STANDARD OPERATING PROCEDURE:

Ren’Py Speller Storybook Project

Table of Contents

1. Introduction

1.1 Background

Communication is a fundamental right for the thousands of Canadians living with severe physical disability, but remains unattainable for many due to limitations of current assistive technology. Brain-computer interface (BCI) offers a new approach for brain-based communication through spelling applications, which decode patterns evoked in the brain when attending to letters on a virtual keyboard. However, many BCI-based spelling applications are not designed for children and can often induce boredom and fatigue. This project aims to develop an engaging and effective way for children with physical disabilities to practice spelling using BCI technologies. This is done through interfacing a Ren'Py built virtual storybook with an established BCI speller software (BCI2000).

1.2 Purpose

The purpose of this Standard Operating Procedure is to provide and describe the steps and processes to start, change and fix errors within the game.

1.3 Key Words:

BCI: Brain Computer Interface which uses EEG headsets (or electrodes) and machine learning algorithms to decode patterns within the brain.

SSVEP: Steady State Visual Evoked Potentials are signals that are evoked within the visual cortex of the brain after receiving a visual stimulus which has a frequency above 6Hz.

BCI2000: A BCI SSVEP based speller software provided by the NCAN group based in New York.

Ren'Py: Python-based game engine.

rpy files: Files that are specific to the Ren'Py game engine. It may include either Ren'Py or python code or both.

label: Ren'Py syntax used to control the flow of the game

2. Methods

2.1 Tools and Resources

Tool/Resource	Purpose
https://www.renpy.org/doc/html/	• Documentation of the Ren'Py game engine. This includes changing, text style, images, screens, etc.
https://lemmasoft.renai.us/forums/viewforum.php?f=8&sid=17fcf4bc5267019d114fc373335f8732 https://www.reddit.com/r/RenPy/	• Places to ask and post questions about Ren'Py functionality. Also used to find posts about similar issues that others might have faced.

Table 2.1 Tools and resources used to create the game.

2.2 Files

File name	Languages	Purpose
screens.rpy	Ren'Py	Consists of screens which includes the main screen, game screen and various preference screens
gui.rpy	Ren'Py	Consists of gui preferences such as text color, font size, font styles, text placement etc.
options.rpy	Ren'Py	Consists of transition preferences
custom_styles.rpy	Ren'Py	Consists of customized/user created text styles for different text situations
Extended_Classes.rpy	Ren'Py and Python	Consists of a class that extends the class VariableInputValue () so that it can change/update more than one variable when it is called
script.rpy	Ren'Py and Python	Consists of code used to control the main flow of the game through global variables, user defined functions and labels
Read_Initial_Files.rpy	Python	Consists of two functions used to read and parse the text in the files under the folder textfiles. One function is

		used to parse the story text files and the other is used to parse the word files.
Page_Class.rpy	Python	Consists of one class that defines the entire text of a page and includes methods used to obtain, display and highlight sentences or words of a sentence.
show_page_image.rpy	Python	Consists of three functions. One function is used to show the page image on only half of the screen, another is used to show the page image filling the entire screen and the last one is used to display the title page.
Creating_Lists.rpy	Python	Consists of a function used to create a list of page objects. The number of page objects depends on the number of pages in a given book.
Updating_Functions.rpy	Python	Consists of two functions. One function is used to update the page text. The other function is used to update the page image corresponding to the page text.

Table 2.2 A brief summary on the important rpy files used in the game.

2.3 Detailed Notes on Main/Important Files

2.3.1 script.rpy

Label Name	**Function**
label initialization	Initializes all the variables and functions that will be used to store and change information including - preference selections - custom input preferences - attempt limit - user input
label start	Starts the game by jumping to either the label passive_setting or label interactive_setting
label interactive_setting	Starts the interactive version of the storybook game (Has yet to be made)
label passive_setting	Starts the passive version of the storybook game. This includes first defining the default preferences - Background song choice - Story choice - Word length - Word File It also defines and initializes variables responsible for storing information which is then used to display the story text and story images on to the game screen. This includes - story name - page image number - page images - story text - page text number - a list of Page objects - a list of text dividers - current Page object.
label prepare_page	Collects all the sentences in a Page object using the get_sentences () method and stores these sentences in the instance attribute sentences. The specific number of sentences of a Page object is also found and stored in the variable called number_of_sentences. A counter i, is initialized to count the number of sentences displayed per

