Lehrstuhl für Medieninformatik
Institut für Information und Medien, Sprache und Kultur
Universität Regensburg

More on QT

Interaction Techniques and Technologies (ITT), SS 2016

Session 7 (19.05.2015), Raphael Wimmer

Overview

These are slides/notes for the lecture, automatically generated from the slide set. Please extend this outline with your own notes.

Goals for this Week

Know

- text input methods
- typical text input speeds
- keyboard hardware

Learn

- measuring typing speed
- PyQT's signals and slots

Practice

- Python
- text input
- study design and conduction

Today

- **14:15 - 14:35** Review of last session, overview of today's session
- **14:35 - 15:20** More on PyQt: signals and slots, widgets, QML, decorators
- **15:20 - 15:45** Discussion of current assignment

Where are We?

- **Conducting and Logging Experiments** (+ intro to Python / PyQT)
- **Documenting and Visualizing Experiments** (+ intro to pylab, matplotlib)
- **Pointing** (pointing devices, Fitts' Law, Steering Law, CD gain, …)
- **Text Entry** (speed, models, keyboard layouts, input techniques)
- **Models of Interaction** (KLM, GOMS)

Same quiz again: Which of the following statements is true?

- Fitts' Law says that the time to select a target increases linearly with distance
- Eye movements can be modeled using Fitts' Law
- A high CD gain is important for pointing on large displays
- Touch screens are rate-control, direct, absolute pointing devices
- A *t test* indicates whether two values are statistically different

Review previous session

Overview

- Speech Input
- Handwriting
- Keyboards

Handwriting

- **OCR**
- **natural handwriting** (hard)
- **simplified alphabets**

. . .

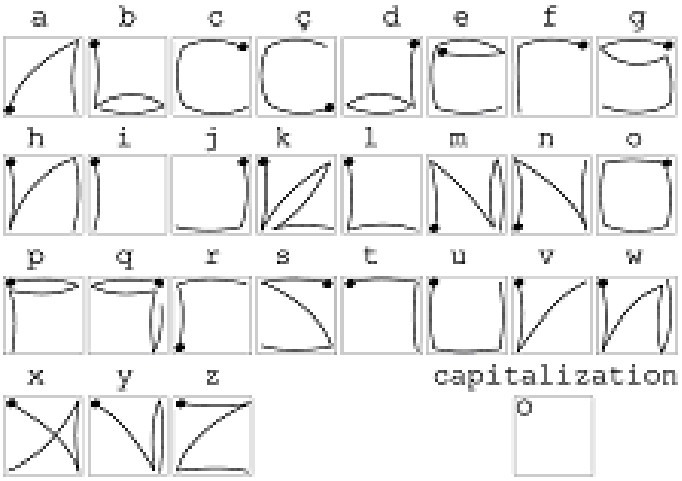


Figure 1: EdgeWrite

. . .

Keyboard implementations

see blackboard

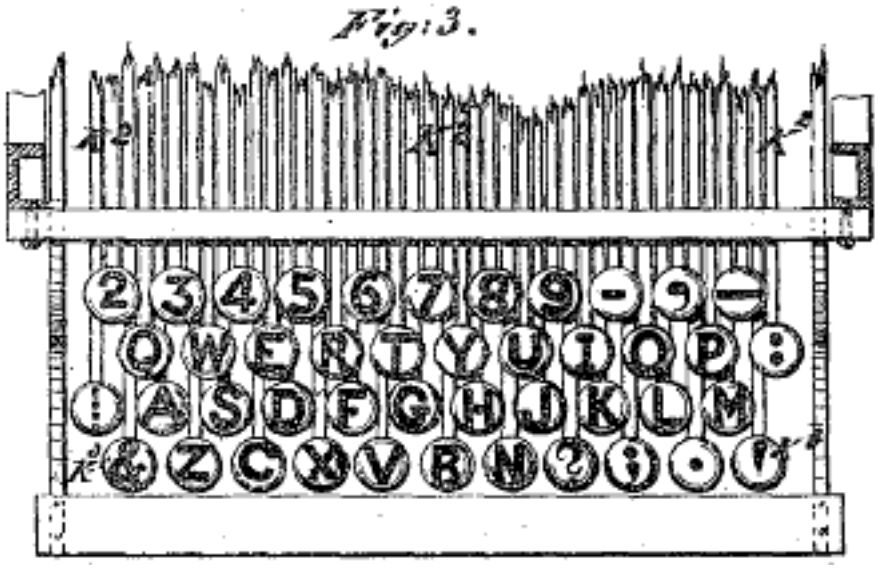
Ghosting / N-key rollover

- simple matrix scanning of contact mats leads to ignored keypresses

Figure 2: Graffiti

- Info in Geekhack.com wiki[1]
- Ghosting Demo by Microsoft Research[2]

