Input Performance

KLM

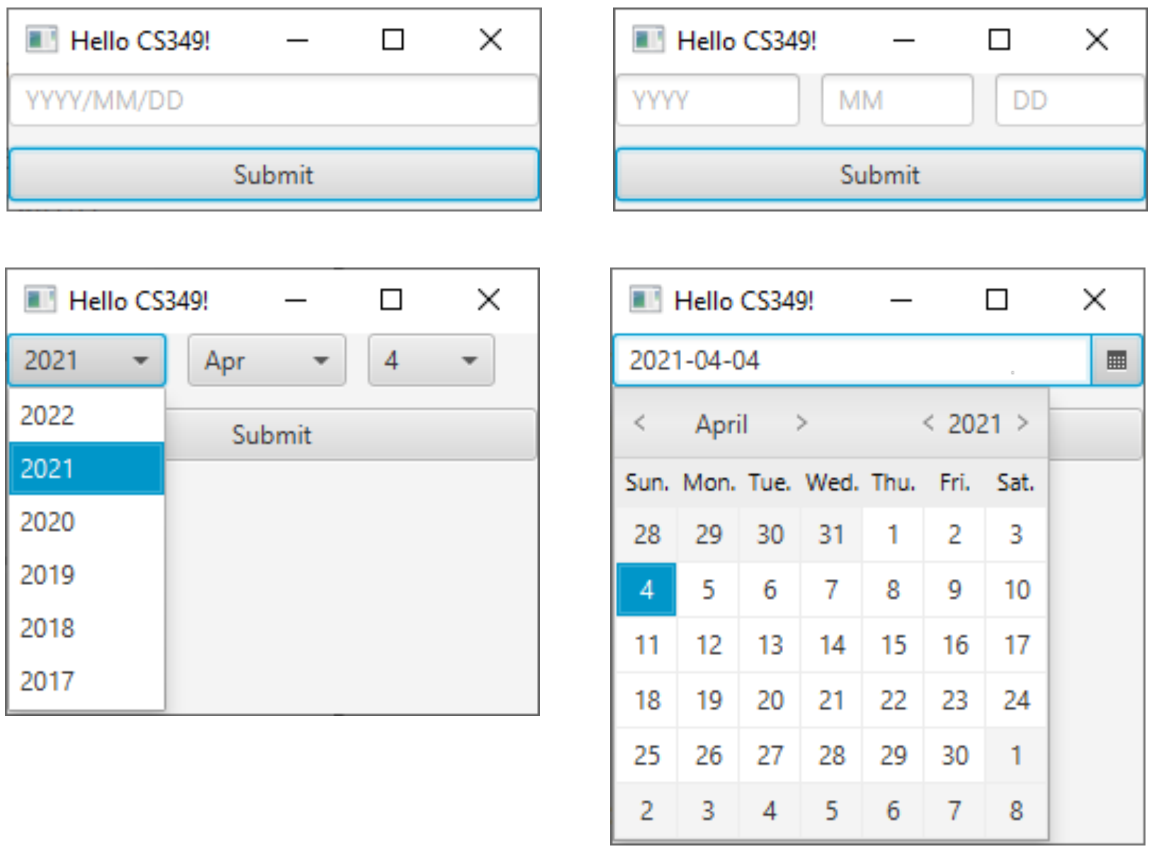
Fitts' Law

Jul 24

CS 349

Input Performance Models

When designing a user interface, designers might have to choose between different designs. Implementing all of them, thought, might require too many resources. There must be a way to estimate the performance of a user interface without testing it.

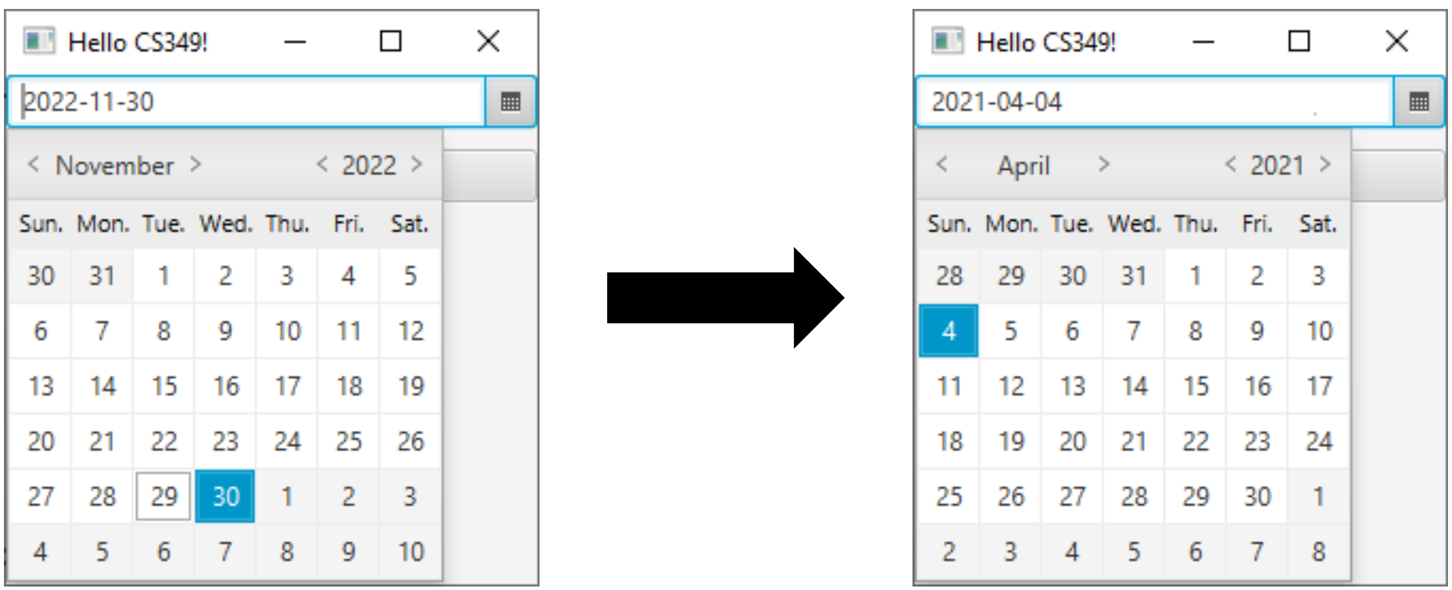


KLM

Input Performance Models

There are models that abstract how people would use input devices and a user interface, which enables designers to predict time, error, fatigue, learning, etc.

Models most often focus on time and error, as they are easiest to measure.



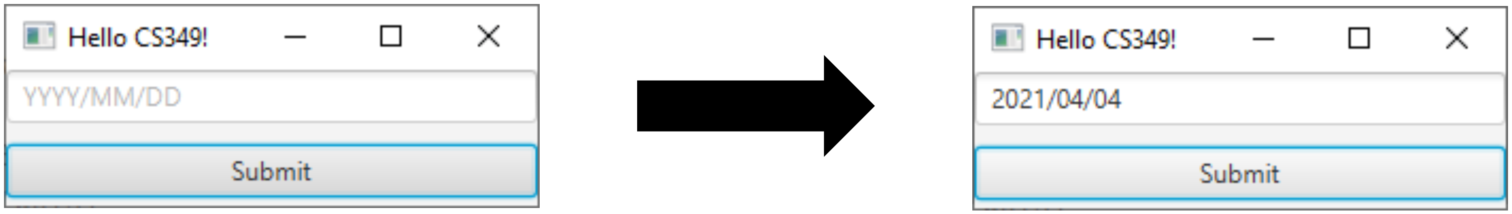
Keystroke Level Model (KLM)

When using KLM, describe each task with a sequence of operators:

- **K**: Keystroke: typing a single keyboard key
- **P**: Pointing: moving the mouse cursor from one location to another
- **B**: Button: pressing or releasing a mouse button
- **H**: Home: move hand between mouse and keyboard
- **M**: Mental Preparation: planning the next routine action, e.g., finding an icon on the screen.

Sum up times for each operator to estimate how long the task takes.

