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

Interaction Technology and Techniques
Assignment 9: KLM

Summer semester 2016

Submission due: Wednesday, 8. June 2016, 23:55

Hand in in groups of max. two.

Your task is to implement a simple UI, understand the basics of the *keystroke-level model* (KLM), write a tool for estimating task completion times via KLMs, and predict/verify the task completion times for three small tasks.

9.1: Read up on KLM

Read the papers linked in GRIPS to gain a better understanding of KLM and GOMS:

- Gray, John, Atwood (1992): "The precis of Project Ernestine or an overview of a validation of GOMS"
- Kieras (2001): "Using the Keystroke-Level Model to Estimate Execution Times"

Answer the following questions:

a) What does the term "critical path" describe with regard to GOMS?
b) Why might the original KLM not perfectly model input performance on a laptop? How could it be adapted?

Hand in the following file:

goms_klm_questions.txt: a text file with answers to these questions.

Points

- **1** Good answer to the first question.
- **1** Good answer to the second question.

9.2: Implement a simple calculator and determine KLM operators

Implement a simple graphical calculator using PyQt. The calculator should support the following 18 inputs:

- digits from 0 to 9 and the decimal point
- operators: +, -, *, /
- =, clear, delete last digit on screen

The UI should be implemented using a .ui-File. Use signals and slots for communication. Pressing a key on the keyboard should do the same as clicking the corresponding button.

(If you need some inspiration on how to handle this, have a look at rpCalc.zip in GRIPS)

Find out typical values for the standard KLM operators (excluding the M operator) specifically for your calculator. To this end, do the following:

- instrument the calculator's code so that all relevant input events (such as keypresses, clicks on buttons) and their timestamps are logged to `stdout` (hint: use decorators or signals).

- perform a number of simple tasks with the calculator, such as repeatedly clicking the same button, switching between mouse and keyboard, etc.
- analyze excerpts of the log data in an iPython notebook in order to determine typical values for the KLM operators. For example, a good value for the K operator might be calculated by taking the mean value of times between rapidly following keypresses.
- document the values for the KLM operators in the notebook and also describe how you generated them, i.e. which tasks were conducted.

Hand in the following files:

calculator.zip: the source code of the instrumented calculator **calculator_klm.ipynb**: an iPython notebook containing the aforementioned analysis

Points

- **3** The calculator has been implemented successfully and elegantly.
- **2** The notebook contains a good description of the process for calculating the KLM values.
- **1** The calculator-specific KLM values are plausible.

9.3: Implement a KLM calculator

Write a small Python application `klm.py` that reads in a file with KLM operators (filename provided as argument) and outputs a prediction for the task completion time.

The KLM operator file may contain comments - started by **#** - that should be ignored. Case should be ignored, i.e. 'K' and 'k' both stand for a *keystroke* operator. A single line may contain one or multiple operators. Each operator may be prefixed by a count. The calculator should print out two estimates: one using the operator values defined by Card, Moran, Newell (1980) and Kieras (2011), and one using the operators calculated in 9.2. An example file might look like this:

```
# KLM operators for logging in to a system
m8k # remember username, enter eight keystrokes for username
k  # tab, switches to password field
m13k # remember and enter password (13 characters),
hpbb # switch to mouse, move over "sign in" button, click.
```

Hand in the following file:

klm.py: a Python script that implements the above requirements and can be used to predict task completion times for keyboard/mouse input tasks.

Points

- **1** The python script has been submitted, is not empty, and does not print out error messages.
- **2** The script correctly reads the operators from the file and outputs correct estimates.
- **1** The script is well-structured and follows the Python style guide (PEP 8).

9.4: Predict and verify task completion times

Using the tool from the previous task, calculate estimates of the task completion times for the following four tasks:

- adding the numbers from 1 to 20 using only the mouse
- adding the numbers from 1 to 20 using only the keyboard
- calculating the result of $(3^2 + 4^2) * 15.2$ using only the mouse.
- calculating the result of $(3^2 + 4^2) * 15.2$ using only the keyboard.

Hint: *You might want to copy the relevant code from klm.py into the notebook or import an appropriate function from klm.py. This allows you to calculate the estimates right in the notebook. Alternatively, you can also calculate the estimates externally with klm.py and only copy the results into the notebook.*

Conduct experiments to determine a set of actual task completion times for *experienced* users. Repeat the experiments several times to get a range of task completion times. Describe how you conducted the experiments (test design, participants, variables, how you mitigated confounding/random variables).

Compare the two estimates (using the operators by Card et al. and your custom set of operators) for each task with the actual task completion times and plot appropriate visualizations.

Hand in the following file:

klm_report.ipynb: an iPython notebook containing a short documentation of your experiment, the predicted task completion times and the actual task completion times.

Points

- **1** KLM estimates for all four tasks have been submitted
- **1** the experiments have been conducted carefully
- **1** the experiments are described in sufficient detail
- **1** the KLM estimates are described in sufficient detail
- **1** experimentally determined task completion times are reported
- **1** visualizations highlight the differences between estimates and actual performances

Submission

Submit via GRIPS until the deadline

All files should use UTF-8 encoding and Unix line breaks. Python files should use spaces instead of tabs. If you need to submit further supporting files, please add a comment describing their use.

Have Fun!