**Analysis & Design, Implementation,**

**and Test Document**

|  |  |
| --- | --- |
| **과 목** | **휴먼 ICT 소프트웨어 공학** |
| **10조**  **구성원** | **20123599 이동직**  **20123421 이현민**  **20146215 지무근**  **20121067 한재현**  **20164891 한윤진** |

[**1. Analysis & Design 3**](#_Toc516243814)

**1.1 Domain model**

[**2. Implementation 4**](#_Toc516243815)

**2.1 Software Architecture**

**2.1.1 MVC Pattern**

**2.1.2 how MVC concept was applied to simple merge**

**2.2 Design Model**

**2.2.1 Class diagrams**

**2.2.2 Sequence diagrams**

**2.2.3 State chart**

**2.3 how program was designed to be testable by Unit-test tools**

**2.4 object-oriented design principles**

**2.4.1 reusability**

**2.4.2 Encapsulation**

**2.4.3 Inheritance and polymorphism**

[**3. Usage of program and the screen shots of examples 11**](#_Toc516243816)

[**4. Functional unit test cases and their results 12**](#_Toc516243817)

**4.1 FileManager Test**

**4.2 DiffManager Test**

**4.3 DiffLine Test**

**4.4 LCS Test**

[**5. System test cases and their results 13**](#_Toc516243818)

**1. Analysis & Design**

**1.1 Domain model (utilize class diagrams)**

Situation :

There is a manager who manages the revision of Wikipedia. The manager wants a program that looks for the original parts of the Wikipedia and new parts of the revision. The manager can decide whether to keep the original document or to change the contents of the new document, and if the user presses a specific button, the content changes automatically.

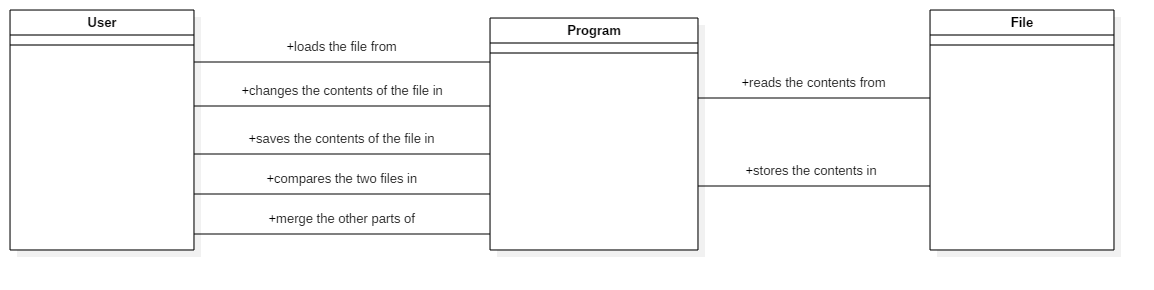


그림 1 class diagram(domain model)

**2. Implementation**

**2.1 Software Architecture**

**2.1.1 MVC Model**

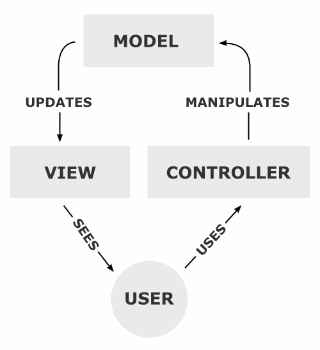


그림 2 MVC model

The MVC model is a development model that divides the application into three roles: Model, View, and Controller. By applying the MVC model, developers can create an application that can separate business logic from the user interface and easily maintain it without affecting each other.

When a user manipulates the controller, the Controller commands the Model. Model can save or load data. The Model manages the files and compares the contents of the files. Based on information of model, View displays visual information to the user.

**2.1.2 how MVC concept was applied to simple merge**

* Model
* There are FileManager class and Diff class as model. The FileManager reads the information of the loaded file. Then, according to the operation of the user, information is sent to Diff to obtain the same or different character blocks. Diff uses the LCS algorithm to store the information needed for text highlighting in a DiffBlock. In addition, it receives and stores the modified file.
* Controller
* There is a Controller class that acts as a controller. The user interface of simpleMerge has Load, Edit, Save, Compare, and Copy buttons.
* When the load and the save buttons are pressed, they ask the model to load or save files. User can modify text when it is true, and it is impossible to fix it when it is false.
* When the compare button is pressed, the LCS algorithm-based comparison is requested to the model.
* View
* Initially, the user is given two panels to load the documents to compare. When the user loads the file by pressing the Load button given on the Panel, the contents of the file are displayed as ListView on the corresponding panel. If the Edit button is pressed, it can be converted from ListView(not editable) form to Textarea(editable) form via Controller and show it to the user again.

