Official Player App

Design Document

Team 4

Allauddin, Qassim
Li, Derek
Mohamed, Yassin
Solovey, Artem

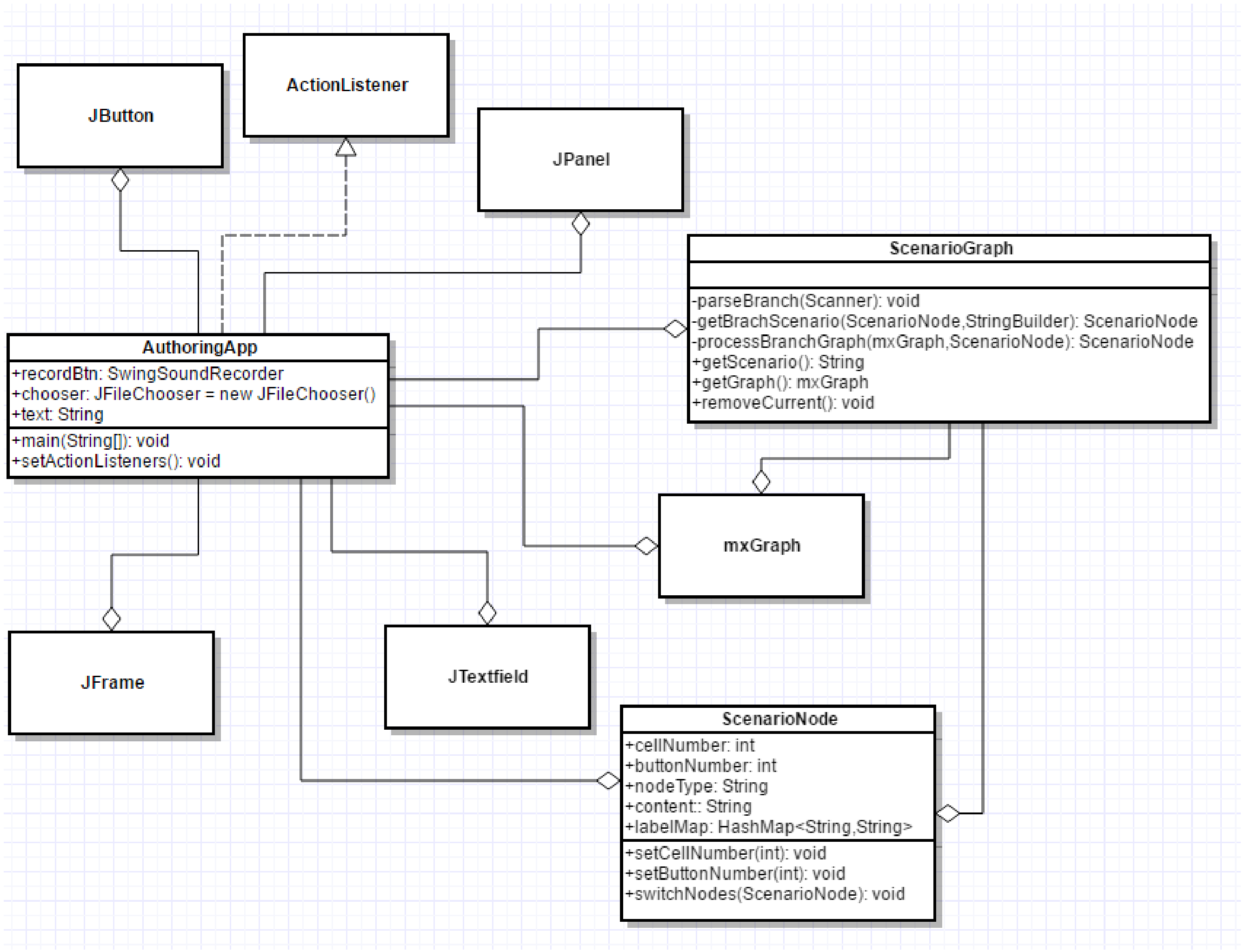
EECS 2311 - Software Development Project (Winter 2017)
Instructor: Bil Tzerpos

Table of Contents

1. Introduction

This document provides a detailed description of the design that was chosen for the official player application.

