

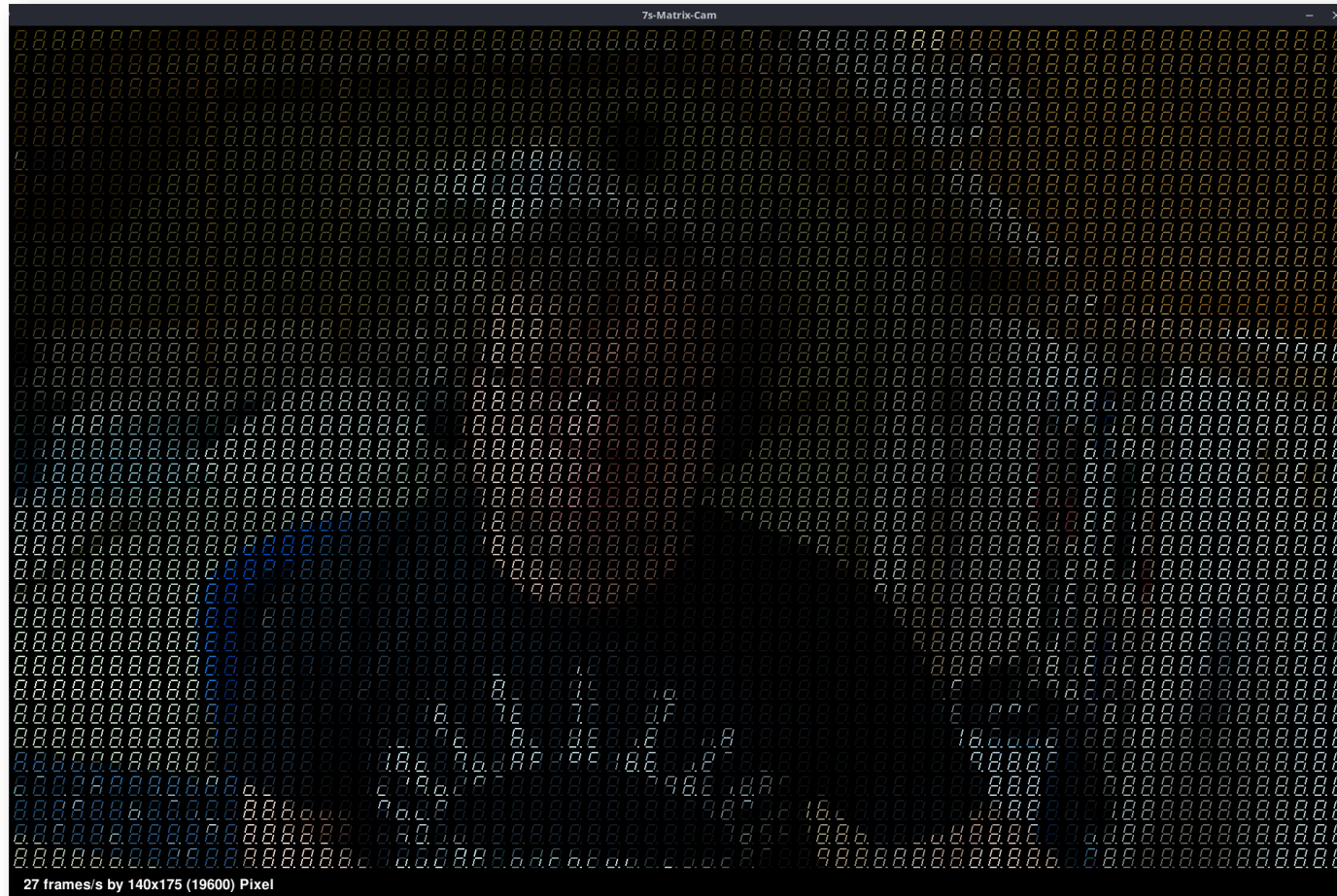


# **Eine Matrix aus 7-Segment- Anzeigen?**

Uwe Berger  
bergeruw@gmx.net

---

# Uwe Berger



# Motivation

## Hackaday 2013

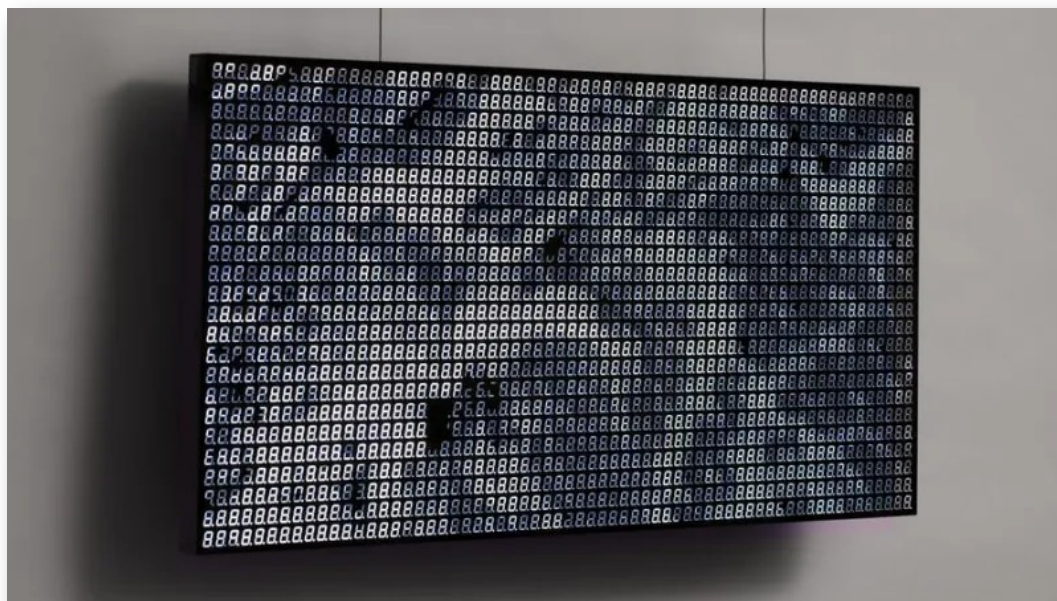
→ <https://hackaday.com/2013/11/21/7-segment-display-matrix-visualizes-more-than-numbers/>



# Motivation

## Hackaday 2023

→ <https://hackaday.com/2023/02/23/sailing-on-a-sea-of-seven-segment-displays/>



# Was erzähle ich heute?

- ASCII-Art
  - Was ist das?
  - Programme, Tools etc.
- Eine 7-Segment Display Matrix
  - Wie funktioniert das?
  - 7s-Matrix-Simulator
  - Was kommt jetzt (vielleicht)?