KLM Example



- Assumption: one hand on keyboard, one on the mouse
  1. Moving mouse to the TextField **P**
  2. Clicking mouse button to set focus on TextField **BB**
  3. Switching to Keyboard **H**
  4. Pressing `2 0 2 1 / 0 4 / 0 4` **KKKKKKKKKK**
  5. Switching to Mouse **H**
  6. Moving mouse to the Button **P**
  7. Clicking mouse button to activate Button **BB**

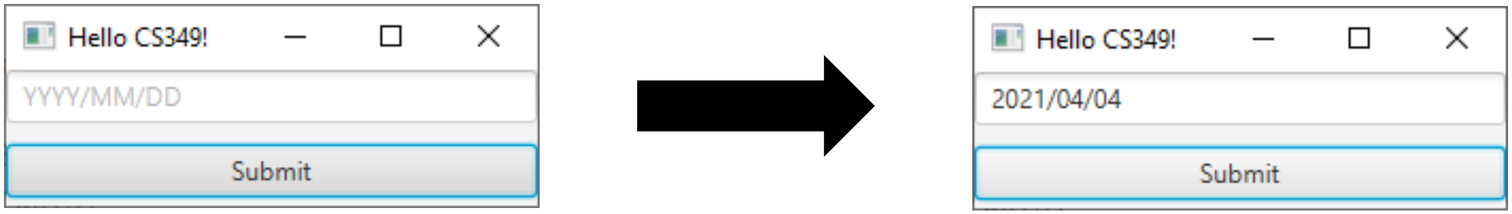
5. – 7. could have been replaced with

5. Press `<TAB>` to move focus to Button **K**
6. Press `<SPACE>` to activate Button **K**

KLM Operators

Code	Operation		Time [s]
K	Key typed	Novice typist	1.20
		Average typist	0.28
		Expert typist	0.12
		Random key	0.50
		Key combination	0.75
P	Point cursor at target		1.10
B	Button pressed / released		0.10
H	Move hand		0.40
M	Mental preparation		1.20+

KLM Example



Assumption: one hand on keyboard, one on the mouse

1.	**P**	1.1 s
2.	**BB**	0.2 s
3.	**H**	0.4 s
4.	**KKKKKKKKKK**	2.8 s
5.	**H**	0.4 s
6.	**P**	1.1 s
7.	**BB**	0.2 s = 6.2 s

5. – 7. could have been replaced with

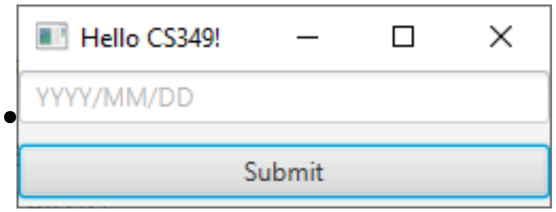
5.	**K**	0.28 s
6.	**K**	0.28 s = 5.06 s

KLM Examples

Use KLM to compare performance time of date entry widgets.

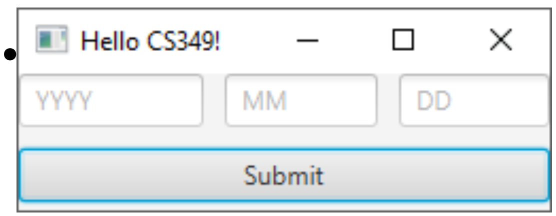
Assumption: one hand on keyboard, one on the mouse

Op	Time [s]
K	0.28
P	1.10
B	0.10
H	0.40
M	1.20+



P BB H KKKKKKKKKK K K = 5.06 s

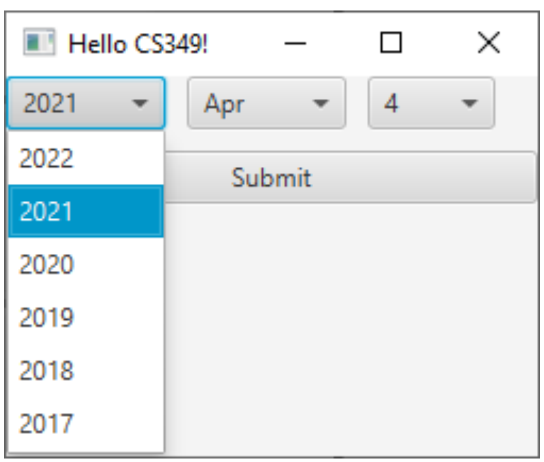
2021/04/04 <TAB> <SPACE>



Assumption: cursor jumps to next field

P BB H KKKK KK KK K = 4.22 s

2021 04 04 <SPACE>



Assumption: auto-select

P BB H KKKK K K K K K K = 4.5 s

2021 <TAB> A <TAB> 4 <TAB> <SPACE>

Including Mental Operators (M)

Most actions when interacting with a UI do not require conscientious cognitive processing. Sometimes, however, users need to contemplate or strategize their actions beforehand. This can be modelled by one (or multiple) **M** operations.

Examples include:

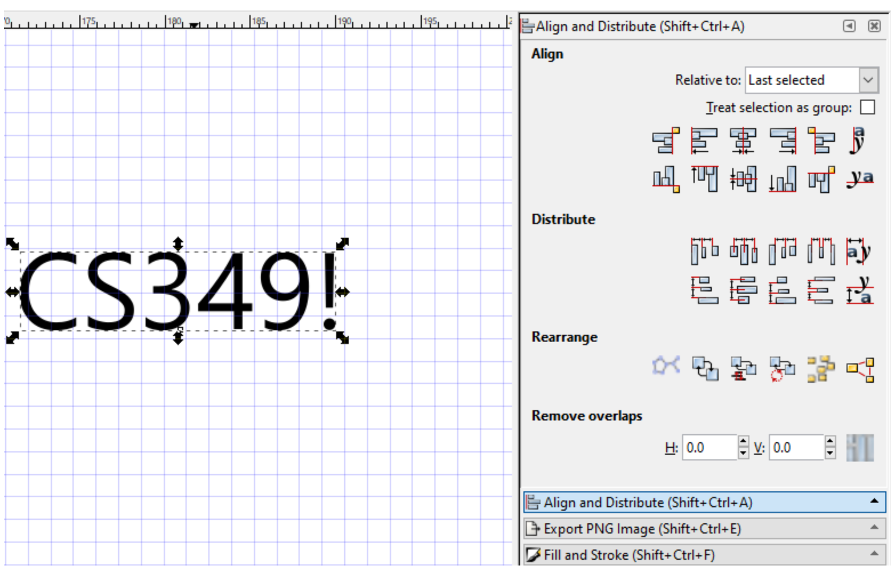
- initiating a new task: e.g., typing a text, then changing the text font.
- retrieving information from semantic (long-term) memory
- find something on the display: e.g., finding a currently visible icon
- think of a task parameter: e.g., setting font-size
- verify that a specification / action is correct (i.e., evaluate feedback)

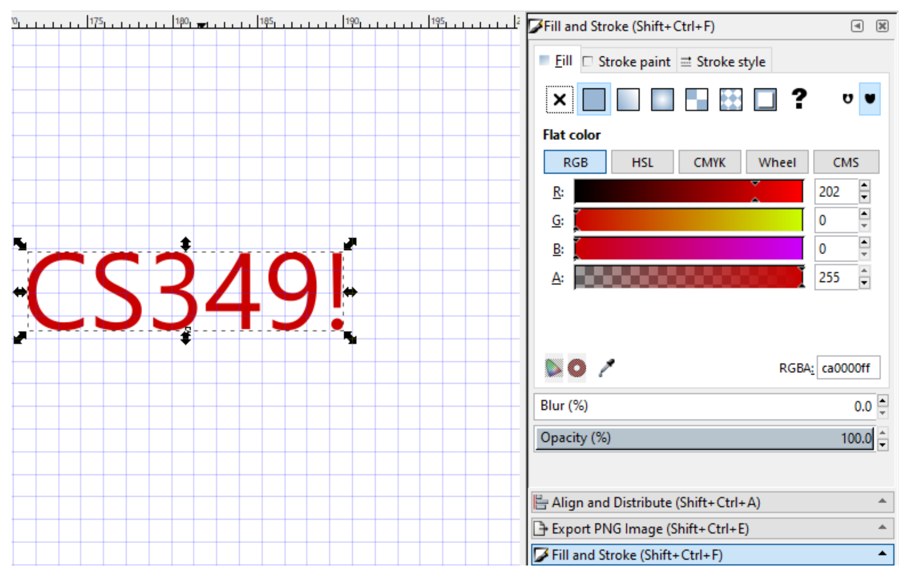
Including Mental Operators (M)

Task: Make the text red.

