Design Document

EECS 2311 Z – *Software Development Project*

Group 15: Aya Abu Allan, Gianluca Corvinelli, Mark Savin

Table of Contents

1. General Information

1.1 Purpose ...…….…… 3

1.2 Description of the Authoring App ...…… 3

2. .High Level Design Overview

2.1 Unique Design Implementations ……... 4

2.2 Goal and Rational of Design Strategy ..…….... 4

2.3 UML Class Diagram ...……..……..…….. 6

2.4 Important Classes and Methods …… 7

3. Important Runtime Scenarios

3.1 Creating a Teaching Scenario ...…… 10

3.2 Editing a Teaching Scenario ,,,……...……. 12

3.3 Sample Teaching Scenarios …….. 14

3.4 Loading a Teaching Scenario ……..…….. 15

4. Maintenance Scenarios

4.1 Overall Maintenance Strategy ……... 16

4.2 Design Improvements for Long Term Sustainability …… 16

Section 1 - General Information

1.1: *Purpose*

The purpose of this document is to provide a description of the high-level organization of the *Authoring App* and a detailed rational behind the implemented design. It will also cover individual components through informative diagrams and discuss the system's maintainability. This *Design Document* is intended for the use of an experienced developer that is unfamiliar with the system to obtain a clear idea of the system's design.

1.2: *Description of the Authoring App*

The *Authoring App* is a tool for visually impaired/capable users to teach braille to visually impaired students. The system is designed to maximize flexibility without compromising simplicity of usage. It will allow teachers to program a Treasure Box Braille (TBB) device to fabricate lessons without any knowledge in programming. Closing the gap between programming aspect and the lesson-making aspect, the teacher is able to perform their duties more efficiently.

More specifically, the system will allow the user to create lessons called '*scenarios*' and specialize them however they wish to meet their individual purposes. They can also edit or load an existing scenario as well as refer to our sample scenarios which are provided as guidelines for how to use functions present in the app. The system will also be able to provide the option of using an external screen reader that is responsible for delivering audio instructions to the user for navigation purposes, if the user themselves is visually-impaired.

Section 2 – High Level Design Overview

The software design pattern that *Authoring App* uses is MVC. The model and the controller are written in java and the view is written in xml. The GUI interface language that is used throughout most of the app is FXML. There are very small instances of Swing being used for notifications throughout the app. Every distinct and new window of the application has a view and a controller. The main models that are used throughout the program are primarily SceneAA, ScenarioAA, and SceneAAWriter. These models will be explained in further detail in Section 5 (Important Classes and Methods). Separate from the MVC components of *Authoring App* the *MainAuthoringApp* class is meant for launching the application.

2.1 *Unique Design Implementations*

Authoring App presents a unique way for users to create the flow of scenarios. The core design of a teaching scenario consists of a list of scenes. A scene is a specific instance of an interaction between the teacher and the student. This interaction can consist of anything from simply asking a question all the way to creating a quiz with feedback. Users can create, delete, and move scenes. This gives the user as much freedom as possible to express their creativity while keeping the simplistic look and feel of a teaching scenario.

Another interesting design concept that *Authoring App* consists of is the visual capability setting. It was introduced in version 2.0 of the application and is in the early stages of its development. However, upon first opening the application the user is prompted with the option to view teaching scenarios differently based on their visual capability. This setting is stored and remembered throughout the use of the application.

Authoring App's design even takes into consideration improving overall time and storage efficiency. Sure, this may not matter for small teaching scenarios where very minor changes are made. However, since we do not limit our users on the size of a scenario, it will show when much larger teaching scenarios are built and maintained.

The specific efficiency being referred to is the use of serialization in conjunction with ScenarioAA objects created. When the user makes a bundle of changes it does not automatically search and modify the text file but instead will wait until the user is finished building/editing the scenario and then save the object separate as a binary file. This saves a lot of processing power when the user must load and edit a teaching scenario. Instead of looping through searching where things are in a text file it is simply deserialized and manipulated as an object. Then once the user is finished manipulating the object (by editing their scenario) it will write the changes only once at the end.