The design architecture for this calculator uses graphs and a GUI. The design also allows for eas implementation, testing, and maintenance of every component.



JButton
ActionListener
JPanel
ScenarioGraph
-parseBranch(Scanner): void
-getBrachScenario(ScenarioNode,StringBuilder): ScenarioNode
-processBranchGraph(mxGraph,ScenarioNode): ScenarioNode
+getScenario(): String
+getGraph(): mxGraph
+removeCurrent(): void
AuthoringApp
+recordBtn: SwingSoundRecorder
+chooser: JFileChooser = new JFileChooser()
+text: String
+main(String[]): void
+setActionListeners(): void
mxGraph
JFrame
JTextfield
ScenarioNode
+cellNumber: int
+buttonNumber: int
+nodeType: String
+content:: String
+labelMap: HashMap<String,String>
+setCellNumber(int): void
+setButtonNumber(int): void
+switchNodes(ScenarioNode): void

Figure 1: UML Diagram of the Authoring app

This UML diagram shows that the Authoring app has two main classes, SerigraphGraph and ScenarioNode. The additional classes used are for implementation of this architecture.

2. AuthoringApp

The Authoring app class is responsible for making scenarios in a GUI. This is achieved using different methods. Some of the important methods are described below. More information can be found in the requirements document

- **setActionListeners(): void** – This method adds and implements the buttons to the GUI. Each button has a method named actionPerformed that takes an ActionEvent, and each button does a different action.

