CS2610 Homework Assignment 1

The Art of Virtual Keyboarding

Release date: 9/9/2015   Due Date: by 9:00AM on 9/30/2015

This homework assignment should be completed in two-member groups (three-member group is okay in special situations). Only one copy of report is needed for each group. Please contact the TA if you couldn't find a partner for this assignment. Late submissions will loss 20% of the total grade each day. We won't accept submissions later than 3 days.

The students may freely choose their desired programming language for this assignment.

Background

Text entry is one of the most frequent Human Computer Interaction tasks. It presents unique challenges in every generation of computing platforms, be it batch process based main frames, personal computers, cell phones equipped with 12-key keypads, smart phones with multi-touch screens or "natural user interfaces" embedded in smart environments. Surprisingly enough, although with a history of more than 140 years, the QWERTY keyboard is still the "most popular" input device on both PCs and touch screen smart phones nowadays. The purpose of this homework assignment is to provide some "warm-up" exercises on UI programming, problem analysis, creative thinking and literature survey.

The QWERTY keyboard layout was originally designed by Christopher L. Sholes, Carlos Glidden, and Samuel W. Soule in 1868. The original design goal of QWERTY layout was not for speed, but for minimizing mechanical jamming instead [1, 2]. As a result many adjacent letter pairs (digraphs) appear on the opposite sides of the keyboard. The rapid speed of typewriting is largely a result of the method of touch-typing without looking at the keyboard, discovered independently by L.V. Longley and F. E. McFurrin in the 1880s. Since the 1920s, touchtyping has been accepted without controversy as the superior typing method, and its wide adoption rapidly increased typing speed [2].

0. **The Baseline on-screen QWERTY keyboard**. We provide a baseline on-screen QWERTY keyboard in Java (http://mips.lrdc.pitt.edu/cs2610-fall15/File:Source_Code.zip) for the following tasks. The virtual keyboard has one sample sentence display region, current text display region and one virtual keyboard region (Figure 1). For simplicity concerns, we only supported upper case characters A to Z and the space bar for this assignment. There is no need to support numeric keys, punctuations, tab, caps lock, control, alt, shift key etc. Please treat the source code we provided as a *minimal* skeleton to get you started, rather than an "exemplar" of good design. **Please feel free to ignore the baseline source code and write your own code from scratch, or use existing virtual keyboard implementations on the internet for this assignment if you credit the original authors and origins properly**. You may choose whatever programming language you are comfortable with for this

project. Remember: most of the virtual keyboard implementations on the internet are over-kill for this specific homework assignment. You may end-up spending more time in customizing the adopted code for the follow-up questions.

1. **Improved affordance and feedback (10 points)**. Please add additional visual/audio cues to support the "***affordance***" of pressing/clicking as argued by Don Norman in his seminal book "The Design of Everyday Things". After pressing a virtual key (for your application, it means moving the mouse cursor above a button and click the left button of the mouse), provide some form of "***feedback***" (in form of color, sound, animation or a combination of them) to indicate state changes.