**2.2 Design Model**

**2.2.1 Class diagrams**

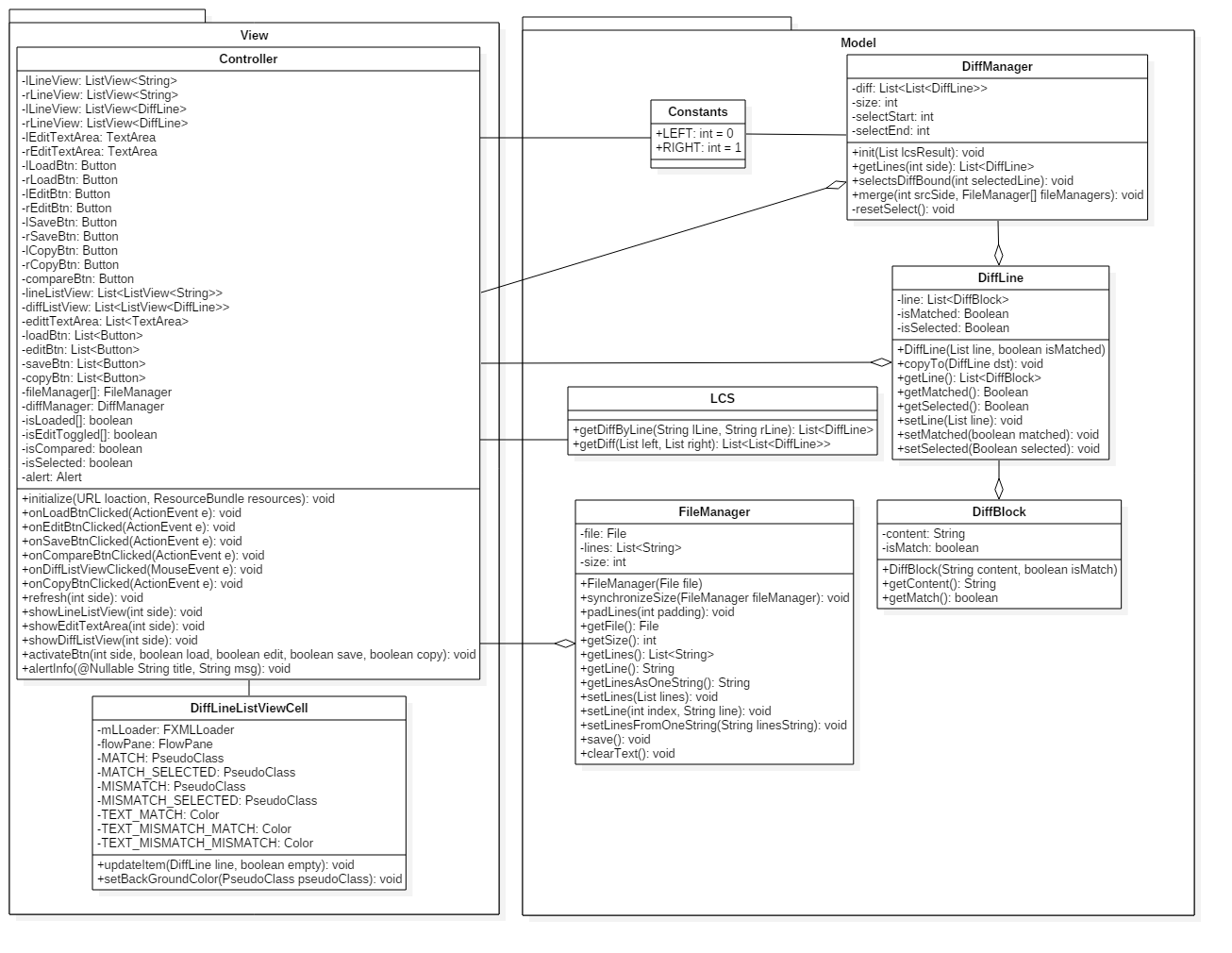


그림 3 Class diagrams

**2.2.2 Sequence diagrams**

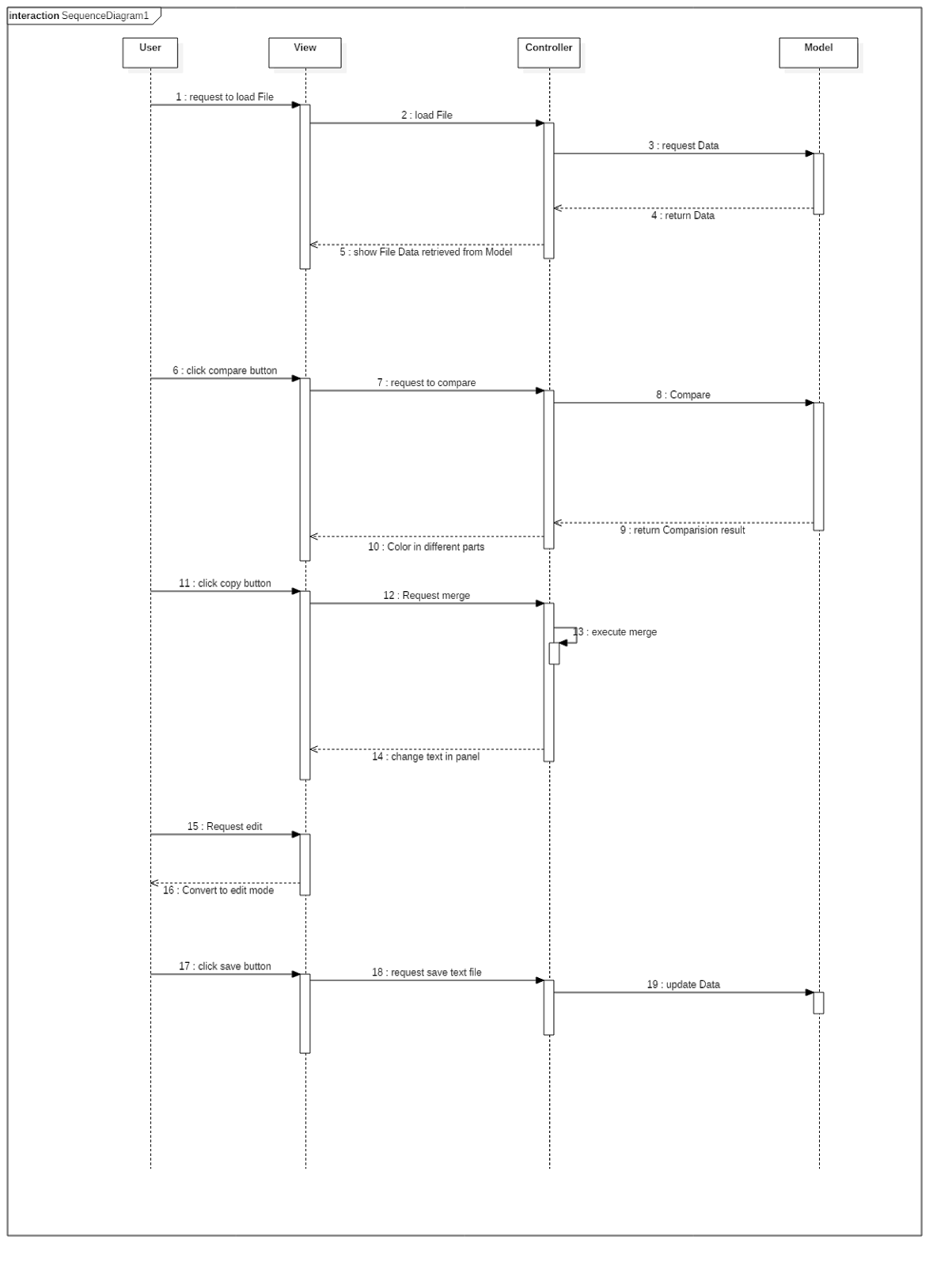


그림 4 sequence diagrams

**2.2.3 State chart**

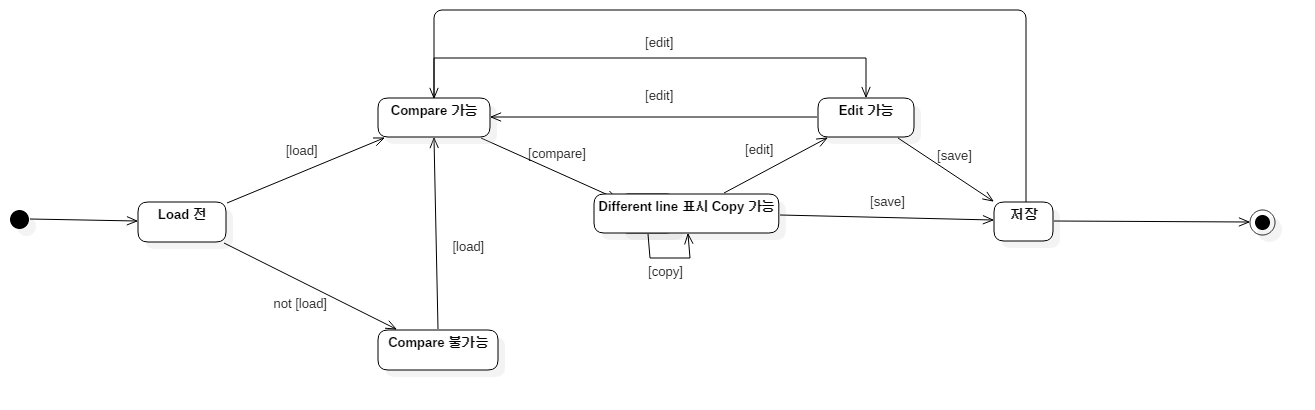


그림 5 state chart

**2.3 how program was designed to be testable by Unit-test tools**

 Since the program was divided into three parts, Model, View, and Controller, it was not difficult to test each function. In the Model section, each data can be checked to see if it works well for the method. In the Controller section, it was confirmed that the functions of the GUI work well.

**2.4 object-oriented design principles**

**2.4.1 Reusability**

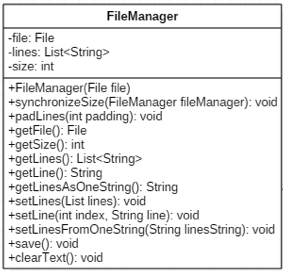


그림 6 FileManager Class

Initially, methods such as saving and loading the left and right files were made separately, but it was expected that the code would be duplicated. By creating the FileManager class, two objects are declared as instance variables in the controller. The roles are the same, only the files they have are different. Therefore, FileManager was created as a class. Reusability is improved.

**2.4.2 Encapsulation**

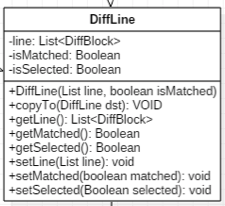
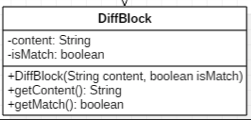


그림 7 DiffLine & DiffBlock Class

Diffline is a class that has an arrayList of java as a member variable. In order to show the results of comparing the line-by-line text of the loaded text, member variables were added to the state of the match and select, including the arrayList.

Members of DiffLine: line, isMatched, isSelected are encapsulated, so unauthorized parties are not accessible. The getter methods such as getLine(), getMatched (), and getSelected () should be used to retrieve the data. The setter methods such as setLine (), setMatched (), and setSelected () should be used when initializing or changing data.

**2.4.3 Polymorphism and inheritance**



Javafx was used in the project. To use javafx, an initializable interface is implemented in the controller class. Also initialize method has been overridden and modified for the project.