Assumption: text highlighted, mouse over text

- Find “Fill and Stroke”-menu: M 1.2 s
- Move mouse cursor to menu and click: P BB 1.3 s
- Move mouse cursor to “R”-bar: P 1.1 s
- Contemplate value of red (being undecisive): MM 2.4 s
- Move mouse to the right value and click: P BB 1.3 s = 7.3 s





KLM Critique

Advantages:

- Easy to model
- Does not require mockup, prototype, or implementation

Disadvantages:

- Some time estimates are generalizing too much
- Some time estimates are inherently variable

KLM Critique

KLM does not model pointing well and instead uses constant 1.1 s for pointing:

- some pointing devices are faster than others
- intuitively, it should take longer to move the mouse a long distance, or point at a small target

Fitts' Law

Fitts' Law

Fitts' Law is a predictive model for pointing time considering pointing device, travel distance, and target size.

- based on rapid, aimed movements
- works for many kinds of pointing "devices": finger, pen, mouse, joystick, foot, ..

Paul Fitts

- Psychologist at Ohio State University
- Early advocate of user-centred design (in terms of matching system to human capabilities)

Distance vs. Size

The travel distance D proportional to the movement time MT, and the target size S is negatively proportional the **time** MT:

$$MT \propto \frac{D}{S}$$

The actual law is slightly more complex (and precise):

$$MT = a + b\log_2\left(\frac{D}{W} + 1\right)$$

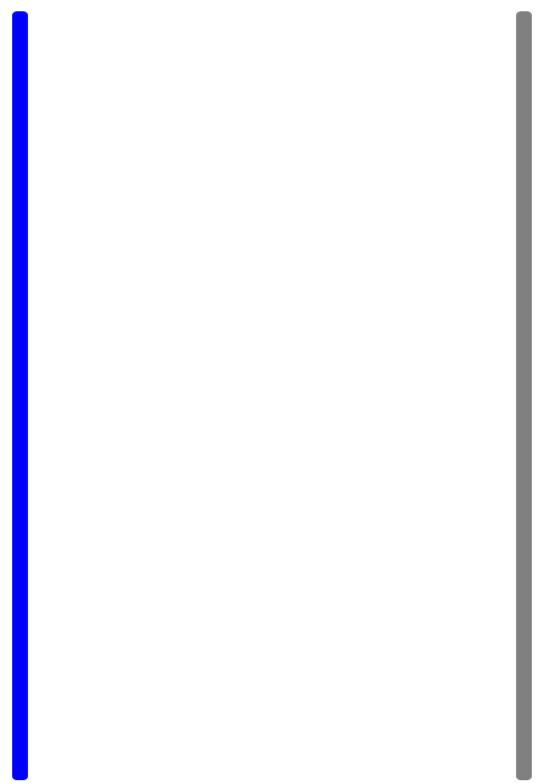
- MT is mean-time
- a & b are device-specific constants
- D is the distance to the target
- W is the target width

Fitts' Law Tests

How did we derive the formula? Experimentation!

When blue rectangle appears, click on it as fast as possible

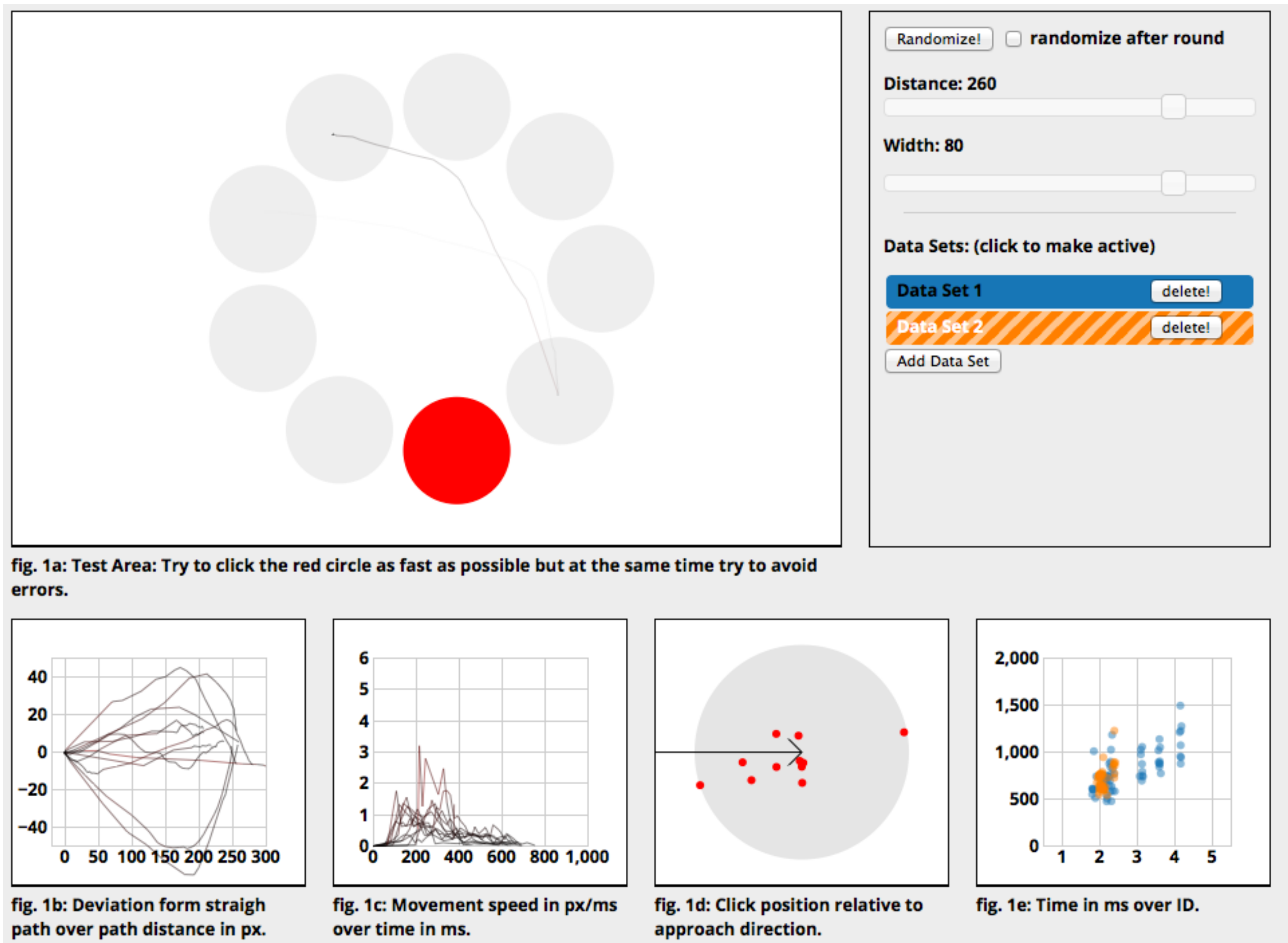
http://ergo.human.cornell.edu/FittsLaw/FittsLaw.html



Fitts' Law Tests

When blue rectangle appears, click on it as fast as possible

http://www.simonwallner.at/ext/fitts/



Index of Difficulty

$$MT = a + b\log_2\left(\frac{D}{W} + 1\right)$$

ID (Index of Difficulty)

IP (Index of Performance): $^1/_b$

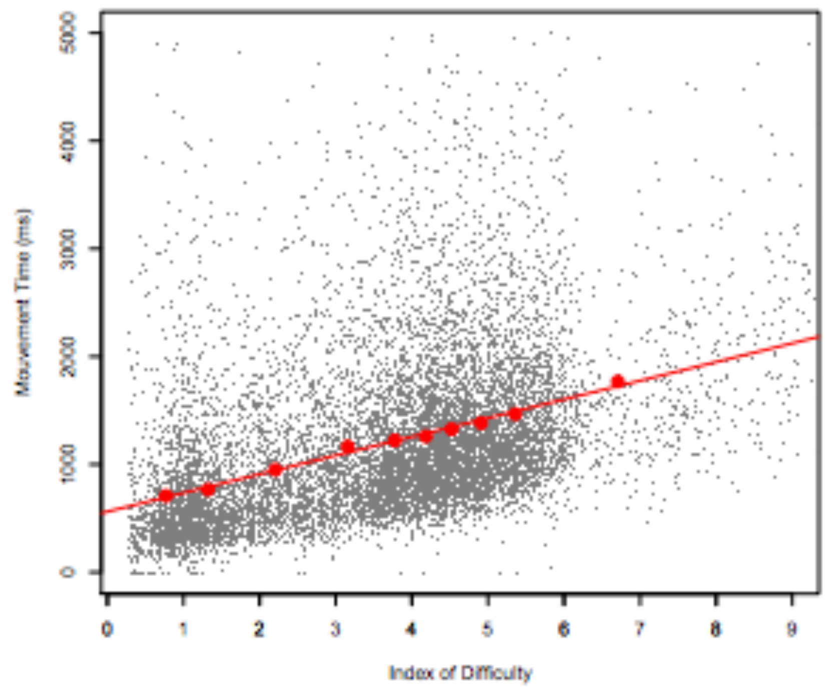


Figure 7. Fitts' linear fit of *MT* by *ID* using the averaging process for one user (# 4). Small (black) points represent all the points. Large (red) points are the means of each quantile.

