Testing Document

Treasure Box Braille: Authoring App

Group 19: Connor Ahearn, Micah Arndt, and Kevin Chan

Table of Contents

1.0 Document Overview

At the time this document is being published, the focus of the project has been making a GUI that can create Scenarios that can be interpreted by the Treasure Braille Box (TBB). Our group focus before the midterm is to create an application that can create the factory scenarios provided by the original repository.

All text file creation is currently based on the assumption that the device used has only one Braille Cell and 4 Buttons. In the future different devices will be accounted for, but for now this is our scope.

2.0 Document format

The format of the document in section 4 is structured as follows:

- The class being discussed
- Summary of the class' purpose
    - Function being discussed
    - Summary
    - Test cases implemented for said class / function
        - Also listing why these cases were created
    - Why the cases listed above are sufficient
- EcLemma Summary

3.0 Provided Classes

The provided classes for this project are not being tested because we were told to assume they were fully functioning and tested at the beginning of the project. All the classes we hold this assumption for are listed below:

a. BrailleCell.java

b. Player.java

c. ScenarioParser.java

d. ToyAuthoring.java

e. VisualPlayer.java

4.0 Implemented Classes

This section highlights the implemented classes and the constructors and methods contained within. It describes the test used and test coverage.

4.1 Printer.java

The printer class is the portion of the application tasked with printing the information from the GUI onto a text file. It does this by receiving information in the form of *Block* objects (**Block.java** listed above). It then stores the information in a collection until the **print()** method is called, which is when the information is printed to the text file. The text file it produces is based on the examples given for the simulator to run.

The class itself doesn't serve a complex purpose. It has one specific task, and as such only has a public constructor and 2 public methods.

Tests for this class are created under 2 different cases:

- Basic Functionality > These tests ensure that the basic functions of the class work.
- Advanced Features > These tests ensure advanced features included with the printer's functionality operate correctly.

Printer() (Original Constructor)

- Creates a new Printer object
    - Takes a filename for the new file, as well as how many cells and buttons the new scenario will use
    - Places the information required for the first 3 lines of the text file into the **lines** collection

The constructor was tested with the **testInitial1()** test in **testPrinter.java**. It printed the correct default text for the cells and buttons at the top.

Printer() (Simplified Constructor)

Exact same as original constructor but assumes *buttons* to be 4 and *cells* to be 1. This aligns with our midterm focus. It is tested in **testOldBlock1()** and **testOldBlock2()** by being called instead of the original constructor.

addBlockList() (Method)

Takes an ArrayList of Block objects as a parameter and calls addBlock() for each block in the provided list. addBlock() adds all the required information of an individual block into strings to be printed on the file. Many test cases test the functionality of addBlockList().

testOldBlock1() – Tests one standard block using the old constructor

testOldBlock2() – Tests 2 standard blocks using old block constructor

testOldBlockList() (1 and 2) – Deprecated tests kept for redundancy for standard blocks

testNoBlock(), testNoCorrect(), testNoIncorrect() – Tests blocks with blank block parameter

print() (Method)

Takes the private ArrayList of lines to be printed and prints them to a file with the provided name. This method is tested in **every test** by a Scanner Object in each test case. By default, this method is always tested.

Sound Tags (Feature)

These tests check that when arrow brackets are used to add sound effects to a scenario it works correctly. Incorrect usage throws an OddSpecialCharacter exception.

testSoundTag() (1-7) – Testing various cases where arrow tags are correctly / incorrectly used in scenario creation

Pin Tags (Feature)

These tests check that when ** are used, pins are set accordingly in the scenario. Incorrect usage throws an OddSpecialCharacter exception.

testPinTag() (1-3) – Testing various cases where * are used correctly / incorrectly in scenario creation

- The printer class has 94.4% test coverage.

4.2 Block.java

Datatype used for collecting user input together for each section of the scenario. Uses public String and int fields to store the data.

Block() (Original Constructor)

- Stores all information provided to Constructor parameters in fields

This constructor is tested with the **testConstructor1(), testConstructor2()**, **testConstructor3()**, **testConstructor4()**, **testConstructor5()** test, which runs the constructor with some test input and checks the fields of the new Block. They also check all cases of exception throwing.

Block() (Simple Constructor)

Calls the original constructor with the buttonsUsed parameter set to 2. Tested in the **testSimpleConstructor()** test. Since its calling the other constructor, we can assume it functions correctly in abstract cases.

The Block class has 100% test coverage.

4.2.1 InvalidBlockException.java

If illegal arguments are passed to the block constructor, an **InvalidBlockException()** will be thrown.

4.3 BrailleInterpreter.java

Object that converts characters to braille pin binary equivalent. Braille pin equivalents are stored and retrieved from a HashMap, and the pin equivalents were based on the BrailleBox simulator code.

BrailleInterpreter() (Constructor)

- Adds all Braille Possibilities to the HashMap, initializing its field

This class is tested by running the constructor in a JUnit test **testConstructor()**. This is an @After test, so its done before any other unit tests.

getPins() (Method)

Takes a character as a parameter and returns a Binary string that corresponds to the pin positions for the Braille equivalent. Currently only alphabetical and space characters are permitted. The tests, **testGetPins1()** and **testGetPins2()** check that both correct and incorrect inputs and respond accordingly, filling the requirements of the testing.

- The Braille Interpreter class has 100% test coverage.

4.3.1 InvalidCellException.java

If a character is not stored in the HashMap of the Braille Interpreter class, then a **InvalidCellException()** will be thrown.

4.4 ScenarioCreator.java

ScenarioCreator.java

GUI class that allows the user to create Scenarios. At the time of publishing, this class was not capable of actually using its input, so **testInit()** simply ran the constructor and used its **.launch()** method.

• The Scenario Creator class has 97.9% test coverage. Proper GUI testing wasn't taught to us, so this test % was mostly through manual testing initiated by a unit test.

4.5 AudioPlayer.java

AudioPlayer.java

Audio Player was a provided class that inherited from player. Since the majority of this class' functions were reading off the Scenario Parser's output. Since Scenario Parser is a provided class, its assumed its output is correct. Since this is the case, and Audio Player creates audio from this output, it was tested via manual testing by our team, and there are no problems left that we could find.

4.6 Loader2.java

Loader2.java

Utility class used for converting completed Scenario files into Blocks that the ScenarioCreator can fill its fields with. This class is designed with limited functionality, it does not support some of the special functions that the printer does. If Advanced Features are used such as custom sound and pin settings, it may either delete some text portions or not load the file at all.

The class is designed as a Utility class, and only has one public function, **load(File file)**. This takes the file to be loaded and returns a BlockList containing all the scenario information. Data not traditionally stored in Blocks are stored in a "Fake" block at the beginning of the ArrayList. This includes the amount of Braille Cells used by the Scenario and how many buttons the device uses.

The Loader is tested by the **testOneBlock** and **testTwoBlocks** unit tests. OneBlock tests just the basic functionality with a single block, TwoBlocks checks with a more complicated scenario with more blocks. Both these tests use the Printer class to create the Scenario file, so we know the Application's ecosystem works in a full cycle, so long as pre-conditions for the loader are met.

• The Loader2 class has 81.2% test coverage.