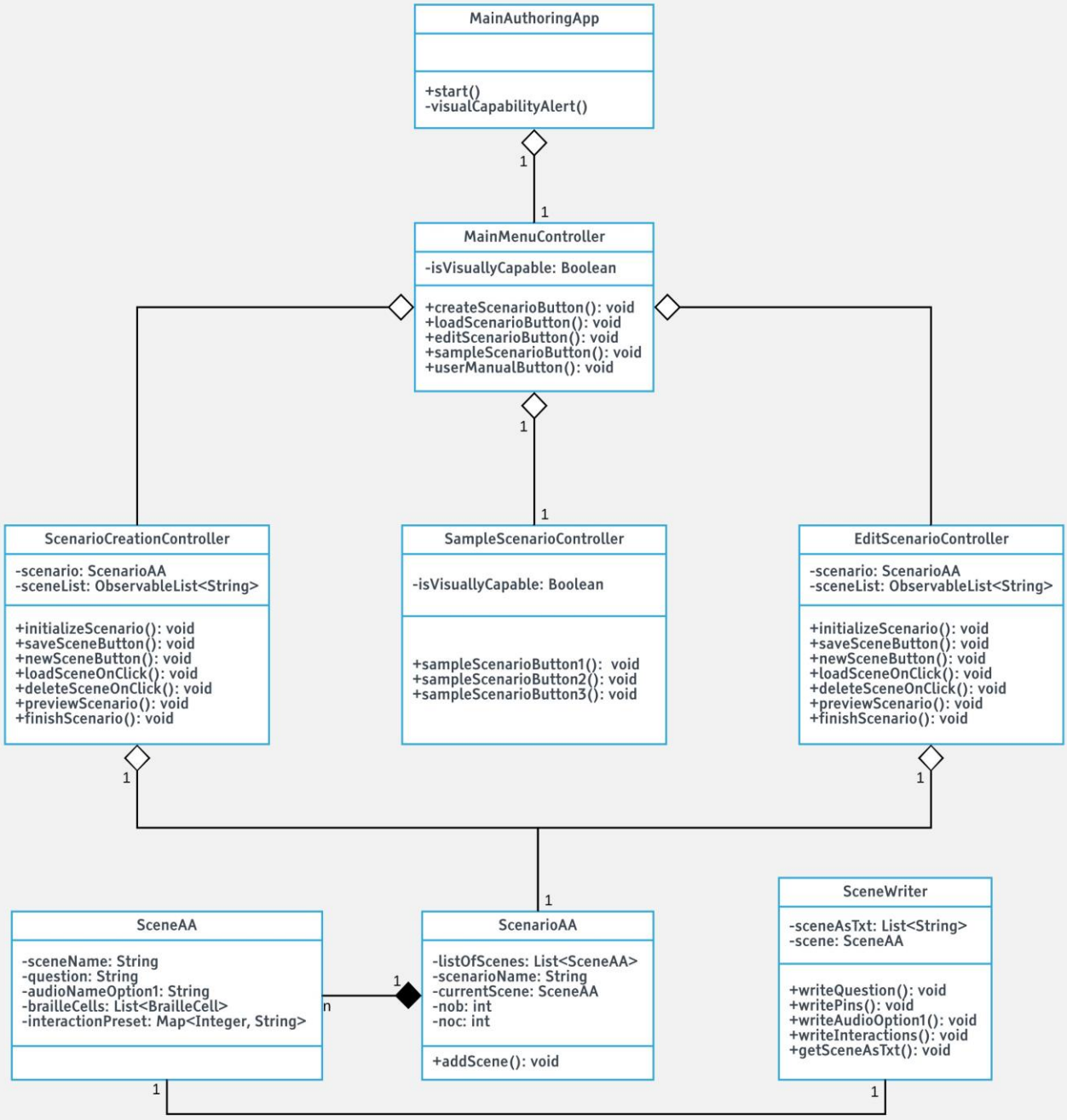
2.2 *Goal and Rational of Design Strategy*

The rational behind all design concepts and strategy will be found in much greater details in their respective section of the document. This is especially true for important classes and methods.

The decision to use MVC and FXML went hand in hand. The goal was to create an easy to understand code structure that fits with the flow of the application. Every new window that is opened has its own controller tied to the view. This makes it easier to pass vital information between windows as the user progresses through the application.

The primary motivation for creating the SceneAA and ScenarioAA classes was to store all data of a created teaching scenario. This allows the ability to write to the text file only once as opposed to during the lengthy process of creating/editing a scenario.