	Page object. Finally, a jump command to the label display_sentences is called.
label display_sentences	This label first checks whether or not there are sentences that have yet to be displayed. If there are none, the control of the game goes to the label turn_page. If there are, however, the control remains in the label display_sentences. If the control remains in this label, the variable word_change, which stores the input entered by the user, is cleared and the orientation of how the story will be displayed is specified. A dictionary to store sentence attributes and a counter to count attempts are also initialized. Finally, the Page object methods select_words_to_highlight () and display_sentence () are called.
label re_display_sentence	This label is responsible for checking whether or not the input entered by the user, which is stored in the variable word_change is correct. If it is correct, a reward message will be displayed. If it is incorrect, the variable word_change is cleared, and this label will be recalled which will then present the user with the same sentence. The maximum number of times that the same sentence is shown depends on the attempt limit. If the entered input is still incorrect after exceeding the attempt limit, an encouraging message with the correct spelling is displayed. Note that the counter i initialized in label prepare_page and checked in label display_sentences is updated by one each time the control is in this label.
label turn_page	This label calls two functions: update_pages () and update_page_image (). These functions update the Page object and updates the page image.

Table 2.3.1 Description of the function of the specific labels found in the main script rpy file.

2.3.2 Read_Initial_Files.rpy

Function Name	Caller	Function
read_story_file ()	label passive_setting in script.rpy	Responsible for reading all the text in the story text file and parsing it into a list of strings where each string corresponds to all the text on a page.
read_dolch_file ()	label passive_setting in script.rpy	Responsible for reading all the words in the word text file and parsing it into a dictionary where the keys are the words in the text file.

Table 2.3.2 Description of functions in Read_Initial_Files.rpy

2.3.3 Page_Class.rpy

Methods	Caller	Function
find_nth_punctuation ()	get_sentences () in Page_Class.rpy	Responsible for finding the nth text divider's index position. For example, this method can find the index position of the 3rd period found in the entire text of a Page object. It also checks for whether or not the text divider is right before or right after a quotation mark and adjusts the index position accordingly to include it.
get_sentences ()	label prepare_page in script.rpy	Responsible for collecting all the sentences found in the entire text of a Page object. It does this by first collecting and sorting all the index positions of all the text dividers into a list called sorted_order_of_puncutati

		ons. The sentences are then picked out by indexing the entire text using the index positions in the list.
get_words_in_sentence ()	select_words_to_highlight () in Page_Class.rpy	Responsible for collecting all the words in a given sentence by using a translation table to remove the text dividers/punctuation.
select_words_to_highlight ()	label display_sentences in script.rpy	Responsible for checking whether words or word categories that users want to highlight exist in a given sentence. If they do exist, these words are appended to the instance attribute list words_to_highlight. There are two word categories that are checked - length - word_file There are also two other options plus and minus that pertain to only the word length category. These are Boolean variables and evaluate to either True or False. If plus is True, all the words greater than the specified word length are also included in words_to_highlight. If minus is True, all the words less than the specified word length are also included in words_to_highlight.
highlight_words ()	display_sentence () in Page_Class.rpy	Responsible for applying the highlight to the word chosen. This is done by randomly choosing a word in the instance attribute list words_to_highlight. The

		word is then found in the sentence and substituted with its highlighted version. This method is also responsible for filling a dictionary called highlight_dict that stores a list of highlighted words and the updated sentence for display. This dictionary is returned to the method display_sentence ()
display_sentence ()	label display_sentences () or label re-display_sentence ()	Responsible for displaying a sentence onto the game screen and filling a dictionary called sentence_attributes which stores three things - input_word (word entered by user) - words_to_compare (list of words highlighted) - highlighted_sentence (sentence to be displayed) This dictionary exists and stores this information because it can be used to re-display a sentence when necessary.

Table 2.3.3 Description of methods used by the class Page in the Page_Class.rpy file.

2.3.4 show_page_image.rpy

Function Name	Caller	Function
show_page_image ()	label display_sentences in script.rpy	Responsible for rescaling an image to fit half of the game screen.
show_entire_image ()	label display_sentences in script.rpy	Responsible for rescaling an image to fit the entire game screen.
show_title_page ()	label passive_setting	Responsible for displaying the title page.

Table 2.3.4 Description of functions in the show_page_image.rpy file

2.3.5 Creating_Lists.rpy

Function Name	Caller	Function
listing_pages ()	label passive_setting in script.rpy	Responsible for creating Page objects and appending them into a list of Page Objects called page_list.

