanted to create a graphical user interface to make the final solution easier to use, however I don’t have the current skills to create a high quality and fast interface. For this reason I am using a library[30] that will enable me to create a GUI in the terminal app using C# code.

I have decided to use this library over others because it doesn’t require me to learn any new syntax for a new language, as it uses C# code. This means that I would only need to learn how to use the new attributes and understand how to handle button presses.

2.4 Algorithms

Further explanations of some of the algorithms used can be found in the analysis section, pages **-**.

2.4.1 Bk tree[31]

The bk tree is used to find close matches to other words. It works by passing in a word and then creates a tree, setting the word as the root node. In a bk tree all edges from one node must have a distinct weight. So if a weight already exists from a node, then the weight must be recalculated from this node. For this reason the algorithm must be efficient, so runtime will be reduced when calling the function multiple times.

2.4.1.1 Overview of attributes

Leven
Leven is an object that has type LevenshteinDistance. It is used to calculate the distance from two words, so it is used when adding nodes to the tree.

Tree
Tree is a dictionary that is used to represent the graph. It is a dictionary which has key of type string and the values are a list of connections. A dictionary has been used ere over a matrix because of a property of a bk tree where weights must be distinct for each edge from a node, and a dictionary makes for easier comparison of current weights from a node. Each key value pair in the dictionary represents the connections from one node to all its children nodes. The string (key) is the parent node, and the list of connections (values) are the edges for the parent node to all the children.
So as an example:

- `tree["word"]` would return a list of connections
- `tree["word"][0]` would return the first connection from "`word`"
- `tree["word"][0].word` would return the word (string) of the child that was first added to the node (not necessarily the child with the lowest weight)

Words
Words is an array that is used to store all the true words defined by the word list file. At it's creation the words are read for the file and each word is stored as a value in the array. An array has been used here rather than a list because the size of this collection will never change after initialisation.

Root
Root is a string that is used to store the root value of the tree. I have decided to make root a global variable because it is used before the tree is created, and so it needs to exist both in and out of the tree. Furthermore it is a constant value that is used throughout the whole algorithm, so globalising this variable saves passing it around as a parameter.

Bk tree
This is the only constructor for the bk tree, it takes in a string and sets this value to the global variable root. It also resets the bk tree to an empty graph, because the same instance of the tree may be used for different words. It also calls the necessary functions to setup the tree, inside a for loop.

2.4.1.2 Overview of methods

Create root

This function is used to create the root of the tree. It creates a new connection where the word is the root, and the weight is zero (as weight from root -> root = i). This connection is then added to a list, and the list is added to the tree. Create root is only ever called from the constructor.

Add node

Add node is used as the general function for creating the tree. In the constructor the tree is checked if it is empty: an empty tree will call create root; whereas a non-empty tree will call add node. Add node creates a new connection and then passes this connection to create node.

Create node

Create node is a recursive function that checks all edges of a node to check if a weight already exists. If the weight doesn't exist, then the node can be added to the tree and created as normal. Otherwise, the node with the same weight will be used as the new parent node for the word. This means that the weight needs to be retaliated for word -> parent node, then create node can be called again to check the new parent for this weight. Base case: weight doesn't exists. General case: weight does exist.

Pseudocode for create node

```
function create node(word, parent, weight)
        exists ← check edges(parent, weight)

        if exists is not null
                new weight ← calculate distance(parent, word)
                create node(word, exists, new weight)

        else
                this connection ← new connection
                word of this connection ← word
                weight of this connection ← weight

                if tree does not contain word
                        list connections ← new empty list
                        add this connection to connections
                        add to the tree: word (as the key), connections (as the value)

                end if
        end if
end function
```

Return closest

Return closest is the general function that will be used in the program with the bk tree. This is because it is used to fund the closest words to another word. It works by creating a sub-tree and then checking the weight of all these nodes against a maximum weight. If the connection is less than the maximum weight then the word will be added to a list that will be returned form the function. The main code for this function is nested inside a while loop, this means heat only fifteen words will be returned at a time. A dummy connection is used to store the word of the last parent node, so that when the function is called to get more words the program "knows" where it was last.

2.4.1.3 Overview of tree structure

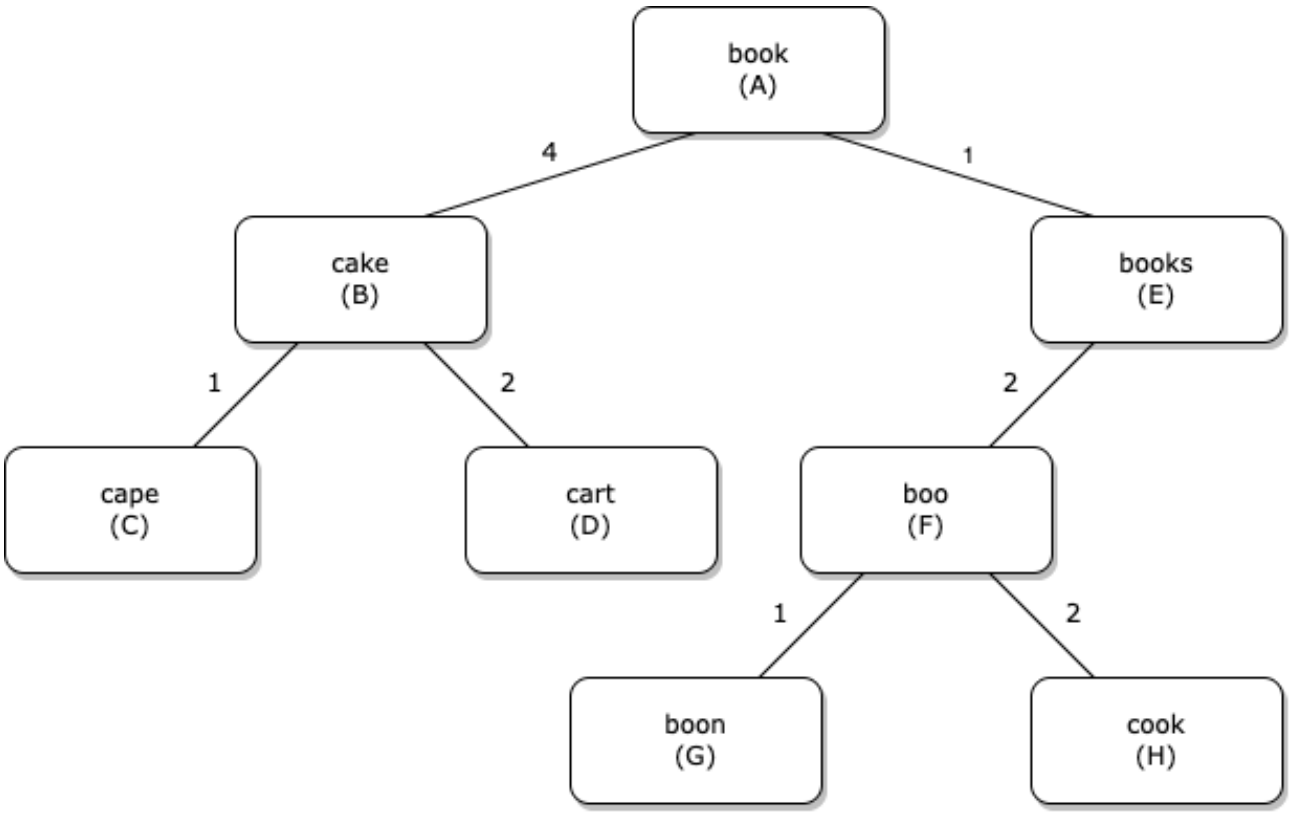
Visual representation

Adjacency list

A: [(B : 4), (E : 1)]
B: [(C : 1), (D : 2)]
C: []
D: []
E: [(F : 2)]

- Each value in the tree is a list of connections
- So [(C : 1), (D : 2)] is a list of connections
- This describes all the connections from the parent node (key)
- A connection is a structure, which has string (word) and a int (weight)
- So (C : 1) is a connection
- Each connection describes the word of the child node, and the weight of the edge (from parent → child)
- The tree is defined as a dictionary with key of string (parent node) and value of list of connections (word and weight of children nodes)

F: [(G : 1), (H : 2)]
G: []
H: []



2.4.2 Bloom filter[32]

The bloom filter is used to test wether a given word is in the set. It is more efficient than a search (binary for example) as comparisons are only made with given values, and the time complicity is a constant for every input. This is because the same number of processes are done for every input.

A bloom filter works by storing a large bit array and checking the indexes in this array to test if they are true. These indexes are gained by hashing the input a number of times, then the output of each hash is used as the index to check. If all the indexes are one, then it can be said that the word is most likely in the set. If any of the indexes are zero, then it can be said that the word is definitely not in the set.

2.4.2.1 Overview of attributes

Filter
Filter is an integer array that is used to store the filter that is read from the text file. It is assigned its value in the constructor, when the sequence of ones and zeros is read from the text file. I decided to use an integer array because each index can only be two values, true or false (one or zero). I didn't use a boolean array here, because it would require extra steps to read and write from the text file. The bit array needs to be big enough to reduce the false positivity rate. But if it is too big, then indexing, reading and storing will be slow, and take more space to store. I used an online calculator[**] to find the optimal bit array size, by inputting the number of hash functions and elements. From this calculator I found that I should use an array at least 1,961,567 bits in length, I decided to use an array of 2,682,975 bits instead to further reduce the false positivity rate.

2.4.2.2 Overview of methods

Constructors

There are two constructors for the bloom filter class, these are uniquely identified using ad hoc polymorphism. The separate constructors are used for general use and first time use.

The first time use constructor is identified by taking in an integer. This integer is left unused, and only purpose is to separate the constructors. This method sets all the indexes in the array to zero, and then sets up the filter by applying the insert function on each word. Then the filter is written to a text file.

The general use constructor is identified by taking no arguments. It reads the filter from the text file and then assigns it to the global variable. This is done using a for loop to iterate over the length of the string returned by the text file.

Lookup

One of the only public methods that is used externally by the program, it is used to see if a string is a part of the set. There were some issues when testing the prototype with one character strings, so all strings are checked for their length. Then one character strings are only returned true if equal to 'a' or 'i'. Otherwise, all the indexes of the hash function are checked, and if they are all one then true is returned. If any of these are not equal to one, then a binary search is performed on the text file of user added words.

Pseudocode for lookup

```
function lookup(word)
	if length of word is 1
		if word is 'a'
			or
		if word is 'i'
			return true
		else
			return false
		end if

	else
		if index of filter at Hash1(word) is 1
			and
		if index of filter at Hash2(word) is 1
			and
		if index of filter at Hash3(word) is 1
			and
		if index of filter at Hash4(word) is 1
			and
		if index of filter at Hash5(word) is 1
			and
		if index of filter at Hash6(word) is 1
			and
		if index of filter at Hash7(word) is 1
			and
		if index of filter at Hash8(word) is 1
			and
		if index of filter at Hash9(word) is 1
			and
		if index of filter at Hash10(word) is 1
			return true

		else
			return the result of binary search(word)

		end if
	end if
end function
```

Circular shift

A function that performs a binary shift on an input, then returns the shifted number as an unsigned integer. Performs the following bitwise operations on input:

- Left shift by one bit
- Right shift by 31 bits
- XORs the two values

This is equivalent to performing a full bitwise binary circular shift.

Pseudocode for circular shift

```
function circular shift (number)
        bits ← number in binary digits

        a ← perform left binary shift on bits by 1 place
        b ← perform right binary shift on bits by 31 places
        c ← perform binary OR function on a and b

        return c
end function
```

Mix

A function that takes in a single argument as an unsigned integer, and also returns an unsigned integer. Performs the following bitwise operations on an input:

- Right binary shift by 23 (randomly generated constant) bits
- XORs this with the original value
- Multiples this by a randomly generated constant (0x2127599bf4325c37 which is equal to 2.3889767e+18)
- Right binary shift by 47 (randomly generated constant) bits
- XORs this with itself

This is an efficient method to mix binary digits in a simulated random way, this means that the randomness can be generated once and recreated accurately each time. Accurate regeneration of the randomness is necessary because ever input always needs to return the same value.

Pseudocode for mix

```
function mix (number)
        bits ← number in binary digits

        a ← perform right binary shift on bits by 23 places
        bits ← perform binary XOR function on bits and a
        bits ← bits * (2.3889767e+18)
        b ← perform right binary shift by 47 places
        bits ← perform binary XOR function on bits and b

        return bits
end function
```

2.4.2.3 Overview of hash functions

To reduce the rate of false positives, I decided to use ten hashing functions for my bloom filter. The functions don't ned to be cryptographic, as the data doesn't need to be unreadable after hashing. Also I need to ensure values can be accurately reproduced each time a word is hashed. Using non-cryptographic hash functions means that the hashing will be more efficient, which is going to be essential when applying ten hashing functions to over one hundred thousand words. More information can be found on each individual hash function.

Pearson hashing (Hash 1)[18]

A hash function that is designed for fast execution on processors with 8 bit registers, I will slightly modify the function for use with 32 bits. It produces a single byte that is strongly dependant on each byte of the input. It is a simple hash

function that only requires a few instructions, and makes use of a 256 item lookup table (array) containing numbers 0 through to 255 in a random order.

Pseudocode for Pearson hash function

```
function pearson hashing (word)
        hash ← 0
        bytes ← word in binary digits
        nums ← array of 256 items in length containing 0–255 in a random order

        for each byte (a) in bytes
                index ← (hash XOR a) MOD 256
                hash ← item at index of nums

        return hash
end function
```

Buzhash(Hash 2)[19]

A simple hash function that avoids the use of multiplication, by using barrel shifts instead. This can make the function more efficient as circular shifts are fast to execute. Also known as hashing by cyclic polynomial, as it cyclicly rotates the digits and produces an output.

Pseudocode for buzhash

```
function buzhash (word)
        hash ← 1

        for each character (a) in word
                b ← circular shift (hash)
                c ← circular shift (a in binary form)
                d ← circular shift (a in binary form + 1)

                hash ← b XOR c XOR d

        return hash
end function
```

Fowler-Noll-Vo (FNV-0) hash (Hash 3)[20]

There are multiple versions of the Fowler-Noll-Vo hash function, I decided to use FNV-0 as it is the most simple and is also non-cryptographic. FNV functions can work in 32, 64, 128, 256 and 1024 bit variants, but I will make mine work with 32 bits as unsigned integers are 32 bits long. It was originally taken from an idea sent as a comment in 1991.

Pseudocode for FNV-0 hash

```
function FNV–0 (word)
        hash ← 0
        bytes ← word in binary digits

        for each byte (a) in bytes
                hash ← (hash * 16777619) XOR a

        return hash
end function
```

djb2 (Hash 4)[21]

This function uses binary shifts and the use of the constant 33. It is still unknown why the value 33 is used, but it has been proved to work better than many other constants. It also uses a randomly generated number as the initial value for the hash.

Pseudocode for dijb2 hash

```
function dijb2 (word)
	hash ← 5381 (randomly generated constant)
	bytes ← word in binary digits

	for each byte (a) in bytes
		b ← left shift of 5 on hash
		hash ← b + hash + 33

	return hash
end function
```

sdbm (Hash 5)[22]

This function was created as a replacement for use in a database library. It has been found to do well with scrambling bits, causing better distribution and fewer splits. sdbm is a good hash function as it is simple and causes good distribution of inputs. The constants used were chosen at random at first creation of the hashing algorithm.

Pseudocode for sdbm hash

```
function sdbm (word)
	hash ← 0
	bytes ← word in binary digits

	for each byte (a) in bytes
		b ← left shift of 6 on hash
		c ← left shift of 16 on hash
		hash ← 65599 + b + c − hash

	return hash
end function
```

PJW hash function (Hash 6)[23]

PJW hash function is a non-cryptographic hash function created by AT&T Bell Labs. The hash function involves shifting the previous hash and adding the current byte, then moving the high bits.

Pseudocode for PJW hash function

```
function PJW (word)
	hash ← 0
	bits ← word in binary digits
	max ← left shift of (bits − (bits / 8)) on 4294967295

	for each character (a) in word
		b ← left shift of (bits / 8) on hash
		hash ← b + a

		if max is not 0
			c ← right binary shift of bits on max
			hash ← hash XOR c ∗ 3 / 4 AND max

	return hash
end function
```

Fast-Hash (Hash 7)[24]

The fast-hash is a simple, robust and efficient general purpose hash function that I found on GItHub. The function works effectively on 32 and 64 bit values, and is most efficient on 32 bits. It is one of the longest hash function i use, but is still efficient.

