DEPARTMENT OF COMPUTER SCIENCE

RFA: Final Report

Capstone Project

prepared by

Lauren Antrobus
ANTLAU001@myuct.ac.za

Merada Richter
RCHMER002@myuct.ac.za

for

Sonia Berman
(p.p. help2read)

on

22 September 2013

1. Table of Contents

2. Executive Summary

The software developed by this project is a tool directed at young children whose English reading and comprehension skills are not strong. *Flua*, the name of the software, aims to encourage and improve children's reading fluency through the use of a gamified environment (implemented using Java). Two games are available to users (designed to be played by a child paired with a volunteer) as well as a dictionary function to aid in broadening vocabulary.

3. Introduction

Flua[1] was designed specifically with the needs of the Help2Read organisation in mind. Learners are mostly primary school children from disadvantaged backgrounds with little reading experience. English is often not a first language for these children.

Currently, Help2Read uses a paired reading program that teams each child with an adult volunteer. The children learn through reading books, playing games and doing comprehension activities under the guidance of the volunteer. In keeping with this model, we designed our games to be played by a child and volunteer team – however the interaction is focused on the child and the volunteer's role is largely supervisory.

Because the users of Flua are typically disadvantaged children, it is likely that they are not experienced with using a computer. For this reason, it was highly important that the user interface is intuitive and easy to use.

Another important aspect that we focused on is user experience: the software needed to be fun to use, so that the children's interest is kept. For this reason we chose to gamify the activities, as well as make use of a fun and attention-grabbing graphical user interface (GUI).

In our approach to designing our software, we used a mixture of traditional and agile software engineering methods. Our planning and design processes followed traditional methods as the whole system was planned and designed before implementation started and we made use of an evolutionary prototype. During implementation, however, we used the agile notion of iterative development, as we developed the system in increments and adapted to difficulties by modifying the design.

[1] **FLU**ency **A**id

3.1. Statement of Scope

This section summarises the scope of the project, according to the agreement between the client and the development team. Please see ***Table 1*** for details.

Table 1: *Statement of Scope*

FUNCTIONS	**Fill-a-Word game:** • Users are presented with an image of a busy scene, and sentences describing something happening in the picture. These sentences will be missing words, which should be filled in by the user. **Create-a-comprehension game:** • Pupils at Help2read often struggle with words such as "who", "what" or "how", which are difficult to sound out and explain. This activity aims to make the pupils more familiar with these kinds of words. Users will be presented with a story, which they can read in a "book" on the GUI. They will then have to create their own comprehension questions about the story. Suggestive words (such as "which" or "who") will be provided in a word tray on the side of the screen. **Dictionary:** • Users can look up a word they might not understand, and can add a word if it does not form part of the database. This will hopefully help to ensure users' understanding and develop their vocabulary.
INPUTS	**Keyboard inputs:** • Words for sentence completion in the Fill-in-the-word game • Questions for the Create-your-own-comprehension game **Mouse inputs:** • Start a game • Select menu/control options • Select words from the word tray • Activate input tray
OUTPUTS	**Monitor outputs:** • Display game resources • Echo user input in input tray • Show dictionary lookup results
CONSTRAINTS	• Due to lack of internet availability, the program must contain all necessary resources as it will be run locally. • The user interface is required to be simple, intuitive and user friendly, as end users are disadvantaged children who are not highly proficient in reading and are unlikely to be very familiar with computers.

4. Requirements Captured

This section details the requirements for the Flua program, as determined by the client. Each requirement is briefly explained before being described in terms of its ***Use cases*** and ***Use case narratives***.

4.1. Functional Requirements

4.1.1. Main Menu

The menu is used to interact with the program via the following options:

- Start a Fill-a-Word game
- Start a Comprehension game
- Exit

It is the main entry point into the graphical user interface.

4.1.2. Dictionary

The dictionary function allows learners to look up words they do not understand, and volunteers to add definitions for words which are not in the current database.

4.1.3. Resources

Resources for the various games and dictionary functionality are loaded and displayed in the user interface. These resources include:

- Images
- Text (for stories, questions and help)
- Dictionary files

4.1.4. User input

Users interact with the interface via keyboard and mouse.

4.1.5. Words for word tray

The word tray contains word hints for learners to assist with vocabulary development.

4.1.6. Local operation

The computers on which the program will run do not have Internet access and are not networked. This requires Flua to be implemented as standalone software complete with all necessary resources prior shipping.

4.2. Non-Functional Requirements

4.2.1. Maintainability

Since there will be little on-site help once the program is released, Flua should not crash or become unusable due to code errors. Should a fault occur in the program, Flua should return to a previously operational state, such as the main menu.