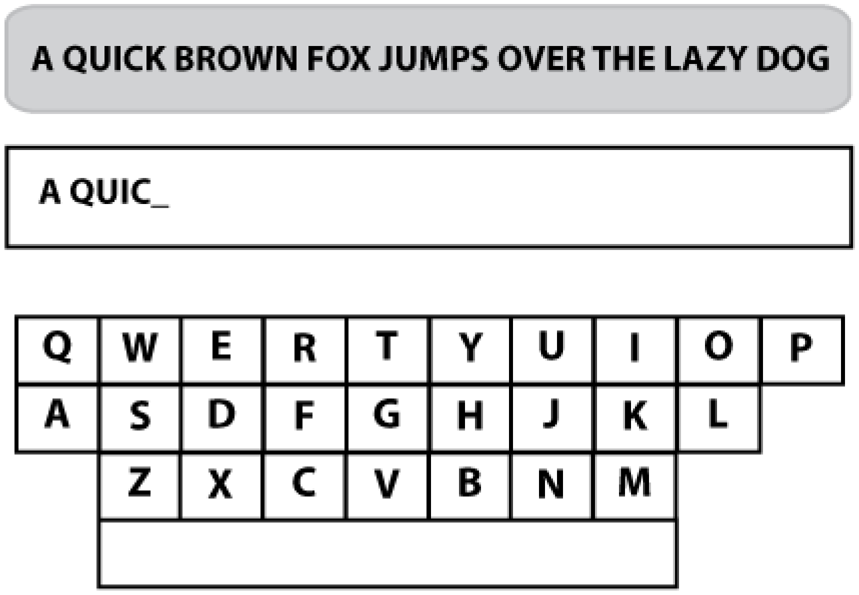

Figure 1. Illustration of a virtual keyboard, with a sample sentence region and a current text display region

2. **Shorthand Writing on Virtual Keyboard** (**20 Points**). Tapping characters one by one on a virtual keyboard could become tedious very quickly. With some extra linguistic information of the intended language, we can reduce the tapping efforts by allowing a user to slide his/her finger/stylus/mouse on each intended character in a word one by one without moving up his/her finger/stylus/mouse. As long as the user is trying to enter a word in the dictionary, the "intelligent user interface" will help the user figure out what is the most likely word (Figure 2). This approach leverages the redundant information in languages (e.g. not all character combinations are valid words) to disambiguate users' input.

   In the baseline source code provided, we already implemented the ability to visualize mouse traces when you press and hold the left mouse button and trace on the keyboard. Again, please treat the implementation you get as a starting point rather than polished, perfect implementations. It's still up to you to implement the algorithm that translates mouse traces into words.

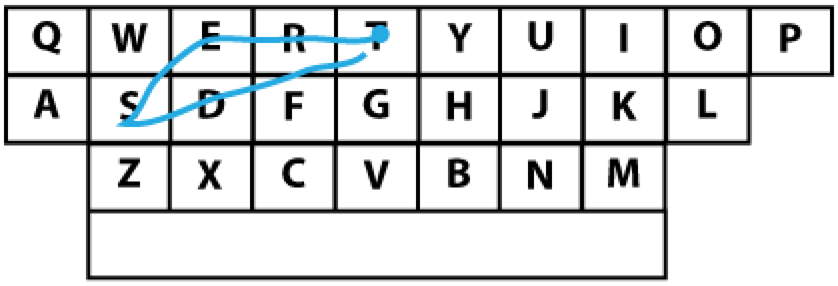

Figure 2. Using a mouse to trace word "TEST" on the virtual keyboard. The blue round dot indicates the starting point of the gesture

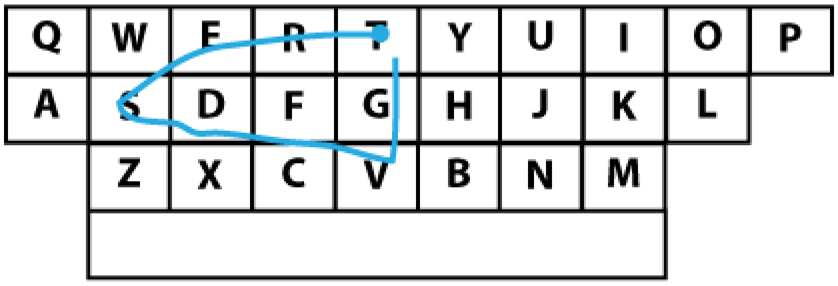

Figure 3. The above trace shouldn't generate word "TEST". What's the difference between figure 2 and this figure?

The original idea of shorthand writing on a virtual keyboard was invented by Zhai and Kristensson [3, 4, 9]. With the popularization of smart phones equipped with large, capacitive sensing [5] touch screens such as iPhone, Android, Palm Pre, Windows Phone 7 etc., many commercial products such as the Gesture Typing mode of Google Keyboard, Swype and SlideIT ([6, 7, 8]) have emerged recently based on this idea. Although creating an easy to use application of this kind is time consuming, building one proof-of-concept prototype like the one in this homework assignment shouldn't take a huge amount of efforts. The system in [3, 4, 9] used gesture recognition technology to achieve fault tolerance, location and scale independence. For a prototype we are building now, if we remove the requirement of location and scale independence (i.e. assuming the user's gesture trace has to cross each and every letter in a word, the trace has to start from the first character of a word, end at the last character of a word), it can be implemented quite easily via some customized string/dictionary searching algorithms.