UML CLASS DIAGRAM AUTHORING APP v2.0.0



2.4: *Important Classes and Methods*

Please refer to the UML Class Diagram on the previous page for any reference to exact method and attribute signatures.

MainAuthoringApp

The sole purpose of this class is to launch the application. It also contains a preliminary pop up in which the user must choose their visual capability. Using this method *visualCapabilityAlert(),* this information will be stored for use throughout the application.

MainMenuController

Trivially this is the class that guides the user throughout the application. Every controller has a method signature *mainMenuButton()* which leads back to this controller. There is a method for each clickable button in the view which will open a new window specifically catered to the button that was clicked.

ScenarioCreationController

This controller is attached to the *ScenarioCreationView* which provides the GUI for a user to create a new teaching scenario. Before this class is used the user will have entered the number of cells, number of buttons and the scenario name. This information gets passed through the *initializeScenario()* method and stored in a ScenarioAA object. The most important attribute in this class is scenario which is of type ScenarioAA. The ScenarioAA class will be explained in further detail below. This controller is responsible for reactively changing the GUI based on user input as well as store every change the user makes into the scenario object. The user does this by creating and editing scenes. The SceneAA method is explained in detail below. The user can save a scene, create a new scene, load a scene, and delete a scene.

Calling the *previewScenario()* method will write a temporary text file and run it. It is deleted after the preview ends. Calling the *finishScenario()* method will create the text file using SceneWriter (explained below) as well as serialize the ScenarioAA object, saving it as a binary file to access when going back and editing a scenario.

SampleScenarioController

This controller primarily handles the users input for viewing 1 of 3 pre-set teaching scenarios. Each button from its respective view has a method call that runs an instance of a *ScenarioParser* object.

EditScenarioController

This controller is attached to the *EditScenarioView* which provides the GUI for a user to edit a previously existing teaching scenario. Before this class is used the user will have selected a previous teaching scenario file. After deserialization the object gets passed through the *initializeScenario()* method and stored in the scenario attribute. The ScenarioAA class is explained in further detail below. This controller is responsible for reactively changing the GUI based on user input as well as store every change the user makes into the scenario object. The

user does this by creating and editing scenes. The SceneAA method is explained in detail below. The user can save a scene, create a new scene, load a scene, and delete a scene.

Calling the *previewScenario()* method will write a temporary text file and run it. It is deleted after the preview ends. Calling the *finishScenario()* method will overwrite the existing text file using SceneWriter (explained below) as well as re-serialize the ScenarioAA object, saving it as a new binary file to access when going back and editing the scenario again.

SceneAA

From a design aspect a scene is a specific instance of an interaction between the teacher and the student. This scene consists of any number of the following items: name, question, braille cells and interaction pre-sets. This class is used to simplify the scenario class. It provides the individual details of a teaching scenario.

ScenarioAA

The primary motivation to have a ScenarioAA class was to store all data of a created teaching scenario. This allows the ability to write to the text file only once as apposed to during the lengthy process of creating/editing a scenario. A scenario contains a list of scenes, explained above.

SceneWriter

The primary goal of the scene writer class is to transfer the data stored on a ScenarioAA object to a text file written in the proper teaching scenario format. Calling this class in any function that wishes to write a scenario requires you to loop through every SceneAA in the ScenarioAA.

All methods write individual options for each scene and then *getSceneAsTxt()* is called to combine it all together in the proper format and store it in *sceneAsTxt* list of strings.