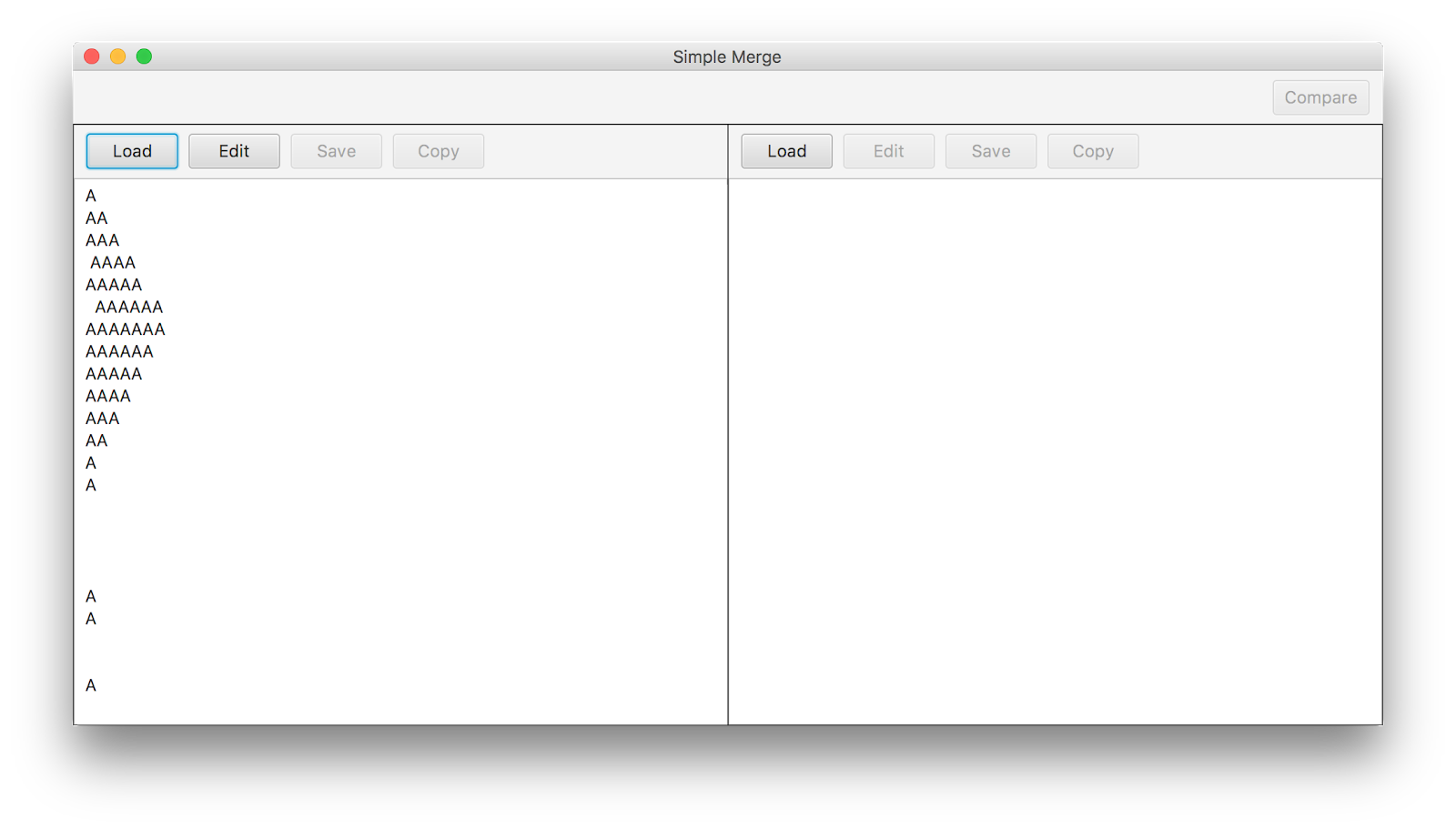


The main class also inherited the javafx application for the same reason, and the start method was overridden for the project.

