Treasure Box Braille (TBB) Application

November 29, 2018

By:

Risheed Malatombee

Yoni Fihrer

Steve Sohn

Daniel Santaguida

Table Of Contents

1. Introduction:

To ensure accuracy and satisfactory performance of the application, we have outlined what functions need to be tested. It is especially important to test that every braille prompt is correct in its representation. We also test to ensure the application follows the correct path, particularly in the case of branching (i.e., non-linear layout).

In our preliminary launch of the application, we do not expect to have all testing in place. Much of the application lies in the graphical user interface, which in our determination, is best tested visually by running the application.

2. Implemented Tests:

Test	Expected	Actual	Test Results	Test Description
1) Creating new Scenarios	Creates a new text file incorporating the filename, file reference, buttons, cells	Filename mentioned at top, followed by the cells and buttons, followed by the content of the file.	Passed – Aligns with our expected and actuals	This test is to demonstrate that whatever the user places into the slots for creating a file is substantial and is saved correctly.
2) Top 5 Keys	Upon pressing hot keys, processes function per hot key.	Pressing each of the hot-keys directs user to appropriate screens and functions.	Passed – Aligns with our expected and actuals	This test is to demonstrate that the hotkeys are in top 5 buttons and move around and operate with ease.
3) Editing existing Scenarios	Allows changes to buttons and cells, as well as translation of the file content.	User can edit scenario based on the set of nodes, able to add more changes at will.	Passed – Aligns with our expected and actuals	This test is to ensure user is able to edit scenarios by adding more nodes with changes associated to each node.
4) Create Audio File	Upon pressing the record button, records voice, and upon ending the recording	Able to get to recording session, start and end sessions and see recorded files.	Passed – Aligns with our expected and actuals	This test is to ensure that a wav file exists with the same time properties as what was recorded, and same name property is associated to it.

	session saves to a .wav file.			
5) Initialize text to speech	After running file manipulations, opens Authoring application to verbally read out text.	Authoring application opens and can read out text.	Passed – Aligns with our expected and actuals	This test is to demonstrate that the editing screen directly goes to the authoring application and commences the text to speech function.
6) Set cells and buttons	Able to set cells and buttons per editing screens button layout.	Cell and button manipulation can change.	Passed – Aligns with our expected and actuals	This test it to demonstrate that the buttons and cells can be instantiated and changed upon request.
7) Braille character representation	Each letter of the English alphabet is correctly represented in Braille	Upper-case and Lower-case letters use correct grammar and are represented in braille.	Passed – Aligns with our expected and actuals	This test is to demonstrate whether the pins on the authoring application is correctly shown per scenario.
8) Graphical structure integrity	Each time a button or node is added, it is visually and audibly updated by the application	Graph demonstrates the button and node action accordingly.	Passed – Aligns with our expected and actuals	This test is to demonstrate that the editing screen maneuvers through the processing nodes.
9) Button prompts are followed per Opening	Scenario is followed correctly as Directed by user	Scenario is followed correctly as directed by user	Passed – Aligns with our expected and actuals	This test is to demonstrate whether the scenario respond to the buttons.
10) Screen reader reliability	Every graphical element is relayed audibly to the user	Graphical elements have accessibility contexts read aloud	Passed – Aligns with our expected and actuals	This test is to demonstrate whether the graphical representation is accessible.

2.1 Test Derivation and Sufficiency:

1) For the first test, the goal in mind here was to start up the program with either using existing or new scenarios, upon creating a new scenario. We derived this test based on the competency of the program to translate user input into an actual text file to be used for further inspection later, if the test case were to pass, this would mean that the file was filled accordingly and translated without error.

2) For the next test, we were tasked with developing hotkeys for the top 5 most frequent buttons, with that said, we developed the hotkeys with the purpose of allowing the user to perform functions and translations at a button press. We derived this test case to solely check if the appropriate functions and translations were performed to the right places with the right components.
3) For the next test, this just confirms roughly what was done in the first test, however, this was for existing scenarios. We derived this test case for the sole function of checking if the scenario exists and was able to translate without error. Cases that would fail would be a null file, non-existing file, or some error with file translation.

4) For the next test, we were tasked with allowing the user to record their voice. With that said, we derived this test case with the purpose of checking if the wav file exists by checking if recorded file exists within the programs library by name. This test does not check the contents of the wav file as that was very difficult to do.