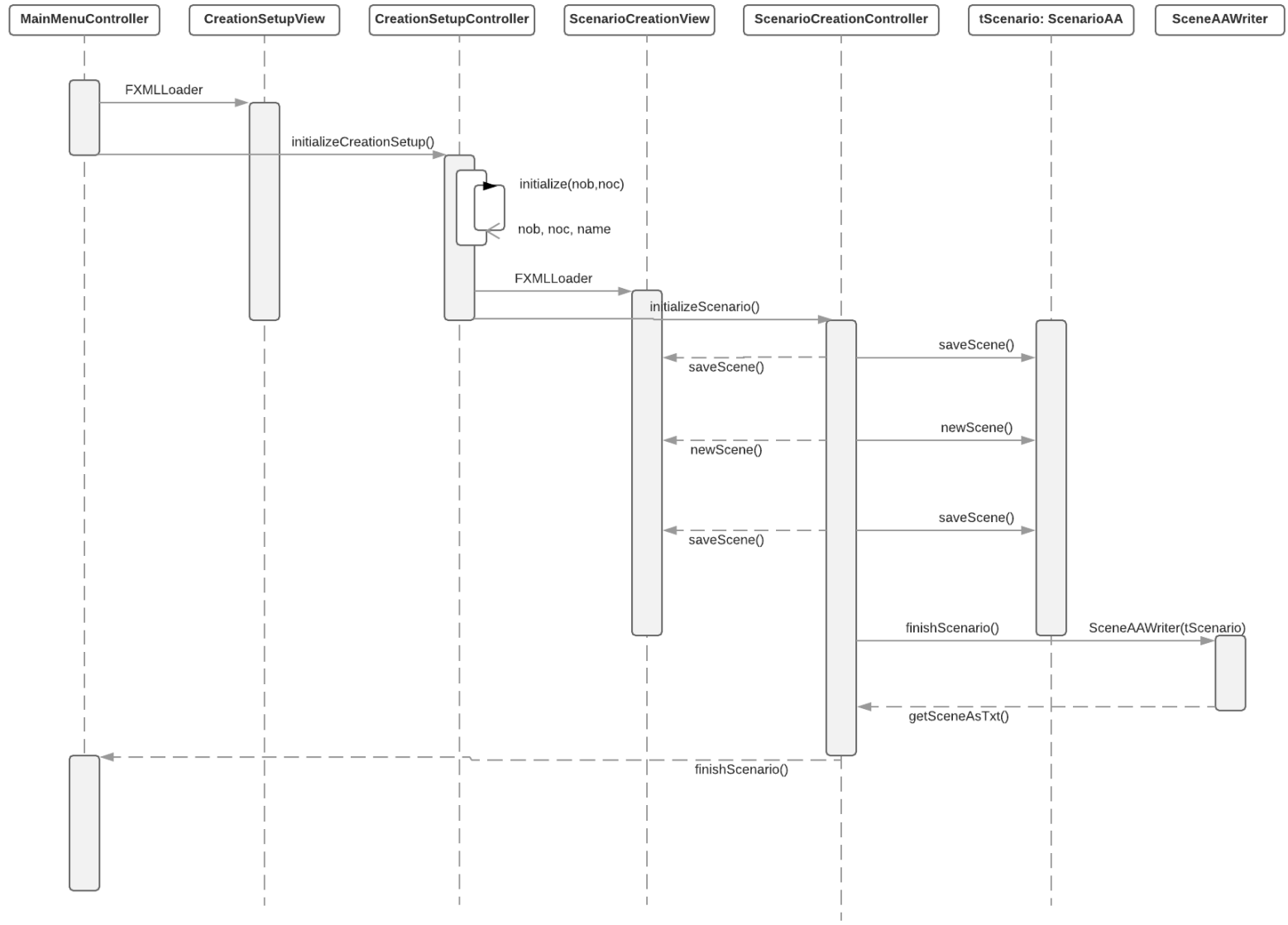
Section 3: Important Runtime Scenarios

All the crucial and important runtime scenarios will be illustrated as a sequence diagram followed by further explanation. Some non-vital yet important methods, objects and sequences will also be mentioned, however will not contain a sequence diagram due to their simplicity.

The runtime scenarios that will be discusses are as follows:

- Creating a scenario
- Editing a scenario
- Sample scenarios
- Loading a scenario