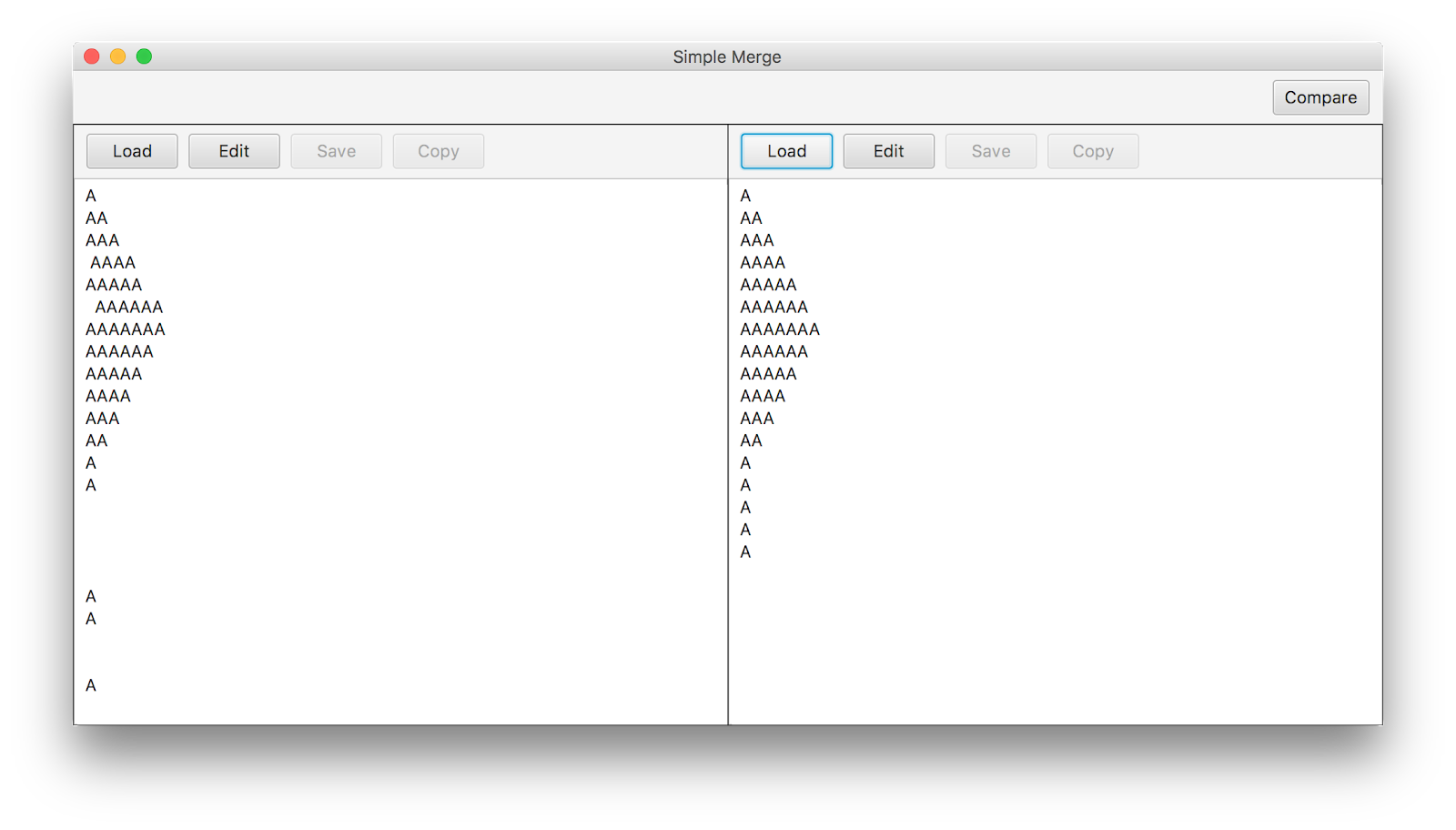


Diffline is a class created using arrayList, which is briefly described in 2.4.2. In this class, a listcell was used to make it visually easy