5) For the next test, we derived this test to check that when our program translates from the Editing Screen to the Authoring App, it does so by taking the changes made, along with the scenario used, applies the changes, and then analyzes if the text to be read out aligns with the commands specified and then a sound object is utilized.

6) For the next test, as this applies to the Start Screen and the Editing Screen, this test was derived to accommodate the changes made to the scenarios cells and buttons, basically this test checks if the cells and buttons matches the values entered in the slots on the screen, furthermore, when editing scenarios, it analyzes if the cell changes align with the buttons pressed as the buttons reference a number associated to it.

7) When coming across the seventh test, we derived this based on the system being able to comprehend not only letters but analyze the grammar that is incorporated. As such we created this test simply for the sake of the grammar represented in braille.

8) For the next test we wrote, while making the program we wanted to go with a nodal approach that's represented with a graph when the user decides how they want their text to represented. As such, we made this test to make sure the graph coordinates it actions with the buttons manipulating the text.

9) For the next test, while making the editing screen we created buttons to assist the user with how they wanted the text represented, as such when audibly reciting the scenario, we wanted to incorporate a test such that it analyzes and outputs the scenario per the users liking.

10) This last test goes hand in hand with the one above, with that said, we wanted the actions of the buttons in the editing screen to effectively relay the elements to the user as indicated by the user themselves. With that in mind, we created this test to make sure what we wanted audibly relayed was what we got.

2.2 Overall Rationale and Sufficiency:

It would not be difficult for non-braille users (in this case, the developers) to overlook any incorrect braille representations. As such, it would be particularly important to test and ensure the accuracy of any braille represented in our application.

The application also incorporates a small number of menu prompts, leading to different "areas" of the program. For example, on the main menu screen upon loading of the application, the user has the option to create a new scenario file. We would ensure that upon selecting that option (or any option for that matter), the application updates the text and buttons on the screen to be consistent with the part of the program that facilitates the creating of a new scenario.

As such, with the above tested for and the scenario being adhered to as expected, the basic functionality of the application would be achieved.

Furthermore, we test for correct audio cues being played when a prompt is selected, such as a "correct ding" audio file being played when the correct response to a question is selected.

Lastly, accessibility context is provided for all visual graphical elements that are present on the screen. As such, anytime a graphical component (such as a node, or the text within the node) is updated, added, or removed, this information is relayed audibly to the user. We wish to ensure that every graphical element relays this audible element.

3. Test Coverage:

For the sake of testing our code, we developed test cases using JUnit externally such that we can see where our code would change. With that said, for each of the test cases provided along with the derivation and the sufficiency, we have demonstrated how we recreated the occurrences and events such that all test cases were thoroughly defined and examined.

For the Test Coverage portion which we developed in Eclemma, we were able to see that our work demonstrated an overall coverage of 61.4% pass rate. With that said, we deemed this as sufficient as we were told in class that our work needed to be over 50% to be deemed as sufficient. With that said, we feel as though our work could have been better with the help of more test cases and Junit cases to alleviate the work done by the program, as there are a variety of cases we felt would have been very tedious and monotonous. For the editing screen and the start screen which harboured majority of the user interpretation, each relayed a pass rate of 81.3% and 82.1% respectively, which helped confirm that our code was secure and logically made sense, and which was used throughout the programs integrity. In terms of how this translated to the Authoring Application, although the testing coverage demonstrated a rather low coverage than expected with a coverage of 73.6%, we confirmed this was the case since a lot of the work done in the Authoring application was left roughly the same but used hints of other programs to demonstrate its functions. Basically, we simply used the Authoring Application as a voice from text to speech. Overall, our coverage summated to roughly 61% due the fact that some of the classes had small significances to the entirety of the code but played its own parts. All in all, we deemed our code to be sufficient, especially since our code passed all our cases that we felt were crucial to this project's definition.

4. Conclusion:

In conclusion, by completing analysis on the testing coverage and the test cases, we felt as though we did a thorough job of determining how to apply the test cases. As demonstrated from our test coverage which was deemed as 61% we defined this as sufficient as were told that code over 50% was acceptable. Furthermore, we then had a coverage of the test cases we provided and received a resounding 97.6% in terms of passes, where the other 2.4% accounted for the uncertainty and error that may have occurred (Which was something we never encountered though multiple runs and testing). Despite that, we applied test cases in places we

deemed as most vulnerable and dealt with the user mostly, but accommodating for those areas, It helped us flesh out our code and fix any lingering areas.