<TalkBox>

<TALKBOX>

TEST PLAN

Version <1.0>

<02/24/2019>

VERSION HISTORY

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Seohyun Jeong	<20/02/19>	Andrew Persaud	<21/02/19>	Test Plan for midterm submission.

UP Template Version: 12/31/07

TABLE OF CONTENTS

1. INTRODUCTION

⬙ 1.1 PURPOSE OF THE TEST PLAN DOCUMENT

the purpose of this document is to test cases and let other developers trace the possible bug. In addition, this document will discuss the unit test and GUI test
and the test coverage.

2. UNIT TESTING

⬙ 2.1 TEST RISKS/ ISSUE

This test is based on Junit test. each important class for our project is tested under Junit. Every class is intended to test, However, some GUI methods are tested manually and some backup classes are ignored.

⬙ 2.2 ITEMS TO BE TESTED

Test Case	Short Description	Test Class
TalkBoxTest	Using Talkbox class, to test talkbox instance operation.	TalkBox
ControllerTest	Test each controller method to see that it does modify the actual model/view	Controller
MainFrameTest	Test every component on the main configurator window, then check coverage to see which methods weren't covered	MainFrame
MainFrameSimTest	Launch the simulator in coverage mode, then use every component to see which parts weren't used	MainFrameSim
GuiTest	Manual tests to cover the most frequent events a user would trigger.	MainFrame Record AudioSelectionPa -nel

⬙ 2.3 TEST APPROACH

A tester class was used to automatically check certain components of each class. This was much more useful for the model and view classes; however, may aspects of the view classes had to be tested manually.

- Overall coverage is 92%.

See (Tester.java) for exact junit tests.

TalkBoxTest

Tested each component of the model, by creating some lists and arrays and comparing them to those generated by the model (TalkBox).

Junit Test	implementation
getAudioList	Identical audio list generated, then compared to the audio list stored in Talkbox
getNumberOfAudioButtons	Number of buttons should be 5 for default audio sets
setAudioFilenames	Added a test audio set containing a unique audio file, and checked to see whether it existed in the model afterwards
AddIcon	Added a test icon set containing a unique icon path, and checked to see whether it it existed in the model afterwards

- 4/4 junit test runs
- Coverage: 88.5%

ControllerTest

- Tested the important methods in the control class

Junit Test	Implementation
addAudioSet	Added an audio set, then compared the number of audio sets before and after. Then confirm whether the audioList exists at the last index of the database.
addAudio	Added “testFile” to Audio Set 1, then checked to see whether it contained “testFile”.
removeAudio	Removed “hi.wav” from Audio Set 1, then checked to see whether it existed afterwards.
getNumberOfButtons	Number of buttons should equal 5
RemoveIcon	Removed “happy.png” from IconList 1, then checked to see whether it existed afterwards.

- 5/5 Junit test runs
- Coverage: 71.4%

MainFrameTest

JUNIT Test	Implementation
testGUI	Open menu, select audio set, pressed start button, play sound file, press swap, then check to see whether each button was pressed

- 1/1 Junit test runs

- Coverage: 94.8%

MainframeSimTest

- Tested every GUI component by pressing all play buttons, swap, and exit
- Coverage: 97.6% (note: two methods were not used, as they are only used in JUNIT testing)

3. GUI TESTING

⬙ 3.1 TEST RISKS/ ISSUE

Some elements of the application were unable to be tested. For instance, many events that require real time user input are difficult to replicate. However, to supplement this we performed manual testing to observe any potential flaws.

GUI Test_MainFrame,AudioSelectionPanel and RecordDialog

Junit Test	Implementation
setupGUI	Instantiate the mainframe, then checked to see whether it shows the Configurator window.
CustomCheckboxtest	Selected the checkbox for the custom audio set, then checked to see whether it works fine using isSelected() method and seeing whether the appropriate buttons are enabled.
SelectButton	Selected the checkbox for the custom audio set, then checked to see whether the setbutton's button text dynamically changes to the currently selected audio file in the audio file list.
SearchTextField	When the string "cool" is entered into the audio search field, the audio list should dynamically transform into a new audio list consisting of audio files with names containing "cool".
AudioNameTextField	Set String "emotion" to the audio set name field, then checked to see whether it sets the new audio set name to "emotion".
RecordTextField	Set string "Thanks" to the record set name field, then checked to see whether it sets the new audio file name to "Thanks.wav".

