Braille Box Authoring App

Design Document

EECS 2311: Software Engineering Project
Team 9
May 19, 2018

Authors

Jeremy Winkler

Nisha Sharma

Tyler Thomson

Revision Level	Date	Pages affected	Comments
1.0.0	May 19, 2018	1-10	Final Submission
1.0	March 16, 2018	1	Initial draft

Table of Contents

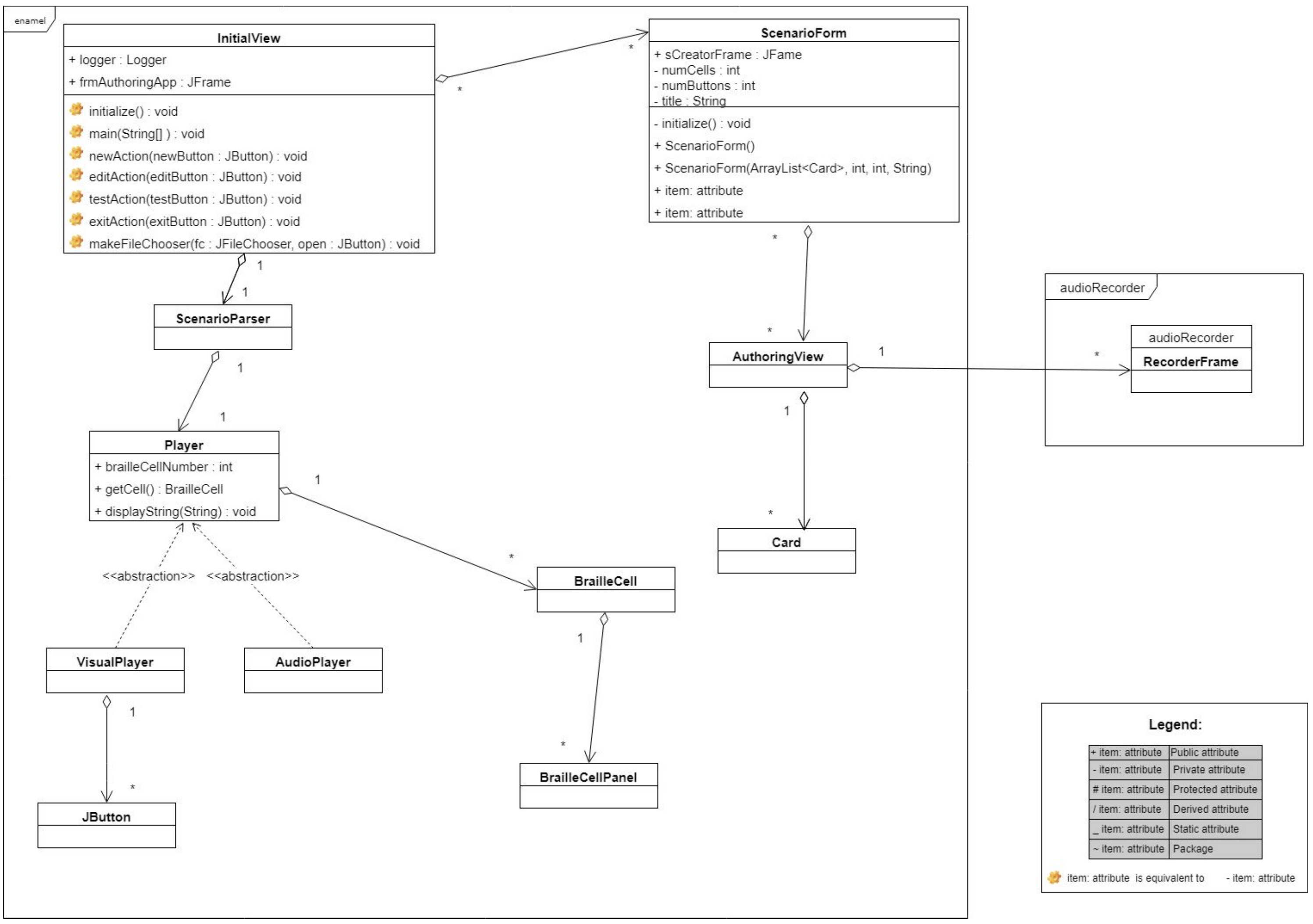
List of Figures

1.0 Introduction

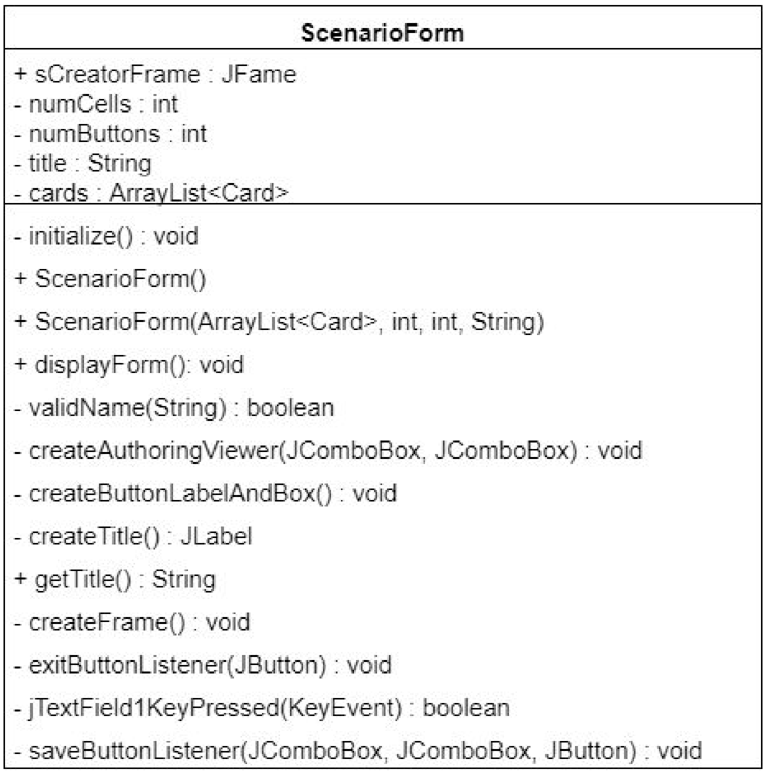
This document discusses organization and structure of the AuthoringApp. The document sheds light on high level organization of the system along with briefly explaining important classes, methods and their interactions. It also lists the important objects that get created at the runtime and their connection with each other. The document utilizes UML class diagrams and sequence diagrams to clearly illustrate the internal structure of the app.

