Ladder-gram Assignment

2810ICT

By

Student's Names: Cassie Kristenson and Kirra Rafton

Student Numbers: s5053148 and s5055297

Student Emails: [cassie.kristenson@griffithuni.edu.au](mailto:cassie.kristenson@griffithuni.edu.au)  
                  [kirra.rafton@griffithuni.edu.au](mailto:kirra.rafton@griffithuni.edu.au)

Github link: <https://github.com/cassie12345/Assignment>

Table of Contents

[**1.0 Project Overview** 3](#_Toc491036364)

[**2.0 User Requirements** 3](#_Toc491036365)

[The following outlines the user requirements for the program: 3](#_Toc491036366)

[**3.0 Software Requirements** 3](#_Toc491036367)

[**4.0 Software Design** 4](#_Toc491036368)

[4.1 High Level Design 4](#_Toc491036369)

[4.2 Functions in the software 4](#_Toc491036370)

[4.3 Data Structures in the software 5](#_Toc491036371)

[4.4 Configuration Management and Version Control 5](#_Toc491036372)

[**5.0 Unit Tests** 6](#_Toc491036373)

[**6.0 Requirement Acceptance Tests** 6](#_Toc491036374)

[**7.0 User Instructions** 7](#_Toc491036375)

**1.0 Project Overview**

This document provides an overview of a ladder-gram program. It will include a problem statement, user requirements for the game, software requirements, software design, unit test module, and software configuration management and user instructions

The objective of ladder-gram is to transform a source word into the target word in the least number of steps. During each step, you must replace one letter in the previous word so that a new word is formed, but without changing the positions of the other letters. The program needs to be able to read dictionary.txt, which contains the Official Scrabble Second Edition Dictionary, file in order for the game to select the word according to user's input.

**2.0 User Requirements**

The following outlines the user requirements for the program:

* Users required to enter a start word
* Users required to enter a target word
* Users are required to enter two words of the same length in order to continue
* Users are required to enter either ‘l’ or ‘s’, if they want the shortest or longest path from a start words to a target word
* Users are required to supply a list of words that are not allowed to be used within the steps from a start word to a target word if they wish

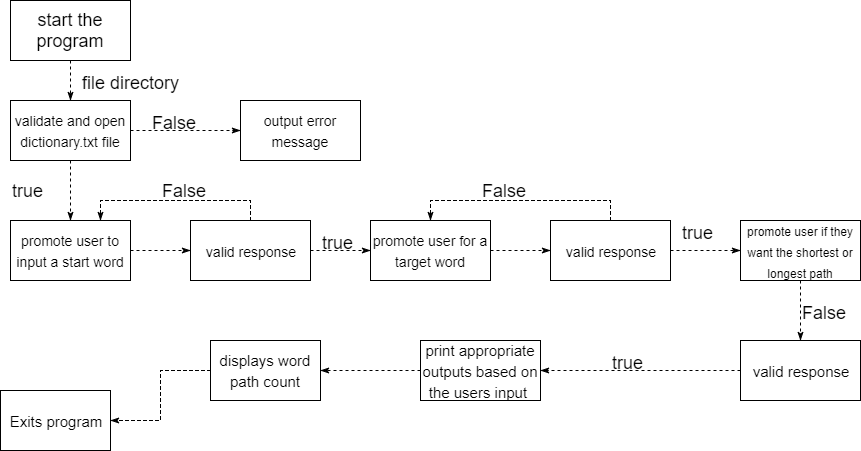
**3.0 Software Requirements**

The following outlines the software requirements for the program:

* Program imports re module for searching
* Required to read dictionary.txt file which contains the Official Scrabble Second Edition Dictionary
* Prompt the user to supply a list of words that are not allowed to be used within the steps from a start word to a target word
* Prompt user to enter ‘s or shortest’ or ‘l or longest’, whether they want to want to know the shortest path from a start word to a target word
* All possible user errors are handled correctly

**4.0 Software Design**

4.1 High Level Design



4.2 Functions in the software

same\_letters

* The same function checks if the start word and target word have matching letters
* Parameters used are item and target word

Check\_word(text)