Pseudocode for fast-hash hash

```
function fast−hash (word)
	a ← 9.8007717e+18
	hash ← 144 XOR (length of word ∗ a)
```

```
        b ← 0

        for each character (c) in word
                hash ← (hash XOR mix (c)) * a

        d ← (length of word AND 7) MOD length of word

        e ← (d - 1) * 8
        if e is less than 0
                e ← 0
        end if

        b ← b XOR left shift of d on (character at d in word)

        hash ← hash XOR mix (b) * a
        hash ← mix (hash)

        return hash
end function
```

Rabin fingerprint (Hash 8)[25]

The Rabin fingerprint is a method for implementing fingerprints using using polynomials, this particular one was proposed by Michael Rabin. A fingerprint is an algorithm that can map a large data item to a shorter bit string. In my application, fingerprints and hash functions can be used interchangeably.

Pseudocode for Rabin fingerprint

```
function rabin fingerprint (word)
        hash ← first character in word
        length ← length of word

        for each character (a) in word
                b ← (a * length) ^ (index of a)
                hash ← hash + b

        return hash
end function
```

Fletcher-32 (Hash 9)[26]

The fletcher-32 algorithm is a checksum, a checksum is like a hash function but they are more commonly used for error detection in sequences. In my application, checksums and hash functions can be used interchangeably. The goal of the fletcher32 checksum is to take all the properties of a cyclic redundancy check but lower the effort required, therefore reducing the time needed. The constants used are randomly generated at time of me writing the function.

Pseudocode for fletcher32

```
function fletcher32 (word)
        a ← 0
        b ← 0

        for each character (c) in word
                a ← a + (c MOD 65535)
                b ← (a + b) MOD 65535

        hash ← (left binary shift of 16 on b) OR a

        return hash
end function
```

CRC32 (Hash 10)[27]

CRC32 refers to a cyclic redundancy check for sequences of 32 bits in length. It uses a systemic cyclical function that encoded a value by adding a fixed length value. Commonly CRCs are used to generate a short, fixed length binary sequence for each block of data. It makes use of a 256 item lookup table (array) containing numbers 0 through to 255 in a random order. The constants used are randomly generated at time of me writing the function. In my application, cyclic redundancy checks and hash functions can be used interchangeably.

Pseudocode for CRC32

```
function crc32 (word)
	hash ← 4294967295
	bytes ← word in binary digits
	nums ← array of 256 items in length containing 0–255 in a random order

	for each byte (a) in bytes
		index ← ((hash OR a) AND 255) MOD 256
		hash ← (right binary shift of 8 on hash) OR (number at index in nums)

	hash ← hash OR 4294967295

	return hash
end function
```

2.4.3 Connection

Connection is a basic structure used to provide a template for each connection in the bk-tree. The connection structure is only used in the bk-tree. Connection has: a string which is used to store the word for the child node; and an integer which is used to store the weight (difference) between the parent and child nodes. A connection was used here to ensure each connection between nodes will always hold the necessary information for generation. It also allows me to easily create lists of connections to add to the dictionary.

2.4.4 Levenshtein distance

The Levenshtein distance is used to find the minimum distance between two strings, this algorithm will be used when recommending alternative words for the user. It works by comparing each letter in each string against one another and uses a matrix to store the values. These matrix values are then also used by the algorithm when comparing letters. The minimum value in the matrix is then returned as the distance.

2.4.4.1 Overview of attributes

Distance

Distance is a global integer for the Levenshtein distance class because it is always used in every method. It represents the distance between two strings, which is the sole purpose of this algorithm.

2.4.4.2 Overview of methods

Constructors

The constructor in this algorithm is just used to set the initial distance to zero. This means that if the same instance is ending used multiple times, the initial distance will always be zero before calculation.

Calculate

The general method used for the complex calculation of the distance between two strings. It works by comparing each letter and identifying whether any change ie needed. It takes in two strings as arguments, these are the two that the distance will be calculated from, and retains the distance as an integer.

Pseudocode for calculate

```
function calculate (word1, word2)
```

```
        cost ← 0

        if word1 or word2 has no length
                return the length of the other word

        matrix ← new two dimensional array (length of word1 + 1 by length of word2 + 2)

        for each letter (letter1) in word1
                for each letter (letter2) in word2
                        if letter1 is the same as letter2
                                cost = 0

                        else
                                cost = 1

                item at position (position of) letter 1 by (position of) letter 2 in matrix ← cost

        return last item in matrix
end function
```

Calculate all

This method is used when creating the bk-tree, as it will rerun an array of the distances between one word (the root) and every other word in the dictionary (the nodes). This method works by calling the calculate method and cycles through the array using a for loop.

Pseudocode for calculate all

```
function calculate all (word1, words)
        distances ← new empty array with length same as the number of words

        for each word (index) in words
                word ← item at index in words
                position of index in distances ← calculate (word1, word)

        return distances
end function
```

2.4.5 Define words

To define the words I will be using an API to fetch definitions from the a database. I have decided to do this so that I don't need to source and store definitions myself for each word. This also ensures that definitions are kept up to date, as I won't need to update definitions if they ever change. I found an API that has definitions for all English words and returns a JSON file when called. Example JSON file can be found in the appendix[**]. The define words utility is a separate namespace to the others, as it works by using multiple classes in a complex OOP model.

2.4.5.1 Overview of methods

Define word

Located inside the program class, this method takes in a single word as a string (to define) and returns a list of strings (definitions). This returns a list of strings because the algorithm returns a definition for each word type (noun, pronoun, verb, adjective, adverb, preposition, conjunction, interjection), rather than just one definition. This is because during my first interview Gabi said that she would rather have multiple, short definitions rather than one long definition.

The web request works by creating a new web request object with the API's URL and the word passed as an argument. A response object is then created and set to the response returned from the web request. Then a data stream is created and the response is turned into a stream that can be read by the stream reader. Using the NetwonSoft.JSON dependency, this stream is the deserialised and turned into an useable object.

Pseudocode for define word

```
function define word (word)
	URL ← “https://api.dictionaryapi.dev/api/v2/entries/en/" + word
	response ← JSON file returned from web request of URL
	object ← deserialise response

	definitions ← new dictionary of type string and string
		(“noun", “null")
		(“pronoun", “null")
		(“verb", “null")
		(“adjective", “null")
		(“adverb", “null")
		(“preposition", “null")
		(“conjunction", “null")
		(“interjection", “null")

	finals ← new empty list

	for each definition (a) in object
		if part of speech for a is not present in definitions
			add part of speech and first definition for part of speech to finals

	return finals
end function
```

2.4.5.2 Overview of JSON deserialisation

JSON deserialisation is the process of turning a JOSN file into a useable object where the elements can be accessed directly.

The JSON file is returned from a HTTP web request and then deserialised using a method. This enables it to be stored to an object where each element can be easily accessed, rather than accessing elements through string manipulation. For successful deserialisation, each element in the returned file must be recreated as a class in the algorithm. Examining the example returned[**] I can identify what elements are present and what I need to fully deserialise (each of the top level elements is its own class):

- Word – string
- Phonetic – string
- Phonetics – list of **phonetics**
  - Text – string
  - Audio – string
  - Source URLs – list of strings
  - License – **license**
    - Name – string
    - URL – string
- Meanings – list of **meanings**
  - Part of speech – string
  - Definitions – list of **definitions**
    - Definition – string
    - Synonyms – list of strings
    - Antonyms – list of strings
    - Example – string
  - Synonyms – list of strings
  - Antonyms – list of strings
- License – **license**
  - Name – string
  - URL – string
- Source URLs – list of strings

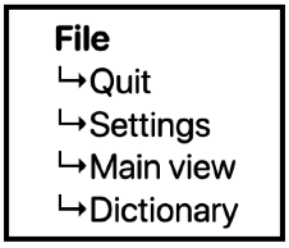
2.5 User interface

All the views are split into partial classes across files. this allows the designer to be split from the logic, which will make it easier when debugging and adding features. All the view designers inherit from the master view, as this builds the menu bar.

Where views displayed words, a lookup text field was used to find the correct word. This was done instead of each word being a button, because buttons use a lot more processing power than labels and would therefore slow the program down a lot making it unusable.

2.5.1 Master view

Used as a base for the other views to inherit the menu bar from. This view is never actually called on its own. This allows the menu bar to be updated globally if needed and removes some repeated code.



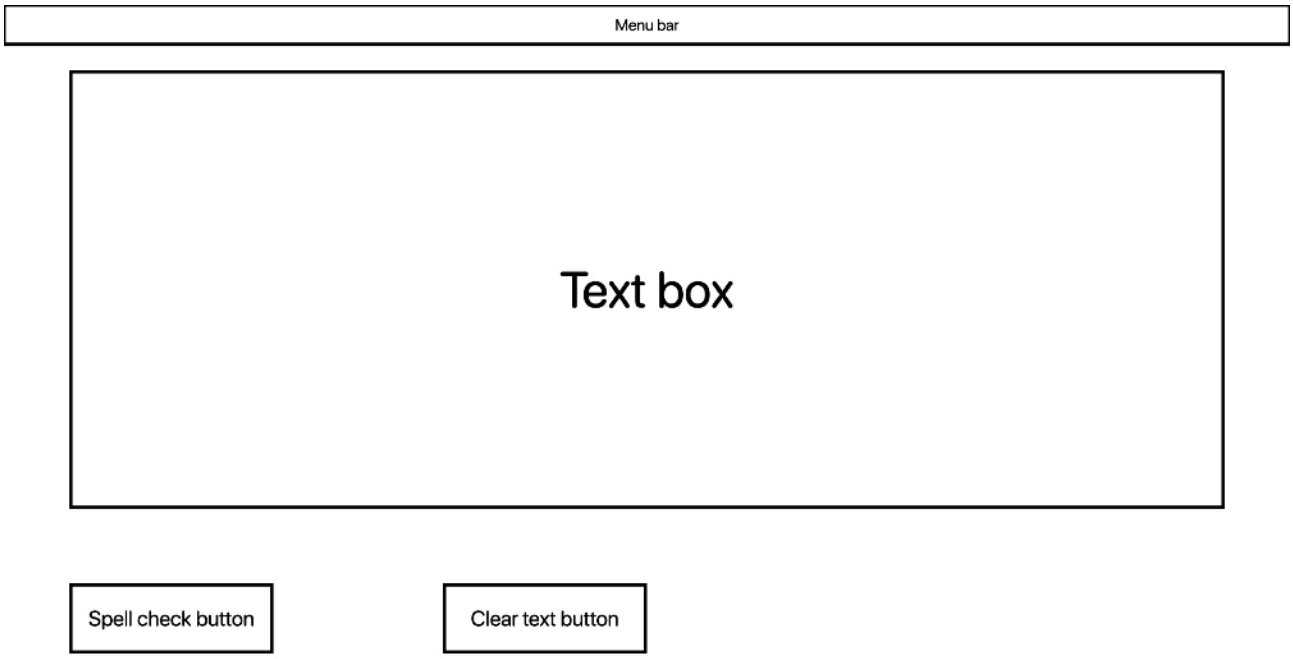
Menu bar design that is inserted from the master view

2.5.2 Main view

The main view is used as the initial view the user sees when the program is started. It has a large text field and buttons to clear the text and begin the spell checking.

The spell check button will get the text in the text box, and if the text is null then an error message is displayed. otherwise the spell check view is called, passing in the text as an argument.

The clear button sets the text in the box equal to an empty string.



2.5.3 Dictionary view

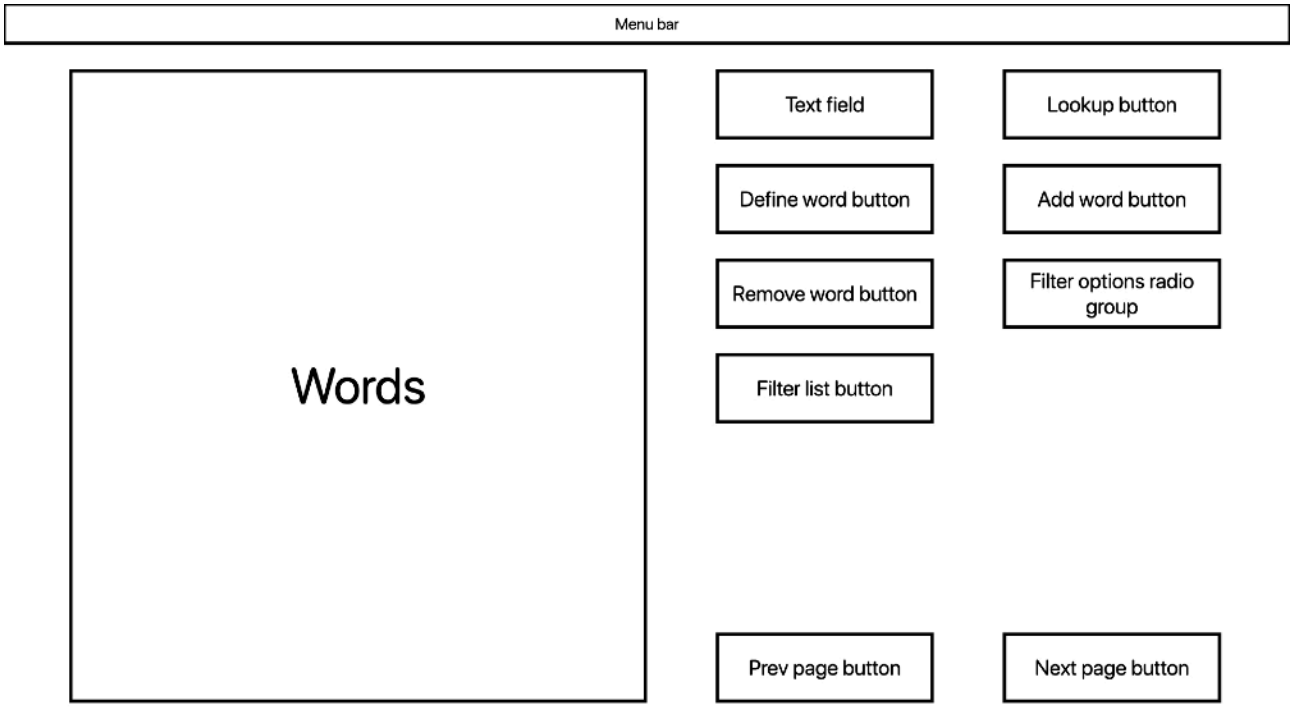
The dictionary view will be used to: display the list of true words; define words; add words to the list; and remove words from the list. Words will be displayed in a list and next/previous page buttons will be used to navigate through the list.

The lookup button will make select the word in the text field to make any further operations work on this word. It also searches the word list for words beginning with the string and only show words matching.

The define word button uses the define word algorithm to dimply a dialogue box with one definition for each word type.

The add word button adds the word to the added words file and updates the displayed word list to show this new change. The remove word button removes the word from the text file and updates the displayed word list to show this new change.

The filter buttons let the user choose between: all words, default words and added words. it then shows this change in the word list when the button is pressed.



2.5.4 Settings view

The settings view has clear labels and buttons that allow the user to customise the program and save the settings easily. the user can also reset the settings to default, and update the interface to show the changes.

