Treasure Box Braille

Testing Document

Contents

1. Test Cases Introduction

The following test cases have been selected to insure our application runs as intended. However, we do not fully depend on these tests cases because the application has components which depend and interact with each other. This means that in addition to these test cases, quality assurance by running the application and testing the features manually before launch is also key to a stable release. Not all the following test cases have been implemented at this time and further cases may be added as the need arises.

Controller

I. **test00_fields -** Tests that there are the correct number of fields in the class and that they are initialized to the right values.

II. **test01*_ctorController** – Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are labeled alphabetically in place of the * character.

III. **test02*_initializeList**– Tests the initialization of the list and navigation panel components. Tests that once this method is run, there are five JLabel components inside of view.navigationPanel. This test will also insure that the current list is in the first index.

IV. **test03*_updateLabels**– Tests that all five components of the navigationPanel are successfully updated with new JLabel texts. The highlight position determines what indexes are called for the component updates. Check that this index is within bounds, otherwise throw an exceptionOutOfBounds.

V. **test04_nextButton**- Tests that the nextButton method correctly calls updateLabels ONLY when the highlight position is at its lowest position and the list is longer than what is currently being displayed. Check that when the current keyPhrase is “#JUNCTION” that the chooseButton() method is called.

VI. **test05*_prevButton**- Tests are similar to the nextButton. Test that this method correctly calls updateLabels when highlightPosition is at the highest position and the list continues beyond what is displayed. Check that if the current keyPhrase is “#JUNCTION” that the chooseButtion() method is called.

VII. **test06*_removeButton**– Tests that if the current keyPhrase is “#JUNCTION” then a JOptionPane is opened to prompt the user that this action is not undoable. Also tests that any call to remove() within this method is followed by updateLabels().

VIII. **test07_setPinButton**- Tests that this method correctly invalidates any input that is not a single letter OR an 8 digit number consisting of 0’s and/or 1’s.

Project Name:	
Treasure Box Braille	
Test Case ID : testa1_controller	**Test Designed By: ExMech Team**
Test Priority (Low/Medium/High): Med	**Test Designed date:** <date>
Module Name: Controller	**Test Executed by:** ExMech Team
Test Title: Test with program UI	**Test Execution date:** <date>
Description: Verify a working welcome screen	
Pre-conditions: User clicked on program	
Dependencies: Successful installation of program and associated files	

Steps	Test Steps	Test Data	Expected Result	Actual Result	Status(Pass/Fail)	Notes
1.	Click on executable jar file	Button click *Treasure.jar*	Application run	Application run	✗✓	
2. *	Create new story	Button click "New Story"	Display cell/button dialog	Display cell/button dialog	✓	
3.	Choose number of Cells and Buttons	Scroll list for Cells : Choose 3. Scroll list for Buttons : Choose 3 Button click "OK"	Empty Scenario loaded with 3 cells and 3 buttons	Empty Scenario loaded with 3 cells and 3 buttons	✓	
1. *	Load Existing Story	Button click "Load Existing"	Open File Chooser in Factory Scenarios directory	File Chooser opened in Factory Scenarios directory	✓	
2.	Parse existing story	Button click *some existing file,*	File successfully parse it to program	File parsed into program with appropriate headings matching correct elements	✓	

FocusPolicy (b)

I. **test00_fields** - Tests that there are the correct number of fields in the class and that they are initialized to the right values.

II. **test01_ctorLoadParser** - Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects.

III. **test02_addOrder –** Tests that the Vector<Component> order field contains the correct Vector list of components. Ensures manually set focus order is in effect.

ListManager (c)

I. **test00_init -** Tests initialization of the constructor. Checks that the number of cells and number of buttons is correctly set.

II. **test01_add -** Tests that adding the next node will correctly increment the index as well as insert the correct key phrase and data into the node.

III. **test02_size -** Tests that the size of the current list grows are more nodes are added.

IV. **test03_index -** Tests that the index correctly increases are new nodes are added.

V. **test04_getData -** Tests that retrieving data from nodes out of bounds will throw the proper IndexOutOfBoundsException.

VI. **test05_getKeyPhrase -** Tests that retrieving key phrases from nodes out of bounds will throw the proper IndexOutOfBoundsException.

VII. **test06_createJunction -** Tests the creation of a junction. Assert that the names of each new branch is the same as the given name in the parameter.