# ASCII-Art

- Kunstrichtung, bei der man mit Buchstaben, Ziffern und Sonderzeichen Piktogramme, Bilder, Videos u.ä. darstellt
- Früher (vor Erfindung des Terminals) gab es auch den Begriff „Typewriter Art“ → Paul Smith

# Beispiele für „einfache“ ASCII-Art

	( <u>  </u> ) (oo)		0	<u>  </u> 0	. <u>  </u> .
	/-----\ /	<u>  </u>	/H\	<u>  </u> \-< ,	{o, o}
*	/             /\-----/\	/o)\	/ \	-( <u>  </u> )/( <u>  </u> )-	/) <u>  </u> )
	<u>  </u> <u>  </u>	\(o/ --			-"-"

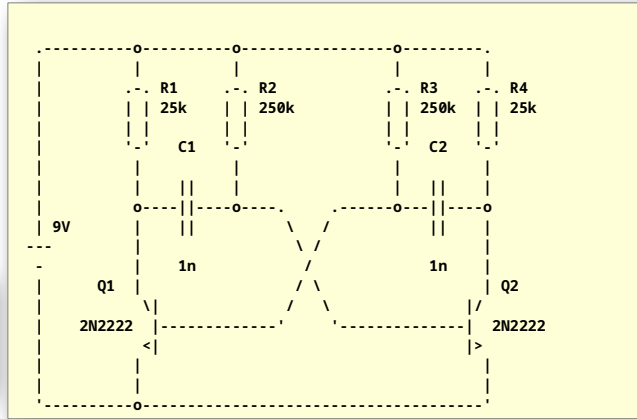
$$\frac{\binom{m}{k} \binom{m-k}{l} \binom{m-k-l}{r}}{\binom{m}{k} \binom{m-k}{l} \binom{m-k-l}{r}} = \frac{\binom{m}{k} \binom{m-k}{l} \binom{m-k-l}{r}}{\binom{m}{k} \binom{m-k}{l} \binom{m-k-l}{r}}$$

```

      \|||/
      (o o)

~oo0~~( )~~~~~
|
| Einfaches
| Beispiel für
| ASCII-Art
|
| ~~~~~oo0~~~
|
|  |__|__|
|  ||  ||
| oo0 0oo

```



```

      _ _
      ,,:\\:~\
    ,/' '\/'\ /
  _\, : '.,-'.'-':.
-./"'" : : :\\,
  ::. ,:____;_;-
  : " ( .-'*o*', );
    \.. \ \_--' '\ /
      \:._...- _.'
      ,,: .
    /"'" | | \
  ::. ) : :
  |" ( \ |
  :.( _ , : ;
    \'\`-' _/ /
      \
      ... , _'
      |,| : | | |
      |\' | | |
      |,| | | |
    /,--. ; | |'...-.
  /;'"' ; '...- ) )
  \:._(____ ) ))'
      SSt`-'-'-'

```



# „Aufwendige“ ASCII-Art

[illegible]

Bild: <https://aa-project.sourceforge.net/gallery/alfons.png>, (C) 2001 by Jan Hubicka



# Programme, Tools ...

...zur Erstellung/Konvertieren von Bildern oder Video-Sequenzen in ASCII-Art basieren meist auf:

- aalib → <https://aa-project.sourceforge.net/aalib/>
- libcaca → <http://caca.zoy.org/wiki/libcaca>

Die zugrundeliegenden Algorithmen versuchen meist Bildbereiche des Originals durch äquivalente ASCII-Zeichen zu ersetzen (Kontur, visuelle Helligkeit/Kontrast, Farben etc.).

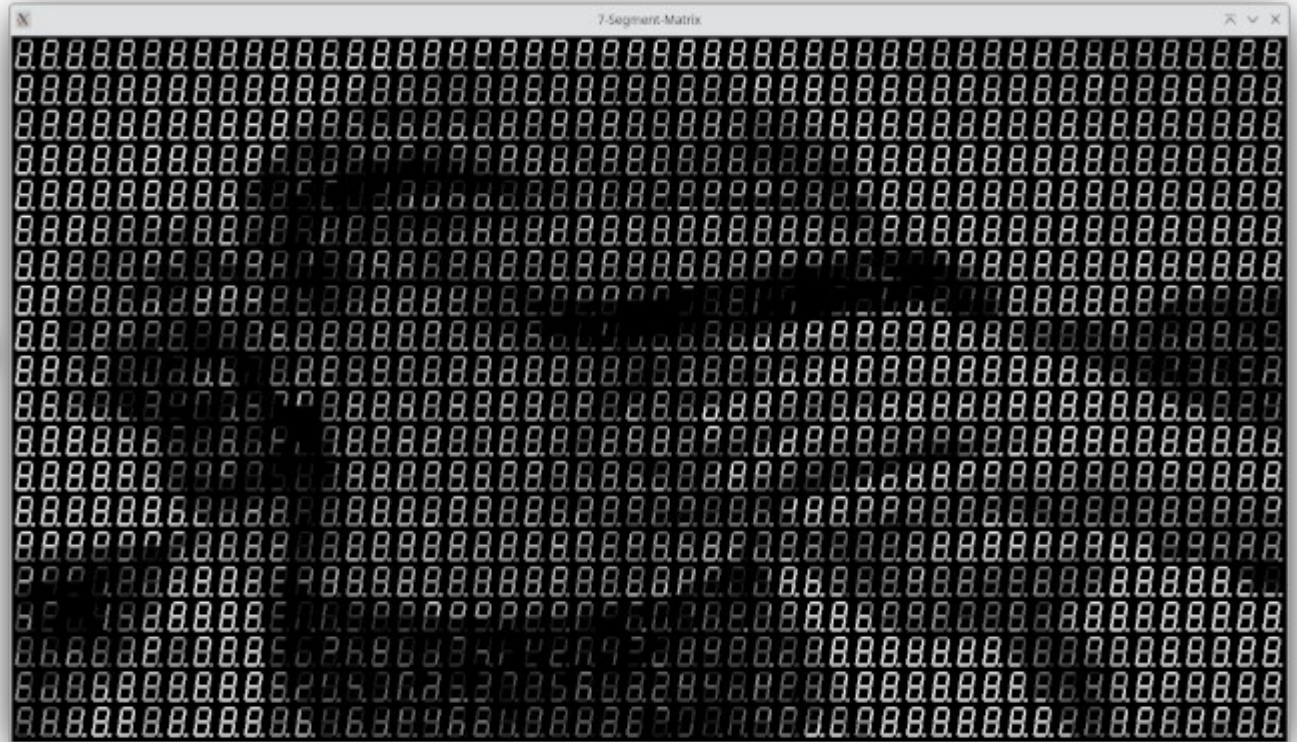
# Programme, Tools (Beispiele)

- Bilder in ASCII konvertieren:
  - aview (nur .pnm-Bilder als Input)
  - jp2a (für .jpg, .png)
- Videos als ASCII anzeigen:
  - mplayer -vo aa video.avi
- Webcam-Output (o.ä.) als ASCII-Art-Stream:
  - Hasciicam

# ASCII-Art...

...aber das soll eigentlich nicht Thema dieses Vortrages sein!

