Treasure Box Braille

Design Document

Table of Contents

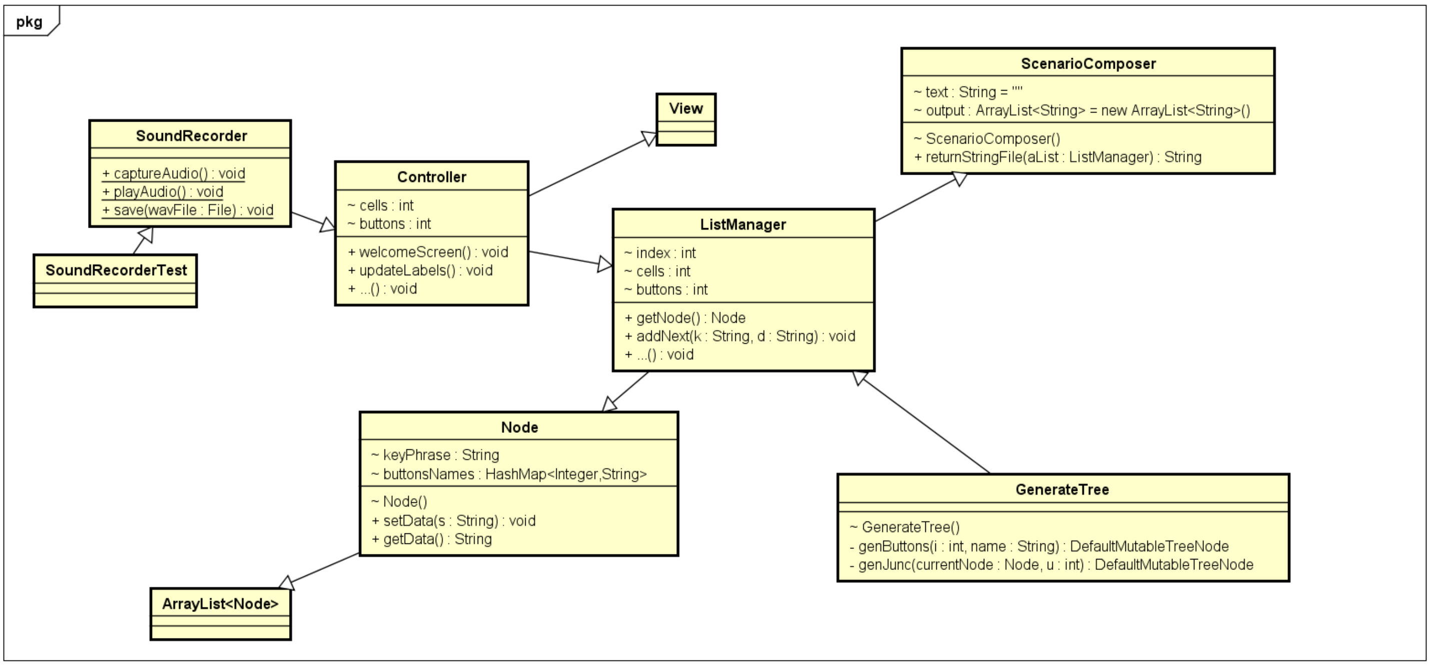
Introduction

This design document is intended for an experienced developer to become acquainted with the application and quickly understand the overall structure of the program.

The included UML diagrams give a quick overview of the classes and how they relate to each other. Sequence diagrams have been included to depict how important objects interact during runtime.

UML Diagrams

Treasure Box Braille Application



Summary of Important Classes:

Our application is based upon the ‘Model / View / Controller’ concept of development. The ‘ListManager’ object stores information about the scenario. Therefore, ‘ListManager’ objects act as models for the scenarios. We are using JFrame to develop our user interface so that our application can be easily made compatible with Windows, Mac OS, and Linux OS.

ListManager

Represents the data structure of the scenario. It uses an array list of ‘Node’ objects. Using this model allows for manipulation of scenario events and gives us control of how events are related. This allows the ‘Controller’ to interact with the scenario.

Node

Information container that represents events inside of a scenario. Contains information regarding the type of event, its data, and how this event relates to other events.

View

Visual representation of the ‘ListManager’ model. Gets the necessary data from ‘ListManager’ to display a correct model to the user. JFrame is used extensively in this class to display information to the user.

Controller

Arranges the ‘View’ to represent the ‘ListManager’ model. The controller takes the user input and performs the appropriate actions to manipulate the ‘View’ and ‘ListManager’.

SoundRecorder

Creates a buffer in which audio is recorded to. Once the user is satisfied with the recording they can save the recording as a ‘.wav’ file and a new sound event node is automatically created.

