Braille Box Authoring App

Requirements Document

EECS 2311: Software Engineering Project

Team 9

May 19, 2018

Authors

Jeremy Winkler (214 915 854)

Nisha Sharma (213 251 830)

Tyler Thomson (215 081 904)

Revision Level	Date	Pages affected	Comments
1.0.0.0	May 19, 2018	ALL	Final Submission
1.0.0	February 23, 2018	1-6	Midterm Submission
1.0	February 05, 2018	1	Initial draft

Table of Contents

1.0 Introduction:

For the midterm submission, our group has created classes to enable the user to save, create, and edit files. As we created these classes, we created the corresponding test cases to make sure our program works as intended. However, our group has not created any test cases for the GUIs as we believe that our manual testing is sufficient. The classes given to us are assumed to be working and so we shall not provide test cases for these.

2.0 Data Classes:

2.1 Card:

The card class is for storing each question along with it's corresponding data. Most of the methods in card are very simple as most of them don't check any conditions and just set the data to the inputted value.

Card(int id, String name, String type) (constructor)

The card constructor was tested to make sure that each field of the card class was initialized correctly. This includes the id, name, type, lists, and text. This is also testing all of the get methods as we used these methods to receive the info.

setText(String text)

This method is used for replacing the text the card contains with the new text given. As it is a basic method which just sets the text to the given text, we had a few different inputs, along with empty and null strings, just to make sure it works correctly.

addText(String text)

The addText method is for if we want to add text onto the current text. We have inputs for checking if it adds text correctly if there is text already stored, if the text stored is null and if we try to add both null and empty strings.

setSound(String newSound)

This method is used for setting the sound of the card if the user wants to play a sound in the prompt. We tested inputting basic strings as well as null and empty strings and all of them worked as intended.

setBList(ArrayList<DataButton> newList)

Used to set the card's list of buttons to the given list. It does not make a copy of the list, it just switches it's reference to the new list. In this test we checked that changing the list after does affect the card's list and made sure it switches to a new list properly.

setCells(ArrayList<BrailleCell> newList)

setCells is very similar to the setBList method as it takes in a new list and sets the card's reference to the new list. The testing we did was also similar with testing a new list, updating that list, and switching to a new list

setName(String s)

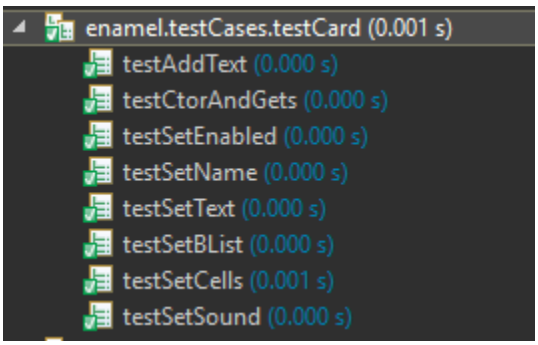
This method for card just sets the new name to the passed in argument. This method was tested with basic, null, and empty strings to make sure it worked as intended.

setEnabled(Boolean enabled)

This method was tested to switch whether the buttons were enabled or not for the card. It was initially set to false, changed to true, and changed back.

Card has 100% test coverage. The testing done is sufficient as most of these methods are simple setter methods which just set the current value to the given value. So as long as we test these with a few inputs and the slightly more complicated methods more thoroughly, like we did, then these test cases are enough.

Card.java	100.0 %	98	0	98
Card	100.0 %	98	0	98
setText(String)	100.0 %	4	0	4
setSound(String)	100.0 %	4	0	4
setName(String)	100.0 %	4	0	4
setEnabled(Boolean)	100.0 %	4	0	4
setCells(ArrayList<BrailleCell>	100.0 %	4	0	4
setBList(ArrayList<DataButton	100.0 %	4	0	4
getType()	100.0 %	3	0	3
getText()	100.0 %	3	0	3
getSound()	100.0 %	3	0	3
getName()	100.0 %	3	0	3
getId()	100.0 %	3	0	3
getEnbled()	100.0 %	3	0	3
getCells()	100.0 %	3	0	3
getButtonList()	100.0 %	3	0	3
addText(String)	100.0 %	22	0	22
Card(int, String, String, Boolea	100.0 %	28	0	28



All tests passed with expected results

2.2 DataButton:

This class is similar to card as most of the methods are just basic getters and setters. This class does however have a copy constructor so we had to test that as well

DataButton(int id)

This constructor just sets the id to the passed in id and the strings to empty. The tester for this constructor also tests the getter methods as that is how it receives the data.

