Final Report

**1 Introduction [Idea]**

For my program, I wanted to create something that would help someone read German. I picked this because I studied German for a long time at school, and looking up all the words I didn’t know or forgot used to take a lot of time and would frustrate me. Therefore I wanted to create something that will save the user time when studying a text by translating words they don’t know and being able to create an accurate vocabulary for the user as they use the program more.

For my first approach to the idea, I knew I had to import the text to be studied, along with some German vocabulary the user should know. I thought about providing vocabulary sheets in a drop down menu for different levels (A1,A2 etc) or just the user uploading their own vocabulary sheet. At the start, I decided to get the user to just upload their own vocabulary because I will be starting with extremely simple text and vocabulary.

I will highlight words the user should know (words in the vocabulary) and then the user can click on the word to get the translation. If the word is not in the vocabulary, the computer should translate the word for the user.

**2 Design**

**2.1 Initial Design**

Diagram, text

Description automatically generatedText, letter

Description automatically generated

2

1

3

2

1

More features (bottom image)

1. **click feature** – so the user can click on a word and get the translation
2. **hover feature** – when you hover over a word, shows translation and have a tick/cross button to say if the user knows/doesn’t know the word

Features (top image)

1. **Upload button** – to upload the text to be studied, the German and English vocabulary
2. **mainTextBox** – to load the text after words have been highlighted to see if they are in the vocabulary
3. **translationBox** – to display the translation of the word the user clicks on

More on the hover feature(adding or removing vocabulary):

On the hover feature with the tick/cross button, this will add/keep/remove words from the vocabulary. This is how the vocabulary becomes more accurate to the user.

I thought to improve vocabulary, the program could also make ‘guesses’ of words they think the user may know. For example if the user knows the word for ‘football’, they may know words that are football-related like ball, goal etc. Also in German, a lot of the words follow patterns. For example, the verb spielen means to play, but from this word you can make other words like der Spiel (game), spielerish (playful) and Spieler (player) etc, so you could say the user should know all words with the stem spiel-. This is something I didn’t get a chance to look into in my program as it could be very complicated to program, but something I could look into in the future. There may be some sources online that talk about this relationship of words too.

**2.2 Problems with the initial idea:**

1. *How will the computer be able to translate words the user doesn’t know?*

If the user clicks on a word that isn’t in their vocabulary, there will be no translation for the word stored in the files uploaded. Ideally, I would like for my program to look up the unknown words as the aim is to save the user time looking up vocabulary. However, I decided to change this, and instead the user will have to look up a word and type it into the program and then add the word to the vocabulary.

1. *How will I create this click/hover feature?*

I also decided to change this for now as I wasn’t too sure how to create a click/hover over text in a textEdit. I changed it so that the user will have to search the word they want to know and the program will output the translation or ask for the translation (if unknown). This is something I would like to add in the future after gaining more skills with Qt.

1. *What format will I upload my files in?*

I would be using QFile and then QTextStream to get files. I will have three text files.

1. **Text:** the text the user will be studying (below is the example I used when testing my program-it’s very simple)
2. **Dictionary:** the German vocabulary
3. **Translation**: the English translation for the German vocab list. The lines for the dictionary and translation match e.g. Hallo = Hello, Danke = Thank you, Nein = No,…

1



A picture containing text

Description automatically generatedText

Description automatically generated

3

2

**2.3 Initial idea redesigned**

2

Features:

1. **Upload button** – to upload files
2. **Search\_word button** – click to search a word in the wordSearchbar
3. **wordSearchbar** – type in word to be searched
4. **Maintextbox-**where the main text is displayed
5. **translationBox** – to display translation
6. **enterTranslationbox** – where you type the translation of a word that isn’t in the dictionary
7. **Add\_new\_word\_to\_dictionary** – add button to take word from enterTranslationBox and add to dictionary
8. **englishBox +germanBox** – to display dictionary and translation

Diagram

Description automatically generated

8

7

6

5

4

3

1

Notes:

I have removed the ability to add/remove words from the vocabulary at this point as I was going to add all the other features first and come back to this.