# 7-Segment Display Matrix



# 7-Segment Display Matrix

Also:

- Einzelne 7-Segment-Anzeigen (7s-Digits) sind in einer Matrix (7s-Matrix) angeordnet
- Die einzelnen Segmente der 7s-Digits der 7s-Matrix sind in ein xy-Koordinatensystem eingeteilt
- In diesem Koordinatensystem ist jeder Punkt einzeln ansteuerbar
- Auf dieser 7s-Matrix soll gezeichnet oder Bilder dargestellt werden können

# 7-Segment Display Matrix

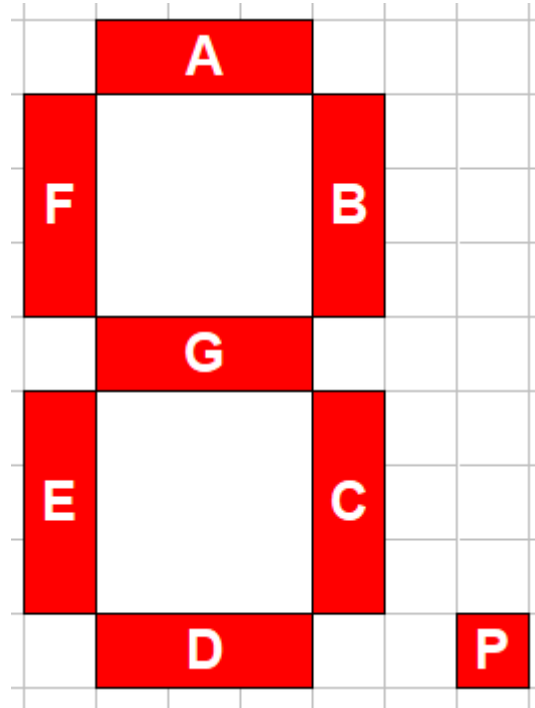
Zu klärende Frage:

- Wie teilt man die einzelnen Digit-Segmente sinnvoll in ein xy-Koordinatensystem ein?

Lösungsvarianten:

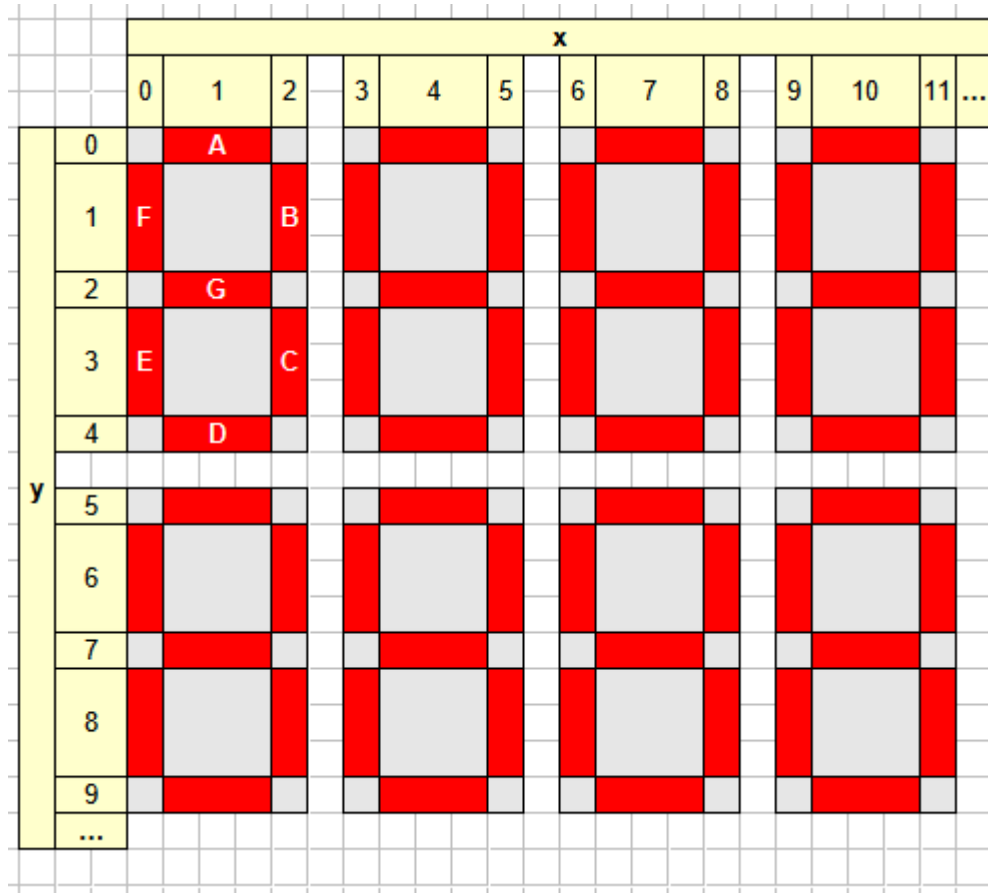
- Variante 1: „symmetrische“ Verteilung der Segmente
- Variante 2: „gepackte/komprimierte“ Verteilung der Segmente

