Scenario File Format Documentation

EECS 2311 – Software Development Project

Instructor: Bil Tzerpos

1. Introduction:

This document explains how to use the ScenarioParser class, as well as the format expected from the scenario file. The file format supported for the scenario files is plain text.

1.1 How to Use the ScenarioParser class:

See the ToyAuthoring class in the provided code as an example.

When creating a ScenarioParser object you must pass a Boolean variable as an argument. True implies that the user is visually capable, while false implies that the user is visually impaired.

You must then call the setScenarioFile method on the ScenarioParser object providing a path to the ScenarioFile as an argument. This will execute the scenario either visually or aurally.

If the file does not exist, then the program would log it as an error and exit.

1.2 Mandatory Requirement of Scenario File:

There is always a possibility that you enter an incorrect file name or enter the file name of a non-scenario .txt file. The mandatory requirement that distinguishes a .txt file from a scenario .txt file, is that the scenario .txt file has the first two lines indicating the number of cells and number of buttons, as shown below:

```
Cells 2
Button 3
```

Every scenario file must have these as the first two lines of the document, where the numbers can be any positive integer as it represents how many Braille cells and JButtons will appear on the visual player. If the first two lines are not formatted as above, then the program will read an error message, log a more detailed error and then exit.

1.3 Text-to-speech or Carry out a Feature:

The difference between whether the program would interpret the lines in the scenario file as something to say out loud using text-to-speech, or as an action to affect the program is if there is a starting key phrase at the beginning of the line. There are a pre-defined set of key phrases, and if the line of the scenario file does not start with any of those key phrases, it is assumed to be text that should be

spoken out loud. An example is as follows:

Welcome to the moon!
/~sound:moonsound.wav

If those happened to be lines in the scenario file, it would interpret “Welcome to the moon!” as text to be spoken, because there was no indicator of a key phrase at the start of that line. While the line “/~sound:moonsound.wav”, starts with the key phrase “/~sound:”, which means that the text directly after the colon is to be interpreted as the sound file name to play. Assuming a file called “moonsound.wav” is in the project folder, the program would play that file. All the key phrases would be listed in the second section of this document, along with how to use each key phrase.

1.4 Format:

To maintain correctness of the program, the lines in the scenario file should start on a new line if there is a change in functionality of how to interpret the program. An example is as follows:

Good:

Welcome to the moon!
/~sound:moonsound.wav

Bad:

Welcome to the moon! /~sound:moonsound.wav

In the good example, there are two different actions occurring. One is that the first line is text to be spoken to the customer. Second is that the second line indicates to play the sound file called “moonsound.wav”.

In the bad example, two different actions are combined into one line. Since there is no key phrase at the start of the line, then the program considers that line to be text to be spoken. Therefore, the program will speak “Welcome to the moon! “, followed by “/~sound:moonsound.wav” (the program will say “slash” and “tilde”).

2. Key Phrase and Rules:

The various key phrases described below have rules that have to be followed to be considered well-formed. If that's not the case, then it is assumed that the scenario file is not formatted correctly and the program will log an error into a file called "ERROR_LOG.txt" and exit. Furthermore, the key phrases must be at the start of each line, and if there is no key phrase then the text is assumed to be spoken.

2.1 Key phrase: /~pause:

Structure for using pause key phrase is /~pause:num1, where num1 is any positive number. An example is as follows:

```
/~pause:3
```

To use the pause key phrase, it must be read as a separate line in the scenario file. Num1 is the number of seconds that the writer of the scenario wants to pause for, and in the case of the example, the program pauses for 3 seconds.

*Please note that if num1 is not a positive number, then the program will log an error message and exit, as it does not match the requirements for using this key phrase and therefore the scenario file is not well formatted. *

2.2 Key phrase: /~disp-string:

Structure for using disp-string key phrase is /~disp-string:string1, where string1 is a string to be displayed using the Braille characters. An example is as follows:

```
/~disp-string:yes
```

To use the disp-string key phrase, then it must be read as a separate line in the scenario file. This key phrase corresponds to the displayString () method in the Simulator class of the chosen simulator API. In the above example, this will change the Braille cells to display the word "yes" in Braille.

