Dr. Bil Tzerpos

EECS 2311

Authoring App Documentation

Group 6

Testing Documentation

Team Members: Tedi Papajorgji, Kevin Suarez, Jessanth Harrymanoharan, and Ibrahim Manjra

Table of Contents

Introduction

The Junit testing we decided to do covers the “Scenario” Class found in the “main.core” package exclusively. This is because the Scenario class contains all the relevant data and a bulk of the testable methods methods. The other classes simply use the Scenario class to get its variables and use its methods. Therefore by testing the Scenario class it is safe to assume we test the majority of the classes.

Methods Tested

testBufferWriteBraille()

This method has to return true if the size of the string parsed through it is less than or equal to the scenarios braille cell size and false otherwise. It also has to add the string to a buffer in the correct format. To test this we use three assertEquals methods, testing if a true or false is returned, when a valid and invalid string is entered respectively and if the format of the buffer is correct.

testBufferWriteReplay()

This method has to return true if it successfully adds the repeat commands to the buffer. To test this we use three assertEquals methods, testing if a true is returned on calling the method (which it always should) and if the format of the buffer is correct.

testBufferWriteReplayButtonAction()

This method has to return true if it successfully adds the repeat commands to the first button and the button buffer. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if the format of the buffer is correct.

testBufferWriteSectionHeader()

This method should add the Section Header to the beginning of the buffer and return true if it does so. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if the format of the buffer is correct

testBufferWriteSkip()

This method has to return true if it successfully adds the skip commands to the buffer. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if the format of the buffer is correct.

testBufferWriteSkipButtonAction()

This method has to return true if it successfully adds the skip commands to the button buffer. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if the format of the button buffer is correct.

testBufferWriteSound()

This method has to return true if it successfully adds the sound commands to the buffer. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if the format of the buffer is correct.

testBufferWriteTTS()

This method has to return true if it successfully adds the text to speech commands to the buffer. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if the format of the buffer is correct.

testBufferWriteVoice()

This method has to return true if the Integer parsed through it outside of the allowable voice choices (1 - 4) and false otherwise. It also has to add the integer to a buffer in the correct format. To test this we use three assertEquals methods, testing if a true or false is returned, when a valid and invalid string is entered respectively and if the format of the buffer is correct.

testClearBlockButtonBuffer()

This method has to return true if it successfully clears the button buffer. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if button buffer is cleared.

testClearBlockTextBuffer()

This method has to return true if it successfully clears the block text buffer. To test this we use two assertEquals methods, testing if a true is returned on calling the method (which it always should) and if block text buffer is cleared.

testCopyBlockBuffersToScenarioBuffer()

This method has to return true if it successfully adds the button buffers and block text buffers to the scenario buffer. To test this we use five major assertEquals methods, testing if a true is returned on calling the method (which it always should), and if the remaining lines of the Scenario Buffer successfully match the button buffer and block text buffer in the correct order.

testGetBlockButtonBuffer()

This method returns the contents of the button buffer, it is enough to check if a null buffer is returned on invoking the method using assertEquals.

testGetBlockTextBuffer()

This method returns the contents of the block text buffer, it is enough to check if a null buffer is returned on invoking the method using assertEquals.

testGetBrailleCells()

This method returns the number of braille cells, it is enough to check if the number is the same as the one used when the Scenario class was instantiated using assertEquals.

testGetButtons()

This method returns the number of buttons, it is enough to check if the number is the same as the one used when the Scenario class was instantiated using assertEquals.

testGetDirectory()

This method returns the directory, it is enough to check if the directory is the same as the one used when the Scenario class was instantiated using assertEquals.

testGetFileName()

This method returns the file name, it is enough to check if the name is the same as the one used when the Scenario class was instantiated using assertEquals.

testGetHeader()

This method returns the current header, it is enough to check if the value is the same as the one used when the last Scenario.bufferWriteSectionHeader() method was calledclass was instantiated using assertEquals.

testGetMissingSections()

This method returns the contents of the missingSections, it is enough to check if a null buffer is returned on invoking the method using assertEquals.

testInitialize()

testIsInMissing()

This method returns whether or not a string value is in the missingSections variable.