VII. **test07_junctionGoto -** Tests that going to a node in a junction by index returns the correct node. Checks that an uninitialized branch correctly returns null.

VII. **test08_junctionSearch -** Tests that searching a node by name will return the correct index. If the name is not found, then assert that -1 is returned instead.

VII. **test09_remove -** Tests that removing a node will only remove the current node and correctly changes the index. Removal of a junction node should remove the junction (and all its branches) and append the rest of the list. Removal of a button node should remove the contents of the button but should not modify the rest of the data structure.

Project Name:	
Treasure Box Braille	
Test Case ID : testc1_ListManager	**Test Designed By: ExMech Team**
Test Priority (Low/Medium/High): Med	**Test Designed date:** <date>
Module Name: Controller	**Test Executed by:** ExMech Team
Test Title: Test with program UI	**Test Execution date:** <date>
Description: Verify correct traversal of list	
Pre-conditions: User already in/created a story on program	
Dependencies:	

Steps	Test Steps	Test Data	Expected Result	Actual Result	Status(Pass/Fail)	Notes
1.	~already in program~ Click on "Add User Input"	Button click *Add User Input*	New *Create a question* dialog box opens	*Create a question* dialog box pops up	✓	
2.	Select and name desired buttons (branches). Press Ok upon completion	Checkbox check desired buttons to enable 'branches' Enter name for branches in enabled textboxes. Button Click "OK"	Upon checkbox check for desired buttons, textboxes should appear beside buttons for names of new branches. Branches created after OK clicked	Text boxes appear, accept text and new Screen is refreshed with new branch after "OK" is clicked	✓	
3.	Add elements under new branch	Button Click and add several elements to the new branch to fill list	New elements successfully added to list and list expands and contracts in size as needed	New branch accepted addition and removal of element	✓	
4.	Navigate back to master branch	Button click up arrow "^" till top of list is reached and passed	Program successfully navigates to the top of current branch/list and back to master branch after the head of last branch is passed	Highlighted element changed on button press till top of list was reached and upon another button click, the list reverted back to the main list/branch/storyline	✓	
5.	Navigate to other branches and branches	Button click *down arrow* "v" till and past bottom of the main list	Dialog box pops up after the last element in the top list/story line which choices of what branch to traverse next	Dialog box pops up and upon button click, changes the current list to the new branch/list that is clicked	✓	

LoadParser

I. **test00_fields -** Tests that there are the correct number of fields in the class and that they are initialized to the right values.

II. **test01_ctorLoadParser -** Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects.

III. **test02*_fromText**- Test the fromText(**String** scenarioFile) method in the class LoadParser. Test cases checks that the method reads the String scenarioFile and that the method checks the first and splits the first two lines of the file and gets the cell and button number from the file and then creates a ListManager class object. Checks that method Throws IllegalArgumentException if the cell and button values are not positive integers. The test cases also ensure that scenario file is iterated through, elements in the file are identified and nodes with corresponding data are created for every valid KeyPhrase identified in the file. Checks that IOException is thrown by class if error occurs. Test case also checks that the created ListManager object is a valid list and is returned All tests for fromText(**String** scenarioFile) are labeled alphabetically in place of the * character.

IV. **test03*_junction** - Tests the junction(**ListManager** result) method in the class LoadParser. This test case is responsible for checking that junction nodes are created in **ListManager** result. It checks that the method creates separate list objects for different junctions and catches IllegalArgumentException thrown by the method. Test also ensure that a final path is created where all the junctions lead up to. All tests for junction(**ListManager** result) are labeled alphabetically in place of the * character.

V. **test04_nextLine** – Tests the nextLine() method in the class LoadParser. This test case checks that the next line in a file reader is returned when the method is called. Test case also catches IOException.

VI. **test05_scenario_01 –** Tests LoadParser() by loading the example scenario file: scenario_01.txt and checking that navigating through the parsed data structure using ListManager yields the expected node data and list lengths.

VII. **test05_scenario_02 –** Tests LoadParser() by loading the example scenario file: scenario_02.txt and checking that navigating through the parsed data structure using ListManager yields the expected nodes and list lengths.

VIII. **test05_scenario_03 –** Tests LoadParser() by loading the example scenario file: scenario_03.txt and checking that navigating through the parsed data structure using ListManager yields the expected nodes and list lengths.