Please add features to the virtual keyboard you implemented in question 1. In addition to tapping each character as supported in question 1, the finished application should allow users to trace words on the keyboard like the one shown in figure 2. Here are a few things that need your extra attention during implementation – a. how do you handle duplicate characters in words like (**bee** vs **be**)? b. how do you resolve ambiguity if one input trace can map to multiple valid words? c. How do you make sure the trace shown in figure 3 won't output word "TEST"? Please note that there is no single correct answer for these questions, explain your design decisions in the homework submission and defend them. An English word list with related word frequency is available as a pure text file for you to use [11], you may want to sort the words alphabetically first in your implementation.

Bonus points (up to 5 points): Again, as suggested by Don Norman, designing accurate feedback is crucial for building an intuitive "conceptual model" of our system. Design (and implement if you have time) some feedback mechanisms to visualize in real time why the computer gives the current output given the current input trace. Do you still remember the concept of "Gulf of Evaluation"?

3. **Performance measurement and visualization** (**20 Points**). Benchmarking the performance of text input (or speech input, or machine translation) may not be as easy as one initially thought. For example, direct string matching cannot be used to detect the error rate (i.e. the number differences between the intended input and the actual input). If the intended input is "YOU ARE NOT A JEDI YET" but the actual input is "OU ARE NOT A JEDI YET", or "YYOU ARE NOT A JEDI YET", comparing the input and output via direct string comparisons will lead to a result of zero matches due to misalignments. You should use the Levenshtein distance [10] (a.k.a. the edit distance, the minimal number of insertion, deletion and replacement operations needed to transform one string to the other) when analyzing differences between the input and the output. Please design an interactive visualization technique to visualize the differences between the current input buffer and the expected sentence. You can show in real time which character is matched with which character in the test string (and which ones are considered deletions or insertions).

What to submit

1. Upload a demo video (in the form of screen recordings) of your finished program to the homework assignment wiki page. Please use voice overs and/or on-screen captions to explain your application implemented. You can use either Cam Studio [12] (Windows, free) or iShowU HD [13] (Mac, free evaluation) to record the video.

2. Email a brief report (no more than 3 pages in total) that describes your solutions to both the instructor and the TA. Make it clear the design decisions you made, and the reasons (pros and cons) of those decisions. Use Figures when necessary; make sure your descriptions are accurate and succinct.

3. Email a copy of the compressed source code of your project to the TA (please also include a README file to describe how to build your application. Please also include **a web link** to a compressed copy of pre-built, ready to run application in the readme file). Please make sure the total size of your source code package is no more than 1.5 MB. Please contact the instructor and the TA if you couldn't make the source of your project smaller than the size limit.

References

[1] Cooper, W. E. Cognitive aspects of skilled typewriting. New York: Springer-Verlag. 1983

[2] Yamada, H. A historical study of typewriters and typing methods: From the position of planning Japanese parallels. Journal of Information Processing, 2, 1980, 175–202.

[3] Zhai, S., Kristensson, P-O., Shorthand Writing on Stylus Keyboard, in Proc of CHI 2003.

[4] Kristensson, P-O., Zhai, S., SHARK2: A Large Vocabulary Shorthand Writing System for Pen-based Computers, in Proc. of UIST 2004.

[5] Capacitive Sensing, http://en.wikipedia.org/wiki/Capacitive_sensing

[6] Gesture Typing on Google keyboard https://support.google.com/nexus/answer/2811346?hl=en

[7] Swype http://www.swype.com/

[8] SlideIT https://play.google.com/store/apps/details?id=com.dasur.slideit.vt.lite&hl=en

[9] A demo video of SHARK http://www.youtube.com/watch?v=-ZCGKqjTqvE

[10] Levenshtein distance http://en.wikipedia.org/wiki/Levenshtein_distance

[11] An English word list with frequency http://mips.lrdc.pitt.edu/courses/cs2610/restricted/word_freq.txt

[12] Cam Studio http://camstudio.org/

[13] iShowU HD http://store.shinywhitebox.com/ishowuhd/main.html