to see. The inherited DiffLineListVewCell was created and modified for the project.

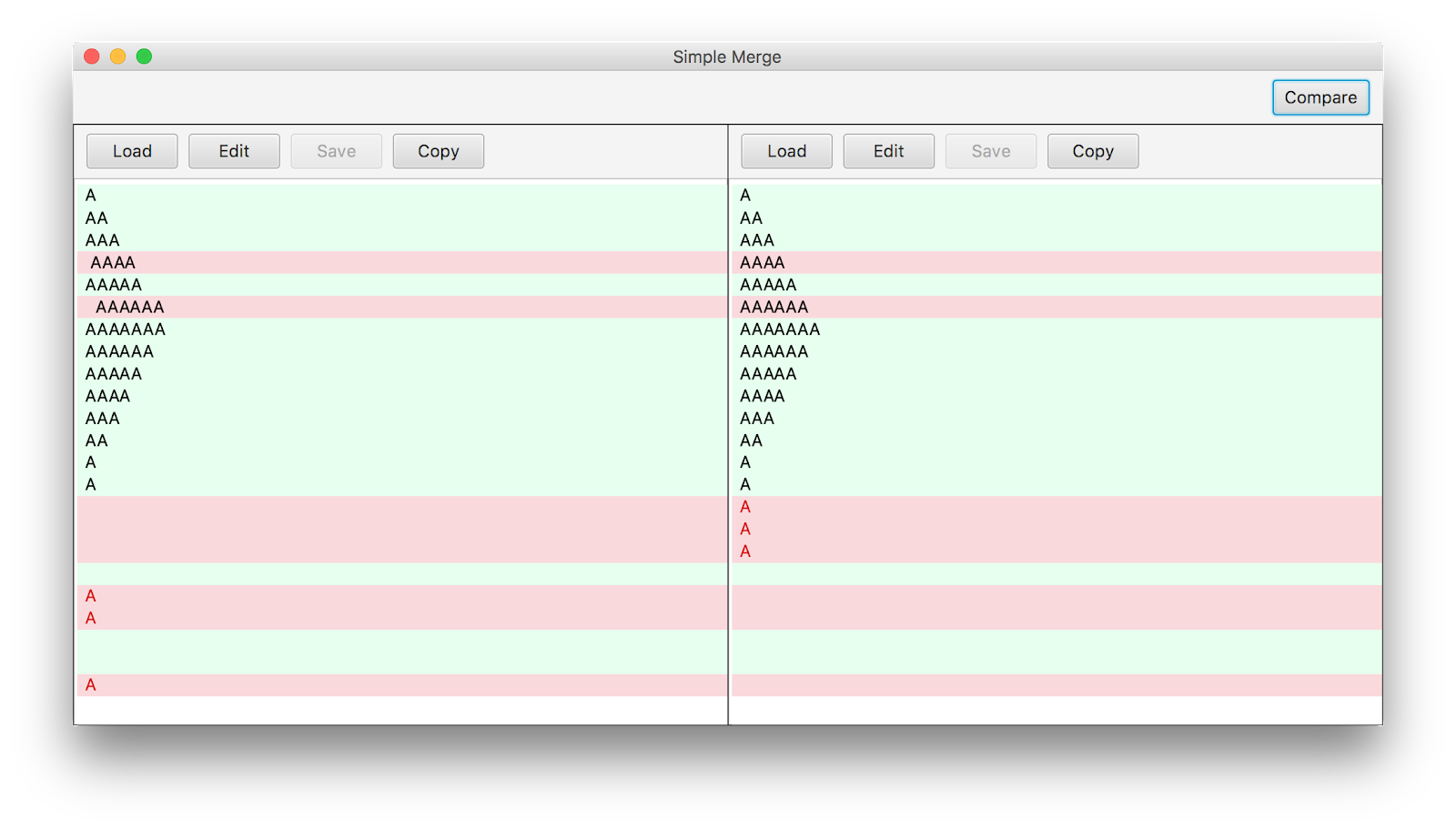
**3. Usage of program and the screen shots of examples**