Project Name:	
Treasure Box Braille	
Test Case ID : testd1_LoadParser	**Test Designed By:** ExMech Team
Test Priority (Low/Medium/High): High	**Test Designed date:** <date>
Module Name: Controller	**Test Executed by:** ExMech Team
Test Title: Test with program UI file	**Test Execution date:** <date>
Description: Verify correct parsing of existing scenario	
Pre-conditions: User has an existing scenario file to load	
Dependencies:	

Steps	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**	**Notes**
1.	Click on executable jar file	Button click *Treasure.jar*	Application run	Application run	✓	
2.	Load Existing Story	Button click "Load Existing"	Open File Chooser in Factory Scenarios directory	File Chooser opened in documents directory (test performed on a system missing some install files)	✓	
3.	Parse existing story	Navigate and Button click *some existing file,*	File successfully parse it to program	File parsed into program with appropriate headings matching correct elements	✓	
4.	Navigate back to master branch	Vary Button click up arrow "^" and Button click *down arrow* "v"	Program successfully navigates to the top of current branch/list and back to master branch after the head of last branch is passed and other branches based on user choice	Highlighted element changed on button press till top of list was reached and upon another button click, the list reverted back to the main list/branch/storyline	✓	
5.	Simulate imported story	Button click "*Simulate*" in file menu of program	Program converts current story to readable scenario file and opens simulator	Dialog box pops up and upon button click, changes the current list to the new branch/list that is clicked	✓	

Node

IX. **test00_fields -** Tests that there are the correct number of fields in the class and that they are initialized to the right values.

X. **test01*_ctorNode -** Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are labeled alphabetically in place of the * character.

XI. **test02*_junction** – Tests the junction(**ArrayList**<**ArrayList**<Node>> buttons, **HashMap**<Integer, String> buttonsNames,**ArrayList**<Node> next) method of Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits buttons and buttonsNames from values provided when the method is called. All tests for junction(**ArrayList**<**ArrayList**<Node>> buttons, **HashMap**<Integer, String> buttonsNames,**ArrayList**<Node> next)are labeled alphabetically in place of the * character.

XII. **test03*_button** – Tests the button(**String** name, **ArrayList**<Node> prev) method in Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits name and prev from values provided when the method is called. It also checks that the method returns the created node object. All tests for button(**String** name, **ArrayList**<Node> prev)are labeled alphabetically in place of the * character.

XIII. **test04*_buttonNext** – Tests the buttonNextt(**ArrayList**<Node> prev) method in Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits prev from values provided when the method is called. It also checks that the method returns the created node object. All tests for buttonNextt(**ArrayList**<Node> prev)are labeled alphabetically in place of the * character.

XIV. **test05*_tail** – Tests the tail(**ArrayList**<Node> next) method in Node class. This test case checks that a new node object is created and that the fields in the object are set to the correct values. Test case also ensures that the object inherits next value provided when the method is called. It also checks that the method returns the created node object. All tests for tail(**ArrayList**<Node> next) are labeled alphabetically in place of the * character.

XV. **test06_setKeyPhrase** – Tests the setKeyphrase(**String** s) method of Node class. This test case checks that the key phrase of current node is set to **String** s value provided when the method is called.

XVI. **test07_setData** – Tests the setData(**String** s) method of Node class. This test case checks that the data of current node is set to **String** s value provided when the method is called.

XVII. **test08_getKeyPhrase** – Tests the getKeyPhrase() method of Node class. This test case checks that correct key phrase of type string is returned when the method is called.

XVIII. **test09_getData** – Tests the getData() method of Node class. This test case checks that correct Data of type string is returned when the method is called.

XIX. **test10_getButtons** – Tests the getButtons() method of Node class. This test case checks that correct Buttons of type ArrayList<ArrayList<Node>>is returned when the method is called.

XX. **test11*_braileToLetter** – Tests the braileToLetter(String s) method. This test case checks that for a set of inputted char, the correct brail sequence is returned. It takes in the String s and checks it against the braille index then calls the braileToLetters method with the String s and checks that it returns the correct value. All tests for braileToLetter(String s) are labeled alphabetically in place of the * character.

RequestFocusListener

IV. **test00_fields -** Tests that there are the correct number of fields in the class and that they are initialized to the right values.

V. **test01_ctorLoadParser -** Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects.

VI. **test02_ancestorAdded –** Tests that once RequestFocusListener is initialized ancestorAdded method has correctly applied requestFocusInWindow() on the component.

ScenarioComposer

I. **test00_fields -** Tests that there are the correct number of fields in the class and that they are initialized to the right values.

