1 Von der einfachen Linie bis zum Plotten eines Liniendiagramms mit automatischer Skalierung

1.1 Grundlagen

In der gegenwärtigen PicoMiteVGA-Version von MMBASIC werden drei Modi zur Darstellung von Text und Grafiken unterstützt:

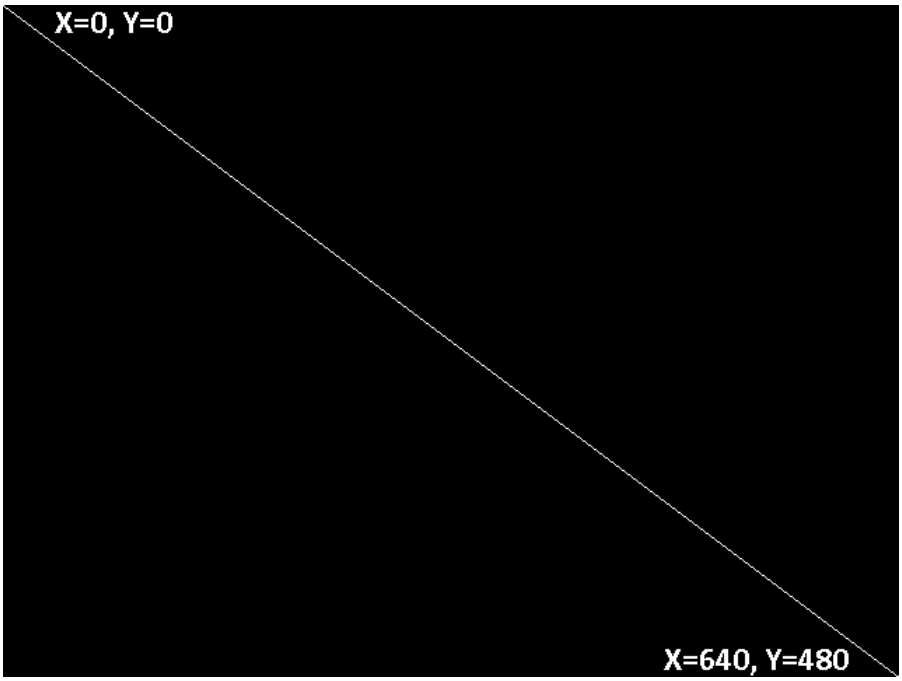
MODE 1 = Monochrome mit einer Auflösung von 640 x 480 (standard bei Startup)
MODE 2 = 16 Farben mit einer Auflösung von 320 x 240
MODE 3 = 16 Farben mit einer Auflösung von 640 x 480 (nur RP2350)

Soll eine Linie gezeichnet werden, so findet sich hierfür der Befehl:

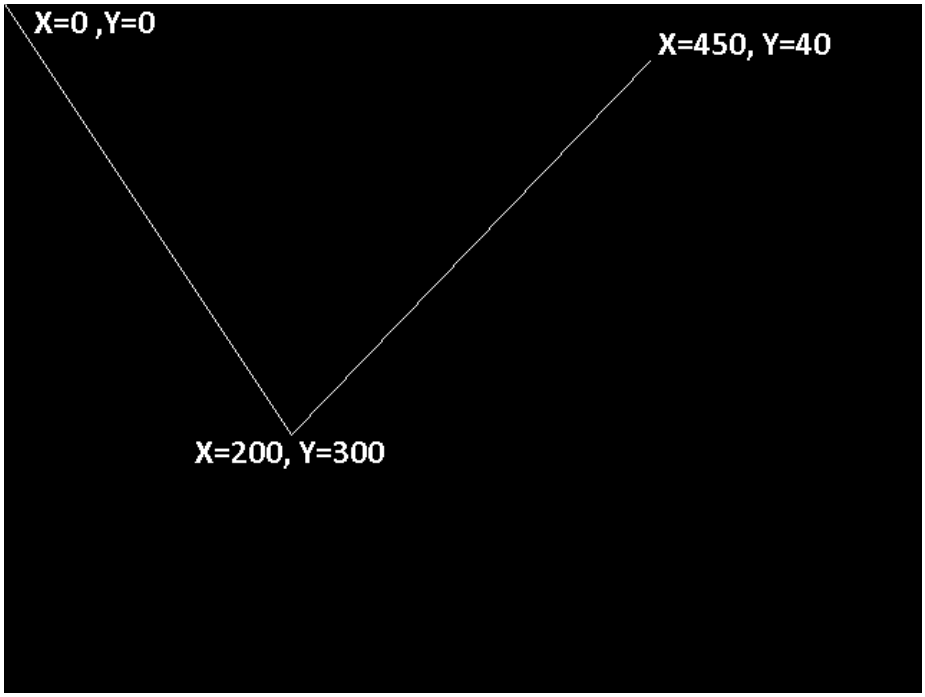
LINE x1, y1, x2, y2 [, LW [,C]]

Der Beginn der Linie stellen die Koordinaten X1 und Y1 dar, das Ende hingegen X2 und Y2. Zusatzparameter stehen in eckigen Klammern:
LW ist die Liniendicke (in Pixeln) und nur für horizontale und vertikale Linien gültig. Dieser Wert ist standardmäßig 1. C bestimmt die Farbe der Linie.

Soll nun eine Linie gezeichnet werden, gilt es zu beachten wie MMBASIC die Koordinaten verarbeitet. Für dieses Beispiel wird MODE 3 mit einer Auflösung von 640 x 480 Pixeln gewählt. In der oberen linken Ecke des Bildschirms befindet sich die Koordinate X=0, Y=0 und in der unteren rechten Ecke des Bildschirms die Koordinate X=640, Y=480.
Wird nun die Linie mit dem Befehl *Line 0,0,640,480* gezeichnet, ergibt sich dieses Bild:



Um nun aber ein Liniendiagramm zu erstellen, benötigt es selbstverständlich mehrere aneinandergereihte Linien, wobei die jeweilige End-Koordinate der vorherigen Linie, die Start-Koordinate der darauffolgenden Linie darstellt:



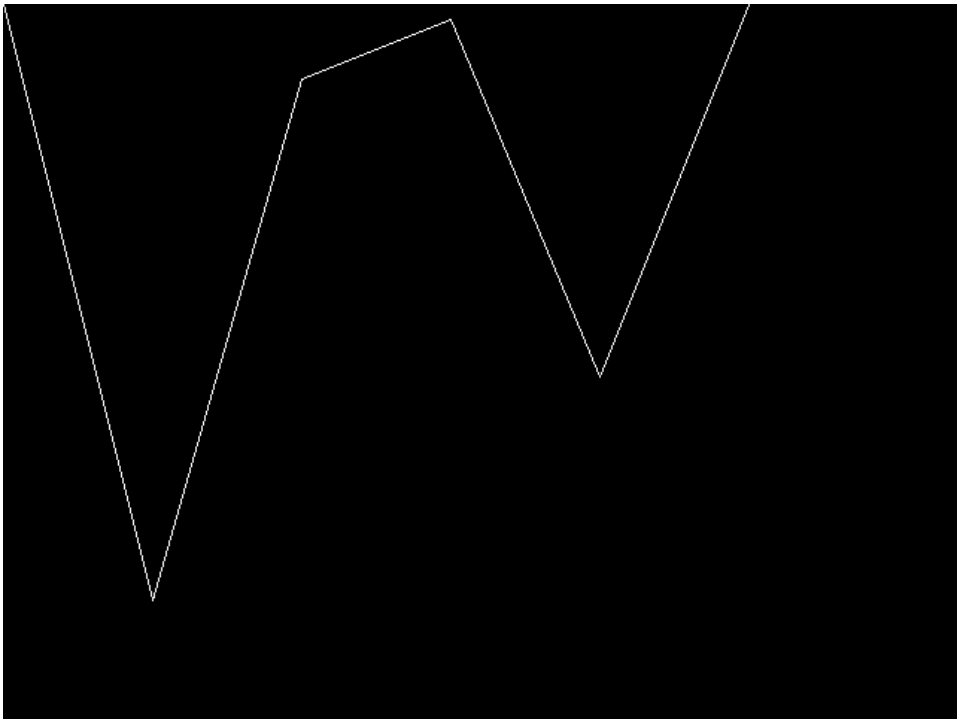
Das kurze Programm hierzu sieht folgendermaßen aus:

```
OPTION EXPLICIT
MODE 3
CLS

Line 0,0,200,300
Line 200,300,450,40
```

Sollen nun jedoch ganze Diagramme gezeichnet werden, ist es unsinnig jedes Mal Linie an Linie zu reihen und dafür auch noch immer wieder den selben Befehl zu verwenden. Daher wird dieser Prozess mit Hilfe des nachfolgenden Programms automatisiert:

```
OPTION EXPLICIT
MODE 3
CLS

Dim y(5)
y(1)=400
y(2)=50
y(3)=10
y(4)=250
y(5)=0
Dim integer x=0,y_old=0,n

For n=1 To 5
  x = x + 100
  Line x-100, y_old, x, y(n)
  y_old=y(n)
Next n
```

Das Programm ergibt dann die gewünschte nahtlose Aneinanderreihung von Linien:



Es wird zunächst ein eindimensionales Array *y(5)* erstellt, welches fünf y-Werte abspeichert. Darauf folgt die Deklaration der verwendeten Variablen und eine For-Schleife, welche mit jedem Durchlauf die Variable *n* um 1 erhöht bis die Schleifenbedingung *n=5* erreicht ist.

Der eigentliche Teil passiert nun in dieser For-Schleife:

Beim jedem Durchlauf der For-Schleife wird *x* um den Wert 100 inkrementiert. Dieser x-Wert stellt die fortschreitende Rechtsverschiebung des späteren Graphen dar (z. B. Zeitachse). Das heißt, jeder abgetragene y-Wert ist um genau 100 Pixel zum jeweils vorherigen auf der x-Achse entfernt. Vor dem Betreten der For-Schleife ist x = 0.

Nun wird diese Betreten und x inkrementiert: x = x + 100, d. h. x = 0 + 100 = 100. Jetzt wird bereits die erste Linie gezeichnet, und zwar mit:

Line x-100, y_old, x, y(n)

Das bedeutet für den ersten Durchlauf (*n=1*) ergeben sich die eingesetzten Werte:

x-100 = 100-100 = 0
y_old = 0
x = 100
y(1) = 400

… und somit:

Line 0,0,100,400

Anschließend wird *y_old* nun in Vorbereitung für den nächsten Durchlauf der alte y-Wert dieses Durchlaufs zugewiesen (*400*); Schließlich soll die darauffolgende Linie nahtlos mit der alten Linie verbunden sein.

Der zweite Durchlauf (*n=2*) der For-Schleife erfolgt und x wird wieder um den Wert 100 inkrementiert:

x = x + 100
x = 100 + 100
x = 200

Line x-100, y_old, x, y(n)

x-100 = 200-100 = 100
y_old = 400
x = 200
y(2) = 50

… daraus ergibt sich schließlich:

Line 100,400,200,50

daraufhin wird *y_old* erneut in Vorbereitung für den nächsten Durchlauf der alte y-Wert dieses Durchlaufs zugewiesen (*50*). Dieser Vorgang wiederholt sich solange bis die Schleifen-Bedingung (*n=5*) erfüllt ist.

Rekapitulation:

Mit diesem Wissen ist es jetzt möglich ein Array von beliebigen y-Werten in eine entsprechende Linienrepräsentation entlang der x-Achse im festen Abstand von je 100 Pixeln zueinander abzutragen.
Es fällt dabei ein Problem deutlich ins Auge: Die abgetragenen y-Werte schlagen nach unten, statt – wie gewohnt – nach oben aus. Das ist der Tatsache geschuldet, dass am oberen Bildschirmrand zu zählen begonnen wird (y=0). Um den Graphen nun korrekt zu zeichnen, muss
der *Line*-Befehl wie folgt abgeändert werden:

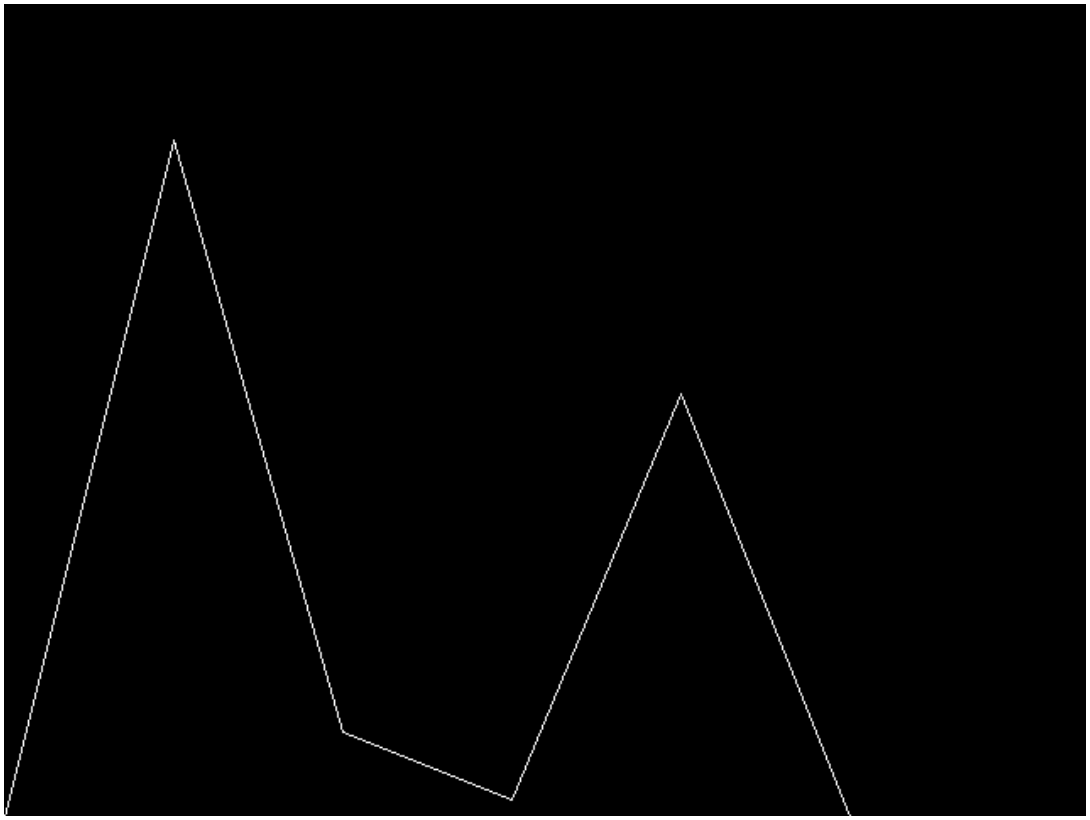
Line x-100, MM.VRes-y_old, x, MM.VRes-y(n)

MM.VRes ist dabei eine interne Variable von MMBASIC und ruft die vertikale Resulution bzw. Auflösung des jeweiligen *MODE* ab. Da der *MODE 3* festgelegt ist, entspricht *MM.VRes* = 480.

