Programmaufbau Random generator

Projektname:
RandomNumberX

Technologie:
WPF, C#, .NetCore
oder UWP ?

1
Anzahl Zahlen
Verteilbereich ?
Auswahl der Generatoren

Start

Aufbau GUI

1
Lese Programm-parameter

Erzeuge Zufalls-zahl

Verarbeite Zahl

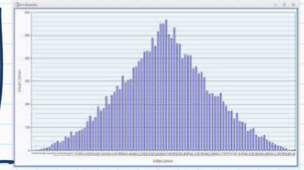
Ende ?
nein
ja

Stop

Verarbeite neue Zufallszahl:

Zahl speichern

Zahl grafisch darstellen

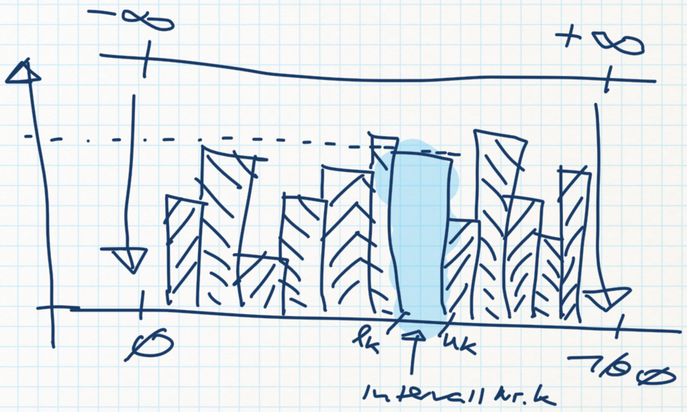


Statistik aktualisieren

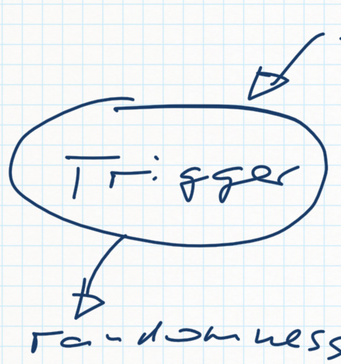
kleinster

→ Seite 4

„Binning"



What is randomness? → something we cannot predict!



↳ CPU clock or seed of System.Random!
→ Pseudo randomness

applications:
generation of test data
(security if not a concern!)

RNGCryptoServiceProvider
↳ uses operating system entropy to generate initial seed

lowest Δ → $\log_2$ → seed
(Windows, Linux)

thermal temp, mouse event delta
mouse event timer

	1004	1012	1024	1037	1039	1045
Δ_1	...	8	12	7	5	11
Δ_2	.....	4	11	4	6	
Δ_3	.....	7	7	(2) lowest delta		

Naiver Ansatz:

RandomNumberGenerator

<<Interface>>
IRandomNumberGenerator

void GetBytes(byte[] buffer)

long Next(long min, long max)

RNG-CryptoServiceProv

SecureRandomness

secureProvider

"has-a"

Entwurfsmuster „Strategie":

abstrakt

RNG

RandomBehavior randomBehavior

long PerformRnd(long min, long max)

void SetRandomBehavior(RandomBehavior rb)

<<Interface>>

RandomBehavior

long Rnd(long min, long max)