4.2.2. Documentation

All aspects of the program should be well recorded and this documentation included with the release to allow developers to quickly comprehend the system should it be extended (please see the next point).

4.2.3. Extensibility

There is great potential for Flua and many more functions can be added to the program to make it a better teaching tool. It should be easy to expand the program and add additional functionality without having to rewrite major sections of code.

4.3. Usability Requirements

4.3.1. Easy to learn

Since the users of this system are not guaranteed to be familiar with computer programs, the interface should be intuitive and simple to learn. Complex user interactions should be avoided.

4.3.2. Enjoyable to use

Users should enjoy using Flua so that they are encouraged to continue using it, thereby further developing their reading fluency.

4.3.3. Interesting and colourful interface

Bright, bold colours are appealing to young children, and are likely to retain interest for longer. The interface should reflect these traits in order to increase user creativity and receptiveness to learning.

4.4. Use cases

Figure 1 details the use cases of the Flua program.

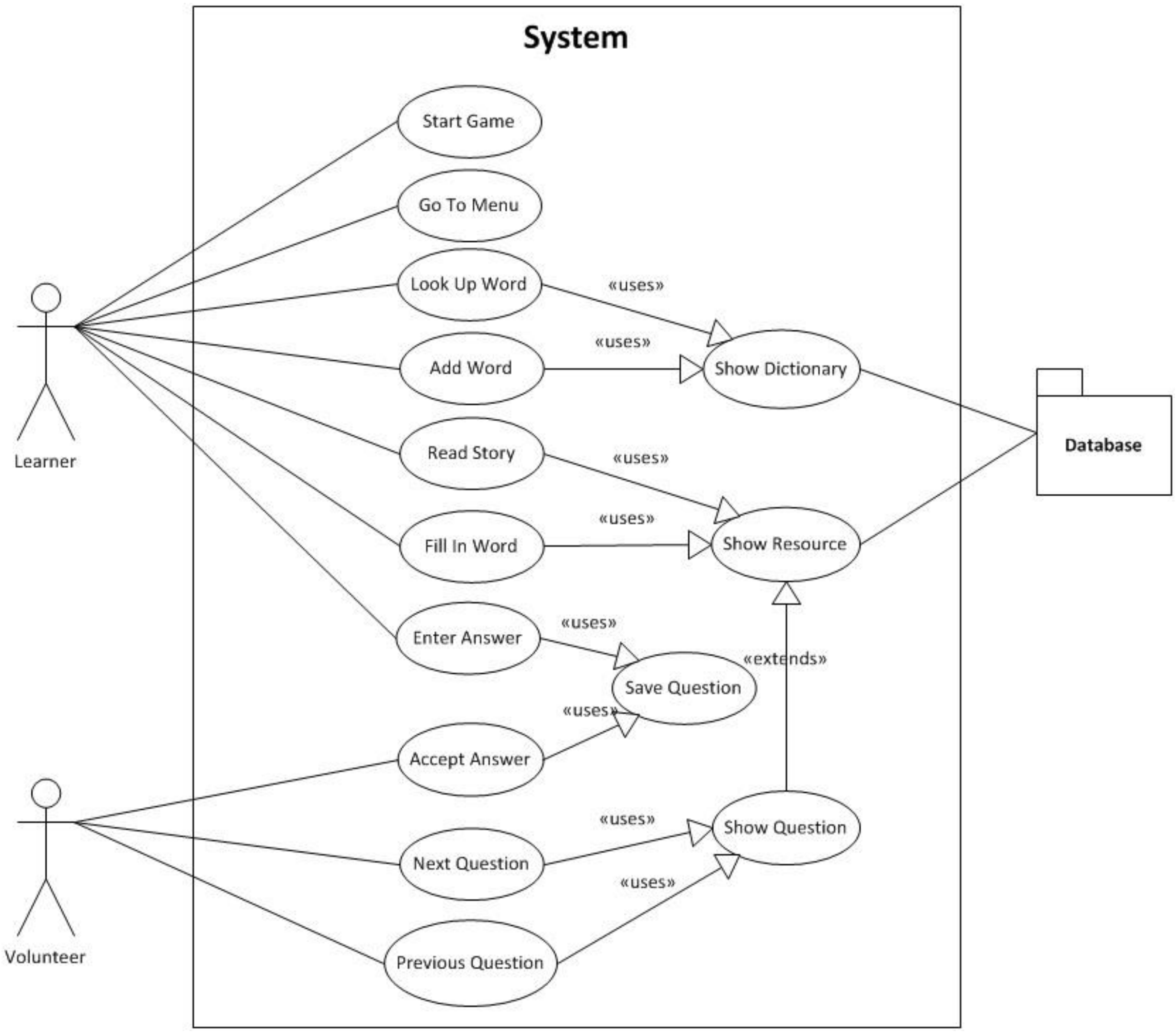


Figure 1: *Use case diagram of Flua*

For the final class implementation of the program, please see ***Figure 3***.

