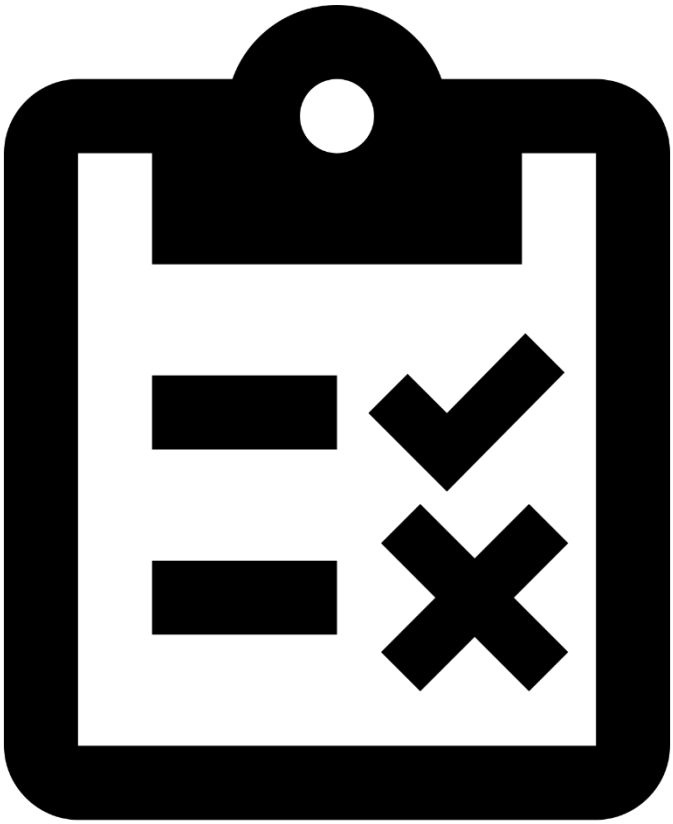


Design Documents

Jinho Hwang

Table of Content

Contents

1. Introduction

This section explains the purpose and scope of this design document of Authoring app

1.1 Purpose of this Design Document

The following document is created to:

1. explain the high-level structure of Authoring app system
2. show important classes / methods
3. show how each class / method interact with each other.
4. show important objects created at runtime and how they connected to each other

1.2 Scope

The scope of the design document is to show help fellow developers to easily understand how the program works with aids of a structured document. This document will cover the general structure and interaction between classes and methods and object on runtime. The specification such as how exactly each method is implemented are omitted as they are unimportant to show a big picture for Authoring app.

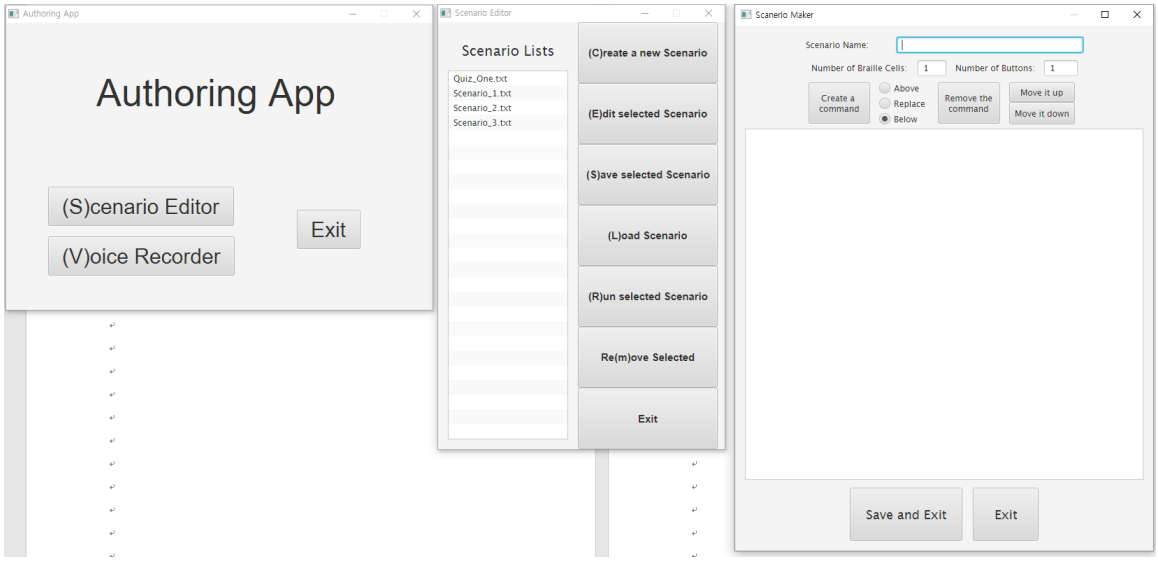
1.3 Context

The Authoring app is a visually-impaired educator assistant application which give an easy way for the educators to create scenarios to provide an entertaining braille learning experience to the students the educators are teaching.