implements

System.Random

PseudoRandomness

public long Rnd(long min, long max)
{
    return base.Next(min, max);
}

SampleRNG

public SampleRNG() {
    SetRandomBehavior(new PseudoRandomness());
}

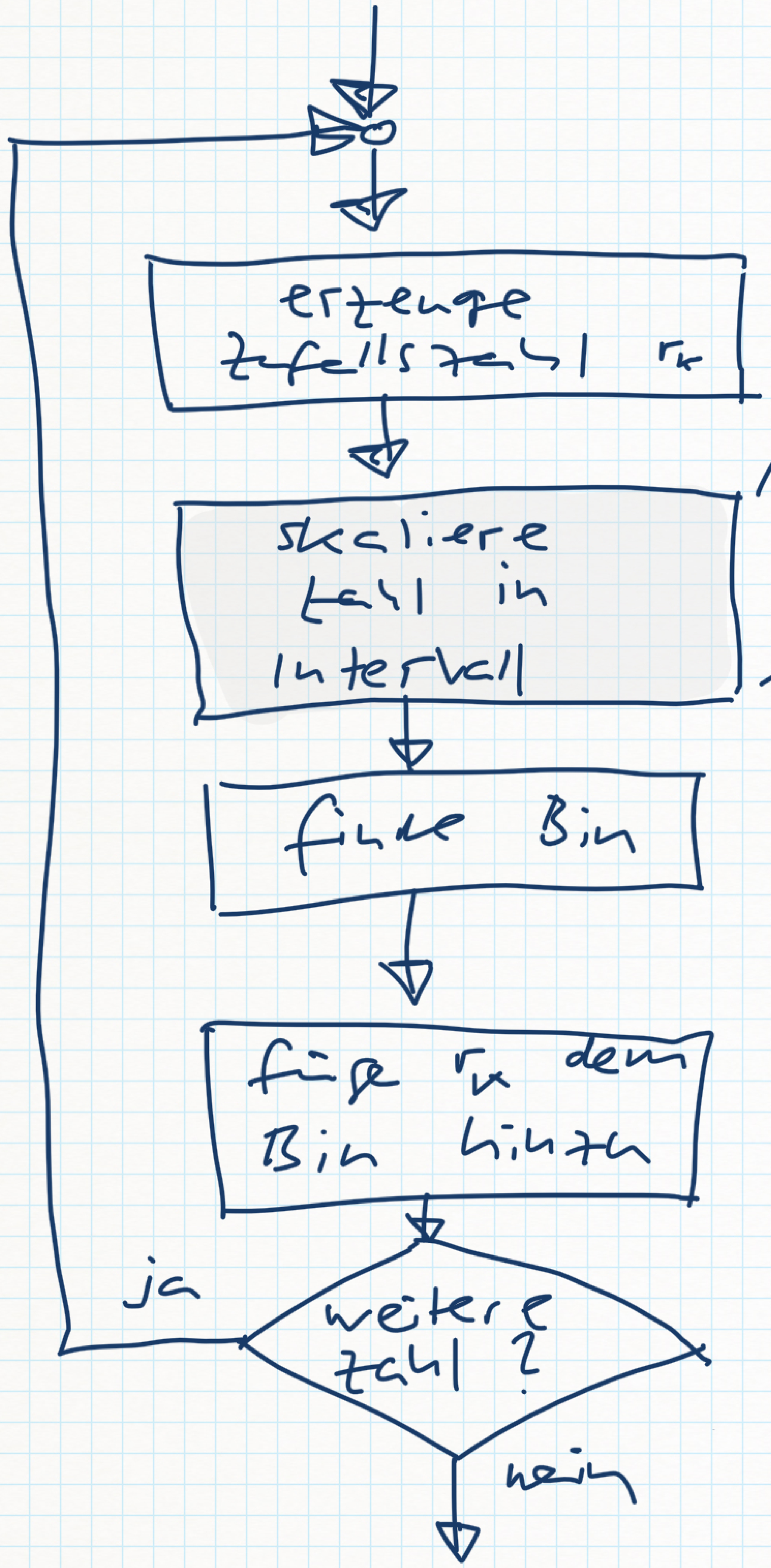
public long PerformRnd(long min, long max){
    return randomBehavior.Rnd(min, max);
}

umbenennen:

RandomBehavior → Randomness

PerformRnd → GetRandomNumber

Einsortieren der Zufallszahlen in Bins:



$$r_d = \left(\frac{r_e - l_{min}}{l_{max} - l_{min}} + \frac{d_{min}}{d_{max} - d_{min}}\right) \cdot (d_{max} - d_{min})$$

$$= \left(0{,}65 + \frac{0{,}5}{1{,}25}\right) \cdot 1{,}25 = (0{,}65 + 0{,}4) \cdot 1{,}25 = 1{,}3125$$