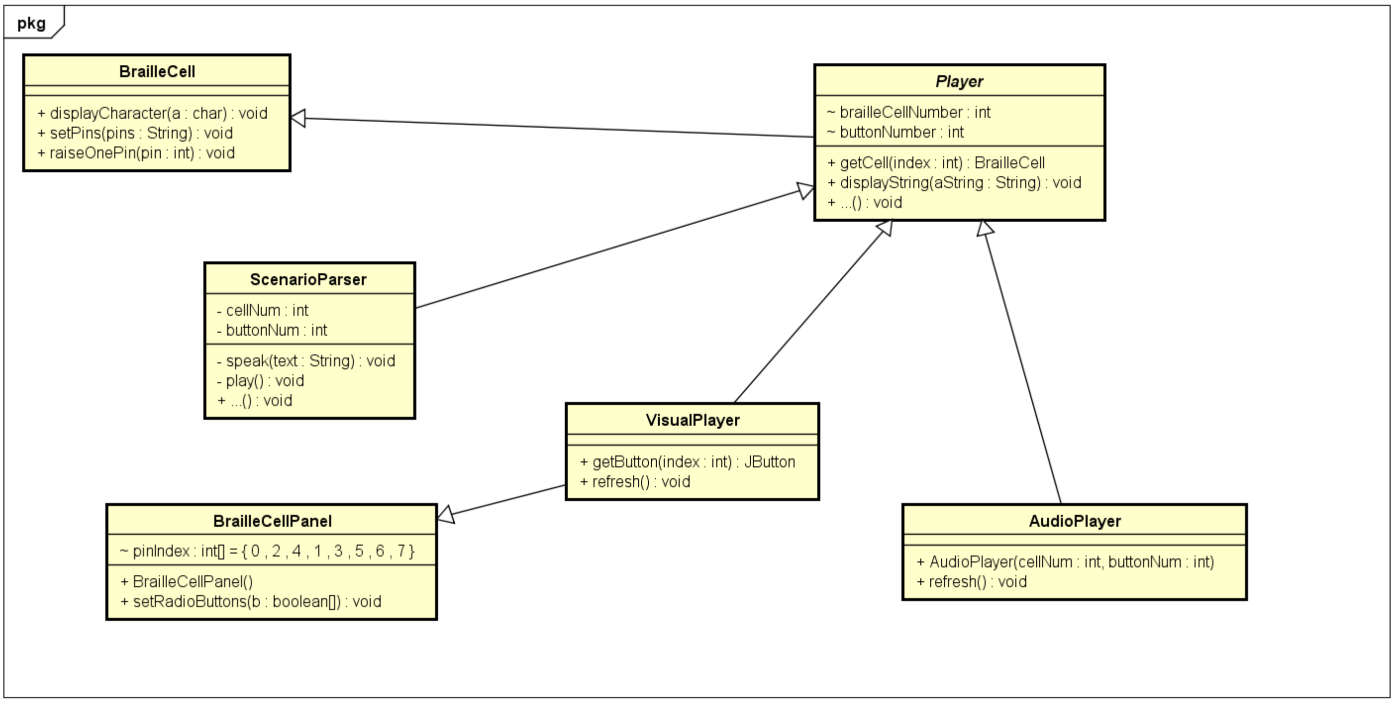
ScenarioComposer

Parses existing scenarios (‘.txt’ files) into the application’s model format. The imported scenario is converted into a ‘ListManager’ object. Allows for loading existing files.

GenerateTree

This class complements the ‘ScenarioComposer’ in that it converts the ‘ListManager’ object into a ‘.txt’ file. ‘GenerateTree’ class is a key aspect of exporting scenario files into a ‘.txt’ file. Allows of saving the current scenario as a file.

Braille Cell Panel Simulator


pkg
BrailleCell
+ displayCharacter(a : char) : void
+ setPins(pins : String) : void
+ raiseOnePin(pin : int) : void
Player
~ brailleCellNumber : int
~ buttonNumber : int
+ getCell(index : int) : BrailleCell
+ displayString(aString : String) : void
+ ...() : void
ScenarioParser
- cellNum : int
- buttonNum : int
- speak(text : String) : void
- play() : void
+ ...() : void
VisualPlayer
+ getButton(index : int) : JButton
+ refresh() : void
BrailleCellPanel
~ pinIndex : int[] = { 0 , 2 , 4 , 1 , 3 , 5 , 6 , 7 }
+ BrailleCellPanel()
+ setRadioButtons(b : boolean[]) : void
AudioPlayer
+ AudioPlayer(cellNum : int, buttonNum : int)
+ refresh() : void

