Testing Document

Connor Ahearn, Micah Arndt and Kevin Chan

Table of Contents

1. Outline

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 BrailleBox. 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. 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. 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. Implemented Classes

a. 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.

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 **test1Block()** by being called instead of the original constructor.

addBlock() (Method)

Takes Block objects as a parameter to be added to the file. Appends the file to be printed with the information for one more block. This is done via a private ArrayList within the Printer class storing each line it will print in order. addBlock() adds the lines required.

This method is tested with the **test1Block()** and **test2Block()** tests, which test adding one and two blocks respectively. If adding one block works, and adding another block after that works then by Induction adding infinitely many blocks should hold up.

addBlockList() (Method)

Takes an ArrayList of Block objects as a parameter and calls addBlock() for each block in the provided list. This is done by an enhanced for loop calling addBlock(). Since we know addBlock() is tested, all that

has to be checked is that the loop never goes out of bounds or anything. **testBlockList1()** and **testBlockList2()** both do similar tests as **test1Block()** and **test2Block()**, but by using addBlockList().

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.

The printer class has 98.6% test coverage.

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. Throws am **InvalidBlockException()** if the information provided is not valid.

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.

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. Throws a **InvalidCellException()** if the character isn't stored in the HashMap. **testGetPins1()** and **testGetPins2()** test both correct and incorrect inputs and respond accordingly, filling the requirements of the testing.

This class has 100% coverage.

ScenarioCreator.java

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

This class had a lot of manual tests with user inputs.

This class has 76.1% coverage.

AudioPlayer.java

Audio alternative version to VisualPlayer.java in provided code. Not actually capable of testing output since its audio based, manual tests with test files were used to test this file.