Software Engineering Group Project
Maintenance Manual

Author: Shane Waters
Config Ref: Maint-Man_GP15
Date: 1st May 2020
Version: 1.0
Status: Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

This document describes the procedures for future maintenance for the Welsh vocabulary tutor application. It should inform the maintainer how to go about fixing/maintaining the program if bugs or errors occur. As bugs are impossible to avoid providing a maintenance manual is an essential document to complete when producing a program.

1.2 Scope

This document contains descriptions of the following sections related to the application: structure, algorithms, data areas, required files, interfaces and previously known bugs. This document will contain all information future developers and maintainers will need in order to perform maintenance.

1.3 Objectives

The objective for this document is to describe the software so that a programmer familiar with JavaFX can maintain it properly. It will contain solutions to bugs/problems that might occur in future development. This document will enable a maintainer to successfully diagnose and produce a solution to program wide errors or bugs.

This document will give future planning, ensuring that maintaining the application is as simplified as possible. Listing previously known issues will enable maintainers and developers to work through bugs and errors.

2. BODY

2.1 Program description

The Welsh Vocabulary Tutor application was made to help English speakers learn the Welsh language. It uses Tree Maps to store words in a dictionary. The program will provide a dictionary generated from a JSON file on start-up, the dictionary can be sorted to select either English or Welsh as the primary language. A search function enables the user to navigate to a desired word, by inputting the prefix for the word, the program will constrict the results to exclusively show words with the inputted prefix. The search function works for both languages, searching by the primary language.

The program contains a practice list, the user can select a word they wish to learn which resultantly gets transferred to their practice list. Words can be appended or subtracted from the practice list at any point. The practice list works in the same fashion the dictionary does, including the search function. From the practice list, randomly selected Welsh words can be generated into flashcards, when a flashcard is selected it will produce the English translation.

The program includes a testing page. This provides 3 tests of varying types, which will generate a random question of the test type that was selected. After the question has been answered feedback will be provided to the user, displaying how they performed. There is also an option to generate a full test, this will create a series of tests which will perform in the same fashion as the individual test.

2.2 Program structure

The Welsh vocabulary tutor application's program structure can be described with help of design diagrams and a list of modules and methods located in the Design Specification.

A component Diagram of the program can be found in the Design Specification section 3.1, figure 1 [2].

A Use Case Diagram of the program can be found in the Design Specification section 4.1, figure 2 [2].

Some sections of the program are too complicated to be understood from just observation so by using decomposition each complex module can be broken down to its lowest level of classes and calls. This is done using Sequence diagrams.

The Sequence diagrams of the program can be found in the Design Specification section 5.1.1 – 5.1.4, figures 3.1 – 3.4 [2].

Some complex modules within the program are better explained using a state chart rather than a sequence diagram. State charts also provide even more details on what is happening for modules which already have a sequence diagram.

The State charts of the program can be found in the Design Specification section 5.3, figures 3.5, 3.6 and 3.7 [2].

A Class diagram of the program can be found in Design Specification section 5.5.1, figure 4 [2].

An Object diagram of the program, cam ne found in the Design Specification section 5.5.2, figure 5 [2].

The Welsh vocabulary tutor application's classes and their respective methods can be found in the Design Specification in section 4.2 [2].

2.3 Algorithms

The most important decision regarding algorithms in the Welsh Vocabulary Tutor application was choice how the words are going to be stored in the program. Main choices were Sorted Array List and Tree Map. Tree Map proved to be better solution for the dictionary application because adding to tree map is faster as it doesn't have to be fully sorted each time, Tree Maps are faster to search through and it is easier to implement partial searching within them.

More detailed information about significant algorithms used in the Welsh Vocabulary Tutor app are in the Design Specification in section 5.4 [2].

2.4 The main data areas

When the Welsh Vocabulary Tutor app is launched, the program will serialize the data from a json file and put it inside two Tree Maps as Word objects. These Tree Maps are located in the Dictionary class and are named *english_words* and *welsh_words*. The Dictionary class is described in the Design Specification in section 5.5.1 [2].