II. **test01*_ctorScenarioComposer -** Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are labeled alphabetically in place of the * character.

III. **test02*_returnStringFile**- Tests the returnStringFile(ListManager aList) method in ScenarioComposer class. This test case checks the file returned by the method. It checks that the for loop iterates through the given ListManager object aList and checks the keyphrase of each node. It checks that based on the keyphrase of a node, the returnStringFile method performs its related set of actions. It also checks that the file writer object writes correctly to the text file. The test cases are also responsible for ensuring junction nodes in each ListManager aList are created to scenario file requirements. All tests for returnStringFile are labeled alphabetically in place of the * character.

SoundRecorder

I. **test00_fields -** Tests that there are the correct number of fields in the class and that they are initialized to the right values.

II. **test01*_ctorSoundRecorder** – Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are labeled alphabetically in place of the * character.

III. **test02*_CaptureAudio**– Tests the capture audio method. The test cases ensure check that the audio format is set and is supported, checks that the data line is opened for recording and closed afterwards to be reused. It also checks that the information received from the data line is stored as byteArrayOutputStream for play back and saving audio files. The test also ensures that the correct exceptions are thrown in case of errors. All tests for CaptureAudio method are labeled alphabetically in place of the * character.

IV. **test03*_PlayAudio**– Tests the PlayAudio method. Implemented test cases ensure that the data saved as byteArrayOutputStream is converted to InputStream data. It checks that the audio being played is of the correct format and that an audioLine is opened for the InputStream. The test also checks that the line is closed after use to prevent LineUnavailableException from being thrown. It also checks that the JDialogBoxes for saving and exporting files are displayed immediately after playback.

The method playAudio is also responsible for playing audio imported to the Audio Studio. Tests ensuring the imported file is a wav file and that the wav file is playable are also included in this test case. The test also ensures that the correct exceptions are thrown in case of errors. All tests for PlayAudio method are labeled alphabetically in place of the * character.

V. **test04_getFormat**- Tests the getFormat method. This test case ensures that the method returns the correct audio format when called.

VI. **test05*_save**- Tests the save(File) method. Implemented test cases ensure that the data saved as byteArrayOutputStream is converted to InputStream data. It checks that the audio being played is of the correct format and that the InputStream data is written to the file provided to the method. The case also checks that this file is stored for exporting and then closes the byteArrayOutputStream. All tests for save(File) method are labeled alphabetically in place of the * character.

Project Name:	
Treasure Box Braille	
Test Case ID : teste1_soundRecorder	**Test Designed By:** ExMech Team
Test Priority (Low/Medium/High): High	**Test Designed date:** <date>
Module Name: Controller	**Test Executed by:** ExMech Team
Test Title: Test with program UI file	**Test Execution date:** <date>
Description: Ensure Sound recorder captures, and coverts sound as expected	
Pre-conditions: Scenario file is being created or opened, new node	
Dependencies:	

Steps	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status(Pass/Fail)**	**Notes**
1.	Open sound Recorder program	Button click *"Audio Recording"*	Audio studio pop up with only *"capture"* button enabled	Application pops up with all buttons but *"capture"* disabled	✓	
2.	Begin capturing audio	Button click "*Capture*" in *audio studio* program	*Capture* button is disabled, beep sound plays indicating recording has begun, *play* button enabled and recording starts.	*Capture* button is disabled, *play* button enabled afterwards and recording starts.	✓	
3. *	Stop recording	Button Click " *Stop"*	Recording stops after *stop* is clicked, *play and capture* buttons enabled and *stop* becomes disabled.	Audio recording stops with a beep , *stop* button disabled, *play and capture* enabled.	✓	
4.	Playback	Button click *"Play"*	Upon clicking *play* recorded audio plays. At the end of the recording, a series of popups determine what action is required next♦	Audio plays back, after play, a sequence of pop ups determine if a new recording should be made or to save current recording.	✓	
3. *	Re-record audio	Button click "Capture" _____ ♦ Follow pop up from playback menu to re-record	Previous recording is discarded, *Capture* button is disabled, beep sound plays indicating recording has begun, *play* button enabled and recording starts.	*Capture* button is disabled, *play* button enabled afterwards and recording starts. Previously recorded (unsaved) recording discarded	✓	

View

I. **test00_fields -** Tests that there are the correct number of fields in the class and that they are initialized to the right values.