DataButton(DataButton other)

This is a copy constructor which creates a copy of the other DataButton. As the fields of this class are int and String, then it is fine that we are just setting each field to the other DataButton's value.

getID()

This method returns the id and was tested with a few different DataButtons as there is no setter for this field.

getText()

Returns the button's text and was tested with adding different text and checking the returned string.

getAudio()

Returns the button's audio file and was tested with setting the audio to different strings. A few random strings were passed in using setAudio just to make sure it works.

addText(String newText)

This method adds on text to what was already stored in the button. A few different strings were passed in to make sure it works consistently.

setAudio(String audioPath)

This method sets the audio to the passed in string. It doesn't check whether the string is a valid audio file, so we just passed in a few strings and compared them to what was returned.

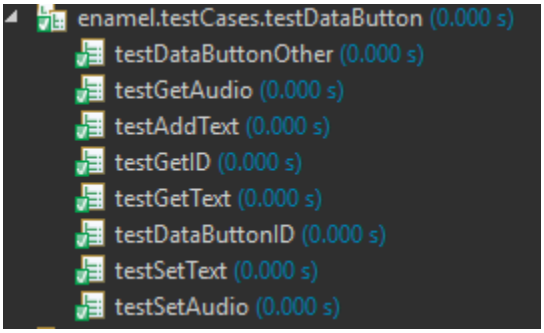
setText(String newText)

Sets the text to the passed in value. Once again, a few random strings were passed in just to make sure it works.

DataButton has 93.1% test coverage. Since most methods are very basic methods which just sets the value stored to the given value, not much unexpected can happen so as long as we test with a few test inputs it is good enough. For the one conditional method both conditions were met so this is good as well.

DataButton.java	93.1 %	94	7	101
DataButton	93.1 %	94	7	101
setText(String)	100.0 %	4	0	4
setCells(ArrayList<BrailleCell>)	0.0 %	0	4	4
setAudio(String)	100.0 %	4	0	4
getText()	100.0 %	3	0	3
getID()	100.0 %	3	0	3
getCells()	0.0 %	0	3	3
getAudio()	100.0 %	3	0	3
addText(String)	100.0 %	24	0	24
DataButton(int)	100.0 %	27	0	27
DataButton(DataButton)	100.0 %	26	0	26

The methods regarding cells was not tested in this class because these methods are very simple to test manually, as such we decided to not write tests for these.



All tests passed with expected results

3.0 Parsers

3.1 FileToCardParser (FTCP)

This parser is for reading a given text file and storing the data into a list of cards.

FileToCardParser()

The constructor was very simple as it was just initialized the card list, initial prompt, and ending prompt. Using the get cards, initial, and ending methods we checked if these fields were initialized.

checkNumLines(String scenarioFile)

Checks the number of lines in the file a. The number of lines is for when we reach the end of the file to add the last card onto the list. The tester for this method only checks whether the number of lines is correct. Two blocks of text were checked as well as an empty and null block of text which all returned the right value.

checkButtonsAndCells()

This method reads the first two lines and checks if they match “Cell #” and “Button #”. This method throws an exception if it doesn’t match. We tested with a basic input and made sure the exception was thrown for empty, null, and not matching inputs.