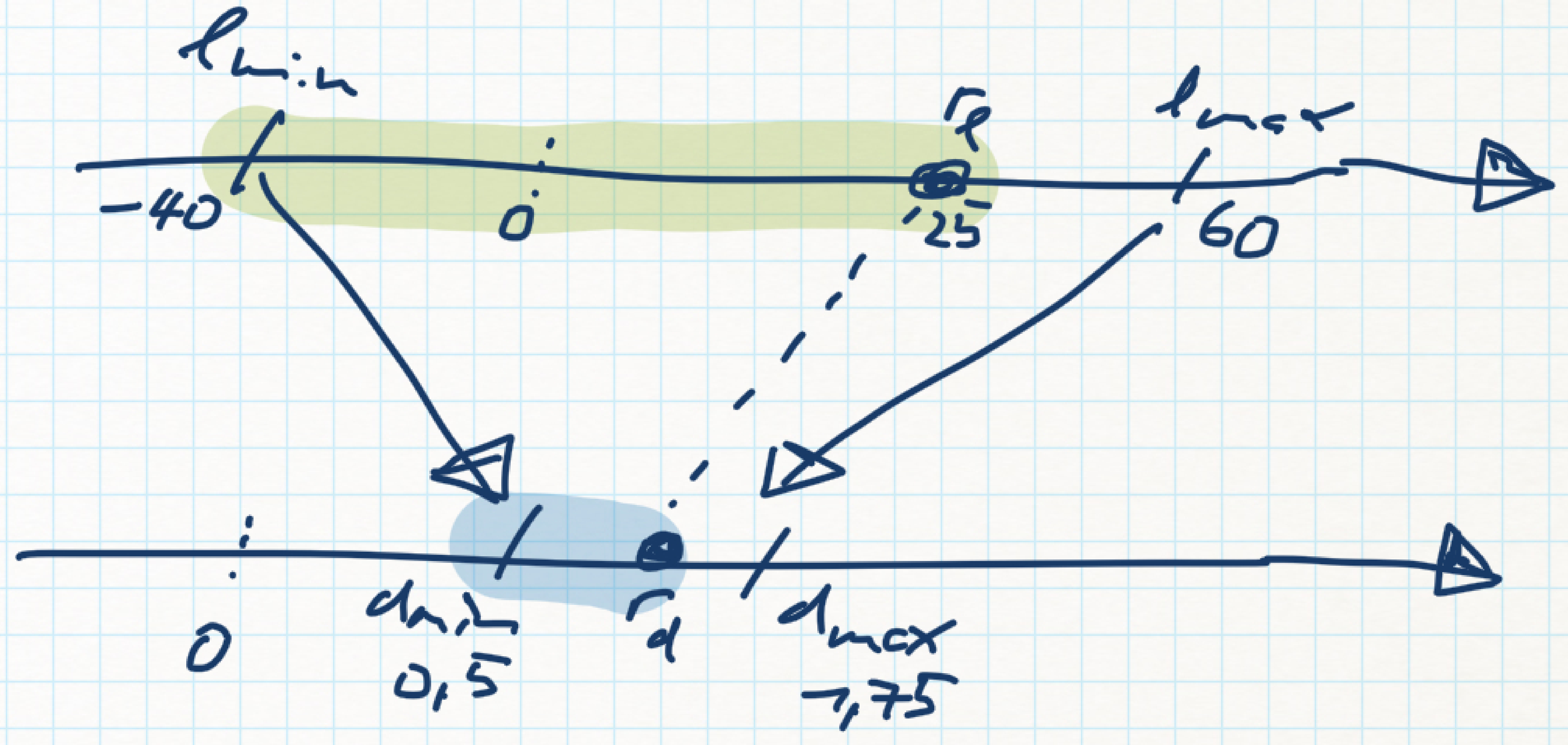
```
double scale( long l, long lmin, long lmax, double dmin, dmax)
{
    double r = ((l-lmin)/(lmax-lmin))·(dmax-dmin) + dmin;
    return r;
}
```

$$\delta(r_d) \stackrel{!}{=} \lambda(r_e)$$

$$\delta(r_d) = \frac{1}{d_{max} - d_{min}} \cdot r_d - \frac{d_{min}}{d_{max} - d_{min}}$$

$$\Rightarrow \lambda(r_e) = \frac{1}{l_{max} - l_{min}} \cdot r_e - \frac{l_{min}}{l_{max} - l_{min}}$$

