Design Document

Treasure Box Braille: Authoring App

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

Table of Contents

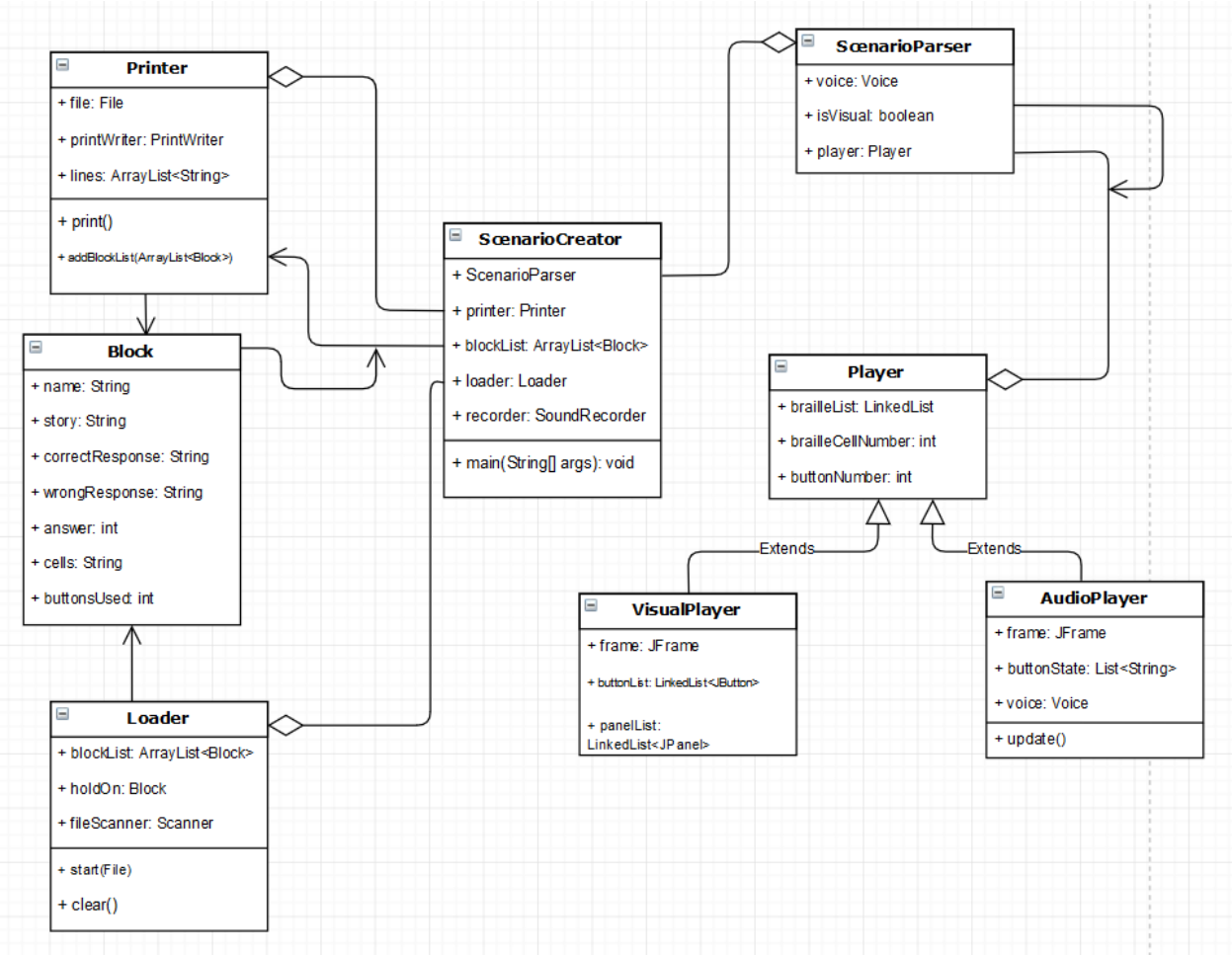
1.0 Introduction

The purpose of this document is to describe the way the application operates at a high level. This is done using a class diagram and a sequence diagram.