2.0 Class Diagram

Following is the overall UML diagram of the Authoring App with most important classes for the app’s functionality. The app is launched from the InitialView. For creating a new Scenario, the app requires instantiating Scenario Form which in turn instantiates Authoring Viewer. Multiple instances Initial View can be created by launching the app multiple times. From the initial view multiple Scenario forms can be created. Scenario Form can be started again from within the AuthoringView. From the AuthoringView, multiple Recorders can be launched. During the testing mode, the app follows the [UML]diagram for starter code to launch different components.


enamel
InitialView
+ logger : Logger
+ frmAuthoringApp : JFrame
initialize() : void
main(String[]) : void
newAction(newButton : JButton) : void
editAction(editButton : JButton) : void
testAction(testButton : JButton) : void
exitAction(exitButton : JButton) : void
makeFileChooser(fc : JFileChooser, open : JButton) : void
ScenarioForm
+ sCreatorFrame : JFame
- numCells : int
- numButtons : int
- title : String
- initialize() : void
+ ScenarioForm()
+ ScenarioForm(ArrayList<Card>, int, int, String)
+ item: attribute
+ item: attribute
ScenarioParser
Player
+ brailleCellNumber : int
+ getCell() : BrailleCell
+ displayString(String) : void
<<abstraction>>
<<abstraction>>
VisualPlayer
AudioPlayer
JButton
BrailleCell
BrailleCellPanel
AuthoringView
Card
audioRecorder
audioRecorder
RecorderFrame
1
*
Legend:
+ item: attribute Public attribute
- item: attribute Private attribute
# item: attribute Protected attribute
/ item: attribute Derived attribute
_ item: attribute Static attribute
~ item: attribute Package
item: attribute is equivalent to - item: attribute

The initial view class is shown in expanded view in the class diagram above. Class diagrams for other major classes can be found below.