Zur besseren Vorstellung kann man natürlich auch 480 in den *Line*-Befehl eintragen:

Line x-100, 480-y_old, x, 480-y(n)

Das Resultat ist der korrekt dargestellte Graph:



Nun sind ein paar Linien schon ganz nett, aber weit entfernt von einem echten Diagramm mit ablesbaren Werten für die jeweiligen Achsen. Im nachfolgenden Teil, soll dem Abhilfe geschaffen werden.

1.2 Liniendiagramm mit automatischer Skalierung am Beispiel eines Geigerzählers

Nach der Einführung des vorherigen Kapitels soll es jetzt um ein anwendungsbezogenes Beispiel gehen, das den Anforderungen an eine automatische Skalen- sowie Graphenanpassung entsprechend der Messwerte gerecht wird. Hierzu soll eine Visualisierung programmiert werden, welche es erlaubt, den von einem Geigerzähler gemessen radioaktiven Zerfall grafisch auszuwerten und in eine Äquivalentdosis anzugeben.

Vorbetrachtung:

Ein Geigerzähler ist ein Detektor für ionisierende Strahlung, wie beispielsweise Alpha-, Beta-, Gamma-, Photonen- und Röntgenstrahlung. Zerfällt ein radioaktives Nuklid wird das Geiger-Müller-Zählrohr für diesen Moment leitend: ein Strom fließt für diesen kurzen Moment. Die Zerfalls-Ereignisse werden durch einen Mikrocontroller gezählt. Für den ersten Test dieses Programms wird jedoch kein Geigerzähler benötigt, stattdessen wird das Verhalten (zunächst) durch eine Zufallsfunktion simuliert, schließlich soll es in diesem Abschnitt weniger um die Hardware-Ebene gehen, sondern um die grafische Auswertung. Für Interessierte erfolgt jedoch im Kapitel X auch die Hardwarekomponente inkl. Schaltplan für den Nachbau.

1. Erzeugen (pseudo-) zufälliger Daten, Umrechnung in Äquivalenzdosis und Ausgabe

Ziel ist es zunächst einmal überhaupt Daten zu generieren, die angezeigt werden können. Da die manuelle Eingabe und das Füllen eines y-Werte-Arrays für hunderte Werte müßig wäre, übernimmt das die eingebaute Random-Funktion von MMBASIC. Zudem ist es nötig, in einem festen Zeitintervall präzise zu stoppen und die jeweiligen radioaktiven Zerfälle zu speichern. Es ist üblich, dass man die radioaktiven Zerfälle pro Minute (CPM) oder gar pro Sekunde (CPS) angibt. Diese Counts Per Minute (resp. Counts Per Second), werden dann mit Anbruch der neuen Minute (resp. Sekunde) auf 0 gesetzt und das Zählen beginnt erneut für *diesen* Zeitintervall. Anschließend werden die gezählten Zerfälle in eine Äquivalenzdosis umgerechnet, in diesem Beispiel in µSv/h.

Das Programm hierzu sieht wie folgt aus:

```
Option EXPLICIT
MODE 3
CLS

SetTick 1000, setProgDrawFlag              'progression of draw graph periodically

Dim integer minute=1                       'Array position of minuteCount
Dim integer minutesCount(1440)             'Array to store all minutes of a day
Dim integer count=0                        'stores actual counts
Dim integer progDraw=0
Dim float convFactor=0.004


Do
 count = Rnd*10                            'generate some random counts

 If progDraw Then                          'every minute progress drawing the graph
   Text 500,20,"Minute: "+Str$(minute)
   Text 500,40,"Count : "+Str$(count)
   Text 500,60,"uSv/h : "+Str$(count*convFactor,2,2)
   minutesCount(minute)=count              'store actual CPM to array
   minute=minute+1
   progDraw=0
   count=0
 EndIf
Loop

Sub setProgDrawFlag
 progDraw=1
End Sub
```

Gehen wir das Programm der Reihe nach durch; *SetTick 1000, setProgDrawFlag* veranlasst den Interpreter dazu, einen periodischen Interrupt des Programmes von 1000 ms durchzuführen. Es ist wichtig zu erwähnen, dass dieser Wert später zwingend auf 60000 ms gestellt wird, da schließlich die Counts Pro Minute von Interesse sind und nicht die Counts Pro Sekunde. Für dieses Beispiel wurde dieser Wert jedoch temporär herab gesetzt, sodass im späteren Verlauf nicht erst eine Minute abgewartet werden muss, bis der Graph Stück für Stück gezeichnet wird. Der dahinter befindliche Parameter *setProgDrawFlag* ist der Name der Subroutine, die nach Ablauf dieser Zeit abgerufen wird. Interrupt-Subroutinen sollten zu kurz wie möglich gehalten werden, daher wird in eben dieser nur ein sogenanntes *Flag* gesetzt, bzw. die Variable *progDraw = 1* gesetzt und die Subroutine sofort wieder verlassen. Es folgt schließlich die übliche Deklaration von Variablen.
In der *Do-Loop* ist die erste Aufgabe des Interpretes der Variable *count* mittels Rnd*10 einen Zufallswert zuzuweisen. Rnd greift dabei auf eine interne MMBASIC-Funktion zurück einen