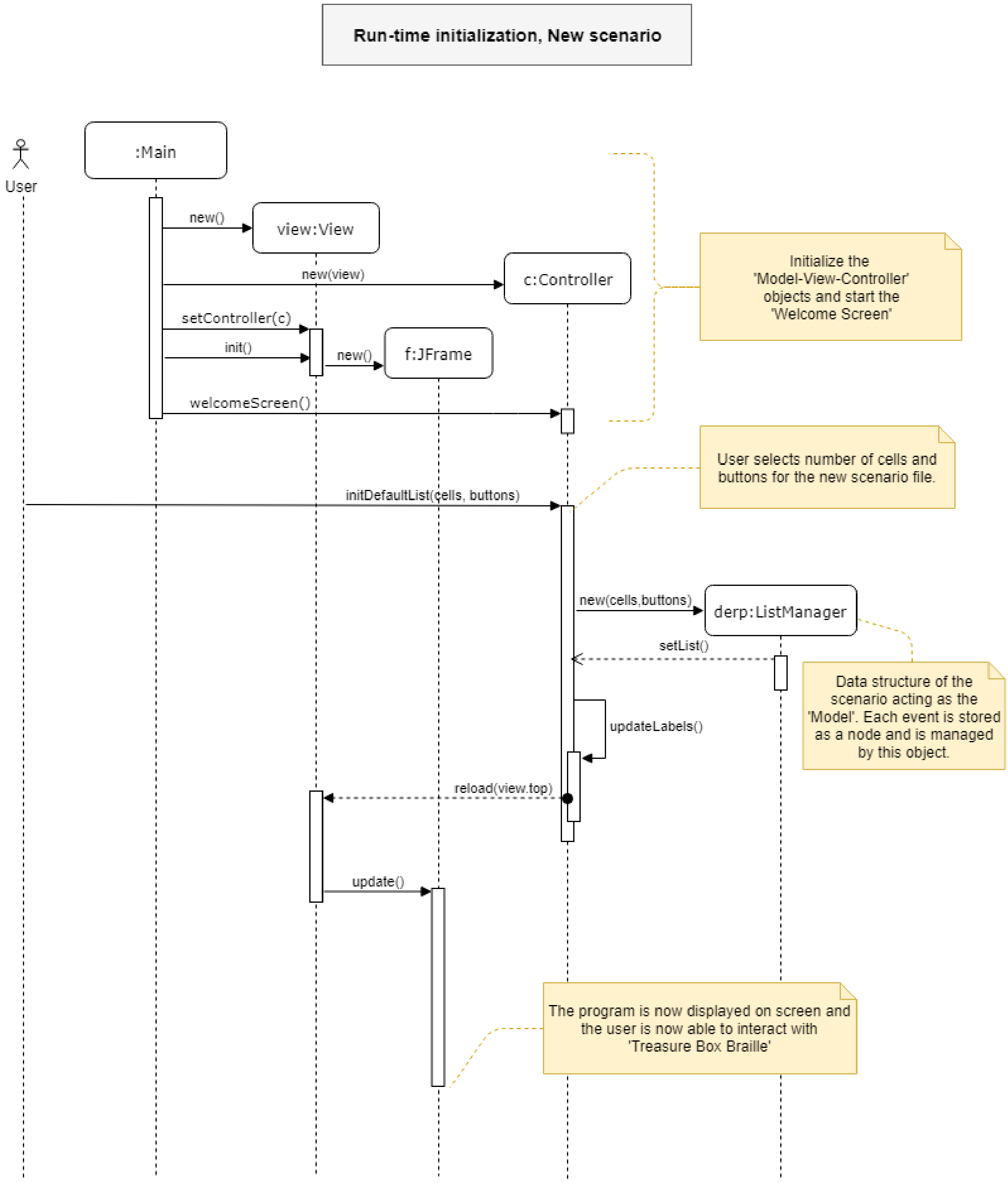
These classes were provided to us by the client. The UML diagram is shown here as a reference only. For more information, please contact Professor Vassilios Tzerpos.

The simulator is used in absence of the real ‘Treasure Box Braille’ cell panel. It simulates the pin configurations by visually displaying a virtual braille cell panel.