The runtime of the application is much easier to understand if you look at it from the perspective of the **ScenarioCreator**. ScenarioCreator is the class with the main method, it has all the GUI's for creating scenarios along with code to launch testing scenarios in our app. From start-up to close, if the Authoring App is running, ScenarioCreator is running. Through Aggregation, ScenarioCreator (As seen in the class diagram) uses every class in the application, although not always directly.

2.0 Class Diagram

The class diagram for the Authoring app can be seen below:



While looking at the diagram above, its important to note that some classes didn't make it onto the diagram. These classes were omitted to make the diagram easier to understand. These classes were deemed less important than the ones you're seeing above. The omitted classes are listed below:

- BrailleCell.java
- BrailleCellPanel.java
- BrailleInterpreter.java
- SoundRecorder.java
- ToyAuthoring.java
- All Exceptions used throughout the program

Most of these classes are unessential to understand at a high level or are straight forward enough that they don't need to be in the diagram. The focus of the program can be seen in 3 main areas:

ScenarioCreator – The runtime of the entire program runs through this class
Block – The data type universally used throughout the program's working parts
ScenarioParser – The Testing of scenarios is all done through this program

2.1 ScenarioCreator

ScenarioCreator was originally intended to be the GUI used to create scenarios, but it evolved into being the central class of the application. All scenarios are written using the ScenarioCreator, while advanced features and functions are completed via aggregation of other classes.

Printer.java is used to create and save scenario files.
Loader.java is used to load previously saved scenario files back into ScenarioCreator.
SoundRecorder.java is used for recording sound that is to be added to the scenario.
AudioPlayer.java is used to provide audio accessibility for the braille cell simulator.

2.2 Blocks

Blocks, are the building blocks of a scenario in the application and they allow users to effortlessly control the flow of the scenario. Since the braille cell pins are automatically reset at the end of a block, each block acts as a small section of the scenario. Therefore, within each block, users can set a letter or a word to be displayed on the braille cell(s), and the buttons can serve as multiple choice answers or to simply skip to the next section if you are introducing a new word to a student.