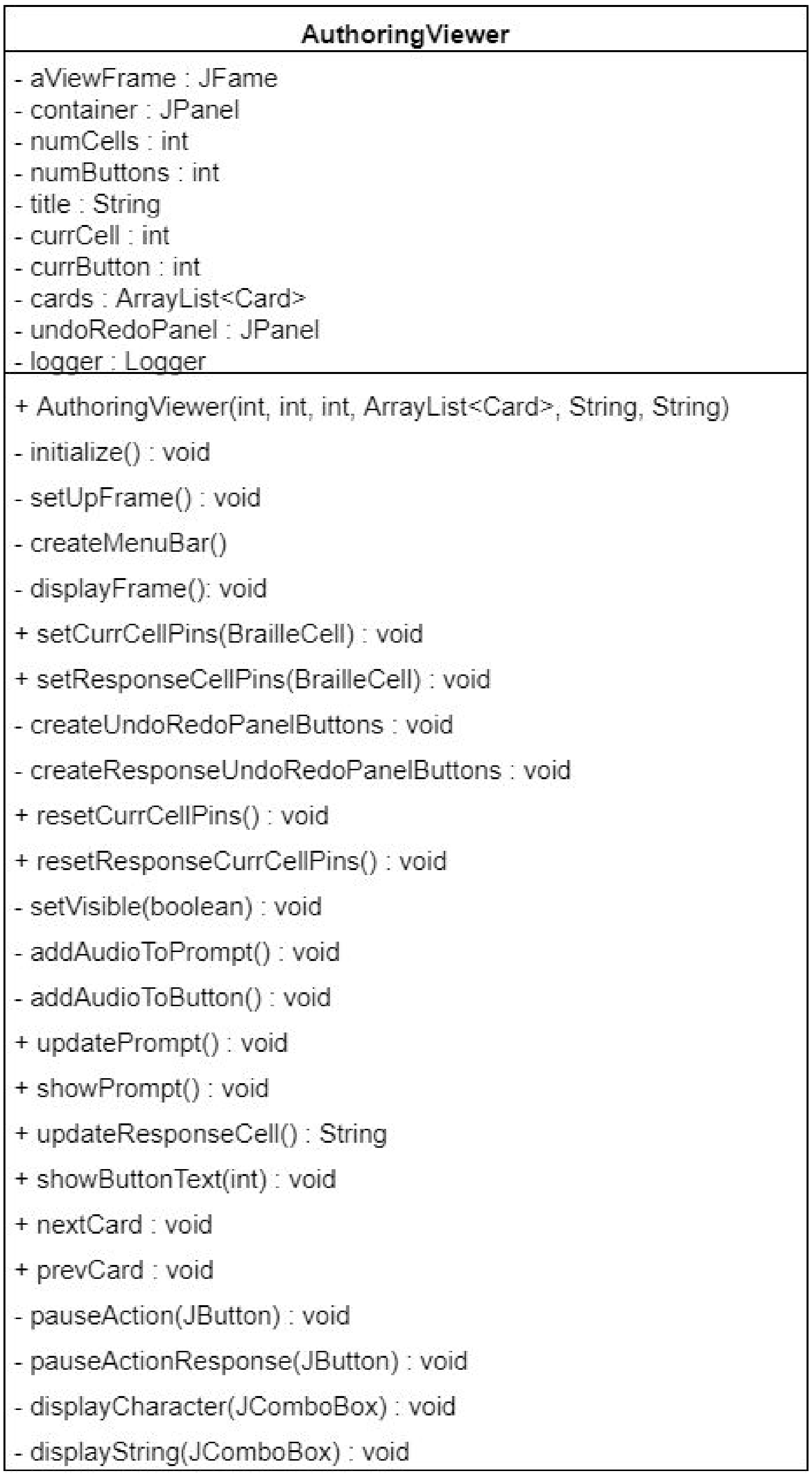


ScenarioForm

+ sCreatorFrame : JFame
- numCells : int
- numButtons : int
- title : String
- cards : ArrayList<Card>

- initialize() : void
+ ScenarioForm()
+ ScenarioForm(ArrayList<Card>, int, int, String)
+ displayForm(): void
- validName(String) : boolean
- createAuthoringViewer(JComboBox, JComboBox) : void
- createButtonLabelAndBox() : void
- createTitle() : JLabel
+ getTitle() : String
- createFrame() : void
- exitButtonListener(JButton) : void
- jTextField1KeyPressed(KeyEvent) : boolean
- saveButtonListener(JComboBox, JComboBox, JButton) : void

ScenarioForm is another class with a GUI created as a JFrame and button functionality. Objects of this class are created from within the InitialView and AuthouringViewer class. The main function of this class is to create a form for the user to input initial setup of the scenario. This initial set up is number of cells, number of buttons and title of the scenario. The title of the scenario is also used to suggest the name of the scenario file to be saved.

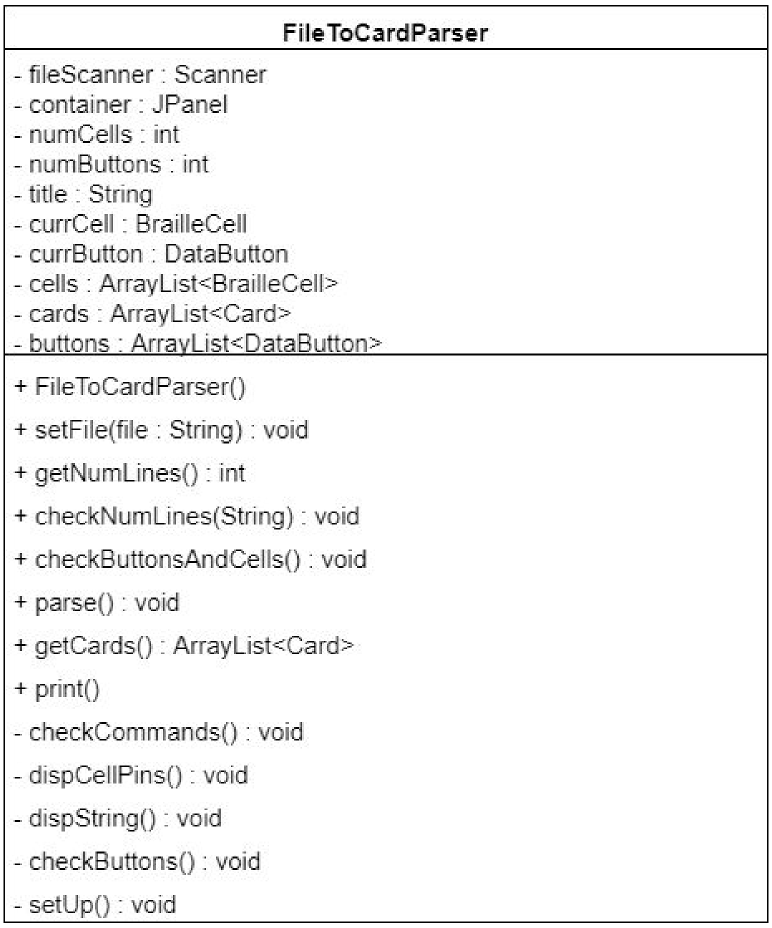


AuthoringViewer

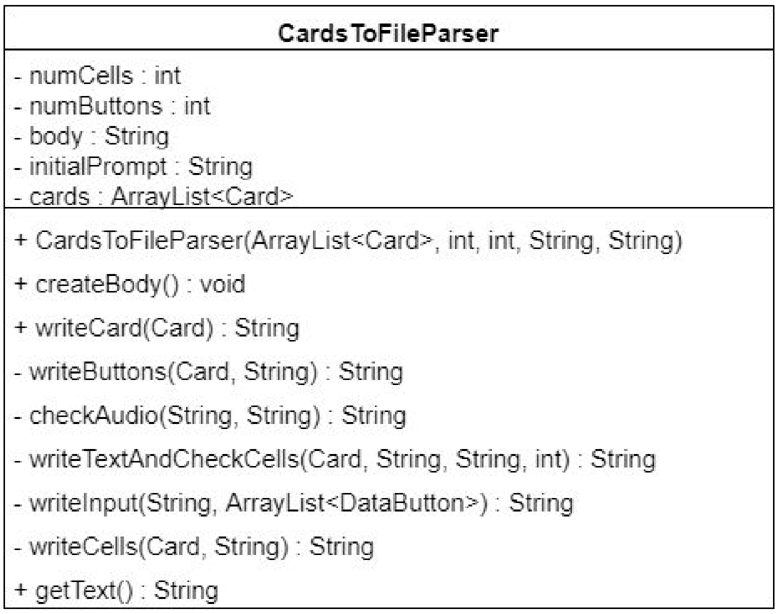
- aViewFrame : JFame
- container : JPanel
- numCells : int
- numButtons : int
- title : String
- currCell : int
- currButton : int
- cards : ArrayList<Card>
- undoRedoPanel : JPanel
- logger : Logger