# 7-Segment-Digit

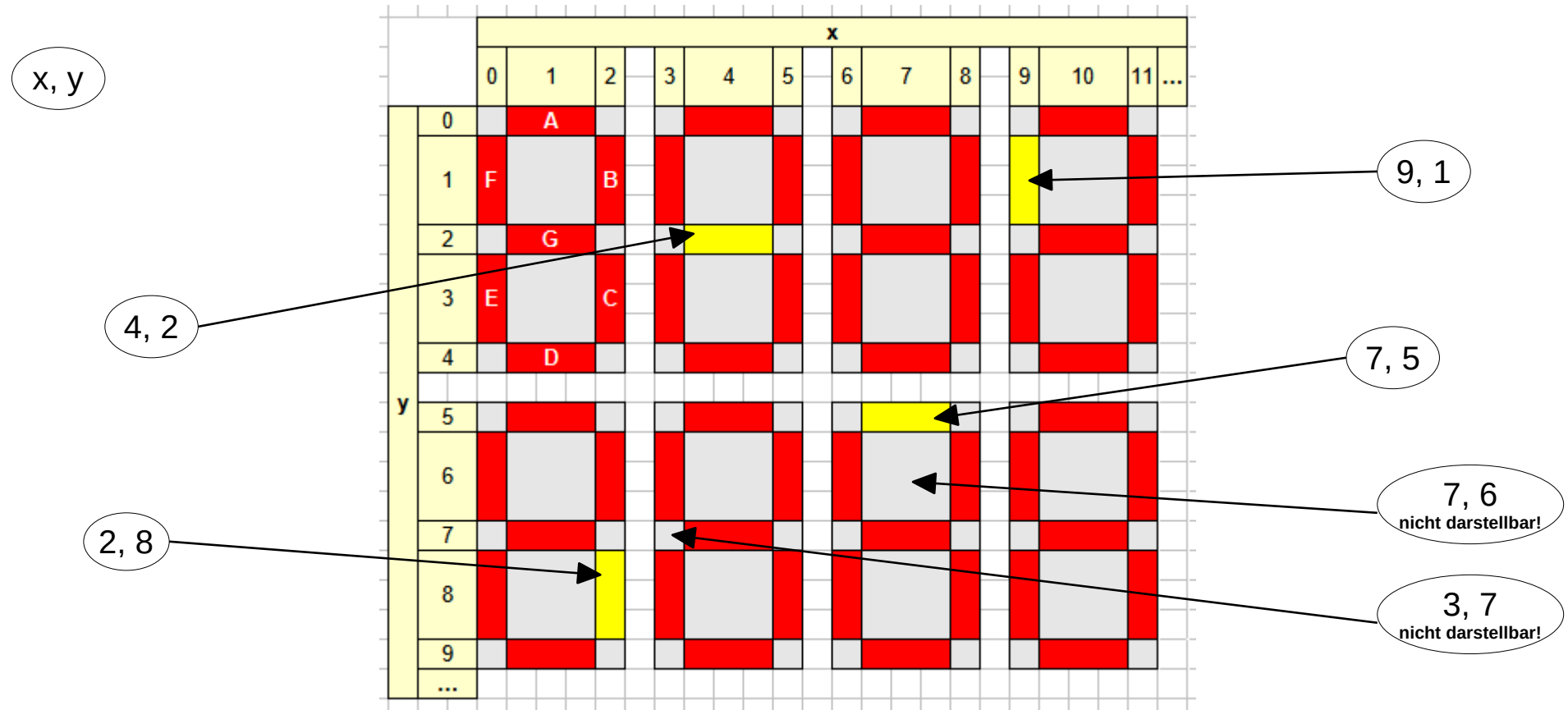




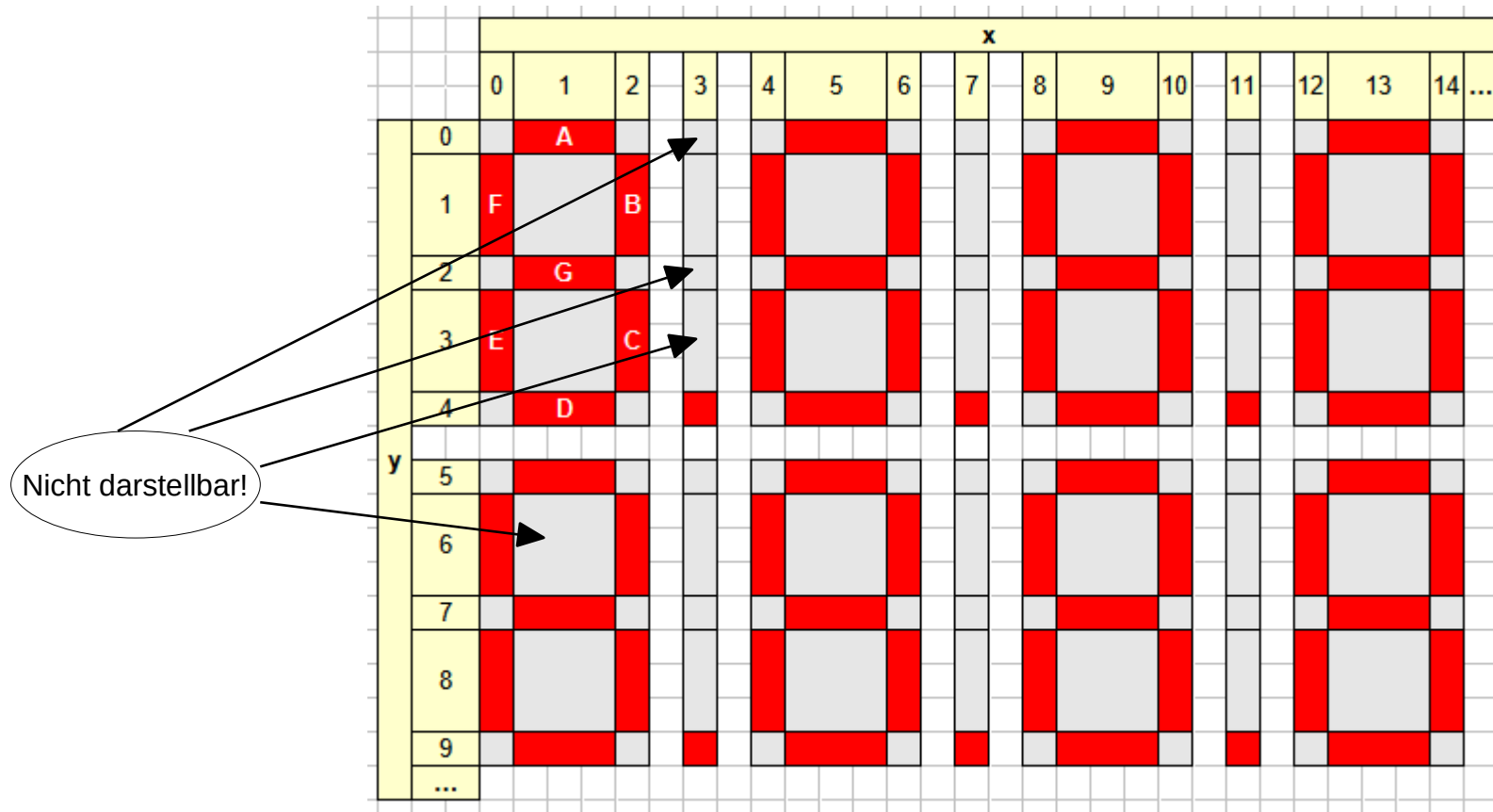
# V1: Sym. Verteilung der Segmente



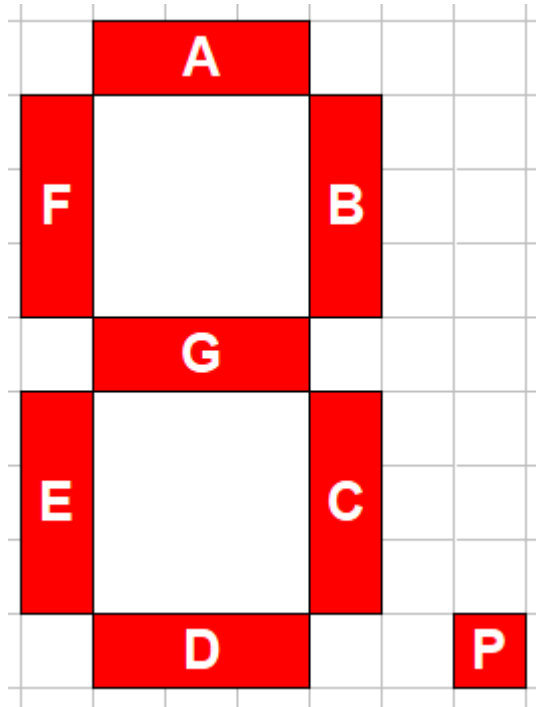
# V1: Sym. Verteilung der Segmente



# V1: Sym. Verteilung der Segmente



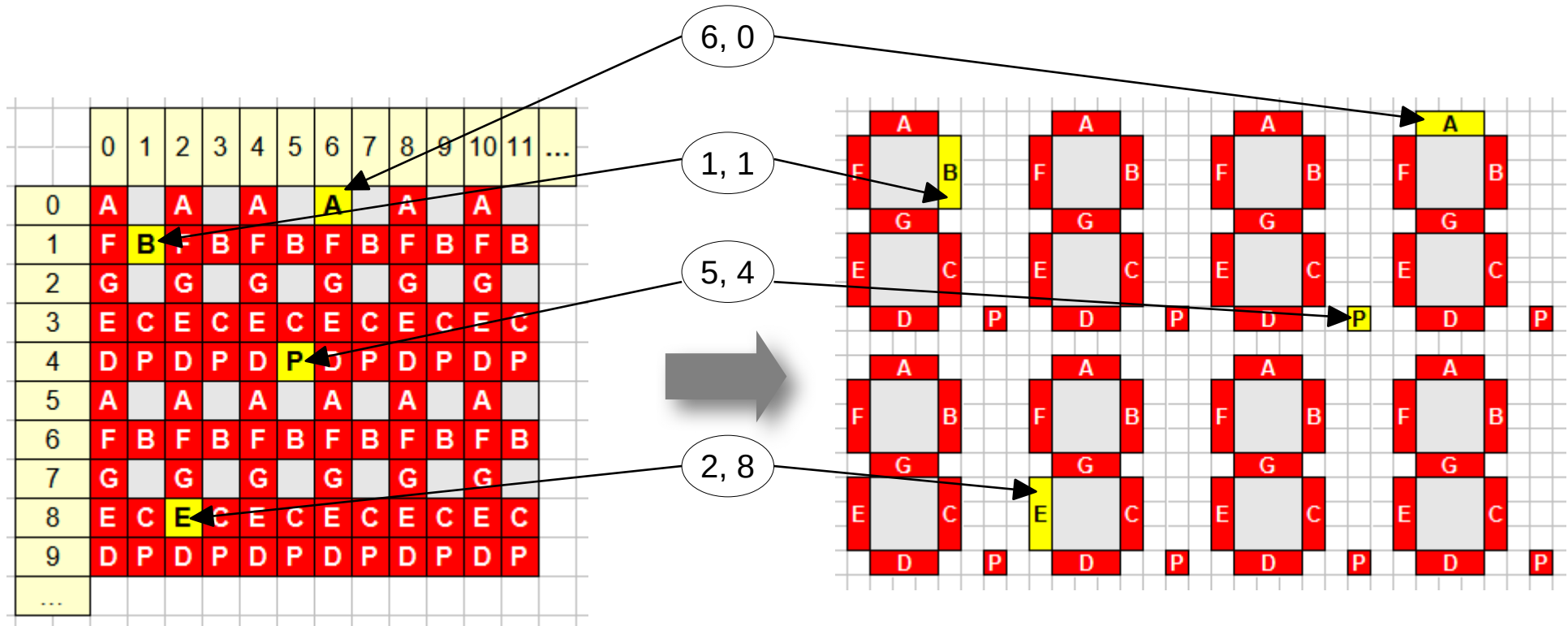
# V2: gepackte Verteilung der Segm.



*...in Gedanken...*

	0	1
0	A	
1	F	B
2	G	
3	E	C
4	D	P

# V2: gepackte Verteilung der Segm.



# 7s-Matrix-Simulator...

Da es relativ aufwändig wäre, die Hardware „einfach mal so“ aufzubauen, wurde zuerst ein Hardware-Simulator in Software implementiert:

- ...um überhaupt mal anzufangen und die Machbarkeit zu prüfen
- ...um diverse Algorithmen ausprobieren
- ...um den Spieltrieb zu befriedigen

# 7s-Matrix-Simulator

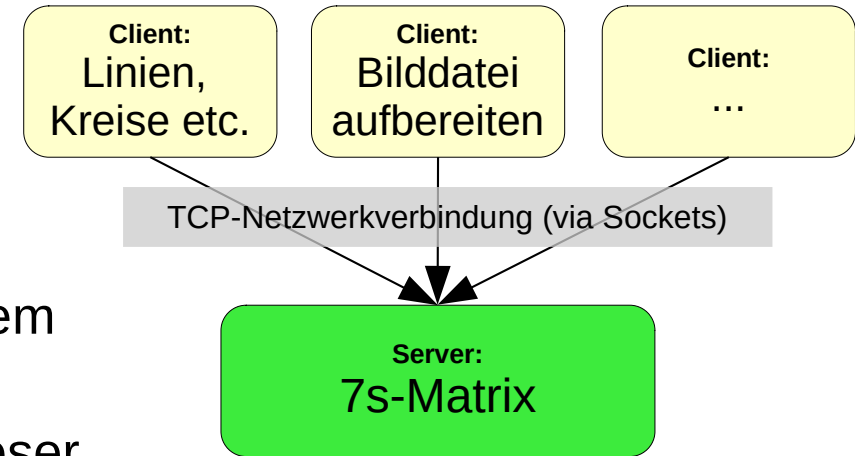
- Client/Server-Architektur

- Client

- generiert/konvertiert Bilder
    - sendet das Ergebnis zum Server

- Server

- generiert initial die 7s-Matrix in einem Grafikfenster
    - stellt empfangene Bilddaten auf dieser dar
    - ...ist der eigentliche Simulator und soll später durch Hardware ersetzt werden