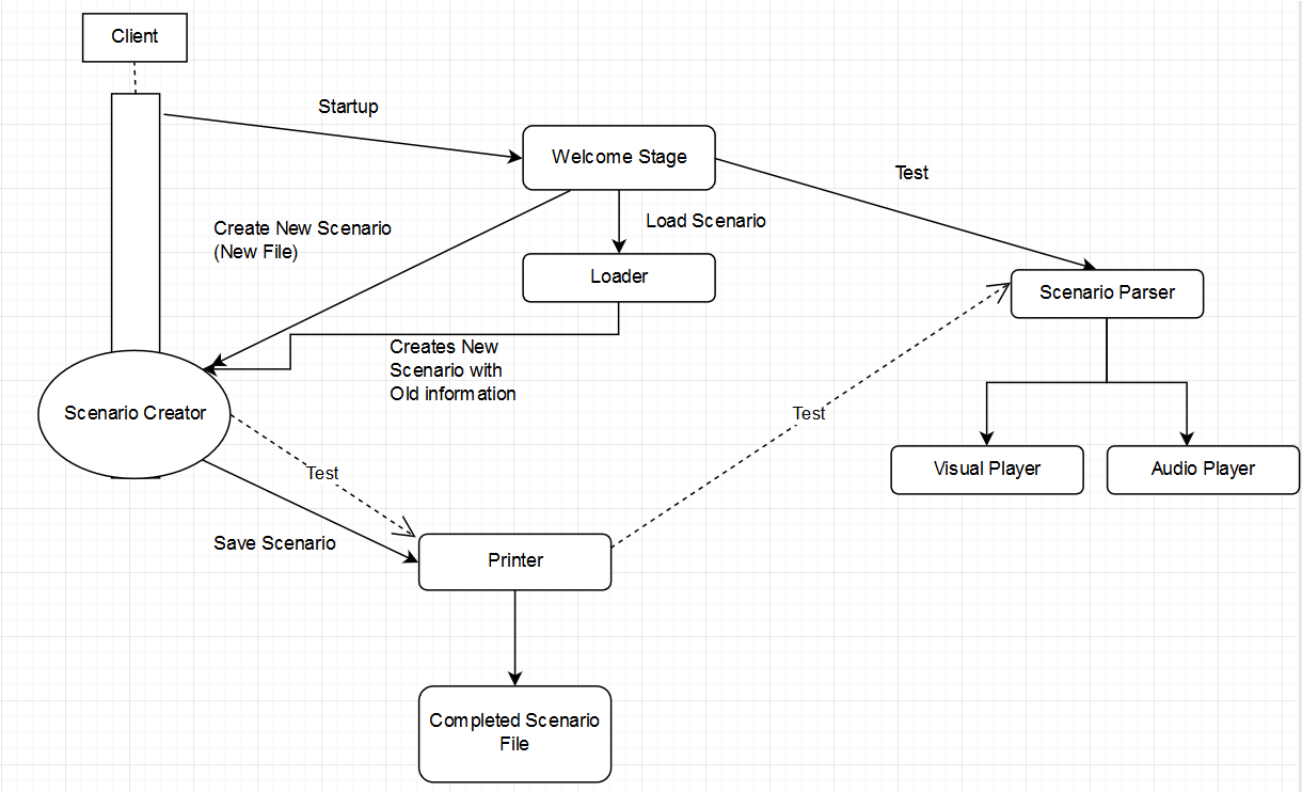
This makes each scenario just a collection of blocks played in sequence by the Treasure Braille Box. Blocks act as the standard datatype in our program, which makes transferring collected user input between the 3 main classes that use it (ScenarioCreater, Loader and Printer) much easier in practice.

2.3 Scenario Parser

ScenarioParser is central to testing scenarios. This code was provided to us by the professor.

3.0 Sequence Diagram

The sequence diagram of the Authoring App can be seen below.



In the diagram above, the rectangle over the Scenario Creator represents the portion of the application that runs through that class but isn't necessarily involved in the scenario Creation process. It takes you to the Welcome Stage GUI where you're given 3 options: New file, Edit file and Test file. Each of these options are arrows branching off the Welcome Stage to the various areas of the program.

4.0 Individual Classes

This section provides a detailed description of each class as well as important methods contained within.

4.1 AudioPlayer.Java

Audio Player is the class designed for visually impaired users to test scenarios. It does this by making use of the systems screen reader to describe the visual elements of the GUI and changes that occur in the braille cell(s) simulator.

This class contains methods used during testing, and is responsible for describing which pins are being displayed on the Treasure Braille Box.

4.2 Block.Java

Blocks are the standard data type throughout the application. It holds all the required information for a section of a scenario, from the number of braille cells used and buttons available, to all of the user input. This data type is then communicated between the Printer, Loader2 and ScenarioCreator.

The block() method found in this class works together with the saveSection() method found in the ScenarioCreator class to retrieve and store user input into blocks.

4.3 BrailleCell.java

This was a provided class with the project, it helps display braille on the Visual Player. It uses the HashMap function to allocate a letter of the alphabet to a 8-digit binary string. Methods in this class are used to determine which pins are displayed on the braille cell.

4.4 BrailleCellPanel.java

This was a provided class with the project, it helps display braille on the Visual Player.

4.5 BrailleInterpreter.java

The Braille Interpreter takes the pin settings on the braille cell / characters and translates to the opposite notation. Braille settings are saved as binary strings.

4.6 CorruptFileException.java

Corrupt File Exceptions were solely created for loading scenarios. If a scenario file is loaded and any part of the file isn't recognized, this exception is thrown, and the file isn't loaded.