+ AuthoringViewer(int, int, int, ArrayList<Card>, String, String)
- initialize() : void
- setUpFrame() : void
- createMenuBar()
- displayFrame(): void
+ setCurrCellPins(BrailleCell) : void
+ setResponseCellPins(BrailleCell) : void
- createUndoRedoPanelButtons : void
- createResponseUndoRedoPanelButtons : void
+ resetCurrCellPins() : void
+ resetResponseCurrCellPins() : void
- setVisible(boolean) : void
- addAudioToPrompt() : void
- addAudioToButton() : void
+ updatePrompt() : void
+ showPrompt() : void
+ updateResponseCell() : String
+ showButtonText(int) : void
+ nextCard : void
+ prevCard : void
- pauseAction(JButton) : void
- pauseActionResponse(JButton) : void
- displayCharacter(JComboBox) : void
- displayString(JComboBox) : void

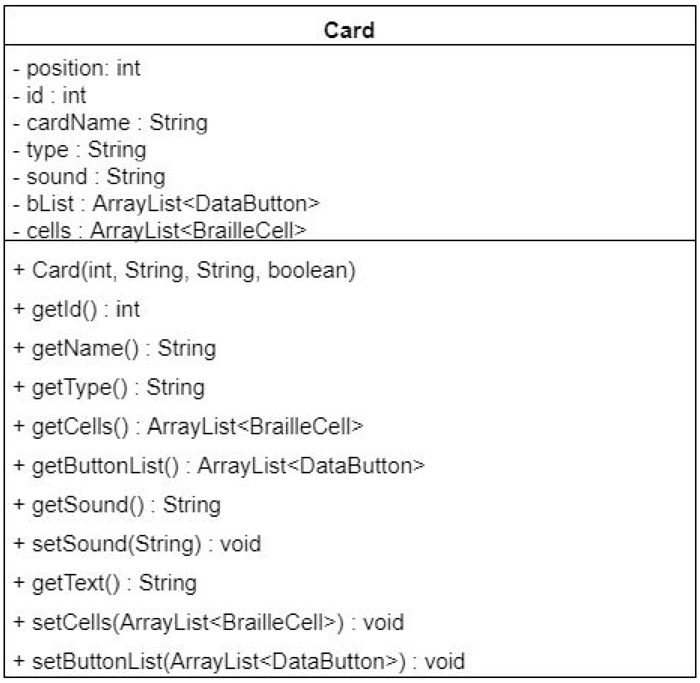
AuthoringViewer is a GUI class where most of the work happens. Interacting with this UI, user can add different functionalities to the scenario file, from adding pauses, audio files to displaying strings and raising pins on braille cell.



FileToCardParser handles the parsing of content from already saved scenario file back to the AuthoringApp GUI for user to edit.


CardsToFileParser
- numCells : int
- numButtons : int
- body : String
- initialPrompt : String
- cards : ArrayList<Card>
+ CardsToFileParser(ArrayList<Card>, int, int, String, String)
+ createBody() : void
+ writeCard(Card) : String
- writeButtons(Card, String) : String
- checkAudio(String, String) : String
- writeTextAndCheckCells(Card, String, String, int) : String
- writeInput(String, ArrayList<DataButton>) : String
- writeCells(Card, String) : String
+ getText() : String

CardsToFileParsser handles the parsing of the content from the Authoring App GUI , added by the user, to create scenario files with proper format.

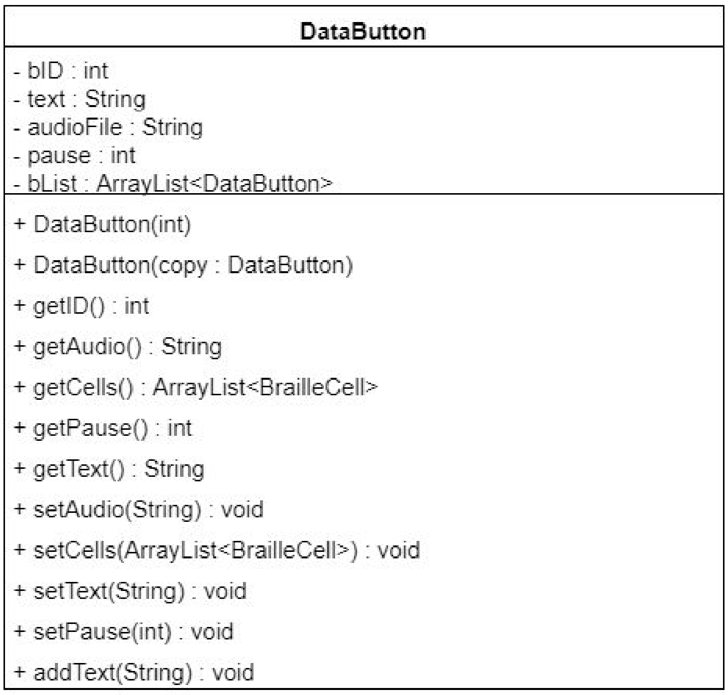


Card

- position: int
- id : int
- cardName : String
- type : String
- sound : String
- bList : ArrayList<DataButton>
- cells : ArrayList<BrailleCell>

+ Card(int, String, String, boolean)
+ getId() : int
+ getName() : String
+ getType() : String
+ getCells() : ArrayList<BrailleCell>
+ getButtonList() : ArrayList<DataButton>
+ getSound() : String
+ setSound(String) : void
+ getText() : String
+ setCells(ArrayList<BrailleCell>) : void
+ setButtonList(ArrayList<DataButton>) : void

The Card objects are used to divide the Scenario into different sections that can be rearranged by changing the order by the user with ease.