4.5. Use case narratives

Use case narratives for the most pertinent use cases are given below in brief format.

4.5.1. Start Game

Actors: Learner, Volunteer

Precondition: User is on the main menu screen.

A user selects a game (either Fill-a-Word or Create-a-Comprehension), and the System initializes a new game of that kind.

Alternative: The user exits the program.

4.5.2. Go to Menu

Actors: Learner, Volunteer

Precondition: User is on the game screen.

A user clicks on the main menu button, and the System returns to the menu screen.

Alternative: The user continues playing the current game, starts a new game of the same type, looks up a word or exits the program.

4.5.3. Look Up Word

Actor: Learner

Uses: Show Dictionary, alt. Add Word

Precondition: User is on the game screen.

Learner selects the dictionary function and the System opens the dictionary. The Learner types in a word to define and the System displays the definition.

Alternative: If the word is not in the dictionary, the Volunteer is requested to define it (uses Add Word).

4.5.4. Read Story

Actors: Learner, Volunteer

Uses: Show Resource

Precondition: Create-your-own-comprehension game chosen

The System displays a story the Learner has not yet read in the resource section. The Learner reads the story, using the mouse to turn pages. The Volunteer assists the learner in reading and sounding out words.

At the end of the story, the Learner is requested to enter questions for the comprehension.

4.5.5. Enter Answer

Actor: Learner

Precondition: User is on the game screen.

Uses: Accept Answer, Show Question, Next/Previous Question

The Learner types an answer. The Volunteer checks the answer. An accepted answer is saved and the next question is displayed.

Alternative: The Learner moves to the next or previous question. The current answer is not saved. Any accepted answer is saved and can be returned to for further editing.

5. Design Overview

5.1. Flua architecture

Figure 2 details the architecture of the program. Flua was designed to have a simple, elegant code structure which modularizes functionality to allow for efficient development and code reuse.