3.1: *Creating a Teaching Scenario*



Sequence Explained

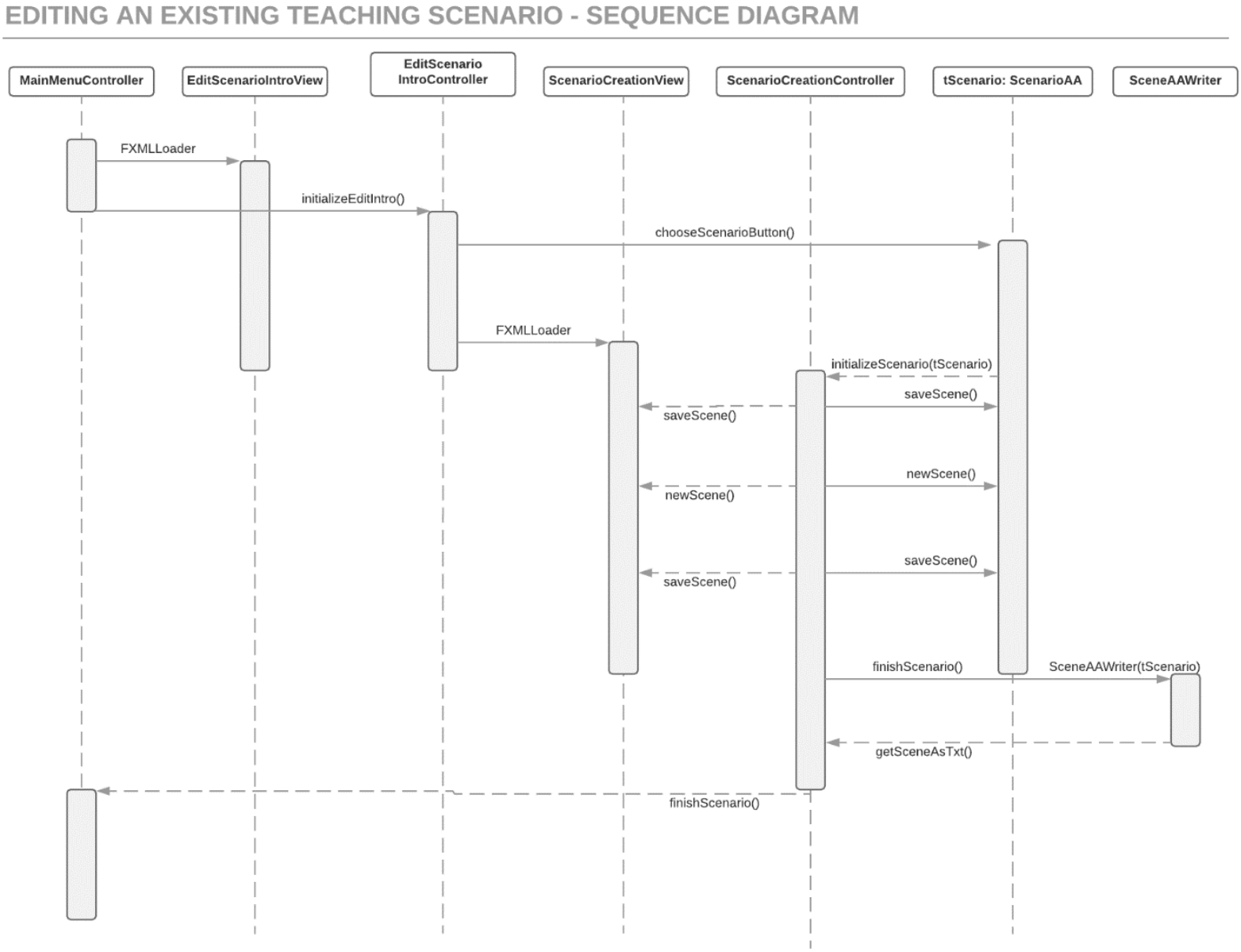
The user starts off at the main menu of the application then proceeds to press the create scenario button. In doing so this launches the creation setup view and controller. The user will input the required number of buttons (nob), number of braille cells (noc), and the teaching scenario name. When the continue button is pressed, this information is passed to the *ScenarioCreationController's* initialization method. The new ScenarioAA created, named *tScenario,* is now filled with the newly passed information. The remaining sequences of saving, creating, and deleting scenes, impacts *tScenario* as well as the *ScenarioCreationView*. The view

is constantly updated with then information the user inputs this way the user can keep track of what they created via the GUI. At the same time in the background the ScenarioAA object named *tScenario* is updated with all the new information. Once the user is done with creating the teaching scenario they click the finish scenario button which does a couple of important things. The first and most important is it creates the text file (teaching scenario file) using SceneAAWriter with all the information from *tScenario*. Also, it creates a binary file using serialization which will save this objects data for when the user wishes to make changes to this teaching scenario in the future. Lastly the user will be automatically returned to the main menu where they began in the beginning.