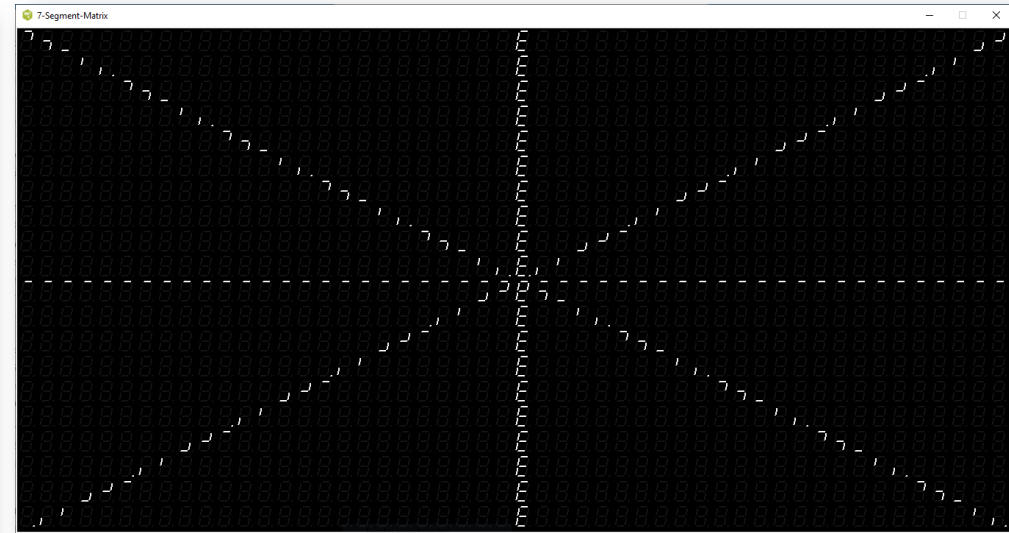
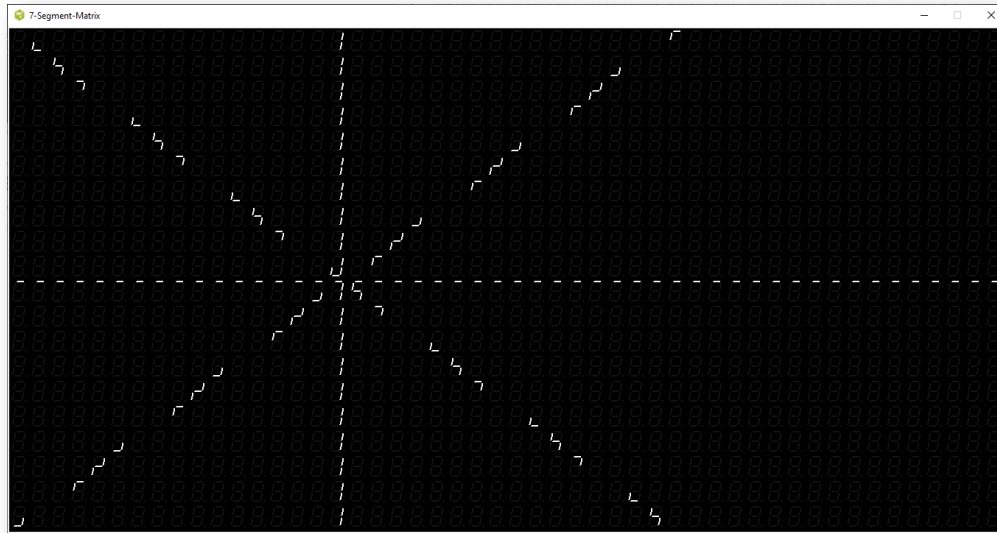


# 7s-Matrix-Simulator

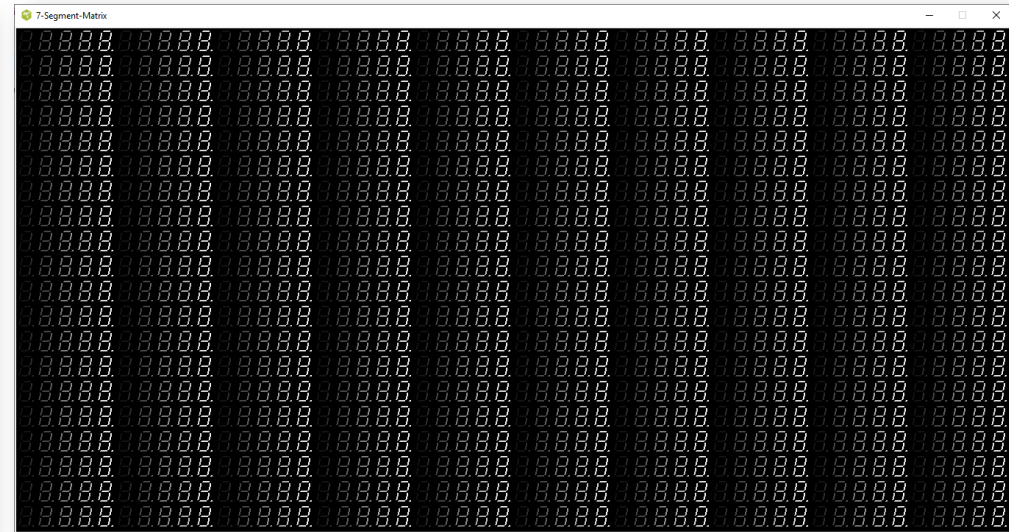
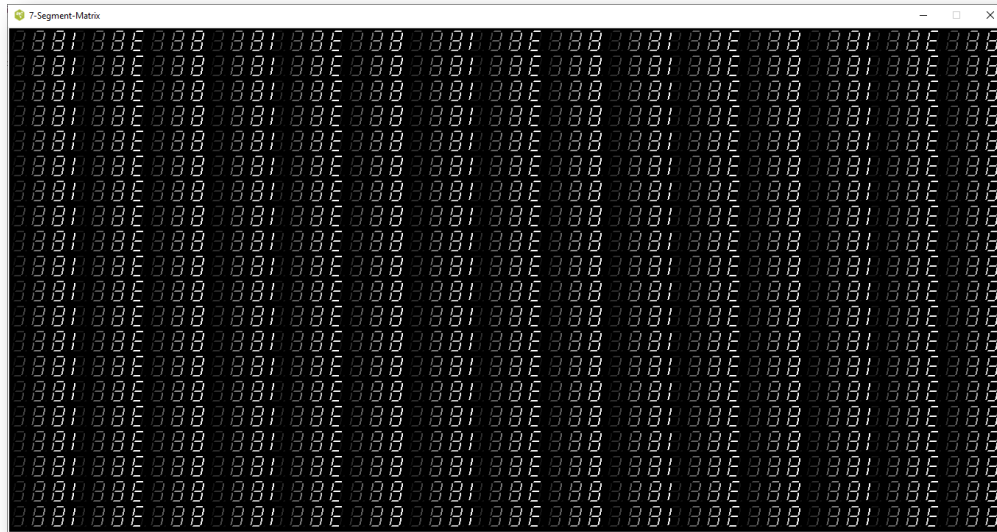
## Evaluationsstufen:

- Erste Version in Tcl/Tk geschrieben...
- Wegen relativ geringer Performance → Portierung nach Python:
  - Kommunikation zwischen Client/Server via UNIX-FIFO-File
  - Weil Pythons tkinter auch nicht umwerfend schneller → Experimente mit pygame (später mehr dazu...)

