

MINI DICTIONARY

Data Structures – CS163

23APCS2 | Group 3



Last Modified Day:

August 22, 2024

Table of Contents

[A. Abstract 3](#_Toc175892945)

[B. Introduction 3](#_Toc175892946)

[1. Overview: 3](#_Toc175892947)

[2. Key features: 3](#_Toc175892948)

[C. Group Information 4](#_Toc175892949)

[Group ID: 3 4](#_Toc175892950)

[Member: 4](#_Toc175892951)

[1. Le Vinh Thuan (Team Leader) – 23125019 (ANY%) 4](#_Toc175892952)

[2. Tu Cong Thanh – 23125018 (ANY%) 5](#_Toc175892953)

[3. Le Thi Tuyet Tram – 23125093 (ANY%) 5](#_Toc175892954)

[4. Nguyen Thu Uyen – 23125048 (ANY%) 6](#_Toc175892955)

[5. Ngo Hoang Bao Thach – 17125001 (ANY%) 6](#_Toc175892956)

[D. Data Storage 7](#_Toc175892957)

[1. Data Storage Overview 7](#_Toc175892958)

[2. Data Loading and Management Approach 7](#_Toc175892959)

[3. Rationalize 8](#_Toc175892960)

[E. Project Architecture 9](#_Toc175892961)

[1. Overview 9](#_Toc175892962)

[Project Structure 10](#_Toc175892963)

[2. Main data structure 10](#_Toc175892964)

[a. Ternary Search Tree Implementation 10](#_Toc175892965)

[b. Suffix Array Implementation 11](#_Toc175892966)

[c. Red-black Tree Implementation 13](#_Toc175892967)

[F. Implementation Detail 14](#_Toc175892968)

[Time complexity: 14](#_Toc175892969)

[1. Base-Frame.h: 15](#_Toc175892970)

[2. Home-page.h: 16](#_Toc175892971)

[3. Search-page.h: 16](#_Toc175892972)

[4. Add-page.h: 18](#_Toc175892973)

[5. Edit-page.h: 19](#_Toc175892974)

[6. game-setting-page.h: 20](#_Toc175892975)

[7. Gameplay-page.h: 21](#_Toc175892976)

[8. list-window.h: 22](#_Toc175892977)

[9. More-page.h: 23](#_Toc175892978)

[G. Technical Problems and Solutions: 24](#_Toc175892979)

[Back-end: 24](#_Toc175892980)

[Data Consistency Issues 24](#_Toc175892981)

[Search with definition 24](#_Toc175892982)

[Front-end: 24](#_Toc175892983)

[H. Feature Demonstration: 25](#_Toc175892984)

[Home Page 26](#_Toc175892985)

[History/Favorite Page 27](#_Toc175892986)

[Dictionary/Search Page 27](#_Toc175892987)

[Game Page 28](#_Toc175892988)

[More Page 29](#_Toc175892989)

[Game: Guess the word 30](#_Toc175892990)

[Add Word Page 30](#_Toc175892991)

[Edit Word Page 31](#_Toc175892992)

[I. Conclusion 32](#_Toc175892993)

[References 32](#_Toc175892994)

# Abstract

In this project, we focus on implementing a dictionary application using the programming language C++, Ternary Tree and Binary Tree data structure, Cross-Platform GUI Library wxWidgets and CMake. It is thanks to the help of CS163 course’s lecture notes, Dr. Dobb’s article about Ternary Search Tree, and Le Minh Hoang’s Suffix Tree article.

The main objective of this project is to use Ternary Tree to develop a user-friendly and responsive mini dictionary application. The primary aim is to demonstrate the use of data structure in making efficient word searches, additions, edits, and deletions, help users look up the meanings of words and allowing users to switch between various types of dictionaries, such as English-to-English, English-to-Vietnamese, and Vietnamese-to-English.

The report outlines the project’s objectives, key features, and the algorithms employed to optimize performance. We have developed and implemented all the functionality required, albeit not able to fully preprocess data and clean code.

# Introduction

## Overview:

The mini dictionary group project is a practical application of the data structures and algorithms learned in the CS163 course. We are required to manage tasks in Trello and source code in Git. Only array and vector are allowed.

Dictionaries are indispensable tools in language learning and communication, and their digital versions offer enhanced accessibility. The project’s motivation stems from the need to create an efficient dictionary application capable of handling large datasets, such as large number of words, slang terms, and emojis.

By implementing this project, the group aims to demonstrate the effective and suitable use of data structures in managing and manipulating large datasets.

## Key features:

* Switch between data sets, including:
  + English-to-English
  + English-to-Vietnamese
  + Vietnamese-to-English
  + Emoticon
  + English slang
* Search:
  + Using keywords
  + Using definitions
* View Search History
* View, add or remove words from Favorite list
* Modifying dictionary entries:
  + Add/Edit/Remove words
  + Reset dictionary
* Additional features:
  + View a random word and its definition.
  + Play Guess the meaning game.
  + Play Guess the word game.

# Group Information

Include the group ID and a list of group members along with their names and roles in the project. It's also important to mention the tasks assigned to each member and their individual contribution percentages.

## Group ID: 3

## Member:

### Le Vinh Thuan (Team Leader) – 23125019 (ANY%)

* Planning meetings and manage coding progress
* Build Ternary Tree structure
* Gathering Data Set
* UI implementation:
  + Home Page
  + Gameplay Page
* Feature:
  + Search using keywords.
  + Display words of the day.
  + Provide a word with four definitions, and users guess its meaning.
  + Provide a random definition with four keywords, and users choose the correct word.

### Tu Cong Thanh – 23125018 (ANY%)

* Build Ternary Tree structure
* Build Suffix Array structure
* Build Red-Black Tree structure
* Fine-tuning and wrapper
* Gathering Data Set
* UI implementation:
  + Base frame
  + Search Page
  + More Page
  + Game Page
  + Dark Mode
* Feature:
  + Switching Data Sets
  + Game algorithms
  + Search with definition
  + Reset dictionary to its original state.

### Le Thi Tuyet Tram – 23125093 (ANY%)

* UI Design:
  + Light Mode
* UI implementation:
  + Game Page
  + Edit Page
  + Add Page
  + More Page
* Feature:
  + Users can add a new word and its definition.
  + Users can edit the definition of an existing word.
  + Users can remove a word from the dictionary.

### Nguyen Thu Uyen – 23125048 (ANY%)

* Documentation:
  + Report
  + README.md
* UI Design:
  + Light Mode (Home Page, Search Page, Dialogs, Add/Edit, Game Page, History and Favorite Page)
  + Dark Mode
* UI Implementation:
  + History Page
  + Favorite Page
  + Search Page
  + Dark Mode
* Features:
  + View the history of search words again.
  + Users can view a random word and its definition.
  + Users can view their favorite list.
  + Users can remove/add a word from their favorite list.

### Ngo Hoang Bao Thach – 17125001 (ANY%)

* Documentation:
  + Report
* Gather Slang dataset
* Features:
  + Refactor get random words
* UI Implementation:
  + Dark mode: (Main frame)
  + Dark mode: Search page
  + Dark mode: Game setting page
  + Dark mode: Game play page

# Data Storage

Discuss how data is stored in the project. Explain the rationale behind the chosen data storage approach.