Sequence Diagrams

Initialization and New scenario

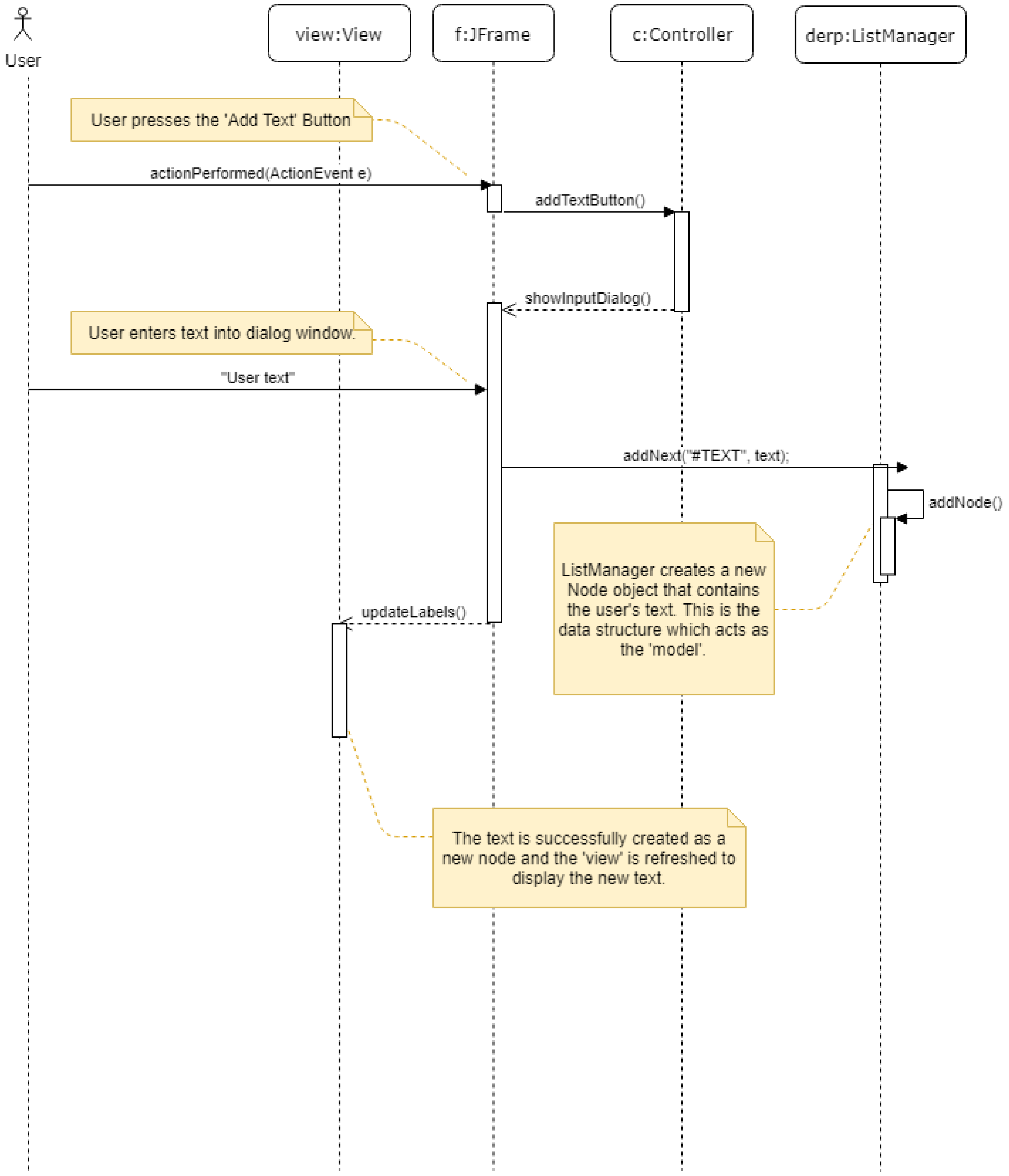
Treasure Box Braille


Run-time initialization, New scenario
User
:Main
new()
view:View
new(view)
c:Controller
setController(c)
init()
new()
f:JFrame
welcomeScreen()
Initialize the 'Model-View-Controller' objects and start the 'Welcome Screen'
User selects number of cells and buttons for the new scenario file.
initDefaultList(cells, buttons)
new(cells,buttons)
derp:ListManager
setList()
Data structure of the scenario acting as the 'Model'. Each event is stored as a node and is managed by this object.
updateLabels()
reload(view.top)
update()
The program is now displayed on screen and the user is now able to interact with 'Treasure Box Braille'