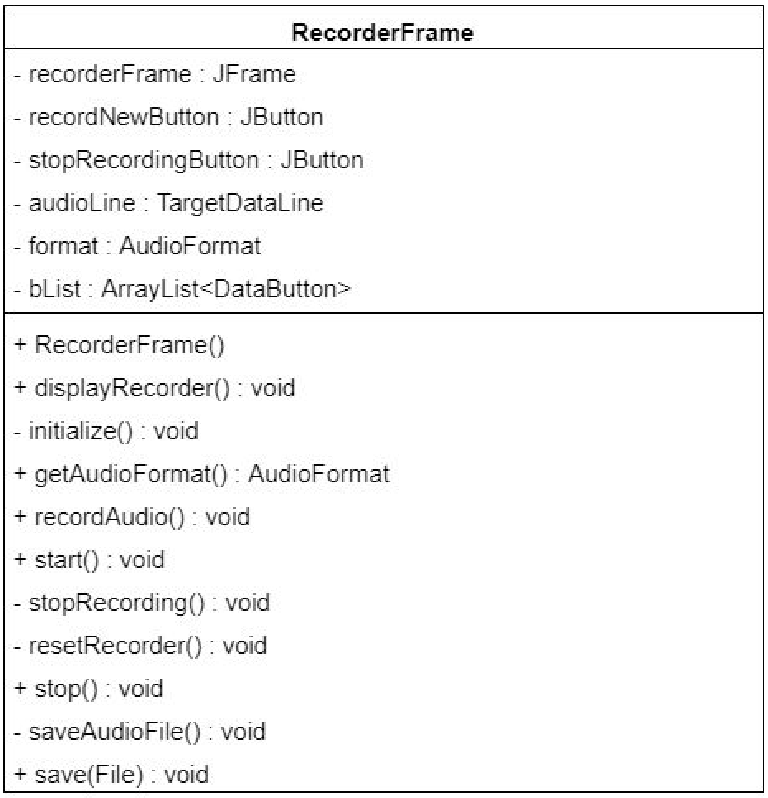


DataButton

- bID : int
- text : String
- audioFile : String
- pause : int
- bList : ArrayList<DataButton>

+ DataButton(int)
+ DataButton(copy : DataButton)
+ getID() : int
+ getAudio() : String
+ getCells() : ArrayList<BrailleCell>
+ getPause() : int
+ getText() : String
+ setAudio(String) : void
+ setCells(ArrayList<BrailleCell>) : void
+ setText(String) : void
+ setPause(int) : void
+ addText(String) : void

Data Button handles multiple functionalities added to button clicks.



RecorderFrame

- recorderFrame : JFrame
- recordNewButton : JButton
- stopRecordingButton : JButton
- audioLine : TargetDataLine
- format : AudioFormat
- bList : ArrayList<DataButton>

+ RecorderFrame()
+ displayRecorder() : void
- initialize() : void
+ getAudioFormat() : AudioFormat
+ recordAudio() : void
+ start() : void
- stopRecording() : void
- resetRecorder() : void
+ stop() : void
- saveAudioFile() : void
+ save(File) : void

RecorderFrame class takes care of recording audio files, saving them and playing back recently saved files.

3.0 Sequence Diagram

This section contains different sequence diagrams depicting how objects interact at runtime. It should be noted that the time passes from top to bottom. The start up page or InitialView of the app has only four features allowing user to create a new scenario, to edit an existing scenario, to test an existing scenario and to exit the app.

The sequence diagram below (Fig 1) lists one of the overall success scenarios. The user starts by running the application which creates an object of class InitialView. Clicking “New” button invokes the constructor of “ScenarioForm” class which creates the scenario form that takes information from the user regarding initial setup of the scenario including number of cells, number of buttons and title of the scenario. Once the user is satisfied with the selection, the user clicks on “Create a New Scenario” button. This actions invokes the constructor of “AuthoringView” class that creates a JFrame with different GUI components for the user to interact with for creating a scenario. From within the “AuthoringView” GUI, when the user performs “Record Audio” action by selecting “Record” option under “Audio” menu on menu bar, constructor of “RecorderFrame” class is invoked which initializes the GUI for user to “record” audio and save it as “.wav” file. Once done user is back to “AuthoringView” GUI where any further changes can be made to the scenario before clicking “Save” and/or exiting the “AuthoringView” GUI. From the InitialView of the GUI, the user simply exits the app by clicking “Exit” button.