The rest of this section will go based off of the tester's methods instead of class's as the rest of the parsing is done in one method and was tested separately to make it simpler.

testGetCards()

This method is for ckecking whether the general parser works overall. This is currently checking different parts of both scenario 1 and 2 to make sure the card text, button text, and cell states are all working as intended.

testCommands()

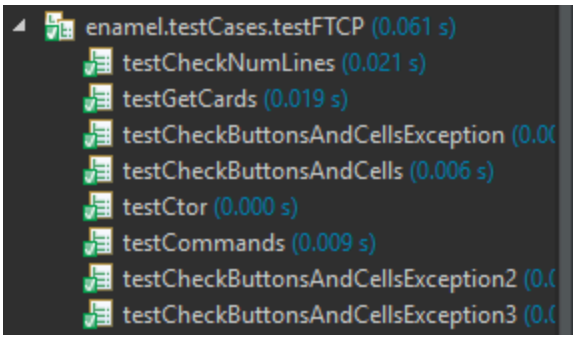
This test if to test all the commands not tested by testGetCards(). This includes pause, display character, and display string. Each of these were tested both inside and outside of responses to make sure it works everywhere.

print() (in FTCP)

This method is just a method we used for manual testing purposes to make sure that the data was being displayed correctly and is not part of the actual program, so we believe no testing of this is necessary.

FileToCardParser only has 90.6% coverage, however, most of this is specific conditions and exception handling. The testing for this class is enough as all of the common functionality is thoroughly tested. Also, a good indication is that this class works for the given scenarios as explained in 3.3.

FileToCardsParser.java	90.6 %	1,120	116	1,236
FileToCardsParser	90.6 %	1,120	116	1,236
setUp()	100.0 %	63	0	63
setFile(String)	89.2 %	33	4	37
print()	100.0 %	129	0	129
parse()	94.9 %	93	5	98
nextCard()	100.0 %	65	0	65
getNumLines()	100.0 %	3	0	3
getCells()	100.0 %	3	0	3
getCards()	100.0 %	3	0	3
getButtons()	100.0 %	3	0	3
dispCellPins()	80.8 %	80	19	99
checkNumLines(String)	91.1 %	41	4	45
checkCommands()	83.6 %	367	72	439
checkButtonsAndCells()	87.2 %	82	12	94
checkButtons()	100.0 %	141	0	141
FileToCardsParser()	100.0 %	14	0	14



All tests passed with expected results

3.2 CardsToFileParser (CTFP)

CardsToFileParser(ArrayList<Card> cards, int numButtons, int numCells, String initialPrompt, String endingPrompt)

The constructor was for setting the cards, number of buttons, number of cells, as well as initial and ending prompts. Since the only get method this class has is getText(), we were only able to test if the text was empty as it should be. The other parts of this constructor are essentially tested within the other tests.

The rest of this section will go based off of the tester's methods instead of class's as the rest of the parsing is done in one method and was tested separately to make it simpler.

testCellButton ()

This method is to test whether the file correctly has the right number of cells and buttons and has the correct initial and ending prompts. We tested the CTFP with an empty card list and basic inputs for the rest to make sure these sections were being converted to a string correctly.

testWriteButtons()

This method is for testing to make sure that the buttons and all of it's data is being displayed correctly. First, we tested with a basic input, then input with a sound file, and finally testing if the display pin indicator properly gets converted to the corresponding text. Overall this tests all functionality of the buttons.

testCells()

This method tests whether the cells are properly converted into the text format. This method tests with 2/3 cells, with a new cell, and with a null cell.

testPrompt()

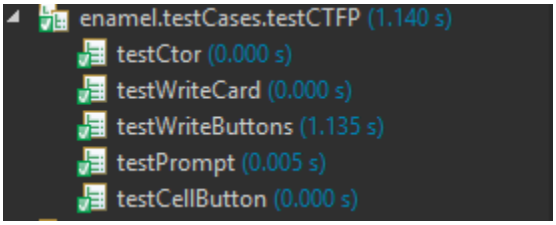
This method is for testing whether each card's prompt is properly being displayed. It is tested with null, empty, and basic prompts to make sure it works as intended.

testWriteCard()

This method is an overall tester for the CTFP class as it tests a combination of card prompt, cells, and buttons. For this we only have 1 test as each functionality was already tested separately.