Add Text

Treasure Box Braille

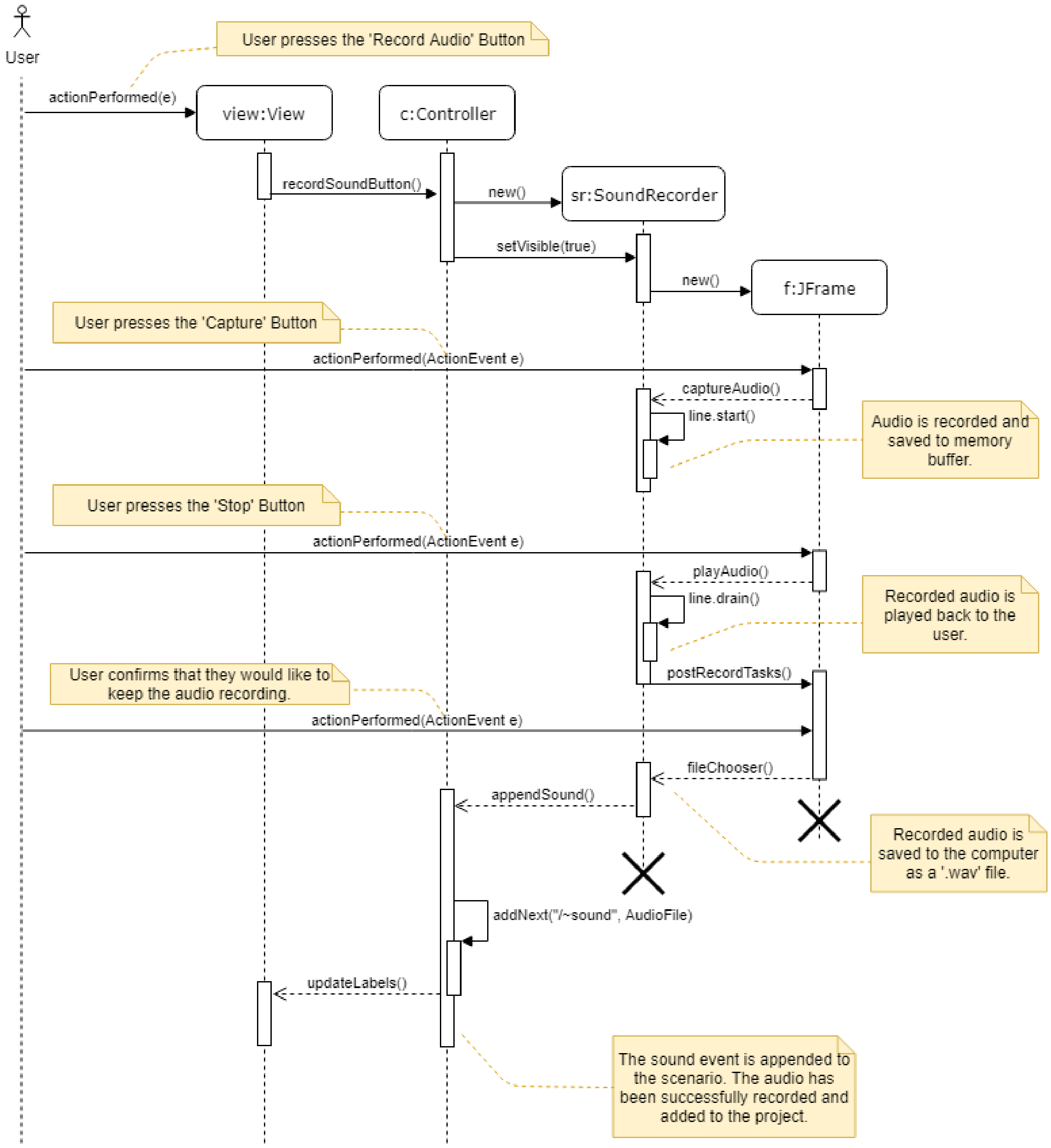
Adding text to existing scenario by pressing 'Add Text' button.


User
view:View
f:JFrame
c:Controller
derp:ListManager
User presses the 'Add Text' Button
actionPerformed(ActionEvent e)
addTextButton()
showInputDialog()
User enters text into dialog window.
"User text"
addNext("#TEXT", text);
addNode()
ListManager creates a new Node object that contains the user's text. This is the data structure which acts as the 'model'.
updateLabels()
The text is successfully created as a new node and the 'view' is refreshed to display the new text.

Audio Recording

Treasure Box Braille

Record Audio, Create new audio event



User presses the 'Record Audio' Button

User presses the 'Capture' Button

Audio is recorded and saved to memory buffer.

User presses the 'Stop' Button

Recorded audio is played back to the user.

User confirms that they would like to keep the audio recording.