Table 2.3.5 Description of function in Creating_Lists.rpy file

2.3.6 Updating_Functions.rpy

Function Name	Caller	Function
update_pages ()	label turn_page in script.rpy	Responsible for updating the global Page object a_page with the most current Page object obtained from the global list of Page objects called pages. It is accessed from the list by using the global variable pg_text_number.
update_page_image ()	label turn_page in script.rpy	Responsible for updating the global variable pg_image_number which will be used to update the Image object pg_image.

Table 2.3.6 Description of functions in Updating_Functions.rpy file

2.3.7 screens.rpy

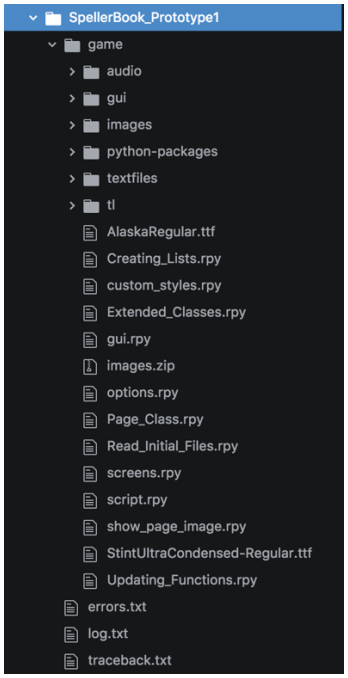
The screens that are important or have been altered are the following:

Screen name	Custom	Function
custom_say_screen ()	Yes	Used to show story text for stories that have an orientation that is “vertical” or has a traditional storybook style (one side is the picture, while the other side is the text) and uses the custom_input () screen
custom_input ()	Yes	Used to show and accept the input entered by the user. Also, it displays the attempts left the user has to spell a word
quick_menu ()	No	Shows the small white textbuttons at the bottom of the game screen that allows users to exit the game screen easily
main_menu ()	No	The main screen when the game is launched from the Ren’Py launcher. Consists of 5 image buttons
word_file_based ()	Yes	The preference screen that shows the default dolch list word files available. It also enables the user to input the name of their own word file
word_length_based ()	Yes	The preference screen that shows the different default word lengths available. It also enables the user to input their own word length. Note there is also a plus and minus sign to allow users to specify if they want to include word lengths that are greater than or less than the specified word length.

story_choices ()	Yes	The preference screen that shows the different default stories available. It also enables the user to input the name of their own loaded story
invalid_input_error ()	Yes	This screen appears after the user enters a custom input in the preference screens that does not make sense
preferences ()	No	This is the main preference screen that shows all the possible preferences available to the user
nvl ()	No	Used to show story text for stories that have an orientation that is “horizontal” or has a non-traditional storybook style (shows the picture first and then overlays the text on top of the picture using a semi-transparent textbox)
nvl_dialogue ()	No	Used to format the story text that shows up on the nvl screen and uses the custom_input () screen

Table 2.3.7 Brief description of important/key screens used to create the main menu and game screens.

3. Folders



Folder Name	Purpose
SpellerBook_Prototype1	Encompasses all folders, .rpy files, and error traceback text files related to the game
game	Encompasses all folders and .rpy files used to control and build the game
audio	Consists of .mp3 or .wav music or sound files used in sound effects and background music in the game
gui	Consists of sub folders responsible for the images used for background screens, textboxes, buttons, sliders, etc.
gui/button	Includes images used for image buttons on main screen
gui/overlay	Includes the background images for the confirm screen, game menu screen and main menu screen
gui/slider	Includes images used for the sliders found in the preference screen

images	Includes all the page images of the loaded stories. Also, this is the place where new page images of new stories should be placed
textfiles	Includes all the story text files and the dolch list word text files. Also, this is the place where new story text files and new word text files should be placed

Table 2.4 A brief summary of the key folders used to create and build the game

4. How to Load Custom Story

The following steps describe how to load in a new story.

<u>Images</u>

1. Find an eBook online and take screen shots of the pictures. All of the default stories already loaded in the game are sourced from the Calgary Public Library.

2. After the screen shots of every picture has been taken, save the screen shots as png files for better quality and follow the format shown below or in Figure 4.1.