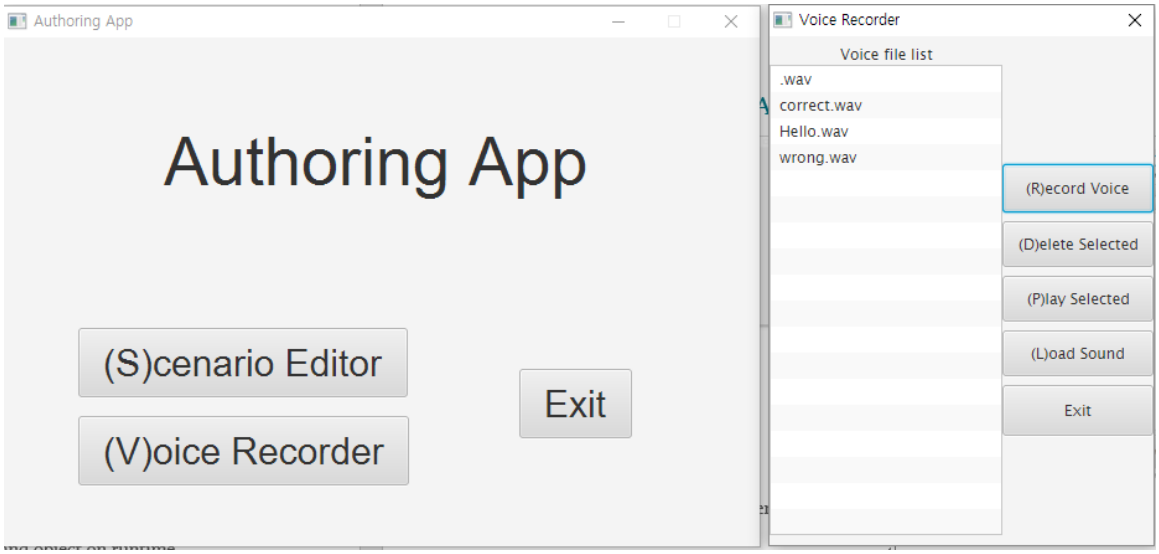
1.4 Functions of Authoring app

The Authoring app allows user to create, edit, load, save, and simulate Scenario files. A Scenario file is a case-study made by user to use it to educate students about braille interactively in the way user wants. The app also supports basic sound file manipulation such as recording, playing, loading into the program and saving into different local directory. The TTS is not directly supported by this program, but by external program such as NVDA.

2. A brief glance of Authoring app



Main Menu, Scenario Editor and Scenario Maker.



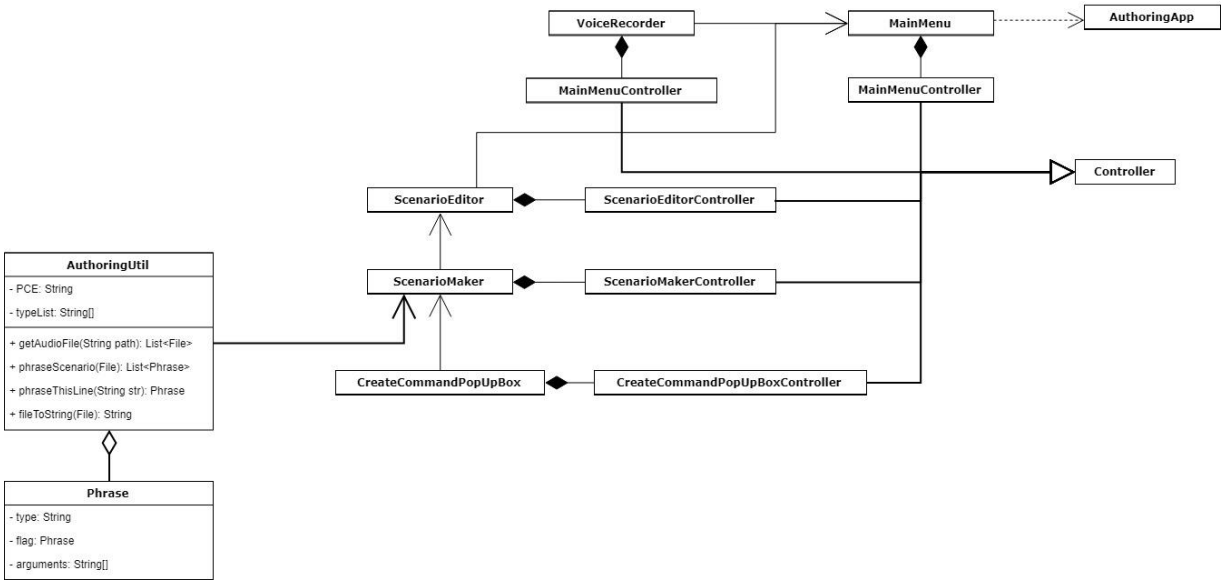
Main Menu and Voice Recorder for Scenario use.