Zufallswert zwischen 0 und 0.999999 zu generieren, welcher wiederum mit 10 multipliziert wird. In der darauffolgenden If-Schleife wird abgefragt, ob progDraw=1 ist (das kann abgekürzt werden, indem nur *If progDraw* abgefragt wird). Wurde nun nach 1000 ms durch den SetTick-Interrupt *progDraw* tatsächlich auf 1 gesetzt, dann wird diese Schleife betreten und alle Parameter, die von Interesse sind, werden auf den Bildschirm ausgegeben. *Text 500,20,"Minute: "+Str$(minute)* gibt die aktuelle Minute seit Programmstart aus, während *Text 500,40,"Count : "+Str$(count)* den Count-Wert (Counts Per Minute) des Pseudo-Zufallgenerators anzeigt. *Text 500,60,"uSv/h : "+Str$ (count*convFactor,2,2)* hingegen berechnet die Äquivalenzdosis in µSv/h indem der *count*-Wert mit einem Anpassungsfaktor multipliziert wird (die *,2,2* bedeuten, dass vor dem Komma *2* Stellen Platz „reserviert“ werden und *2* Nachkommastellen angegeben werden). Dieser Faktor ist abhängig von der jeweilig verwendeten GM-Röhre und kann oftmals dem Datenblatt entnommen oder abgeleitet werden. Er ist an dieser Stelle noch nicht wichtig und wird, wie in der Variablendeklaration oben bestimmt, auf den Wert *0.004* festgelegt.
Die darauffolgende Zeile weist dem *minutesCount()*-Array in der aktuell befindlichen *minute* den in dieser Minute zustandegekommenen *count*-Wert zu (kurz gesagt: *minutesCount(minute)=count*). Dieses *minutesCount()*-Array wird später wichtig, da es alle y-Werte zum zeichnen des Graphen beinhaltet. Nachdem jetzt der aktuelle *count*-Wert dort gespeichert wurde, kann die *minute* um *1* inkrementiert werden. Die restlichen zwei Variablen *progDraw* sowie *count* werden *0* gesetzt, sodass sie in der nächsten Runde, welche vom periodischen *SetTick*-Interrupt ausgelöst wird, von neu besetzt werden können. Die If-Schleife wird mit dem *EndIf* beendet, ebenso wird die *Do-Loop* beendet.

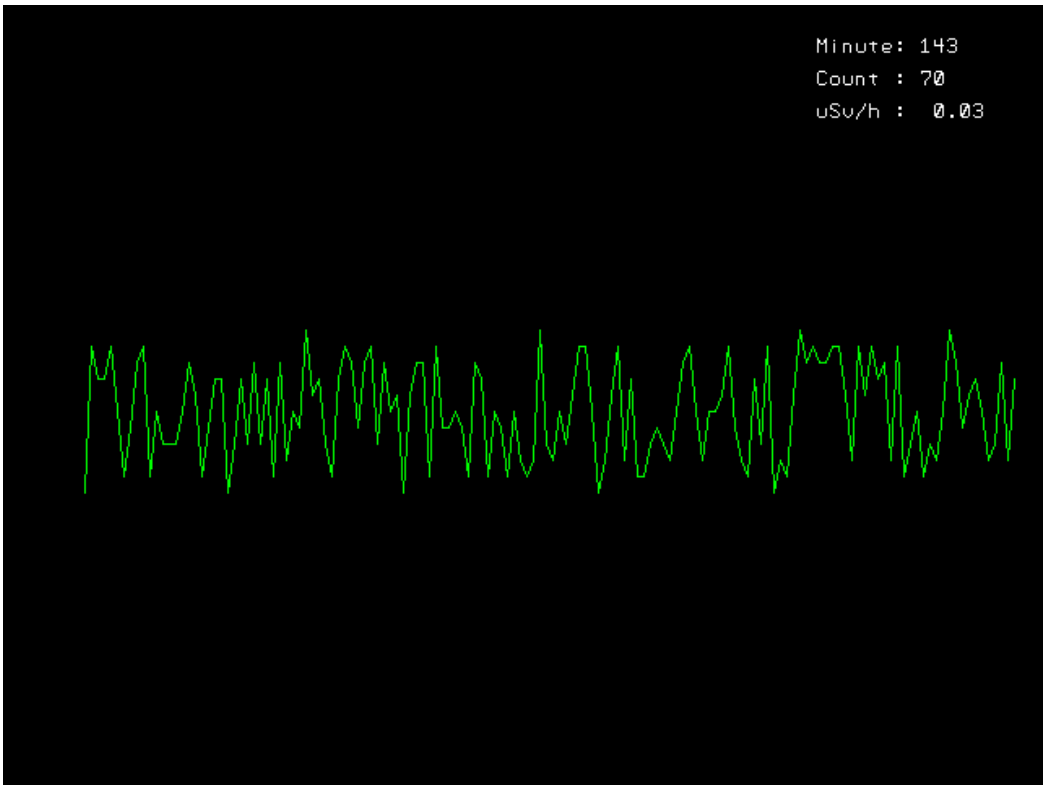
2. Zeichnen des Graphen

In diesem Abschnitt werden die pseudo-zufälligen *count*-Werte des gedachten Geigerzählers nun um die grafische Funktionalität ergänzt. Dazu wird der Programmcode um eine Zeichenfunktion in einer Subroutine erweitert. Wie gewohnt zunächst das vollständige Listing. Änderungen und Anpassungen sind **Fett** markiert.

```
Option EXPLICIT
MODE 3
CLS

SetTick 100, setProgDrawFlag              'progression of draw graph periodically

Dim integer minute=1                      'Array position of minuteCount
Dim integer minutesCount(1440)            'Array to store all minutes of a day
Dim integer count=0                       'stores actual counts

'Declare Drawing variables ++++++++++++++++++++++++++++++++++
Dim integer xProgGraph=50, cGraph, cGraph_old
Dim integer progDraw=0
Dim float convFactor=0.004, yGraphAdj=10

Do
 count = Rnd*10                           'generate some random counts

 If progDraw Then                         'every minute progress drawing the graph
   Text 500,20,"Minute: "+Str$(minute)
   Text 500,40,"Count : "+Str$(count)
   Text 500,60,"uSv/h : "+Str$(count*convFactor,2,2)
   minutesCount(minute)=count             'store actual CPM to array
   drawGraph()
   minute=minute+1
   progDraw=0
   count=0
 EndIf
```

```
Loop

Sub setProgDrawFlag
 progDraw=1
End Sub

Sub drawGraph
  xProgGraph = xProgGraph + 4                    'Progress 4 pixels on x-axis
  cGraph = count * yGraphAdj                     'Graph enlargement
 'Finally Draw Line Graph ----------------------
  Line xProgGraph-4,300-cGraph_old,xProgGraph,300-cGraph,,RGB(green)
  cGraph_old = cGraph
End Sub
```

Das Bild des laufenden Programms:



Entsprechend der Fett-Markierung im Programm-Listing wurde die *SetTick*-Zeit erneut auf 100 ms verringert, sodass die geplottete Funktion, bzw. der Graph schneller ersichtlich wird. Zudem wurden neue Variablen hinzugefügt, und die hauptsächlich an der Zeichnung des Graphen beteiligten Variablen unter der Überschrift *Declare Drawing variables* neu zusammengefasst. Das Programm wird relativ schnell größer, sodass es sich spätestens jetzt anbietet, eine gewisse Struktur vorzugeben. Wie eingangs angedeutet, findet sich in der *Do-Loop* nun eine dedizierte Subroutine *drawGraph()*, welche zum Zeichnen des Graphen abgerufen wird.
Die Struktur der Subroutine ist analog des Beispiels aus dem Grundlagen-Teil. So entspricht *xProgGraph = xProgGraph + 4* dem fortschreitenden x-Wert, sodass die neuen Linien jeweils um den angegebenen Wert (*4*) inkrementiert werden. Achtung: *xProgGraph* beginnt nun nicht bei dem (ehemals x-)Wert = 0, sondern bei 50. Man sieht gut auf dem Bild, dass das Plotten nicht direkt am linken Bildschirmrand startet, sondern um besagte 50 Pixel nach rechts versetzt. Dem y-Wert hingegen entspricht die Variable *cGraph*. Er soll für den ***c****ount*-Graphen stehen, denn er setzt sich aus *count * yGraphAdj* zusammen. Letzterer ist notwendig, um den *cGraph* (ehemals y-Wert,

letztlich die Höhe des Ausschlags) insgesammt anzupassen. Diese Anpassung verzehnfacht die Höhe und macht sie besser deutlich (*yGraphAdj* = 10).