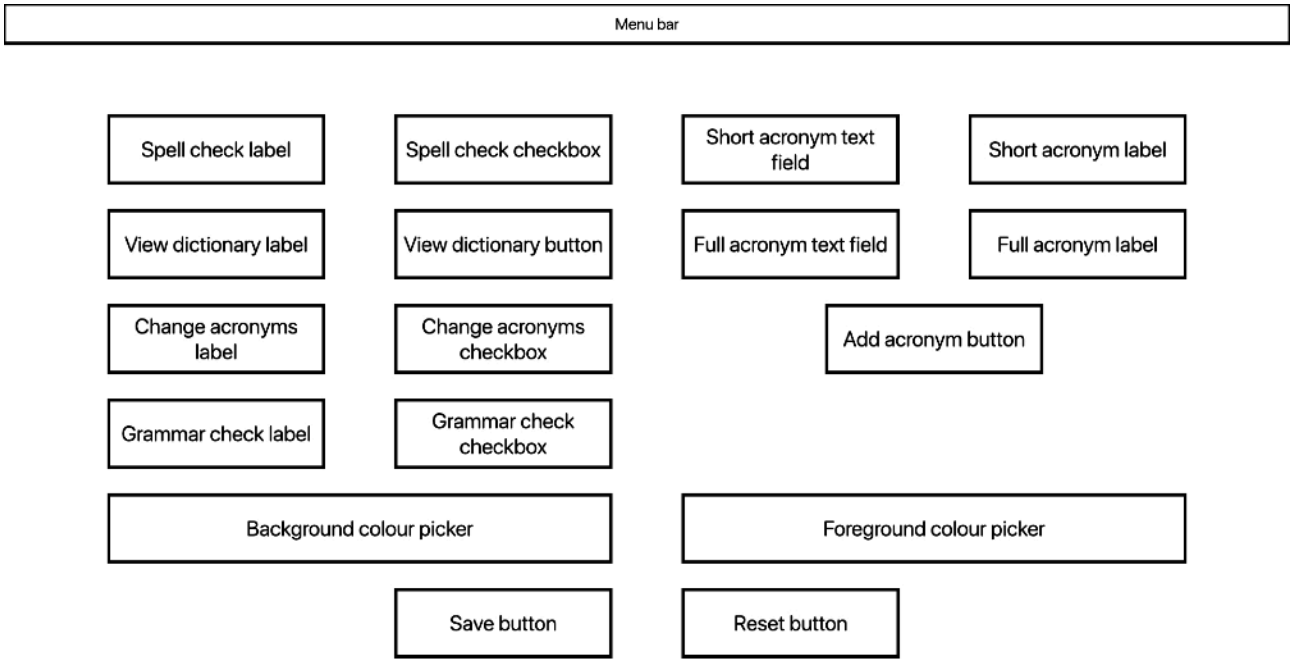
Acronyms can be added by entering the short and full acronym and pressing the button. This is then stored to a text file in the form shortAcronym*fullAcronym. When these are read from the text file, the string is split into an array based on the index of the star.

When necessary check boxes are used instead of text fields, as this reduces the need for human input and error handling. This can't be done for all settings though, as some require more detail or number input.

There are also colour pickers at the bottom of the view that allow the user to change the foreground and background colours severalty and change the interface on save.

When the settings are saved they are saved to a text file so they can be reread when the program is closed and restarted. This was one of the requirements outlined by Gabi in my interview, so it is essential this system is robust.

When the program is first run, an instance of the settings page is created in the background. this allows the interface to fetch the settings and change colours where needed. The user is unable to change the font colour and/or size.



2.5.5 Spell check view

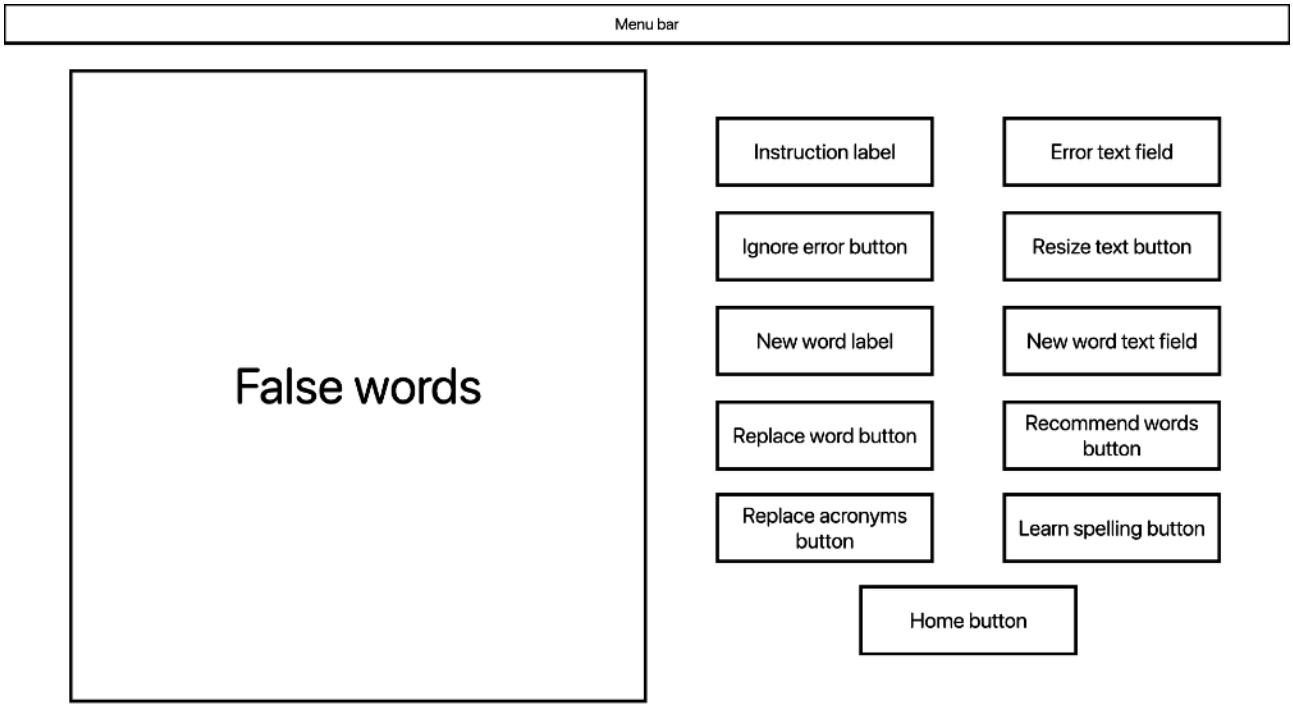
False words here are displayed in the same way words are displayed in the dictionary. However in tis view there are more elements, as it is used to manage all the incorrect words. False words are identified by the bloom filter algorithm and then stored separately in an array. Once the error has been handled, it is removed from the array ad not displayed in the false words list.

The instruction label is necessary on this view because there are many functions that can be done and can impact a user's text. So this ensures that processes are carried out intentionally and in the right order by the user to avoid frustration and make the program more intuitive to use.

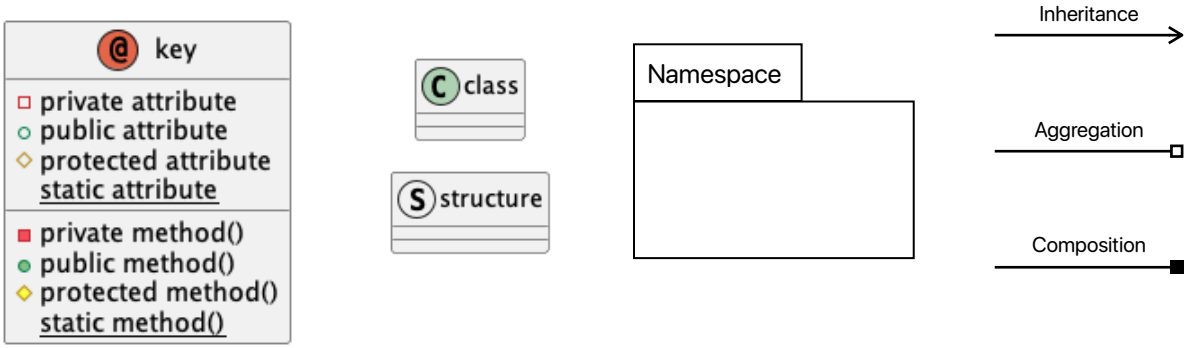
The new word elements require the user to input a word in both the error text field and new word text field. Then when the button is pressed, the error is changed to the new word in the text.

Pressing the recommend words button will use the bk-tree to find close matches to the error. This is all done fairly quickly due to the algorithm being efficient and makes good use of the calculate all method in the Levenshtein distance algorithm. A dialogue box is then shown to the user, and each of the words is a unique button. When pressed the error will be changed to the selected word. The user can sloe choose to get more distinct words, which increases the maximum distance and recalls the traversal function in the bk-tree.

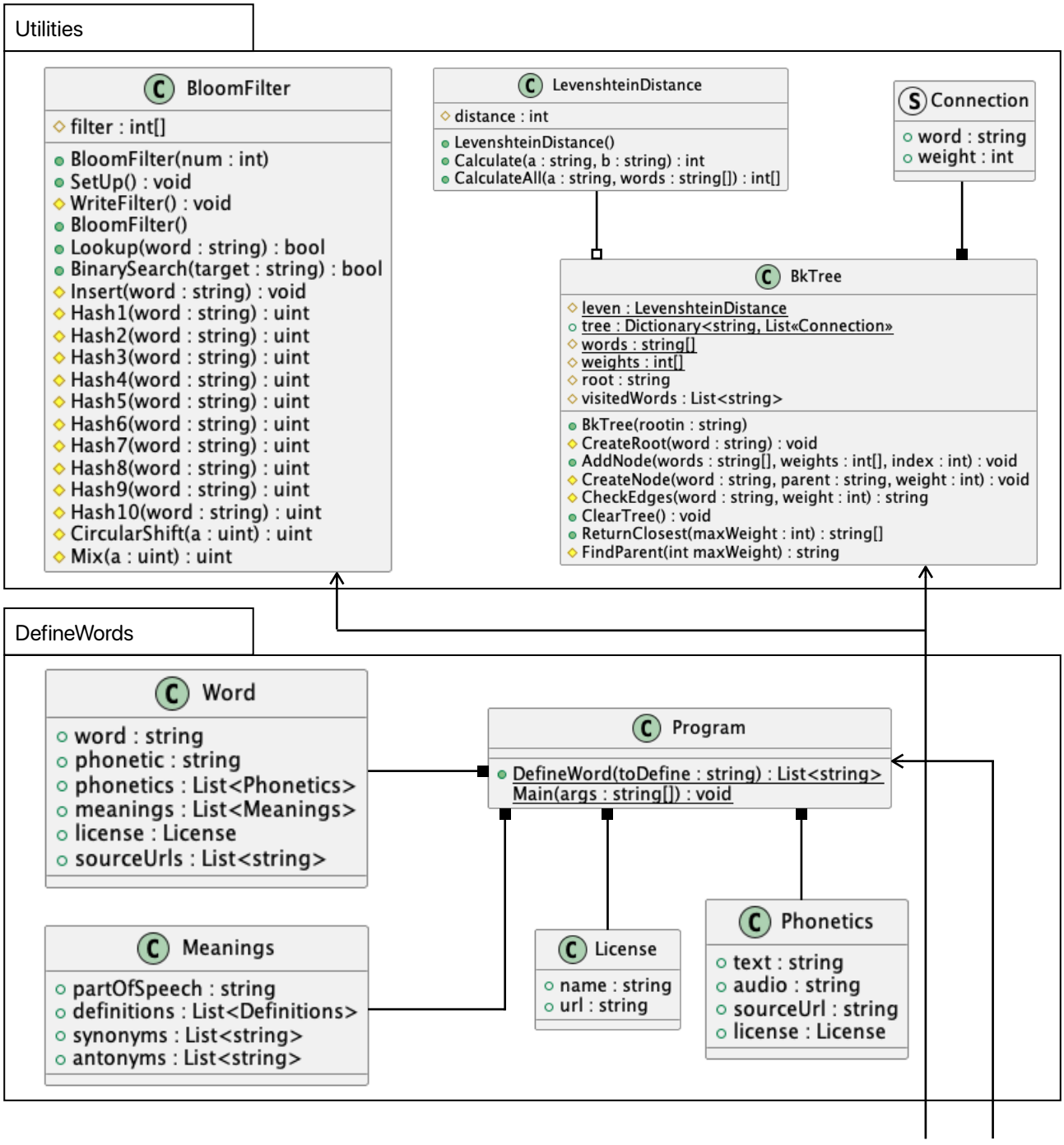
If no errors are found in the text, then the spell check view is not shown. instead a simple message is displayed to the user that allow them to go back to the main view. The menu bar is not present in this message.


Menu bar
False words
Instruction label
Error text field
Ignore error button
Resize text button
New word label
New word text field
Replace word button
Recommend words button
Replace acronyms button
Learn spelling button
Home button

2.3 Class diagrams

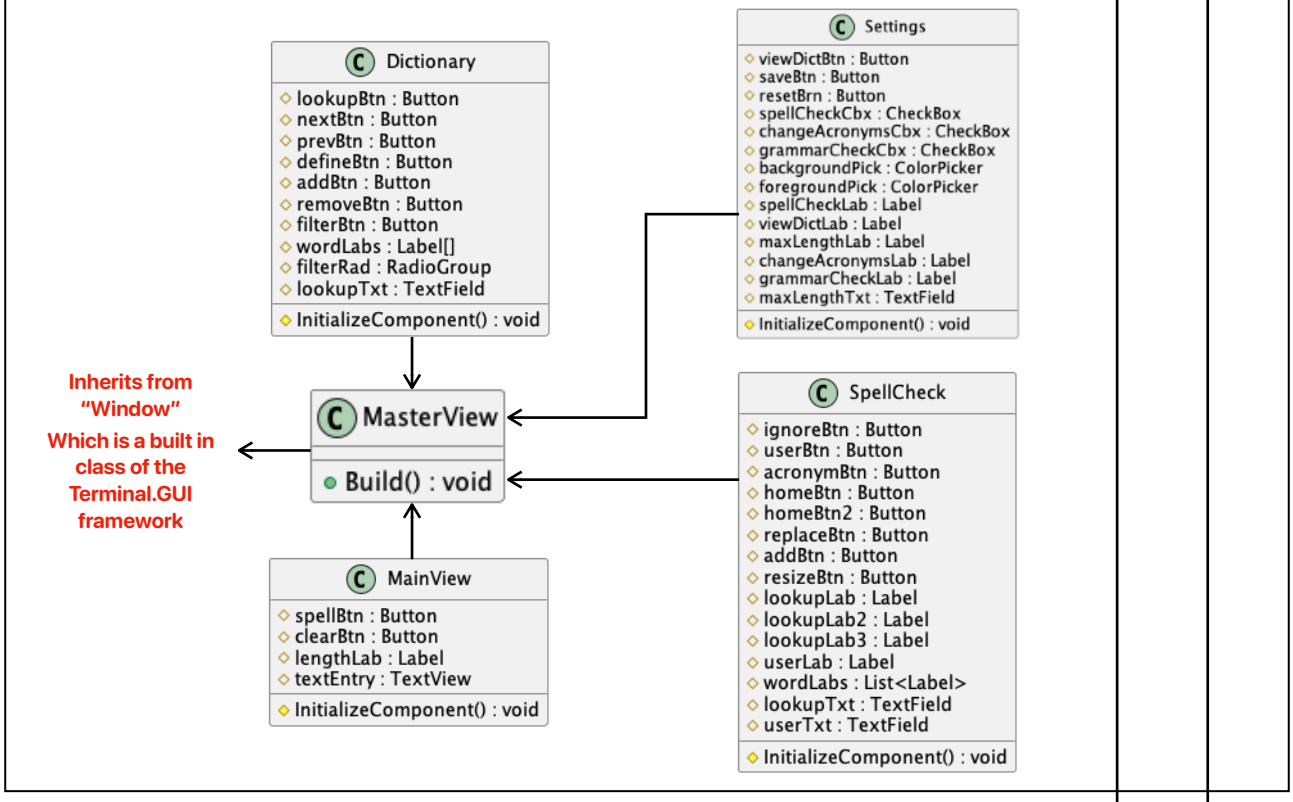


2.3.1 Utilities



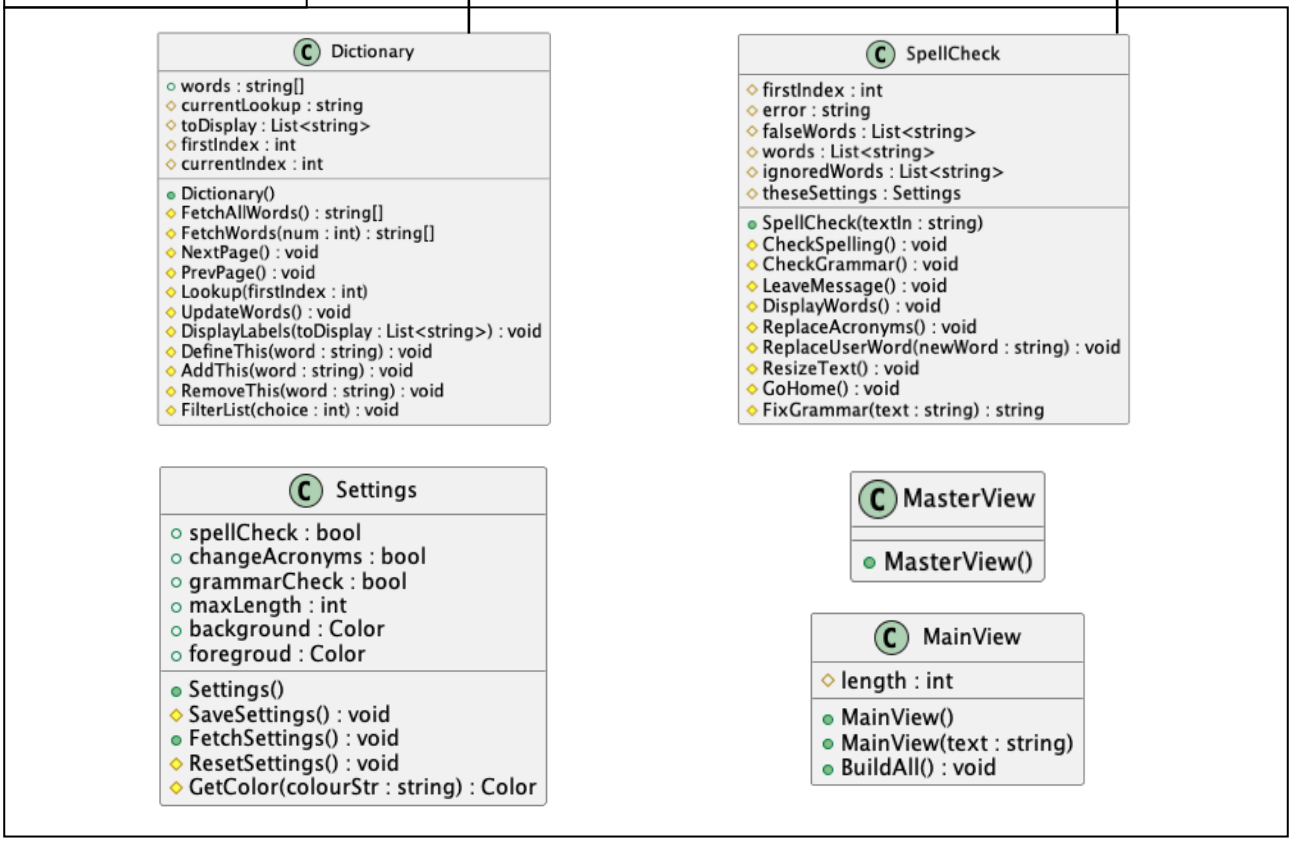
2.3.2 View designers

FINAL



2.3.3 Views

FINAL



3 Technical solution

3.1 Table of contents 44

3.1 Table of contents

The full technical solution can also be found on my public GitHub repository: https://github.com/lewisdrake28/NEA