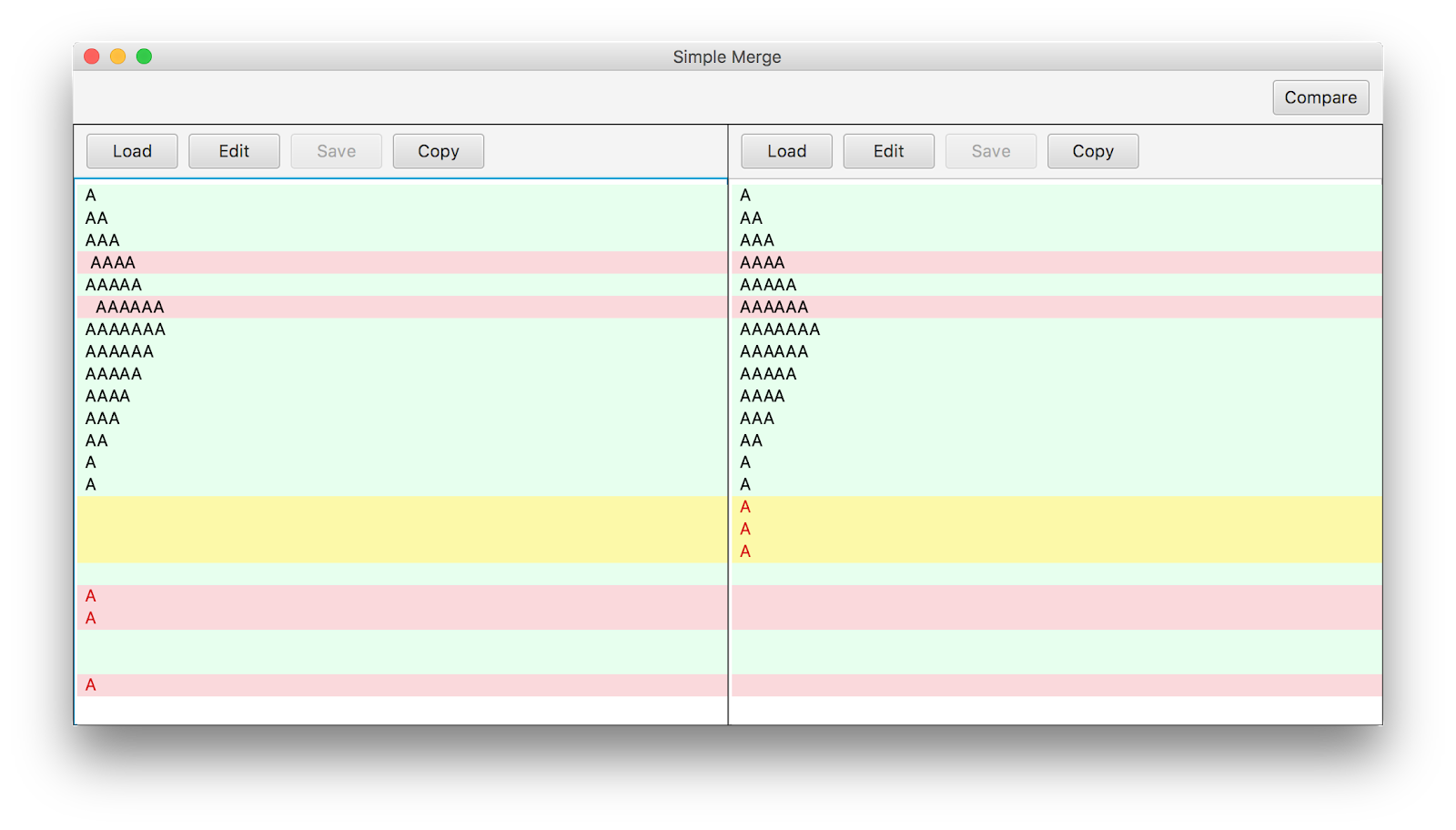
1. Load the left file.



