# Authoring app testing documents

### Introduction

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

#### Test cases ran

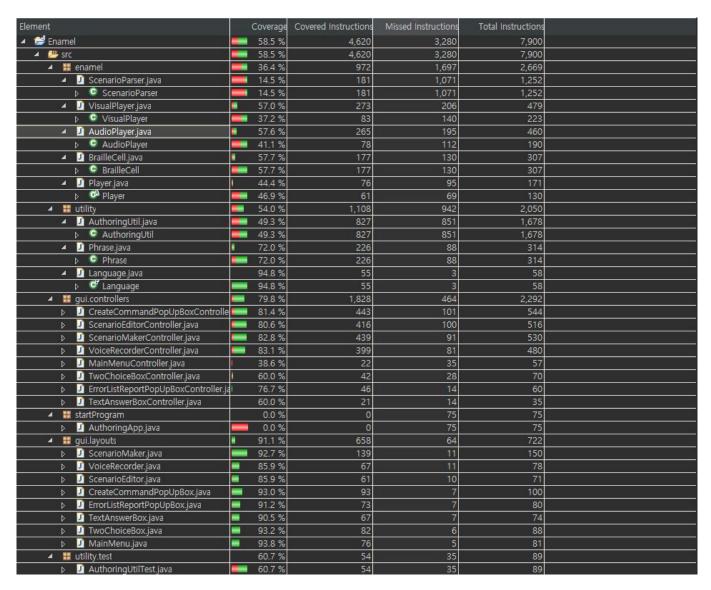
```
1⊕ /**...
4  <mark>package</mark> utility.test;
 6⊕ import static org.junit.Assert.assertEquals;[]
      * @author Jinho Hwang
    public class AuthoringUtilTest {
         public void testPhraseParsing() {[
1620
         @Test(expected = IOException.class)
         public void testWrongPhraseParsing() {
             String testStr = "/~disp-cell-pins:0 11100000 gep";
Phrase phrase = AuthoringUtil.phraseThisLine(testStr);
         // This test should pass because phrasing doesn't catch if it should be an integer
1740
         public void testPhraseNonIntegerParsing() {
181⊕
         public void testScenarioParsing() {[]
         // Testing factory scenario 2 because 1 was covered on prev test.
249
         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.
261⊕
             public void testScenarioPhrasingWithFactoryScenario3() {
271⊕
             public void authroingAppTest() {[]
```

From testPhrasePrasing to testScenarioPhrasingWithFactoryScenario3 test parseability ( translatability 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**



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.