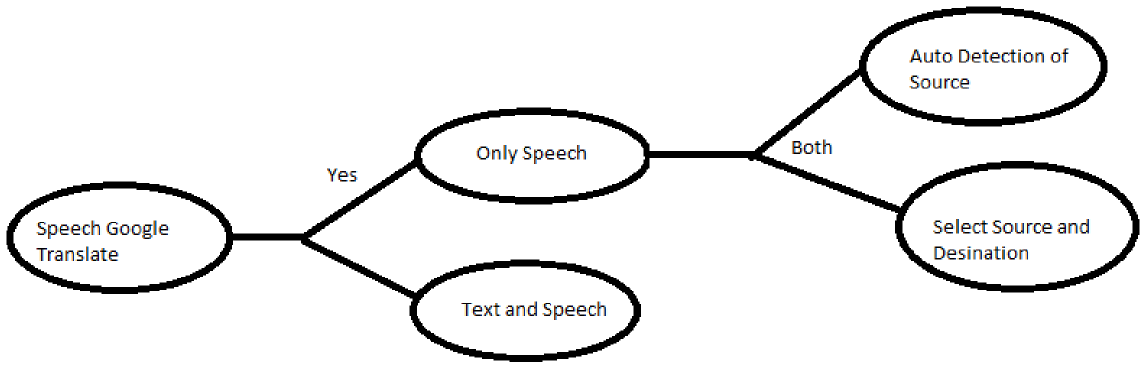
Audio UI Report

Dalton Rothenberger

Overview

My project is a voice controlled Google Translate program. The program allows the user to select between two options for translating: selecting a source and destination language or selecting a destination language and using auto-detection for the source language. I limited it two these two options as they seemed to be the two major ways that google translate is used. The program is capable of translating to over 100 different languages. I had to do some remapping of the various keys for the language map because some of the options would not be selectable through voice. For example “chinese (traditional)” had to be changed to “chinese traditional” because the parenthesis were not something a user would naturally speech. This was something that a normal text-based or graphical interface would not have to worry about so it was interesting to have to find a work around to this for speech.



Text-to-Speech Interface

The program also tries to use text-to-speech as much as it can. The goal of using text-to-speech was to make it so that the program could be entirely controlled and used without needing to read any of the text interface. Even with this in mind, looking at the text interface still provides useful information while using the program. The whole menu, except listing the languages, is text-to-speech. The reason listing the languages was not made text-to-speech was because having it list out roughly 100 languages took a long time. One problem that arose with when using text-to-speech for the translated text, is that the text-to-speech library used does not handle non-roman script languages. So translating from English to Spanish would result in a text-to-speech interpretation of the Spanish translation but something like English to Japanese would not because the text-to-speech library cannot handle Kanji/Katakana/Hiragana.

Menu Design

When designing the menu system I had to think about the various affordances a user would expect and also a way to signify to the user what would be proper input to navigate the menus. I decided on taking two possible input for each menu option. One options is the number listed next to the item in the menu and the other is the full text of the menu option. I signify that these two options exist in the first prompt the program outputs so that they know how to interact with the program and to reduce frustration. The reason I decided to include the full text option was because that would be the perceived affordance of how the menu would work. I included the numbers as input as well because this would make it easier to select options because the speech recognition was not reliable when it came to selecting using the full text method. The whole menu system properly handles invalid input and unrecognized input. The menu will send out a prompt about an error occurring and then prompt the user for another input. Also, at nearly any point in the menu the user can exit the program through voice commands. This was implemented to avoid the hassle of having to return to a main menu to exit the program.

Improvements

Since the program was designed with the intention of being fully useable only through voice it does not take any text input. Adding text based input would reduce frustration when the speech recognition errors multiple times in a row. Another improvement would be to find another text-to-speech library that can handle more languages than the one used. Mainly a library that can handle non-roman script languages. Not being able to hear the translated output can diminishes the experience that the program is trying to provide to the user.