Contents

1 Summary of progress that has been made

The last week and a half or so I have focused exclusively on implementation. I have attempted to take notes here and there to make sure I don't forget why I did things a certain way (and write verbose commit messages for the same purpose), but the paper sections with more detail will be forthcoming later.

Moreover, a good bit of this implementation required sitting down and *thinking* about how to best do things. There are lots of small issues that pop up – issues difficult to anticipate beforehand – that make supporting certain features less trivial than first meets the eye.

1.1 Personal project

Tangential to the project at hand is my own personal keyboard project. I spent a lot of time in the last week refactoring code from this project, and studying how things might be carried over. In no particular order:

- Structuring remapping code into layers by means of (conditional) function calls.
- Handling modifier key behavior *independent from remapping behavior*. I took pains in this other project to create functions that will allow for normal keyboard shortcuts when using Greek and Hebrew keyboards.

- Creating so-called dynamic regex hotstrings to support various forms of expansion. This will probably be less important for less frequently used foreign language layers, but the ability to add personal shortforms as desired (even when typing in a different alphabet) is something I wanted to support. One might shorten καί to just κ, γάρ to just γ, and so forth.
- (Technical): figuring out how to support different behavior based on the last key pressed. This will be discussed below.

Additionally, this other project serves as an example of what is possible with the software being used. It is significantly more complex at a high level than "just" typing in a certain language.

I won't attempt to justify in great detail why I spent time on this project. It seemed like the right thing to do at the time, and has definitely helped me see where I want to go with this project, particularly how to move forward once the codebase gets bigger and more involved.

1.2 Diaeresis and vowel quantity

Adding support for the last few main diacritics took me far longer than expected. The reason for this is that in adding (full) support for macrons and breves, I had to create the framework for using decomposed sequences of Unicode characters. Up until this point, I had been using only precomposed characters, which are not terribly difficult to work with once you have basic status variables set up to keep track of which diacritics to send.

In particular, I wanted to come up with a method of handling decomposed characters that *mirrored the structure of handling precomposed characters*. Part of this is for ease of design on my end, but part of it is also because this will, hypothetically, make it significantly easier for other people to make use of this project in the future.

At a high level, the combining characters are stored in alias variables (e.g., "acute", "macron", "smooth"), and then concatenated with the base letter before being sent. The order of concatenation is somewhat language specific. In Greek, for example, breathings come before accents, and priority here is left to right rather than the typical bottom to top. The "normal" way Unicode handles multiple combining diacritics that occupy the same general space on the letter (e.g., the top) is to display the combining character immediately following the base character "closer" (e.g., right above the base character), and then the next character "further out" (e.g., above the first combing character, "one step removed" from the base character).

The upshot of all this is that you have to think a bit about what order you send decomposed Unicode if you want it to display correctly. I took this into account, and perhaps more importantly, I coded it in such a way that *the correct sequences are sent regardless of entry order*. To take an example, if you type an alpha with smooth breathing and an acute accent, but type the accent before the breathing, you would ordinarily get an output character with the accent displayed to the left of the breathing (which is incorrect). However, with the implementation I am using, the correct sequence gets sent even in this case, since it is the combination of diacritics that is being tracked explicitly rather than their entry order.

1.3 Backspacing and sequences

Backspacing becomes much more difficult when you start adding sequences of characters that are sent and deleted, rather than individual characters (as in precomposed Unicode). Further complicating the situation is if you start mixing the two: is it possible to design a system of tracking variables and function calls to always backspace the last semantic character if vowels with macrons are sent precomposed, but vowels with macrons and acute accents, e.g., are sent decomposed?

It turns out it is, although it took me a while to get there. By keeping track of the number of keys sent (precomposed always sending just one) and checking if the last key sent was a diacritic sequence (see below on how I did that), you can conditionally send higher numbers of backspaces when necessary, but only one (as expected) the rest of the time.

1.4 Checking the last key sent

Autohotkey has a variety of useful built-in variables that are tracked for you. One of them is called $A_{PriorHotkey}$, which returns the last hotkey activated by your script.

The problem with $A_{PriorHotkey}$ is twofold:

1. If you don't define hotkeys for all keys on the keyboard, it can give you misleading results. (I.e., if all "keys" are not "hotkeys"). This is not always a problem, but can be really nasty to debug later if it turns out that you do in fact need to register some keys that you think you are but are not because $A_{PriorHotkey}$ does not recognize them.

2. If the same hotkey can do different things depending upon other variables, $A_{PriorHotkey}$ may not be specific enough. This can sometimes be avoided by not using blinded hotkeys (those triggered regardless of modifier state),

but then we would run into other problems if we ever wanted to remap modifier behavior in groups. And if you have custom flag/sentinel variables unrelated to modifiers, then $A_{PriorHotkey}$ will not work regardless.

I got around the second issue by keeping track of the time that the last diacritic sequence began to get sent (i.e., the time the beginning of the sending function started to get executed), and then comparing this to the time the last hotkey in general got sent. We only care if the last thing that got sent was a hotkey *and* if said hotkey triggered a diacritic sequence. Here's the general concept in the form of the backspacing function:

```
deleteLastFullCharacter()
{
  timeOfLastHotkey := A_TickCount - A_TimeSincePriorHotkey
  if((timeOfLastHotKey - lastDiacriticKey) < 50)
  {
    SendInput {Backspace %numKeysToBackspace%}
  }
    else
  {
    SendInput {Backspace}
  }
  return
}
```

It is not necessary to implement final sigma support in this way (allowing for flag variables and conditional execution based on the last key pressed) since $A_{PriorHotkey}$ will work fine for the s key, which is where sigma is located.

1.5 Final sigma support, punctuation

Supporting final sigma required finishing off implementation of Greek punctuation characters. I chose to value meanings/semantics over form: ? corresponds to the Greek ;, and ; corresponds to the Greek middle dot.

The basic idea behind the final sigma support is that if there is a word boundary immediately following a sigma, it automatically gets transformed into its final form. There are a couple exceptions. For example, if there is an apostrophe (indicating elision), the sigma shouldn't be final; if there is a dash (indicating, in general, that a writer wishes to indicate a prefix like προσ-), the sigma shouldn't be final; etc.

2 Summary of short term goals

2.1 Survey results

I got focused on the code this week, but know that I ought to get a preliminary email with the basic results sent out. This is currently priority #1.

2.2 Adding a fully decomposed option for Greek

The framework is now in place for me to implement the full complement of Greek diacritics in decomposed Unicode. Doing this will allow for A/B testing during the beta-testing phase, should we get there.

2.3 Refactoring to allow for a full English/Greek dual layout

This will be modeled on my keyboard project, which uses a similar concept to juggle ~7 different layers.

2.4 Writing things up

While the process is still fresh in my mind, I should probably write at least a draft form of the relevant paper sections. I also still need to go back and correct many of the things discussed earlier: more specificity, a longer discussion of "why" with respect to the keymap, etc.

2.5 Hebrew

This is still a priority. However, Greek takes first precedence.