Zu guter Letzt wird die Linie gezeichnet:

Line xProgGraph-4,300-cGraph_old,xProgGraph,300-cGraph,,RGB(green)

Dabei findet sich keine abweichende Besonderheit zu dem Grundlagenbeispiel. Deutlich wird allerdings, dass statt *MM.VRes* (=480 Pixel) nun mit dem Wert 300 gearbeitet wird. Das ermöglicht etwas „Luft nach unten“ und hebt den Graphen insgesamt mehr in die Mitte.

Rekapitulation:

Was wurde bisher erreicht?

- Es wird periodisch ein Interrupt durchgeführt (*SetTick*), welcher Auslöser für das Zeichnen des Graphen ist und im *finalen* Programm ein mal die Minute ausgeführt wird

- Vorläufige Datenerzeugung durch Pseudo-Zufallswerte

- Darstellung dieser Daten in Form von numerischer Ausgabe sowie von geplotteten Linien

Probleme und Fehlerbetrachtung:

- Wenn der *minutesCount(1440)* Array den Wert 1440 überschreitet: Fehlermeldung, da die definierten Array-Grenzen überschritten werden

- Wenn *minute > 144, dann* verharrt der aktuelle x-Wert, der geplottet wird, am rechten Bildschirmrand

Lösung:

Nehmen wir uns zunächst dem Problem der Fehlermeldung an, welche bei Überschreitung des *minutesCount(1440)* Arrays erscheint. Zunächst wird in die *Do-Loop* folgende If-Bedingung eingefügt:

```
If minute=1440 Then
  End
EndIf
```

Somit erfolgt eine korrekte Beendigung des Programms sobald *minute* 1440 erreicht ist. Im finalen Programm kann an dieser Stelle ein Reset aller Variablen erfolgen und ggf. eine Speicherung vorhandener Daten, z. B. auf SD-Karte, veranlasst werden. Wesentlich wichtiger für das Testen der Software ist jedoch das zweite genannte Problem; und zwar, dass ab *minute* 144 das Plotten am rechten Bildschirmrand „einfriert“. Die Lösung ist recht trivial und wird ebenfalls in die *Do-Loop* integriert:

```
 If xProgGraph=626 Then          'if end if scale, redraw
  xProgGraph=50                  'reset graph x-axis progress pixel to 50
  cGraph=0
  cGraph_old=0
  CLS
 EndIf
```

Das heißt, wenn *xProgGraph* am rechten Bildschirmrand angekommen ist (bei dem Wert 626), dann wird die Variable xProgGraph einfach auf ihren Anfangswert von 50 zurückgesetzt, ebenso die weiteren Variablen, welche auf den Graph bezogen sind. Schlussendlich erfolgt mittels *CLS* eine Säuberung des Bildschirms.

3. Autoscaling-Funktion des Graphen

Nun ist es oftmals der Fall, dass Messwerte deutlich nach oben ausschlagen können. In diesem Fall würden sie das Programm in seiner bisherigen Form überfordern, indem diese Extremwerte im schlimmsten Fall einfach bis zum oberen Bildschirmrand (und darüber hinaus) gezeichnet werden. Umgangssprachlich würden sie die Skala „sprengen“. Es ist also geboten Extremwerte zu berücksichtigen und diese im Verhältnis zu den restlichen Messwerten zu setzen; kurz der Graph sollte automatisch skalieren und gemessen am höchsten Messwert alle anderen Werte im Verhältnis zum Maximum verkleinert darstellen. Um diese Aufgabe soll es in diesem Teil des Kapitels gehen. Ein Vorschlag für die Umsetzung kann folgendermaßen aussehen, wie gewohnt sind in Fett-Markierung alle Änderungen im Vergleich zum vorherigen Programm hervorgehoben:

```
Option EXPLICIT
MODE 3
CLS

SetTick 100, setProgDrawFlag            'progression of draw graph periodically

Dim integer minute=1                    'Array position of "minuteCount"
Dim minutesCount(1440)                  'Array to store all minutes of a day
Dim integer newScaleMinutes=0           'If min > scale & redraw
Dim integer count=0

'Declare Drawing variables ++++++++++++++++++++++++++++++++++
Dim integer xProgGraph=50, cGraph, cGraph_old
Dim float uScaleIndex, convFactor=0.006, convFactor_old, uScaleVal
Dim integer progDraw, maxedOut
Dim float uScaleMax, yGraphAdj=10, scaleFactor=0.005

'Redraw Graph variables ++++++++++++++++++++++++++++++++++++
Dim integer xRedrawGraph=50, rGraph, rGraph_old
Dim integer recalcStart

'MAIN LOOP ##########################################################
Do
 count = Rnd*10                 'Get actual counts

 If minute = 50 Then count = count * 10          'Autoscaling Test (maxedOut)!
 If minute = 70 Then count = count * 50          'Autoscaling Test (maxedOut)!


 If minute >= 1440 Then
  End                                            'maximum array length (full)
 EndIf

 If progDraw Then                                'Every minute progress drawing the Graph
  Text 500,20,"Minute: "+Str$(minute)
  Text 500,40,"Count : "+Str$(count)
  Text 500,60,"uSv/h : "+Str$(count*convFactor,2,2)
  minutesCount(minute)=count                     'Store actual CPM to Array
  checkMaxScale()                                'Check if scale is maxed out
  If Not maxedOut Then                           'Not maxed out
```

```
    drawGraph()
   EndIf
   minute = minute+1                    'Increment minute
   progDraw = 0
   count = 0
  EndIf

  If xProgGraph=626 Then                'If End of Scale, redraw
   xProgGraph=50                        'Reset Graph x-axis progress pixel to 50
   xRedrawGraph=50
   cGraph=0
   cGraph_old=0
   rGraph=0
   rGraph_old=0
   CLS
  EndIf

Loop
'#############################################################

'Draw Graph ------------------------------------------------
Sub drawGraph
  xProgGraph = xProgGraph + 4                   'Progress 4 pixels on x-axis
  cGraph = count * yGraphAdj                    'Graph enlargement
 'Finally Draw Line Graph ----------------------
  Line xProgGraph-4,300-cGraph_old,xProgGraph,300-cGraph,,RGB(green)
  cGraph_old = cGraph
End Sub

'Set flag for periodical graphDraw (every minute triggered) =============
Sub setProgDrawFlag
  progDraw = 1
End Sub

'Recalculating values for the new graph & draw them ======================
Sub recalcArray
 For recalcStart=newScaleMinutes+1 To minute-1
   xRedrawGraph = xRedrawGraph + 4
   rGraph = minutesCount(recalcStart) * yGraphAdj       'Graph enlargement
   'Redraw Line Graph ----------------------------------------------
   Line xRedrawGraph-4,300-rGraph_old,xRedrawGraph,300-rGraph,,RGB(green)
   rGraph_old = rGraph
  Next recalcStart
  xProgGraph = xRedrawGraph                             'Assign coordinates to regular graph drawing
  cGraph = rGraph                                       'to seamlingless fit the two lines
  cGraph_old = rGraph_old                               ' ...
  xRedrawGraph = 50                                     'Reset for new round
  rGraph_old = 0                                        'Reset for new round
End Sub

'Check whether a new scale is needed ======================================
Sub checkMaxScale
  uScaleMax=(200/yGraphAdj)*convFactor
  Do While count*convFactor > uScaleMax                 'Time to adj. Graph & Scale?
   yGraphAdj=yGraphAdj-scaleFactor                      'Set new "scale factor"
   uScaleMax=(200/yGraphAdj)*convFactor                 'Set scale maximum
   maxedOut=1
  Loop

  If maxedOut=1 Then
   CLS
   recalcArray()
   maxedOut=0
```

```
  EndIf
End Sub
```