2.5 Files

The Welsh Vocabulary Tutor app utilises a file called "dictionary.json" located in the root folder. This file is used to store the all the words which is used by the dictionary and what words are part of the users practice list. The program assumes it will contain json format of a list of objects with 4 fields: "english", "welsh", "wordtype" and "marked" in this order. If a field is missing it will have a default value inserted instead as "marked" may be missing when the program is first launched. This file is loaded upon start up, if the file cannot be found then the program will still function, and a new file will be created when saving the dictionary. It is not required to have this file upon starting the program for the first time. If this file is corrupt, it will be replaced with a new functional version upon saving the dictionary. This file is written to every time the program closes.

2.6 Interfaces

An interface in the instance of this application is what is utilised to enable running the program. A total of 4 components are needed to successfully use the program. A computer running a version of Windows as this program has been developed for the use of a Windows system. The application requires a display, this can be a monitor, TV or anything display that can display the Windows environment. The display can be of any size and resolution as the program has been produced keeping different resolutions in mind. The minimum resolution being 640x480 and no maximum due to the program being able to adapt to any resolution. A mouse and English keyboard are required for the user to input information into the program.

2.7 Suggestions for improvements

Version 1.0 of the program contained everything which was needed to do to meet the functional requirements. There are some extra additions and some changes which could be made to make the program more efficient:

- When deserializing the JSON file it is first put into a list then into a TreeMap. It would be more efficient to have it go straight into a TreeMap but GSON does not provide an easy solution for this. This would slightly improve load times but not by much.
- The testing section of the UI currently uses a lot of fixed values to make components appear and work in the correct way. It would be more suitable to have more dynamic values as it currently has some hard-coded sections. (e.g: when generating tests and removing them it removes specific indexes rather than finding the index meaning it is a bit harder to add new tests if they require new logic.)
- A clear all button could be added to the practice list encase the user wants to delete their whole practice list and restart. The only way they can currently do this is to delete them one by one.
- A search filter could be implemented encase a user wants to be more specific with what they are trying to find. They may want to filter by wordtype rather than having to search for a specific word.
- Adding sound affects so the program is more appealing to the user.
- The program only supports one user but could be improved to support multiple users. This would require the saving of the practice list to be changed and a screen on start up for users to differentiate themselves.
- An analytics page could be created so users can see a history of the results they have obtained. This will help users understand where they have struggled as it could provide feedback on certain words which they find difficult. This would require a lot more external saving which would take a long time to implement.
- A reset button to repeat a test could be implemented, currently to reset the test the user is required to re-click the test tab. This is not completely intuitive and may not initially obvious to a typical user.

2.8 Things to watch for when making changes

The program has been designed and created in a way were most classes (in separate packages) are independent from one another so changes can be made easily. Modifying data types within the Dictionary class will cause problems throughout the program as entrysets are used to iterate through the current data type so it would be advisable to use a similar map datatype (searching is also dependant on entrysets and maps). The UI is also flexible and can be modified without causing issues elsewhere, however the tests should be modified / added carefully as a lot of fixed values are used due to some presentation issues. The testbuilder class also utilises some hardcoded values to determine whether the user has enough words to generate the test as it is not worked out on runtime.

2.9 Physical limitations of the program

Physical limitations will detail the minimum requirements for the program to run. The program has been designed for a Windows environment but should also work on any platform that runs a Java VM, this has not been tested but in theory will work. As Windows is more demanding than this program: if a processor can run Windows it will be able to run this program. The program requires ~100mb of storage space, this is due to including JavaFX which totals ~80mb.

There is a calculated estimated ~8gb for 1.1 million words. In the entire English language, there is an estimate of 1.05 million words, although a fluent/native English or Welsh speaker knows ~30 thousand words of their selected language. Therefore, we can estimate 0.22gb of memory is needed for a user to have the ability to achieve a fluent level of knowledge in their chosen language. This gives a total of 0.44gb for 30 thousand words in both languages.

As explained in the interfaces section (2.6) to access and use the program a computer, keyboard, mouse and display is needed. This display has no physical limitations, the computer must be able to connect and produce and image on the display. There is no specific resolution required, the program can successfully scale from the minimum of 640x480 and has no maximum.