II. **test01*_ctorView** – Tests the constructor, ensuring the proper instances in the class are initialized and checks new objects created are done so with the appropriate initial state plus other information about the objects. All tests for constructor are listed below in place of the * character.

III. **test02*_actionNextBtn** – Tests that the nextButton method in the controller class is called when this button is pressed.

IV. **test03*_actionPrevBtn** – Tests that the prevButton method in the controller class is called when this button is pressed.

V. **test04*_actionMenuClr** – Tests that the newMenuItem method in the controller class is called when this menu item is pressed.

VI. **test05*_actionMenuOpen** – Tests that the openMenuItem method in the controller class is called when this menu item is pressed.

VII. **test06*_actionMenuSave** – Tests that the saveMenuItem method in the controller class is called when this menu item is pressed.

VIII. **test07*_actionMenuSim** – Tests that the simulateScenario method in the controller class is called when this menu item is pressed.

IX. **test08*_actionBranchBtn** – Tests that the branchItButton method in the controller class is called when this button is pressed.

X. **test09*_actionTextBtn** – Tests that the addTextButton method in the controller class is called when this button is pressed.

XI. **test10*_actionResetBtn** – Tests that the addResetButtons method in the controller class is called when this button is pressed.

XII. **test11*_actionSoundBtn** – Tests that the soundButton method in the controller class is called when this button is pressed.

XIII. **test12*_actionRecordBtn** – Tests that the recordSoundButton method in the controller class is called when this button is pressed.

XIV. **test13*_actionPauseBtn** – Tests that the addPauseButton method in the controller class is called when this button is pressed.

XV. **test14*_actionSetPinBtn** – Tests that the setPinButton method in the controller class is called when this button is pressed.

XVI. **test15*_actionSetWrdBtn** – Tests that the setDispStringButton method in the controller class is called when this button is pressed.

XVII. **test16*_actionClrPinBtn** – Tests that the clrPinButton method in the controller class is called when this button is pressed.

XVIII. **test17*_actionRemoveBtn** – Tests that the removeButton method in the controller class is called when this button is pressed.

XIX. **test18_actionEnterKey** – Tests that the ENTER key executes any button event in the same manner as a left mouse click.

XX. **test19_actionUpKey** – Tests that the UP key executes the “prevButton” button event in the same manner as a left mouse click.

XXI. **test20_actionDownKey** – Tests that the UP key executes the “nextButton” button event in the same manner as a left mouse click.

XXII. **test21_actionAltKey** – Tests that the ALT key opens the File Menu in the same manner as a left mouse click.

2. Test cases and Their Derivation

Controller

These test cases were derived from the most key methods inside of the controller class. The tests were developed to test key aspects of the class. It is not feasible and nor is it completely beneficial to have test cases for every single method in this class. At a minimum, our application will run with its basic features as long as these test cases pass.

FocusPolicy

This class was found on docs.oracle.com regarding the Focus Subsystem. These tests are derived from reading the documentation from oracle and determining what aspects of the code are being used in our application. In our case, we use the contructor and addFocus() method.

ListManager

Test cases are meant to test that ListManager correctly handles adding and removing new data. ListManager acts as our data structure so it is important that the current index is properly incremented or decremented. The tests also cover the junction node creation and navigation. These tests were derived from comparative tests for lists, mainly the successful navigation and modification of the data structure.

LoadParser

These test cases were derived from a well-formed understanding of the required format of the scenario file and the structure of our ListManager data structure. The class is responsible for converting already existing scenario files to ListManager objects that can be iterated through with our program. The test cases test the core functionality of the program based on the requirements for the LoadParser class.

Node

Test cases in this class are derived from an understanding of the basic functionality of a node. The test cases ensure that the class successfully performs actions such as node creation, getting and setting fields of current node , keeping tract of current node location, and keeping track of special nodes such as a junction node or a tail node.

RequestFocusListener

Test cases were derived for this convenience class based on what parts of the class is being used. The class was sourced from ‘Rob Camick’ (https://tips4java.wordpress.com/2010/03/14/dialog-focus/). We are only using the override of the ancestorAdded() method to force NVDA to focus on certain components. We test that requestFocusInWindow() has successfully forced focus on the component. This is the only method used by us in the class.

ScenarioComposer

Test cases for the class ScenarioComposer are written to ensure that the scenario file the class is responsible for creating follows the valid documented scenario format style. The ScenarioComposer class takes the information from our data structure ListManager and writes items in the data structure according to the keyphrase and data of every Node object in the provided ListManager Object.