Fitts’ Law in the Wild

- Large scale study logging cursor movements with real applications

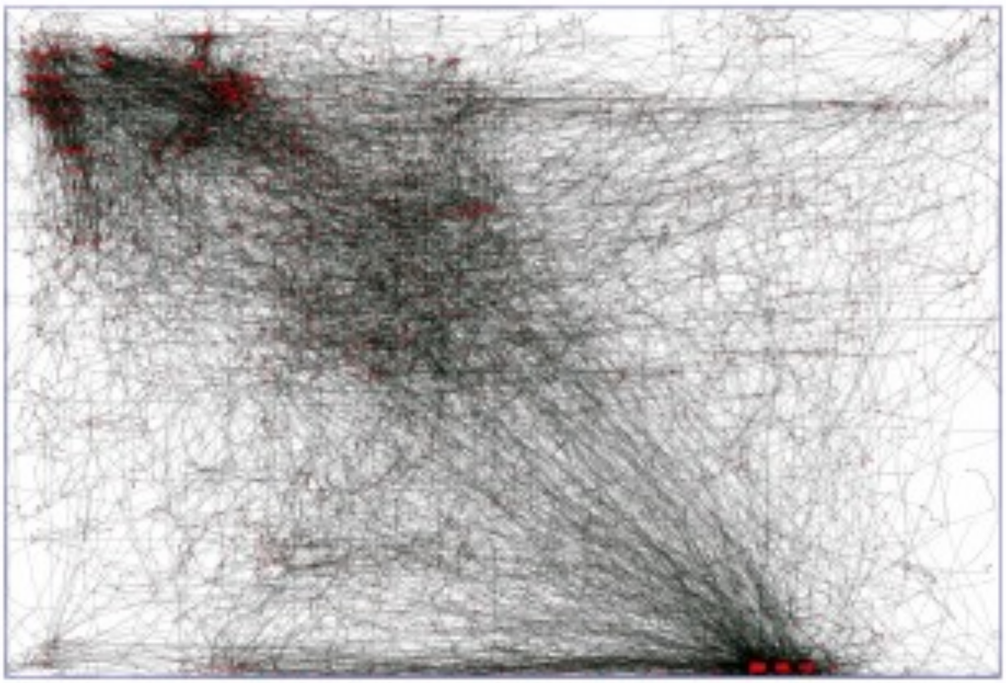


Figure 1. Mouse trajectories (black) and clicks (red).

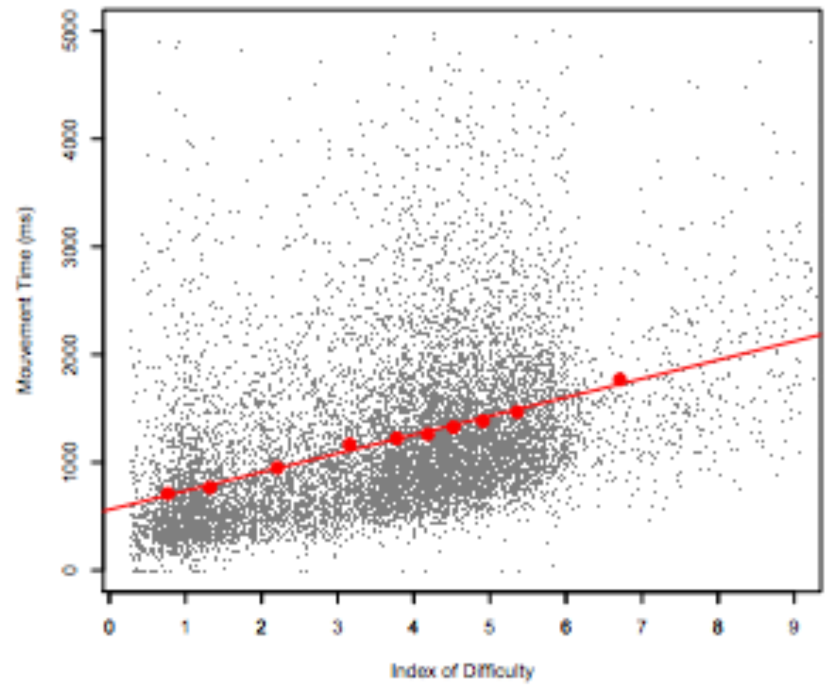


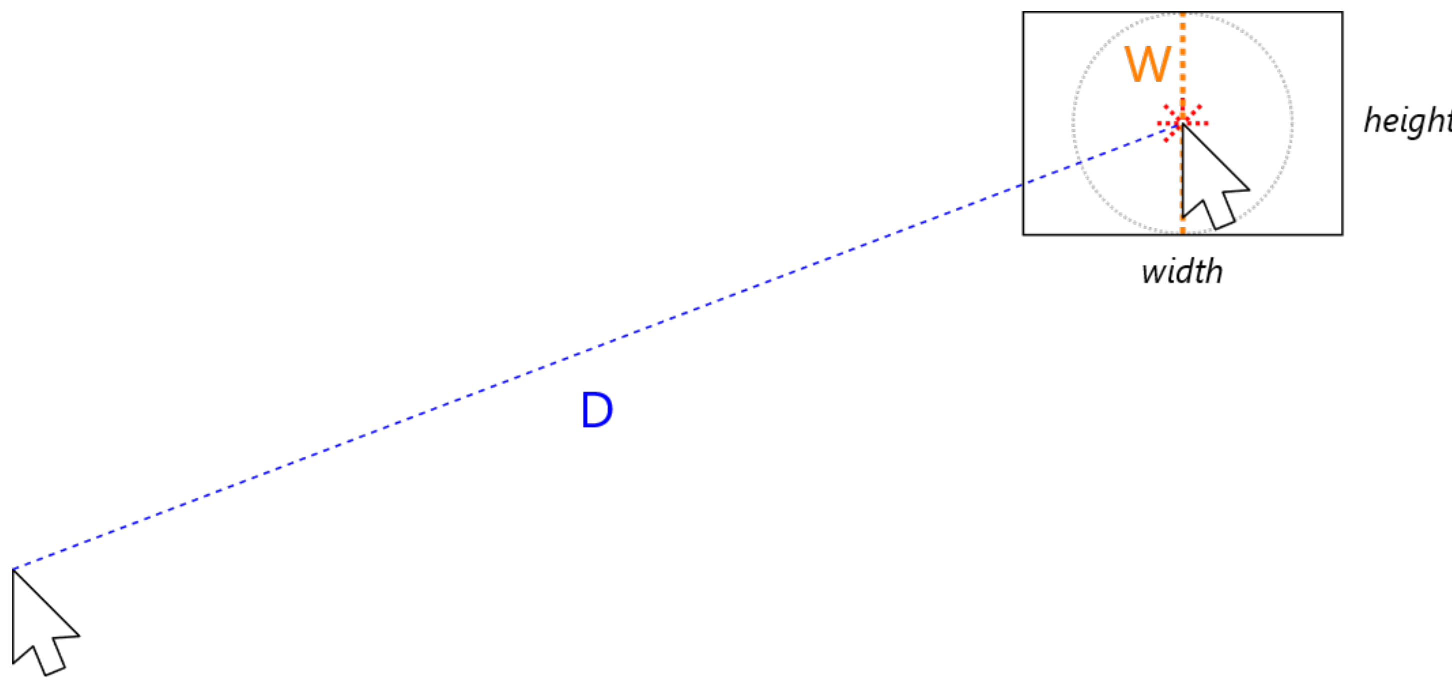
Figure 7. Fitts’ linear fit of *MT* by *ID* using the averaging process for one user (# 4). Small (black) points represent all the points. Large (red) points are the means of each quantile.