$$\Leftrightarrow \lambda(r_e) = \frac{r_e - l_{min}}{l_{max} - l_{min}} = \frac{25 - (-40)}{100} = \frac{65}{100}$$



$$\lambda(r_e) = a \cdot r_e + b$$

$$\lambda(l_{min}) = 0$$

$$\Leftrightarrow \quad a \cdot l_{min} + b = 0 \quad (1)$$

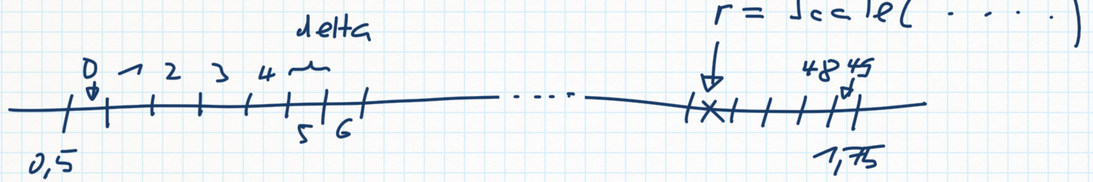
$$\lambda(l_{max}) = 1$$

$$\Leftrightarrow \quad a \cdot l_{max} + b = 1 \quad (2)$$

$$(2) - (1): \quad a(l_{max} - l_{min}) = 1$$

$$\Leftrightarrow \quad a = \frac{1}{l_{max} - l_{min}}$$

$$(3) \text{ in } (1): \Rightarrow b = -\frac{1}{l_{max} - l_{min}} \cdot l_{min} \quad (4)$$



$$delta = \frac{1{,}75 - 0{,}1}{50} = \frac{1{,}65}{50} = 0{,}033$$

num of bins

$$Index \quad i = a \cdot r + b$$

$$0{,}5 + i \cdot delta = r \iff i \cdot delta = r - d_{min}$$

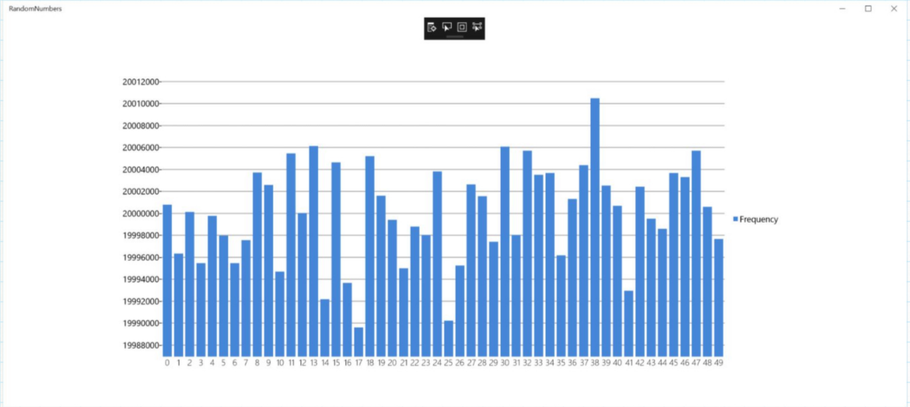
$$\iff i = (r - d_{min}) / delta$$

$$i(d_{min}) = 0 / delta = 0$$

$$i(d_{max}) = (d_{max} - d_{min}) / delta$$

$$(1{,}75 - 0{,}1) / 0{,}033 = 50$$

ungültiger Index! [0,49]



Verteilung der 10^9 Zufallszahlen

Mittelwertschätzer

$$\| \quad m_k = \frac{1}{k} \cdot x_k + \frac{k-1}{k} \cdot m_{k-1} \quad \|$$

Anfang/Ende von Intervall i:

$$[d_{min} + i \cdot delta \;,\; d_{min} + (i+1) \cdot delta)$$

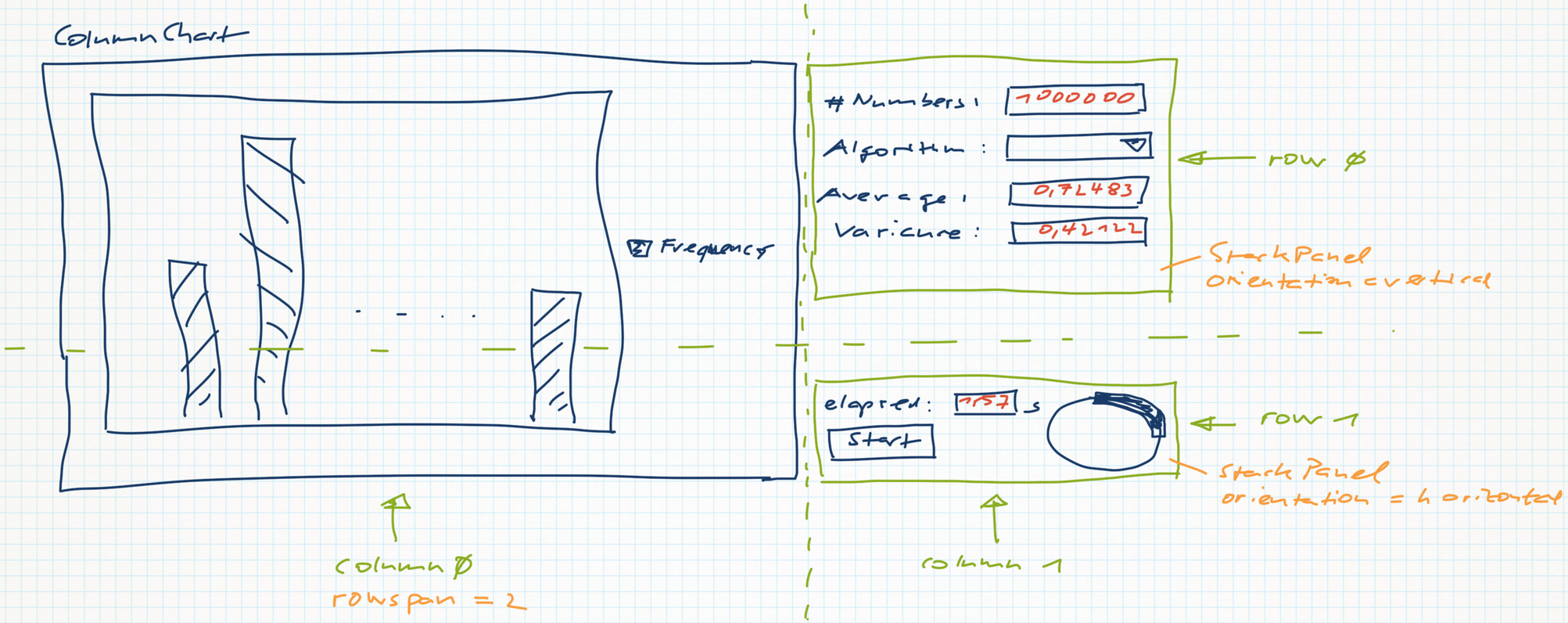
abgeschlossen ≙ untere Grenze dabei!

offen ≙ obere Grenze nicht dabei!