Code section	Details	Complex properties
CsProj code	Provides details of framework and packages used	
Utilities namespace	Provides the complex utilities for the program	Namespace/OOP model
Bk tree	Used to find close matches	Graph theory
		Dictionary
		Ad hoc polymorphism
		Single dimensional arrays
		Recursive function
Bloom filter	Used to test if a member is part of a set	Single dimensional array
		Hashing functions
		Binary search
		Writing to text files
		Reading from text files
		Binary shifts
Connection	Structure used for a connection between two nodes	Structure
Levenshtein distance	Used to find the distance between two strings	Muti dimensional array
		Matrix
Define words	Makes an API call and parses a JSON file	List
		Dictionary
		Complex OOP model
		Parametrised API call
		Parsing JSON files
		Web request
		JSON deserialisation
Program	Calls the first window	Error handling
Master view designer	Provides a model for the other designers to inherit the menu bar from	Base class
		Inheritance

Main view designer	Front end designing for the main view	Inheritance
Dictionary view designer	Front end designing for the dictionary view	Inheritance
Settings view designer	Front end designing for the settings view	Inheritance
Spell check view designer	Front end designing for the spell check view	Inheritance
Master view	Back end processing for the master view	
Main view	Back end processing for the main view	Ad hoc polymorphism
Dictionary view	Back end processing for the dictionary view	Reading from text files
		Writing to a text file
Settings view	Back end processing for the settings view	Writing to a text file
		Reading from text files
Spell check view	Back end processing for the spell check view	Lists
		String manipulation

4 Testing

4.1 Testing values

For testing purposes, five random true strings and five random false strings will be used as constants throughout this section. Using constant test values will allow me to predict the results, and easily identify any failed tests.

The random true words will be generated from a website[33]:

- budget
- stay
- variation
- drive
- expression

The random false strings will be generated from a simple program[34]:

- urrltsj
- tp
- thjs
- matn
- undert

4.2 Testing overview

To fully test my whole program, I will test each sub-objective separately and use a table to keep track of failed tests and data used. I will also make a testing video, so when I come to fix failed tests I can identify how and why they failed. Some tests will be done multiple times with different inputs, to test the robustness of a feature. Some other test may be done entirely to test the error handling and input filtering.

Testing video: https://youtu.be/PtI0XVDNnKA

Test no.	Test description	Objective testing	Test data	Expected result	Pass/fail	Video evidence
Objective 1 - interface						
1	Does the interface start with a simple welcome page that allows the user to start spell checking and access other elements?	1.a.	The program is started.	Program should start with a box to start typing, buttons to access other elements, and any other relevant elements.	Pass	00:00 - 00:08
2	Does the interface mark words that are incorrect in an appropriate way?	1.b.	Incorrect word - "urrltsj".	The word should be displayed in the false words list in the spell check view.	Pass	00:09 - 00:16
3			Incorrect word - "tp".		Pass	00:17 - 00:21
4			Correct word - "budget".	The word should not be displayed in the false words list in the spell check view.	Pass	00:22 - 00:27
5			Correct word - "stay".		Pass	00:28 - 00:32
6			String that is larger than the max length of string (>2048 bytes). Unable to test with suitable data.	Error message should displayed asking user to reduce the size of the string.	Pass	00:33 - 00:34
7			Null string.	Message should be displayed informing the user that no errors have been found.	Pass	00:35 - 00:36
8			String made of numbers - "12345".	Strings containing numbers or special characters should be ignored.	Fail	00:37 - 00:42
9			String made of special characters - "!@£$%".		Fail	00:43 - 00:50
10			String made of numbers, special characters and letters - "12!@ab".		Fail	00:51 - 00:58

11			String made of numbers and special characters - "12!@".		Fail	00:59 - 01:05
12	Can the word be changed to one that has been recommended by the algorithm?	1.c.i.	False string - "matn".	A message should be displayed with recommend words an the user should be able to select one, then the false word should be changed to this new word.	Pass	01:06 - 01:36
13	Can the word be changed to one that has been inputted by the user?		String that is larger than the max length of string (>2048 bytes). Unable to test with suitable data.	Error message should be displayed asking user to reduce the size of the string.	Pass	01:37 - 01:38
14			User enters a null string and "Replace" button is pressed.	Word should be changed to the null string, and treated as a true word.	Pass	01:39 - 01:50
15			User enters "mate" and "Replace" button is pressed.	Word should be changed to the string that has been entered by the user, and treated as a true word.	Pass	01:51 - 02:05
16			User enters a string made of numbers and "Replace" button is pressed.		Pass	02:06 - 02:20
17			User enters a string made of special characters and "Replace" button is pressed.		Pass	02:21 - 02:37
18	Can the error be ignored?	1.c.ii.	"Ignore error" button is pressed.	The word should be ignored as a false word and it should be removed from the false words list. The false words list should be updated to show this change.	Pass	02:38 - 02:50
19	Can the word be added to the dictionary?	1.c.iii.	"Learn spelling" button is pressed.	The word should be added to the dictionary and the text should be checked for spelling errors again. The word should not ever reappear in the false words list, unless later removed.	Pass	02:51 - 03:07
20	Can the user customise the interface by changing the font size?	1.d.	"Change font size" button is pressed.	The font size should be changed to the size selected by the user.	Fail	03:08 - 03:16

21	Can the user reset the font size to the default settings?		"Reset font size" button is pressed.	The font size should be reset to the default settings defined by the interface.	Fail	
22	Can the user customise the interface by changing the font colour?		The user selects the foreground colour using the picker, and then presses the "Save" button.	The foreground colour should be changed to the colour selected by the user.	Pass	03:17 - 03:31
23	Can the user customise the interface by changing the backgorund colour?		The user selects the background colour using the picker, and then presses the "Save" button.	The background colour should be changed to the colour selected by the user.	Pass	03:32 - 03:37
24	Can the user reset all settings back to their default value?		"Reset to default settings" button is pressed.	All settings should be returned to the default values beings by the interface, and the foreground and background colours should be updated to default.	Pass	03:38 - 03:48
Objective 2 - spell check						
25	Does the program check all words entered by the user?	2.a.	Sequence of strings of words varying in length, mostly correct and some incorrect - "quik brown foz jumped over lazee dog".	The words "quik", "foz" should be identified as false words and all others should be ignored.	Pass	03:49 - 04:48
26			String that is larger than the max length of string (>2048 bytes). Unable to test with suitable data.	Error message should displayed asking user to reduce the size of the string.	Pass	04:47 - 04:48
27			Null string.	Message should be displayed informing the user that no errors have been found.	Pass	04:49 - 04:53
28	Are the incorrect words (gathered from objective 2.a.) identified in an appropriate way?	2.b.	Incorrect word - "urrltsj".	Words should appear in the false words list in the spell check view.	Pass	04:54 - 05:15
29			Incorrect word - "tp".		Pass	
30			Correct word - "budget".	The word should not be displayed in the false words list in the spell check view.	Pass	
31			Correct word - "stay".		Pass	

Objective 3 - alternative word recommender						
32	Does the program allow the user to request alternative words for ones that have been identified as false?	3.a.	Null string is entered in the lookup field, and the "Recommend words" button is pressed".	An error box is shown telling the user that the string entered is not in the false words list, and prompts the user to select another word.	Pass	05:16 - 05:21
33			A true word is entered in the lockup feels, and the "Recommend words" button is pressed - "drive"		Pass	05:22 - 05:27
34			False word is entered in the lookup field, and the "Recommend words" button is pressed - "matn".	Dialogue box is shown asking the user to select a new word that has been recommended.	Pass	05:28 - 06:55
35	Does the algorithm only return words that are a close match to the false word?	3.b.		Dialogue box is shown asking the user to select a new word that has been recommended. The words shown in the dialogue box should be close to "matn", such as: "mate", "man", etc.	Pass	
36	Can the user replace the false word with any of the recommended words?	3.c.	False word is entered in the lookup field, and the "Recommend words" button is pressed. Then a word is selected from the list - "matn" and "mate" is selected.	Dialogue box is shown asking the user to select a new word that has been recommended. A user can select "mate" and the "matn" is then changed to "mate". "matn" is then removed from the false words list.	Pass	
37	Can the user increase the scope criteria for 'close words' and get more recommended words?	3.d.	False word is entered in the lookup field, and the "Recommend words" button is pressed. Then the "More words" button is pressed" - "matn".	Words in the dialogue box should be changed to show the new words, some of the words may be unchanged.	Fail	06:56 - 07:37
38	Can the user define all words that have been recommended?	3.e.	False word is entered in the lookup field, and the "Recommend words" button is pressed. Then the "Define word" button is pressed" - "matn".	A new dialogue box is shown with multiple brief definitions for the requested word.	Fail	
Objective 4 - grammar check						

39	Does the program check for missing spaces before and/or after punctuation?	4.a.	String with a missing space after a full stop is entered and "Spell check" button is pressed - "drive.stay".	Grammar error is identified.	Pass	07:37 - 07:51
40	Does the program fix any missing spaces found before and/or after punctuation?	4.b.		"drive. stay" is returned and replaces the original incorrect string.	Pass	
41			String with missing spaces before and after brackets is entered and "Spell check" button is pressed - "budget(variation)expression".	"budget (variation) expression" is returned and replaces the original incorrect string.	Pass	07:52 - 08:08
Objective 5 - change the status of features						
42	Can the user toggle the status of the spell check?	5.a.	The "Spell check" check box is toggled.	Status of the feature is changed, and the check box is changed to reflect the status.	Pass	08:09 - 08:15
43	Can the user toggle the status of the grammar check?		The "Grammar check" check box is toggled.		Pass	08:16 - 08:18
44	Can the user toggle the status of the acronym changer?		The "Change acronym" check box is toggled.		Pass	08:19 - 08:24
45	Can the settings be reset to their default values?		The "Reset" settings button is pressed.	All the settings should be rest to default, and a dialogue box should be displayed to show this change.	Pass	08:25 - 08:34
46	Are all the settings saved externally and reapplied when the program is restarted?	5.b.	The "Save" settings button is pressed.	A dialogue box should be displayed informing the user the settings have bene saved.	Pass	08:35 - 08:41
47			The program is restarted.	The settings saved last time should be reapplied.	Pass	08:42 - 08:55
Objective 6 - personalise the dictionary						

48	Can the user add false words to the dictionary?	6.a.	A false word is entered in the main view text field, then the false word is entered in the lookup field. The "Learn spelling" button is then pressed - "matn"	The word was added to the dictionary and a dialogue box informing the user is displayed.	Pass	08:56 - 09:36
49	The dictionary should be sorted alphabetically form a-z.	6.b.i.	The dictionary is started.	The dictionary should be sorted alphabetically form a-z.	Pass	09:37 - 09:47
50	Can the user filter the word list to show only: all words; added words; or default words?	6.b.ii.	The "All words" radio button is selected, then the "Filter list" button is pressed.	All words in the dictionary should be displayed.	Pass	09:48 - 10:01
51			The "Default words" radio button is selected, then the "Filter list" button is pressed.	Only default words in the dictionary should be displayed.	Pass	
52			The "Added words" radio button is selected, then the "Filter list" button is pressed.	Only words added to the dictionary should be displayed.	Pass	
53	Can the user search for words in the dictionary?	6.b.iii.	A true word is entered in the text field and the "Lookup" button is pressed - "drive".	Any strings starting with the inputted string should be displayed in the word list now.	Pass	10:02 - 10:12
54			A false word is entered in the text field and the "Lookup" button is pressed - "tp".		Pass	10:13 - 10:19
55			A null string is entered in the text field and the "Lookup" button is pressed.	All words in the dictionary should be displayed.	Pass	10:20 - 10:21
56	Can the user add words not currently in the dictionary?		A true word is entered in the text field and the "Add word" button is pressed - "drive".	The word should be added to the dictionary and now shown in the word list. A dialogue box should be displayed to inform the user of this.	Pass	10:22 - 10:48
57			A false word is entered in the text field and the "Add word" button is pressed - "tp".		Pass	10:49 - 11:11

58	Does the program define selected words for the user when requested?	6.b.iv.	A true word is entered in the text field and the "Define word" button is pressed - "budget".	A dialogue box should be displayed with a single definition for each word type.	Pass	11:12 - 11:27
59			A false word is entered in the text field and the "Define word" button is pressed - "urrltsj".	An error message should be displayed informing the user that the word is not real and cannot be defined.	Pass	11:28 - 11:40
60	Can the user remove words from the dictionary?	6.b.v.	A word currently in the dictionary is entered in the text field and the "Remove word" button is pressed - "expression".	A dialogue box should be displayed informing the user the word has been removed form the dictionary.	Pass	11:41 - 12:19
61			A word not currently in the dictionary is entered in the text field and the "Remove word" button is pressed - "undert".		Pass	
Objective 7 - replace acronyms						
62	Can the user define and save new acronyms with both the short and full definition?	7.a.	The short and full acronyms are inputted in the text fields, and the "Add acronym" button is pressed - "omw" (short acronym) and "on my way" (full acronym).	The acronym should be saved to the text file (in the background) and a dialogue box should be displayed to inform the user.	Pass	12:20 - 12:57
63	Can the user change short acronyms to the full expanded version?	7.b.	The user enters the short acronym in the main view text field, and the "Spell check" button is pressed. The short acronym is identified as a false word, and the "Replace acronyms" button is pressed - "omw" (short acronym) and "on my way" (full acronym).	"omw" is changed to "on my way" and the word is removed from the false word list.	Pass	12:58 - 13:08

4.4 Failed tests analysis

4.4.1 Tests 8-11

These test failed because of the special characters, this is because there was no extra checking for special characters before the spell check algorithm is done. This means that when they are put through the bloom filter, they are all identified as wrong words. This is because when the bloom filter array was created, no words with special characters were used. To fix this, all strings entered could be checked to see if they contain special characters. If they do then they could be ignored by the spell checking algorithm. This is an easy fix and could be implemented quickly, but I would need to be cautious of apostrophes and dashes.