## Data Storage Overview

We organize data in a structured format to efficiently manage and update dictionary entries. The data is stored in a hierarchical directory structure within a main folder named "Data". This folder contains:

* **History**: history.txt – Stores words that users have searched.
* **Favorites**: favorite.txt – Contains words that users have marked as favorites.
* **Datasets**: Five subfolders corresponding to different datasets: EngtoEng, EngtoViet, ViettoEng, Emotion, and Slang words.

Each dataset subfolder includes:

* **Data.txt**: Contains the initial set of words and their definitions.
* **Insert.txt**: Stores new words added by users.
* **Delete.txt**: Contains words that users have removed from the dataset.

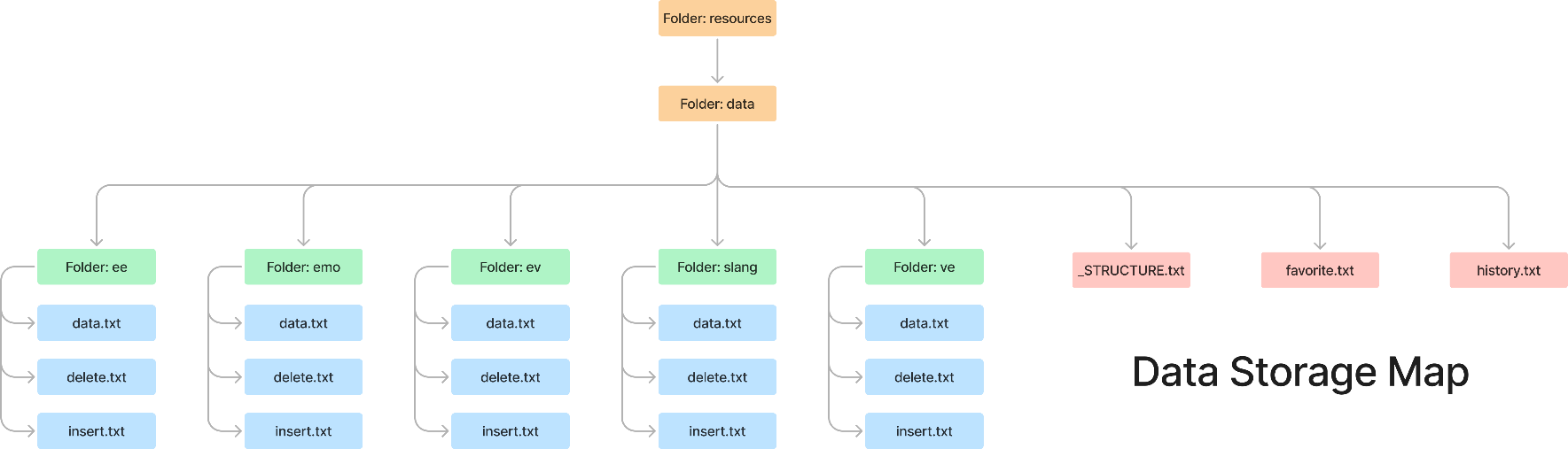


Figure 1. Data Storage Map

## Data Loading and Management Approach

When a dataset is loaded, the application follows a systematic approach to build and maintain the data structures:

* 1. **Loading Initial Data**: The application first reads data.txt to build a primary data structure, a Ternary Search Tree, that contains all the existing words and their definitions. This forms the foundation of the dataset.
  2. **Incorporating User-Added Words**: Next, the application processes insert.txt to integrate any new words added by users into the tree. This allows the dictionary to be dynamically updated with user contributions.
  3. **Removing Words**: Finally, the application reads delete.txt to remove specific words from the data structure. This ensures that the dataset remains current and reflects user deletions.

## Rationalize

**Separation:** By segregating data into different files (data.txt, insert.txt, delete.txt), the approach isolates different types of operations (initial data load, additions, deletions, and reset). This separation simplifies the process of updating the dictionary and enhances maintainability.

**Efficient Updates:** This method allows for efficient updates to the data structure. Loading the entire dataset from data.txt initially and then applying incremental changes from insert.txt and delete.txt avoids the need to rebuild the data structure from scratch every time a user makes a change. This will also allow users to reset the dictionary with ease due to the separative nature of this approach.

**User Interaction Handling:** The separate history.txt and favorite.txt allows the application to track user behavior and preferences without affecting the core dictionary data and simplify the process of data mining.

**Scalability:** The approach supports scalability by managing updates through discrete files. This modular method makes it easier to handle large datasets and multiple user interactions without compromising performance.

**Data Integrity:** Using separate files for different operations ensures that the core dataset remains consistent and correct. This approach helps prevent errors that could arise from mixing data insertion and deletion operations.

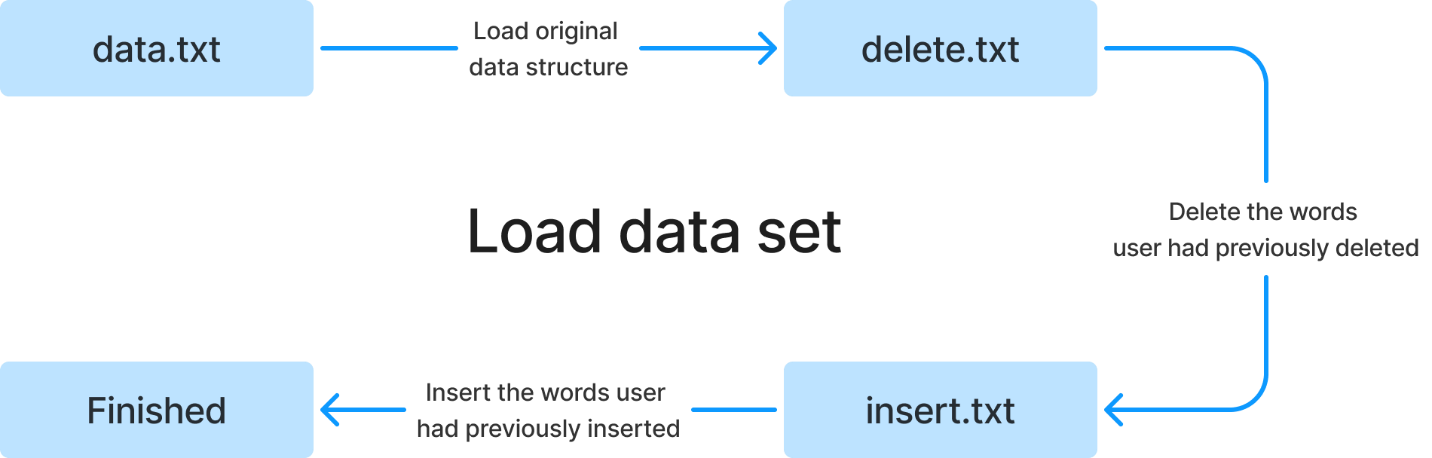


Figure 2. Initial Data Set Loading

# Project Architecture

Describe the overall architecture of the project, including the meaning and structure of project folders. Explain the main data structures employed in the project and their alternatives, highlighting why certain choices were made.

## Overview

We organize our project into several folders, using CMake to handle all of them and build the project:

* The deps folder to store necessary information for CMake to create all dependent library.
* The out folder store building artifacts.
* A diagram of a company

  Description automatically generated with medium confidenceThe Resources include all resources for our project.