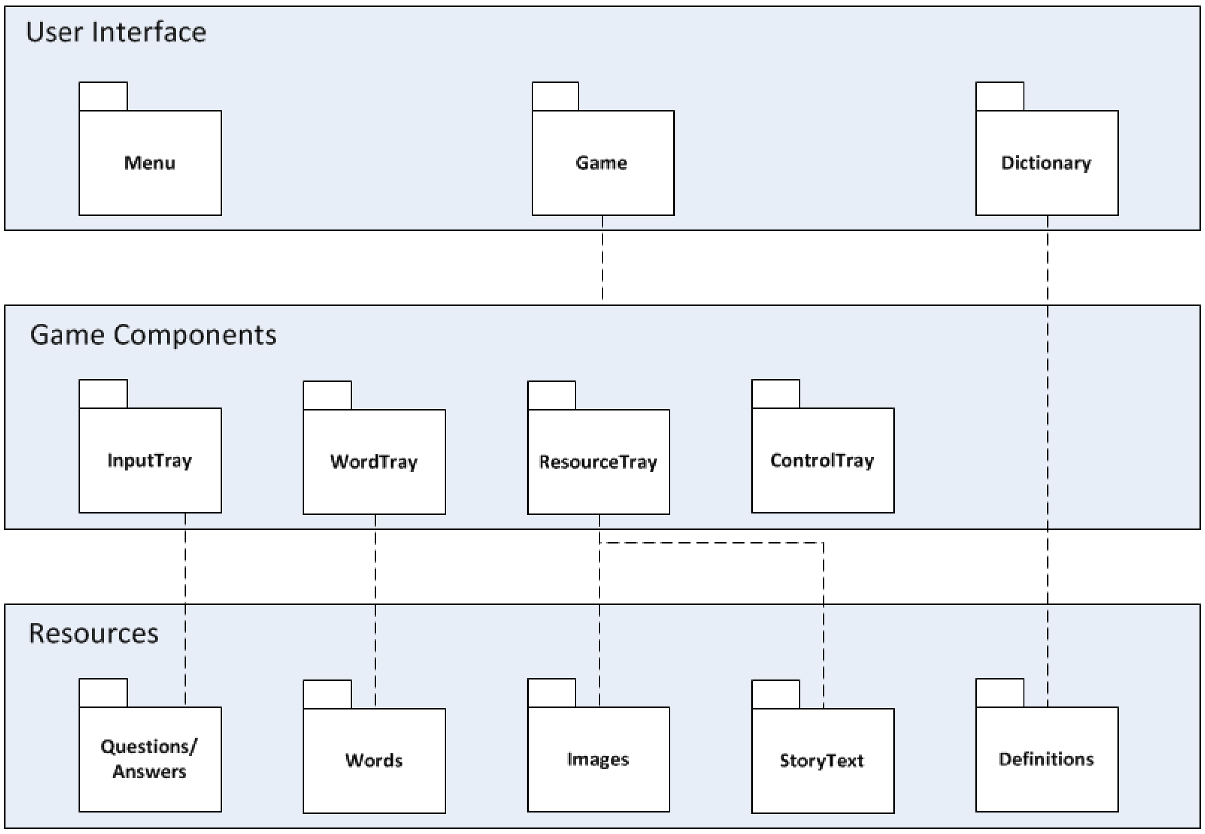


Figure 2: **Archtecture diagram of Flua**

5.2. Flua class design

Figure 3 details the design class diagram for the program.

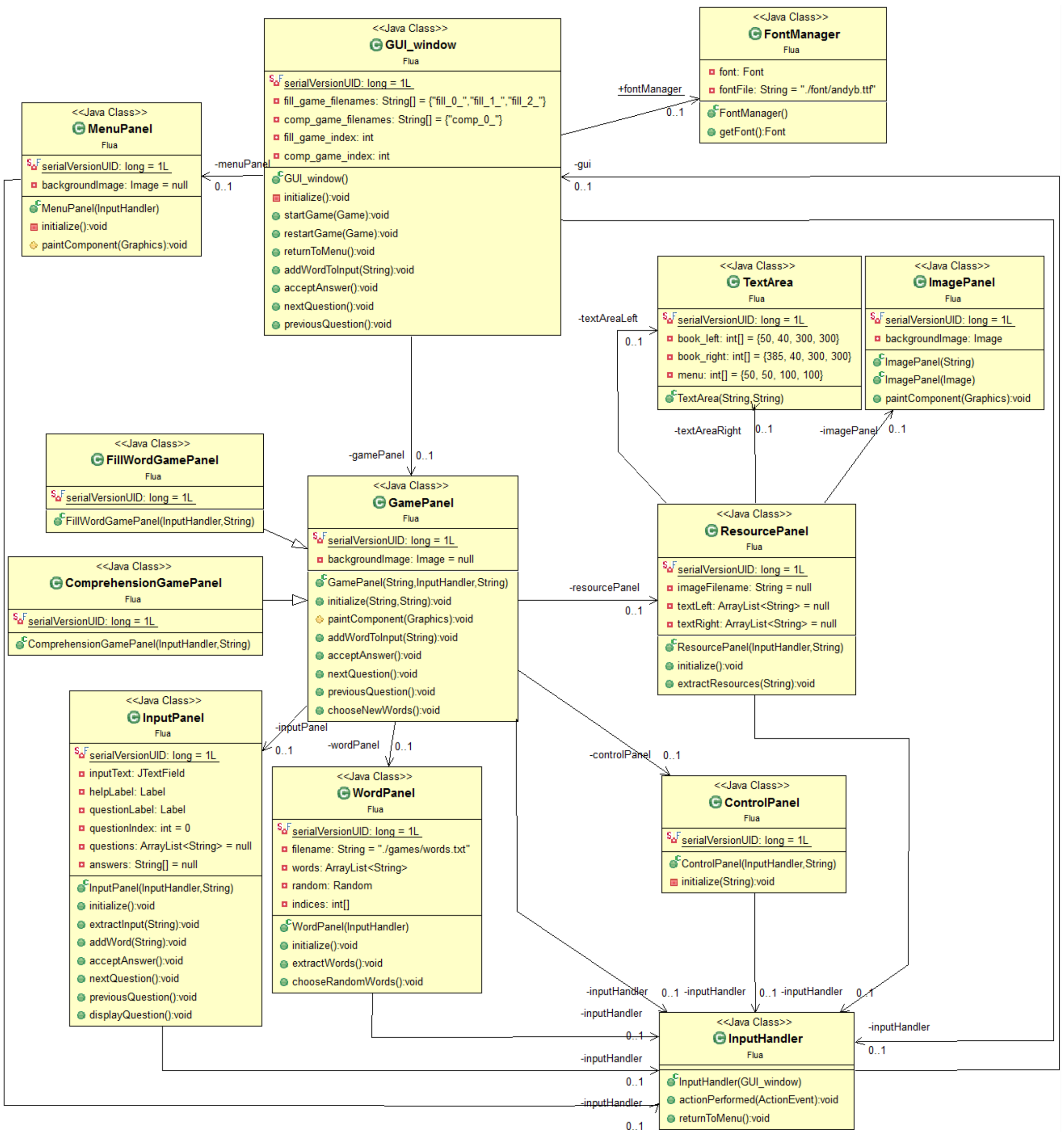


Figure 3: Design class diagram for Flua's user interface

5.3. Data Organisation

Flua is divided into three layers:

- The user interface (including games and the dictionary)
- The game Trays (Resource, Controls, Input and Word Trays)
- Resource files

In order to efficiently manage data and reduce the size and number of messages passed between classes, each game Tray manages its own resources. This includes extracting information and initializing all required data structures.