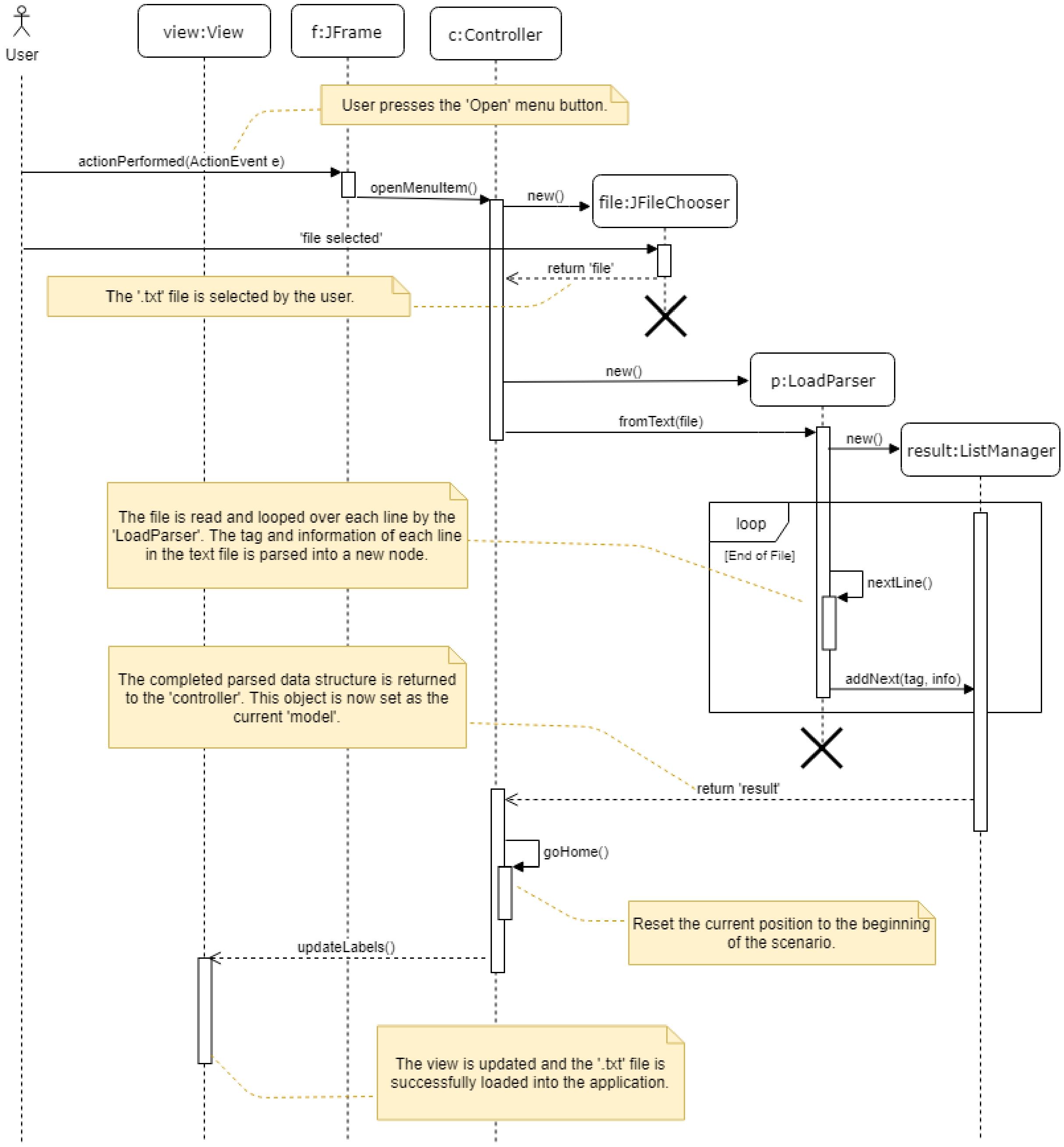
Recorded audio is saved to the computer as a '.wav' file.

The sound event is appended to the scenario. The audio has been successfully recorded and added to the project.

Load Scenario

Treasure Box Braille

Load an existing scenario '.txt' file



User presses the 'Open' menu button.

The '.txt' file is selected by the user.

The file is read and looped over each line by the 'LoadParser'. The tag and information of each line in the text file is parsed into a new node.

The completed parsed data structure is returned to the 'controller'. This object is now set as the current 'model'.