Chapuis et al. Fitts' Law in the Wild: A Field Study of Aimed Movements (2007)

2D Targets: W as Minimum of Target width and height

$$MT = a + b \log_2 \left(\frac{D}{W} + 1 \right)$$

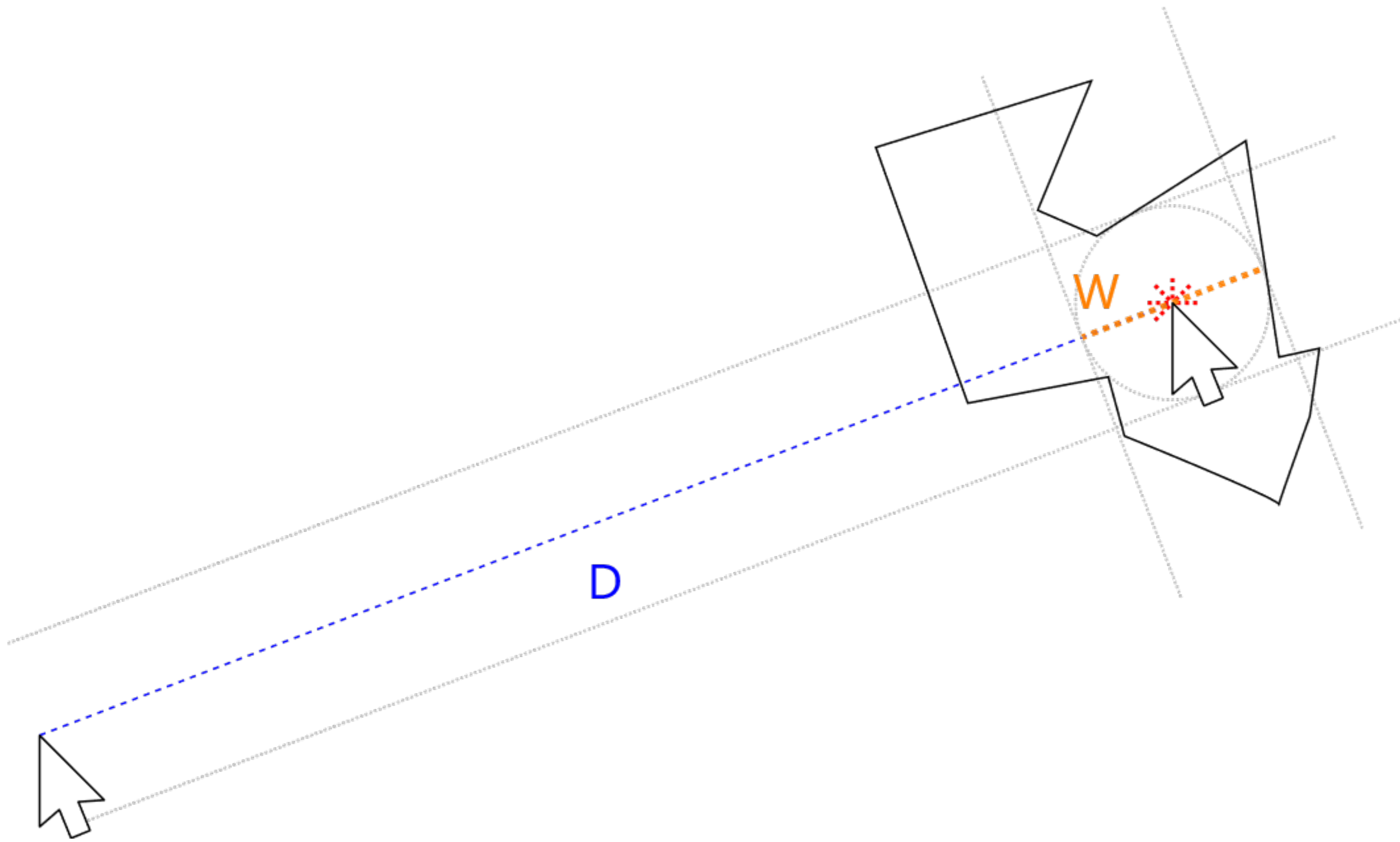
For simplification, we usually interpret W as the minimum of the targets' $width$ and $height$: $W = \min(width, height)$.



2D Targets: W as Cross Section of the Pointer Trajectory

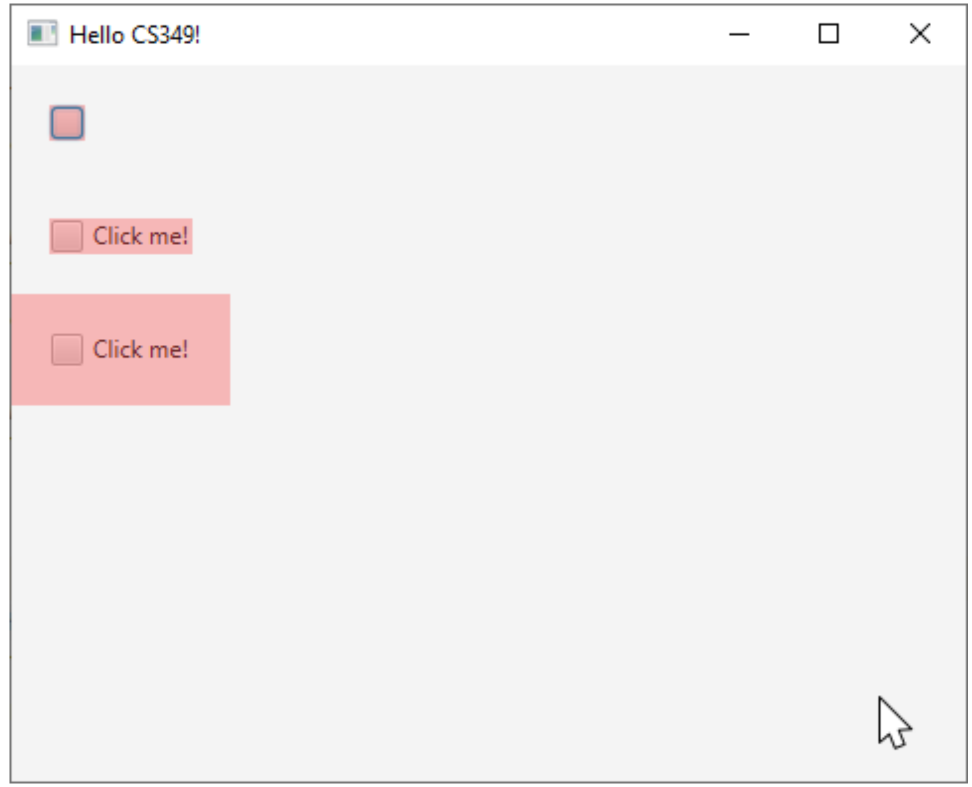
W can be interpreted as the minimum error a user can make along the moving direction (“overshooting”) and perpendicular to it (“off-target”).

W can be represented as largest circle that can be inscribed in the target.



Example

1. Checkbox *vs*
2. Labelled checkbox *vs*
3. Labeled checkbox w. margins



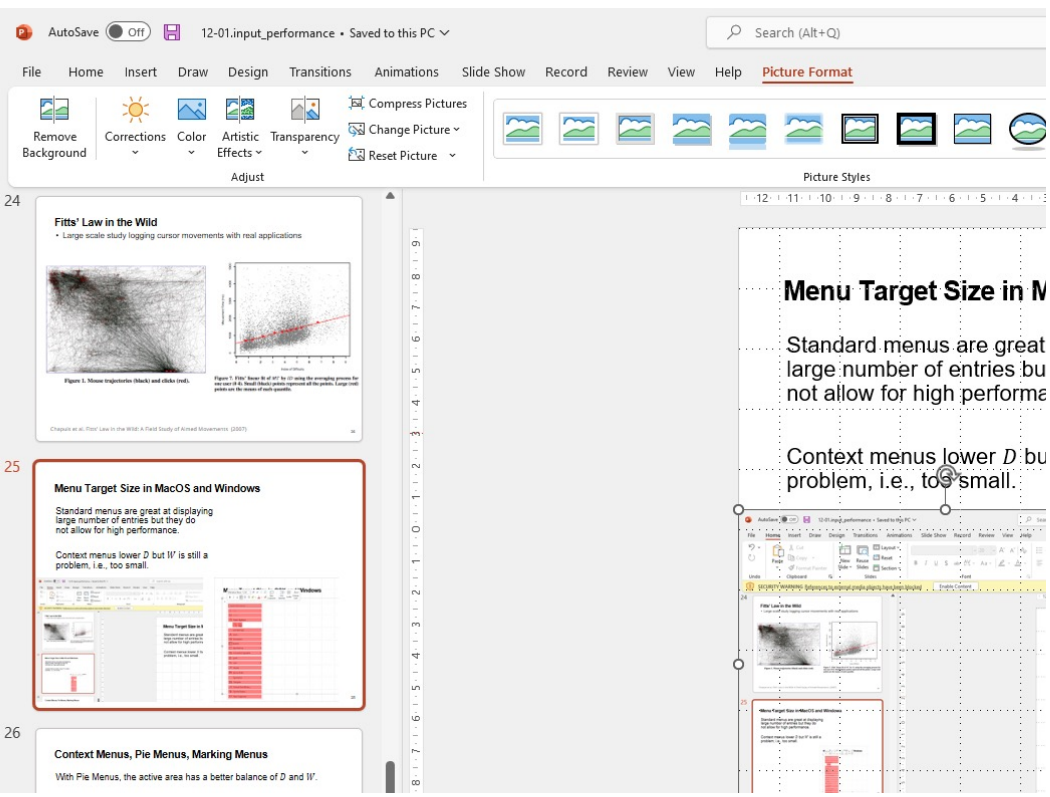
Assuming: $a = 150$, $b = 450$:

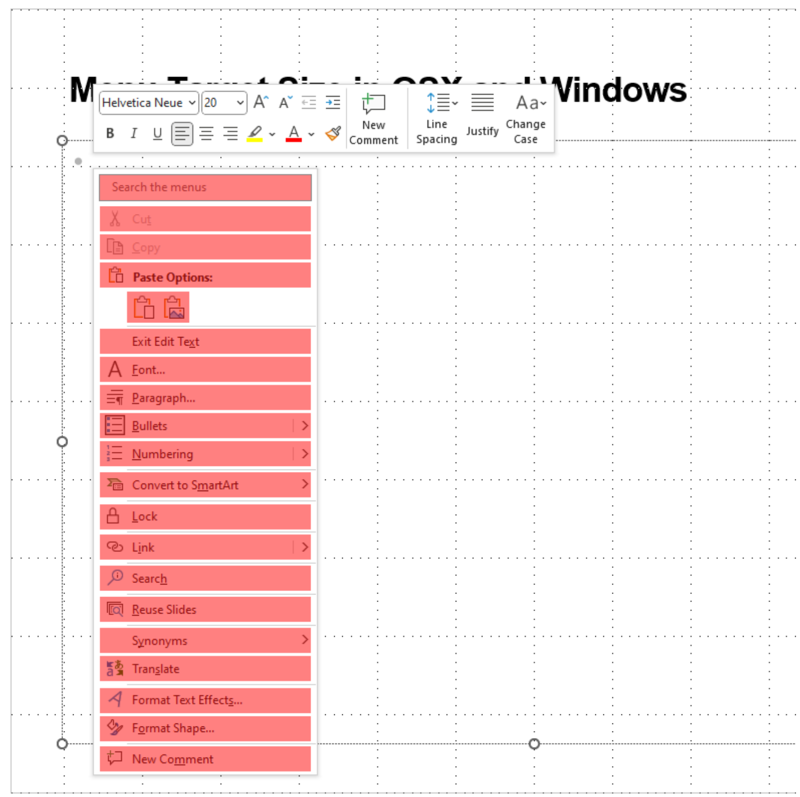
1. $MT = a + b\log_2\left(\frac{500}{18} + 1\right) = a + 4.85b \Rightarrow MT = 2331\text{ ms} = 2.33\text{ s}$
2. $MT = a + b\log_2\left(\frac{445}{18} + 1\right) = a + 4.49b \Rightarrow MT = 2258\text{ ms} = 2.26\text{ s}$
3. $MT = a + b\log_2\left(\frac{425}{54} + 1\right) = a + 3.15b \Rightarrow MT = 1567\text{ ms} = 1.57\text{ s}$

Menu Target Size in MacOS and Windows

Standard menus are great at displaying large number of entries, but they do not allow for high performance.

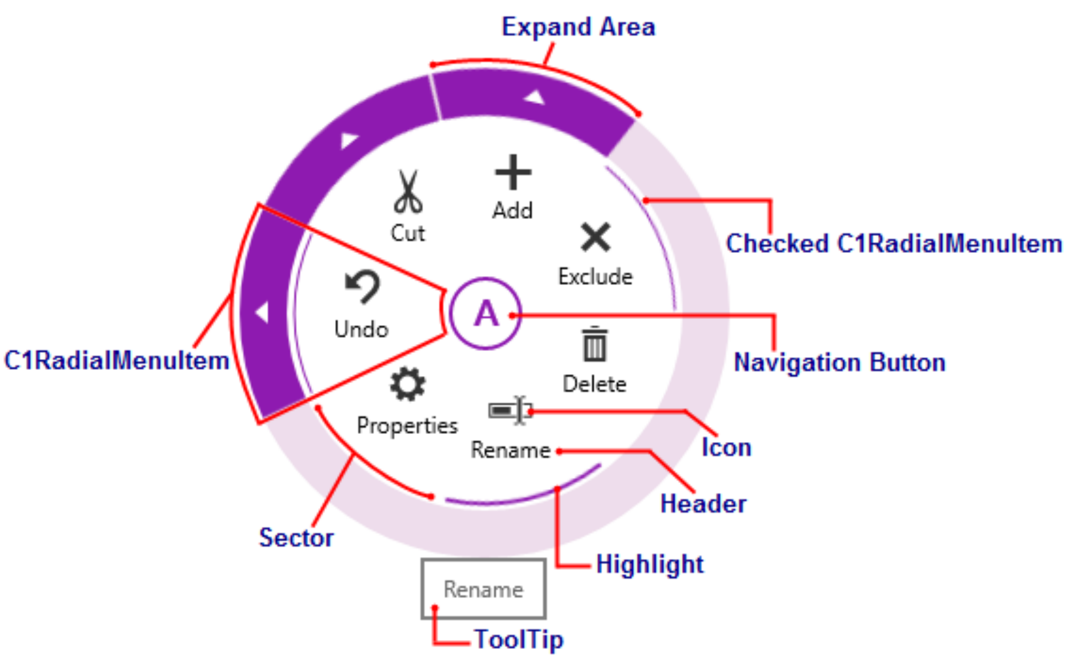
Context menus lower D, but W is still a problem, i.e., too small.