Figure 3. Resources hierarchy

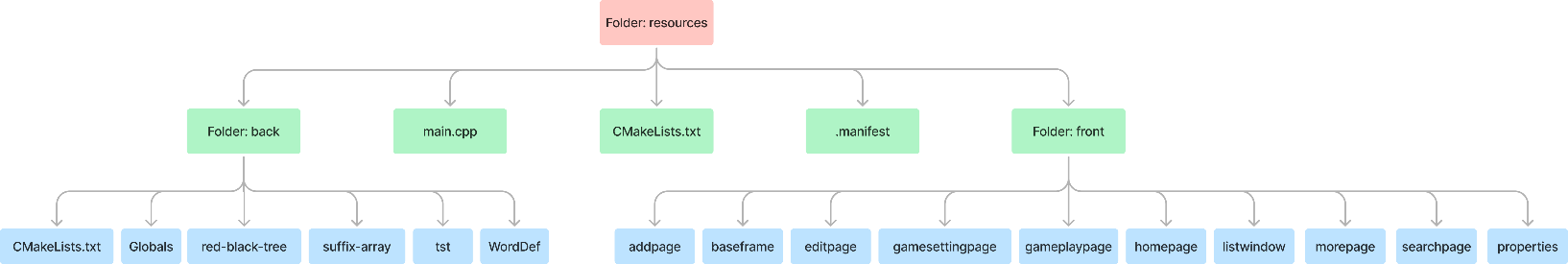
* The Sources include all of our project’s source code.

Figure 4. Sources hierarchy

## Project Structure

## Main data structure

The project utilizes two primary data structures to handle different aspects of dictionary search functionality: the Ternary Search Tree (TST) and the Suffix Tree. Each of these structures is chosen for its specific advantages in efficiently managing and retrieving dictionary data, catering to different search requirements.

### Ternary Search Tree Implementation

A Trie is a data structure used to store strings, where each node primarily contains an array pointing to its children, representing the prefix path from the root to that node. This array can be thought of as a direct-access map. However, if the number of elements in the universe of the map (or in other words, the number of characters in the alphabet) becomes too large, the tree can become inefficient due to the infrequent usage of many characters. Therefore, a common approach is to refactor the direct-access map into a different mapping structure, such as a Binary Search Tree (BST), as seen in C++'s std::ordered\_map. This leads to us choosing Ternary Search Tree (TST), which balances the structure and efficiency.

The Ternary Search Tree is employed as the main data structure for storing and searching words along with their definitions. A TST is a hybrid between a binary search tree and a trie, making it particularly well-suited for managing dictionary data where prefix-based searches are common.

struct TreeNode

{

char32\_t val = 0;

TreeNode\* left = 0, \* mid = 0, \* right = 0;

bool eow = false; //End of word

std::string defi; //Definition

};

Figure 4. Instances of TreeNode

A screenshot of a computer

Description automatically generatedEach node in the TST includes val (a character), eow (to check whether this is the end of a word), defi (if eow = True then defi will store the definition of that word), and 3 pointers to left, mid, and right. This would help us in understanding and implementing basic functions such as insertion, deletion, search and randomization.

Figure 5. An example visualization of a TST

### Suffix Array Implementation

A suffix tree is a powerful data structure for string manipulation. Its main function is to represent all the suffixes of a string in the form of a compressed trie. Although there are many important and strong algorithms related to suffix tree, its critical weakness is that it consumes a vast amount of memory, and the linear algorithms used to build suffix trees are highly complex and require very delicate observations. To compensate for this drawback, the suffix array was introduced as a reasonable alternative to the suffix tree. It can store the order of all suffixes in lexicographical order as the starting positions of the suffixes in the string. In terms of functionality, the suffix array does not support as many operations as the suffix tree, but it can be implemented quite easily. Moreover, there are several algorithms that can build a suffix array in linear time without first constructing the suffix tree. Therefore, the suffix array is considered a secondary data structure for storing information used in ***search by definition***.

**Practical Implementation:**

* We will store all the text definitions consecutively, separated by a special character (e.g., “@”). For each index corresponding to the “@” character, we map it to a string representing the definition for the substring before the separator “@”.
* After loading all the strings to build the suffix array, we append a special character ‘\0’ at the end.

**Key Algorithm:**

* **Search:** Given that we have a suffix array to represent strings in lexicographical order, we can use binary search:
  1. Initialize two variables, start = 0 and end = length of string – 1, and set k as the number of characters of the string to find in the suffix array.
  2. If the first k characters of the suffix at the start index in the suffix array are greater than the string to find, then all suffixes following the suffix at the start position are also greater, so the search can be terminated. Similarly, if the first k characters of the suffix at the end index are less than the string to find, the search can be terminated.
  3. Compute mid = (start + end)/ 2 and compare the first k characters of the suffix at the mid position in the suffix array with the string to find in lexicographical order.
  4. If the result is greater, search in the range (start, mid – 1).
  5. If the result is smaller, search in the range (mid + 1, end).
  6. If equal, add the suffix array value at mid to the dictionary, then search both (start, mid – 1) and (mid + 1, end).

**Complexity:** In the worst case, the algorithm must scan through all the characters of the text, resulting in O(n) complexity.

**Note:** The values obtained are the positions of suffixes starting with the string to find in the entire text. To find the word, locate the “@” immediately following the position and use the map to get the result.

**Construction:** This is the most crucial algorithm to build the suffix array. We use the prefix doubling algorithm by Manber & Myers, with a worst-case complexity of O(nlogn) and an average complexity of O(n).

### Red-black Tree Implementation

In this program, we use two additional data structures**: ordered\_map** and **set**. Both data structures internally utilize a red-black tree to store information.

The applications of these two auxiliary data structures in the program are as follows:

* **ordered\_map** is used to store words that have been edited and added and mapping indexes to the corresponding words in the suffix array.
* **set** serves as a buffer to store result words, preventing duplicates andstore words that have been deleted from the original dataset.

# Implementation Detail

Provide a list of the structures/classes used in the project and their relevant functions. Choose several main flows or use cases and explain how these structures and methods collaborate with each other to achieve the desired functionality.

## Explain the Algorithm:

Write a technical report explaining your design and algorithms when loading a dataset, searching, adding, updating, or deleting a word.

Loading a dataset:

Searching:

Adding:

Updating:

Deleting:

## Time complexity:

**TST:**

* Let S be our alphabet.
* Estimated tree depth: O(k×∣S∣), where k is the number of characters in the longest string inserted (it can be optimized, up to O(k×log∣S∣) if the tree is balanced).
* Operations like delete, insert, and search all have complexities approximately equivalent to the tree depth (can be optimized if the tree is balanced).

-- For random inputs: the time to "list out" the alphabet times the depth = O(k×∣S∣)

-- Inorder traversal: The worst-case scenario is the total number of nodes in the tree = O(total number of characters in all words)

**Suffix array:**

* Let n be the number of characters in the text that needs to be built:
* -- Building the tree: Worst case O(nlogn), average case O(n) (for the implemented algorithm in the project). -- Searching: O(n×k) worst case, average case O(logn×k).

We use wxWidgets as our library to build the GUI. Each file contains a class which is also a component that is inherited from the wxWindow.

## Base-Frame.h:

Figure 6. Base-frame notable features

* The very first class of our project contains all 5 main pages including the Home page, Search page, Game page, History page, Favorite page, and Detail page.
* This class is responsible for navigating users to those above pages when the users need it.

