Authoring app testing documents

Introduction

This document was created to keep track of testing of the authoring app.

Test cases ran

```
/**
package utility.test;

import static org.junit.Assert.assertEquals;

/**
 * @author Jinho Hwang
 *
 */
public class AuthoringUtilTest {

    public void testPhraseParsing() {

    /* Exception catch should work but it does not work.
     *
    @Test(expected = IOException.class)
    public void testWrongPhraseParsing() {
        String testStr = "/~disp-cell-pins:0 11100000 pep";
        Phrase phrase = AuthoringUtil.phraseThisLine(testStr);
    }
    */


    // This test should pass because phrasing doesn't catch if it should be an integer
    public void testPhraseNonIntegerParsing() {


    public void testScenarioParsing() {

    // Testing factory scenario 2 because 1 was covered on prev test.
    @Test
    public void testScenarioPhrasingWithFactoryScenario2() {

        File file = new File("./FactoryScenarios/Scenario_2.txt");
        AuthoringUtil.phraseScenario(file);



    }

    // Testing factory scenario 3 because 1 2 was covered on prev test.
        public void testScenarioPhrasingWithFactoryScenario3() {


        public void authroingAppTest() {


}
```

From testPhrasePrasing to testScenarioPhrasingWithFactoryScenario3 test parseability (translatablilty of the scneario file sentences into valid scneario parses). These tests are given either well-formed or not well-formed scenario file as an input, and are tested if they give an error or an exception. These are derived to test the core utility, AuthoringUtil, that parses a scenario File by dividing it to individual lines, and checking the validity of each lines.

"authoringAppTest" executes the GUI of the program and testing must be done manually. This

creates possibilities that manual test would not catch all the cases of possibilities of errors. This case was derived to test the whole program. This test case pops the actual program up to test, since GUI part is unable to test automatically, but manually.

Test cases Coverage

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Enamel	58.5 %	4,620	3,280	7,900
src	58.5 %	4,620	3,280	7,900
enamel	36.4 %	972	1,697	2,669
ScenarioParser.java	14.5 %	181	1,071	1,252
ScenarioParser	14.5 %	181	1,071	1,252
VisualPlayer.java	57.0 %	273	206	479
VisualPlayer	37.2 %	83	140	223
AudioPlayer.java	57.6 %	265	195	460
AudioPlayer	41.1 %	78	112	190
BrailleCell.java	57.7 %	177	130	307
BrailleCell	57.7 %	177	130	307
Player.java	44.4 %	76	95	171
Player	46.9 %	61	69	130
utility	54.0 %	1,108	942	2,050
AuthoringUtil.java	49.3 %	827	851	1,678
AuthoringUtil	49.3 %	827	851	1,678
Phrase.java	72.0 %	226	88	314
Phrase	72.0 %	226	88	314
Language.java	94.8 %	55	3	58
Language	94.8 %	55	3	58
gui.controllers	79.8 %	1,828	464	2,292
CreateCommandPopUpBoxControlle	81.4 %	443	101	544
ScenarioEditorController.java	80.6 %	416	100	516
ScenarioMakerController.java	82.8 %	439	91	530
VoiceRecorderController.java	83.1 %	399	81	480
MainMenuController.java	38.6 %	22	35	57
TwoChoiceBoxController.java	60.0 %	42	28	70
ErrorListReportPopUpBoxController.ja	76.7 %	46	14	60
TextAnswerBoxController.java	60.0 %	21	14	35
startProgram	0.0 %	0	75	75
AuthoringApp.java	0.0 %	0	75	75
gui.layouts	91.1 %	658	64	722
ScenarioMaker.java	92.7 %	139	11	150
VoiceRecorder.java	85.9 %	67	11	78
ScenarioEditor.java	85.9 %	61	10	71
CreateCommandPopUpBox.java	93.0 %	93	7	100
ErrorListReportPopUpBox.java	91.2 %	73	7	80
TextAnswerBox.java	90.5 %	67	7	74
TwoChoiceBox.java	93.2 %	82	6	88
MainMenu.java	93.8 %	76	5	81
utility.test	60.7 %	54	35	89
AuthoringUtilTest.java	60.7 %	54	35	89

The test cases cover 58.5% of the whole program including the starter code.

Sufficiency of the test

The tests above is sufficient to cover most of the cases due to following reasons:

1. The top 5 test cases cover the core of scenario parsing.
   - These test cases test if a scenario file is well-formed or not, and test if the well-formed scenarios are valid.
2. The last test case covers the actual functionality of the program.
   - Since the core functionality, the prasing of the given scenario file, is tested, only thing left to test is the interaction between windows on gui; gui only needs to be tested if each button does its functionality as why they were created.