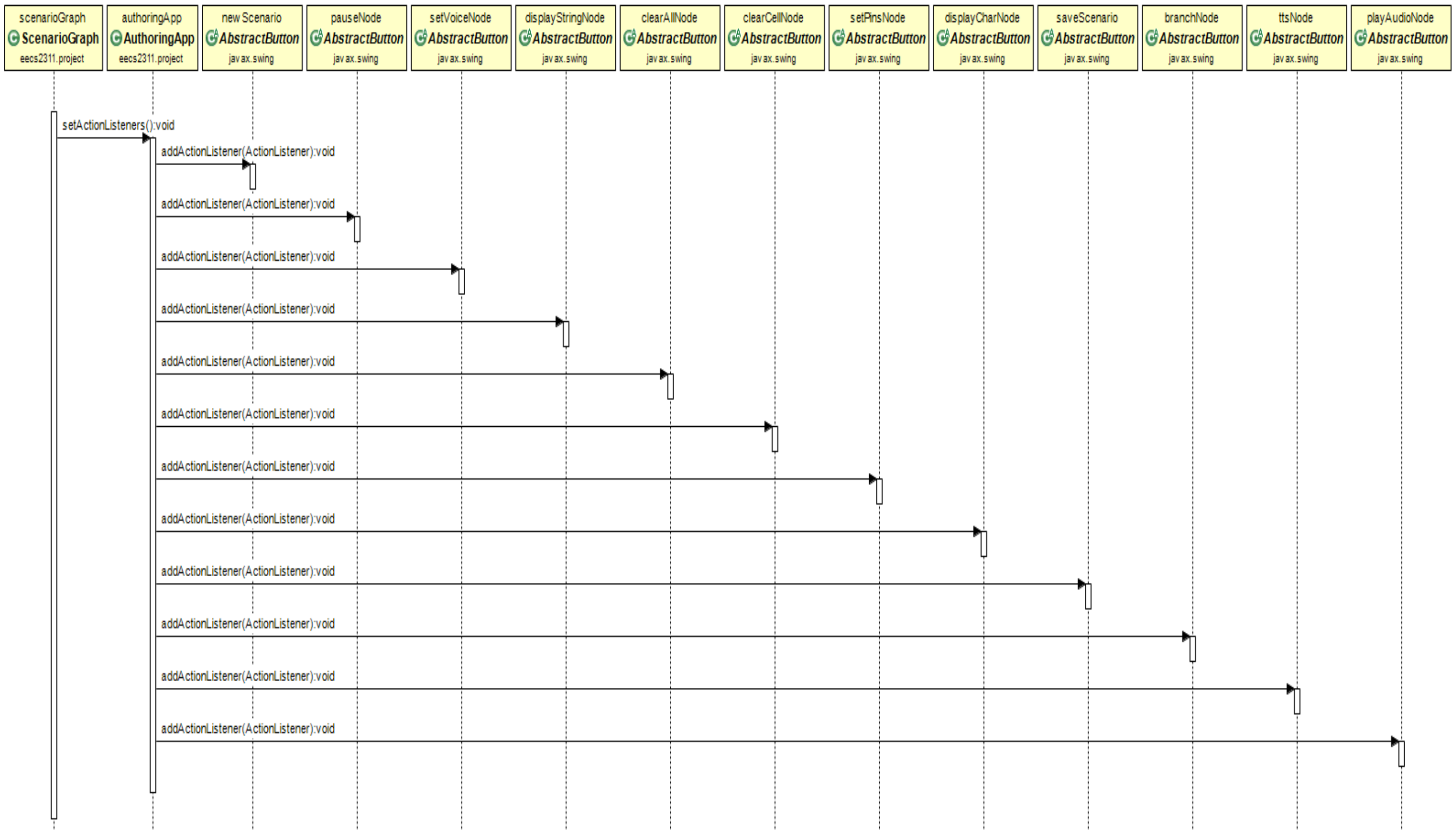


Figure 2: The Authoring app is calls ScenarioGraph to addActionListener method to create new buttons.

- **Main(String[]): void** – The main method sets up the GUI including all the buttons.

3. ScenarioGraph

The ScenarioGraph class is responsible for creating the graph using mxGraph, and is managed by the Authoring App. This is done using the important methods below. More information about the methods can be found in the requirements document.

- **parseBranch(Scanner): void** – This method allows for two branches to be created, then it reads each branch and returns to the main branch.
- **getScenario(): String** – This method iterates through the scenario graph and returns a string which contains the whole scenario.

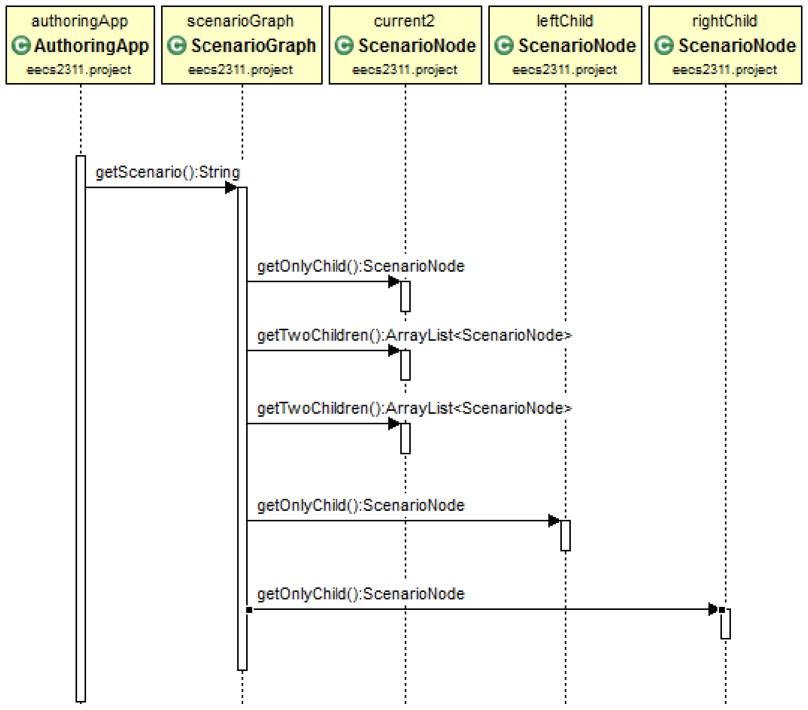


Figure 3: Sequence diagram of getScenario(): String

- **getBranchScenario(ScenarioNode,StringBuilder): ScenarioNode** – This method assists the **parseBranch** method.

