AUTHORING APP TESTING DOCUMENTATION

Group 7

Sara Attalla
Sarah Feroz
Sean Doyle

Date
23-Feb-2018

Table of Contents

1. Introduction ... 1
2. Scope ... 1
3. Application Testing CheckList ... 1
4. Derivation of Test Cases ... 2
5. Test Cases Descriptions and Implementations ... 4
6. Test Coverage ... 8
7. Discussion ... 8
8. Conclusion ... 9

1. INTRODUCTION

The purpose of this testing document is to provide details about the implementations used to test and validate the functionality of the Authoring App features.

2. SCOPE

This document contains an Application Testing Checklist, which will provide a brief description of the main procedures of the Authoring App as well as their expected results. Results are assessed on a P/F bases with brief descriptions about the actual outcome of the procedures. Following the checklist is a description of the test cases that have been implemented as well as a discussion on how these test cases were derived. And why they are sufficient. Finally, the documentation is concluded with a test coverage discussion.

3. APPLICATION TESTING CHECKLIST

Application Testing Checklist			
Tested By Gr. 7	Tester	Date	February 23, 2018
Application Name	Authoring App		

Procedure	Expected Result	Pass/Fail (P/F)	Actual Results/Comments
Application Functionality			
Performs primary functionality and maintains stability	Yes	P	Opens to Main Menu and GUI looks professional and functions with out any visible errors
Basic Application Testing			
Performs as expected when other applications are open	Yes	P	No errors or hangs when multiple applications are running
Starts from the Main menu	Yes	P	Opens correctly from the Main menu and functions with no errors
Has option to chose a file to play scenario	Yes	P	Opens a file chooser properly and if a valid file will play scenario
Does not crash when Scenario is not in correct format	Yes	P	Program does not crash, but error message does not pop up

Application Testing Checklist			
Tested By Gr. 7	Tester	Date	February 23, 2018
Application Name	Authoring App		

Procedure	Expected Result	Pass/Fail (P/F)	Actual Results/Comments
Error window pops up if incorrect file format selected	Yes	F	
Author a Scenario opens to the editor window	Yes	P	
Application provides the proper tools to obtain user input about buttons, cells and pins	Yes	P	Does not accept negative numbers or numbers that are out of bounds
Editor window allows for interactive authoring of questions and answers	Yes	P	
Option to create a new scenario or edit an existing file	Yes	P	Editing scenario files mostly works as intended. Saves new files to the specified path
Handles creating the correct Scenario File format	Yes	P	Creating the proper format mostly works as intended
Applications terminates and exits as expected	Yes	P	

4. DERIVATION OF TEST CASES

Test cases for the Authoring App are brought out by first referring to the application-testing checklist which establishes the basic functionality of the application. Once the basic functionality is been established, using the requirements of the Authoring App which guided its creation, a scenario was developed for each of the main prerequisites. From these perquisites, different pathways that the user could take to interact with the application (ie inputs) were identified. From identifying a large scope of possible inputs, an expected result was predicted as well as a plan on how to test that prerequisite.

Scenario 1: Create a new Scenario File

> ***Expected:*** *File is created and saved where user specifies in the correct format while user only interacts with a user-friedly GUI without having to manually format.*

Approach: *First test that a new file is being created, then test if file is in correct format. Junit testing was done to ensure file exists. Then using the scenario formatting document provided, it was possible to create a text file with an expected format with a predetermined set of text fields. Next, calling our ScenWrite class, responsible for obtaining user fields and transferring them to a text document in the correct format, it is then possible to compare the two text files using a FileReader. However, an important part of this implementation is to verify that the correct text fields are being stored in the correct variables. This is achieved through JUnit testing on the Scenario class as well as Event List class which are where user inputs are stored.*

Actual: *Files are outputted and saved in appropriate format.*

Scenario 2: Edit an Existing File

Scenario 2a: Change a variable like the title, button, cells or pins

Expected: *The variable to be changed without losing functionality*

Approach: *tested all appropriate accessor methods*

Actual: *Variables are able to be changed through the Event Editor window without losing functionality*