There are several buttons to navigate users to other pages, such as homeBtn, dictionaryBtn, gameBtn, historyBtn, favoriteBtn, and moreBtn. And includes some methods to catch the event when user click on those buttons.

## Home-page.h:

Figure 7. Home-page notable features

* This class contains our home page contents. It also includes some more buttons to navigate to other pages.
* This home page also displays a random word when the user opens the app.

## Search-page.h:

This is where we solve the request of user when they want to know the meaning of a word.

A screenshot of a computer

Description automatically generated

Figure 8. Respage notable features

A screenshot of a computer

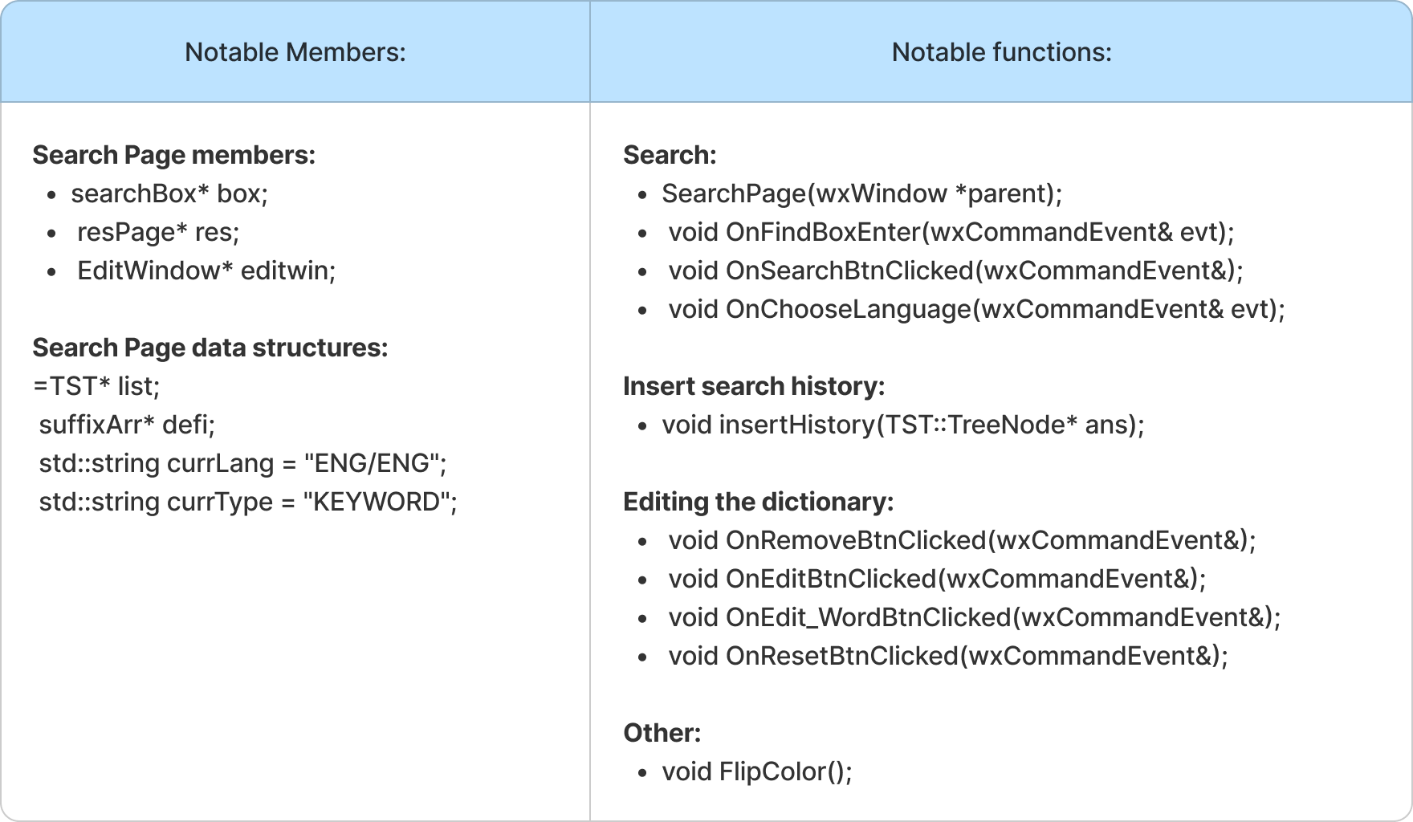
Description automatically generated

Figure 9. Search Page notable features

Figure 10. Search Box notable features

* This is where we spend our effort most. Users can switch between datasets, and search by definition or search by word. Users also can edit a word’s definition, add a word to their favorite list, and delete the word.
* In addition, they can click on the random button to get a random word, add a new word by clicking the button next to the random button, and can reset the dictionary to its initial state by the rightmost button.

## Add-page.h:

This is where users can create a new word and add it into the dictionary, by easily entering the word, type, and definition.