4.4.2 Test 20

I didn't implement the font size changer because I couldn't find a simple way to do it with the user interface library I was using (Terminal.GUI). It is possible for the user to change the font size in their own terminal app, but there is no implemented way to change this in the program. For this reason, I didn't see a purpose in spending a lot of time looking for a solution.

4.4.3 Test 21

The font size can't be reset to default because the font size can't be changed within the program. This is because of reasons discussed in section 4.4.5. If the user were able to change the font size then resetting it would be simple, as it is currently done for all the other settings.

4.4.4 Test 37

This test failed for an unknown reason, and further testing needs to be done to find the cause of this failure.

4.4.5 Test 38

The user cannot directly define all the words recommended by the bk-tree because of the way these words are displayed. The dialogue box used is to show the words is not very dynamic, so very few elements can be added for it to still be useable. So to keep the box simpler to use, I decided to not have the ability to define words directly from this box. But all words can be defined from the dictionary view.

5 Evaluation

5.1 Overall effectiveness of system

Problem stated in section 1.1: "The problem I aim to solve is how current spell checkers lack the ability to allow users to customise recommendations and save preferences.".

My solution effectively solves the problem I outlined as it allows the user to check the spelling of words and customise settings. These settings can then be saved and are reapplied each time the program is started. It also adds some extra features such as basic grammar check and having the ability to define words. The program could also be used as a dictionary instead of a spell checker due to it being broken down to separate views.

5.2 Evaluation of objectives

Objective	Not met/partially met/fully met	Further comments
1.a.	Fully met	• A title page could be used instead, that allows the user to select a view or see instructions
1.b.	Partially met	• Incorrect words are shown in a list on a separate view • But there is no formatting applied to incorrect words in the text field on the main view
1.c.i.	Fully met	• There is no formatting to be undone
1.c.ii.	Fully met	
1.c.iii.	Fully met	
1.d.	Partially met	• The user can change foreground and background colours • The user cannot change the font size or type
2.a.	Fully met	• Incorrect words are found effectively • But there is no special formatting applied to these words
2.b.	Fully met	
3.a.	Fully met	• This process is quite slow and could probably be optimised to run faster
3.b.	Fully met	• The algorithm only recommends up to ten words at a a time, and more can be requested
3.c.	Fully met	• There is no formatting to be undone
3.d.	Fully met	• The user can rest the "More words" button, which reruns the algorithm and increases the maximum scope
3.e.	Not met	• I have not met this objective as I felt it would help keep the interface simpler and easier to use
4.a.	Fully met	• Was originally an extension objective • Only missing spaces are checks for • Could be improved to check for: capital letters; missing punctuation; and erroneous punctuation
4.b.	Fully met	
5.a.	Fully met	• All features can be changed independently • Some of them aren't necessary, as the feature is only used on an indecent button press
5.b.	Fully met	• All settings are saved and reapplied on starting the program
6.a.	Fully met	• Even if a word hasn't been identified as incorrect, it can be added to the dictionary (only in the dictionary view)

6.b.i.	Fully met	• Automatically sorted • User can't sort form z-a
6.b.ii.	Fully met	• The filtering works but it a bit slow • Would have been nice to get rid of the button required to filter and just have the radio group
6.b.iii.	Fully met	• The user can search fr all words beginning with a specified string
6.b.iv.	Fully met	• Instead of just showing one definition, multiple brief definitions are shown (as requested by my end user in the interviews)
6.b.v.	Fully met	• Any word can be removed from the dictionary, even if it has not been previously added
7.a.	Fully met	• Acronyms can be added to the dictionary • Acronyms are changed when found in the text • Acronyms can't be removed or viewed using the interface, only by opening them from the text file
7.b.	Fully met	

5.3 End user feedback

After I finished writing my code, I sat down with Gabi and asked her to provide some feedback on the program. I also showed her my objectives and asked her to judge the program against them. I recorded a voice recording of the conversation and condemned it down to the following points:

- Positive points:
  - The settings page was one of her favourite features and she especially liked the saving and fetching of applied settings
  - The basic grammar checking is a nice and simple feature
  - The dictionary view was also heavily praised by Gabi, and she liked how the definitions were returned with multiple simple definitions (as was outlined in the objectives)
- Negative points:
  - More detailed instructions (or an instruction file/page) would have been nice to make it easier to use the app
  - She would have preferred to choose between being able to scroll on the word lists or use next/previous page buttons
  - It would have been ice to change the font type and size, but she did say this wasn't a priority in my initial interviews
- General comments:
  - The program meets most of the objectives to a high standard
  - It isn't the easiest app to use, but with some daily use she could get used to it

5.4 System improvements

Using the feedback provided by my end user and my own judgement, I have come up with the following points should i make the program again.

5.4.1 Algorithmic improvements

The algorithms all word effectively, but could be optimised to make them return values faster when called. This would allow the user to use the program faster and easier. It would also reduce the chance of users clicking buttons multiple times and overwhelming the program.

I would also increase the size of the bit array in the bloom filter this would reduce the rate of false positives, therefore increasing the effectiveness of the spell checker. This would however increase the time complexity of the bloom filter algorithm, so extra optimisation would be needed to ensure the program is still useable.

I am especially happy with having the ability to define words, as this was something that I didn't think i would initially be able to do with my prior knowledge and took a lot of research to make it possible.

5.4.2 Usability improvements

If I was to make the program again, I would spend more time searching for a framework or library to use for the graphical user interface. This is because Terminal.GUI wasn't very dynamic, so it was difficult to create views with unknown (lengths of) data. However, Terminal.GUI had simple syntax and allowed me to quickly create the views without needing to learn a new language. So I think it is suited better for simple project that use constants and static data. But Terminal.GUI provided a simple framework and had sufficient documentation which made creation simpler and les time consuming.

I would also spend more time asking Gabi (my end user) how she would like to interface to look and how navigation could work across views. This would have saved me time when creating the program, and would have allowed me to create detailed interface diagrams to propose as potential solutions. It would also ensure that I would meet all objectives and requirements to a high standard, where Gabi is happy with the usability feel of the interface.

6 References

Ref. no.	Page no.	Reference name	Reference source
1	5	Grammarly	https://www.grammarly.com
2	6	Online-spellcheck	https://www.online-spellcheck.com
3	8	Triangle inequality definition	https://en.wikipedia.org/wiki/Triangle_inequality
4	8	Example of the triangle inequality	https://en.wikipedia.org/wiki/File:TriangleInequality.svg
5	8	Metric definition	https://en.wikipedia.org/wiki/Metric_(mathematics)
6	8	String metric definition	https://en.wikipedia.org/wiki/String_metric
7	8	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	9	Bloom filter	https://en.wikipedia.org/wiki/Bloom_filter
11		Example of a Bloom Filter	https://en.wikipedia.org/wiki/File:Bloom_filter.svg
12	11	BK tree	https://en.wikipedia.org/wiki/BK-tree
13	11	Example of a BK tree	https://commons.wikimedia.org/wiki/File:Bk_tree.svg#/media/File:Bk_tree.svg
14	11	Hamming distance	https://en.wikipedia.org/wiki/Hamming_distance
15	15	Bloom filter calculator	https://hur.st/bloomfilter/?n=101988&p=1.0E-5&m=&k=10
16	15	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	16	List of hash functions	https://en.wikipedia.org/wiki/List_of_hash_functions
18	16	Pearson hashing	https://en.wikipedia.org/wiki/Pearson_hashing
19	16/31	Hashing by cyclic polynomial (Buzhash)	https://en.wikipedia.org/wiki/Rolling_hash#Cyclic_polynomial
20	16/31	Fowler-Noll-Vo (FNV-0) hash	https://en.wikipedia.org/wiki/Fowler–Noll–Vo_hash_function

21	16/31	djb2 hash	http://www.cse.yorku.ca/~oz/hash.html
22	16/32	sdbm hash	http://www.cse.yorku.ca/~oz/hash.html
23	16/32	PJW hash function	https://en.wikipedia.org/wiki/PJW_hash_function
24	16/32	Fast-Hash	https://en.wikipedia.org/wiki/Jenkins_hash_function
25	16/33	Rabin fingerprint	https://en.wikipedia.org/wiki/Rabin_fingerprint
26	16/33	Fletcher-32 hash	https://en.wikipedia.org/wiki/Fletcher%27s_checksum
27	16/33	CRC32 hash	https://en.wikipedia.org/wiki/CRC-32
28	23	Word list	https://github.com/dwyl/english-words/blob/master/words_alpha.zip
29	25	Dictionary API	https://dictionaryapi.dev/
30	25	Terminal.GUI	https://github.com/gui-cs/Terminal.Gui
31	21	BK tree explanation	Page 10
32	28	Bloom filter explanation	Page 8
33	47	True word generator	https://randomwordgenerator.com/
34	47	False string generator	Page 66 Lines 01 - 20

7 Appendix

7.1 Example JSON data returned


```
001: [
002:     {
003:         "word": "hello",
004:         "phonetics": [
005:             {
006:                 "audio": "https://api.dictionaryapi.dev/media/pronunciations/en/hello-au.mp3",
007:                 "sourceUrl": "https://commons.wikimedia.org/w/index.php?curid=75797336",
008:                 "license": {
009:                     "name": "BY-SA 4.0",
010:                     "url": "https://creativecommons.org/licenses/by-sa/4.0"
011:                 }
012:             },
013:             {
014:                 "text": "/h…ôÀàl…ô ä/",
015:                 "audio": "https://api.dictionaryapi.dev/media/pronunciations/en/hello-uk.mp3",
016:                 "sourceUrl": "https://commons.wikimedia.org/w/index.php?curid=9021983",
017:                 "license": {
018:                     "name": "BY 3.0 US",
019:                     "url": "https://creativecommons.org/licenses/by/3.0/us"
020:                 }
021:             },
022:             {
023:                 "text": "/h…ôÀàlo ä/",
024:                 "audio": ""
025:             }
026:         ],
027:         "meanings": [
028:             {
029:                 "partOfSpeech": "noun",
030:                 "definitions": [
031:                     {
032:                         "definition": "\"Hello!\" or an equivalent greeting.",
033:                         "synonyms": [],
034:                         "antonyms": []
035:                     }
036:                 ],
037:                 "synonyms": [
038:                     "greeting"
039:                 ],
040:                 "antonyms": []
041:             },
042:             {
043:                 "partOfSpeech": "verb",
044:                 "definitions": [
045:                     {
046:                         "definition": "To greet with \"hello\".",
047:                         "synonyms": [],
048:                         "antonyms": []
049:                     }
050:                 ],
051:                 "synonyms": [],
052:                 "antonyms": []
053:             },
054:             {
055:                 "partOfSpeech": "interjection",
056:                 "definitions": [
057:                     {
058:                         "definition": "A greeting (salutation) said when meeting someone or acknowledging someone,Äôs arrival or
presence.",
059:                         "synonyms": [],
060:                         "antonyms": [],
061:                         "example": "Hello, everyone."
062:                     },
063:                     {
064:                         "definition": "A greeting used when answering the telephone.",
065:                         "synonyms": [],
066:                         "antonyms": [],
067:                         "example": "Hello? How may I help you?"
068:                     },
069:                     {
070:                         "definition": "A call for response if it is not clear if anyone is present or listening, or if a
telephone conversation may have been disconnected.",
071:                         "synonyms": [],
072:                         "antonyms": [],
073:                         "example": "Hello? Is anyone there?"
074:                     },
```

```
075:                     {
076:                         "definition": "Used sarcastically to imply that the person addressed or referred to has done something
the speaker or writer considers to be foolish.",
077:                         "synonyms": [],
078:                         "antonyms": [],
079:                         "example": "You just tried to start your car with your cell phone. Hello?"
080:                     },
081:                     {
082:                         "definition": "An expression of puzzlement or discovery.",
083:                         "synonyms": [],
084:                         "antonyms": [],
085:                         "example": "Hello! What,Äôs going on here?"
086:                     }
087:                 ],
088:                 "synonyms": [],
089:                 "antonyms": [
090:                     "bye",
091:                     "goodbye"
092:                 ]
093:             }
094:         ],
095:         "license": {
096:             "name": "CC BY-SA 3.0",
097:             "url": "https://creativecommons.org/licenses/by-sa/3.0"
098:         },
099:         "sourceUrls": [
100:             "https://en.wiktionary.org/wiki/hello"
101:         ]
102:     }
103: ]
```

7.2 Python code

```
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()
```

7.3 Generate random strings

```
01: using System;
02:
03: public class HelloWorld
04: {
05:     public static void Main(string[] args)
06:     {
07:         Random r = new Random();
08:         for (int a = 0; a < 5; a++)
09:         {
10:             int num = r.Next(1,15);
11:             string word = "";
12:             for (int b = 1; b < num; b++)
13:             {
14:                 int alpha = r.Next(97,123);
15:                 word += (char)alpha;
16:             }
17:             Console.WriteLine(word);
18:         }
19:     }
20: }
```