Scenario 2b: Add a question and answer to the end of the file

Expected: *EventList.add maintains functionality and adds to the end of the list such that when using the accessor methods, the appropriate object is returned*

Approach: *test adding and remove methods of EventList when adding to the last index*

Actual: *Funtions as expected*

Scenario 2c: Add a question and answer somewhere else in the file

Expected: *EventList.add maintains functionality and adds to the middle of the list at a specified index such that when using the accessor methods, the appropriate object is returned*

Approach: *use EventList.add(index, element) to add to the middle of the list and check that the Event is added properly by using accessor methods. Also check that all other elements have shifted their indexes.*

Actual: *Functions as expected*

Scenario 2d: Delete an existing question

Expected: *Event list deletes an event and adusts all other indeces on the list whether removing from the head, middle or tail*

Approach: *use EventList.remove(index, element) to check that the Event is removed properly and indexes are adjusted by accessing the Events before and after*

Actual: *Works as expected.*

Scenario 2e: Edit an existing question

Expected: *changes in the event list are reflected if the event is called and changed*

Approach: *test accessor methods if Event and EventList*

Actual: *Works as expected*

Scenario 3: Play a File

Scenario 3a: File is not in the correct format

Expected: *Error window pops up to let user know – reads from error log*
Approach: *Check error log reader by creating a text file with the expected output and reading ERROR_LOG.txt or intentionally choose a file with the wrong file format and record results*

Actual: *Still under construction*

Scenario 3b: File is in correct format

Expected: *Audio player reads from file*

Approach: *Test with correct file format, which can be created by interacting with the GUI*

Actual: *Works as Expected*

5. TEST CASES DESCRIPTIONS AND IMPLEMENTATIONS

AuthoringApp Test Suite: AllTests			
Test Case	Description & Implementation	Pass or Fail	Notes
EventListTest Class: The set up of this test class included creating a new EventList tester, and 5 different events and adding each event to the EventList tester			
testgetSize	Returns true if the size of the EventList is 5. Retruns true if the size is updated after every add and remove. Implemented by adding 5 items to EventList tester and asserting the target size. Assertions are also made after an event is removed or added.	Pass	Size field stays private

testUpdateIndexes	Returns true if successfully updates the indexes of the EventList every time an event is added. Implemented by adding to the front, end and middle and asserting true of the correcting indexes after each addition.	Fail	Bug detection – will be fixed for phase 2
testAdd	Tests adding to an index, by implementing updateIndexes. Asserts true if the event at a given index is equal to the expected event after a call to add. Assertions are made after an add to front, end and middle.	N/A	Since adding implements the method testUpdateIndexes, there is a ripple effect from the previous bug. This will be resolved for phase 2
testGetTimeline	Asserts true if correct timeline is returned. Checks that the timeline field of the created Scenario object is the same as the expected EventList.	Pass	
testRemove	Asserts true if timeline size decreases by one and the indexes are updated on the eventList	Pass	Although the remove method implements the estUpdateIndexes, there is no ripple effect of the bug here.
ScenarioEventTest: Two different scenarioEvent objects are created for the set up			
testGetTitle	Asserts true if correct title is returned. Checks that the title field of the created scenarioEvent object is the same as the expected string	Pass	Title is kept private
testGetQuestion	Asserts true if correct Question is returned Checks that the question field of the created scenarioEvent object is the same as the expected string	Pass	Question field is kept private
testOverWrite	Asserts true if Event that is used to call the method has it's values overwritten by use inputs Checks that all fields of an EventList object created during setup are	Pass	This is the only way to change any of the fields.