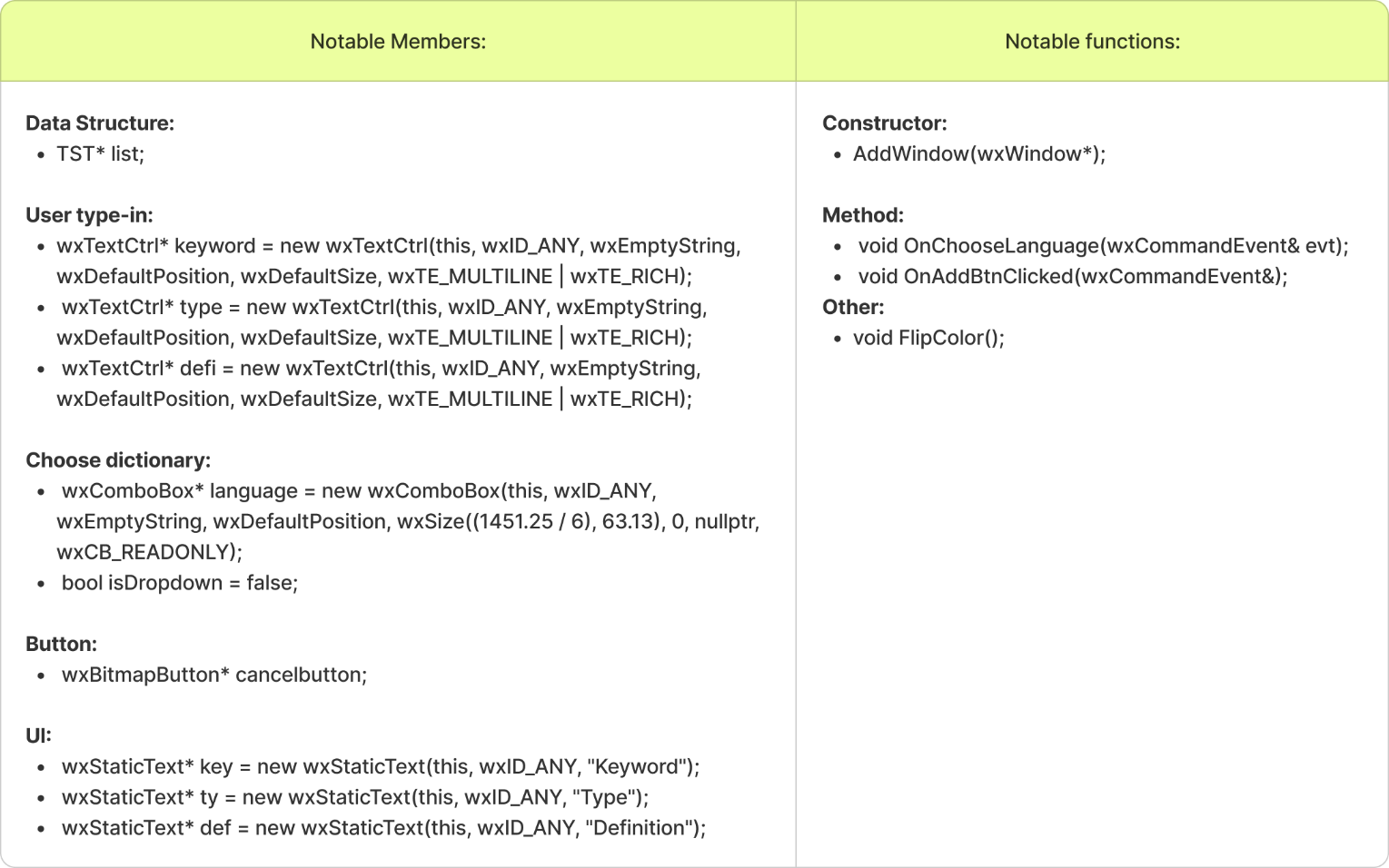


Figure 11. Add-page notable features

Functions and members related to this class:

* void OnChooseLanguage(wxCommandEvent& evt);

This function just simply helps the users to choose which dataset they would like to add a new word to.

* void OnAddBtnClicked(wxCommandEvent&);

After users add enough information about the new word, they can click the button “Add word” on UI design for add-page.

When the button is clicked, this function takes the information related to the new word such as keyword, type and definition to insert into the respective Ternary Search Tree. Then the new word is also added to the respective ordered\_map which will be written into file when users stop this app.

## Edit-page.h:

When users click the edit button, another window will appear to help edit the meaning of the searched word, after finishing, users can click the green button on the right to save.

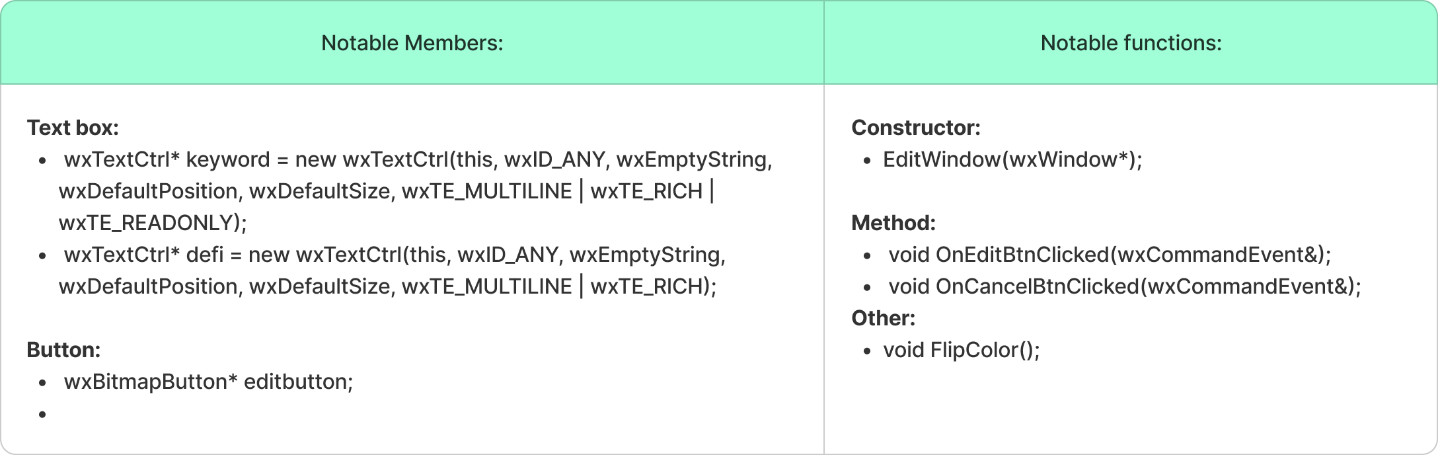


Figure 12. Edit-page notable features

Functions and members related to this class:

* void OnEditBtnClicked(wxCommandEvent&);

After users edit the definition of the word, they can click the “Edit word” button on UI design for edit-page.

When the button is clicked, this function takes the new definition and makes a change on the definition of the searched word on Ternary Search Tree. Then the edited word is also added to the respective ordered\_map which will be written into file when users stop this app.