2.10 Rebuilding and Testing

From the root file within the gitlab (https://gitlab.dcs.aber.ac.uk/sec26/gp15) the entire source code of the program can be found within “src/Welsh_Vocabulary_Tutor_GP15”. Inside here are the follow folders and files:

- Lib/: Contains all the libraries included within the program.
    - Javafx-sdk-11.0.2/: All the jar files required for the JavaFX library.
    - Gson-2.8.6.jar/: The jar file required for the GSON library.
- Out/: Compiled code
- Src/: Contains all the source files within their respective packages for the program.
    - Module-info.java:/ Configuration class which allows integration of JavaFX without the need of VM launch options. Allows JavaFX classes to see local packages.
    - Uk/ac/aber/cs221/gp15/: All the packages the program requires.
        - Dictionary/: Contains all the classes which handle the dictionary part of the program.
        - Json/: Contains the class required to parse the JSON file into the dictionary. Utilises GSON.
        - Tests/: Contains all the classes required to make tests which the user can do to test their knowledge on their practice list.
        - Ui/: Contains all the classes required to display the user interface
        - BaseApplication.java: The class which contains the main method, is the first class to be ran when launcher the program.
- Dictionary.json: The dictionary file which saves all the words within the program.
- Icon.png: The icon for the program.

To rebuild the program, you will need to load the whole project file “Welsh_Vocabulary_Tutor_GP15” into a IDE such as IntelliJ IDEA. Once loaded you will have to choose a java JDK version 11+. The libraries used within the program should automatically be added but if they are not, you can navigate to project structure libraries and mount the JavaFX folder and the GSON jar file as new libraries. The program can then be compiled by running the main method within BaseApplication.java.

If any kotlin errors appear the project must be fully rebuilt. No VM compile options are required as the module-info.java file allows JavaFX to communicate with packages. If module-info.java was to fail, then these VM options can be used:

“--module-path "<file path to JavaFX lib folder>" --add-modules javafx.controls,javafx.fxml”

The program can be tested by running the Junit tests located within the gitlabs “src/junit” folder. These Junit tests will require the Junit library which can be mounted from a local Junit folder or by using Maven built into IntelliJ IDEA. Junit will give instant feedback upon launching them if the program has failed any tests. New tests can be added by creating new Junit files following Junit syntax to write tests.

3. KNOWN BUGS

- **Inconsistent Duplicates**
    - Due to nature of TreeMaps, if two words sorted in English share the same English they will also share the same key within the Map. As TreeMaps will not allow duplicate keys, the second copy of the word will replace the first. If the words however have different Welsh words, then they will both appear when sorting in Welsh as they will have different Welsh keys for the map. This means that a certain word may appear once in English but twice in Welsh. Although this is not technically wrong, it is a slightly inconsistent.

REFERENCES

[1] Software Engineering Group Projects – Producing a final report / 2.2 (Release) - C. J. Price

[2] Software Engineering Group Project – Design Specification / 1.0 (Release) – Shane Waters

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
0.1 Draft	N/A	27/04/2020	Template.	JAT69
0.2 Draft	N/A	28/04/2020	Filled in some of the body's contents.	JAT69
0.3 Draft	N/A	28/04/2020	Filled in Algorithms and the main data areas sections.	KRB21
0.4 Draft	N/A	28/04/2020	Added an alternative solution for the program description section..	KRB21
0.5 Draft	N/A	28/04/2020	Altered wording in section 1 and created a program description in 2.1.	MAL102
0.55 Draft	N/A	29/04/2020	Fixed some typos, wrote better references and filled in section 2.3.	JAT69
0.56 Draft	N/A	29/04/2020	Filled in most of the references needed in the program structure section.	KRB21
0.6 Draft	N/A	29/04/2020	Filled in sections 2.7 and 2.8	SHW30
0.61 Draft	N/A	30/04/2020	Filled in section 2.6 and joined version 0.56 and 0.6 of the document.	JAT69
0.7 Draft	N/A	01/05/2020	Added Interfaces and Physical limitations section.	MAL102
0.8 Draft	N/A	01/05/2020	Connected versions 0.61 and 0.7, filled additional suggestion of improvement.	JAT69
0.9 Draft	N/A	01/05/2020	Altered wording.	MAL102
1.0 Release	N/A	01/05/2020	Finalisation of document	SHW30, IEB7, MAL102