SoundRecorder

The test cases implemented for this class include tests for methods in java sound api. Tests handle cases such as checking that the program sets the audio format for recorded audio, checking that audio lines are opened and closed for reuse and checking that saved files are .wav files and not of a different format. These tests were derived from an understanding of the java sound api and sound utilities .

View

These test cases were developed to test the action event of each button, menu item or any other component in the main frame of the application. For our application to perform all of its required features, all test cases for this class must pass. The implementation of each action event can be found in the Controller class.

3. Why are these test cases sufficient?

Controller

These test cases are sufficient to ensure the application runs and that the basic features of the application still work. Features such as navigation are covered in the test cases. Many more test cases could be developed, however a diminishing return would mean that 100% coverage is not a good indicator that the application is working as intended.

FocusPolicy

FocusPolicy overrides certain methods in FocusTraversalPolicy to allow a Vector<Component> field to dictate the order of focusing when pressing tab. The tests listed are sufficient because we are only using the constructor to override the normal traversal policy.

ListManager

These test cases are sufficient because they cover all the major methods of the class. There are many methods which are simply not used or were created for testing purposes. The test cases cover major aspects of a data structure such as adding and removing. It also tests the junction creation which allows for branching of the scenario. ListManager essentially manages many ArrayLists, so the testing covers the additional features of ListManager.

LoadParser

The test cases written for the class LoadParser are sufficient because they are written to test most possible scenarios for individual methods of the class. The tests check and ensure that the LoadParser class returns a valid ListManager object which can be iterated through with our program.

Node

The implemented test cases for this class are sufficient because they cover all the methods in the class. The tests ensure that these methods run as they are expected to and return whatever necessary values when called. It tests specifically, the conversion of characters to braille cell and ensures the correct braille cell value is returned for a given character.

RequestFocusListener

Test cases for this class are sufficient because they ensure that the desired component is being focused by NVDA. We test that requestFocusInWindow() has successfully forced focus on the component.

ScenarioComposer

The test cases for the ScenarioComposer class are sufficient as they can cover and ensure all the functionality of traversing the provided ListManager object, check that the class is identifying each node and performing several actions based on the current node and checking the successful writing of the final Scenario File.

SoundRecorder

The test cases implemented for this class are sufficient because they cover every part of the basic functionality of the sound Recorder Program.

View

These test cases are sufficient because they ensure that all components of the application are calling their respected methods in the Controller class. Furthermore, some of the test cases ensure that all operations in the application can be controlled using the keyboard.

4. Test Coverage Discussion

Test Coverage is currently in progress. Test cases have been implemented for the data structure / backend classes of the application and will continue to progress.

Test Coverage for SoundRecorder class

We were able to achieve high coverage values on basic functionality of the SoundRecorder program. The main method being `captureAudio()` has a coverage of 77.1% for simple testing done on the class

Main (Aug 21, 2018 6:31:40 PM)

Element	Coverage	vered Instructions	lissed Instructions	Total Instructions
SoundRecorder.java	70.7 %	827	342	1,169
SoundRecorder	81.5 %	560	127	687
KeyListenerTester	0.0 %	0	78	78
fileChooser()	42.9 %	24	32	56
isImport()	0.0 %	0	32	32
playAudio()	81.2 %	56	13	69
playDaBeep()	59.4 %	19	13	32
captureAudio()	77.1 %	37	11	48
main(String[])	0.0 %	0	9	9
close()	0.0 %	0	8	8
optionboxYesNoCancel(	72.0 %	18	7	25
optionbox(String, String	90.0 %	18	2	20
getFormat()	100.0 %	19	0	19
infoBox(String, String)	100.0 %	12	0	12
save(File)	100.0 %	34	0	34
SoundRecorder(Controll	100.0 %	319	0	319

Test Coverage for ListManager class

Many of the test cases have been implemented. The ListManager class stores the data which is parsed from the scenario file. All the major functions are currently covered. Prev, next, remove methods test cases are outlined above and will be implemented soon.