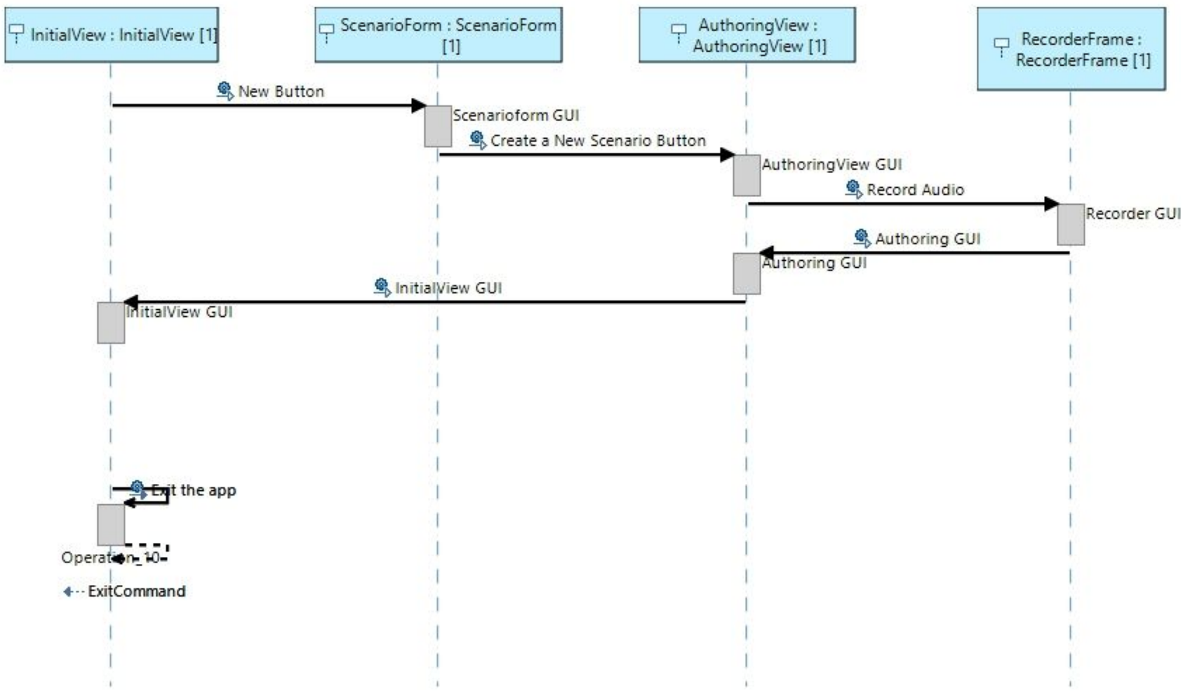


Fig 1: Sequence Diagram for an Overall Success Scenario

Interaction shown in Fig 2 below is for a use case involving editing a scenario. The diagram below depicts relation between sequence of object creation of four major classes. The IntialView class instantiates AuthoringView GUI from which the user makes edits to the scenario form, then records audio and then finally comes back to AuthoringView GUI to make any changes, save and exit the app.