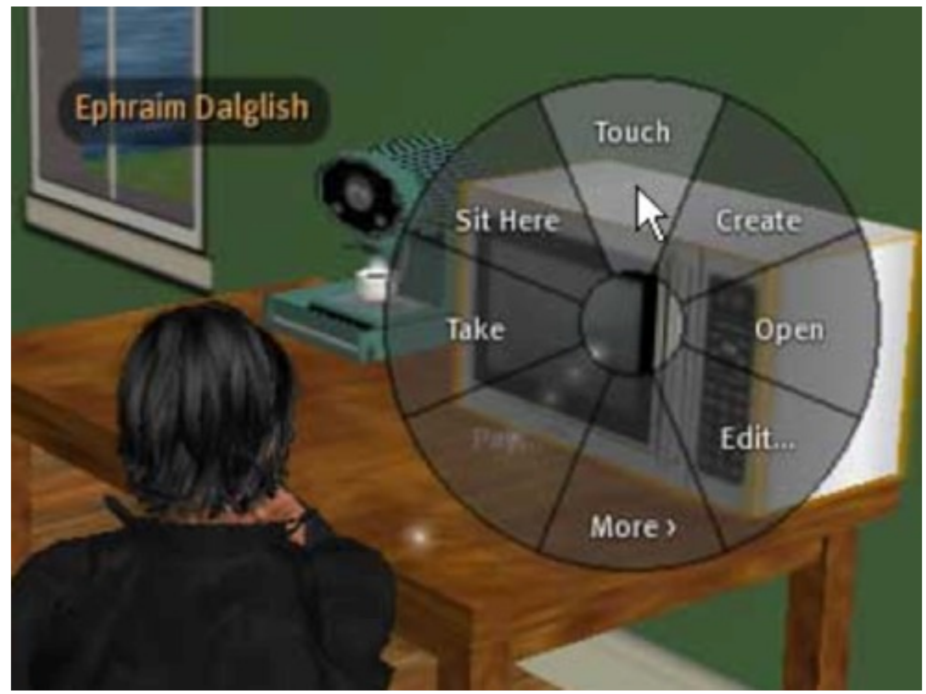


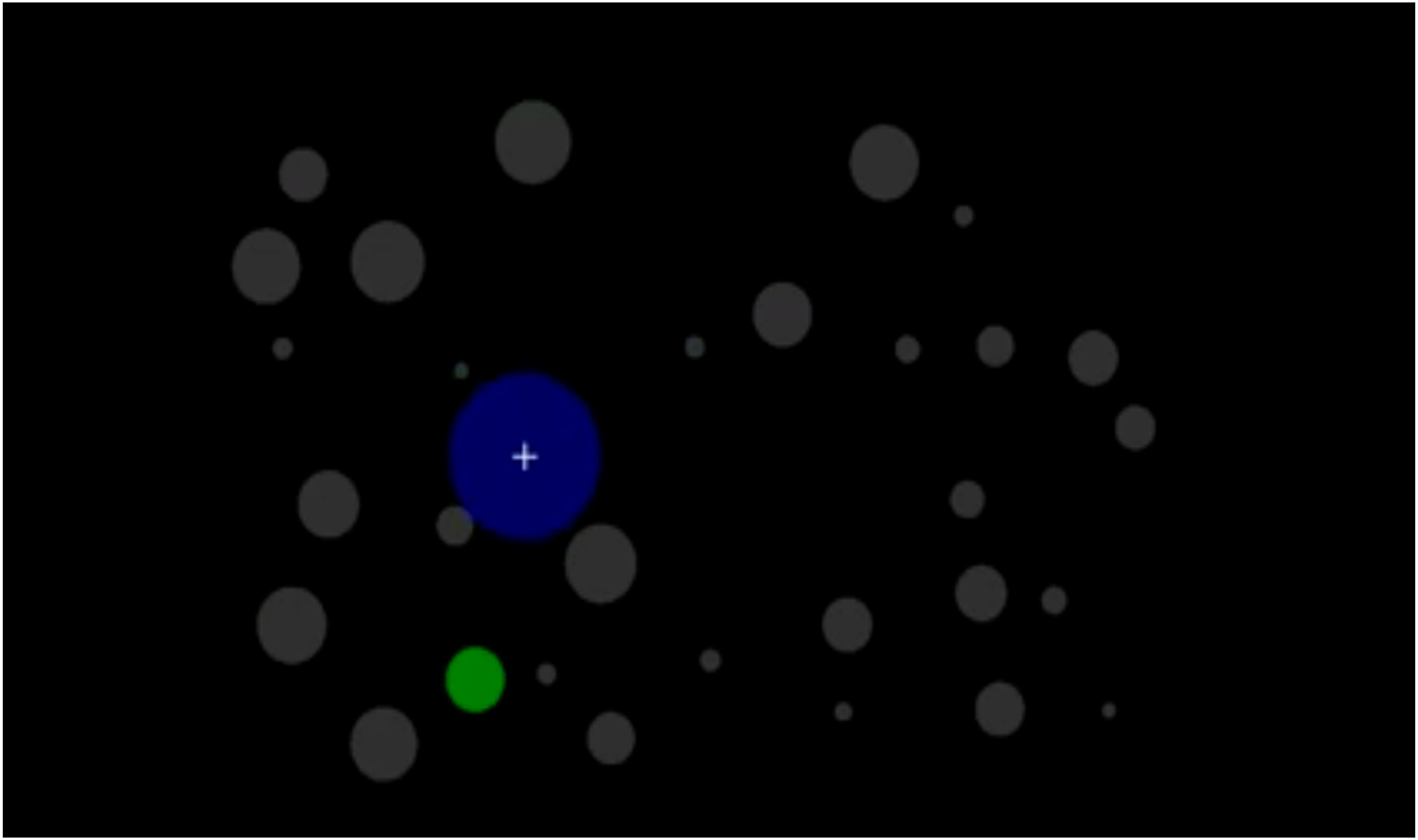


Context Menus, Pie Menus, Marking Menus

With Pie menus or radial menus, the active area has a better balance of D and W.







Bubble Cursor (Grossman and Balakrishnan, 2005) http://youtu.be/JUBXkD_8ZeQ

Motor Space *vs.* Visual Space

Dynamically change CD Gain based on position of cursor:

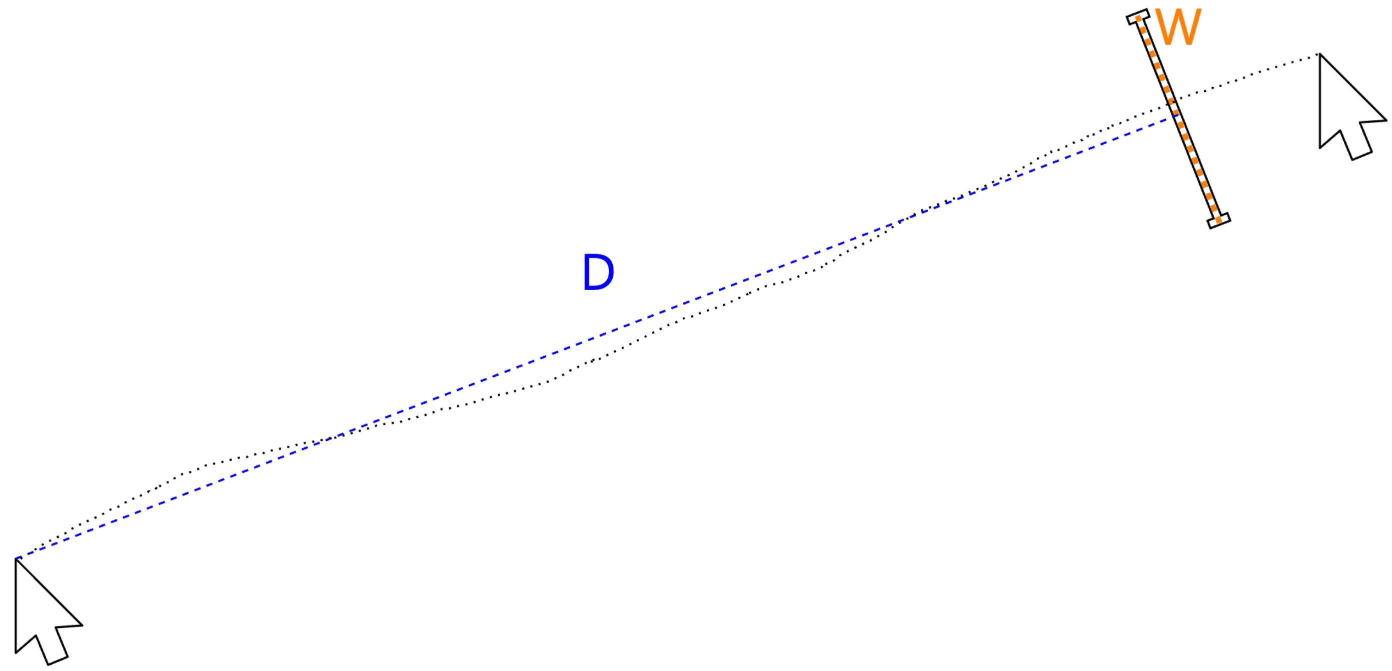
- Make the cursor move more slowly when over the save button makes it larger in “motor space” even though it looks the same size in “screen space”.


```
var prevCoords = Point2D(0.0, 0.0)
val btnSticky = Button("Sticky").apply {
    addEventFilter(MouseEvent.MOUSE_ENTERED) {
        prevCoords = Point2D(it.screenX, it.screenY)
    }
    addEventHandler(MouseEvent.MOUSE_MOVED) {
        prevCoords = Point2D(prevCoords.x + (it.screenX - prevCoords.x) / 4.0,
                             prevCoords.y + (it.screenY - prevCoords.y) / 4.0)
        Robot().mouseMove(prevCoords.x, prevCoords.y)
    }
}
```

Crossing Selection

Steering the cursor through a target of width W.