2.3 Key phrase: /~repeat and /~end-repeat

Structure for using repeat and end repeat key phrase is /~repeat and /~endrepeat, an example is as follows:

/~repeat
… <Text to be repeated>
/~endrepeat

To use the repeat and endrepeat key phrase, it must be read as separate lines in the scenario file. The line that is /~repeat means that the lines in the file after /~repeat will be considered as text to be spoken, and stored. The line that is /~endrepeat means that it will stop storing text to be spoken, and continue with normal operations. The text between /~repeat and /~endrepeat are stored for use, with the key phrase /~repeat-button:, which sets a button to repeat the stored text when pressed. In addition, for another occurrence of /~repeat, will wipe out the previously stored text in the first occurrence of /~repeat and /~endrepeat.

*Please note that if there is a line that has /~repeat, then it is expected that there is an occurrence of /~endrepeat somewhere in the scenario file, after the /~repeat line. If there is no such occurrence, then the program will log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.4 Key phrase: /~repeat-button:

Structure for using repeat-button key phrase is /~repeat-button:num1 , where num1 is the index of the JButton. An example is as follows:
/~repeat-button:1

To use the repeat-button key phrase, it must be read as a separate line in the scenario file. This key phrase sets the button indicated by num1, to repeat the text that was stored between /~repeat and /~endrepeat, when pressed. If there is nothing stored, then nothing will play.

*Please note that num1 must be a valid index of the JButton. Failure to provide this requirement will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.5 Key phrase: /~skip-button:

Structure for using skip-button key phrase is /~skip-button:num1 string1, where num1 is the index of the JButton and string1 is the identifier to skip to, and that num1 and string1 are separated by a space. An example is as follows:
/~skip-button:0 GoHere

… <Any other lines>
/~GoHere
… <Program starts off exactly after recognizing the string1 identifer>

To use the skip-button key phrase, it must be read as a separate line in the scenario file. This key phrase sets the button indicated by num1, to skip to the next occurrence of string1 in the scenario file, when that button is pressed. In the above example, … represents any lines in the scenario file that occur before the next occurrence of /~GoHere. The program resumes normal action the line after /~GoHere.

*Please note that the next occurrence of string1 (GoHere in this case), must appear in a new line and start with "/~" to distinguish itself as an identifier. Also, that num1 must be a valid index of the JButton. Failure to provide these specifications will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.6 Key phrase: /~user-input

Structure for using user-input key phrase is /~user-input, an example is as follows:
/~user-input

To use the user-input key phrase, it must be read as a separate line in the scenario file. This key phrase allows the user to press the button, as it pauses the program, and waits for a button press to be received to continue the program. This key-phrase must be used in conjunction with the /~skip-button: and /~repeat-button: key phrases, as the other two key phrases indicates that the buttons have been set to perform an action when being clicked.

*Please note that the usage of this key phrase is volatile. There must be a key-phrase /~skip-button:num1 string1 that occurred before the appearance of /~user-input, otherwise the program will be stuck in an infinite loop because only skipping to another portion of the scenario exits the loop. *

2.7 Key phrase: /~sound:

Structure for using sound key phrase is /~sound:string1, where string1 is the name of the sound file located in the project folder. An example is as follows:
/~sound:hi.wav

To use the sound key phrase, it must be read as a separate line in the scenario file. This key phrase plays the sound file with the name string1, or in the case of the above example, hi.wav. The sound file must be in the project folder, and be a .wav file.

*Please note failure to provide these requirements will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.8 Key phrase: /~reset-buttons

Structure for using reset-button key phrase is /~reset-buttons, an example is as follows:

```
/~reset-buttons
```

To use the sound key phrase, then it must be read as a separate line in the scenario file. This key phrase resets the action listeners of all of the JButtons, so that when you press one of the buttons, nothing will occur.

2.9 Key phrase: /~skip:

Structure for using skip key phrase is /~skip:string1, where string1 is the identifier to skip to. An example is as follows:

```
/~skip:Yes
…       <Any other lines>
/~Yes
…       <Program starts off exactly after recognizing the string1 identifier>
```

To use the skip-button key phrase, it must be read as a separate line in the scenario file. This key phrase skips to the next occurrence of string1 in the scenario file. In the above example, … represents any lines in the scenario file that occur before the next occurrence of /~Yes. The program resumes normal action the line after /~Yes.