7.4 C# code

```
<!--//////////////-->
<!-- 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>
//////////////////
// Utilities.cs //
//////////////////
// install dependencies
using System.Text;
using System.Net;
using Newtonsoft.Json;
// suppress warnings
# pragma warning disable
namespace Utilities
{
    ////////////
    // 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);
```

```
0076:         }
0077:         protected void CreateNode(string word, string parent, int weight)
0078:         {
0079:             // checks if an edge from the root node already exists with the same weight
0080:             // if it doesn't, then create node as nornmal from the root
0081:             // if it does, then use that node (that has the same weight) as the new parent
0082:             // due to the nature of BK-trees every edge from a parent must have a distinct weight
0083:             string? exists = CheckEdges(parent, weight);
0084:             if (exists is not null)
0085:             {
0086:                 // but weight from the parent might be different to the weight from the node, so recalculate
0087:                 int newWeight = leven.Calculate(parent, word);
0088:                 CreateNode(word, exists, newWeight);
0089:             }
0090:             else
0091:             {
0092:                 Connection thisConnection = new Connection();
0093:                 thisConnection.word = word;
0094:                 thisConnection.weight = weight;
0095:                 if (!tree.ContainsKey(thisConnection.word))
0096:                 {
0097:                     tree.Add(thisConnection.word, new List<Connection>());
0098:                     tree[parent].Add(thisConnection);
0099:                 }
0100:             }
0101:         }
0102:         // search each edge from a specified parent to see if a particular weight already exists
0103:         protected string CheckEdges(string word, int weight)
0104:         {
0105:             List<int> edges = new List<int>();
0106:             for (int a = 0; a < tree[word].Count; a++)
0107:             {
0108:                 edges.Add(tree[word][a].weight);
0109:             }
0110:             for (int a = 0; a < edges.Count; a++)
0111:             {
0112:                 if (edges[a] == weight)
0113:                 {
0114:                     return tree[word][a].word;
0115:                 }
0116:             }
0117:             return null;
0118:         }
0119:         public void ClearTree()
0120:         {
0121:             for (int a = 0; a < tree.Count; a++)
0122:             {
0123:                 tree.Remove(words[a]);
0124:             }
0125:             tree.Clear();
0126:         }
0127:         // input 1 when ReturnClosest is called in program
0128:         // public List<string> ReturnClosest(int maxWeight)
0129:         public string[] ReturnClosest(int maxWeight)
0130:         {
0131:             // used as a dummy to store the index of last word used as parent
0132:             Connection dummyConnection = new Connection();
0133:             dummyConnection.word = "";
0134:             dummyConnection.weight = 0;
0135:             string parent = FindParent(maxWeight);
0136:             List<Connection> closeWords = tree[parent];
0137:             while (closeWords.Count < 15)
0138:             {
0139:                 if (!(dummyConnection.weight < closeWords.Count))
0140:                 {
0141:                     break;
0142:                 }
0143:                 parent = closeWords[dummyConnection.weight].word;
0144:                 closeWords.AddRange(tree[parent]);
0145:                 dummyConnection.weight++;
0146:             }
0147:             string[] wordsOut = new string[closeWords.Count];
0148:             for (int a = 0; a < closeWords.Count; a++)
0149:             {
0150:                 wordsOut[a] = closeWords[a].word;
0151:             }
0152:             return wordsOut;
0153:         }
```

```
0154:         protected string FindParent(int maxWeight)
0155:         {
0156:             bool exists = false;
0157:             while (!exists)
0158:             {
0159:                 for (int a = 0; a < tree[root].Count; a++)
0160:                 {
0161:                     if (tree[root][a].weight < maxWeight && tree[root][a].word != root)
0162:                     {
0163:                         return tree[root][a].word;
0164:                     }
0165:                 }
0166:                 if (!exists)
0167:                 {
0168:                     maxWeight++;
0169:                 }
0170:             }
0171:             return null;
0172:         }
0173:     }
0174:     /////////////////
0175:     // BloomFilter //
0176:     /////////////////
0177:     class BloomFilter
0178:     {
0179:         protected int[] filter = new int[2682975];
0180:         // FIRST TIME USE
0181:         //
0182:         // constructor used to reset all indexes in the filter to 0
0183:         // then setup the filter to the text file
0184:         public BloomFilter(int num)
0185:         {
0186:             for (int a = 0; a < filter.Length; a++)
0187:             {
0188:                 filter[a] = 0;
0189:             }
0190:             SetUp();
0191:         }
0192:         // reads the list of words from text file
0193:         // inserts each word into filter
0194:         // write the filter to a text file
0195:         public void SetUp()
0196:         {
0197:             string[] words = File.ReadAllLines("Resources/TrueWords.txt");
0198:             foreach (var a in words)
0199:             {
0200:                 Insert(a);
0201:             }
0202:             WriteFilter();
0203:         }
0204:         // write the filter to a text file
0205:         protected void WriteFilter()
0206:         {
0207:             string filename = "Resources/Filter.txt";
0208:             string[] stringArray = filter.Select(x => x.ToString()).ToArray();
0209:             string result = String.Concat(stringArray);
0210:             File.WriteAllText(filename, result);
0211:         }
0212:         // GENERAL USE
0213:         //
0214:         // read the filter from a text file and store as an int array
0215:         public BloomFilter()
0216:         {
0217:             string filename = "Resources/Filter.txt";
0218:             string text = File.ReadAllText(filename);
0219:             for (uint a = 0; a < filter.Length; a++)
0220:             {
0221:                 filter[a] = text[(int)a] - 48;
0222:             }
0223:         }
0224:         public bool Lookup(string word)
0225:         {
0226:             // there are only two one letter words that are in the set - "a" and "i"
0227:             // however all one letter words return true, so check all one letter words
0228:             if (word.Length == 1)
0229:             {
0230:                 return (word == "a" || word == "i");
0231:             }
```

```
0232:             else
0233:             {
0234:                 // if all the indexes are 1, then the item may be in the set
0235:                 // if any of the indexes are 0, then the item is definitely not in the set
0236:                 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)
0237:                 {
0238:                     return true;
0239:                 }
0240:                 // if any of the indexes are 0, then do a binary search to check the list of user-added words
0241:                 else
0242:                 {
0243:                     return BinarySearch(word);
0244:                 }
0245:             }
0246:         }
0247:         public bool BinarySearch(string target)
0248:         {
0249:             target = target.ToLower();
0250:             string[] words = File.ReadAllLines("Resources/AddedWords.txt");
0251:             int min = 0;
0252:             int max = words.Length;
0253:             int mid;
0254:             while (min < max)
0255:             {
0256:                 mid = (min + max) / 2;
0257:                 if (words[mid] == target)
0258:                 {
0259:                     return true;
0260:                 }
0261:                 else if (words[mid].CompareTo(target) < 0)
0262:                 {
0263:                     min = mid + 1;
0264:                 }
0265:                 else
0266:                 {
0267:                     max = mid - 1;
0268:                 }
0269:             }
0270:             return false;
0271:         }
0272:         protected void Insert(string word)
0273:         {
0274:             filter[Hash1(word)] = 1;
0275:             filter[Hash2(word)] = 1;
0276:             filter[Hash3(word)] = 1;
0277:             filter[Hash4(word)] = 1;
0278:             filter[Hash5(word)] = 1;
0279:             filter[Hash6(word)] = 1;
0280:             filter[Hash7(word)] = 1;
0281:             filter[Hash8(word)] = 1;
0282:             filter[Hash9(word)] = 1;
0283:             filter[Hash10(word)] = 1;
0284:             WriteFilter();
0285:         }
0286:         // Pearson Hashing
0287:         protected uint Hash1(string word)
0288:         {
0289:             uint hash = 0;
0290:             byte[] bytes = Encoding.UTF8.GetBytes(word);
0291:             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 };
0292:             foreach (var a in bytes)
0293:             {
0294:                 hash = nums[((hash ^ a) % (uint)nums.Length)];
0295:             }
0296:             hash %= (uint)filter.Length;
0297:             return hash;
0298:         }
0299:         // Hashing by cyclic polynomial (Buzhash)
0300:         protected uint Hash2(string word)
0301:         {
```

```
0302:             uint hash = 1;
0303:             foreach (char a in word)
0304:             {
0305:                 hash = CircularShift(hash) ^ CircularShift((byte)a) ^ CircularShift((uint)(byte)a + 1);
0306:             }
0307:             hash %= (uint)filter.Length;
0308:             return hash;
0309:         }
0310:         // Fowler–Noll–Vo (FNV–0) hash
0311:         protected uint Hash3(string word)
0312:         {
0313:             uint hash = 0;
0314:             byte[] bytes = Encoding.UTF8.GetBytes(word);
0315:             foreach (var a in bytes)
0316:             {
0317:                 hash = (hash * 16777619) ^ a;
0318:             }
0319:             hash %= (uint)filter.Length;
0320:             return hash;
0321:         }
0322:         // dijb2
0323:         protected uint Hash4(string word)
0324:         {
0325:             // randomly generated number
0326:             uint hash = 5381;
0327:             byte[] bytes = Encoding.UTF8.GetBytes(word);
0328:             foreach (var b in bytes)
0329:             {
0330:                 hash = ((hash << 5) + hash) + 33;
0331:             }
0332:             hash %= (uint)filter.Length;
0333:             return hash;
0334:         }
0335:         // sdbm
0336:         protected uint Hash5(string word)
0337:         {
0338:             uint hash = 0;
0339:             byte[] bytes = Encoding.UTF8.GetBytes(word);
0340:             foreach (var a in bytes)
0341:             {
0342:                 hash = 65599 + (hash << 6) + (hash << 16) - hash;
0343:             }
0344:             hash %= (uint)filter.Length;
0345:             return hash;
0346:         }
0347:         // PJW hash function
0348:         protected uint Hash6(string word)
0349:         {
0350:             uint hash = 0;
0351:             uint bits = (sizeof(uint) * 8);
0352:             uint max = (uint)(0xFFFFFFFF) << (int)(bits - (bits / 8));
0353:             for (int a = 0; a < word.Length; a++)
0354:             {
0355:                 hash = hash << (int)(bits / 8) + word[a];
0356:                 if (max != 0)
0357:                 {
0358:                     hash = hash ^ (max >> (int)bits * 3 / 4) & (~max);
0359:                 }
0360:             }
0361:             hash %= (uint)filter.Length;
0362:             return hash;
0363:         }
0364:         // Fast–Hash
0365:         protected uint Hash7(string word)
0366:         {
0367:             // randomly generated number
0368:             uint a = unchecked((uint)0x880355f21e6d1965);
0369:             uint hash = 144 ^ ((uint)word.Length * a);
0370:             uint b = 0;
0371:             for (uint c = 0; c < word.Length; c++)
0372:             {
0373:                 hash = (hash ^ Mix(c)) * a;
0374:             }
0375:             int d = word.Length & 7;
0376:             d %= word.Length;
0377:             int e = (d - 1) * 8;
0378:             if (e < 0)
0379:             {
```

```
0380:                 e = 0;
0381:             }
0382:             b ^= (uint)word[d] << d;
0383:             hash ^= Mix(b) * a;
0384:             hash = Mix(hash);
0385:             hash %= (uint)filter.Length;
0386:             return (uint)hash;
0387:         }
0388:         // Rabin Fingerprint
0389:         protected uint Hash8(string word)
0390:         {
0391:             uint hash = word[0];
0392:             uint length = (uint)word.Length;
0393:             for (int a = 1; a < length; a++)
0394:             {
0395:                 hash += (uint)Math.Pow(word[a] * length, a);
0396:             }
0397:             hash %= (uint)filter.Length;
0398:             return hash;
0399:         }
0400:         // Fletcher-32
0401:         protected uint Hash9(string word)
0402:         {
0403:             uint a = 0;
0404:             uint b = 0;
0405:             for (int c = 0; c < word.Length; c++)
0406:             {
0407:                 a = (a + word[c] % (uint)0xffff);
0408:                 b = (a + b) % (uint)0xffff;
0409:             }
0410:             uint hash = (b << 16) | a;
0411:             hash %= (uint)filter.Length;
0412:             return hash;
0413:         }
0414:         // CRC32
0415:         protected uint Hash10(string word)
0416:         {
0417:             // randomly generated number
0418:             uint hash = 0xffffffff;
0419:             uint index;
0420:             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 };
0421:             byte[] bytes = Encoding.UTF8.GetBytes(word);
0422:             foreach (var b in bytes)
0423:             {
0424:                 index = ((hash ^ b) & 0xff) % (uint)nums.Length;
0425:                 hash = (hash >> 8) ^ nums[index];
0426:             }
0427:             hash ^= 0xffffffff;
0428:             hash %= (uint)filter.Length;
0429:             return hash;
0430:         }
0431:         // helpers
0432:         // for hash2
0433:         protected uint CircularShift(uint a)
0434:         {
0435:             uint b = a << 1 | a >> 31;
0436:             return b;
0437:         }
0438:         // for hash7
0439:         protected uint Mix(uint hash)
0440:         {
0441:             long a = (long)hash;
0442:             a ^= a >> 23;
0443:             // randomly generated number
0444:             a *= 0x2127599bf4325c37;
0445:             a ^= a >> 47;
0446:             hash = (uint)a;
0447:             return (uint)hash;
0448:         }
0449:     }
0450:     ////////////////
```

```
0451:     // Connection //
0452:     ////////////////
0453:     public struct Connection
0454:     {
0455:         // node
0456:         public string word;
0457:         // edge to parent
0458:         public int weight;
0459:     }
0460:     ////////////////
0461:     // Levenshtein //
0462:     ////////////////
0463:     class LevenshteinDistance
0464:     {
0465:         protected int distance;
0466:         public LevenshteinDistance()
0467:         {
0468:             distance = 0;
0469:         }
0470:         public int Calculate(string a, string b)
0471:         {
0472:             int cost;
0473:             // if a or b has no length, return the other one
0474:             if (a.Length == 0)
0475:             {
0476:                 return b.Length;
0477:             }
0478:             else if (b.Length == 0)
0479:             {
0480:                 return a.Length;
0481:             }
0482:             // row and column sizes are (word.Length + 1) because row/column 1 store index values
0483:             int[,] matrix = new int[a.Length + 1, b.Length + 1];
0484:             // store index values in row/column 1, example
0485:             // 0 1 ...
0486:             // 1
0487:             // 2
0488:             // ...
0489:             for (int c = 0; c < matrix.GetLength(0); c++)
0490:             {
0491:                 matrix[c, 0] = c;
0492:                 for (int d = 0; d < matrix.GetLength(1); d++)
0493:                 {
0494:                     matrix[0, d] = d;
0495:                 }
0496:             }
0497:             for (int c = 1; c < matrix.GetLength(0); c++)
0498:             {
0499:                 for (int d = 1; d < matrix.GetLength(1); d++)
0500:                 {
0501:                     // if the values at the index are the same then no change is needed
0502:                     if (a[c - 1] == b[d - 1])
0503:                     {
0504:                         cost = 0;
0505:                     }
0506:                     // if not then a change (substitution, deletion or addition) is needed
0507:                     else
0508:                     {
0509:                         cost = 1;
0510:                     }
0511:                     int value1 = matrix[c - 1, d] + 1;
0512:                     int value2 = matrix[c, d - 1] + 1;
0513:                     int value3 = matrix[c - 1, d - 1] + cost;
0514:                     int[] values = { value1, value2, value3 };
0515:                     matrix[c, d] = values.Min();
0516:                 }
0517:             }
0518:             // final cost is stored in the bottom right of the matrix
0519:             distance = matrix[a.Length, b.Length];
0520:             return distance;
0521:         }
0522:         public int[] CalculateAll(string a, string[] words)
0523:         {
0524:             int[] distances = new int[words.Length];
0525:             for (int b = 0; b < words.Length; b++)
0526:             {
0527:                 distances[b] = Calculate(a, words[b]);
0528:             }
```

```
0529:             return distances;
0530:         }
0531:     }
0532: }
0533: /////////////////
0534: // DefineWords //
0535: /////////////////
0536: namespace DefineWords
0537: {
0538:     // create template to store JSON data retruned by API call
0539:     class Word
0540:     {
0541:         public string? word { get; set; }
0542:         public string? phonetic { get; set; }
0543:         public List<Phonetics>? phonetics { get; set; }
0544:         public List<Meanings>? meanings { get; set; }
0545:         public License? license { get; set; }
0546:         public List<string>? sourceUrls { get; set; }
0547:     }
0548:     class Phonetics
0549:     {
0550:         public string? text { get; set; }
0551:         public string? audio { get; set; }
0552:         public string? sourceUrl { get; set; }
0553:         public License? license { get; set; }
0554:     }
0555:     class License
0556:     {
0557:         public string? name { get; set; }
0558:         public string? url { get; set; }
0559:     }
0560:     class Meanings
0561:     {
0562:         public string? partOfSpeech { get; set; }
0563:         public List<Definitions>? definitions { get; set; }
0564:         public List<string>? synonyms { get; set; }
0565:         public List<string>? antonyms { get; set; }
0566:     }
0567:     class Definitions
0568:     {
0569:         public string? definition { get; set; }
0570:         public List<string>? synonyms { get; set; }
0571:         public List<string>? antonyms { get; set; }
0572:         public string? example { get; set; }
0573:     }
0574:     class Program
0575:     {
0576:         public static List<string> DefineWord(string toDefine)
0577:         {
0578:             WebRequest request = WebRequest.Create("https://api.dictionaryapi.dev/api/v2/entries/en/" + toDefine);
0579:             HttpWebResponse response = (HttpWebResponse)request.GetResponse();
0580:             Stream dataStream = response.GetResponseStream();
0581:             StreamReader reader = new StreamReader(dataStream);
0582:             string responseFromServer = reader.ReadToEnd();
0583:             IEnumerable<Word>? iword = JsonConvert.DeserializeObject<IEnumerable<Word>>(responseFromServer);
0584:             Word[] aword = new List<Word>(iword).ToArray();
0585:             Word word = aword[0];
0586:             // makes sure there is only one definition for each word type
0587:             Dictionary<string, string> definitions = new Dictionary<string, string>()
0588:             {
0589:                 {"noun", "null"},
0590:                 {"pronoun", "null"},
0591:                 {"verb", "null"},
0592:                 {"adjective", "null"},
0593:                 {"adverb", "null"},
0594:                 {"preposition", "null"},
0595:                 {"conjunction", "null"},
0596:                 {"interjection", "null"},
0597:             };
0598:             // if the definition stored for the word type is null, then store the definiton
0599:             for (int a = 0; a < word.meanings.Count; a++)
0600:             {
0601:                 if (definitions[word.meanings[a].partOfSpeech] == "null")
0602:                 {
0603:                     definitions[word.meanings[a].partOfSpeech] = word.meanings[a].definitions[0].definition;
0604:                 }
0605:             }
0606:             // convert values in dictionary to list
```

```
0607:             List<string> finals = new List<string>();
0608:             foreach (KeyValuePair<string, string> kvPair in definitions)
0609:             {
0610:                 if (kvPair.Value != "null")
0611:                 {
0612:                     string key = char.ToUpper(kvPair.Key[0]) + kvPair.Key.Substring(1);
0613:                     finals.Add(key + " - " + kvPair.Value);
0614:                 }
0615:             }
0616:             return finals;
0617:         }
0618:         static void Main(string[] args)
0619:         {
0620:             string word = Console.ReadLine();
0621:             List<string> definitions = DefineWord(word);
0622:             foreach (var definition in definitions)
0623:             {
0624:                 Console.WriteLine(definition);
0625:             }
0626:         }
0627:     }
0628: }
0629: Ôª¤/////////////////
0630: // Program.cs //
0631: /////////////////
0632: // install dependencies
0633: using Views;
0634: using Terminal.Gui;
0635: // suppress warnings
0636: # pragma warning disable
0637: Application.Init();
0638: try
0639: {
0640:     // used to fetch settings from saved file
0641:     Settings thisSettings = new Settings();
0642:     Application.Run(new MainView());
0643: }
0644: finally
0645: {
0646:     Application.Shutdown();
0647: }
0648: ////////////////////////////////////////
0649: // View designers/MasterView.Designer.cs //
0650: ////////////////////////////////////////
0651: // install dependencies
0652: using Terminal.Gui;
0653: // suppress warnings
0654: # pragma warning disable
0655: namespace Views
0656: {
0657:     public partial class MasterView : Window
0658:     {
0659:         public void Build()
0660:         {
0661:             Title = "Spell Checker " + this;
0662:             var menuBar = new MenuBar(new MenuBarItem[] {
0663:                 new MenuBarItem("_File", new MenuItem[] {
0664:                     new MenuItem("_Quit", "", () => {
0665:                         Application.Shutdown();
0666:                     }),
0667:                     new MenuItem("_Settings", "", () => {
0668:                         Application.Run(new Settings());
0669:                     }),
0670:                     new MenuItem("_Main View", "", () => {
0671:                         Application.Run(new MainView());
0672:                     }),
0673:                     new MenuItem("_Dictionary", "", () => {
0674:                         Application.Run(new Dictionary());
0675:                     })
0676:                 })
0677:             });
0678:             Add(menuBar);
0679:         }
0680:     }
0681: }
0682: ////////////////////////////////////////
0683: // View designers/Settings.Designer.cs //
0684: ////////////////////////////////////////
```

```
0685: // install dependencies
0686: using Terminal.Gui;
0687: // suppress warnings
0688: # pragma warning disable
0689: namespace Views
