Testing of the Talkbox System

Course: EECS2311

Group: #4

Alberto Mastrofrancesco, Jordan Malek, Rohan Talkad

February 22nd 2019

Summary

Talkbox provides a highly customizable virtual lexigram interface that allows speech disabled people to communicate. The Talkbox Configurator allows the user to configure the sounds and attributes of each button in the Talkbox Simulator. The two interfaces have been tested thoroughly, as have their utility classes that provide the core functionalities such as recording sound and serializing objects. We tested the major use cases for the two user interfaces under number of scenarios and ensured that the directory updated accurately as the user customized the Talkbox keys. We provided a wide range of features such as adding/removing audio sets and changing the buttons' attributes (color, text, and emoji). We also tested to make sure the software did not crash under illegal operations such as opening a corrupted or non-existent Talkbox directory. The total testing coverage from the JUnit test cases is around 45%, but there have been several manual tests to ensure the functionality of the user interface is flexible across different OS platforms such as Linux and Windows.

Classes used by Talkbox Configurator & Simulator

Major Classes	Summary
Configuration	Configuration of the Talkbox instance itself; creating TalkboxData directory and updating its contents as the user changes the buttons
ConfigSerialization	Serialization and deserialization of the TalkboxData object
ButtonConfiguration	Configuration for the button's color, text, image, and emoji
FileIO	File format verification, copying/deleting files and folders, reading/writing to text files
FileSelector	Selecting files and directories
MusicRecorder	Recording WAVE sound files from the user's microphone
MusicPlayer	Playing of WAVE sound files

Testing Implementation

Configuration is responsible for creating and updating the TalkboxData directory as the user customizes the interface. It is composed of ButtonConfiguration objects, each of which is associated with a "button_config_" folder. This contains a sound folder and a text file which has the button's attributes (its name, colour, and whether it has a sound file yet). Configuration also constructs a new instance from an old TalkboxData directory and communicates with the two user interfaces to update their appearance. It also handles the serialization of the Configuration instance each time a button configuration is changed. It also handles deserialization to load a configuration from a serialized .tbc file. We checked that the TalkboxData directory created all the button_config_ folders properly and in order. We checked whether the serialized object was up to date at all times. We also checked that, when the user changed an element in the Configurer GUI, that it reflected in the configuration's directory (e.g., adding/removing audio files,

color changes, button text changes, etc.). When a user deletes an audio set, we checked that the directory updates so that the files of the associated button configurations were deleted and the remaining ones were renamed. We also checked whether the directory was updated when the user added an audio set. We feel the testing is sufficient because it covers all of the uses of the Talkbox Configurator, the main ones being the setting up and maintenance of the Talkbox directory. Hence, our coverage is about 96% for Configuration.

ConfigSerialization handles the serialization and deserialization of Configuration objects into serialized object files. In our case, the extension of the serialized file was .tbc and stored the number of audio sets and buttons. We then deserialized the object and checked if the cast object preserved the same field attributes as it did before becoming serialized. The Talkbox Configurer and Simulator require an accurate and up-to-date serialized file, and hence this was thoroughly tested for. This makes up all the functionality for our two programs so the testing is sufficient. The testing coverage for this class is about 84%.

For **ButtonConfiguration,** we tested whether the button configurations changed according to the user input. Each ButtonConfiguration instance is linked to a "button_config_" file where the user's updates reflect in the folder's text file and contents. Thus, we tested whether calls to adding sound files and colors accurately updated the directory of the button and its text file. We feel that our testing is sufficient because all the functionalities of the class were tested. Hence, our testing coverage is almost 91% for this class.

FileIO provides various file utilities such as read/writing to text files, copying/deleting files, and file format verification. The Talkbox Configurator relies on text files to store the button states, so testing whether text files could be written and read accurately was very important. Copying and deleting were important in manipulating the Talkbox directory files (such as adding/deleting audio sets and creating new configurations). The last feature was important for verifying whether user-input audio files were of WAVE format. We also checked whether files copied and deleted properly. This makes up most of the functionality used by the Configurator and Simulator so the test cases implemented are sufficient. The testing coverage is almost 97% for FileIO.

FileSelector is the main user interface for choosing files and directories for picking audio files and Talkbox directories. We tested whether it could properly open an existing file and throw no exceptions. We checked whether it would update according to changing events (such as files being deleted). This is sufficient because it covers almost all the functionality of the class (loading sound files and picking a TalkboxData directory). The testing coverage is 83% for this class.

MusicRecorder is the main class responsible for recording audio and translating it into a WAVE format file. This is used in the Recording interface of the Configurator GUI. We tested whether the file recorded was of the length specified by when the UI's start and stop buttons were clicked and whether the recorder recorded properly. This makes up all the functionality used by the Configurator so the tets sufficient. The testing coverage is 93% for this class.

MusicPlayer plays back audio files of WAVE format. The main methods are are play(), pause(), and stop() which are called by the both the Configurator and Simulator for playing the user-imported audio files. As well, other information such as the format and length of the track are provided. We tested a sample of audio files and checked whether it correctly determined the track length and audio format. We

also checked whether the audio played at the beginning of the audio file (0:00) and ended at the end. This makes up all the functionality used by the Configurator and Simulator so we feel the testing is more than sufficient. The testing coverage is about 91% for this class.