- Coverage: 85.8%
-

⬙ 3.2 ITEMS TO BE TESTED

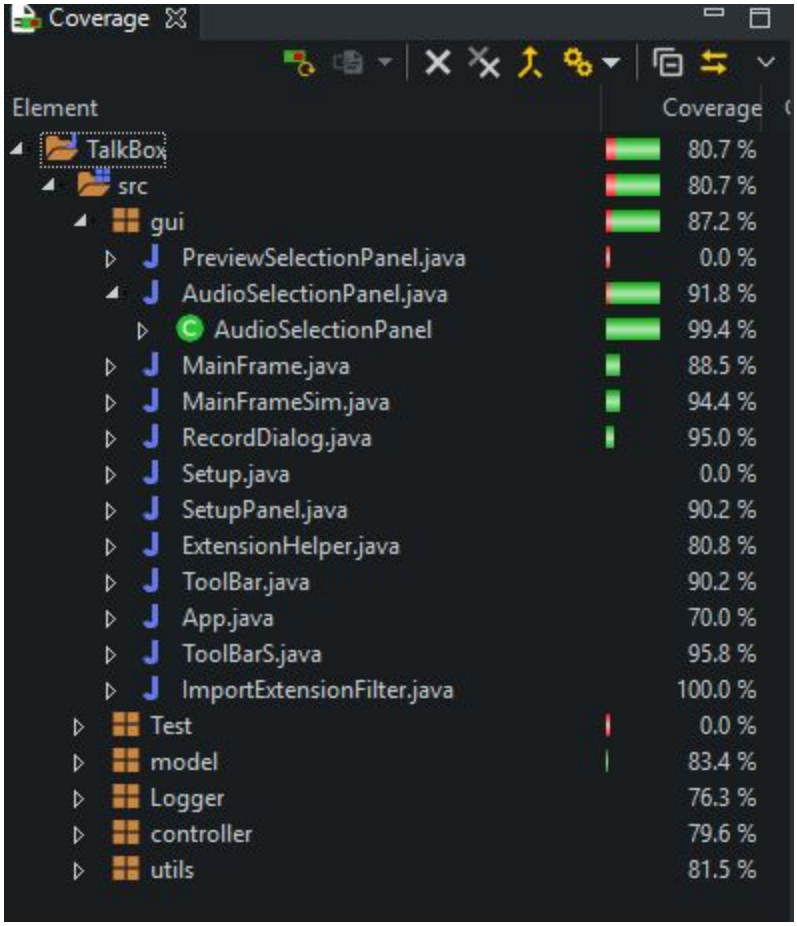
Test Case	Short Description	Status
(3.2.2) Exit versus Window Close	User is not able to exit through the app, without terminating the program	Resolved
(3.2.3) Custom Audio Set: uncheck	When unchecked, the buttons that are only used to make custom sets do not grey out, but remain useable	Resolved
(3.2.4) Exit from Simulator	File > Exit should return the user back to the Configurator; however, it just terminated the program	Resolved
(3.2.5) Audio set length	Having an audio set that has too many elements pushes some of the buttons out of view	Resolved
(3.2.6) Record functionality	When the application is converted to an executable jar, the record functionality fails.	Resolved

⬙ 3.3 TEST APPROACH

To resolve these bugs, we must first launch the program in eclipse using coverage mode. Then, we perform the actions that cause these bugs, and then confirm where in the code the error occurred.

4. COVERAGE

⬙ 4.1 Coverage(screenshot of Coverage)



⬙ 4.2 Conclusion

Both the model and the GUI see over 80% coverage.

The missing instructions in the model pertain to getters that retrieve the number of buttons and a specific audio list from the model. These are not used, as the methods that would use those arguments are not implemented in this particular version, and will be implemented for the final version.

The lowest coverage seems to be in Controller; however, the missing instructions are related to the missing coverage in model. The getters and setters in model are actually used by controller, so the missing instructions in model result in missing instructions in controller.