	changed after the overwrite method is called		
testResponseRight	Asserts true if the “Response Right” field is accessed correctly Checks that the responseRight field of the created scenarioEvent object is the same as the expected string	Pass	responseRight field is kept private
testResponseWrong	Asserts true if the “Response Wrong” field is accessed correctly Checks that the responseWrong field of the created scenarioEvent object is the same as the expected string	Pass	responseWrong field is kept private
getCorrectAnswer	Asserts true if correct Answer is returned Checks that the Correct Answer field of the created scenarioEvent object is the same as the expected string	Pass	correctAnswer field field is kept private
ScenarioTest: Set up for this unit test includes creating a new Scenario object with a title, cell number, button number and eventList			
testGetTitle	Asserts true if correct title is returned. Checks that the title field of the created Scenario object is the same as the expected string	Pass	Title field remains private
testGetButtonNumber	Asserts true if correct button number is returned. Checks that the button field of the created Scenario object is the same as the expected string	Pass	Title field remains private
testGetCellNumber	Asserts true if correct cell number is returned. Checks that the title field of the created Scenario object is the same as the expected string	Pass	Title field remains private
testsetButtonNumber	Asserts true if the button field is	Pass	Title field remains private

	successfully set. Uses the get method to checks that the button field of the created Scenario object is the same as the expected. As a result, the method checks to make sure that a predefined field is updated and does no longer equal the old value		
testsetScenarioEventList	Asserts true if the Scenario object's field EventList is successfully update when set to a new EventList object. Uses the get method to iterate every element of the list field of the created Scenario object is the same as the expected. By this, the method checks to make sure that a predefined field is updated and does no longer equal the old value	Pass	
testsetCellNumber	Asserts true if the cell field is successfully set. Uses the get method to checks that the cell field of the created Scenario object is the same as the expected. Checks to make sure that a predefined field is updated and does no longer equal the old value	Pass	Title field remains private
testgetScenarioEventList	Asserts true if correct cell EventList is returned. Checks that the all the events in the EventList field of the created Scenario object is the same as the expected string	Pass	
setTitle	Test case ensures title field of Scenario object is successfully updated. Asserts true if the get title returns the correct field value after calling the setTitle test case.	Pass	Title field remains private

ScenWriterTest: The set up of this class includes the creates of an EventList, tester, with 5 events added to it. Next, a new Scenario is created with an inputed cell number, button number, title and the EventList tester.

An “actual output” file is also created at set up to store the output of calling scenWrite.write(Scenario s, File f) as well as a file object for the expected file output. The “actual output file is a blank file named ScenWriterTestActual.txt. The expected file output is a file, ExpectedScenWritertest.txt that contains the expected output file that is formatted appropriately, given the user’s input information that was used to create a scenario.			
testWrite	This method asserts true if the created scenario is outputted to a file and formatted correctly according to the scenario formatting documentation. The correct format is pre defined and stored in a separate file (see ScenWriterTest class set up description above). This test case creates two file readers – one for the expected output file, called ereader and one to read the actual output file called areader. The readers go through both texts line by line and compare that the strings are equal.	Pass	This test is only viable given that the expected scenario file is correctly formatted by the tester. If there are errors in formatting in pre defined expected scenario file, then the test case may return a false positive or false negative.

6. TEST COVERAGE

Tester Name	**Execution Time**	**Did all the Tests Pass?**	**Test Coverage**
EventListTest	0.007 s	4/5	94.4%
ScenarioEventTest	0.000 s	6/6	78.7 %
Scenario Test	0.001 s	7/8	100.0%
ScenarioFileReader Test	N/A	N/A	N/A
ScenWriterTest	0.000 s	1/1	78.3 %

7. DISCUSSION

While the Authoring App cannot declare that it has accounted for all possible defects, but the testing provided for the Authoring app is sufficient for this stage of the project. The testing targets key requirements of the projects. This testing was developed in response to every possible path that the user can interact with the Authoring App. As a result, going through every pathway or scenario possible in which the user can interact with the app (see Section 4), the derived test cases are a result of the responses to every type of user input. Consequently, testing all the methods involved in each user input pathway exposed bugs, but also determines that the Authoring App is sufficiently usable at this stage of the project on a technical level. Furthermore, the claim that this approach sufficiently covers testing of a wide scope of the program is reflected in the Test Coverage metrics (see Section 6).

8. CONCLUSION

At this stage of the Authoring App development, this testing documentation demonstrates that sufficient testing methodology has been attained (Section 7) through the description of test cases (Section 5) , how they were derived (Section 2, 3, 4) and how they were implemented (Section 5) while including coverage metrics (Section 6).