Important Classes and Methods

When creating a teaching scenario, the two most vital classes is the *ScenarioCreationController* and ScenarioAA. The interaction between these two classes is a large bulk of the creation process. In this sequence the *ScenarioCreationController* gives the user the ability to customize the flow of their teaching scenarios. The ScenarioAA class simply stores this information so that the file can be created and changed in the future.

3.2: *Editing a Teaching Scenario*



Sequence Explained

Note that editing a scenario is nearly identical to creating one, however, the main change to its operation is highlighted in the sequence diagram above. Instead of a creation setup where the user inputs fresh values for the scenario, existing scenario information is passed as the user loads an existing scenario to edit. It is worth mentioning that when the user selects a file to edit, a call will be made to retrieve the binary file associated with it which stores all relevant information about the teaching scenario. The existing scenario that the user is trying to edit will only be re-

written (to the text file) if the user hits the finish scenario button. Everything else in terms of software design is identical between creating and editing scenarios.

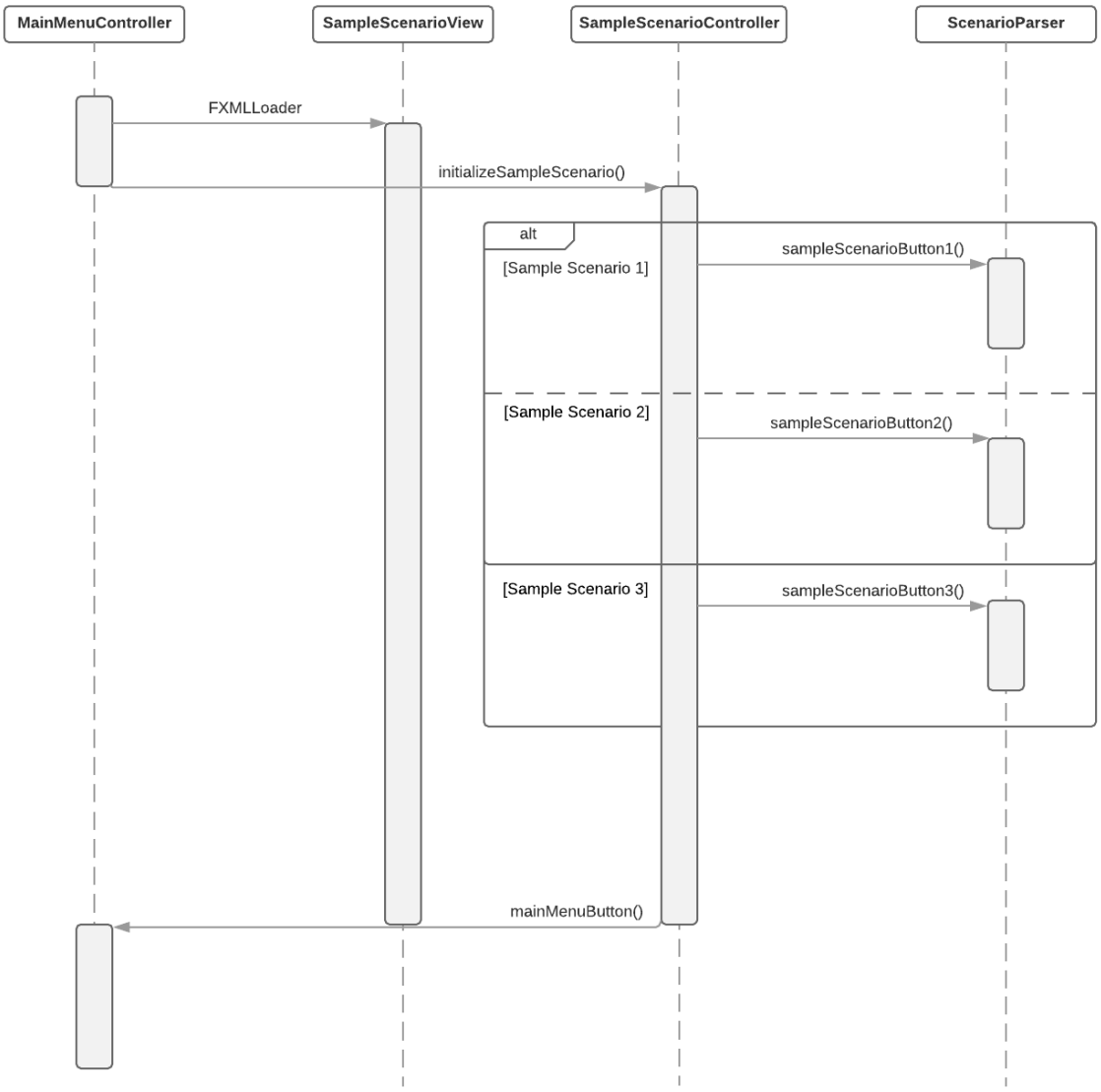
The user starts off at the main menu of the application then proceeds to press the *Edit Scenario* button. In doing so this launches the edit intro view and controller. The user will now select the teaching scenario that they wish to edit using a file chooser. The file that they choose must be a text file and contain the binary file associated with it in another folder (done automatically for them). Once the user presses continue the binary file will be converted back as a ScenarioAA object with all the previously saved data intact. This object will then be passed to the initialization of the *EditScenarioController*. Then the controller and view will also be updated with all the previously created information for the user to see on the GUI. The remaining sequences of saving, creating, and deleting scenes, impacts the ScenarioAA object as well as the ScenarioCreationView. The view is constantly updated with then information the user inputs this way the user can keep track of what they created via the GUI. At the same time in the background the ScenarioAA object named *tScenario* is updated with all the new information. Once the user is done with editing the teaching scenario they click the finish scenario button. This will overwrite the previous text file (teaching scenario file) using SceneAAWriter with all the information from the new *tScenario*. Also, it overwrites the binary file which will save this objects data for when the user wishes to make changes to this teaching scenario again. Lastly the user will be automatically returned to the main menu where they began in the beginning.