1. Load the right file



1. Click the compare button

****

4. Continue copying to the side a user wants to change.

**4. Functional unit test cases and their results**

**4.1 FileManager Test**

A file manager is a class that loads a file from a local file and manages the state and contents of the file. Testing has been done to see if the files are changing as expected.

* File length synchronization test
* It has been confirmed that the lengths of the two files are changed to the same value to compare two files.
* Get lines as one string test
* It has been confirmed that multiple lines of text can be made into a single String to make the comparison easier.
* Set lines as one string test
* In contrast to the above method, this method is used to return a multi-line List after the operation is performed.
* It has been determined that the size of the List was the same as the expected and the contents were the same, in order to see if the String was successfully passed to the List of instances of the FileManager.
* Copy to another fileManager
* This method copies the contents to the other fileManager. After copying, it is confirmed that the contents of files of two fileManagers are the same.

**4.2 DiffManager Test**

This class creates a set of blocks with the same match status based on the selected block, and performs the merge in the direction of the panel the user wants.

* Selected bound check and merge test
* Based on the index of the selected block, it has been confirmed that the surrounding blocks are exactly bound to the bounds and the exact values are matched. and it has been determined that it is exactly changed to the same content after the merge.

**4.3 DiffLine Test**

DiffLine is a class that takes a list of DiffBlocks determined by the LCS algorithm and manages them by line. It is also responsible for receiving other DiffLines and copying them to the same contents in the DiffLine. Therefore, it was necessary to test whether the copy of DiffLine was done correctly in many cases. There are four major cases that can occur when copying, and the test was performed. The details are as follows.

* empty DiffLine to empty DiffLine copy test
* empty DiffLine to not empty DiffLine copy test
* not empty DiffLine to empty DiffLine copy test
* not empty DiffLine to not empty DiffLine copy test

The test was expected in all cases. It has been confirmed that the functionality of DiffLine is performing as intended.

**4.4 LCS Test**

The LCS is a class that uses the LCS algorithm to find a different part between two strings and divide it into DiffBlocks to create a list. This class should be able to find other parts exactly and contain information that the parts are different. Therefore, a test was conducted to verify that this information is correct.

* Get diff by line test
* It has been confirmed that the diff has been performed correctly on a line-by-line basis and the predicted block has appeared.
* It has been determined that diff works correctly when the two strings are equal or different from each other.
* When the contents of the String used in the test are the same, the size of the diffBlock is expected to be 1, and a value of 1 was actually returned. On the other hand, if the strings are different from each other, the expected value is obtained by testing with assertEquals () with the expected diff block.
* Get diff test
* Diff has been successfully performed on a list of Strings.
* It has been confirmed that the result is correct after checking the size of the List

**5. System test cases and their results**

System test case’s sequence is as follows.

* Press left load button.
  + Click on the Load button on the left to see what is not loading first.
* Turns off the file selection window without doing anything.
  + Exit without selecting a file in the file selection window.
* Load one file on the left.
  + Press the Load button once again to load the file.
* Edit the loaded text file on the left.
  + Edit the selected file on the left with the text area.
* Save the edited text file on the left.
  + Save the edited file.
* Load another file to the right.
  + Load another file to the right to compare with the file on the left.
* Compare the two files.
  + Compare the two files loaded on both sides.
* Find another part on the left and pick it.
  + With the comparison completed, select another part of the two files.
* Merge from left to the right.
  + Merge the selected parts from left to the right.
* Save the file on the left.
  + Save the loaded file to the left.

This sequence has been tested with unit tests completed.

It was successful to take two similar files and merge them into one as the same file.