COSC345 Software Engineering

CommunAphasia:

Final Release Documentation

RedSQ Team:

Winston Downes	(2872112)
Max Huang	(4427762)
Mitchie Maluschnig	(9692597)
Sam Paterson	(3175949)

01 October 2018

How to use the app

In order to build our project, you must open “CommunAphasia.xcworkspace” file rather than the project, then build targeting the iPad Pro (10.5 inch) simulator (currently does not resize well to other iPad models). From the main menu, the user will be greeted with a pop-up asking if the user would like to take a tutorial to show them how to use the app, but this only occurs if it is the first time running the app. The tutorial walks the user through how to build an “Image to Text” or “Text to Image” sentences, using plurality and tense selections. If they decide not to take the tutorial, they will have two options: “Image to Text” or “Text to Image”.

Image to Text – the user is presented with a selection of images, organised into categories which can be changed by tapping the tabs on the right-hand side. The user can pick an image (or multiple) from the selection; for each image chosen, the user may be presented with additional options depending on the parts of speech pertaining to the word selected, such as tense or number, denoted by the different coloured borders. If the selected image represents a noun, the user will be presented with a pop-up and they can choose whether they are describing a singular or plural instance of that noun; conversely, if the chosen image represents an adjective or verb, the user can choose whether it is past, present, or future tense. The selected image (with its type) will then be placed in the bottom area of the screen, to show which images the user has selected. Once the user is happy with their ‘sentence’, they can tap “Done” and be greeted with the ‘image sentence’ as words, which includes some processing (adding of appropriate function words and making it more coherent), along with their original image selection at the bottom of the screen.

Text to Image – the user inputs a sentence and taps “Done” – they will then be greeted with a screen which shows the result of converting that sentence into images, with the original text at the bottom of the screen.

Pressing the “back” button, in both cases, will move back a screen and let you re-input a different ‘sentence’ from scratch.

How it works

<u>Text-to-image</u>

The user types in a sentence they want to construct, the text is parsed to our algorithm and the corresponding images are pulled out of the database and displayed on screen. If

the user enters a sentence containing words which do not qualify as standard English, they are confronted with an error message reading “Invalid sentence. Please try again.” If the user enters valid words which are not contained within the database, those words and highlighted red and the user is presented with synonymous, which are in the database, to choose from – they can then move onto the next word by using the ‘forward’ and ‘back’ arrows to cycle through the invalid words; at each word, they are presented with a scroll view that allows them to select alternative words which are in our database. As the user selects alternate words, they replace the words in red, until eventually the sentence only contains words available from the database. Once an invalid word is replaced, it is highlighted green to indicate it is valid.

All the available words to be used to construct sentences are contained in a CSV file named “images.txt”, using the first element of each line.

Image-to-text

The user is, by default, presented with a selection of commonly used images from which they can use to make sentences, they can change this selection by selecting the category tabs on the right. Within a category, the images are separated based on their grammatical type (e.g. noun, verb) and these are segregated into columns which are ordered by the frequency that the user selects the images. If there are too many images in a column the user can scroll down to see more images, but we dynamically scale the image types to cover many columns if a lot of scrolling would be required.

The user constructs a sentence by tapping on the images they want (in order) and these are placed in the field at the bottom of the screen. Once the user is finished, they tap the “Done” button and the images are parsed into our image-to-text algorithm which identifies the parts of speech for that word (e.g. noun) and generates a coherent sentence by padding the sentence out with the appropriate function words depending on the current picture and the surrounding ones. For example, if the current picture is a cat (noun), the definite article “the” will be placed in front of the word “cat”. If the next image is the picture representing the verb “eat”’, then the third-person verb to be ("is”) will be placed next in the string, after the word “cat” and the present participle form of the word (“eating”) is placed next in the sentence, thereby forming the sentence “the cat is eating”. Some of the images in our app are not intuitive and we do not expect the user to memorise what every picture is, so we have added a functionality where a long press of an image will cause a popover to display, showing the user the word that the image is trying to represent.