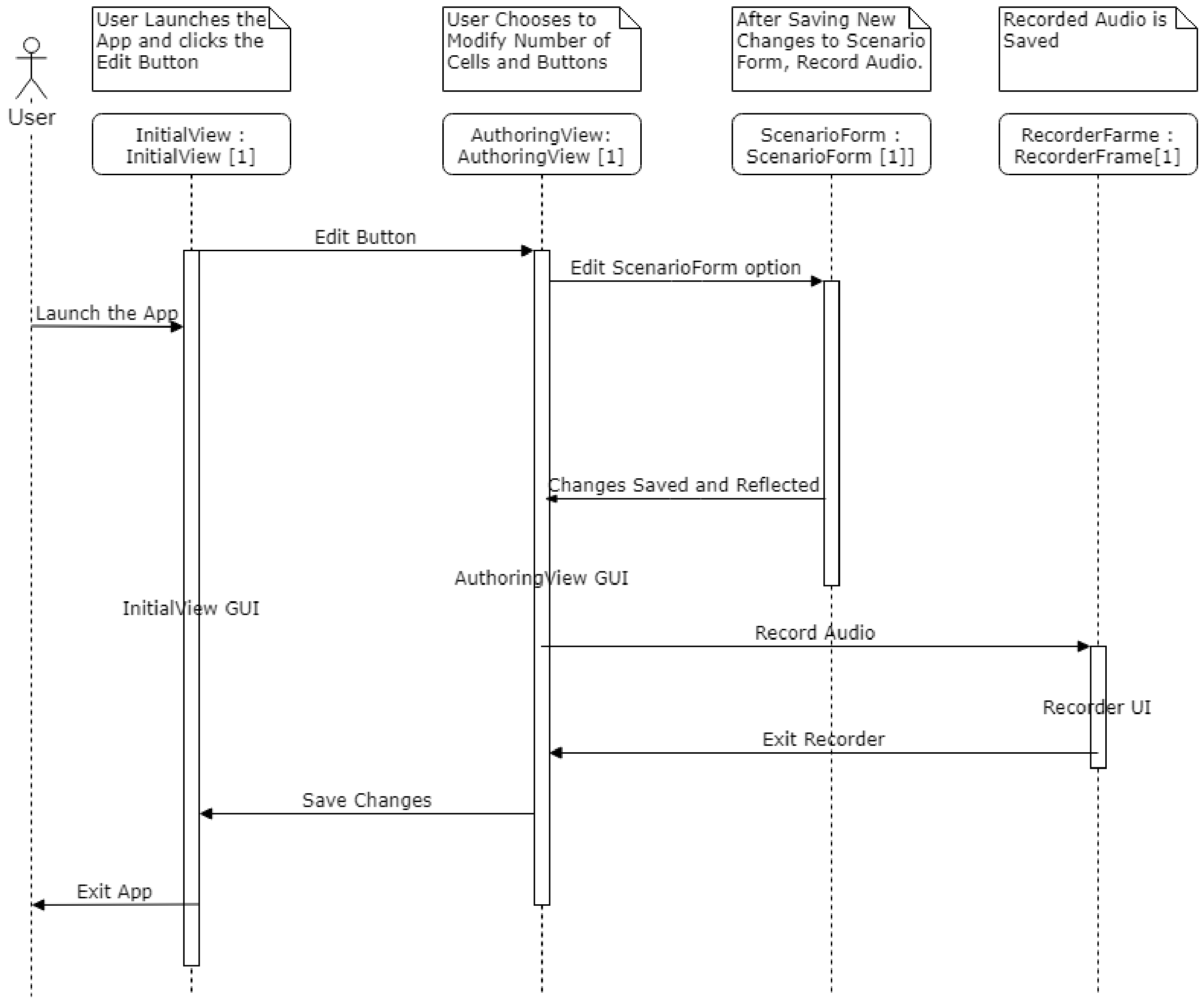
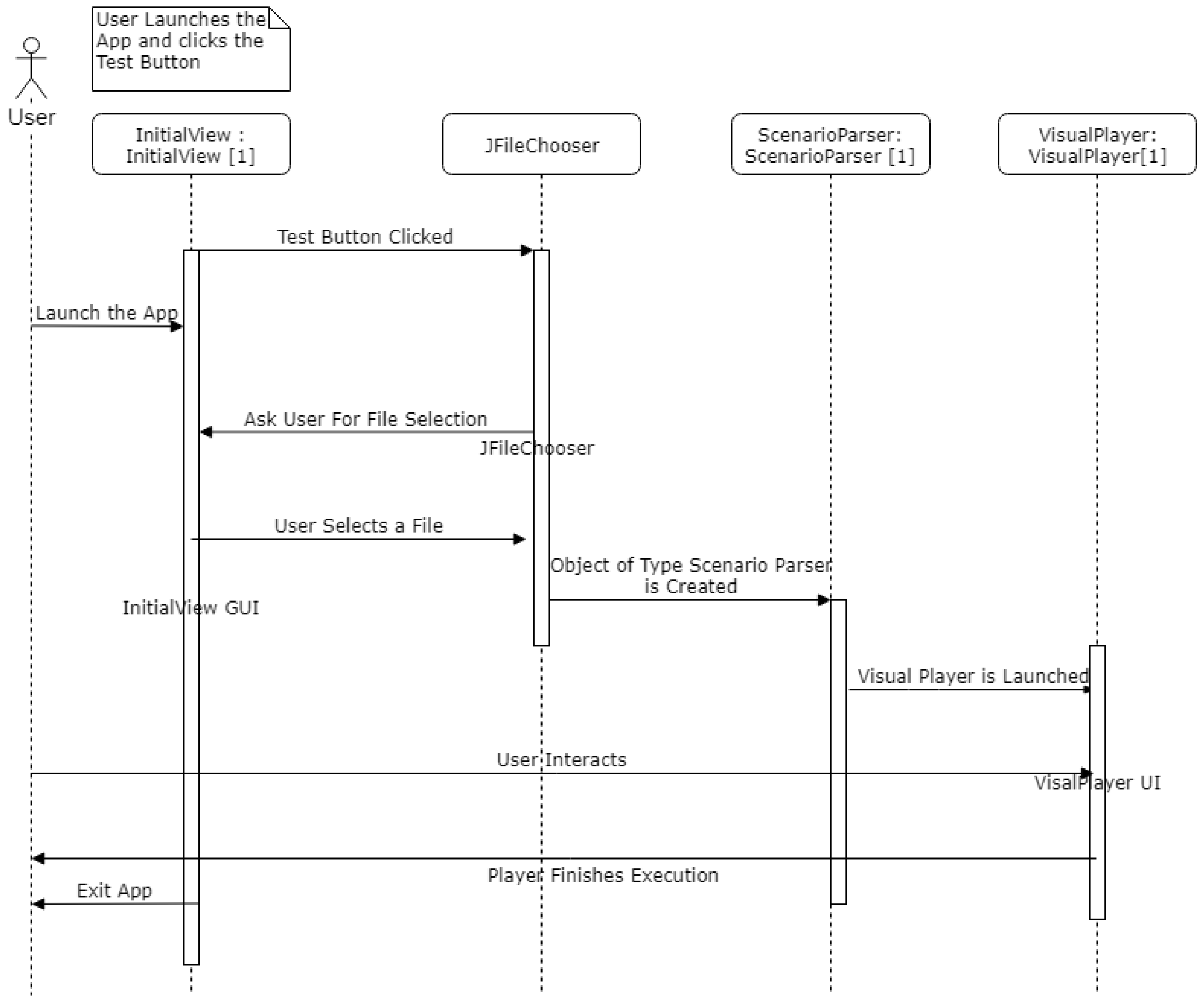

User Launches the App and clicks the Edit Button
User Chooses to Modify Number of Cells and Buttons
After Saving New Changes to Scenario Form, Record Audio.
Recorded Audio is Saved
User
InitialView : InitialView [1]
AuthoringView: AuthoringView [1]
ScenarioForm : ScenarioForm [1]]
RecorderFarme : RecorderFrame[1]
Edit Button
Edit ScenarioForm option
Launch the App
Changes Saved and Reflected
AuthoringView GUI
InitialView GUI
Record Audio
Recorder UI
Exit Recorder
Save Changes
Exit App

Fig 2: Sequence Diagram for an Editing a Scenario

Interaction shown in Fig 3 below is for a use case involving testing a scenario.This involves choosing a file and creating and interacting with VisualPlayer.



