The Art Of Virtual Keyboarding

By

Jose Michael Joseph, Nathan Ong
CS2610
Prof. Jingtao Wang

What it’s About:

Virtual Keyboarding allows the user to transcend the limitations imposed by a physical device and gives them features that would be not feasible to implement physically. Virtual Keyboarding thus has the potential to save a lot of time as it gives us an alternate mode of input that could be considered more efficient than the current mode.

Our Considerations:

Our primary concern was ease of use. We designed the system to be very user friendly and as intuitive as possible. To make it intuitive we used substantial visual feedback, like lines that trace the path and keyboard buttons that light up when you go over them. This feedback enables the user to create a firm conceptual model in their mind which enables the user to further use the application efficiently.

To get into the specifics, out of all the frameworks we considered we thought it was most appropriate to build in Java. This is because Java gave us a larger reach into the audience, since it can run on most devices irrespective of the Operating System, and helped us to develop the application in a much lesser amount of time than it would have taken us on any other system. For this we have to thank the extensive collection of Java library files that are present on the web that enables the user to create efficient applications.

The other consideration that we had while using Java Swing was whether to use Buttons or rectangles that we could draw ourselves. After much deliberation we decided that it would be better to use custom made rectangles. The reason for this is that rectangles offered more affordance while being used as keyboard button since they could change color and texture as visual feedback. Also the while one clicked the button and dragged it, it would only look like one button was pressed down whereas for a rectangle, all the rectangles on the traced path would light up sequentially. Therefore, since rectangles provided us much better feedback we decided to use custom drawn rectangles.

We also thought a lot on how we would choose the strings that we would give as input to the dictionary. We finally decided that to streamline the process we would check the angles at which the lines drawn by the user were being formed. We would use this data to find out “corner points” in the list of strings that were given us to. These corner points would then be sent with the rest of the string to the dictionary. The only difference is that the corner points had additional markers to identify them as corner points. This greatly reduced our computational effort and helped us produce more efficient results.

Pros:

- Intuitive user interface
- Extensive visual feedback
- High aesthetic value
- Sleek minimalistic design

Cons:

- Computational cost increases with increase in string length
- Keyboard keys don’t provide “depth perception”

Usage:

The user would ideally open up our application and use the mouse to click on a desired alphabet. The user would then drag the cursor around the various alphabets which he or she wants to use to form the complete word. The system then takes this information and tries to predict the word that the user was trying to create. If the word predicted by the system is different from the one intended by the user then the system provides additional options of words which it thinks could also be possible matches. The user is then given the ability to pick one of these words from the displayed options and use it as the final word.

This system thus works most seamlessly with the mouse as an input device. But from our experience we have also found that it works reasonably well in laptops that support touchscreen since the touchscreen maps the input data given to it into various forms of mouse actions.

Advantage of such a feature:

The primary advantage of such a feature is that it dramatically reduces the time taken by the user to write a word as instead of typing each word individually when the input medium is mouse click as the user now has the ability to just swipe through the various letters of the word and have the system create the word for the user.

Disadvantage of such a feature:

One of the disadvantages of such a feature is that if the word the user wants to create have letters that reside on opposite lengths of the keyboard then the swipe the user has to make would include a lot of letters that were traversed on the path between the two intended letters. Therefore at times the letters that are not part of the final word consume more memory than those that are actually part of the word. Although not strictly a disadvantage, more number of characters would take more processing power.

Another disadvantage of our system is that while swiping to create more than one word it only shows the replacement words for the last word of the sentence. It also shows only up to four replacement words for each incorrect word entered. We based this decision on trial and error and decided to stick with this minimalistic approach in order to ensure that the screen space was not cluttered and that the user always had access to only the essentials.

Thirdly, we found that the size of the keys played a primary role in the design layout. If we kept it too small the user would inadvertently overshoot the keys while swiping multiple times and this would result in incorrect results. What we could do to solve this would be to increase the size of the keys thus making it much easier for the user to zone into the correct key. But another problem this would cause is that the keys would then be relatively farther apart and thus it would take the user much more time to make the same sentences as before.This is in absolute coherence with Fitts' Law which states that if we increase the distance and the width, the index of difficulty will remain proportional.

Finally the biggest disadvantage is that there is a huge tendency for the user to overshoot which would result in the algorithm producing incorrect results or not showing any result at all.

Inspiration:

The primary inspiration of our system came from the android third party application called "Swype[1]" which works on the same principles and uses slightly different algorithms to produce similar results. The primary difference between our system and Swype is that Swype stores the times that it gets the intended word wrong and also stores what the intended word actually was. It then uses this data to more efficiently predict the words in the future.

Citations:

[1]:http://www.swype.com/

[2]Dictionary used: http://www.kilgarriff.co.uk/bnc-readme.html#raw