Important Classes and Methods

When editing a teaching scenario, the most important classes do not change from creating one. It is still the *ScenarioCreationController* and ScenarioAA. The interaction between these two classes is a large bulk of the editing process. In this sequence the *ScenarioCreationController* gives the user the ability to change the flow of their teaching scenarios. The ScenarioAA class simply stores this information so that the file can be created and changed in the future.

3.3: *Sample Teaching Scenarios*

VIEWING A SAMPLE SCENARIO - SEQUENCE DIAGRAM



Sequence Explained

The user starts off in the main menu. The user then clicks the sample scenario button which will take them to a new window with 3 optional sample scenarios as well as the option to return to the main menu. Depending on which sample scenario is selected will affect what the scenario parser will play using the *VideoPlayerClass*. The user then returns to the main menu.

Important Classes and Methods

Both a controller and view are dedicated to the sample scenario function. The SampleScenarioController does not use serialization to access the stored sample scenarios. It simply reads a text file that was hand generated. This means that the user cannot edit any of the sample scenarios in the main menu.

3.4: *Loading a Teaching Scenario*

Sequence Explained

As mentioned before, this sequence is far too simplistic for a sequence diagram. However, it is a main feature of the application and will most likely be used more than any other function.

Straight from the main menu the user will click the *Load Scenario* button. This will open a simple file chooser. Once the appropriate file is selected the scenario will play using the scenario parser class. Note that at the start of the application the user will indicate if they are visually capable or not. If the user is visually capable it will run as normal with the GUI. If they indicated, they are visually incapable then an audio player will play with no visuals.

Section 4: Maintenance Scenarios

4.1: *Overall design maintenance strategy*

Authoring App will stick with MVC using Java and FXML moving forward. Although we will not change design structures it is important to note the tedious efforts to maintain so many individually controller/view combinations that some smaller windows will be adopted into Swing. This will save half the space to look for code.

Various efforts will be made to consistently refractor existing and future code implemented. All class, method and attribute names will be constantly looked at and modified. All classes including the two main controllers will be revisited and if need be we will extract as an interface or subclass. We will look to implement shotgun surgery to clean up any current or future lazy classes, generality, and all other code smells.

4.2: *Design improvements for long term sustainability*

Improvements that are looking to be made is bundling certain classes as an abstract class or interface. One that will be done in a future version of *Authoring App* would be to merge most methods and attributes between *ScenarioCreationController* and *EditScenarioController* into an abstract class named *TeachingScenarioController*.