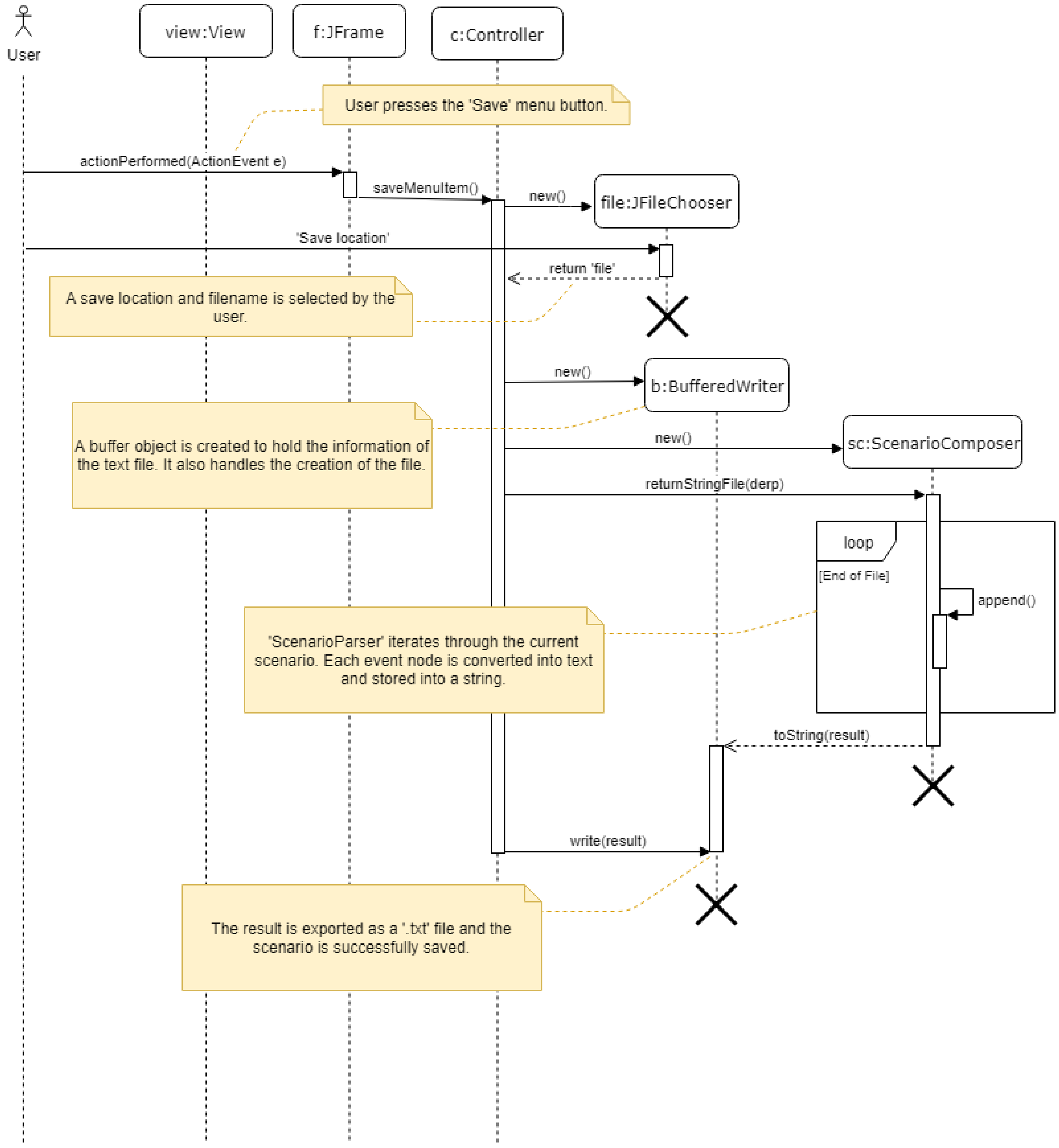
Reset the current position to the beginning of the scenario.

The view is updated and the '.txt' file is successfully loaded into the application.

Save Scenario

Treasure Box Braille

Save current scenario as a '.txt' file



User presses the 'Save' menu button.

A save location and filename is selected by the user.

A buffer object is created to hold the information of the text file. It also handles the creation of the file.

'ScenarioParser' iterates through the current scenario. Each event node is converted into text and stored into a string.

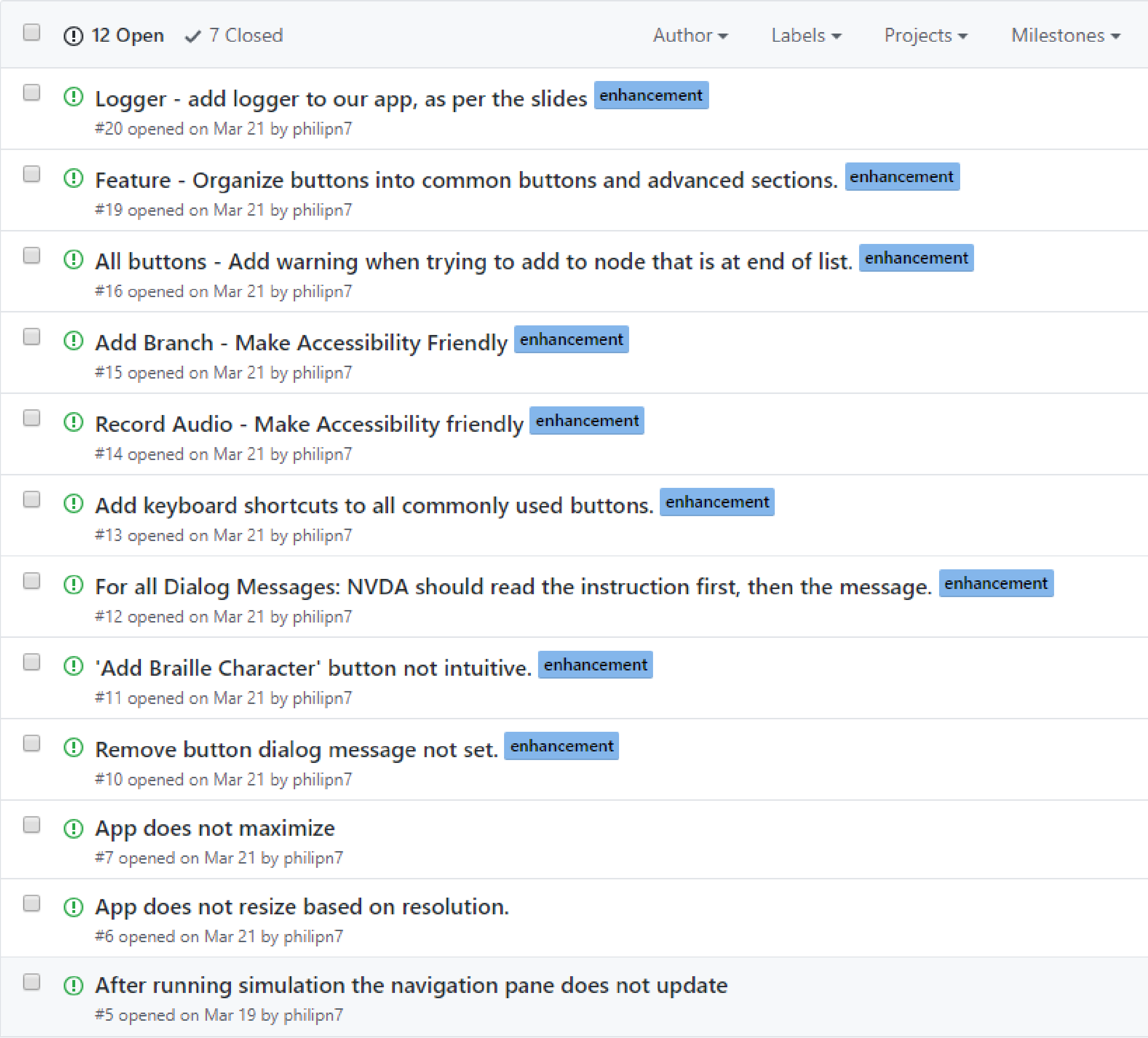
The result is exported as a '.txt' file and the scenario is successfully saved.