   [story_name]_pg[#].png
   Ex. I_Want_My_Hat_Back_pg3.png

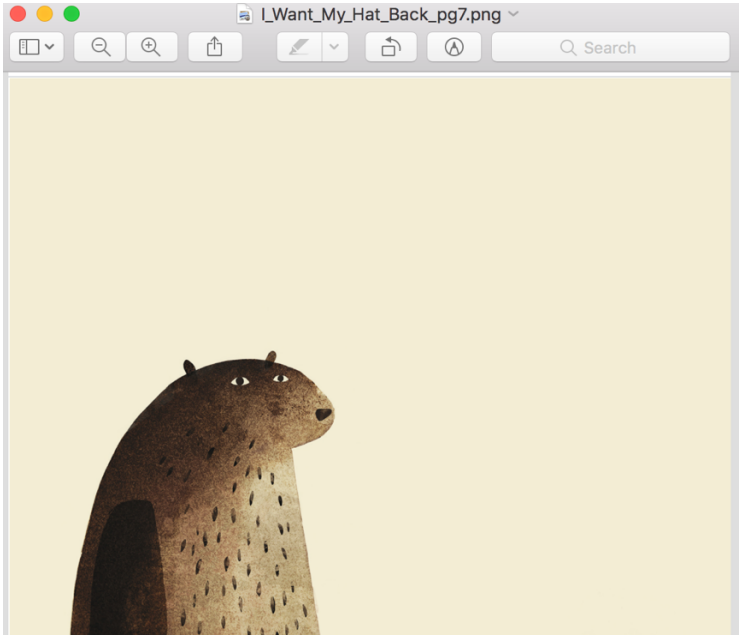


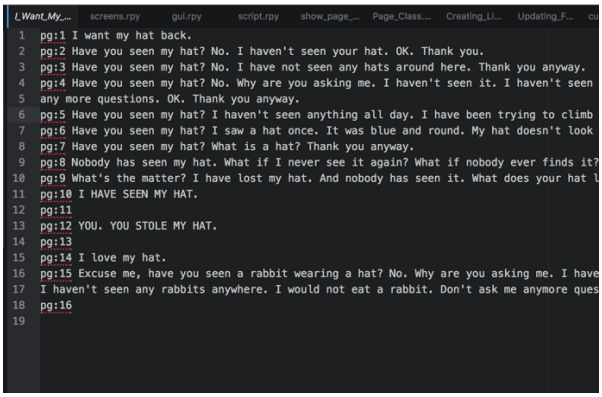
Figure 4.1 A renamed page image for the story I Want My Hat Back

   The spaces in the title of the book are separated by underscores and the page number is appended to the end of the title.

3. Once all the page images are renamed, save them to the folder named gui/images.

Text

4. For the story text, open a new text file and follow the format in figure 4.2 below. Note that the code is insensitive to newlines and there should be no problems so long as the label “pg:#” is written. If there is no text on a page, simply type out the label “pg:#” and leave the space following it blank.



```
pg:1 I want my hat back.
pg:2 Have you seen my hat? No. I haven't seen your hat. OK. Thank you.
pg:3 Have you seen my hat? No. I have not seen any hats around here. Thank you anyway.
pg:4 Have you seen my hat? No. Why are you asking me. I haven't seen it. I haven't seen
any more questions. OK. Thank you anyway.
pg:5 Have you seen my hat? I haven't seen anything all day. I have been trying to climb
pg:6 Have you seen my hat? I saw a hat once. It was blue and round. My hat doesn't look
pg:7 Have you seen my hat? What is a hat? Thank you anyway.
pg:8 Nobody has seen my hat. What if I never see it again? What if nobody ever finds it?
pg:9 What's the matter? I have lost my hat. And nobody has seen it. What does your hat l
pg:10 I HAVE SEEN MY HAT.
pg:11
pg:12 YOU. YOU STOLE MY HAT.
pg:13
pg:14 I love my hat.
pg:15 Excuse me, have you seen a rabbit wearing a hat? No. Why are you asking me. I have
I haven't seen any rabbits anywhere. I would not eat a rabbit. Don't ask me anymore ques
pg:16
```

Figure 4.2 Format of story text file

5. Name the story text file following the format below and save it to the folder named gui/textfiles.

   [story_name].txt
   Ex. I_Want_My_Hat_Back.txt

6. Lastly, go to the Preferences screen and select Browse and Select Stories. In the custom input box as shown in Figure 4.3 enter the name of your custom story. Include the underscores in your title but you do not need to include the .txt ending. Then press start.

Figure 4.3 Story Selection screen with custom story input

5. How to choose Custom Word Lengths or Load Word Files

The following steps show how to input a custom word length or import a custom word file.

Word Length

1. Go to the Preferences screen and look for preference Word Origin and choose Length Based.

2. In the custom input box as shown in Figure 5.1 enter the number of your custom length. Do not include a space before or after the number (this is space sensitive and has yet to be fixed).