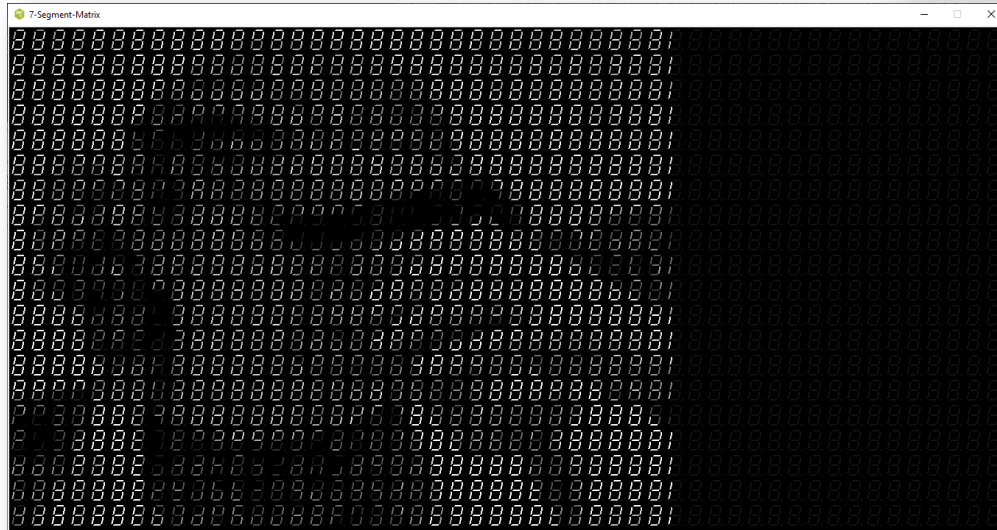
# 7s-Matrix-Simulator (V1 vs. V2)



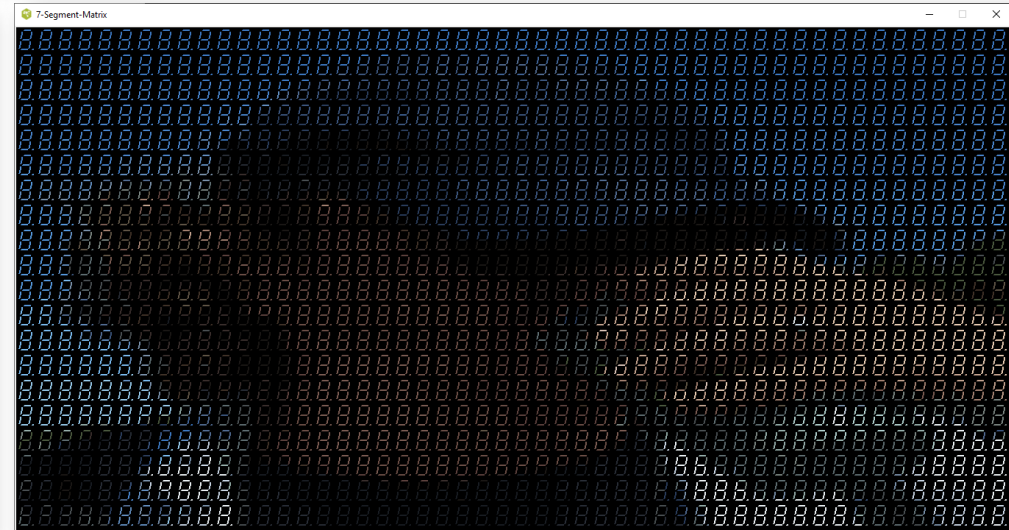
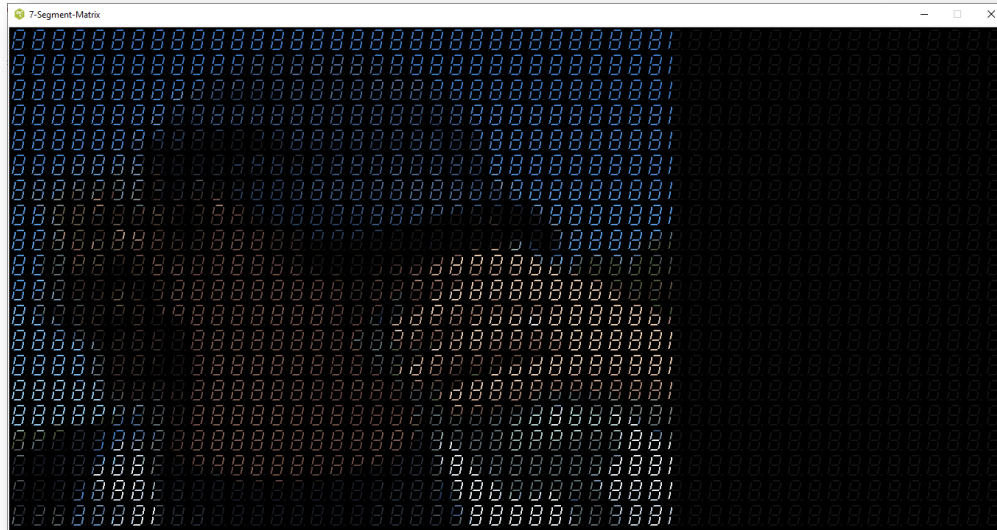
# 7s-Matrix-Simulator (V1 vs. V2)



# 7s-Matrix-Simulator (V1 vs. V2)



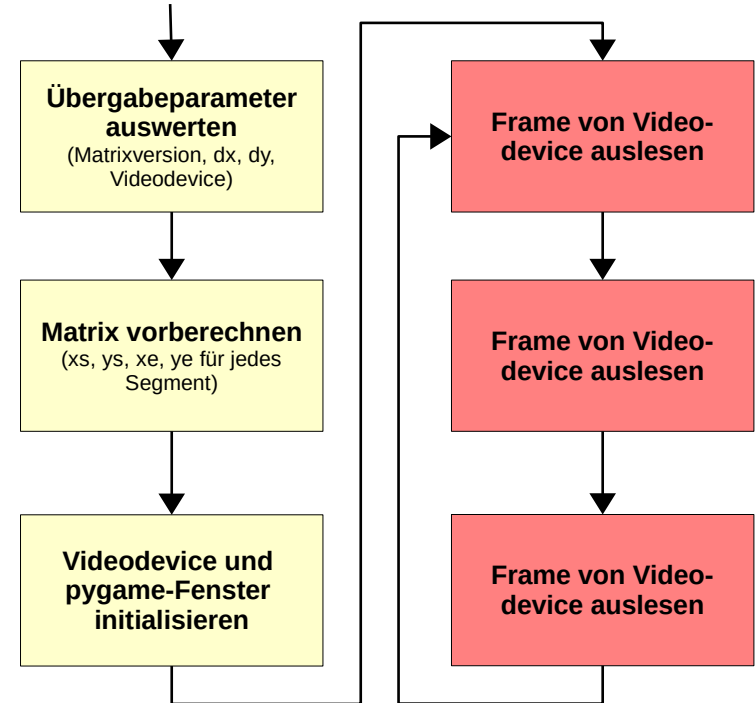
# 7s-Matrix-Simulator (V1 vs. V2)