The Loader is specifically designed to load ***our*** authoring app's scenarios, files created with other applications may not load properly as it may contain characters or formatting that cannot be interpreted. The message with the exception will describe the cause.

4.7 InvalidBlockException.java

Invalid Block Exceptions are thrown when invalid data has been passed to a block constructor. This includes passing empty required fields, non-alphabetical characters to the braille cell and non-numeric characters to fields that require an integer value. The message with the exception describes the cause.

4.8 InvalidCellException.java

Invalid Cell Exceptions are thrown when the Braille Interpreter is used to translate a character that isn't in the alphabet into Braille.

4.9 Loader2.java

Loader2 is the second iteration of the loader, and the one used in the final build for loading previously saved scenarios. It works for most scenarios, but some of the advanced functions related to custom sound and pin settings can be lost at times. The only function public for this utility class is called load() and it takes a File parameter.

The loadBlock() method is used to take the saved scenario text file and convert it back into blocks so that it can be loaded into the Scenario Creator.

4.10 OddSpecialCharacterException.java

Odd Special Character Exceptions are thrown by the printer when scenarios have an odd amount of special characters in a story section. Since special characters are used for advanced functions (<, > and *) if there's an odd amount, the printer can't decipher what the user wants to happen in terms of custom sounds or pin settings.

4.11 Player.java

This was a provided class that acts as the parent class of both Visual Player and Audio Player. This class provides methods for simulating a braille cell. It contains methods to control when pins are raised and when they are to be lowered or cleared.

4.12 Printer.java

The printer class is the class that converts a block list to a full scenario file. It's called by the Scenario Creator, and the information put into the block list that's passed to it are from the user input.

The only methods called by ScenarioCreator are the constructor, addBlockList() and print(). The addBlockList method is used to send blocks to the printer, while the print method is called to print the information from the blocks to a file. Both methods are called when the user chooses to save the scenario.

4.13 ScenarioCreator.java

Scenario Creator is the main class of the program. All classes are invoked in some way by Scenario Creator's main method. Scenario Creator also houses the authoring application itself, where users can enter data to be printed into scenario files or load information from old ones they have made so that they can edit the scenario using the software. The graphical user interface is built using JavaFX, as it supports more customization options than java SWING.

Most of the methods are used to create the various windows, GUI elements and dialogs. The method copySoundFile() is used to import .wav sound files into the scenario. The runTest() method is used together with the ScenarioParser class to test saved scenarios using the simulated braille cell(s).

4.14 ScenarioParser.java

This was a provide class and acts as the point from which the Player classes are called from. The methods are used to interpret key phrases in the scenario file.

4.15 SoundRecorder.java

This class is responsible for handling new sound recordings made from within the application. Sound files can also be imported as long as they are of the .wav format. We recommend using another (better) application for adding audio to scenarios, like Audacity, but the feature for sound recording is here.

The sound recorder requires a recording length to be set prior to beginning recording. After the constructor is called, the record() function provides sound recording capabilities.

4.16 ToyAuthoring.java

This was a provided class designed to be a testing method for Scenario Parser.

4.17 VisualPlayer.java

This was a provided class used to visualize how the scenario plays out using the braille cell(s) simulator.

4.0 Maintenance Scenario

Maintaining the authoring app is designed to be simple. This is accomplished through a variety of means, from the organization and method / variable names used throughout the application to the use of JavaDoc comments within the code. Additionally, the included documentation (Design Document, Requirements Document, Testing Document and User Manual) makes the application easy to understand at a high level.

The modular design of the application is meant to allow for easy maintenance, whether it is to fix errors that occur or to add new features into the software. A majority of the graphical user interface can be altered from within the ScenarioCreator class, as comments are provided to help identify what element of the GUI each part of the code represents. Additionally, the way recording sound is handled and the type of audio file accepted can be changed through the Sound Recorder class. The format of the scenario can be maintained through the Printer class. The maintenance of more complicated features such as testing scenarios will involve multiple classes, specifically the Player, Visual Player, Audio Player and Scenario Parser.