Maintenance Scenarios

This section will discuss elements which will allow future maintenance of ‘Treasure Box Braille’. The various aspects of maintenance will be covered.

Scenario 1: Correcting User-found Bugs

A logger system has been implemented. The log files can be submitted by the user to the developers so that the problem can be diagnosed and hopefully addressed. This requires the cooperation of the client to send us appropriate information to successfully fix the issues. In this scenario, open communication must be kept with the client either via email or phone support. Developers must also have time allocated for fixing bugs. The issue tracking system in GitHub is used to keep a list of known bugs.



□ 12 Open ✓ 7 Closed

□ Logger - add logger to our app, as per the slides enhancement
#20 opened on Mar 21 by philipn7

□ Feature - Organize buttons into common buttons and advanced sections. enhancement
#19 opened on Mar 21 by philipn7

□ All buttons - Add warning when trying to add to node that is at end of list. enhancement
#16 opened on Mar 21 by philipn7

□ Add Branch - Make Accessibility Friendly enhancement
#15 opened on Mar 21 by philipn7

□ Record Audio - Make Accessibility friendly enhancement
#14 opened on Mar 21 by philipn7

□ Add keyboard shortcuts to all commonly used buttons. enhancement
#13 opened on Mar 21 by philipn7

□ For all Dialog Messages: NVDA should read the instruction first, then the message. enhancement
#12 opened on Mar 21 by philipn7

□ 'Add Braille Character' button not intuitive. enhancement
#11 opened on Mar 21 by philipn7

□ Remove button dialog message not set. enhancement
#10 opened on Mar 21 by philipn7

□ App does not maximize
#7 opened on Mar 21 by philipn7

□ App does not resize based on resolution.
#6 opened on Mar 21 by philipn7

□ After running simulation the navigation pane does not update
#5 opened on Mar 19 by philipn7

Scenario 2: Application fails JUnit test(s)

The purpose of JUnit tests is to ensure the application works as intended on the ‘backend’ level. This system is not perfect, but it is essential to maintaining the bare-minimum working order of the application. As preventative maintenance, the tests should be kept up-to-date and more tests should be created as the application becomes more complex. Any failed tests should be addressed before the program is released to the client. This will ensure the quality of the product and prevent issues faced by the user.

A monthly review of the JUnit tests will ensure that they are relevant and can successfully detect problems soon after compiling.

Scenario 3: ‘Treasure Box Braille’ requires new features

As applications grow over time they must adapt to the user’s needs. New feature requests can be added to GitHub and the prioritized so that new features can be enjoyed by all clients. This is a key aspect to maintaining our software as it ensures the continued quality of ‘Treasure Box Braille’. Developers will need to be assigned to this maintenance role as new advancements in technology or platforms means that the application must be updated with features.

Scenario 4: Maintenance General Practices

This document outlines some of the decisions made in the design of our software. It is key that the developer understands the functions of the classes before they are modified. Familiarity with ‘Treasure Box Braille’ means that a developer can make informed decisions about future changes to the application.

It is also important to document and communicate the changes in some form so that future development can reference back to it. Documentation of this kind can also be shared with users as a summarized ‘change log’ / ‘patch notes’.

Maintenance programming is a key aspect of keeping the software functioning correctly even after ‘active’ development.