0690: {
0691:     public partial class Settings : MasterView
0692:     {
0693:         protected Button viewDictBtn = new Button();
0694:         protected Button saveBtn = new Button();
0695:         protected Button resetBtn = new Button();
0696:         protected Button acronymBtn = new Button();
0697:         protected CheckBox spellCheckCbx = new CheckBox();
0698:         protected CheckBox changeAcronymsCbx = new CheckBox();
0699:         protected CheckBox grammarCheckCbx = new CheckBox();
0700:         protected ColorPicker backgroundPick = new ColorPicker();
0701:         protected ColorPicker foregroundPick = new ColorPicker();
0702:         protected Label spellCheckLab = new Label();
0703:         protected Label viewDictLab = new Label();
0704:         protected Label maxLengthLab = new Label();
0705:         protected Label changeAcryonymsLab = new Label();
0706:         protected Label grammarCheckLab = new Label();
0707:         protected Label shortAcrLab = new Label();
0708:         protected Label fullAcrLab = new Label();
0709:         protected TextField maxLengthTxt = new TextField();
0710:         protected TextField shortAcrTxt = new TextField();
0711:         protected TextField fullAcrTxt = new TextField();
0712:         protected void InitializeComponent()
0713:         {
0714:             // build labels
0715:             spellCheckLab.Text = "Spell check";
0716:             spellCheckLab.X = 2;
0717:             spellCheckLab.Y = 2;
0718:             viewDictLab.Text = "View dictionary";
0719:             viewDictLab.X = spellCheckLab.X;
0720:             viewDictLab.Y = spellCheckLab.Y + 2;
0721:             maxLengthLab.Text = "Maximum length                For no max length, enter \"0\"";
0722:             maxLengthLab.X = spellCheckLab.X;
0723:             maxLengthLab.Y = viewDictLab.Y + 2;
0724:             changeAcryonymsLab.Text = "Change acronyms";
0725:             changeAcryonymsLab.X = spellCheckLab.X;
0726:             changeAcryonymsLab.Y = maxLengthLab.Y + 2;
0727:             grammarCheckLab.Text = "Grammar check";
0728:             grammarCheckLab.X = spellCheckLab.X;
0729:             grammarCheckLab.Y = changeAcryonymsLab.Y + 2;
0730:             Add(spellCheckLab, viewDictLab, maxLengthLab, changeAcryonymsLab, grammarCheckLab);
0731:             // build check boxes
0732:             spellCheckCbx.X = spellCheckLab.X + 20;
0733:             spellCheckCbx.Y = spellCheckLab.Y;
0734:             spellCheckCbx.Checked = spellCheck;
0735:             changeAcronymsCbx.X = spellCheckCbx.X;
0736:             changeAcronymsCbx.Y = changeAcryonymsLab.Y;
0737:             changeAcronymsCbx.Checked = changeAcronyms;
0738:             grammarCheckCbx.X = spellCheckCbx.X;
0739:             grammarCheckCbx.Y = grammarCheckLab.Y;
0740:             grammarCheckCbx.Checked = grammarCheck;
0741:             Add(spellCheckCbx, changeAcronymsCbx, grammarCheckCbx);
0742:             // build buttons
0743:             viewDictBtn.Text = "View";
0744:             viewDictBtn.X = spellCheckCbx.X;
0745:             viewDictBtn.Y = viewDictLab.Y;
0746:             saveBtn.Text = "_Save";
0747:             saveBtn.X = maxLengthLab.X;
0748:             saveBtn.Y = grammarCheckLab.Y + 2;
0749:             saveBtn.IsDefault = true;
0750:             resetBtn.X = maxLengthLab.X;
0751:             resetBtn.Y = saveBtn.Y + 2;
0752:             resetBtn.Text = "Reset to default settings";
0753:             Add(viewDictBtn, saveBtn, resetBtn);
0754:             // build text fields
0755:             maxLengthTxt.X = spellCheckCbx.X;
0756:             maxLengthTxt.Y = maxLengthLab.Y;
0757:             maxLengthTxt.Width = 8;
0758:             maxLengthTxt.Text = maxLength.ToString();
0759:             Add(maxLengthTxt);
0760:             // build color elements
0761:             backgroundPick.X = maxLengthLab.X;
0762:             backgroundPick.Y = resetBtn.Y + 2;
```

```
0763:             backgroundPick.Text = "Background colour";
0764:             backgroundPick.SelectedColor = background;
0765:             foregroundPick.X = 38;
0766:             foregroundPick.Y = backgroundPick.Y;
0767:             foregroundPick.Text = "Foreground colour";
0768:             foregroundPick.SelectedColor = foreground;
0769:             Add(backgroundPick, foregroundPick);
0770:             // build acronym elements
0771:             shortAcrTxt.X = foregroundPick.X;
0772:             shortAcrTxt.Y = grammarCheckLab.Y;
0773:             shortAcrTxt.Width = 8;
0774:             shortAcrLab.X = shortAcrTxt.X + 9;
0775:             shortAcrLab.Y = shortAcrTxt.Y;
0776:             shortAcrLab.Text = "Short acronym";
0777:             fullAcrTxt.X = shortAcrTxt.X;
0778:             fullAcrTxt.Y = shortAcrTxt.Y + 2;
0779:             fullAcrTxt.Width = 8;
0780:             fullAcrLab.X = shortAcrLab.X;
0781:             fullAcrLab.Y = fullAcrTxt.Y;
0782:             fullAcrLab.Text = "Full acronym";
0783:             acronymBtn.X = shortAcrTxt.X;
0784:             acronymBtn.Y = fullAcrTxt.Y + 2;
0785:             acronymBtn.Text = "Add acronym";
0786:             Add(shortAcrTxt, shortAcrLab, fullAcrTxt, fullAcrLab, acronymBtn);
0787:         }
0788:     }
0789: }
0790: ////////////////////////////////////////
0791: // View designers/MainView.Designer.cs //
0792: ////////////////////////////////////////
0793: // install dependencies
0794: using Terminal.Gui;
0795: // suppress warnings
0796: # pragma warning disable
0797: namespace Views
0798: {
0799:     public partial class MainView : MasterView
0800:     {
0801:         protected Button spellBtn = new Button();
0802:         protected Button clearBtn = new Button();
0803:         protected Label lengthLab = new Label();
0804:         protected TextView textEntry = new TextView();
0805:         protected void InitializeComponent()
0806:         {
0807:             //
0808:             textEntry.X = 1;
0809:             textEntry.Y = 2;
0810:             textEntry.Width = 76;
0811:             textEntry.Height = 15;
0812:             textEntry.WordWrap = true;
0813:             Add(textEntry);
0814:             //
0815:             spellBtn.Text = "Spell check";
0816:             spellBtn.X = 1;
0817:             spellBtn.Y = 20;
0818:             clearBtn.Text = "Clear text";
0819:             clearBtn.X = spellBtn.X + spellBtn.Text.Length + 6;
0820:             clearBtn.Y = spellBtn.Y;
0821:             Add(spellBtn, clearBtn);
0822:         }
0823:     }
0824: }
0825: ////////////////////////////////////////
0826: // View designers/Dictionary.Designer.cs //
0827: ////////////////////////////////////////
0828: // install dependencies
0829: using Terminal.Gui;
0830: using NStack;
0831: // suppress warnings
0832: # pragma warning disable
0833: namespace Views
0834: {
0835:     public partial class Dictionary : MasterView
0836:     {
0837:         protected Button lookupBtn = new Button();
0838:         protected Button nextBtn = new Button();
0839:         protected Button prevBtn = new Button();
0840:         protected Button defineBtn = new Button();
```

```
0841:         protected Button addBtn = new Button();
0842:         protected Button removeBtn = new Button();
0843:         protected Button filterBtn = new Button();
0844:         protected Label[] wordLabs = new Label[101988];
0845:         protected RadioGroup filterRad = new RadioGroup();
0846:         protected TextField lookupTxt = new TextField();
0847:         protected void InitializeComponent()
0848:         {
0849:             // build lookup elements
0850:             lookupTxt.X = 50;
0851:             lookupTxt.Y = 4;
0852:             lookupTxt.Width = 14;
0853:             lookupBtn.Text = "Lookup";
0854:             lookupBtn.X = lookupTxt.X;
0855:             lookupBtn.Y = lookupTxt.Y + 2;
0856:             Add(lookupBtn, lookupTxt);
0857:             // build navigation elements
0858:             nextBtn.Text = "Next page";
0859:             nextBtn.X = 64;
0860:             nextBtn.Y = 21;
0861:             prevBtn.Text = "Prev page";
0862:             prevBtn.X = nextBtn.X - 14;
0863:             prevBtn.Y = nextBtn.Y;
0864:             Add(prevBtn, nextBtn);
0865:             // build functional buttons
0866:             defineBtn.X = lookupTxt.X;
0867:             defineBtn.Y = lookupBtn.Y + 2;
0868:             defineBtn.Text = "Define word";
0869:             addBtn.X = lookupTxt.X;
0870:             addBtn.Y = defineBtn.Y + 2;
0871:             addBtn.Text = "Add word";
0872:             removeBtn.X = lookupTxt.X;
0873:             removeBtn.Y = addBtn.Y + 2;
0874:             removeBtn.Text = "Remove word";
0875:             Add(defineBtn, addBtn, removeBtn);
0876:             // build fitering elements
0877:             filterRad.X = lookupTxt.X;
0878:             filterRad.Y = removeBtn.Y + 2;
0879:             filterRad.RadioLabels = new ustring[] { "All words", "Default words", "Added words" };
0880:             filterBtn.X = lookupTxt.X;
0881:             filterBtn.Y = filterRad.Y + 4;
0882:             filterBtn.Text = "Filter list";
0883:             Add(filterRad, filterBtn);
0884:         }
0885:     }
0886: }
0887: ////////////////////////////////////////
0888: // View designers/SpellCheck.Designer.cs //
0889: ////////////////////////////////////////
0890: // install dependencies
0891: using Terminal.Gui;
0892: // suppress warnings
0893: # pragma warning disable
0894: namespace Views
0895: {
0896:     public partial class SpellCheck : MasterView
0897:     {
0898:         protected Button ignoreBtn = new Button();
0899:         protected Button userBtn = new Button();
0900:         protected Button acronymBtn = new Button();
0901:         protected Button homeBtn = new Button();
0902:         protected Button homeBtn2 = new Button();
0903:         protected Button replaceBtn = new Button();
0904:         protected Button addBtn = new Button();
0905:         protected Button resizeBtn = new Button();
0906:         protected Label lookupLab = new Label();
0907:         protected Label lookupLab2 = new Label();
0908:         protected Label lookupLab3 = new Label();
0909:         protected Label userLab = new Label();
0910:         protected List<Label> wordLabs = new List<Label>();
0911:         protected TextField lookupTxt = new TextField();
0912:         protected TextField userTxt = new TextField();
0913:         protected void InitializeComponent()
0914:         {
0915:             // build lookup elements
0916:             lookupLab.Text = "Enter word from left side";
0917:             lookupLab.X = 50;
0918:             lookupLab.Y = 2;
```

```
0919:             lookupLab2.Text = "in the field below.";
0920:             lookupLab2.X = lookupLab.X;
0921:             lookupLab2.Y = lookupLab.Y + 1;
0922:             lookupLab3.Text = "Then handle error";
0923:             lookupLab3.X = lookupLab.X;
0924:             lookupLab3.Y = lookupLab2.Y + 1;
0925:             lookupTxt.X = lookupLab.X;
0926:             lookupTxt.Y = lookupLab3.Y + 1;
0927:             lookupTxt.Width = 14;
0928:             Add(lookupLab, lookupLab2, lookupLab3, lookupTxt, lookupTxt);
0929:             //
0930:             ignoreBtn.X = lookupLab.X;
0931:             ignoreBtn.Y = lookupTxt.Y + 2;
0932:             ignoreBtn.Text = "Ignore error";
0933:             Add(ignoreBtn);
0934:             //
0935:             resizeBtn.Text = "Resize text to " + theseSettings.maxLength;
0936:             resizeBtn.X = lookupLab.X;
0937:             resizeBtn.Y = ignoreBtn.Y + 2;
0938:             Add(resizeBtn);
0939:             // build change error elements
0940:             userLab.Text = "Enter new word";
0941:             userLab.X = lookupLab.X;
0942:             userLab.Y = resizeBtn.Y + 2;
0943:             userTxt.X = lookupLab.X;
0944:             userTxt.Y = userLab.Y + 1;
0945:             userTxt.Width = 14;
0946:             userBtn.Text = "Replace";
0947:             userBtn.X = lookupLab.X;
0948:             userBtn.Y = userTxt.Y + 1;
0949:             replaceBtn.Text = "Recommend words";
0950:             replaceBtn.X = lookupLab.X;
0951:             replaceBtn.Y = userBtn.Y + 2;
0952:             Add(userLab, userTxt, userBtn, replaceBtn);
0953:             //
0954:             acronymBtn.Text = "Replace acronyms";
0955:             acronymBtn.X = lookupLab.X;
0956:             acronymBtn.Y = replaceBtn.Y + 2;
0957:             Add(acronymBtn);
0958:             //
0959:             addBtn.Text = "Learn spelling";
0960:             addBtn.X = lookupLab.X;
0961:             addBtn.Y = acronymBtn.Y + 2;
0962:             Add(addBtn);
0963:             //
0964:             homeBtn.Text = "Home";
0965:             homeBtn.X = lookupLab.X;
0966:             homeBtn.Y = addBtn.Y + 2;
0967:             Add(homeBtn);
0968:         }
0969:     }
0970: }
0971: //////////////////////////
0972: // Views/SpellCheck.cs //
0973: //////////////////////////
0974: // install dependencies
0975: using Terminal.Gui;
0976: using Utilities;
0977: using System.Text.RegularExpressions;
0978: using NStack;
0979: // suppress warnings
0980: # pragma warning disable
0981: namespace Views
0982: {
0983:     public partial class SpellCheck
0984:     {
0985:         protected int firstIndex;
0986:         protected string? error;
0987:         protected List<string> falseWords = new List<string>();
0988:         protected List<string> words = new List<string>();
0989:         protected List<string> ignoredWords = new List<string>();
0990:         protected Settings theseSettings = new Settings();
0991:         public SpellCheck(string textIn)
0992:         {
0993:             words.Clear();
0994:             falseWords.Clear();
0995:             words = textIn.Split(' ').ToList();
0996:             InitializeComponent();
```

```
0997:             DisplayWords();
0998:             if (falseWords.Count == 0)
0999:             {
1000:                 LeaveMessage();
1001:             }
1002:             else if (falseWords[0] == "")
1003:             {
1004:                 LeaveMessage();
1005:             }
1006:             ignoreBtn.Clicked += () =>
1007:             {
1008:                 error = lookupTxt.Text.ToString();
1009:                 if (error is not null)
1010:                 {
1011:                     ignoredWords.Add(error);
1012:                     MessageBox.Query("Ignored", error + " has been ignored as an erorr", "Ok");
1013:                     DisplayWords();
1014:                 }
1015:             };
1016:             acronymBtn.Clicked += () =>
1017:             {
1018:                 Settings theseSettings = new Settings();
1019:                 if (theseSettings.changeAcronyms)
1020:                 {
1021:                     ReplaceAcronyms();
1022:                     MessageBox.Query("Acronyms replaced", "All acronyms have been replaced", "Ok");
1023:                     DisplayWords();
1024:                 }
1025:             };
1026:             homeBtn.Clicked += () =>
1027:             {
1028:                 GoHome();
1029:             };
1030:             homeBtn2.Clicked += () =>
1031:             {
1032:                 GoHome();
1033:             };
1034:             userBtn.Clicked += () =>
1035:             {
1036:                 error = lookupTxt.Text.ToString();
1037:                 string newWord = userTxt.Text.ToString();
1038:                 if (newWord is not null)
1039:                 {
1040:                     ReplaceUserWord(newWord);
1041:                 }
1042:                 MessageBox.Query("Replaced", error + " has been replaced to " + newWord, "Ok");
1043:                 DisplayWords();
1044:             };
1045:             replaceBtn.Clicked += () =>
1046:             {
1047:                 error = lookupTxt.Text.ToString();
1048:                 string newWord = "";
1049:                 int maxWeight = 1;
1050:                 bool loop = true;
1051:                 string text;
1052:                 if (error is not null && falseWords.Contains(error))
1053:                 {
1054:                     BkTree bktree = new BkTree(error);
1055:                     do
1056:                     {
1057:                         string[] stringButtons = bktree.ReturnClosest(maxWeight);
1058:                         ustring[] buttons = new ustring[stringButtons.Length + 1];
1059:                         text = "";
1060:                         for (int a = 0; a < stringButtons.Length; a++)
1061:                         {
1062:                             buttons[a] = (a + 1).ToString();
1063:                             text += (a + 1) + ": " + stringButtons[a] + "   ";
1064:                         }
1065:                         text += "\nPress escape to exit";
1066:                         buttons[buttons.Length - 1] = "More words";
1067:                         int choice = MessageBox.Query("Recommended words: " + buttons.Length, text, buttons);
1068:                         if (choice == -1)
1069:                         {
1070:                             return;
1071:                         }
1072:                         else if (choice == buttons.Length - 1)
1073:                         {
1074:                             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];
```

```
1153:                 Match match = Regex.Match(word, @"[^\w\s]");
1154:                 // if a word contains a special character
1155:                 if (match.Success && word.Length != 1 && match.Value != "'")
1156:                 {
1157:                     string specialChar = word[match.Index].ToString();
1158:                     string part1 = "";
1159:                     string part2 = "";
1160:                     // checks either side of the special character to ensure it is not null
1161:                     //  then stores the values to new strings
1162:                     if (word[0] != char.Parse(specialChar))
1163:                     {
1164:                         part1 = word.Substring(0, match.Index);
1165:                     }
1166:                     if (word.IndexOf(specialChar) != (word.Length - 1))
1167:                     {
1168:                         part2 = word.Substring(match.Index + 1);
1169:                     }
1170:                     // creates a sub list
1171:                     List<string> postWords = new List<string>();
1172:                     if (match.Index != (words.Count - 1))
1173:                     {
1174:                         postWords = words.GetRange(a + 1, words.Count - a - 1);
1175:                     }
1176:                     // removes all words after & including special character
1177:                     words.RemoveRange(a, words.Count - a);
1178:                     // adds the parts containing & including special character
1179:                     if (part1 is not null)
1180:                     {
1181:                         words.Add(part1);
1182:                     }
1183:                     words.Add(specialChar);
1184:                     if (part2 is not null)
1185:                     {
1186:                         words.Add(part2);
1187:                     }
1188:                     // concats both lists together
1189:                     words.AddRange(postWords);
1190:                     a += 2;
1191:                 }
1192:                 a++;
1193:             }
1194:             falseWords.RemoveAll(item => string.IsNullOrWhiteSpace(item));
1195:             for (int b = 0; b < falseWords.Count; b++)
1196:             {
1197:                 Match match = Regex.Match(falseWords[b], @"[^\w]");
1198:                 if (match.Success)
1199:                 {
1200:                     falseWords.Remove(falseWords[b]);
1201:                 }
1202:             }
1203:         }
1204:         protected void LeaveMessage()
1205:         {
1206:             RemoveAll();
1207:             Label message1 = new Label();
1208:             message1.Text = "Spell check complete";
1209:             message1.X = Pos.Center();
1210:             message1.Y = Pos.Center() - 1;
1211:             Label message2 = new Label();
1212:             message2.Text = "No errors found".PadLeft(message1.Text.ToString().Length / 2);
1213:             message2.X = message1.X;
1214:             message2.Y = message1.Y + 1;
1215:             homeBtn.Text = "Home";
1216:             homeBtn.X = message1.X;
1217:             homeBtn.Y = message2.Y + 2;
1218:             Add(message1, message2, homeBtn);
1219:         }
1220:         protected void DisplayWords()
1221:         {
1222:             // move any special characters to own index
1223:             CheckGrammar();
1224:             // find false words
1225:             if (theseSettings.spellCheck)
1226:             {
1227:                 CheckSpelling();
1228:                 CheckGrammar();
1229:             }
1230:             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)
            {
```

```
1309:                 text = text.Substring(0, theseSettings.maxLength);
1310:             }
1311:             words = text.Split(' ').ToList();
1312:         }
1313:         protected void GoHome()
1314:         {
1315:             string text = "";
1316:             foreach (var a in words)
1317:             {
1318:                 text += a + " ";
1319:             }
1320:             text = FixGrammar(text);
1321:             Application.Run(new MainView(text));
1322:         }
1323:         protected string FixGrammar(string text)
1324:         {
1325:             int speechCount = 0;
1326:             for (int a = 0; a < text.Length; a++)
1327:             {
1328:                 switch (text[a])
1329:                 {
1330:                     // need character before and space after
1331:                     case '.':
1332:                     case ',':
1333:                     case '!':
1334:                     case '?':
1335:                     case ':':
1336:                     case ';':
1337:                     case ')':
1338:                         // before
1339:                         if (a != 0)
1340:                         {
1341:                             if (char.IsWhiteSpace(text[a - 1]))
1342:                             {
1343:                                 text = text.Remove(a - 1, 1);
1344:                             }
1345:                         }
1346:                         // after
1347:                         if (a != text.Length - 1)
1348:                         {
1349:                             if (!char.IsWhiteSpace(text[a + 1]))
1350:                             {
1351:                                 string tempText = text.Substring(a + 1);
1352:                                 text = text.Substring(0, a + 1);
1353:                                 text += " ";
1354:                                 text += tempText;
1355:                             }
1356:                         }
1357:                         break;
1358:                     // need character before and after
1359:                     case '/':
1360:                         // before
1361:                         if (a != 0)
1362:                         {
1363:                             if (char.IsWhiteSpace(text[a - 1]))
1364:                             {
1365:                                 text = text.Remove(a - 1, 1);
1366:                             }
1367:                         }
1368:                         // after
1369:                         if (a != text.Length - 1)
1370:                         {
1371:                             if (char.IsWhiteSpace(text[a]))
1372:                             {
1373:                                 text = text.Remove(a, 1);
1374:                             }
1375:                         }
1376:                         break;
1377:                     // need space before and character after
1378:                     case '(':
1379:                         // before
1380:                         if (a != 0)
1381:                         {
1382:                             if (char.IsWhiteSpace(text[a - 1]))
1383:                             {
1384:                                 string tempText = text.Substring(a);
1385:                                 text = text.Substring(0, a - 1);
1386:                                 text += " ";
```