For an explicit explanation of each functions of program and its usage, please refer to user manual: https://github.com/DandyEnders/Braille-Project/blob/refactorBranch/Documents/Final/EECS_2311_AuthoringApp_UserManual_Final_WINDOWS.pdf

3. Internal Design of Authoring app

This section explains the internal details of Authoring app.

3.1 UML Diagram



Authoring app is made in a Model-View-Controller structure where Model represents data passed by internal process, View represents the visuals of what users actually see (or hear, generally interaction), and Controller that controls Model and View according to input given from user from view and internal calculation from Model.

Each window on the Authoring app has their own View / Controller. View and Controller of each window is inherited from a class “View” and “Controller” respectively, thus minimizing amount of work to create a new window. For example, Scenario Editor menu has ScnearioEditor as a view, ScenarioEditorController as controller, and Model integrated with controller.

AuthoringUtil module holds utility classes responsible of providing utility methods such as loading audio files from a path, parsing Scenario file string, converting file to string. There are several utilities:

1. **FileUtil**

This utility class is responsible of dealing with file. The class provides a way to import text files, reading text from files and importing sound file.

2. **ErrorUtil**

This class is responsible for dealing with errors. The class has methods that can be called to warn user of something. Each different kind of method allows different format of error reporting. The method that takes a simple title and error message is used most often.

3. **LoggerUtil**

This class is responsible of logging everything into a text This class has methods to take log message and put time stamp to export it in a log folder.

4. **ViewUtil**

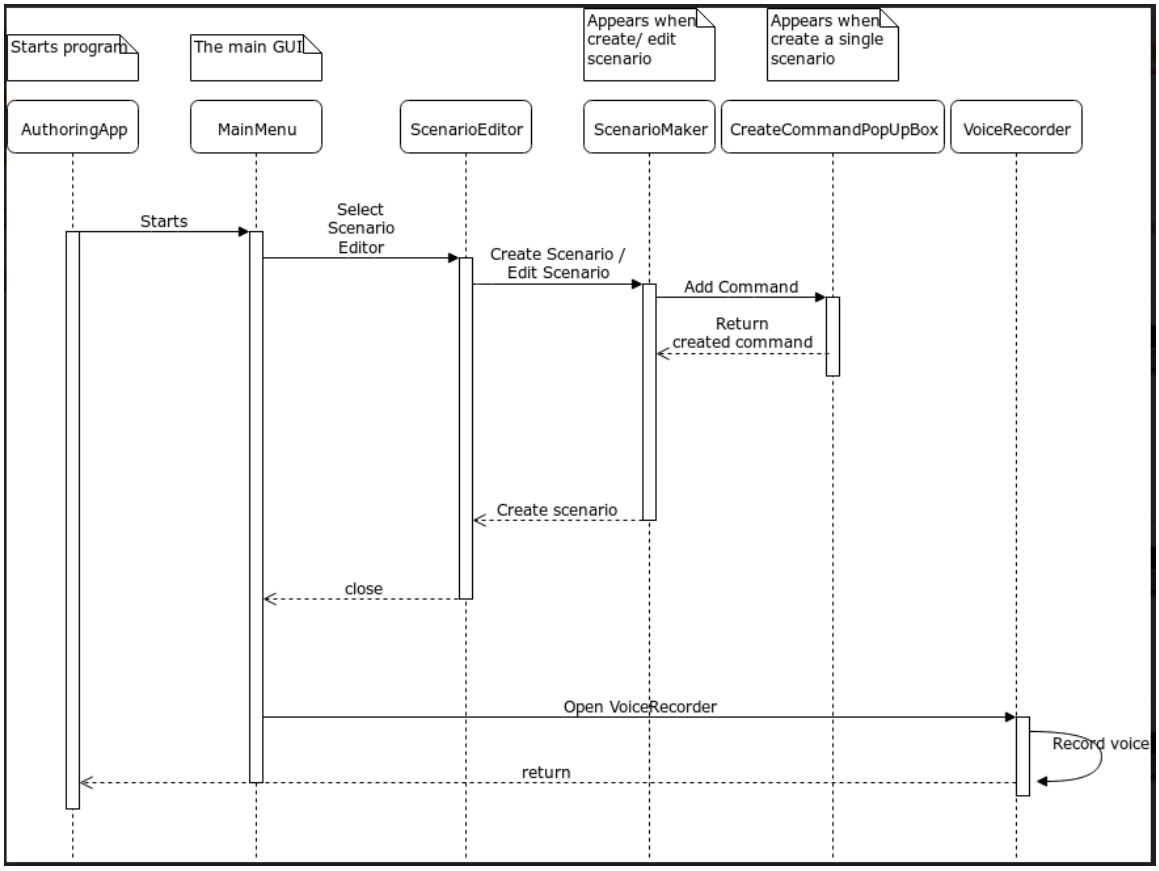
This class is responsible for adding key and mouse loggers for each component in GUI that user is interacting with.