Main (Aug 21, 2018 6:31:40 PM)

Element	Coverage	vered Instructions	lissed Instructions	Total Instructions
ListManager.java	59.2 %	446	307	753
ListManager	59.2 %	446	307	753
remove()	0.0 %	0	127	127
getData(int)	0.0 %	0	27	27
getKeyPhrase(int)	0.0 %	0	27	27
next()	57.1 %	32	24	56
printString()	0.0 %	0	22	22
prev()	71.7 %	43	17	60
junctionSearch(String)	70.9 %	39	16	55
createJunction(HashMap	90.3 %	121	13	134
setIndex(int)	0.0 %	0	11	11
addNext(String, String)	81.0 %	34	8	42
getNode()	72.2 %	13	5	18
getNextList()	0.0 %	0	4	4
getIndex()	0.0 %	0	3	3
junctionGoto(int)	90.3 %	28	3	31
ListManager(int, int)	100.0 %	44	0	44
ListManager(ListManage	100.0 %	22	0	22
getData()	100.0 %	4	0	4
getKeyPhrase()	100.0 %	4	0	4
getLabel()	100.0 %	4	0	4
getLabel(int)	100.0 %	25	0	25
goHome()	100.0 %	8	0	8
printString(String)	100.0 %	25	0	25

Test Coverage for Node class

Major test cases for the Node class is implemented.

Main (Aug 21, 2018 6:31:40 PM)

Element	Coverage	vered Instructions	lissed Instructions	Total Instructions
Node	48.0 %	261	283	544
brailleToLetter(String)	0.0 %	0	226	226
setLabels(String, String)	71.0 %	120	49	169
setData(String)	0.0 %	0	4	4
setKeyPhrase(String)	0.0 %	0	4	4
button(String, ArrayList<	100.0 %	35	0	35
buttonNextt(ArrayList<N	100.0 %	21	0	21
junction(ArrayList<Array	100.0 %	24	0	24
tail(ArrayList<Node>)	100.0 %	18	0	18
Node()	100.0 %	3	0	3
Node(String, String)	100.0 %	28	0	28
getButtons()	100.0 %	3	0	3
getData()	100.0 %	3	0	3
getKeyPhrase()	100.0 %	3	0	3
getLabel()	100.0 %	3	0	3

Test Coverage for LoadParser Class

Coverage score for the LoadParser class is low as it is one of the less used classes. In testing a pre-made document was loaded up and checked to ensure program converted file effectively. This was carried out mainly by the `fromText()` method

Main (Aug 21, 2018 6:57:21 PM)

Element	Coverage	vered Instructions	lissed Instructions	Total Instructions
LoadParser.java	32.7 %	156	321	477
LoadParser	32.7 %	156	321	477
junction(ListManager)	0.0 %	0	245	245
fromText(String)	77.1 %	138	41	179
main(String[])	0.0 %	0	32	32
nextLine()	75.0 %	9	3	12
LoadParser()	100.0 %	9	0	9

Overall Coverage (Work in Progress)

Overall coverage is not ideal at this time as the development process has meant that many classes have constantly changed. Test cases will be implemented in the near future as outlined in this document. We do not expect to implement full coverage as it is not feasible and is counter-productive because test cases are specific and limited while the application is complex and systems are coupled. Further testing must be done by running the application and testing features manually to ensure quality.

Enamel	59.6 %	7,039	4,777	11,816
src	59.6 %	7,039	4,777	11,816
enamel	59.6 %	7,039	4,777	11,816
Controller.java	53.3 %	1,860	1,628	3,488
ScenarioParser.java	48.1 %	580	626	1,206
LoadParser.java	0.0 %	0	477	477
ListManagerTest.java	0.0 %	0	466	466
SoundRecorder.java	70.7 %	827	342	1,169
ListManager.java	59.2 %	446	307	753
Node.java	48.0 %	261	283	544
Logger.java	0.0 %	0	203	203
View.java	94.4 %	1,528	90	1,618
VisualPlayer.java	80.8 %	311	74	385
BrailleCell.java	77.9 %	239	68	307
SoundRecorderTest.java	0.0 %	0	62	62
FocusPolicy.java	21.4 %	15	55	70
Player.java	75.4 %	129	42	171
ValueComparator.java	0.0 %	0	29	29
ToyAuthoring.java	0.0 %	0	9	9
AudioPlayer.java	0.0 %	0	7	7
ScenarioComposer.java	98.8 %	402	5	407
Main.java	87.5 %	21	3	24
RequestFocusListener.java	96.0 %	24	1	25
BrailleCellPanel.java	100.0 %	116	0	116
GenerateTree.java	100.0 %	265	0	265
NodeInfo.java	100.0 %	15	0	15