# A Case Study on Model-Based Design - II

# Objective

- In the previous lectures, we learned about the challenge faced by virtual keyboards designers
  - The objective of the designer is to determine an efficient layout
  - The challenge is to identify the layout from a large design space
  - We saw the difficulties in following standard design life cycle

# Objective

- We explored the possibility of using GOMS in the design and discussed its problems
- In this lecture, we shall see another way of addressing the issue, which illustrates the power of model-based design

# Design Approach

- E saw the problem with GOMS in VK design
  - The problem arises due to the task-based analysis, since identifying and analyzing tasks is tedious if not difficult and sometimes not feasible
- We need some approach that is not task based
  - Fitts' Law and Hick-Hyman Law can be useful for the purpose as they do not require task-based analysis

# Fitts'-Digraph Model

- The alternative approach makes use of the Fitts'-diagraph (FD) model

- FD model was proposed to *compute* user performance for a VK from layout specification
  - Layout in terms of keys and their positions
  - Performance in text entry rate

# Fitts'-Digraph Model

- The FD model has three components
  - **Visual search time (RT)**: time taken by a user to locate a key on the keyboard. The Hick-Hyman law is used to model this time

$$RT = a + b \log_2 N$$

N is the total number of keys, a and b are empirically-determined constants

# Fitts'-Digraph Model

- The FD model has three components
  - **Movement time (MT)**: time taken by the user to move his hand/finger to the target key (from its current position). This time is modeled by the Fitts' law

$$MT_{ij} = a' + b' \log_2\left(\frac{d_{ij}}{w_j} + 1\right)$$

$MT_{ij}$ is the movement time from the source (i-th) to the target (j-th) key, $d_{ij}$ is the distance between the source and target keys, $w_j$ is the width of the target key and a' and b' are empirically-determined constants

# Fitts'-Digraph Model

- The FD model has three components
  - **Digraph probability:** probability of occurrence of character pairs or digraphs, which is determined from a corpus

$$P_{ij} = f_{ij} / \sum_{i=1}^{N}\sum_{j=1}^{N} f_{ij}$$

  - $P_{ij}$ is the probability of occurrence of the i-th and j-th key whereas $f_{ij}$ is the frequency of the key pair in the corpus

# Fitts'-Digraph Model

- Using the movement time formulation between a pair of keys, an average (mean) movement time for the whole layout is computed

$$MT_{MEAN} = \sum_{i=1}^{N}\sum_{j=1}^{N} MT_{ij} \times P_{ij}$$

- The mean movement time is used, along with the visual search time, to compute user performance for the layout

# Fitts'-Digraph Model

- Performance is measured in terms of characters/second (CPS) or words/minute (WPM)
- Performances for two categories of users, namely novice and expert users, are computed

# Fitts'-Digraph Model

- Novice user performance: they are assumed to be unfamiliar with the layout. Hence, such users require time to search for the desired key before selecting the key

$$CPS_{Novice} = \frac{1}{RT + MT_{MEAN}}$$

$$WPM = CPS \times (60 / W_{AVG})$$

$W_{AVG}$ is the average number of characters in a word. For example, English words have 5 characters on average

# Fitts'-Digraph Model

- Expert user performance: an expert user is assumed to be thoroughly familiar with the layout. Hence, such users don't require visual search time

$$CPS_{Expert} = \frac{1}{MT_{MEAN}}$$

$$WPM = CPS \times (60 / W_{AVG})$$

$W_{AVG}$ is the average number of characters in a word. For example, English words have 5 characters on average

# Using the FD Model

- If you are an expert designer
  - You have few designs in mind (experience and intuition helps)
  - Compute WPM for those
  - Compare

# Using the FD Model

- Otherwise
  - Perform *design space exploration* – search for a good design in the design space using algorithm
- Many algorithms are developed for design space exploration such as dynamic simulation, Metropolis algorithm and genetic algorithm
  - We shall discuss one (Metropolis algorithm) to illustrate the idea

# Metropolis Algorithm

- A “Monte Carlo” method widely used to search for the minimum energy (stable) state of molecules in statistical physics
- We map our problem (VK design) to a minimum-energy state finding problem in statistical physics

# Metropolis Algorithm

- We map a layout to a molecule (keys in the layout serves the role of atoms)
- We redefine performance as the average movement time, which is mapped to energy of the molecule
- Thus, our problem is to find a layout with minimum energy