* Checks if the user input is text
* Converts any user input into lower case
* Removes any white space from user input
* Checks if user input is all letters
* Returns output if user enters appropriate input
* Returns corresponding message to user if invalid input
* Prompts user to enter a valid word until they do so
* Parameter is text

build

* Searches to see if the pattern words are matching
* Parameters used are pattern, words, seen and match\_words

find

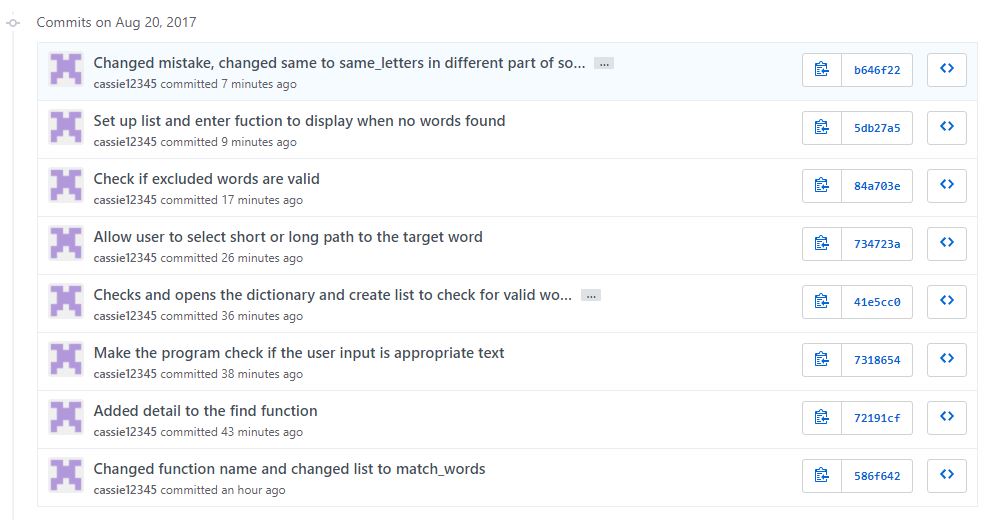
* Removes words if in word list or contain rare letters
* Removes words with rare letter if letter not in target word
* Exits code when there is no path
* Compares word with target word for matching letter
* Exits code upon getting to the last work
* Adds new items to the path match words
* Parameters used are word, words, seen, target\_word, word\_path and short

4.3 Data Structures in the software

* match\_words = [] : Stores letters of the word that need to be checked and replaced
* word = [] : Stores the start and target word from the dictionary that have been selected by the user
* Path = [start] : Stores start word as an index
* Path = {start: True} : Stores start word as a key and gives it the value of true

4.4 Configuration Management and Version Control

Version control allows a software team to better manage changes to source code over time by keeping track of every modification to the code. It allows developers to compare to earlier versions of the code if mistakes have been made. The use of version control systems allows software development teams to preserve efficiency of programs with a complete log of change history for every file, traceability of each change made and the ability to work concurrently without major complications. Therefore, it has been used in this assignment to easily trace our teams work, and work concurrently without having to continuously send program updates. Below is the history log of changes made to our program with a comment giving explanation to the changes made.



**5.0 Unit Tests**

| **No** | **Test Case** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- |
| 1 | Test a wrong filename | Exception Handled | Exception Handled |
| 2 | Test empty input file | Display error message and exit | Display error message and exit |
| 3 | Test if dictionary file works | Be able to read the dictionary file | File able to be read |
| 4 | Tests inputs for word ladder if they work | Progress to get results of the word ladder | Progresses correctly |
| 5 | Tests inputs for word ladder if they don’t work | Display error message | Displays error message |

**6.0 Requirement Acceptance Tests**

Testing was completed, to ensure there are no bugs and that the game runs accordingly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Software**  **Requirement No** | **Test** | **Implemented (Full /Partial/ None)** | **Test Results (Pass/ Fail)** | **Comments (for partial implementation or failed test results)** |
| 1 | Accepts and opens dictionary.txt file | Full | Pass |  |
| 2 | Program doesn’t allow user to compare words of a different length | Full | Pass |  |
| 3 | Program allows users to remove words from the list | Partial | Pass | Allows to enter the word, but says to enter valid word |
| 4 | Program goes from hide to seek in 6 steps | Full | Pass |  |
| 5 | Users has the choice of asking for the shortest path | Full | Pass |  |
| 6 | Users has the choice of asking for the longest path | Full | Pass |  |
| 7 | User cannot remove a start or target word | Full | Pass |  |

**7.0 User Instructions**

The following instructions are for the user to be able to operate the word guessing game:

1. Unzip Python.7z file
2. Input dictionary.txt file into PyCharm
3. Open word\_ladder(1).py file from the unzipped file in PyCharm
4. Run the PyCharm file by pressing button in PyCharm
5. If green arrow cannot be selected, Pycharm 3.6 interpreter must be selected, then the green arrow can be selected
6. Enter the dictionary name
7. Enter a start word
8. Enter a target word of the same length as the start word
9. Enter whether you wish to go through the shortest or longest path
10. Enters words you wish to exclude from this process
11. Get results
12. Exit program