Neben den für die Autoskalierung benötigten neuen Variablen und deren Deklaration, ist in der *Do-Loop* nun für zwei Extremfälle mittels:

```
If minute = 50 Then count = count * 10      'Autoscaling Test (maxedOut)!
If minute = 70 Then count = count * 50      'Autoscaling Test (maxedOut)!
```

gesorgt. Diese beiden Ereignisse dienen dazu die Autoscaling-Funktion zu testen, indem der übliche pseudo-zufällige *count*-Wert mit 10 und 50 multipliziert wird und für (zu) hohe Ausschläge sorgt.

Mit *checkMaxScale()* wurde eine neue Subroutine eingeführt, welche wie der Name schon beschreibt, prüft, ob eine neue Skalierung von Nöten ist. In der Subroutine wird als erstes

```
uScaleMax=(200/yGraphAdj)*convFactor
```

definiert. Mit eingesetzten Werten ist uScaleMax = (200/10)*0.006 = 0.12 . Es folgt nun eine *Do-While-Loop* mit der Schleifenbediungung:

```
Do While count*convFactor > uScaleMax
```

Nehmen wir jetzt für den *count*-Wert beispielsweise 90 an und multiplizieren ihn mit dem *convFactor* von 0.006 erhalten wir 0.54. Die Bedingung der *Do-While*-Schleife ist damit erfüllt, denn 0.54 > *uScaleMax.* So lange wie diese Bedingung noch wahr ist, wird der Schleifeninhalt wiederholt.

Der Schleifeninhalt bestimmt zunächst, dass *yGraphAdj=yGraphAdj-scaleFactor* sei. Wir erinnern uns, yGraphAdj ist dafür zuständig den cGraph (y-Wert, oder Höhe des Graphen) zu bestimmen. Setzt man nun die Werte für das obige Beispiel mit einem *count*-Wert von 90 ein, so ergibt sich der neue *yGraphAdj*-Wert entsprechend:

```
 yGraphAdj=yGraphAdj-scaleFactor
```

yGraphAdj= 10 - 0.005 = 9.995. Es folgt die nächste Zeile der Schleife:

```
uScaleMax=(200/yGraphAdj)*convFactor
```

Wir setzen wieder die Werte ein:

und erhalten uSacleMax=(200/9.995)*0.006 = 0.12006. Die Schleife wird nun neu durchlaufen und die Bedingung auf’s Neue geprüft:

```
Do While count*convFactor > uScaleMax
```

Der *count*-Wert ist immer noch 90, also ergibt sich immer noch 90 * 0.006 = 0.54, jedoch hat sich nun der uScaleMax-Wert verändert (vorher 0.12), er beträgt laut aktuellem Schleifendurchlauf 0.12006. Er steigt also immer weiter an je mehr Durchläufe in der *Do-While-Loop* erfolgen. Irgendwann wird die Bedingung nicht mehr erfüllt sein, nämlich bei einem *uScaleMax*-Wert von 0.541, denn: 90*0.006 < 0.541. Da die *Do-While-Loop* die Variable:

```
maxedOut=1
```

gesetzt hat, springt der MMBASIC-Interpreter in den nächsten If-Block:

```
If maxedOut=1 Then
   CLS
   recalcArray()
   maxedOut=0
  EndIf
End Sub
```

da diese Bedingung erfüllt ist. Hier erfolgt via *CLS* eine Säuberung des Bildschirms, damit der alte Graph verschwindet und zunächst mittels *recalcArray()* neu berechnet werden kann. Die alten Werte müssen schließlich an den neuen Skalen-Faktor angepasst werden. Genau das passiert in der recalcArray() Subroutine, hier noch einmal abgebildet:

```
Sub recalcArray
 For recalcStart=newScaleMinutes+1 To minute-1
   xRedrawGraph = xRedrawGraph + 4
   rGraph = minutesCount(recalcStart) * yGraphAdj              'Graph enlargement
   'Redraw Line Graph ---------------------------------------------------
   Line xRedrawGraph-4,300-rGraph_old,xRedrawGraph,300-rGraph,,RGB(green)
   rGraph_old = rGraph
  Next recalcStart
  xProgGraph = xRedrawGraph                                    'Assign coordinates to regular graph drawing
  cGraph = rGraph                                              'to seamlingless fit the two lines
  cGraph_old = rGraph_old                                      ' ...
  xRedrawGraph = 50                                            'Reset for new round
  rGraph_old = 0                                               'Reset for new round
End Sub
```

Dieser Vorgang ist nicht ganz so trivial, denn es sollen nur alle Werte bis zum jetzigen Zeitpunkt minus eine Minute (*minute-1*) angepasst werden. Verfolgt man ganz genau den *minute*-Wert, so stellt man fest, dass der auftretende Extremwert nicht in der gegenwärtigen *minute* auftrat, sondern in der *minute* davor, weil der If-Block *If progDraw* erst im nachhinein genau diesen Umstand prüft, ob es eine Überschreitung der Skale gab (*checkMaxScale*). Diese *For-Loop wiederholt* also Schritt für Schritt für die Variable *recalcStart=newScaleMinutes+1* bis zum Erreichen von *minute-1* den Schleifeninhalt. Aber der Reihe nach:

```
xRedrawGraph = xRedrawGraph + 4
```

ist bekannt aus allen vorherigen Schritten, bei denen der Line-Befehl verwendet wurde, er ist vom Funktionsprinzip identisch mit *xProgGraph = xProgGraph + 4* und ist für die Rechtsverschiebung um je 4 Pixel jeder neuen Linie zuständig. Ebenso wie:

```
rGraph = minutesCount(recalcStart) * yGraphAdj
```

bekannt ist. Hier wird nun statt des *cGraph* der *rGraph* entsprechend an den veränderten *yGraphAdj*-Wert aus der vorherigen *Do-While-Loop* angepasst. Danach folgt die übliche Zeichenroutine mittels des *Line*-Befehls und schlussendlich das Ende der *For-Loop*:

```
   Line xRedrawGraph-4,300-rGraph_old,xRedrawGraph,300-rGraph,,RGB(green)
   rGraph_old = rGraph
  Next recalcStart
```