CardsToFileParser has 97.8% coverage, but what's missing is either conditional or one for loop which gets tested when parsing the loaded files as explained below. Once again the testing for this class is good as the basic functionality is properly tested and this class works for the given scenarios as explained in 3.3.

CardsToFileParser.java	97.8 %	658	15	673
CardsToFileParser	97.8 %	658	15	673
writeTextAndCheckCells(Card,	95.5 %	317	15	332
writeInput(String, ArrayList<D	100.0 %	93	0	93
writeCard(Card)	100.0 %	34	0	34
writeButtons(Card, String)	100.0 %	105	0	105
getText()	100.0 %	3	0	3
createBody()	100.0 %	85	0	85
CardsToFileParser(ArrayList<C	100.0 %	21	0	21



All tests passed with expected results

3.3 Overall Parsers

This section is just to emphasize that a good indicator for us that the parsers were working as intended was that if you load any of the given scenarios using FTCP and then save that scenario using CTFP, it creates a file which looks quite similar to the original and works the same.

4.0 File Creator

4.1 ScenarioWriter

The scenario writer class is what we used to write to a file after we created the string representation using CTFP.

ScenarioWriter(String filePath)

Constructor for the scenario writer which sets the path to filePath. This constructor is used if we want to create a file. Our test case has a basic input which uses write() to create the file then checks if the created file has the same input.

ScenarioWriter(String filePath, Boolean appendVal)

This constructor is used if we want to add text onto a previously created file. Once again we have a basic input using write() and then we check to make sure it added the text onto the end.

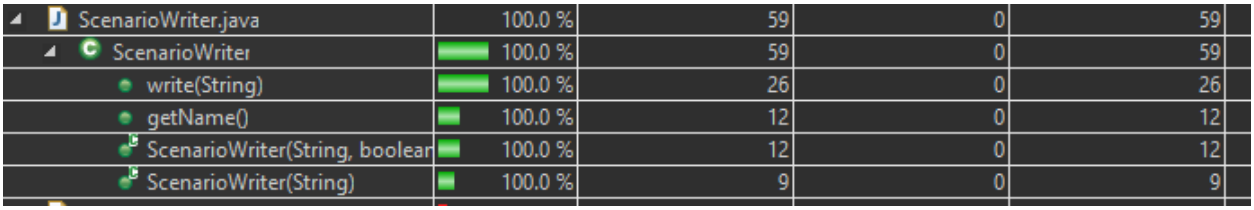
write(String text)

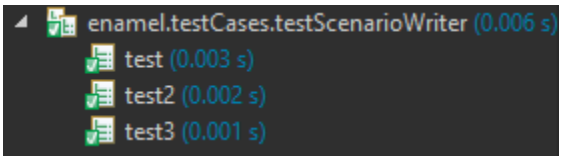
Used to write the text to either the end or overwrite the text in a file depending on which constructor was used to instantiate the object. The tests we did for the constructors both used this method to write to the file and so this was tested in the previous tests.

getName()

Parses the path and returns the name of the file. We tested this with a basic input with a path and without a path and both of them worked.

ScenarioWriter has 100% test coverage. This testing is good enough because the functionality of everything was tested and this was used within other testing so we know it works.





All tests passed with expected results

5.0 GUIs

Our GUIs include Initial View, Scenario Form, Authoring Viewer, and Recorder Frame. All of our GUIs are only manually tested. After a quick manual test we can achieve 97.7%, 97.7%, 92.4%, and 84.7% code coverage, however we are constantly testing our GUIs and often these tests cover the scenarios which were not covered in this quick test. Our application has been tested to successfully run with Mac screen reader, NVDA screen reader for Windows, and ORCA screen reader for Linux. If you encounter any problems, please contact the developers.

6.0 Acceptance Test Cases

An important part of testing is to make sure the user can accomplish what they want. For this we created acceptance test cases to verify different scenarios. The test case ID's correspond to the ones in the requirements document.