5. **ParsingUtil**

This class is responsible for parsing Scenario string imported or made by user. It is used to validate if the string is in a right format to execute in a braille simulator. To simplify the task of parsing the text, there is Phrase class.

The window controller classes use the utility class' methods. Letting all the utility feature implemented by a set of utility class was not a bad idea since many controllers use many same methods among each class.

3.2 Sequence Diagram



The figure above is a simple object interaction simplified and visualized.

AuthoringApp – main method

AuthoringApp class contains main class, thus initiating the program to start. This class calls MainMenu object to be called, which then shows main menu window

.

MainMenu

User can decide either to access ScenarioEditor by pressing Scenario Editor button on the main menu, or Voice Recorder button to edit sounds.

ScenarioEditor

Once the user decides to edit scenario and open scenario editor, the user can either:

1. **Create scenario**

When user create scenario, scenario maker is called with an empty file internally, with a preassigned cell value and button value to avoid possible exception throws which caused by empty field.

2. **Edit scenario**

When user edit scenario, scenario maker is called with the selected Scenario file on the list, with loaded cell and button number, name of the scenario, and command list loaded. If user did not select any Scenario file, then nothing happens.

3. **Save scenario**

Saves the Scenario file in a custom directory other then one designated by Authoring app. If user did not select any Scenario file, then nothing happens.

4. **Simulate scenario**

This calls a popup object with two choice boxes that allows user to choose either to play a visual simulator or audio simulator. Either choices yield an Enamel object which is a simulator for scenario file, with a different setting.

5. **Exit scenario editor**

This will close the scenario editor, going back to main menu.

ScenarioMaker

Once user gets into Scenario Maker object call, they can either:

1. **Create command**

This calls command pop up box which allows user to select kind of command, and its argument.

2. **Move command**

This allows user to move the ordering of individual commands up and down. This changes command list.

3. **Remove command**

This allows user to remove a command from command list. This changes command list.

4. **Save and Exit**

Once the Save and Exit button is clicked, the parse utility class is called to check if user has correctly created Scenario file with a correct format. If the scenario list represents an incorrect Scenario format, it pops error with its description and stops exiting. If the command list represents a correctly formatted Scenario file, then it saves the list as a file and stores it on Scenario Editor list.

5. **Exit**

All progress is discarded and exits Scenario Maker.

CommandMaker (CommandPopUpBox)

Once the user access Command maker object, the user either can:

1. **Select kind of command**

A list of commands are shown to user, letting user to select them. Once the command is selected, argument field is enabled according to the command.

2. **Input arguments**

With different kinds of command, different argument fields are enabled for users to input.

3. **Save**

User then can save this command to input into the ScenarioMaker list.

4. **Discard**

User can discard the command / argument setting.