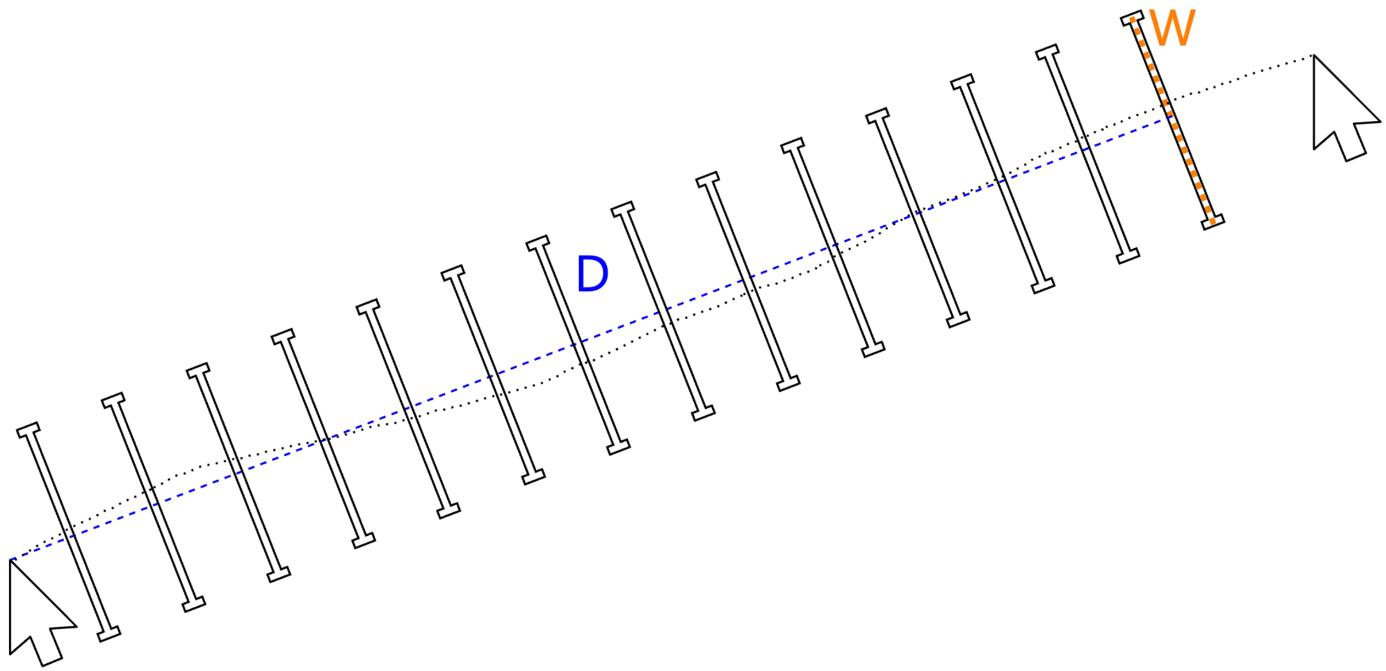
- Fitts' Law is valid for this task as well.



Steering Law

Steering a cursor along a path without exiting the imaginary "tunnel“.

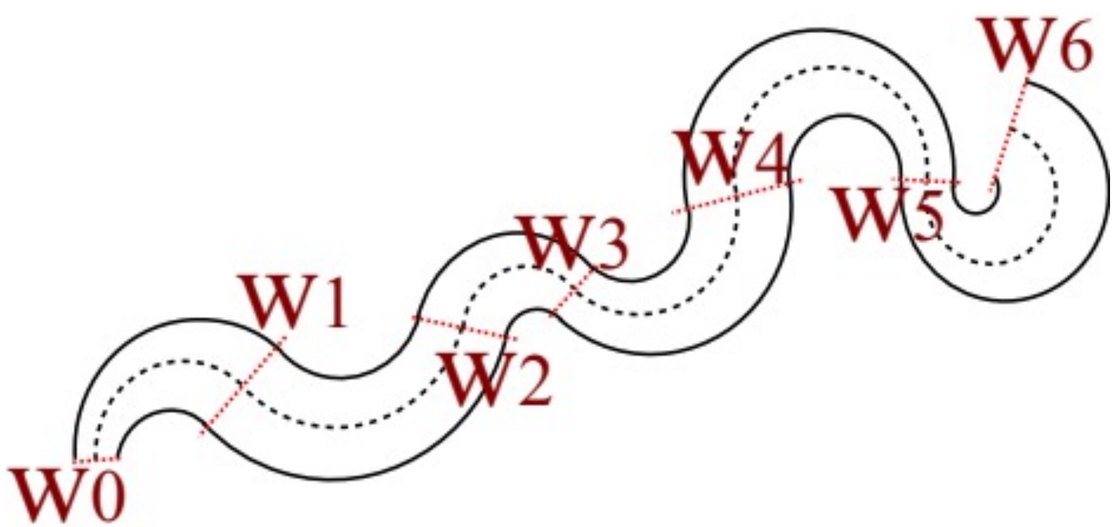
- An adaptation of Fitts' Law: $MT = a + \frac{b}{\ln 2} \times \frac{D}{W} = a + \tilde{b}\frac{D}{W}$



Steering Law for Irregular Paths

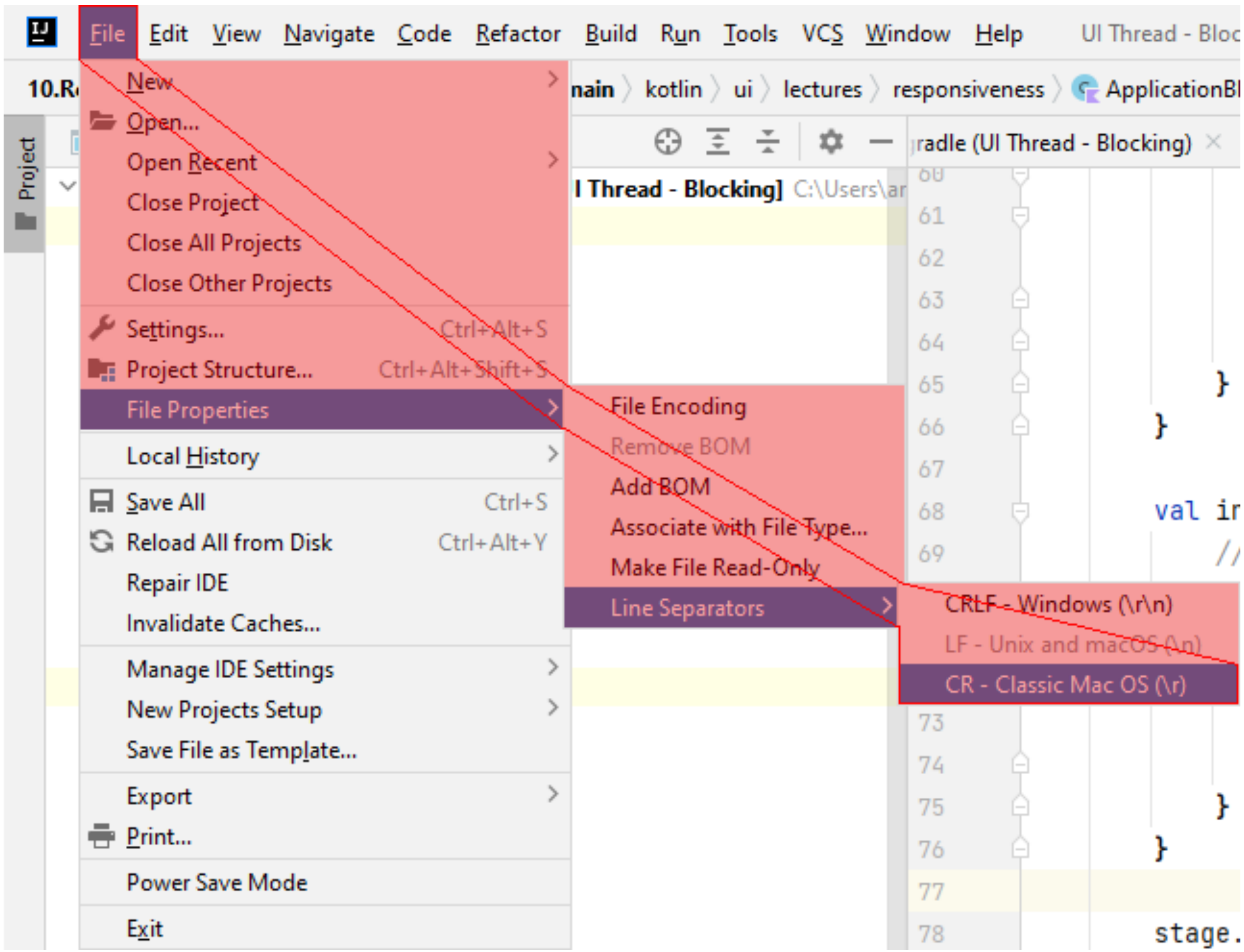
Steering a cursor through along a path of varying width.

- An extension of Fitts' Law: $MT = a + \tilde{b} \int_0^A \frac{1}{W(s)} \, ds$



Steering Law

Moving through a constrained path takes longer



End of the Chapter

Any further questions?