User Launches the App and clicks the Test Button
User
InitialView : InitialView [1]
JFileChooser
ScenarioParser: ScenarioParser [1]
VisualPlayer: VisualPlayer[1]
Test Button Clicked
Launch the App
Ask User For File Selection
JFileChooser
User Selects a File
Object of Type Scenario Parser is Created
InitialView GUI
Visual Player is Launched
User Interacts
VisalPlayer UI
Player Finishes Execution
Exit App

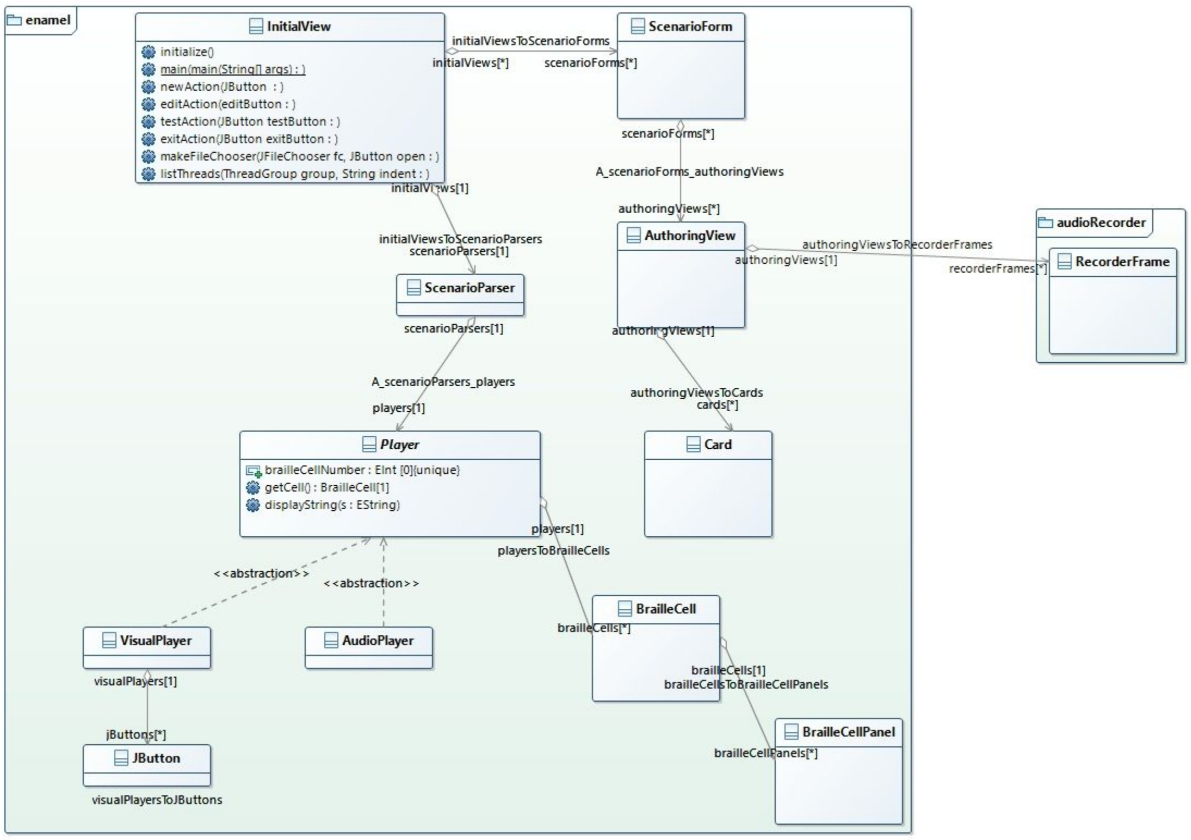
Fig 3: Sequence Diagram for an Testing a Scenario

2.0 Maintenance Scenario

The design of AuthoringApp aims towards simplicity and ease of use. We feel that the maintenance of the app would not be difficult. The class names, method names and attribute names are intuitive and reflect their purpose. Essential JavaDoc comments are generated for simplicity. The code also has other comments illustrating the purpose of most commands. For Bug Fixes changes can be made to affected classes. Changing the GUI look and feel, methods dealing with creation can be altered, for example.
If additional features are added, FileToCardParser and CardToFileParser can be updated to handle parsing of the scenario files correctly.

4.0 Appendix

Class diagram created Using Eclipse UML creator tool:


enamel
InitialView
initialize()
main(main(String[] args) :)
newAction(JButton :)
editAction(editButton :)
testAction(JButton testButton :)
exitAction(JButton exitButton :)
makeFileChooser(JFileChooser fc, JButton open :)
listThreads(ThreadGroup group, String indent :)
initialViewsToScenarioForms
initialViews[*]
scenarioForms[*]
ScenarioForm
scenarioForms[*]
A_scenarioForms_authoringViews
authoringViews[*]
initialViews[1]
initialViewsToScenarioParsers
scenarioParsers[1]
ScenarioParser
scenarioParsers[1]
AuthoringView
authoringViewsToRecorderFrames
authoringViews[1]
recorderFrames[*]
audioRecorder
RecorderFrame
authoringViews[1]
A_scenarioParsers_players
players[1]
authoringViewsToCards
cards[*]
Player
brailleCellNumber : EInt [0]{unique}
getCell() : BrailleCell[1]
displayString(s : EString)
Card
players[1]
playersToBrailleCells
<<abstraction>>
<<abstraction>>
BrailleCell
brailleCells[*]
VisualPlayer
AudioPlayer
visualPlayers[1]
brailleCells[1]
brailleCellsToBrailleCellPanels
jButtons[*]
JButton
BrailleCellPanel
brailleCellPanels[*]
visualPlayersToJButtons