TalkBox

Testing DOcumentation

**Walker, Matt – 214779300 Kaplan, Allen – 215494925 Sakib, Saadman – 215916232 Lassonde School of Engineering Group 14**

**Revision History (Master branch)**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason for Changes | Version |
| [Kaplan, Allen], [Walker, Matt] | 2019-01-09 | Initial Commits, push/pull tests. | Alpha 0.01 |
| [Kaplan, Allen], [Sakib, Saadman], [Walker, Matt] | 2019-01-14 | Merge tests and print line statements. | Alpha 0.01 |
| [Walker, Matt], [Kaplan, Allen] | 2019-01-(16/21) | Converted java project to a maven project, declared SimpleNLG and a maven dependency, developed simple sentence generation, and created a GUI to display generated sentences. | Alpha 0.50 |
| [Walker, Matt] | 2019-01-29 | Implemented Configurator app, the ability to select custom audio files and SimpleNLG dictionary, launch the Simulator from the Configurator app and abstracted GUI elements. | Alpha 0.8 |
| [Walker, Matt] | 2019-02-20 | Separated Configurator and Simulator into two separate Jar files and implemented a serialized object allowing the Configurator to store settings remotely. | Alpha 1.0 |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Implemented Test Cases | M/I/- | Res Code | Resolution to the issues |
| 1. | Audio files are not present in directory | I | F/O | Implemented try catch statements. Upon catch the simulator does not generate custom audio buttons. No user feedback implemented. |
| 2. | User does not select the appropriate directory (audio or dictionary) | - | O | Generic case of 1. Resolution is similar; simulator does not generate custom audio buttons or load the combo boxes. |
| 3. | User attempts to play another audio file before the previous one finished | M | F | Simulator checks if audio clip is currently open and closes the clip before playing the new one. |
| 4. | Sentence is properly generated through utilization of SimpleNLG | M | O | Improved the generation of sentences by requiring the user to specify whether the word is a subject, verb, or object. This relies on the user correctly defining these attributes in the dictionary text file. |
| 5. | Generated sentence is properly converted to audio through utilization of MaryTTS | M | F | In accordance with case 3, the audio is correctly generated and played back. Future updates will include the ability to select the voice speaking and speed of speech. |
| 6. | Prevention from injection attacks | I | O | Removed text fields from user input to prevent code inputs. Issue may still arise if selected file is somehow executed; however, through file extension filters we have removed the possibility of selecting any file that is not a “.wav”. |

1. Classify defects as Major Defect (M), Issue (I), or Minor Defect (-). Only major defects and issues should be discussed at the review.
2. Resolution codes: **F** – fixed, per resolution description; **NC** – no change; **O** – open issue, temporary code which need to be changed before acceptance; **Dup** – duplicate issue; **Bug –** the finding will be resolved through the Bug/Defect process. The number of the bug should be put in the resolution column.

**Test Case Derivation and Implementation:**

**Summary:** The following test cases were derived through examining various attributes of the application at run time. This is because the applications function relies on user input and therefore, not all aspects of the program can be tested for at the same time. Changes in the Configurator application can affect how the Simulator application functions so the program must account for every possible configuration of the device and for possible user error. Because it is impossible to test for every case of user error our goal is to develop the application such that it does not unexpectedly terminate at anytime during its lifecycle. Finally, we would like to provide the user with as much feedback as possible regarding selections and errors; therefore, we have opted to use various labels to inform the user of their selections and any errors that may have occurred.

**Test Case 1: Audio files are not present in directory**

Derivation: The user is responsible for providing their own audio files and selecting a directory containing them. Therefore, the possibility arises that there are no audio files in the selected directory. When this happens, a “File Not Found” exception will be thrown and needs to be handled appropriately. This is achieved by not generating any custom audio buttons in the simulator.

Implementation: An empty directory was created. Next, the configurator application is launched to set the audiopath to be the empty directory. Finally, the simulator app is launched, and the number of custom audio buttons is observed (size of the ArrayList). Should the test be successful, the number of audio buttons will be zero.

*[Include picture of Junit test]*

**Test Case 2: User does not select the appropriate directory (audio or dictionary)**

Derivation: Though like the previous test case, the selected directory may contain files that are not filled with audio, or, a mix of both audio and miscellaneous files. The program should ignore all files in an audio directory that do not have the appropriate extension (.wav) and build buttons with any acceptable files it finds.

Implementation: A directory containing various files of extensions “.wav” and “.txt” were created. The Configurator application is launched to set the audiopath to be the created directory. Finally, the simulator app is launched, and the number of custom audio buttons is observed (size of the ArrayList). Should the test be successful, the number of audio buttons will be the number of “.wav” files in the directory.

*[Include picture of Junit test]*

**Test Case 3: User attempts to play another audio file before the previous one finished**

Derivation: Because the user defines their own audio files there might particularly long ones and if they decide to press another before the previous one finishes playing then the program may throw an error. This was fixed by closing the audio file every time an audio button is pressed; however, we still felt it necessary to test this case.

Implementation: The Simulator app is launched (assuming audio files have been configured) and two audio file buttons are pressed in rapid succession. In this case we expect there to be no error thrown; furthermore, the audio from the second button should be playing.

*[Include picture of Junit test]*

**Test Case 4: Sentence is properly generated through utilization of SimpleNLG**

Derivation: The use of SimpleNLG simplifies this process greatly; however, there is still the possibility of error and therefore, we have decided to test a few examples of expected output given input.

Implementation: The words, “He”, “Chase”, and “the Dog” are chosen with the part of speech being subject, verb, and object, respectively. The features of past tense and question are also selected. The expected phrase to be generated is “Did he chase the Dog?”. We check this assertion by examining the value of the sentence label when the tester presses the play button.

*[Include picture of Junit test]*

**Test Case 5: Generated sentence is properly converted to audio through utilization of MaryTTS**

Derivation: The use of MaryTTS simplifies this process greatly; however, there is still the possibility of error and therefore, we have decided to test a few examples of expected output given input. Unfortunately, there is no way to objectively test what kind of speech is generated; however, we can test that no errors are thrown creating the speech.

Implementation: Various sentences are constructed using words in the standard dictionary and then the play button is pressed while the tests waits to see if an error is thrown. This has been tested for large length sentences as well as if no sentence is currently generated. All cases do not throw an error.

*[Include picture of Junit test]*

**Test Case 6: Prevention from injection attacks**

Derivation: Previously, text boxes were used to add words; however, this was replaced with a dictionary import system. Because this deals with plaintext we need to make sure the user cannot enter code in the dictionary and potentially corrupt the system.

Implementation: This level of protection has not been implemented yet and is currently in the stages of being developed

**Discussion:**

For a prototype release the current implemented test cases are enough. As the program currently stands it is impossible for the system to unexpectedly terminate unless a corrupt audio file is played and provides the user with visual feedback regarding choices in the Configurator application. Our current test cases cover over 90% of the application, with the exception being the corrupt audio files and injection attacks. We have chosen to leave these cases out as we consider them to be preconditions for the system and furthermore, the responsibility of the user.