QWERTY



o

- ~1870
- staggered rows required for key levers
- de-facto standard

Dvorak

- ~ 1936
- optimize key locations to minimize finger movement
- shown to be faster than QWERTY (disputed![3])

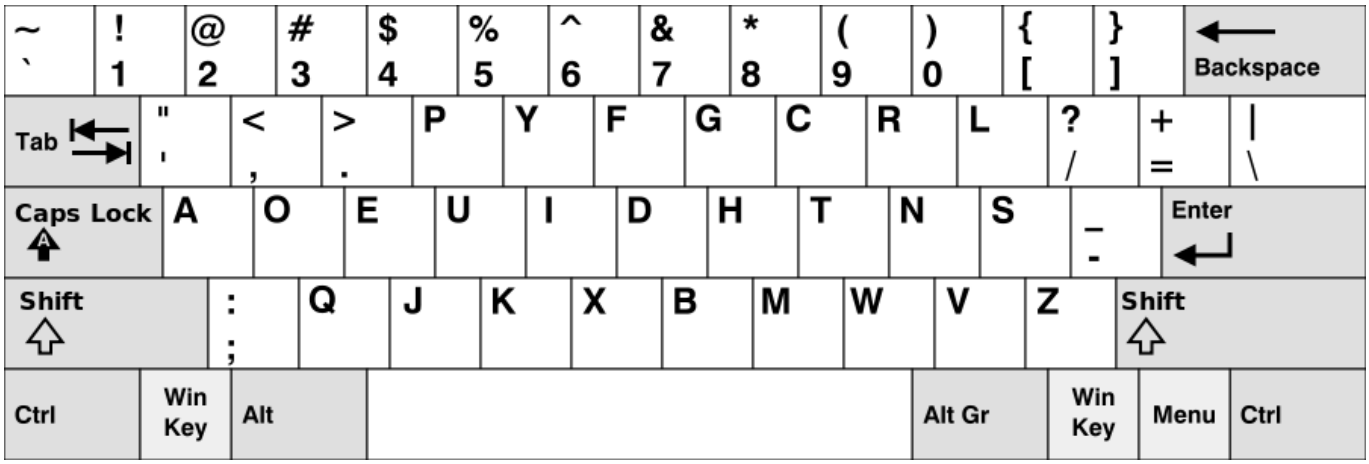


Figure 3: Wikimedia Commons, PD

Figure 4: neo-layout.org

Neo

- since 2004
- http://www.neo-layout.org/
- optimized for German language
- 6 layers, with many Unicode symbols, foreign characters

Stenotype

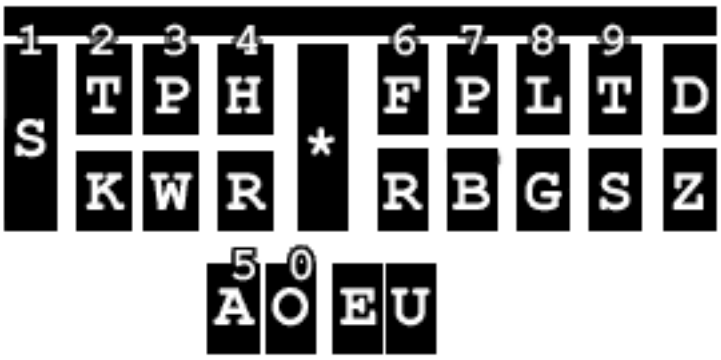


Figure 5: Wikimedia Commons, PD

[1] http://geekhack.org/showwiki.php?title=NKey+Rollover+-+Overview+Testing+Methodology+and+Results
[2] https://www.microsoft.com/appliedsciences/content/projects/KeyboardGhostingDemo.aspx
[3] http://www.utdallas.edu/~liebowit/keys1.html

- Open Steno Project[4]

Typical Text Entry Speeds

Source: WP:Words per minute[5]

- 1 word per minute (WPM) = 5 characters per minute (CPM)
- Handwriting: 30 wpm
- Stenography: 350 wpm
- Speaking: avg. 150 wpm, pro: 350-500 wpm, record: 637 wpm
- Reading: 250-300 wpm (typical adult)
- Morse: good: 20 wpm, pro: 60 wpm, record: 76 wpm
- One-key-keyboard (MacKenzie, 2010)[6]: 5 wpm