## game-setting-page.h:

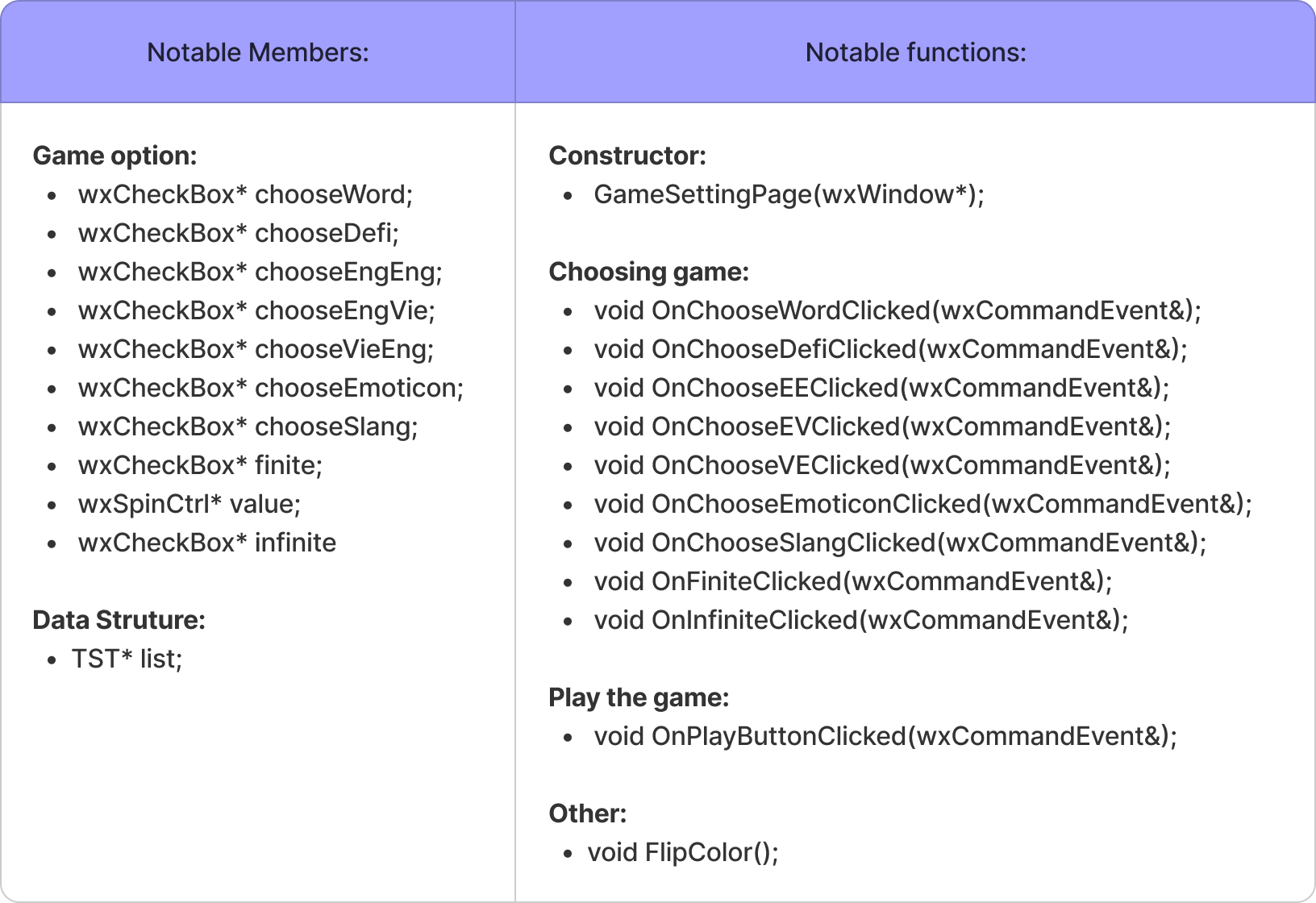


Figure 13. Game-setting-page notable features

* Users can choose 2 types of game to play, guess the word by its meaning or guess the meaning with the given words. They can also choose the dataset they want and set the number of questions they want by change the state of the **wxCheckBox**
* It also includes several methods to catch the events. After setup, the user will be navigated to Game play page to play the game.

## Gameplay-page.h:

This is a word game, which will be displayed in another window for users to play. If they can pass all the questions correctly, a pop-up with the text “You win” will appear and vice versa.

A computer screen shot of a program code

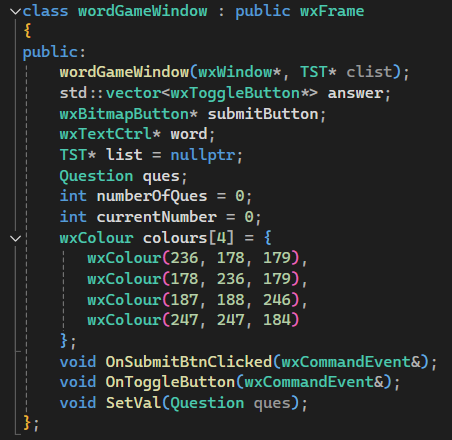
Description automatically generated

Figure 13. wordGameWindow and DefiGameWindow notable features

Functions and members related to this class:

* void OnSubmitButtonClicked(wxCommandEvent&);

For both games, when users click “Submit” button:

* If the answer is wrong, there will be a message “You lose!!!” and end the game
* If the answer is correct, there will be a message “You win!!!” and you can continue with the remaining questions.
* void SetVal(Question ques);

For both games, this function just helps to set a random question.

* void OnToggleButton(wxCommandEvent&);

This function helps users to make sure that their answer is chosen and clicked.

## list-window.h:

A screenshot of a computer program

Description automatically generated

Figure 14. listWindow notable features

Functions and members related to this class:

* bool **loadData**(std::string path, int isFav): Load History or Favorite words to vector data.
  + Time Complexity: O (n), n is the number of words in the file
* bool **saveData2File(**std::string path): Save History or Favorite words to vector data.
  + Time Complexity: O (n), n is the number of words in the file
* void **deleteSelectedRows**(): Delete selected rows
  + Time Complexity: O (n x m), n is the number of words in the file, m is the number of selected rows
* void **AppendRows**(vector<word>& words, int isFav): Add new words to the list
  + Time Complexity: O (n x m), n is the number of words in the list, m is the number of added words

## More-page.h:

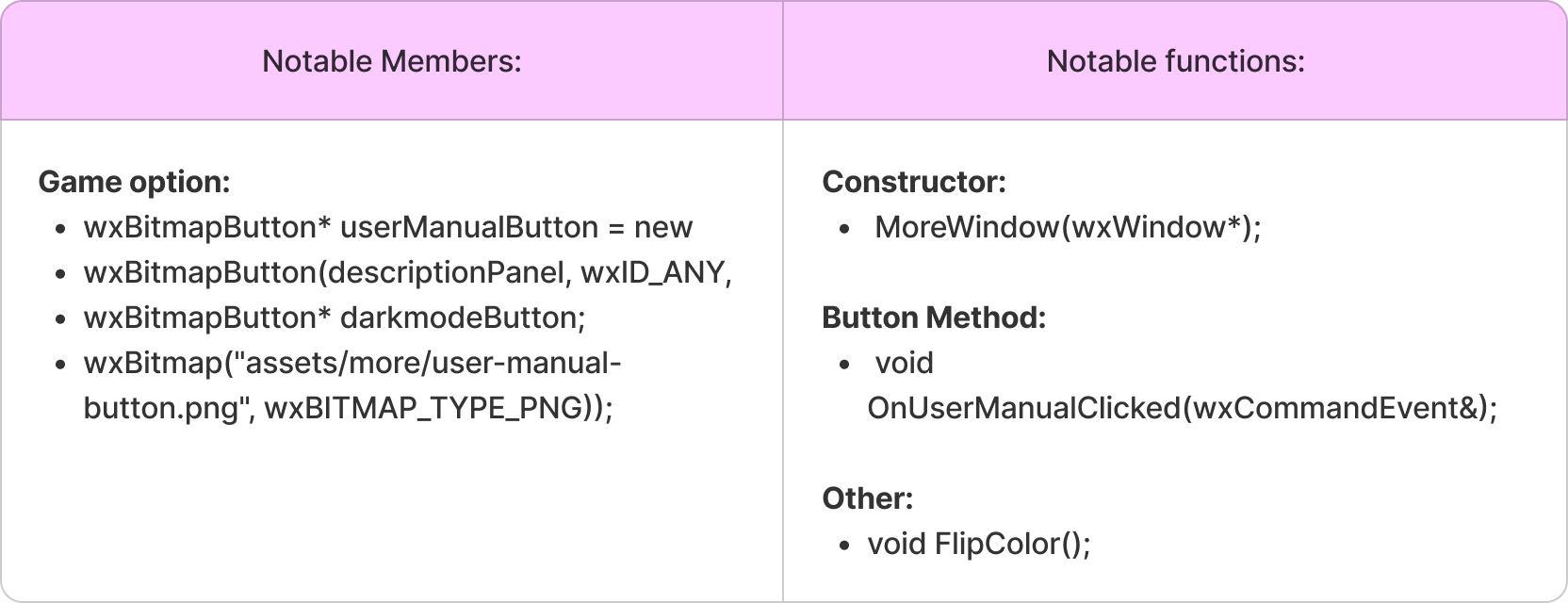


Figure 15. More-page notable features

Functions and members related to this class:

* void OnUserManualClicked(wxCommandEvent&);

This method is used to connect the button “User manual” with the link to our report.

* void FlipColor();

This method will change the current themes of our project from Light Mode to Dark Mode and vice versa.

# Technical Problems and Solutions:

We encountered some big technical challenges that we must resolve together, include:

## Back-end:

### Data Consistency Issues

When we prepared datasets for our project, we could not find 5 datasets with the same format, so we had to re-format the dataset into 2 formats. Due to the lack of time, we cannot perform good preprocessing data and thus encounter many difficulties in reading the content generally (and end up must build one for each kind of data). We had slightly modified them to give better performance, yet they are not fully optimized.