The dictionary is an independent module and also manages its own data.

More detail regarding the data organisation and structures implemented is given in the next section.

6. Implementation

6.1. User Interface

The graphical user interface was carefully designed to allow each function to be implemented efficiently and elegantly.

6.1.1. Game Trays

A game contains four Trays, each with its own unique functionality:

- Resource Tray
  - Displays images and story text
- Control Tray
  - Contains game controls such as return to menu, new game and dictionary functions
- Word Tray
  - Shows random words to assist learners in answering questions
  - Words are interactive and can be entered into the input text fields but clicking on them
- Input Tray
  - Allows learners to input text answers to questions
  - Displays questions
  - Contains a help label giving assistance in answering the question
  - Contains buttons to allow questions to be saved, skipped or returned to

These trays are implemented as separate classes and manage their own resources and user action (see ***Figure 4***).

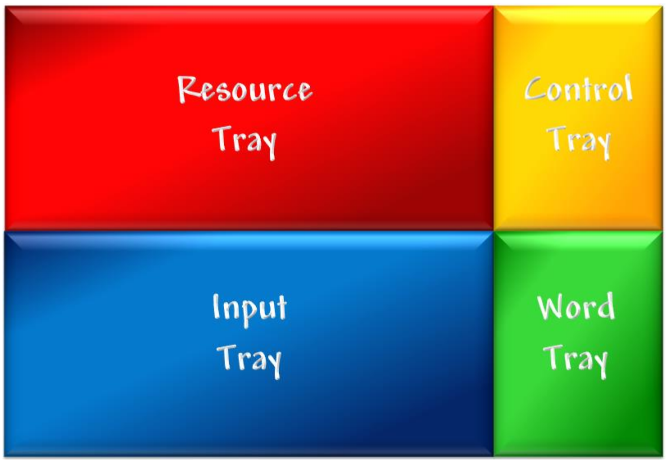
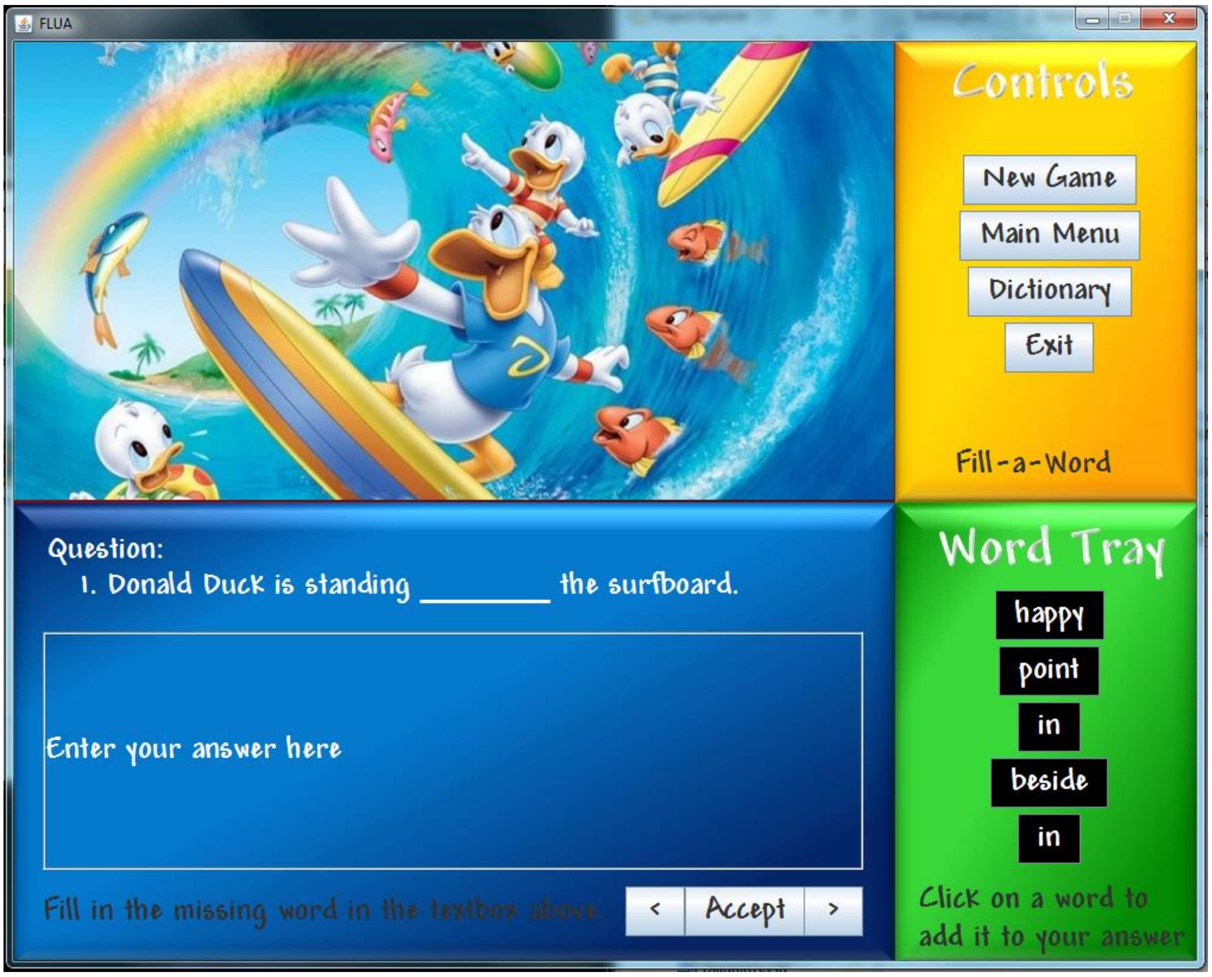