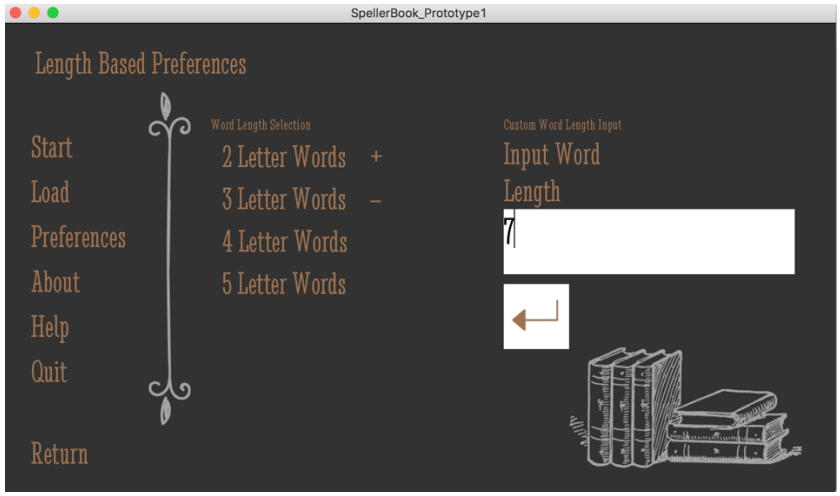


Figure 5.1 Length Based Preferences screen with custom length input

3. Start game.

Word File

1. First create a new text file with a list of words that you would like to target. These words are separated by a comma and you may follow the format as shown in Figure 5.2.

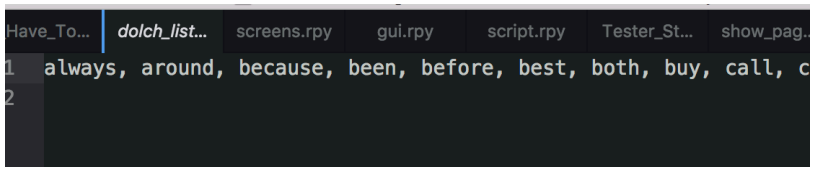


Figure 5.2 Format of word text file

2. Save the finished word text file to the folder named gui/textfiles.

3. Then, go to the Preferences screen and look for preference Word Origin and choose Word File Based.

4. In the custom input box as shown in Figure 5.3 enter the name of your custom word file. Include the underscores in your title but you do not need to include the .txt ending.

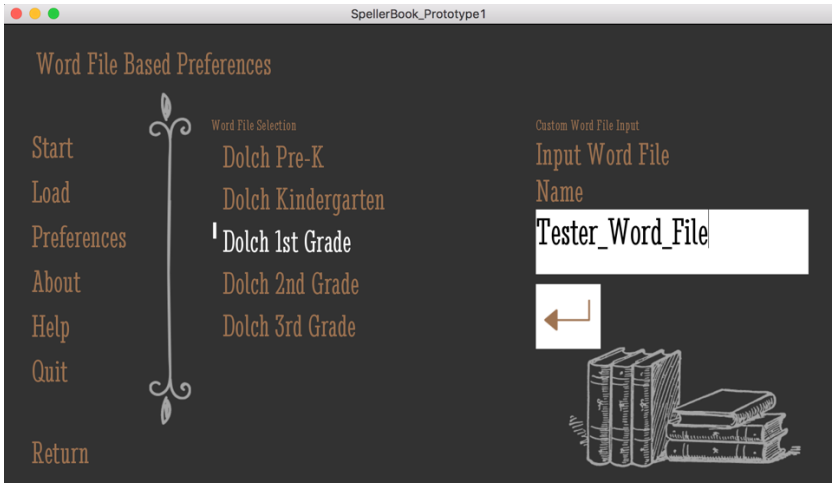


Figure 5.3 Word File Based Preferences screen with custom word file input

5. Start game.

6. Troubleshooting Tips

Some tips for troubleshooting include.

- Looking at the Developer Tools provided by the Ren'Py official website
  - https://www.renpy.org/doc/html/developer_tools.html

- Asking on the forums listed in section 2.1 Tools and Resources

- The console is usually accessed with the command Shift+O. However, because the game continually prompts the user for the input this does not work. To disable the input, we must comment much of the code out and this will depend on what we would like to test.

- Contact Lisa Huang at lisa.huang1@ucalgary.ca