# Metropolis Algorithm

- Steps of the algorithm
  - Random walk: pick a key and move in a random direction by a random amount to reach a new configuration (called a *state*)
  - Compute energy (average movement time) of the state
  - Decide whether to retain new state or not and iterate

# Metropolis Algorithm

- The decision to retain/ignore the new state is taken on the basis of the decision function, where ΔE indicates the energy difference between the new and old state (i.e., ΔE = energy of new state – energy of old state)

$$W(O-N)=\begin{cases} e^{-\frac{\Delta E}{kT}} & \Delta E > 0 \\ 1 & \Delta E \le 0 \end{cases}$$

# Metropolis Algorithm

- W is probability of changing from old to new configuration
- k is a coefficient
- T is “temperature”
- Initial design: a “good” layout stretched over a “large” space

# Metropolis Algorithm

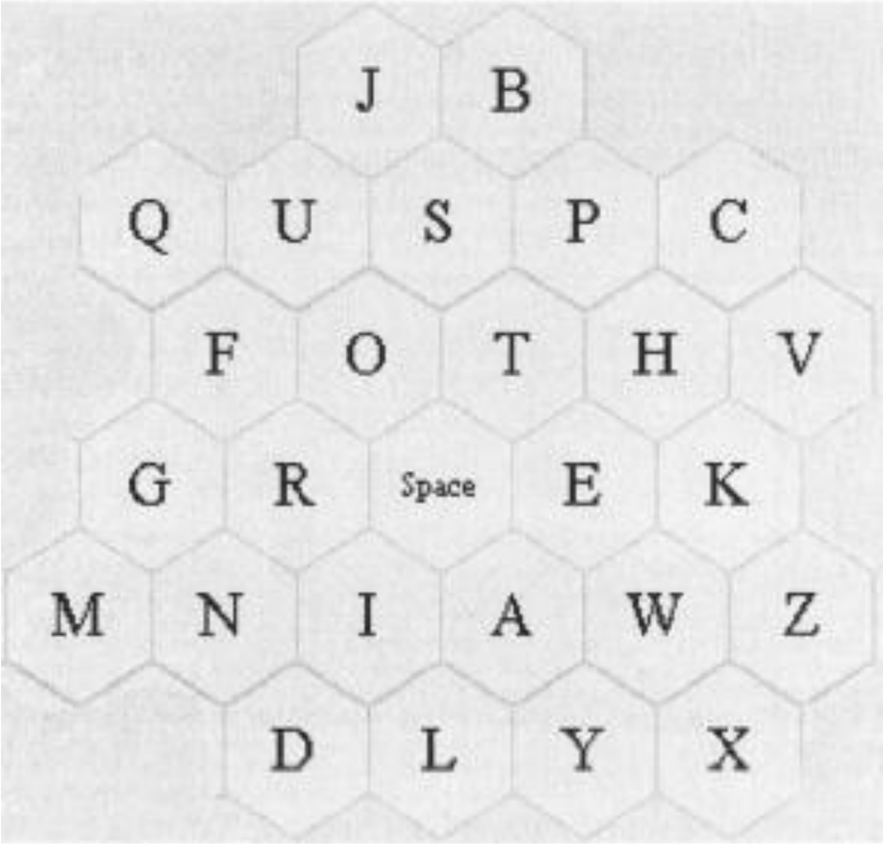
- Note the implications of the decision function
  - If energy of the new state is less than the current state, retain the new state
  - If the new state is having more energy than the current state, don't discard the new state outright. Instead, retain the new state if the probability W is above some threshold value. This steps helps to avoid local minima

# Metropolis Algorithm

- To reduce the chances of getting struck at the local minima further, “annealing” is used
  - Bringing “temperature” through several up and down cycles

# Metropolis Algorithm

An example VK layout, called the Metropolis layout, is shown, which was designed using the Metropolis algorithm