### Search with definition

After researching TST (Ternary Search Tree), our team initially considered using it to store the entire definition of each word. However, during the exploration of this approach, we discovered that to search for a substring within a definition, the TST would need to be stored as a suffix tree, which would result in a very large storage requirement.

On the other hand, the team was also concerned that string-matching algorithms like Knuth-Morris-Pratt, Rabin-Karp, or Aho-Corasick might be faster in constructing and searching words, but the need to repeatedly build arrays based on the searched string led us to favor suffix arrays instead. Furthermore, although there are many algorithms that can construct suffix arrays in linear time (even in the worst-case scenario), due to limited time and the high complexity of these algorithms, we decided to use the Manber & Myer algorithm because of its simplicity and relatively good runtime performance.

## Front-end:

Since we aren’t used to using wxWidgets, and it is also slightly outdated, the UI implementation is needlessly complex and took a lot of time. Better research into the GUI Libraries should be done to choose one that suited the project the most.

# Feature Demonstration:

Showcase all the features of the project, either through a video clip or screenshots. Provide a step-by-step explanation of each feature and its functionality.

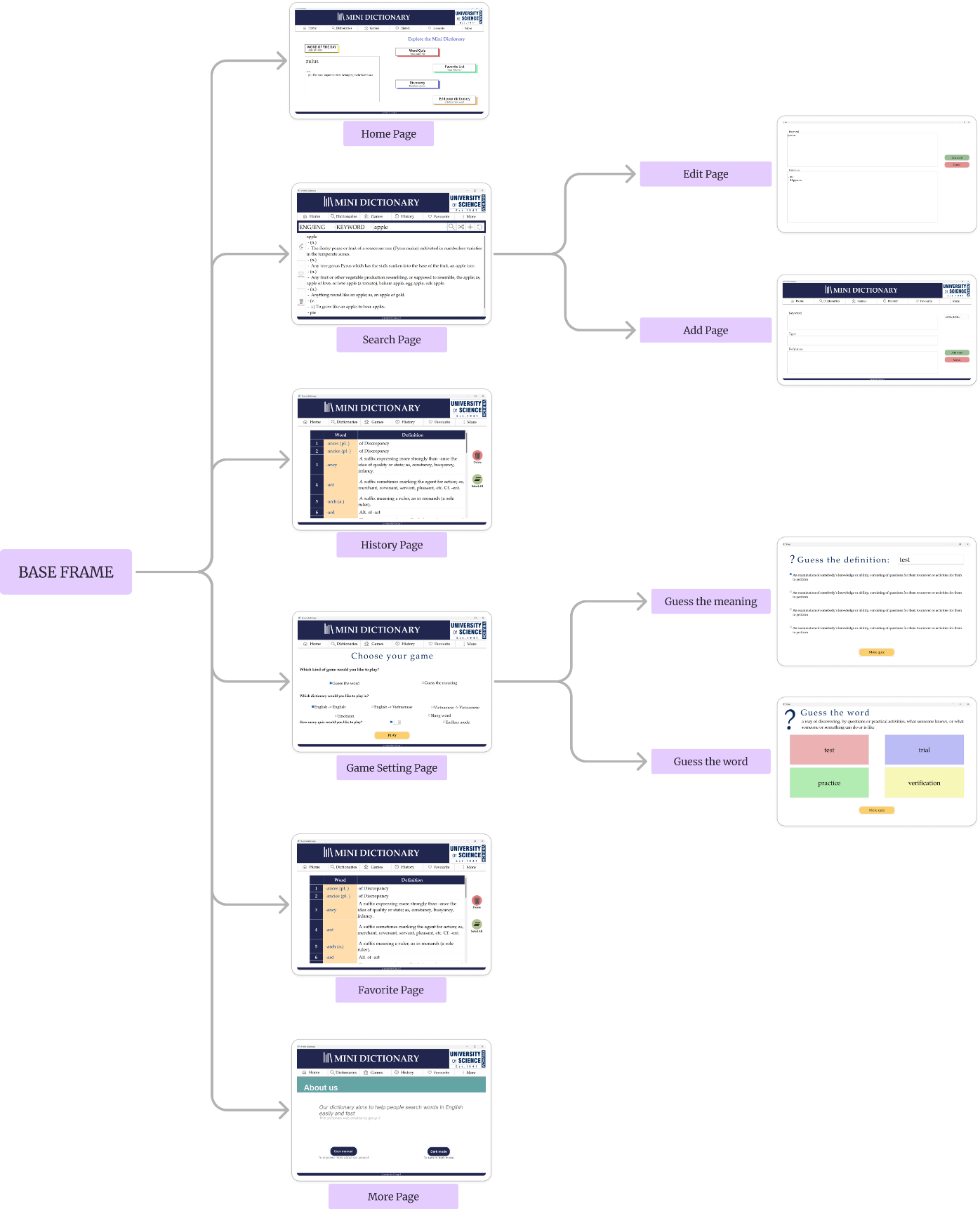


Figure 16. UI Map

## Home Page



Figure 17. Home Screen

Navigation Bar: The navigation bar will always be on the top, under the Header.

* **Home button**: Return to Home Page (figure n, right)
* **Dictionaries button:** Return to Search Page
* **Games button:** Return to Choosing game Page
* **History button:** Return to History Page
* **Favourite button:** Return to Favourite Page
* **More button:** Return to More Page (contains additional setting of the dictionary and more information)

Word of the day: A random word every time you open the dictionary for a little fun 😊

Explore more: This section encourages you to explore more about the dictionary and its additional functions.

* **Word Quiz**: Return to Game Page
* **Favorite List:** Return to Favorite Page
* **Discovery**: Return to Search Page with a random word displayed
* **Edit your dictionary:** Return to Editing Page

## History/Favorite Page

History/Favorite List: The search word history or the favorite list will be present in a grid, the first column is the words, and the second column its meaning. You can choose one or multiple words, by either using your mouse or SHIFT and CTRL key. If the list is too long, there will be a scrollbar next to the list.



Figure 18. History/Favorite Screen

* **Select All**: Select all words present in the list.
* **Delete:** Delete all the selected words.

## Dictionary/Search Page

Search Bar: The search bar will always be under the navigation bar. This includes:

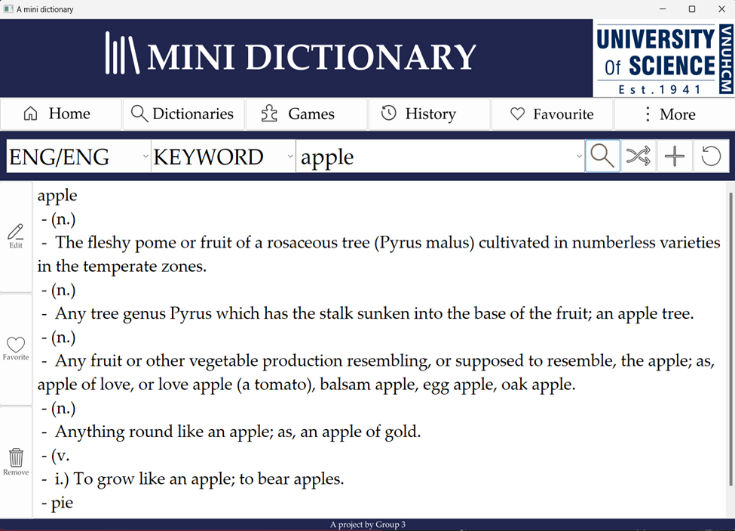


Figure 19. Search Screen

* **Choosing dictionary:** You can change your data set by clicking the button on the far left of search bar (default: ENG/ENG). It will drop down a list of dictionaries available. Choose your desired dictionaries by clicking on one of them.

