Design Document

Overall Design Concept

The GUI of the authoring app was built upon the perspective of the teacher using the application. It is made in such a way that the teacher cannot cause corruption to the scenario file by using our application. The GUI has been built to ensure that there is a logical flow when moving from one interface to the to the other, it does this by presenting only absolutely necessary functionality to the user at the given time. In terms of implementation, the authoring app was designed with the idea of separating each GUI component into their own class, having one class manage the state of the scenario being worked on, and using java interfaces to allow communication between various GUI components. It was built in this way to maintain code readability for the programmer as well as maintain optimal efficiency during runtime.

Starting the Authoring Application

When a user first enters the application, they are presented with a screen asking them to enter a scenario file name, the directory of where the scenario will be saved, the number of braille cells that will be used, and the number of buttons that will be used. It was decided that this would be the first menu the user saw to ensure that the program could not crash later in runtime. Reasons for the program crashing would be; attempting to save the scenario with no file name, attempting to save the scenario with no directory, attempting to add button functionality with no buttons defined, and attempting to display text across the braille cells with no braille cells defined. Because core functionality of the authoring app depended on these four key variables (scenario name, scenario directory, number of braille cells, number of buttons) it was extremely important that these were set before editing the scenario. Upon running the app, the EditorView class is instantiated and initialized to display menu described above. Once the user has successfully dealt with all options on this menu, the EditorPanel class is instantiated and set as the main content pane on the EditorView.

The EditorPanel Class

Upon successful setup of the Authoring app through the first EditorView menu, the EditorPanel is instantiated. It allows the user to see their current scenario, any to-do list items, add new sections to the scenario and save the current scenario buffer to the scenario text file. The EditorPanel contains a button that states “add new section”, when this button is pressed the BlockBuilder class is instantiated. The EditorPanel also implements the EditorPanelController so that methods inside the EditorPanel can be called from other GUI components.

The BlockBuilder Class

When the BlockBuilder class is instantiated, it must be passed an instance of EditorPanelController so that the BlockBuilder instance may update the EditorPanel upon addition of the section. The BlockBuilder will call the refreshBuffer method of the EditorPanel when the user has pressed the “finish section” button. The refreshBuffer method refreshes the “Scenario File” text area contained on the EditorPanel. Upon pressing the “finish section” button on the block builder window, the Scenario class’ copyBlockBufferToScenarioBuffer method is called so that the blockBuffer (which stores the text for the section currently being worked on) can be written to the scenarioBuffer (which stores the text for the

entire scenario file up to the current point in time). The BlockBuilder class also implements the BlockBuilderController so that it may be controlled by other GUI components.

The SoundRecorder Class

The SoundRecorder class is instantiated when the user pressed the “record sound” button on the BlockBuilder GUI. The sound recorder must be passed an instance of BlockBuilderController so that it can call the refreshBuffer method of the BlockBuilder class. The refreshBuffer method of the BlockBuilder will update the “Current scenario section” text area to contain the newly created sound file. When the user presses the “start recording” button on the SoundRecorder GUI, the SoundRecordingUtil class is instantiated. This class simply contains methods for reading from the built in microphone and writing the data to a .wav file.

The WriteToBraille class and the WriteToTTS Class

These classes are almost identical. The WriteToBraille class is instantiated when the user presses the “Write To Braille Cells” button. It allows the user to enter text that will be written to the braille cells. If text that has a length greater than the available braille cells is entered, it will show an error and not add the text. The WriteToTTS class is very similar except it is instantied when the user presses the “Add Text To be Spoken” button on the BlockBuilder GUI. This class allows the user to enter text that will be synthesized to speech. Both the WriteToBraille class and the WriteToTTS class require an instance of BlockBuilderController to be passed to their constructors so that upon successful completion of each action, the BlockBuilder can be updated to show the new additions to the current section.

The Scenario Class

The Scenario class contains all methods required to write to the buffer pertaining to the section being worked on (blockBuffer array lists), and the methods required to transfer the data from the BlockBuffer array list to the scenario buffer array list. When the user presses the “save scenario” button, the Scenario class manages writing all data from the scenario buffer to the actual text file.