Testing Document
Tester: Ameer Bacchus

Coverage

Main (1) (Apr 3, 2019 10:55:25 PM)

Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
TalkBox	24.7 %	865	2,632	3,497
src	24.7 %	865	2,632	3,497
main.java.TalkBox	27.0 %	865	2,337	3,202
ControllerConfig.java	0.0 %	0	628	628
Main.java	55.5 %	493	396	889
ConfigurationApp.java	29.6 %	145	345	490
ControllerSimulator.java	35.0 %	163	303	466
RecordButton.java	0.0 %	0	236	236
SetButton.java	0.0 %	0	226	226
Record.java	0.0 %	0	97	97
TalkBoxButton.java	0.0 %	0	45	45
SwapButton.java	25.0 %	9	27	36
Log.java	71.4 %	40	16	56
AudioButton.java	57.7 %	15	11	26
TalkBoxApp.java	0.0 %	0	4	4
Controller.java	0.0 %	0	3	3
test.java.TalkBox	0.0 %	0	295	295

Our percentages were low because most instructions only get executed when buttons are pressed. When the program is first launched, a large portion of code from the files are not executed since no buttons are being pressed which is why the coverages are low.

Test Case – Adding more than 7 buttons

Previously I hard coded the program so that you can only add max 7 buttons. I changed the code so that the user can add as many buttons as they want. This test case was sufficient to see whether or not there were any issues with the new code. This test case was done manually through the GUI.

When executing this test case there was an issue. The GUI would not open at all. After examining the new code I found out that there was a mistake with one of my for loops, I made a mistake with the exit for the loop so instead of breaking, the loop was running infinitely which is why the GUI would not open.

Test Case – Visual Appearance for Multiple Screens

In order to allow the user to navigate through the buttons easily I created a GUI with arrow keys that allows the user to switch screens by clicking on it. Each screen has a similar appearance with 2 arrows on the bottom, the buttons of course and 2 gifs on the top to make it look nice. This test case was sufficient because the number of buttons will always change depending on the user's desires so the GUI must be versatile in the sense that it should be able to adapt to the number of buttons inputted. This test case was done manually through the GUI.

When trying to add more than 1 screen at first the images did not appear the way I wanted i.e the arrows and gifs were out of place. In order to solve this I created 2 methods, 1 creates a screen that

always has 7 buttons and the other creates a screen that will show the last buttons which can vary depending on the user input. The first method was somewhat easy since it always be the same, a screen with 7 buttons so from a visual appearance standpoint everything was fine with this method. The second method was more challenging since it creates a screen that changes depending on user input. I ran some test cases (inputted multiple buttons for multiple screens) to calculate how I should write my code so the GUI displays the arrows, buttons and gifs in the correct order every time regardless of the user input.

Test Case – Clicking Right Arrow on GUI

Since the user can now add as many buttons as they want, this also means there will be multiple screens that display the buttons. Arrows are used to navigate between the screens. This test case was sufficient to see if all buttons can be seen with no error. This test case was done manually through the GUI.

When clicking the right arrows, they seemed to work fine until the last screen. When trying to click the right arrow on the last screen there was an error because the way I coded it was when the user clicks on the right arrow it displays the next screen however since we are on the last screen there are no more screens to display. Because of this we get an out of index error. I fixed this by making sure that there was no ActionListener created for the last right arrow so when the user clicks on the last right arrow nothing happens.

Test Case – Drag and Drop

I implemented a drag and drop feature that allows the user to drag their files onto to the talk box. It was sufficient to test this to make sure the right files are being stored and that they are able to play when in the simulator. In order to test this feature I added a line that would print the file name that gets stored when someone drags something onto the talk box.

Everything worked fine however I needed a way to distinguish whether it was a picture being stored or a sound file. After testing a few examples I got this feature to work by checking the file name that is passed, and if the last 4 characters are ".wav", it will save it as a sound file and otherwise it gets stored as the picture.

Test Case – Serialization

One of the main issues with our program was that it would not save the previous settings. It is evident why this is a problem since it is not convenient for the user to remake their talk box every time. I used serialization to store the previous audio files on a file that gets read every time the user opens a new talk box. It was important to test this feature since there are many scenarios where things can go wrong.

To test this, I tried multiple scenarios for instance pressing Configure, Launch Simulator and then closing it only to re-open it and launch the simulator to see if my setting were saved. This was a very hard task since it was difficult to figure out where to serialize and where to deserialize. I used the Arrays.deepToString() function to assist me in seeing whether the serialization was working properly (since the audio files were stored in a 2D string array).

Test Case - .jar File

It is important to make sure the .jar file works properly since many things can go wrong when exporting as a .jar file. When I first tried to do this, the default sounds I stored (.wav files) were not playing when I ran the .jar file. In order to test this I printed out the paths of where my sound files were stored and realized instead of making a new Sound file each time with the path of the sound files I stored, I needed to get the URL of where the sound files were stored and play that instead. After doing this the sound files played and worked fine when running the .jar file.

Test Case – Swap Button

The swap buttons are essential part of our talk box. They allow the user to swap through audio sets at the press of a button. It is important to test this feature out since playing audio is essentially one of the main features of the talk box.

I tested this manually by setting up a talk box and trying to switch through the audio sets via the press of the swap buttons. I noticed that all my audio sets were playing fine except the last one. I added break points in my code and quickly realized that my buttons were all out of sync by 1 meaning if I tried to play the last audio set, I would get an index out of bounds error. I quickly fixed this by doing -1 to all of the positions to put them back into sync.

Test Case – Adding Scroll Bar

A cool feature that my group decided would be good for our talk box was having scroll bars. It makes it easy for the user to navigate through their buttons. It is important to test this feature because the scroll bar lets you access your buttons on the configurator and simulator.

When designing the trying to incorporate the scroll bar into my GUI I ran into multiple issues. For one, the scroll bar would appear, but it would have no functionality and just appear as if it was an image. I did some research into this and found out this was happening because I was using a flow layout and apparently scroll bars and flow lay outs aren’t compatible. To fix this I changed the layout of my JFrame to a grid layout and the scroll bar worked as intended.

Test Case – Layout of Config App

When making the GUI for the config app I needed a way to allow the user to configure their device easily. I used a Card Layout in order to do this. I placed multiple button panels on top of each other and whichever audio set the user clicks on, it will correspond to which button panel they are currently configuring. I made it easy to visualize which audio set they are configuring by making the border of the bottom highlight green whenever they click it. It was important to test this feature because we don’t want wrong audio files going to the wrong audio sets.

I first tested this by making the borders of the audio buttons glow red whenever they are clicked. I did this to see whether or not the button panels interacted correctly with the audio set buttons. Everything worked fine and I later went into deeper testing by actually trying this with sound files.

Test Case – Logger

One of the last features we added to our talk box was a logger. This logs the actions of the user and allows them to visualize them. This feature was important to test because the logs are important to us, they give us feed back on how individuals are using the talk box and will notify us of any errors.

We found it difficult to find a way to view the logs. We were going to open them in note pad but then realized not every computer has note pad. To get around this we decided on writing the text into a JFrame and displaying the logs from there. To test this feature we manually clicked on buttons and took note on whether or not the Logger was correctly logging our actions. Everything went fine and test was a success.