**3 Design**

**3.1 Upload button**

Code

Graphical user interface, text

Description automatically generated with medium confidence

I created one upload button that would open a QFileDialog, where the user can choose the file they want to upload from their computer. On clicking this one button, there would be three filedialogs: one for the text, one for the dictionary and one for the translation.

You can see from the code, I have just shown one filedialog (they all have the same format, just different variable names. In the code you can see if the file isn’t opened correctly (for example an incorrect path), then an error message is shown and the user needs to upload a file again. The file is converted to a textstream.

1. Text: QTextStream for text is source
2. Dictionary (german vocabulary): QTextStream for dictionary is dictSource
3. Translation (English vocabulary): QTextStream for translation is transSource

**3.2 Comparing files**

Next I created a textEdit called mainTextBox, where the text will be displayed after comparing the words in text with the dictionary. At first I was unsure how to compare files, however I decided to get each file, split each file to words and store each word in a list.

Then I will compare each elements in a list with the elements in another list to look for common words.

Functions:

1. **Split\_file\_into\_words(QTextStream &file, QStringList &list, bool text)**

How it works:

1. QTextStream file – the file you are looking to store all of its words e.g. source, dictSource, transSource
2. QStringList list – the list where you will store all the words for the corresponding file
3. Bool text – I added this after my first cycle of programming as I ran into an error. If the file is the text, this will be true. If the file is the dictionary or translation, then this will be false. I have included this as for the text, each line will need to be split up into words, whereas a dictionary and translation will not have to split up lines into words. The error I got was for ‘Thank you’ in the dictionary. This split into ‘thank’ and ‘you’, giving me the wrong translation as it meant there was one more word in the dictionary than in the translation (messing up the indexes). This was fixed after I stopped splitting up lines for dictionary and translation.

Code:

Graphical user interface, text, application

Description automatically generated

In the code I have implemented this.

1. **Compare\_words(QStringList &text, QStringList &dictionary, QStringList &modifiedString)**

How it works:

1. QStringList text – the first list. This is the list you want to see if words from the second list (dictionary) are in the first list (text)
2. QStringList dictionary – second list. The list to see if any of these elements are in the first list
3. QStringList modifiedString – words that are in the dictionary that are in the text will be turned to capitals. Words that aren’t will be left unchanged. Every word is added to the modifiedString after it has been changed to capitals/unchanged.

Code

Graphical user interface, text, application

Description automatically generated

You can see in my code I have achieved this using two for loops and an if statement. There is also an added function remove\_punctuation, which I added later on and will explain in the next cycle of my work.

Final result



**3.3 Searching words**

Then I moved onto searching words.

1. **wordSearchBar** – where you enter the word you want to search
2. **searchButton** – click this button once you have typed a word into wordSearchBar
3. **translationBox** – where the word and its translation will be displayed once you have searched a word. If the word isn’t in the dictionary, ask the user to look the word up themselves and type the translation in themselves

Diagram

Description automatically generated

3

2

1

How it works:

Code:



A picture containing diagram

Description automatically generatedGraphical user interface, text

Description automatically generated

The code follows the instructions in the blue boxes above. The only thing I haven’t mentioned is this QStringList output.

1. QStringList output - I was unsure how to display the translation to the user, so instead I created this

Output = {‘Word: ‘, word (whatever was searched), ‘Translation: ‘, translation of word searched}

The program can find the translation as the index of the dictionary when the word is found in the dictionary, will be the same index of the corresponding translation in the English translation file. E.g. dictionary[1]=Hallo, translation[1]=Hello. Hallo = Hello.

I can understand this QStringList is probably not the best way to show the translation in terms of understanding my code, however it still displays the translation in the translationBox as I wanted. Maybe I could have made another textEdit box, so one box has the german word pointed to it, and the other box has the English translation so my code would make more sense.