QWERTY

- Hunt-and-peck: 30-40 wpm
- Professional Typist: 50-80 wpm
- World Record: 216 wpm (1946, using the Dvorak layout)

Stenotype

- Beginner: 100 wpm
- Professional Stenotypist: 200 wpm
- World Record: 360 wpm

Chording Keyboards

http://research.microsoft.com/en-us/um/people/bibuxton/buxtoncollection/detail.aspx?id=7

See also: http://www.loper-os.org/?p=861

Predictive Techniques

- Do not require the user to learn something new - meet them where they are.
- **Autocomplete**:
  - provide suggestions for word completions once one or more letters are typed
  - Autocomplete in Python: github.com/rodricios/autocomplete[7]
  - See also QCompleter[8]
- **Autocorrect**:
  - automatically change an incorrectly spelled word to the probably intended word
  - also used for replacing abbreviations
  - Python example: github.com/foobarmus/autocorrect[9]

[4] http://openstenoproject.org/
[5] https://en.wikipedia.org/wiki%20/Words_per_minute
[6] http://www.yorku.ca/mack/TOCHI2010.html
[7] https://github.com/rodricios/autocomplete
[8] http://doc.qt.io/qt-5/qcompleter.html
[9] https://github.com/foobarmus/autocorrect/

Figure 6: Buxton, 2010

Course Assignment

Let's add chord input to a QWERTY keyboard.

Questions:

- Implementation?
- Choice of chords?
- Evaluation?

More on PyQt

Building UIs with Qt / PyQt

Four approaches

- programmatically define UIs
- graphically assemble UIs in QT Designer
- declaratively build UIs with QML and JavaScript
- use a QWebEngineView or QML WebView to render an HTML UI

(we focus on the first two here)

Example: Build a simple UI

- design a form in Qt Designer
- load it using PyQt:

. . .

```
import sys
from PyQt5 import uic, Qt

app = Qt.QApplication(sys.argv)
win = uic.loadUi("mainwindow.ui")
type(win)
# PyQt5.QtWidgets.QMainWindow
```

Instrumenting applications in Python

Signals and slots

- Qt system for event-driven programming
- QObjects may have signals (outgoing events) and slots (handlers for incoming events)
- Connect using preprocessor directives (C++), XML (.ui files) or `connect()` method of signals (PyQt)

. . .

Example:

```
lcd = QLCDNumber()
sld = QSlider(Qt.Horizontal)
sld.valueChanged.connect(lcd.display)
```

Decorators

- Decorators allow for wrapping functions or classes, modifying their behavior
- Function decorators in Python: function or object constructor which takes the function to be decorated as a parameter and returns a new decorated function
- Often implemented as closures
- Use cases: limiting access to a function, executing other code before or after execution of the decorated function, …
- see also https://github.com/lord63/awesome-python-decorator
- Syntax:

```
...

def my_func():
    pass

my_func = my_decorator(my_func)

...

@my_decorator
def my_func():
    pass
```

Decorator Example

```
def my_func(arg1):
    print(str(arg1))

def my_decorator(func):
  def wrapped_func(*args, **kwargs):
    print("before")
    func(*args, **kwargs)
    print("after")
  return wrapped_func

func2 = my_decorator(my_func)
```

Decorator Example for click handler of a button object

Decorators can also accept parameters:

```
def log_button(message):
    def func_decorator(func):
        def new_func(self):
            print(message + " " + func.__qualname__)
            print(self.text())
            func(self)
        return new_func
    return func_decorator
```

...

```
@log_button("Button clicked:")
def on_click(self):
    # do something
```

Current Assignment

Current state?

Outlook

Next Session

- presentations of chord input techniques
- introduction: UIs and interaction techniques

Further Reading

- Tutorial about event filters: 1[10], 2[11]
- PyQt5 documentation for Signals and Slots[12]
- Qt5 documentation for event filters[13]
- Zetcode tutorial about Signals and Slots[14]
- Github repository with PyQt/QML examples[15]
- Tutorial about custom input methods for Qt (not PyQt!): 1[16], 2[17]

ENDE

[10] http://ynonperek.com/qt-event-filters.html
[11] http://ynonperek.com/course/qt/event-filter2.html
[12] http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html
[13] http://doc.qt.io/qt-5/eventsandfilters.html
[14] http://zetcode.com/gui/pyqt5/eventssignals/
[15] https://github.com/pkobrien
[16] http://www.kdab.com/qt-input-method-depth/
[17] https://www.kdab.com/qt-input-method-virtual-keyboard/