We currently have added an adequate amount of images to our database so the users can construct a decent amount of common sentences. The following sentences are some of the examples of ‘translation’: (words-to-images)

- The blue fish is sleeping
- I am eating dinner
- The big man will want to wait for the small woman to call him

Justifications

We have decided to only support the iPad and not to incorporate cross device support to the iPhone. This is mainly because rescaling most elements of the app, to support a different screen size, would largely exceed the expected workload of this course. Secondly, if the app was resized to an iPhone screen size, we anticipate that it would be hard for the aphasic person to select specific images. The screen size would also limit the number of images we could fit on the screen, most likely 3-4 images as opposed to 20 or so on an iPad. We also wanted to focus on making a good quality app, so we chose to prioritise making the app as good as we could rather than spending time supporting a device which is less practical and performing a task which has less educational benefits (as it is entirely just resizing elements to a smaller screen which is tedious and time consuming). We left this decision until now because we didn’t understand how large the task would be and we would have had to wait until the iPad version is done before making the conversion (so we don’t have to repeat the same tasks twice on different devices).

It was difficult to comprehensively test the UI functionality so we just tested the image to text algorithm for a variety of random sentences to make sure there were no issues. We tested the text to image algorithm with unit testing by generating 1000 random sentences of varying lengths, each with randomised words of varying lengths sourced from a collection of alphanumeric characters. We also tested with valid English words manually to make sure the output is as expected, but used unit testing to ensure that the program will not crash for alphanumeric input.

Issues encountered

- Our app not scale properly for devices other than then 10.5” iPad Pro even if using constraints, the UI elements do not scale well.
- The app cannot compare apostrophe character derived from the input text field to a string containing only an apostrophe character properly. It turns out the apostrophe

from the text field does not have an ASCII encoding so we had to do a comparison with Unicode values. We achieved this by comparing the input apostrophe Unicode value to integer values of 8217, and also 39 because that's what an apostrophe as a string would be.

- If the same image is used multiple times in a single sentence, then the frequency is only incremented once.
- If there are two invalid words which are the same, our program will only be able to find the first occurrence and completely ignores the second, this is due to the way we have implemented check algorithm.
- The app can sometimes lag on the simulators (on the lab computers) but doesn't appear to on actual physical devices.
- Tuples vs custom ImageCell class. During most of our early work of the app we used tuples to pass information around, however this soon became less practical, so we created an ImageCell class to do this instead. We encountered a lot of problems in converting from tuples to ImageCells and still haven't converted all of the tuples, as this would require a refactor of most of our code. This mix of ImageCells and tuples resulted in issues, such as expensive sorting algorithms as we couldn't just call an ImageCells `freq` field because it would sometimes be a tuple. Another issue with ImageCells and tuples was when we put a border on an image we would have to shrink the image to stop the border from overlapping but the conversion from ImageCell to tuple meant this would happen multiple times resulting in image shrinkage.

Justifications

We were unable to find Wernicke's aphasia afflicted persons, so we asked friends to test our app for us. The only instruction we gave them was to tap the tutorial button to start. This helped us get a general idea of how our app would be used by someone who has no idea how it works. We found that "Image to Text" was rather difficult to understand how to use, whereas "Text to Image" seemed quite intuitive to the testee. The biggest obstacle the testees faced was understanding how to chain an image sentence together in the right order to convey their intended message. We realised that due to our testees being non-affected persons, they were able to construct an image sentence, read the corresponding text output and restructure their image sentence again to try and get a meaning semantically closer to that which they were trying to convey. This can be a huge obstacle for aphasic patients

because they would not be able to read the output sentence to make sure it was delivering the intended message, and if not, adjust it accordingly – we fear this is the biggest drawback of our application; however, it is unavoidable for certain words, because they are quite abstract and do not lend themselves well to portrayal. As an example of this, try to imagine a picture for the word ‘may’, or ‘want’. Related to this, we also noticed that users found it hard to understand what the images convey, which is why we implemented the popover functionality showing the selected word. This added functionality will also be able to help aphasic patients, because they will often have some spared comprehension of single words. Even if single wording-naming is quite impaired with particular patients, then it is expected that more time using the app will increase the familiarity with the obscure pictures.