Test Case ID	AT0	**Test Case Title**	Application Launch		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application launches successfully on three operating systems (Windows, Linux and Mac) or not				
Tester's Name	Team 9	**Date Tested**	May 10, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	Prerequisites:		S #	Test Data Requirement
1	App Successfully Installed		1	None
2			2	
3			3	
4			4	

Test Conditions:	The application should be installed on the operating system under test

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Double Click on the Authoring App icon	The App will launch and display the user interface with four options to create a New scenario, Edit or Test a scenario and Exit the app.	As expected	Pass

Test Case ID	AT1	**Test Case Title**	Edit Scenario		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1

Test Case Purpose:	Verify whether the application allows the user to edit a previously created file

Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Click Edit			
3	Select file to edit	Opens view with which contains previously added information	As expected	Pass

Test Case ID	AT2	**Test Case Title**	Add Audio		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1

Test Case Purpose:	Verify whether the application allows the user to add audio to the file				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Open App			
2	Open/make a scenario			
3	Click on “Insert Action” then “Play Audio File”	Opens file chooser to allow user to select audio file	As expected	Pass

Test Case ID	AT3	**Test Case Title**	Record Audio		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1

Test Case Purpose:	Verify whether the application allows the user to raise specific pins				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Open App			
2	Open/make a scenario			
3	Click on “Audio” then “Record”			
4	Click on “Record” then “Stop & Save” when done recording	This opens a file chooser, so user can save the file	As expected	Pass

Test Case ID	AT4	**Test Case Title**	Raise pins		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1

Test Case Purpose:	Verify whether the application allows the user to raise pins				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Click on cells user wants to raise			
4	Click on “Raise pins”	Adds text to prompt to indicate to user which pins it will raise	As expected	Pass

Test Case ID	AT5	**Test Case Title**	Buttons and Cells		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to select the number of buttons and cells from a finite amount				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Click on “New”			
3	Select number of cells and buttons from combo box			
4	Click “Create a Scenario”	Creates a view for user to create file which has selected number of cells and buttons	As expected	Pass

Test Case ID	AT6	**Test Case Title**	User Response		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to have a response				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Click on “Enable User Response”			
4	Write responses for each button	When saved, creates a file were user can respond with button clicks	As expected	Pass

Test Case ID	AT7	**Test Case Title**	Button Response		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to create a response for each button which includes audio and/or text				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Click on “Enable User Response”			

4	Write responses for each button	When saved, creates a file were user can respond with button clicks	As expected	Pass
5 (Extra)	Click on “Insert Action” then “Play Audio File”	When saved, creates a file which plays selected audio file	As expected	Pass

Test Case ID	AT8	**Test Case Title**	Rearrange Scenario		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to rearrange a scenario				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Open App			
2	Open/make a scenario with at least 2 cards			
3	Select card user wants to move			
4	Click “Card Up” or “Card Down” to move card up or down respectively	Moves selected card up or down in the scenario	As expected	Pass

Test Case ID	AT9	**Test Case Title**	Test Scenario		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to test a scenario				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Click “File” then “Save”			
4	Click “File” then “Test”			
5	Click “Yes”	Opens the program which runs the file	As expected	Pass

Test Case ID	AT10	**Test Case Title**	Insert Pause		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to insert a pause				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Click on “Pause”			
4	Enter the duration	Adds text to prompt to indicate to user how long it will pause for		

Test Case ID	AT11	**Test Case Title**	Display Character or String		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to display a character or string on the braille cell(s)				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Click on “Insert Action” then “Display Character” or “Display String”	Adds text to prompt to indicate to user which character to raise and on which cell or what string will be displayed	As expected	Pass

Test Case ID	AT12	**Test Case Title**	Reset or Lower Pins		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application allows the user to reset or lower pins				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Raise some pins to lower later			
4	Click on “Reset”	Adds text to prompt to indicate to user that all cells will be cleared	As expected	Pass
4 (Other)	Lower Pins you wish to lower then click “Raise Pins”	Adds text to prompt to indicate to user which pins to be raised and lowered	As expected	Pass

Test Case ID	AT13	**Test Case Title**	Keyboard Shortcuts		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1