Final result

Graphical user interface, text, application

Description automatically generated Graphical user interface

Description automatically generated

**3.4 Adding word if word is unknown**

(enterTranslationBox, add\_new\_word\_to\_dictionaryButton)

Now if the word is unknown, the user must find the word themselves and input it into the program

1. **enterTranslationBox** – if word is not in vocabulary, type the word here
2. **add\_new\_word\_to\_dictionaryButton** – click when want to add word to dictionary
3. **englishBox** – where translation is displayed
4. **germanBox** – where dictionary is displayed

Diagram

Description automatically generated

4

3

2

1

How it works:

Code

Graphical user interface, text

Description automatically generated

You can see from the code, it takes the word from the wordSearchBar and translated word from the enterTranslationBox and adds them both to the vocabulary. Then the new dictionary and translation lists are outputted to the englishBox and germanBox. This is how I checked the word was being added correctly.

Something I added to this was I had to change the added word to capitals after it was added to the vocabulary, as the user should know this word now. Therefore that’s why you see the compare\_words function again and then an updated modifiedString is pointed to the mainTextBox.

Final Result

Screenshot 1

Graphical user interface, application

Description automatically generated

3

2

1

Geburtstag is searched.

From (1), can see Geburtstag is not in the dictionary

Type translation in (2)

(3) can see Geburtstag is not in the German dictionary.

Screenshot 2 – after click Add, you can see Geburtstag is added in (4)

4

In (5), Geburtstag has changed to capitalsGraphical user interface, application, Word

Description automatically generated

5

**3.5 Cleaning my code**

Now all my main features are added, I went through my code and cleaned it up. This involved making the functions like compare\_words etc. At the start I just put all this code under the upload button. I also needed my program to run more smoothly.

Some things I added:

1. Created three upload buttons and a done button– one for text, one for dictionary, one for translation and a button to click when done.
2. Word turns to capitals once its added
3. Errors - make everything run more smoothly
4. Cant progress without adding all files – error message will appear if haven’t uploaded all three files
5. Cant add a word without searching a word – error message will appear if haven’t searched a word

**3.6 Adding more features**

Some other features I added were: Graphical user interface, application

Description automatically generated

2

1

1. **Save button** – save the new vocabulary list so the user can come back to the program and upload the vocabulary again

I decided it would be more efficient for the save button to create one file with the German and English vocabulary. The first function to handle this (file\_format\_for\_saving\_file) takes the dictionary and translation and writes the vocabulary to a new list called final vocablist. It writes the new vocabulary in the format Hallo=Hello, Danke=Thank you …

Code

Text

Description automatically generated

After changing the format of the vocabulary, I have to make a new function to save the vocabulary. I did this by taking this new list and a filename, creating a file with given filename from whatever the user sets it to and adding the list to the stream. If the file didn’t open correctly, then there will be an error.

Code

Graphical user interface, text, application

Description automatically generated

1. **Word goes red if its been searched** – I thought this would be useful with long text

I looked up a lot online about changing text colour and found setTextColor(QColor(colourname)).

How it works

Code I added the below code into my searchwordButton.

Text

Description automatically generated

Explained in more detail:

This works by creating a new QStringList wordlist, adding the word that has been searched to it and comparing the words from wordlist (this will just contain the one word you searched) and the text. The modifiedString is stored in new QStringList in\_text. The in\_text list will only have one element, either word unchanged, or word in capitals if it is contained in the text. If the word is in capitals, then search the text for the word and set the textcolour for that to red. Until you reach the word, set the text colour to black. This works by me rewriting to the mainTextBox.

1. **Remove\_punc** – tried more complicated text with punctuation. This caused errors.

e.g. halbtags. != halbtags. So halbtags.->halbtags using this function so now I can properly match words.

Code

Graphical user interface, text

Description automatically generated

Problems with this: I removed the punctuation so I can match it to words in the text, however I have not added the punctuation back into the text after comparing. This is something I would add.

1. **Started looking at conjugations**

In the vocabulary, there may only be the verb e.g. spielen, but not the conjugated forms for different tenses. Therefore, the program will identify this word as unknown, although the user will probably know the tenses/conjugations.

The present tense in German follows this rule:

1. Get verb
2. Take stem
3. Add appropriate ending for pronoun

e.g. spielen – to play

stem - spiel (just remove en to get stem)

endings – ich spielE, du spiels, er/sie/ihr spielT, wir/Sie spielEN. These endings are the same for the corresponding pronoun in German.