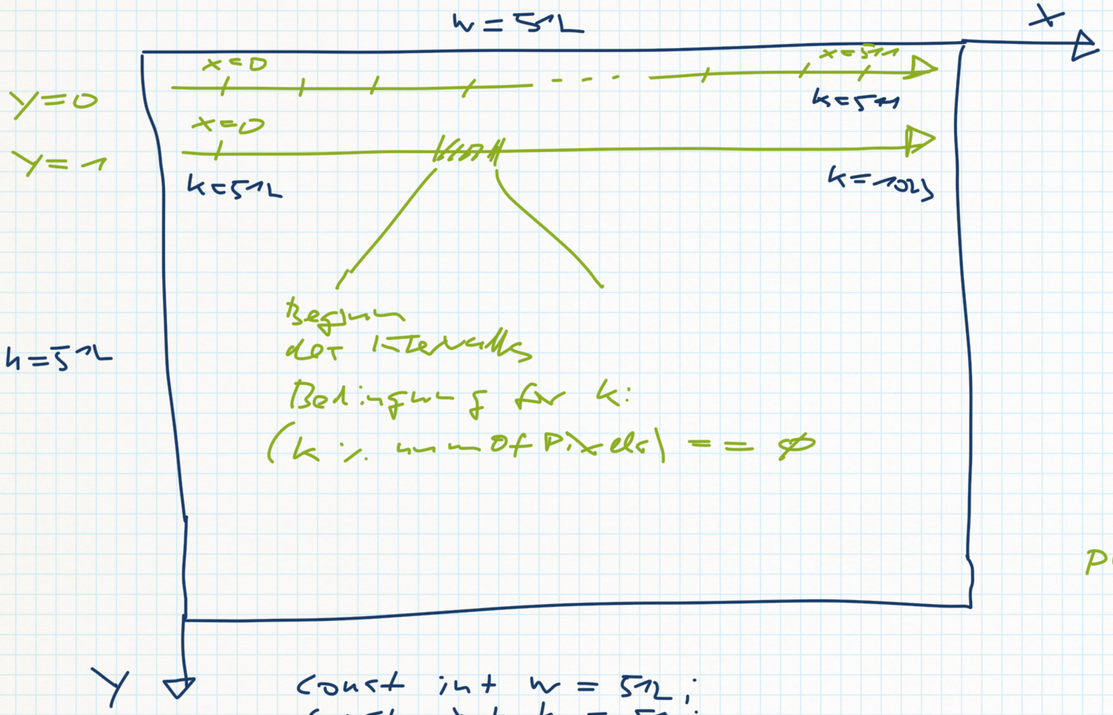
System.Random

* Elemente der Bedienoberfläche:

Balkendiagramm, Textfeld Anzahl Zufallszahlen, Auswahlliste Algorithmus, Anzeige Mittelwert, Anzeige Varianz, Start-/Stopp-Knopf, Fortschrittsanzeige ⟲, Anzeige Laufzeit [Sekunden].



Visualisierung von Regelmäßigkeiten:




```
const int w = 512;
const int h = 512;
int numOfPixels = w*h;
```