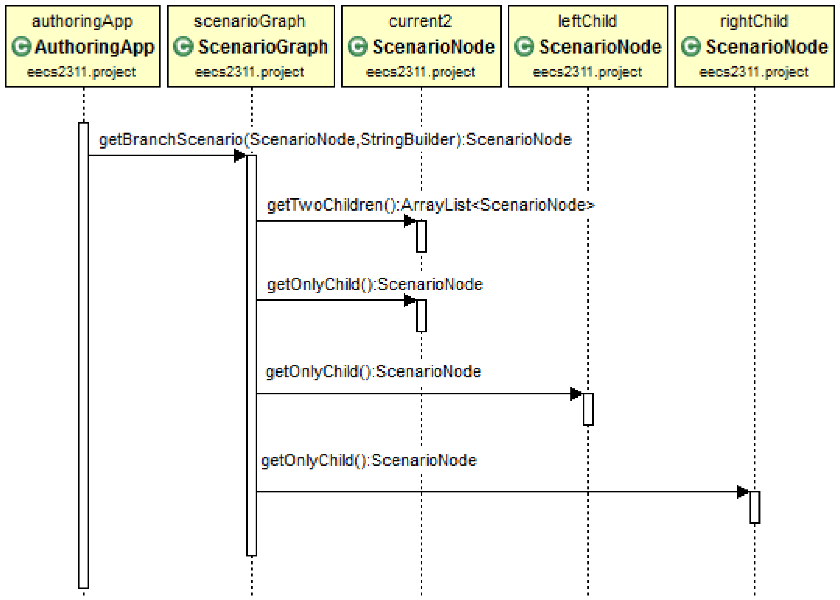


Figure 4: Sequence diagram of getBranchScenario(ScenarioNode,StringBuilder): ScenarioNode

- **getGraph(): mxGraph** – This method creates the design of the graph but does not display it.
- **processBranchGraph(mxGraph,ScenarioNode): ScenarioNode** – This method is to assist the getGraph method.
- **removeCurrent(): void** – This method removes the selected node from the graph.