*Please note that the next occurrence of string1 (Yes in this case), must appear in a new line and start with "/~" to distinguish itself as an identifier. Failure to provide this requirement will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.10 Key phrase: /~disp-clearAll

Structure for using disp-clearAll key phrase is /~disp-clearAll, an example is as follows:

/~disp-clearAll

To use the disp-clearAll key phrase, it must be read as a separate line in the scenario file. This key phrase clears all of the Braille cells on the simulator, as it corresponds to the clearAllCells( ) method in Simulator class of the chosen simulator API.

2.11 Key phrase: /~disp-clear-cell:

Structure for using disp-clear-cell key phrase is /~disp-clear-cell:num1, where num1 is an index of a Braille cell. An example is as follows:

/~disp-clear-cell:1

To use the disp-clear-cell key phrase, it must be read as a separate line in the scenario file. This key phrase clears the Braille cell at index num1, and lowering all the pins. The requirement is that num1 is a valid Braille cell index. This key phrase corresponds to the clear( ) method in BrailleCell class of the chosen simulator API.

*Please note that failure to provide this requirement will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.12 Key phrase: /~disp-cell-pins:

Structure for using disp-cell-pins key phrase is /~disp-cell-pins:num1 string1, where num1 is an index of a valid Braille cell and string1 is an 8-character sequence consisting of 0's and 1's. An example is as follows:

/~disp-cell-pins:1 1001011

To use the disp-cell-pins key phrase, it must be read as a separate line in the scenario file. This key phrase displays an 8-character sequence of 0's and 1's onto the Braille cell specified by num1. The requirements are that num1 is a valid Braille cell index and that string1 is an 8-character sequence that has 0's and 1's. This key phrase also corresponds to the setPins( ) method in the BrailleCell class of the chosen simulator API.

*Please note that failure to provide these requirements will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.13 Key phrase: /~disp-cell-char:

Structure for using disp-cell-char key phrase is /~disp-cell-char:num1 char1, where num1 is an index of a Braille cell and char1 is a valid English alphabet letter. An example is as follows:

```
/~disp-cell-char:1 P
```

To use the disp-cell-char key phrase, it must be read as a separate line in the scenario file. This key phrase displays the character at the specified Braille cell index. The requirements are that num1 is a valid Braille cell index and that char1 is a valid English alphabet letter which can be upper or lower case. This key phrase corresponds to the displayCharacter( ) method in the BrailleCell class of the chosen simulator API.

*Please note that failure to provide these requirements will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.14 Key phrase: /~disp-cell-raise:

Structure for using disp-cell-raise key phrase is /~disp-cell-raise:num1 num2, where num1 is an index of a Braille cell and num2 is an index of a pin. An example is as follows:

```
/~disp-cell-raise:1 5
```

To use the disp-cell-raise key phrase, it must be read as a separate line in the scenario file. This key phrase raises the pin at num2 on the Braille cell index of num1. The requirements are that num1 is a valid Braille cell index and that num2 is a valid pin index. This key phrase corresponds to the raiseOnePin( ) method in the BrailleCell class of the chosen simulator API.

*Please note that failure to provide these requirements will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *

2.15 Key phrase: /~disp-cell-lower:

Structure for using disp-cell-lower key phrase is /~disp-cell-lower:num1 num2, where num1 is an index of a Braille cell and num2 is an index of a pin. An example is as follows:

/~disp-cell-lower:0 3

To use the disp-cell-raise key phrase, it must be read as a separate line in the scenario file. This key phrase lowers the pin at num2 on the Braille cell index of num1. The requirements are that num1 is a valid Braille cell index and that num2 is a valid pin index. This key phrase corresponds to the lowerOnePin( ) method in the BrailleCell class of the chosen simulator API.

*Please note that failure to provide these requirements will cause the program to log an error and exit the program as it indicates that the scenario file is not well formatted. *