There are 5 dictionaries available: English-to-Vietnamese, English-to-English, Vietnamese-to-English, Emoticon, and slang.

* **Choosing search function:** You can change your search function by clicking on the button next to the above button (Default: KEYWORD). It will drop down two options. Choose your desired search function by clicking on either KEYWORD or DEFINITION.
* **Searching:** Type your search into the search box. It will drop down a list of recommended words. Choose your desired word and press the Magnifying Glass button (right of search bar) to search.
* **Randomized word:** If you want to view a random, arbitrary word, press the Shuffle button next to the search button (right of search bar).
* **Add a word:** You can add a custom word to the dictionary by pressing the plus button (right of search bar) to return to Editing Page.
* **Reset the Dictionary:** To reset the dictionary to its original state, you can press the Repeat Button (far right of search bar). This will reset all the data in the dictionary.

Tool bar: The search bar will always be on the left of the screen. This includes:

* **Edit button**: Edit the definition of the currently displayed word.
* **Favorite button:** Add the currently displayed word into the favorite list (can be view in Favorite Page)
* **Delete button:** Delete the currently displayed word from the dictionary.

## Game Page



Figure 20. Game Setting Screen

To choose your game: Press the buttons to customize your word game. This includes:

* **Choose the types of game**: We offer you two options: Guess the word (we show you a definition and 4 words, choose the correct answer among them), and Guess the meaning (we show you a word and 4 definitions, choose the correct answer among them)
* **Choose dictionary:** Choose one of the five available dictionaries for our game, you may only pick one.
* **Choose number of rounds:** Choose a number of rounds you one to participate in by typing in or clicking on the box (right of the question) or choose to play in Endless Mode. The game stops when you lose.

## More Page

About us: More information about us and our project: Our objective, name, and additional features. (And room for more developments)



Figure 21. More About Us Screen

* **User manual**: Direct link to our github repo.
* **Dark mode:** Additional feature that change from light mode to dark mode.

## Game: Guess the word

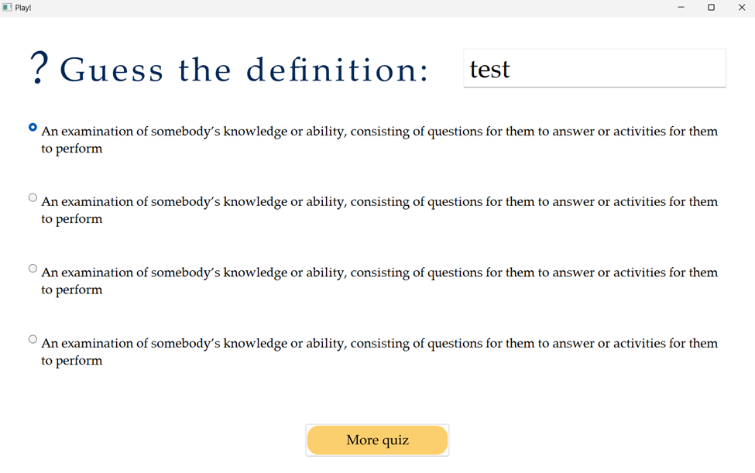


Figure 23. Guess the definition game

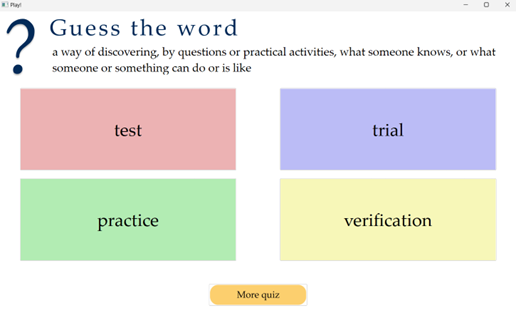


Figure 22. Guess the word game

**Choose the right answer**: by either choosing a word (by pressing the colored box) or choosing a definition (by pressing the circle before the definition), press the yellow button below to check and move on to the next question.

## Add Word Page

To add a word: Fill the blank space below to add a new word to your chosen dictionary:



Figure 24. Add word Screen

* **Keyword**: Type in your new word here.
* **Type:** Type in the type of your new word or left it blank.
* **Definition:** Type in the definition of your new word.
* **Dictionary box:** Chose the dictionary you want to add your word to: Press and choose one of the five available dictionaries.
* Press the green button (Add word) to add the new word, or cancel the addition by pressing the red button (Cancel)

## Edit Word Page

To edit a word: Fill the blank space below to edit the current word you are viewing:

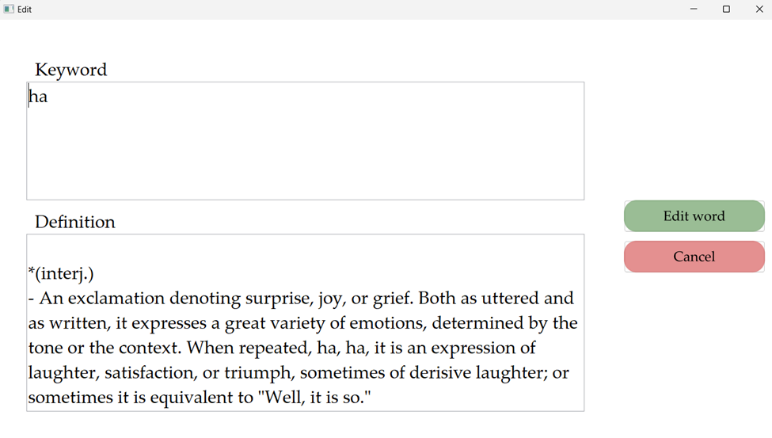


Figure 25. Edit Word Screen

* **Keyword**: Change the word.
* **Definition:** Change the definition of your word.
* Press the green button (Edit word) to make the change, or cancel the modification by pressing the red button (Cancel)

# Conclusion

The mini dictionary project successfully achieved its objectives of creating a functional and efficient dictionary application using C/C++. It was an exciting project to work on, and we gained a lot from it, above and beyond its original scope. We were able to study and research many different topics outside of our class.

Although the application is far from perfect, it works as we desired. The mini dictionary showcases the group's proficiency in data structures handling, and all the core and additional functionalities work successfully. While the project has met its initial goals, we believe there are a lot of potential enhancements for this application in the future, including but not limited to transforming the data sets into one unify format that is more easily and effectively processed, and more efficient use of data structures.

# References

[1] [wxWidgets User Manuals](https://docs.wxwidgets.org/3.2/)

[2] [Lê, M. H., Cây hậu tố và một số ứng dụng trong xử lý xâu](https://drive.google.com/file/d/0BwcTB8a10LBwYUwwNVYzbmZiZnM/view?resourcekey=0-TyQK7KSoZJW-fVpHquw6NQ)