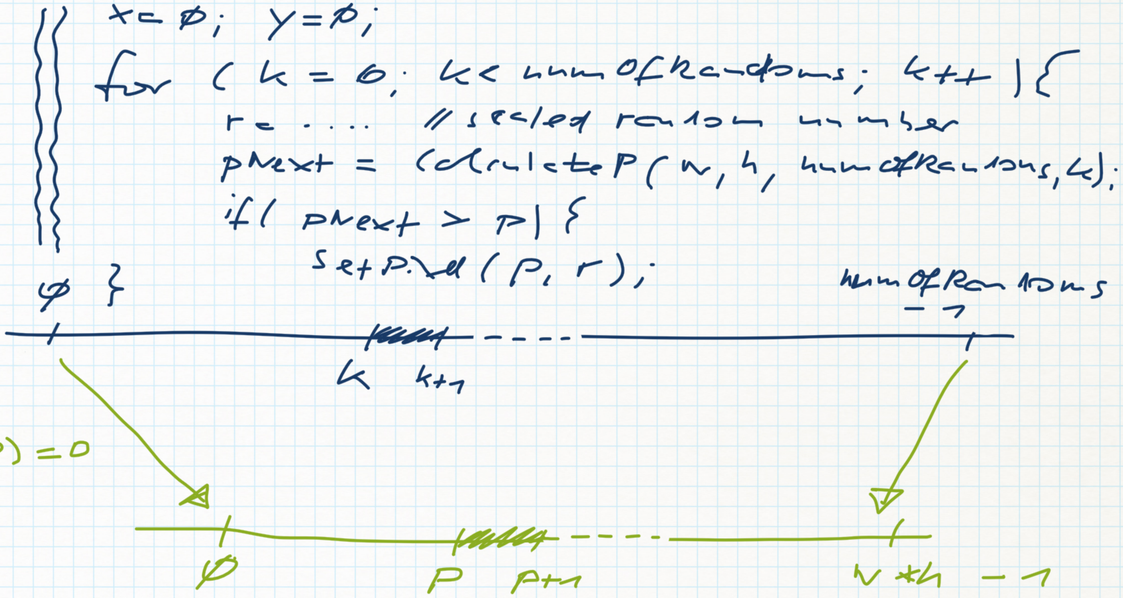
Bsp.: w=8, h=4, numOfPixels = 32

k=0, k=1, ..., k=31, k=32, k=33, ...,

int y = offset / w = 19/8 = 2 ✓
int x = offset − (y*w) = 19 − 2*8 = 3 ✓

offset = 19
offset = 14

int y = offset/w = 14/8 = 1
int x = offset − (y*w) = 14 − (1*8) = 6



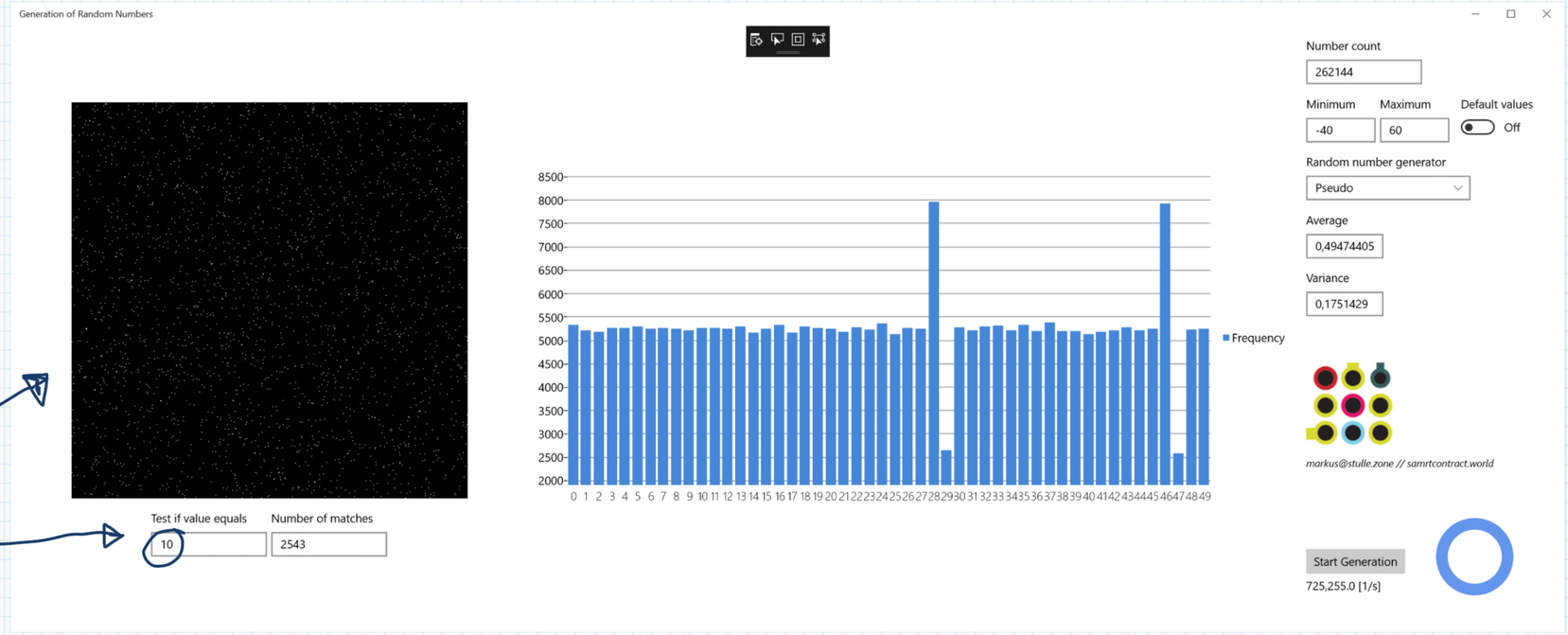
$p(k) = a \cdot k + b$

$p(k=0) = b = \phi$

$p(k = numOfRandoms - 1) = w*h - 1$

$\Rightarrow a = \frac{p(k)}{k} = \frac{w*h-1}{numOfRandoms-1}$

$\Leftrightarrow p(k) = ((w*h-1)/(numOfRandoms-1)) * k$


Generation of Random Numbers
Number count
262144
Minimum
Maximum
Default values
-40
60
Off
Random number generator
Pseudo
Average
0,49474405
Variance
0,1751429
Frequency
markus@stulle.zone // samrtcontract.world
Test if value equals
Number of matches
10
2543
Start Generation
725,255.0 [1/s]
pixel white if number equals