# 7s-Cam...

## 7s-Cam:

- ...aus reinem Spieltrieb entstanden
- Ziel → testen, ob ein Live-Cam-Stream darstellbar ist
- Deshalb auch (erst mal) keine Client/Server-Architektur



# Vergleich Varianten (Matrix-Koord.)

	Variante 1	Variante 2
Abbildbare „Pixel-Fläche“ auf einem Digit	<ul style="list-style-type: none"><li>• dx=3</li><li>• dy=5</li></ul>	<ul style="list-style-type: none"><li>• dx=2</li><li>• dy=5</li></ul>
Nicht anzeigbare Pixel pro Digit	8 (7 Segmente bei 15 Pixel; Digit-Punkt wird nicht verwendet)	2 (8 Segmente bei 10 Pixel; Digit-Punkt wird verwendet)
Pro	<ul style="list-style-type: none"><li>• Symmetrisches Abbild</li><li>• Digit-Verbrauch geringer (z.B. 48x50 Pixel: 160 Digits)</li></ul>	<ul style="list-style-type: none"><li>• Guter Ausnutzungsgrad</li></ul>
Contra	<ul style="list-style-type: none"><li>• Schlechter Ausnutzungsgrad</li></ul>	<ul style="list-style-type: none"><li>• Unsymmetrisches Abbild</li><li>• Höherer Digit-Verbrauch (z.B. 48x50 Pixel: 240 Digits)</li></ul>



# 7s-Matrix in Hardware...?

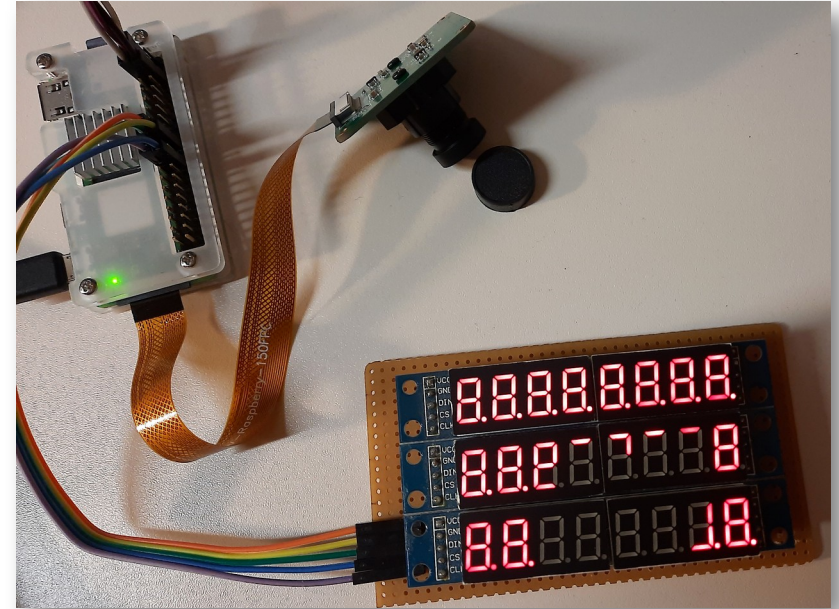
- Dimension 48x50 Pixel (bei V2) entsprechen z.B.:
  - $24 \times 10 = 240$  Einzel-Digits oder
  - $6 \times 10 = 60$  4er-Digits
- Zu lösende Probleme:
  - Ansteuerung Segmente:
    - jedes Segment einzeln (für Bsp. 1920 Segm.!!)
    - möglichst in „Graustufen“ (via PWM)
  - Stromverbrauch: für 48x50 Pixel → ca. 11A (Worst Case)!
  - Mechanischer Aufbau ...



# Ansteuerung via MAX7219

## MAX7219:

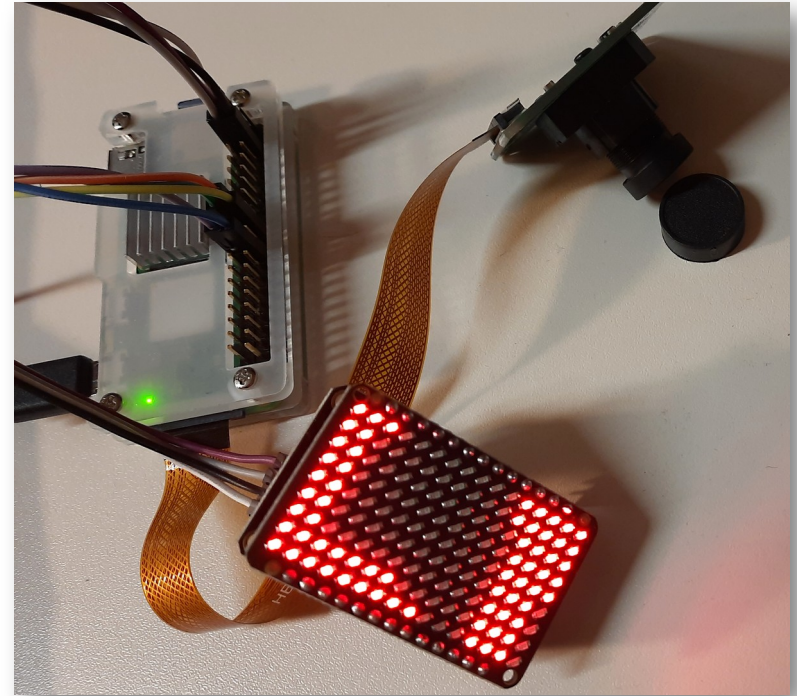
- Treiber zur Ansteuerung von 64 LEDs; Kommunikation via SPI
- 240 Digits → 30 ICs 🖱️
- preisgünstig & gut verfügbar 👍
- einfache Verschaltung 👍
- kaskadierbar 👍
- nur „schwarz/weiß“ möglich 🖱️



# Ansteuerung via IS31FL3731






## IS31FL3731:

- Treiber zur Ansteuerung von 144 LEDs; pro LED 255 PWM-Stufen; Kommunikation I<sup>2</sup>C
- 240 Digits → ca. 14 ICs 👍
- aufwendige Verschaltung (Charlieplexing) 👎
- nicht kaskadierbar (4 I<sup>2</sup>C-Addr.) 👎
- „Graustufen“ möglich 👍



# Ansteuerung via Mikrocontroller

Eigenentwickelte Mikrocontroller-Schaltung/Firmware:

- Aufwendige Modulentwicklung (Hard-/Software) 
- Zusatzbeschaltung notw. (z.B. Strombegrenzung/Treiber für LED) 
- 240 Digits → 10 Module realistisch 
- Funktionsweise via Firmware optimierbar 
- begrenzte Anzahl von GPIOs bei gängigen MCs 

# Weiterführende Informationen

- <https://github.com/boerge42/Tcl-Magie>
- [https://github.com/boerge42/7s\\_matrix/](https://github.com/boerge42/7s_matrix/)



# Fragen?

...ansonsten Danke & Ende!