Figure 4 *The 4 trays comprising the game screen*

The final game screen can be seen in ***Figure 5***.



Question:
1. Donald Duck is standing ________ the surfboard.

Enter your answer here

Fill in the missing word in the textbox above.

Click on a word to add it to your answer

Figure 5: *Example of a game screen*

6.1.2. Questions and User Answers

Game questions are stored in ArrayLists so that we don't have to calculate how many questions are in the resource file. The number of questions can also differ from game to game.

The user answers are stored in a String array of the same length as questions. This ensures that question and answer indices always correlate (it is easier to keep track of a joint index). This allows completed answers and their questions to be displayed easily in the Input Tray.

6.1.3. Game Resources

These resources are also stored in ArrayLists. Each Tray manages its own resources.

Text for the stories is split into two lists: one for the left page and one for right. These correspond to lines in the resource file: the first line will be a left page, the second line a right page and so on. This makes the story easy to display in the Resource Tray.

The words for the Word Tray are randomly chosen from an ArrayList initialized from a resource file of words. These words are stored in an array for the current game and are reset when a new game is started.

6.2. Dictionary

We had initially hoped to make use of a MySQL database for the dictionary data storage, but were unable to find one suited to our needs. Considering the context of Flua's users and their weak literacy skills, the existing databases that we found were unnecessarily extensive and contained overly complex words and definitions.

Our next option was making use of dictionary data from a file, which also proved to be difficult. Many of the sources we found contained only words (without definitions) or too contained words and definitions of an inappropriate level for the children's reading abilities.

Ultimately, we created our own text file for the dictionary data – building on a suggested vocabulary list from ReadingRockets.org[2] and filling their definitions from the dictionary definition generator EasyDefine.com[3]. Once we had data to load, we used a hash table to store the dictionary elements for a few reasons. By choosing this data structure, searching for words becomes easy and efficient. It also lent itself well to a dictionary application, as words can be used for the key and their definitions as the corresponding value.

The operation of the dictionary is given in the flowchart of ***Figure 6****Figure 1*.