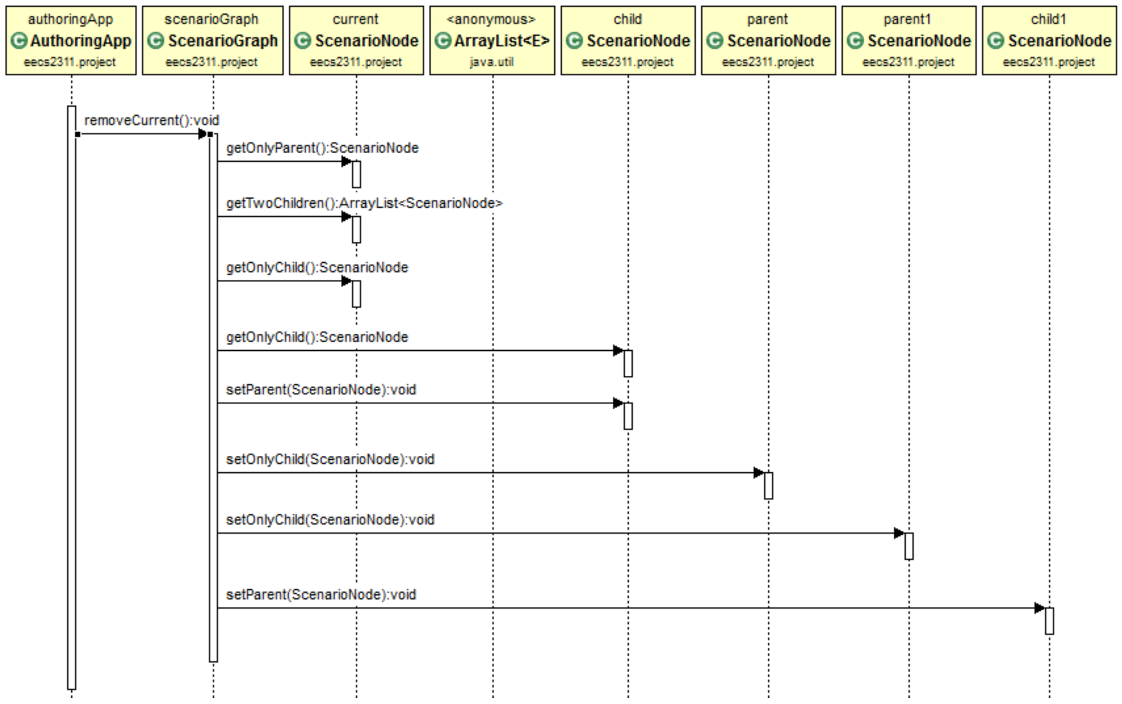

authoringApp AuthoringApp eecs2311.project
scenarioGraph ScenarioGraph eecs2311.project
current ScenarioNode eecs2311.project
<anonymous> ArrayList<E> java.util
child ScenarioNode eecs2311.project
parent ScenarioNode eecs2311.project
parent1 ScenarioNode eecs2311.project
child1 ScenarioNode eecs2311.project
removeCurrent():void
getOnlyParent():ScenarioNode
getTwoChildren():ArrayList<ScenarioNode>
getOnlyChild():ScenarioNode
getOnlyChild():ScenarioNode
setParent(ScenarioNode):void
setOnlyChild(ScenarioNode):void
setOnlyChild(ScenarioNode):void
setParent(ScenarioNode):void

Figure 5: Sequence diagram of removeCurrent(): void

4. ScenarioNode

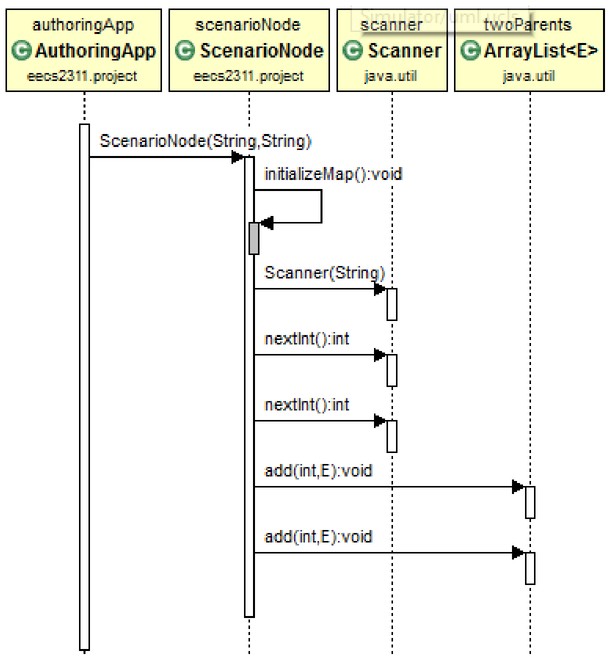


Figure 6: Sequence diagram of ScenarioNode(String,String)

The ScenarioNode class is responsible for letting the graph have objects that perform different events. The important methods are listed below. More information about the methods can be founding eh requirements document.

- **setCellNumber(int): void** – This method sets a cell to a certain number.
- **setButtonNumber(int): void** – This method sets a button to a certain number.
- **switchNodes(ScenarioNode): void** – This method switches the node with another existing node.

5. Maintenance

This code is designed to offer a lot of future functionality. The user can follow to code by the simple names used and is able to easily modify it. For example, changing the color of the buttons, or adding a button can be done simply.

5.1 Adding a button

Using JButton you create a static button. Then in the setActionListener method you would add the button to the addActionListener method and use the actionPerformed method to edit the name and allow it to perform certain actions. It will be similar to the already created buttons that could be used as a reference.