COSC345 Software Engineering

CommunAphasia: Alpha Release Documentation

RedSQ Team:

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

28 May 2018

How to use the app

Build project on CommunAphasia, targeting the iPad Pro (10.5 inch) simulator (we have not got screen sizing working perfectly yet). From the main menu, the user will have two options: “Image to text” (on the left) or “Text to image” (on the right).

Image to text – the user will be 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, 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

Text-to-image

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 invalid words, they are confronted with an error message displaying “Please enter a valid sentence”; if they enter valid words which are not in our database, those words are highlighted red and the user is presented with synonymous words which do exist in the database. As the user selects alternate words, they replace the words in red until eventually the sentence only contains words available from the database.

These sentences can also be translated into words by pressing the corresponding pictures in the order that one would utter the words. All the available words to be used to construct sentences (at this stage) are contained in a CSV file “images.txt”, using the first element of each line. Since adding more images is simply content generation, it should be even more comprehensive by the final release.

Image-to-text

The user is presented with a set of commonly used images from which they can use to make sentences.

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”.

Currently, we do not have a substantial amount of images in our database so only a limited amount of sentences can be made. 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

Issues encountered

There are prepositions that are general, in that they can be used with a verb and not change the meaning of that verb. For example, you can put a stone ‘in’, ‘on’, or ‘near’ something and what you are doing in all these cases is ‘putting’; however, waiting ‘for’ someone is different from waiting ‘on’ someone (such as what a waiter does). It was decided that we would choose the most common verb-preposition pair (‘waiting’ and ‘for’ in this example).

For example, modal verbs: it is not altogether apparent how best to represent, for example, the word ‘can’ as a picture. Initially we had decided to just use the first letter of the modal verb on a coloured tile (eg a ‘C’ on a blue tile for ‘can’); however, we decided to leave all text out of the pictures to prevent disorder-related difficulties. Its simple appearance notwithstanding, a person with Receptive Aphasia will perceive a ‘C’ as a type of picture anyway (that is, the linguistic component of this letter will not be conserved), therefore we decided to use pictures (a thumbs up, in this case), even if the relation between the picture and the word is somewhat cryptic.

We needed a way to give suggestions for any word the user input, for which we had no entry in our database. We decided the best way to do so was to use an online API, we ended up using WordsAPI which allows 2,500 requests a day. The only issue was that we didn’t know how to implement a URL GET request to get our data because it required certain keys in order for us to retrieve information from that site. Max spent most of a week learning about RESTful API calls and using delegates to implement a URL request. Once the request was working, handling and converting a JSON file to a useable format was the next issue to be tackled - turns out it wasn’t hard to convert a JSON file to a String array. We also implemented a timeout of 3 seconds to prevent the request blocking execution of the app. The synonyms in the array are then filtered so that only words that exist as entries in our database are passed on to be displayed as suggested alternatives to the user.

There was an issue where invalid words (i.e. words which do not exist in our database) would cause words in our database to become ‘invalid’ unexpectedly.

We encountered an issue with the lemmatization of words - the NSLinguisticTagger would not lemmatize a word properly without context, which we discovered a tad later than we’d like to admit. We solved it by passing it the entire string and just returning an array and using indexes to identify each corresponding lemmatized word.

We have tried to include as many function/padding words as we can think of, but we understand that it is unlikely to be able to list every possible padding word, so there may be certain instances of padding words which are unable to process as they are not removed, nor are they entries in our database.

Hard to call the method which puts borders around images as the cells are created all over the place. The problem was finding the correct place to call the method as if it was called multiple times the image would shrink.

Just padding words by themselves used to take the user to the next screen.

Some UI elements didn’t work as expected so spent a lot of time trying to fix and get them to work

There was a few issues with merging the main storyboard if more than one person was working on the UI so we often had to discard changes and implement them again

There was issues with async processes and loading wheel so currently dont have a loading wheel when waiting for synonyms from internet

While testing, we found that because we were dropping all padding/function words, if the user just entered these, they would be processed as valid words and would take the user to the next screen but it would be blank, if the user did not enter any words matching entries in our database.

It was hard to test as we couldn't possibly anticipate all possible combinations of words.

We have found that inputs of special characters, such as '$' can crash the app. Due to time limitations, we have not yet solved this issue, but it will be sorted before the final release.

Extra (padding) spaces between words (even if the sentence was valid) could cause the app to crash. We solved this by simply filtering out all the whitespaces before parsing the input string to be processed.

Oddly capitalised words or words which had certain letters capitalised used to crash the app, but this was fixed by forcing all words to lowercase before processing.

Image to text sometimes adds an extra space between some words.