[2] Please see reference section for links

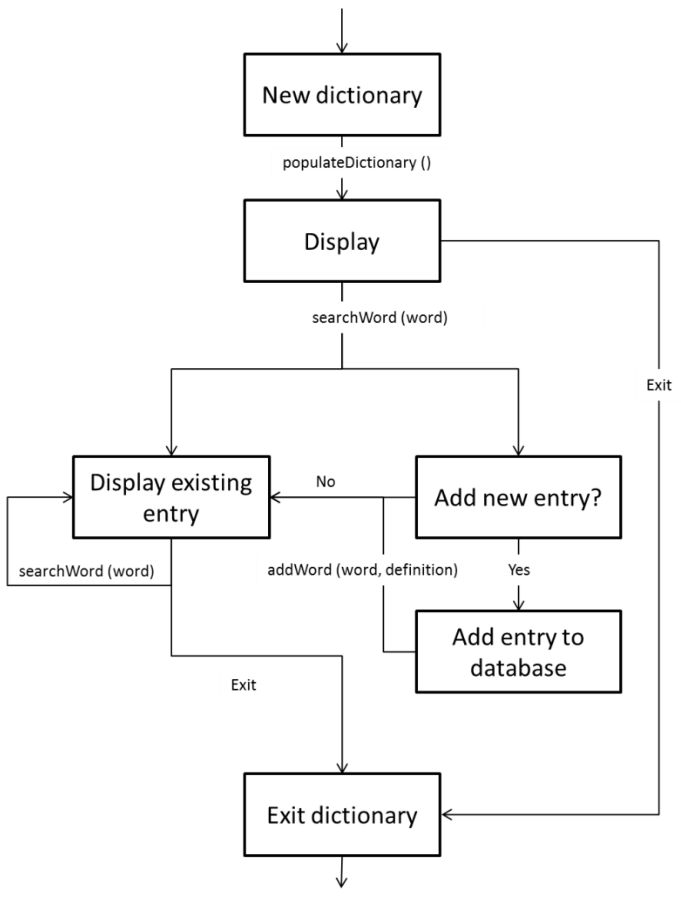


Figure 6 Flowchart depicting dictionary operation

6.3. Sequence Diagram

The sequence diagram for the interaction between the user and Flua is given in ***Figure 7***. For simplicity and ease of reading, individual Trays have not been included in the diagram. Also, method names do not correlate directly, but have been used to aid in the rapid comprehension of system activity.