Test Case Purpose:	Verify whether the application has keyboard shortcuts				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Use shortcuts	Does action according to shortcut used	As expected	Pass
3 (Extra)	If user doesn't know shortcuts, click "Help" then "User Manual"	Opens the user manual so users can check the list of keyboard shortcuts	As expected	Pass

Test Case ID	AT14	**Test Case Title**	Frequency Logging		
Created By	Team 9	**Reviewed By**	Team 9	**Version**	1
Test Case Purpose:	Verify whether the application can log user actions				
Tester's Name	Team 9	**Date Tested**	May 18, 2018	**Test Case (Pass/Fail/Not Executed)**	Pass

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Open App			
2	Open/make a scenario			
3	Press buttons	Logs actions user makes to console	As expected	Pass

6.1 Acceptance Test Cases Summary

Acceptance Test	Description	Success Criteria	Pass/Fail
AT0	The Authoring App launches successfully on all three specified operating systems (Windows, Linux and Mac)	The app launches successfully on all platforms displaying the user interface with four options to create a New scenario, Edit or Test a scenario and Exit the app.	Pass
AT1	The Authoring App allows users to edit scenario file. Rationale: User should be able to load a properly formatted scenario '.txt' file in the authoring app to edit different components associated with it.	The scenario file to be edited loads successfully and all the components of the scenario file can be edited.	Pass
AT2	Inserting audio files to a scenario. Rationale: User should be able to insert audio files of '.wav' format in the scenario. This audio would be played while running test simulator.	Audio file successfully inserted to the scenario file in proper format and the audio gets played correctly during the simulation testing.	Pass
AT3	Recording a new audio file using the Authoring App. Rationale: User should be able to record new audio files and save them in '.wav' format.	The app successfully launches user interface to record the audio. The audio files gets successfully saved in specified folder with specified name in .wav format on choosing to save.	Pass
AT4	Raising pins on the TBB simulator after specifying inside a scenario file created using Authoring App. Rationale: User should be able to raise pins particular pin on braille cell in TBB simulator.	The intended pins get raised during simulation testing.	Pass
AT5	Specify finite number of Cells and Buttons for a scenario during creation of scenario using the Authoring App.	Successfully set the number of Cells and Buttons to a finite number (as mentioned in the specifications).	Pass

	Rationale: User should be able to specify finite number of cells and buttons for a particular scenario on creation.		
AT6	Enable user response for a scenario. Rationale: User should be able to ask a question and receive a response in terms of button clicks.	User input successfully enabled.	Pass
AT7	Respond to a button click with an audio and text-to-speech feedback. Rationale: User should be able to associate a sound file and text feedback with each button click.	Audio file and text-to-speech component related to a button response gets successfully simulated on TBB simulator.	Pass
AT8	Rearrange different sections of a scenario with relative ease. Rationale: User should be able to rearrange the order of different question and response segments of scenario.	Different components can be successfully moved up or down during the creation or editing of a scenario file.	Pass
AT9	Test a saved scenario file created using the Authoring App. Rationale: User should be able to successfully test a saved scenario file with TBB simulator.	The TBB simulator successfully launches from within the authoring app and successfully runs the scenario file created using the Authoring App.	Pass
AT10	Insert Pause to the scenario.	The app allows to successfully insert pauses at different times within the scenario.	Pass
AT11	Display a character and or word on the braille cell.	The app allows user to successfully set a character or word to be displayed.	Pass
AT12	Reset pins or lower pins after some time.	The app allows users to successfully specify action to lower the pins on braille cell.	Pass
AT13	Use a keyboard shortcut to perform a common task.	The app allows user to easily use most commonly	Pass

	Rationale: For example use Ctrl+N to create a new file.	used features using a keyboard shortcut.	
AT14	Perform an action involving button click or choosing an option in the software and see how many times a feature was used. Rationale: Based on most frequently used features, design easy access keyboard shortcuts.	The app records a log of actions keeping a count of how many times it was accessed. The log is displayed either on terminal/console or saved as a logfile.	Pass