Below I made a function that takes the verb, deducts the last two letters to get the stem of the verb, and then adds the appropriate endings. These conjugated forms are added to a list. Text

Description automatically generated

I never used this function, however if I was to use this, what I would do is change how I compare text. I would add something that identifies if the word contains a verb stem.

To do this:

1. When upload german dictionary, identify the verbs (words ending with -en) and change these to verb stems (the word without -en).
2. Then when I compare text, when I reach a word in the text, I will see if it is the dictionary, if it is, then carry on. If it isn’t, then see if the word contains a verbstem e.g. spielen contains spiel, so we guess the user knows this conjugation (they probably will). If word doesn’t contain verbstem, word is not in their vocabulary.

A problem with this is there are words that end with -en that are not verbs. For example, Fahrten (travels). This word is a noun. Nouns in german always start with a capital letter. I would distinguish between words ending with -en by seeing if they begin with a capital. If they do, the word is a noun. If they don’t, the word is a verb so I can carry on with the logic above. However, this will only work if the text has no spelling mistakes and my compare\_words function isn’t case sensitive so I would have to change it slightly to take into consideration this first letter.

**4 Evaluation**

Although my program works, I would say there are still a lot of improvements. I am happy that I have improved my C++ skills as I feel like at the start, I didn’t even know how to display things in textEdits so I have come a long way. I think my program is on its way to looking more ‘smooth’, for example when the text goes to red when it is searched. Below I have discussed improvements I would make to my program and what my next step is.

1. More complicated text

I would say my program works for extremely simple text, so something to move onto would be adding things like grammar rules, conjugations as I discussed earlier, tenses, and everything that makes the German language. At the moment my program just identifies words that are in the user’s vocabulary like nouns. If I add the other features, then it will be able to tell if the user knows tense conjugations and grammar rules. This will make the program more useful and work better for longer, more complicated text. I should also run my program with more complicated text and just see what happens.

1. Add/remove feature

Something I left out of my program was the add/remove feature. This was supposed to add/remove vocabulary after you have searched it, to get the vocabulary more tailored and specific to the user. Maybe a word is in the vocabulary that the user doesn’t know, therefore they should be able to delete it. I don’t think this would be hard to add, I would add a remove button. The user can already add words, so the remove button will be similar, except the word will be deleted from the vocabulary after it has been searched.

1. Some other errors/cleaning up
2. When a word is searched and is turned to red in the text, the word changes to how you spelt the word in the text bar.

e.g. let’s say the mainTextBox contains:

JA DANKE GEBURTSTAG Morgen

We search: GebUrtstag

mainTextBox: JA DANKE GebUrtstag Morgen

Instead of setting the word to the word in the searchbar, it should find the word in the text and then change its colour.

1. You can add words to the dictionary after you have searched any word

Graphical user interface, application

Description automatically generated

I created the condition that you must search a word in order to add a word to the dictionary. However I need to change this condition. The problem with my code is that you only need to search a word ONCE to be able to add to the dictionary. You can see on the left, after I have added the word dankeschon to the dictionary, I have added two other words.

I need to go over my code and double check my condition. It needs to be set to false(reset) after the add button has been clicked.

1. Textedit should be readonly

In the program, you can technically write to the textedits. This should be unavailable.

1. Change the format of how I upload German dictionary and English translation. How it is saved at the end is in a different format to how the words are uploaded. Change how they are uploaded. I think this should be okay. I would create one button instead of two and upload the English and german in one file. A line would be Hallo=Hello, so I would go through the file and once I find a ‘=’, split the line and store into different lists.
2. Other features to add:
3. Add vocabulary lists depending on what level the user is. Put in a drop down menu
4. Look into QSyntaxHighlighter more to highlight text properly, instead of turning words to capitals
5. Look into Regex that you said after my presentation to improve language features

Overall, I really liked writing this program and I am glad to have improved my skills. There are a lot of things to add but my immediate next step would be to clean up my code and then look into more complicated German text so my program can become more useful.