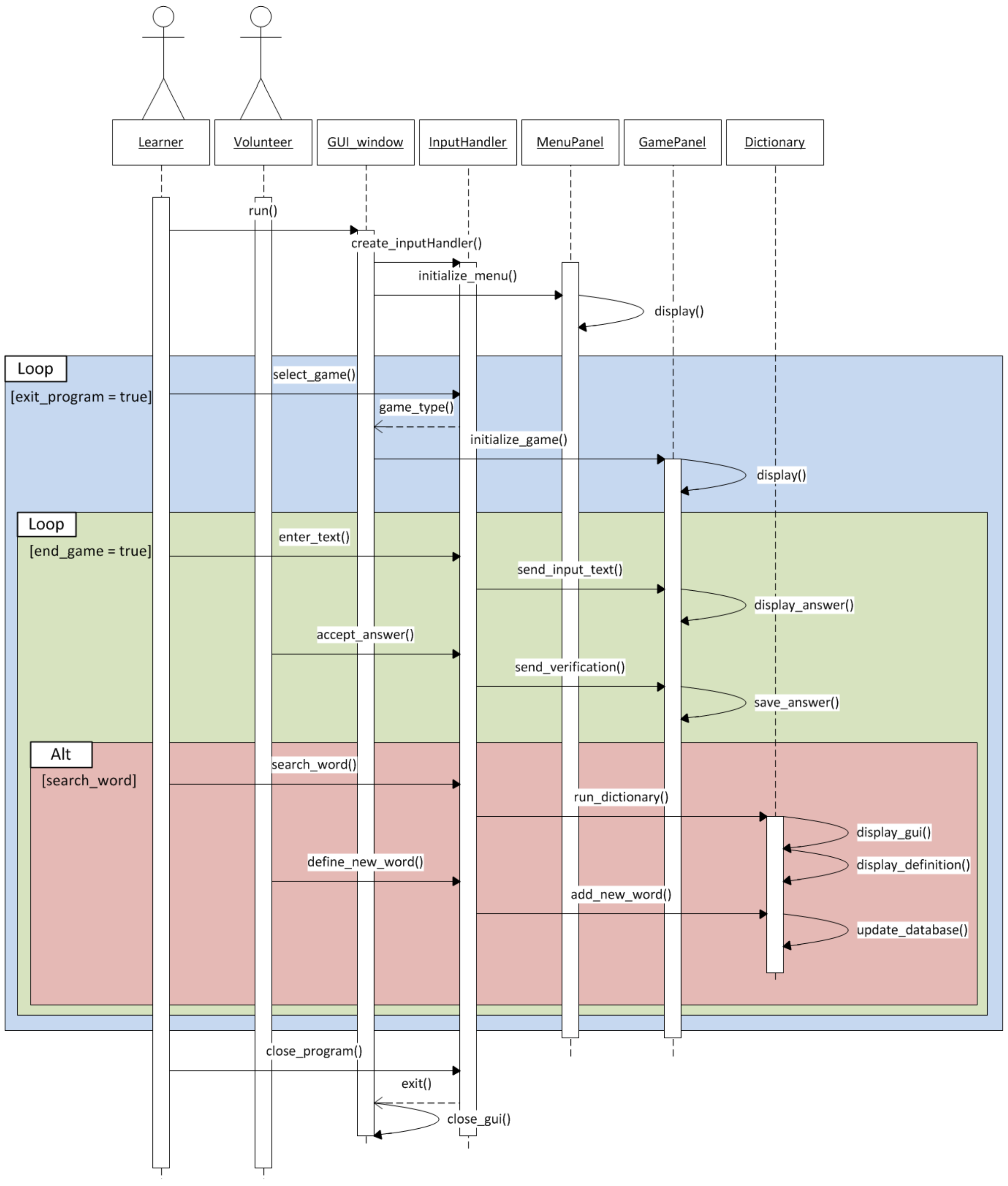


Figure 7: *Sequence diagram for Flua*

7. Program Validation and Verification

Flua was extensively tested at every step of development and has been found to be robust and unlikely to throw errors for user input. ***Table 2*** details the tests performed to validate the interactive parts of the program, as well as the test results[3].

Table 2: *Tests performed to validate FLUA*

TEST CASE	INPUT	EXPECTED OUTPUT	NORMAL DATA	ILLEGAL DATA
MAIN MENU				
New Fill-a-Word game **New Comprehension game**	Mouse click	Initializes a game of the specified type	Passed	N/A
Exit button	Mouse click	Exits program	Passed	N/A
RESOURCE TRAY				
Load and display story text **Load and display image**	Resource files	Passed	Passed	Returns to main menu if files aren't found
CONTROL TRAY				
New Game button	Mouse click, current game type	Opens new game of same type	Passed	N/A
Main Menu button	Mouse click	Returns to the main menu, removing current game data	Passed	N/A
Dictionary button	Mouse click, dictionary resource files	Creates new dictionary and loads it with definitions	Passed	Creates error message indicating dictionary unavailable
Exit button	Mouse click	Exits program	Passed	N/A
WORD TRAY				
Generate random words	File of words	Chooses 5 random words from the given list and creates buttons for them	Passed	Returns to the main menu if files aren't found
Add word to answer	Mouse click on word button in tray	Inserts word into input tray	Passed	N/A
INPUT TRAY				
Display question	Resource files	Displays questions and help text as defined in resource files	Passed	Returns to main menu if files aren't found
Display user input	Text input	Displays user typed text in	Passed	Ignores data

[3] As a relatively small number of tests were required to test all of the program's functionality, an appendix has not been used to detail these tests.

		input tray		
Save user input	Mouse click	Retains answer even if next/previous question is chosen	Passed	N/A
Display next/previous question	Mouse click	Moves between successive questions, stopping at boundaries (first and last questions)	Passed	N/A
Dictionary				
Look up word	Text input for word to search for	Displays definition of required word	Passed	Requests definition to be added to words not in database
Add new word	Text input for word and definition	Adds new word to dictionary	Passed	Does not save the word

Flua passed all tests for valid user input, and will not crash nor throw errors for invalid user input. It is a very good tool for learners to use as very little (if anything) can go wrong during user interaction. This is especially important when dealing with users who are not used to working with computers.

However, the program is not very usable if the required resource files are not present. It will not move beyond the main menu screen. It is therefore vital that the program be shipped complete with all given resources.

8. Conclusion

All required functionality for Flua detailed in ***Functional Requirements*** is operational as shown by the test results. The success of the ***Non-Functional Requirements*** and ***Usability Requirements*** will be determined when the program is given to a test group of the demographic it was designed for.

During implementation, it was decided that it is best for the volunteer to define words instead of the learner (as it is in the use case diagram of ***Figure 1***).

Due to care taken during planning and prototyping, the program code is very well structured. Separate modules exist for the dictionary and user interface, and the modules can be used independently if required. Additional features can be added quickly.

The program is robust for user input and does not fail for any keyboard or mouse interaction from the learner. There is a danger if resource files are not present, so care must be taken to include all files when installing the program.

8.1. Recommendations

Flua can be extended very easily to allow the following functionalities:

- User log in
- Saving user progress in static files (all questions and answers)
- Additional games

The entire infrastructure for the above is included in the design of the program.

The program can be further improved by adding additional resource content such as images, stories and questions.

9. User Manual

For instructions on how to use Flua, please see the accompanying document entitled ***ANTLAU001_RCHMER002_RFA_UserManual***.

10. References

Graham, S., Harris, K.R. and Loynachan, C. 1993. *The Basic Spelling Vocabulary List*. Available from: http://www.readingrockets.org/article/22366/. [2013, September 20]

Jain, P. & Choi, C. 2009. *EasyDefine*. Available from: http://www.easydefine.com/ [2013, September 20]