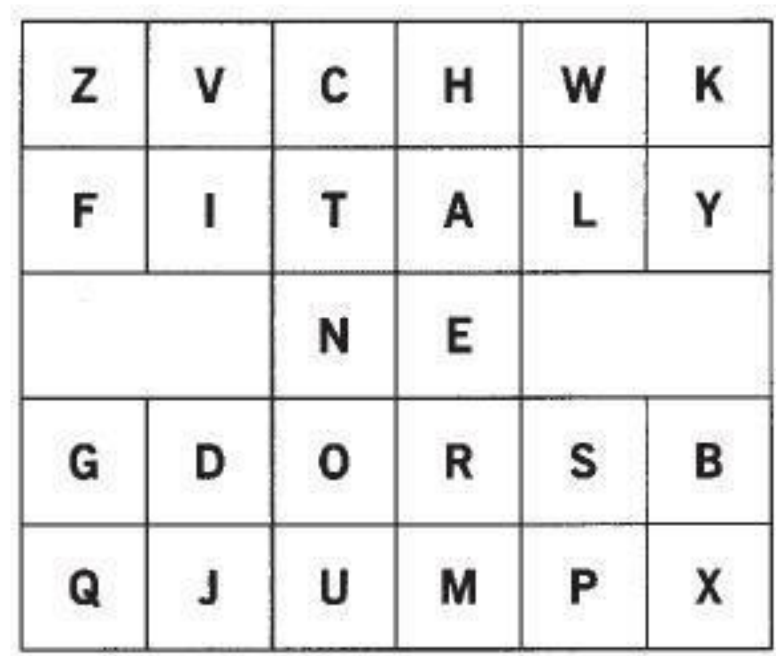


# Some VK Layouts with Performance

- QWERTY
  - 28 WPM (novice)
  - 45.7 WPM (expert)

# Some VK Layouts with Performance

- QWERTY
  - 28 WPM (novice)
  - 45.7 WPM (expert)
- FITALY
  - 36 WPM (novice)
  - 58.8 WPM (expert)



# Some VK Layouts with Performance

- QWERTY
  - 28 WPM (novice)
  - 45.7 WPM (expert)
- FITALY
  - 36 WPM (novice)
  - 58.8 WPM (expert)
- OPTI II
  - 38 WPM (novice)
  - 62 WPM (expert)

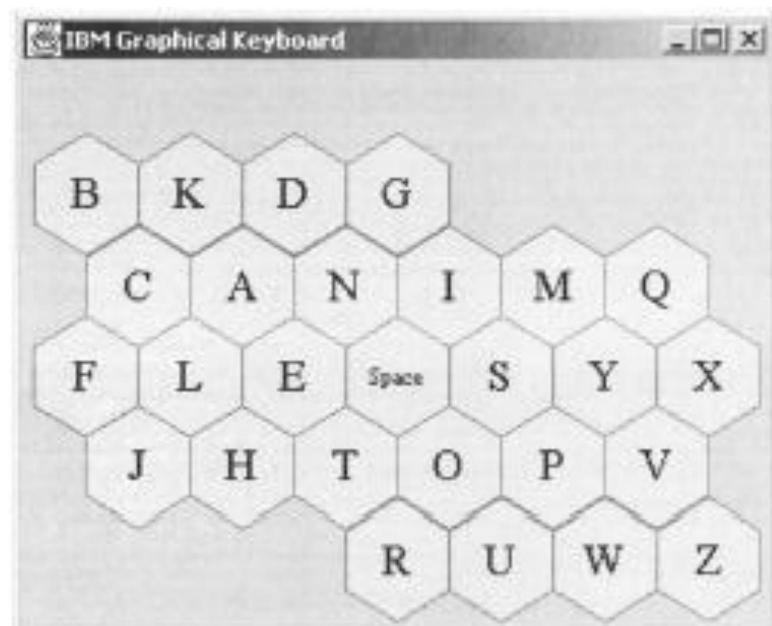
| | | | | | |
|---|---|---|---|---|---|
| Q | K | C | G | V | J |
| | S | I | N | D | |
| W | T | H | E | A | M |
| | U | O | R | L | |
| Z | B | F | Y | P | X |

# Some VK Layouts with Performance

- The layouts mentioned before were not designed using models
- They were designed primarily based on designer's intuition and empirical studies
- However, the performances shown are computed using the FD model

# Some VK Layouts with Performance

- ATOMIK – a layout designed using slightly modified Metropolis algorithm
- Performance of the ATOMIK layout
  - 41.2 WPM (novice)
  - 67.2 WPM (expert)



# Some VK Layouts with Performance

- Note the large performance difference between the ATOMIK and other layouts
- This shows the power of model-based design, namely a (significant) improvement in performance without increasing design time and effort (since the design can be mostly automated)