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

```
1465:         protected int currentFilter;
1466:         public Dictionary()
1467:         {
1468:             words = FetchAllWords();
1469:             firstIndex = 0;
1470:             currentFilter = filterRad.SelectedItem;
1471:             UpdateWords();
1472:             InitializeComponent();
1473:             lookupBtn.Clicked += () =>
1474:             {
1475:                 Lookup(0);
1476:             };
1477:             nextBtn.Clicked += () =>
1478:             {
1479:                 NextPage();
1480:             };
1481:             prevBtn.Clicked += () =>
1482:             {
1483:                 PrevPage();
1484:             };
1485:             defineBtn.Clicked += () =>
1486:             {
1487:                 currentLookup = lookupTxt.Text.ToString();
1488:                 if (currentLookup is not null)
1489:                 {
1490:                     DefineThis(currentLookup);
1491:                 }
1492:                 else
1493:                 {
1494:                     MessageBox.ErrorQuery("Error", "There is no word to define\nEnter a value in the lookup field", "Ok");
1495:                 }
1496:             };
1497:             addBtn.Clicked += () =>
1498:             {
1499:                 currentLookup = lookupTxt.Text.ToString();
1500:                 if (currentLookup is not null && !currentLookup.Contains(" "))
1501:                 {
1502:                     AddThis(currentLookup);
1503:                     UpdateWords();
1504:                 }
1505:                 else
1506:                 {
1507:                     MessageBox.ErrorQuery("Error", "There is no word to add to the dictionary\nEnter a value in the lookup
field", "Ok");
1508:                 }
1509:             };
1510:             removeBtn.Clicked += () =>
1511:             {
1512:                 currentLookup = lookupTxt.Text.ToString();
1513:                 if (currentLookup is not null)
1514:                 {
1515:                     RemoveThis(currentLookup);
1516:                     UpdateWords();
1517:                 }
1518:                 else
1519:                 {
1520:                     MessageBox.ErrorQuery("Error", "There is no word to remove fomr the dictionary\nEnter a value in the lookup
field", "Ok");
1521:                 }
1522:             };
1523:             filterBtn.Clicked += () =>
1524:             {
1525:                 currentFilter = filterRad.SelectedItem;
1526:                 UpdateWords();
1527:             };
1528:         }
1529:         protected string[] FetchAllWords()
1530:         {
1531:             string defPath = "Resources/TrueWords.txt";
1532:             string addedPath = "Resources/AddedWords.txt";
1533:             List<string> words = File.ReadAllLines(defPath).ToList();
1534:             List<string> addedWords = File.ReadAllLines(addedPath).ToList();
1535:             words.AddRange(addedWords);
1536:             words.Sort();
1537:             string[] values = words.ToArray();
1538:             return values;
1539:         }
1540:         protected string[] FetchWords(int num)
1541:         {
```

```
            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();
        }
    }
}
//////////////////////
```

```
1698: // Views/MasterView.cs //
1699: //////////////////////////
1700: // install dependencies
1701: // suppress warnings
1702: # pragma warning disable
1703: namespace Views
1704: {
1705:     public partial class MasterView
1706:     {
1707:         public MasterView()
1708:         {
1709:             Build();
1710:         }
1711:     }
1712: }
1713: ///////////////////////
1714: // Views/Settings.cs //
1715: ///////////////////////
1716: // install dependencies
1717: using Terminal.Gui;
1718: // suppress warnings
1719: # pragma warning disable
1720: namespace Views
1721: {
1722:     public partial class Settings
1723:     {
1724:         public bool spellCheck;
1725:         public bool changeAcronyms;
1726:         public bool grammarCheck;
1727:         public int maxLength;
1728:         public Color background;
1729:         public Color foreground;
1730:         public Settings()
1731:         {
1732:             FetchSettings();
1733:             InitializeComponent();
1734:             saveBtn.Clicked += () =>
1735:             {
1736:                 SaveSettings();
1737:             };
1738:             viewDictBtn.Clicked += () =>
1739:             {
1740:                 Application.Run(new Dictionary());
1741:             };
1742:             resetBtn.Clicked += () =>
1743:             {
1744:                 ResetSettings();
1745:             };
1746:             acronymBtn.Clicked += () =>
1747:             {
1748:                 if (shortAcrTxt.Text.ToString().Length != 0 && fullAcrTxt.Text.ToString().Length != 0)
1749:                 {
1750:                     string shortAcr = shortAcrTxt.Text.ToString();
1751:                     string fullAcr = fullAcrTxt.Text.ToString();
1752:                     AddAcronym(shortAcr, fullAcr);
1753:                     MessageBox.Query("Acronym added", shortAcr + " added as " + fullAcr, "Ok");
1754:                 }
1755:                 else
1756:                 {
1757:                     MessageBox.ErrorQuery("Error", "Enter text in both short and full acronym fields", "Ok");
1758:                 }
1759:             };
1760:         }
1761:         protected void SaveSettings()
1762:         {
1763:             spellCheck = spellCheckCbx.Checked;
1764:             changeAcronyms = changeAcronymsCbx.Checked;
1765:             grammarCheck = grammarCheckCbx.Checked;
1766:             if (!int.TryParse(maxLengthTxt.Text.ToString(), out maxLength))
1767:             {
1768:                 MessageBox.ErrorQuery("Error", "The input for max length must be a whole number\nFor no max length, enter 0",
"Ok");
1769:                 return;
1770:             }
1771:             string settings = "Spell " + spellCheck + "\n"
1772:                 + "Lengt " + maxLength + "\n"
1773:                 + "Acron " + changeAcronyms + "\n"
1774:                 + "Grama " + grammarCheck + "\n"
1775:                 + "Foreg " + foregroundPick.SelectedColor + "\n"
```

```
1776:                 + "Backg " + backgroundPick.SelectedColor + "\n"
1777:                 ;
1778:             string path = "Resources/Settings.txt";
1779:             File.WriteAllText(path, settings);
1780:             FetchSettings();
1781:             MessageBox.Query("Saved", "Settings saved successfully", "Ok");
1782:         }
1783:         public void FetchSettings()
1784:         {
1785:             string path = "Resources/Settings.txt";
1786:             string[] settings = File.ReadAllLines(path);
1787:             spellCheck = bool.Parse(settings[0].Substring(6));
1788:             maxLength = int.Parse(settings[1].Substring(6));
1789:             changeAcronyms = bool.Parse(settings[2].Substring(6));
1790:             grammarCheck = bool.Parse(settings[3].Substring(6));
1791:             foreground = GetColour(settings[4].Substring(6));
1792:             background = GetColour(settings[5].Substring(6));
1793:             Colors.Base.Normal = Application.Driver.MakeAttribute(foreground, background);
1794:         }
1795:         protected void ResetSettings()
1796:         {
1797:             spellCheck = true;
1798:             maxLength = 0;
1799:             changeAcronyms = true;
1800:             grammarCheck = false;
1801:             foreground = Color.Gray;
1802:             background = Color.Blue;
1803:             string settings = "Spell " + spellCheck + "\n"
1804:                 + "Lengt " + maxLength + "\n"
1805:                 + "Acron " + changeAcronyms + "\n"
1806:                 + "Grama " + grammarCheck + "\n"
1807:                 + "Foreg " + foreground + "\n"
1808:                 + "Backg " + background + "\n"
1809:                 ;
1810:             string path = "Resources/Settings.txt";
1811:             File.WriteAllText(path, settings);
1812:             FetchSettings();
1813:             MessageBox.Query("Reset", "Settings reset to default successfully", "Ok");
1814:         }
1815:         protected void AddAcronym(string shortAcr, string fullAcr)
1816:         {
1817:             string path = "Resources/Acronyms.txt";
1818:             string text = shortAcr + "*" + fullAcr + "\n";
1819:             File.AppendAllText(path, text);
1820:         }
1821:         protected Color GetColour(string colourStr)
1822:         {
1823:             switch (colourStr)
1824:             {
1825:                 case "Black":
1826:                     return Color.Black;
1827:                 case "Blue":
1828:                     return Color.Blue;
1829:                 case "BrightBlue":
1830:                     return Color.BrightBlue;
1831:                 case "BrightCyan":
1832:                     return Color.BrightCyan;
1833:                 case "BrightGreen":
1834:                     return Color.BrightGreen;
1835:                 case "BrightMagenta":
1836:                     return Color.BrightMagenta;
1837:                 case "BrightRed":
1838:                     return Color.BrightRed;
1839:                 case "BrightYellow":
1840:                     return Color.BrightYellow;
1841:                 case "Brown":
1842:                     return Color.Brown;
1843:                 case "Cyan":
1844:                     return Color.Cyan;
1845:                 case "DarkGray":
1846:                     return Color.DarkGray;
1847:                 case "Gray":
1848:                     return Color.Gray;
1849:                 case "Green":
1850:                     return Color.Green;
1851:                 case "Magenta":
1852:                     return Color.Magenta;
1853:                 case "Red":
```

```
1854:                     return Color.Red;
1855:                 case "White":
1856:                     return Color.White;
1857:             }
1858:             return Color.Black;
1859:         }
1860:     }
1861: }
```