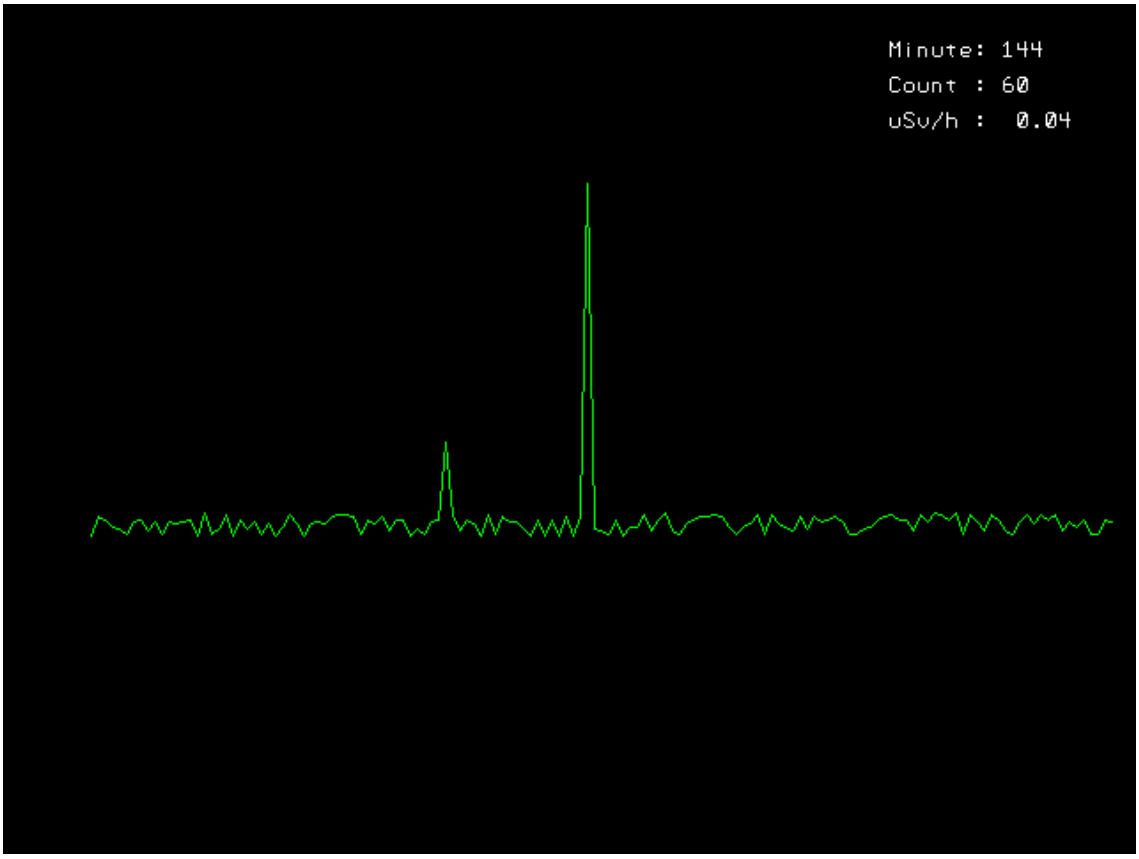
Ist die For-Loop so oft durchlaufen bis die Bedingung *minute-1* erfüllt ist, dann werden im letzten Schritt noch die Koordinaten, welche in den Redraw-Variablen hinterlegt sind, an die regulären Draw-Variablen übergeben und Parameter zurückgesetzt, sodass diese Routine erneut ausgelöst

werden kann. Die Subroutine *recalcArray()* ist abgeschlossen und der Interpreter springt zum If-Block zurück, wo er hergekommen ist:

```
If maxedOut=1 Then
   CLS
   recalcArray()
   maxedOut=0
  EndIf
End Sub
```

Nun wird die Variable *maxedOut* mittels 0 zurückgesetzt, sodass sie – sofern ein neuer Extremwert auftritt – neu ausgelöst werden kann.

Ein Beispiel wie der Graph nach der automatischen Anpassung aussehen kann. Die Werte vor den zwei Ausschlägen waren vorher genau so skaliert wie die Abb x zeigt:



4. Skalen – der finale Schritt zum echten Liniendiagramm

Nachdem bereits die automatische Skalierung des Graphen funktioniert, kann jedoch immer noch nichts am Graphen abgelesen werden, vielmehr können bisher nur Relationen abgeschätzt werden. Was fehlt, ist eine Skalen-Einteilung. Dieser Abschnitt beschreibt nicht nur wie eine Skale eingefügt werden kann, sondern auch wie die Skala automatisch an neue „Extremwerte“ angepasst wird, ganz analog zum Graphen des vorherigen Unterkapitel. Dieser Teil schließt also mit einem vollwertigen Liniendiagramm, welches auf Messwerte dynamisch mit automatischer Anpassung der Skala und des Graphen reagiert. Der vollständige Programmcode sieht wie folgt aus:

```
Option EXPLICIT
MODE 3
CLS

SetTick 100, setProgDrawFlag              'progression of draw graph periodically

Dim integer minute=1                      'Array position of "minuteCount"
Dim minutesCount(1440)                    'Array to store all minutes of a day
Dim integer newScaleMinutes=0             'If min > scale & redraw
Dim integer count=0

'Declare Drawing variables ++++++++++++++++++++++++++++++++++++
Dim integer yRow, xCol, xDash, yDash, xScale, mScaleVal, yScale
Dim integer xProgGraph=50, cGraph, cGraph_old
Dim float uScaleIndex, convFactor=0.006, convFactor_old, uScaleVal
Dim integer progDraw, maxedOut
Dim float uScaleMax, yGraphAdj=10, scaleFactor=0.005

'Redraw Graph variables ++++++++++++++++++++++++++++++++++++
Dim integer xRedrawGraph=50, rGraph, rGraph_old
Dim integer recalcStart

drawAxisGrid()                                  'draw chart once

'Draw all chart stuff ====================================
Sub drawAxisGrid
 CLS

 'Draw dotted Grit ----------------------------
 For yRow=100 To 300 Step 20 'Draw Rows (300-100=200 -> 200/20 = 10 rows)
  For xCol=50 To 626 Step 24 'Draw Columns (626-50=576 -> 576/24 = 24 columns)
   Line xCol,yRow,xCol,yRow,,RGB(lilac)
  Next xCol
 Next yRow

 'Draw solid Axis ----------------------------------
 Line 50,300,626,300,,RGB(green)                 'Draw X-axis
 Line 50,100,50,300,,RGB(green)                  'Draw Y-axis

 'Draw dashed lines --------------------------------
 For xDash=50 To 626 Step 24                     'Draw dashed X
  Line xDash,297,xDash,305,,RGB(midgreen)
 Next xDash

 For yDash=100 To 300 Step 20         'Draw dashed Y
  Line 50,yDash,51,yDash,,RGB(midgreen)
 Next yDash

 'Labels ------------------------------------------
 Text 1,70,"uSv/h ",,1,1,RGB(lilac)               'uSv/h
 Text 626,305,"t ",,1,1,RGB(lilac)                'Time


 'Draw minute Diagram -------
 For xScale=40 To 626 Step 120
  Text xScale,310,Str$(mScaleVal),,1,1,RGB(white) 'Minutes Scale
  mScaleVal=mScaleVal+30
 Next xScale
 mScaleVal=0                                     'Reset value for the next trigger

 For xScale=40 To 626 Step 120
  Text xScale,325,"Min",,1,1,RGB(lilac)          'Minutes Scale
```

```
 mScaleVal=mScaleVal+30
Next xScale
mScaleVal=0                                    'Reset value for the next trigger

uScaleVal=(20/yGraphAdj)*convFactor

For yScale=295 To 95 Step -20
 Text 1,yScale,Str$(uScaleIndex,2,2),,1,1,RGB(white)      'uSv/h Scale w/ 2 dec. pl.
 uScaleIndex=uScaleIndex+uScaleVal
Next yScale
uScaleIndex=0                                  'Reset value for the next trigger

End Sub

'MAIN LOOP ################################################################
Do
 count = Rnd*10                                'Get actual counts

 If minute = 50 Then count = count * 10        'Autoscaling Test (maxedOut)!
 If minute = 70 Then count = count * 50        'Autoscaling Test (maxedOut)!

 If minute >= 1440 Then
  End                                          'maximum array length (full)
 EndIf

 If progDraw Then                              'Every minute progress drawing the Graph
  Text 500,40,"Count : "+Str$(count)
  Text 500,60,"uSv/h : "+Str$(count*convFactor)
  minutesCount(minute)=count                   'Store actual CPM to Array
  checkMaxScale()                              'Check if scale is maxed out
  If Not maxedOut Then                         'Not maxed out
   drawGraph()
  EndIf
  minute = minute+1                            'Increment minute
  progDraw = 0
  count = 0
 EndIf

 Text 500,10,"Minute: "+Str$(minute)

 If xProgGraph=626 Then                        'If End of Scale, redraw
  xProgGraph=50                                'Reset Graph x-axis progress pixel to 50
  xRedrawGraph=50
  cGraph=0
  cGraph_old=0
  rGraph=0
  rGraph_old=0
  drawAxisGrid()
 EndIf

Loop
'###########################################################################

'Draw Graph ----------------------------------------------
Sub drawGraph
 xProgGraph = xProgGraph + 4                   'Progress 4 pixels on x-axis
 cGraph = count * yGraphAdj                    'Graph enlargement
 'Finally Draw Line Graph ----------------------
 Line xProgGraph-4,300-cGraph_old,xProgGraph,300-cGraph,,RGB(green)
 cGraph_old = cGraph
End Sub

'Set flag for periodical graphDraw (every minute triggered) ==============
```

```
Sub setProgDrawFlag
  progDraw = 1
End Sub

'Recalculating values for the new graph & draw them ====================
Sub recalcArray
 For recalcStart=newScaleMinutes+1 To minute-1
    xRedrawGraph = xRedrawGraph + 4
    rGraph = minutesCount(recalcStart) * yGraphAdj            'Graph enlargement
    'Redraw Line Graph ------------------------------------------------
    Line xRedrawGraph-4,300-rGraph_old,xRedrawGraph,300-rGraph,,RGB(green)
    rGraph_old = rGraph
  Next recalcStart
  xProgGraph = xRedrawGraph                                   'Assign coordinates to regular graph drawing
  cGraph = rGraph                                             'to seamlingless fit the two lines
  cGraph_old = rGraph_old                                     ' ...
  xRedrawGraph = 50                                           'Reset for new round
  rGraph_old = 0                                              'Reset for new round

End Sub

'Check whether a new scale is needed ========================================
Sub checkMaxScale
  uScaleMax=(200/yGraphAdj)*convFactor
  Do While count*convFactor > uScaleMax                       'Time to adj. Graph & Scale?
    yGraphAdj=yGraphAdj-scaleFactor                           'Set new "scale factor"
    uScaleMax=(200/yGraphAdj)*convFactor                      'Set scale maximum
    maxedOut=1
  Loop

  If maxedOut=1 Then
    drawAxisGrid()
    recalcArray()
    maxedOut=0
  EndIf
End Sub
```

Nach dem Deklarieren neuer Variablen für die Skalen unter *Declare Drawing variables* ist eine neue Subroutine eingefügt worden, welche zum einen die x- und y-Achsen zeichnet, aber auch ihre Beschriftung bzw. Einteilung in ihre jeweiligen Einheiten der Zeit (Min.) sowie der Äquivalentdosis (µSv/h) vornimmt. Die besondere Herausforderung ist, dass sich unter neu auftretenden Extremwerten nicht nur die Höhe / Skalierung des Graphen ändern, sondern folglich auch die Skale selbst angepasst werden muss. All das findet hier statt.
Oben angefangen, wird die *drawAxisGrid()* Subroutine einmalig vor dem Betreten der Hauptschleife bzw. der *Do-Loop* ausgeführt. Es soll schließlich direkt zu Beginn die neue Skaleneinteilung und Achsenbeschriftung sichtbar sein. Diese Subroutine wird nachfolgend nur durch eine Änderungen der Skalierung, oder bei Erreichen der maximalen Minutenanzahl aufgerufen. Den größten Anteil des Programmcodes innerhalb dieser neuen Subroutine machen die vielen Beschriftungs- und Zeichenbefehle für die Achsen und deren Einteilung aus. Hier gibt es keine Besonderheiten auf denen es sich lohnt näher einzugehen.
Die automatische Anpassung der *µSv/h*-Skale findet in diesem Teil statt:

```
uScaleVal=(20/yGraphAdj)*convFactor

 For yScale=295 To 95 Step -20
  Text 1,yScale,Str$(uScaleIndex,2,2),,1,1,RGB(white)        'uSv/h Scale w/ 2 dec. pl.
  uScaleIndex=uScaleIndex+uScaleVal
 Next yScale
 uScaleIndex=0                                              'Reset value for the next trigger
```

Dabei ist *uScaleVal* der aufsteigend angezeigte *µSv/h*-Wert der y-Achse und setzt sich aus *(20/yGraphAdj)*convFactor* zusammen. Treten keine Extremwerte auf, so ist der Ausgangswert für *yGraphAdj* = 10 und der *convFactor* ist 0.006. Das ergibt einen *uScaleVal* von 0.012.

Die darauffolgende *For-Loop* geht von *yScale*=295 bis 95 in -20‘er Schritten voran und setzt an den jeweiligen y-Koordinaten (*yScale*) den jeweiligen *uScaleIndex.* Dabei handelt es sich um die *µSv/h*-Werte. Der *uScaleIndex* errechnet sich aus *uScaleIndex+uScaleVal.* Setzt man die Werte ein, erben sich sich für den *uScaleIndex* für die entsprechenden Schleifendurchläufe folgende Werte:

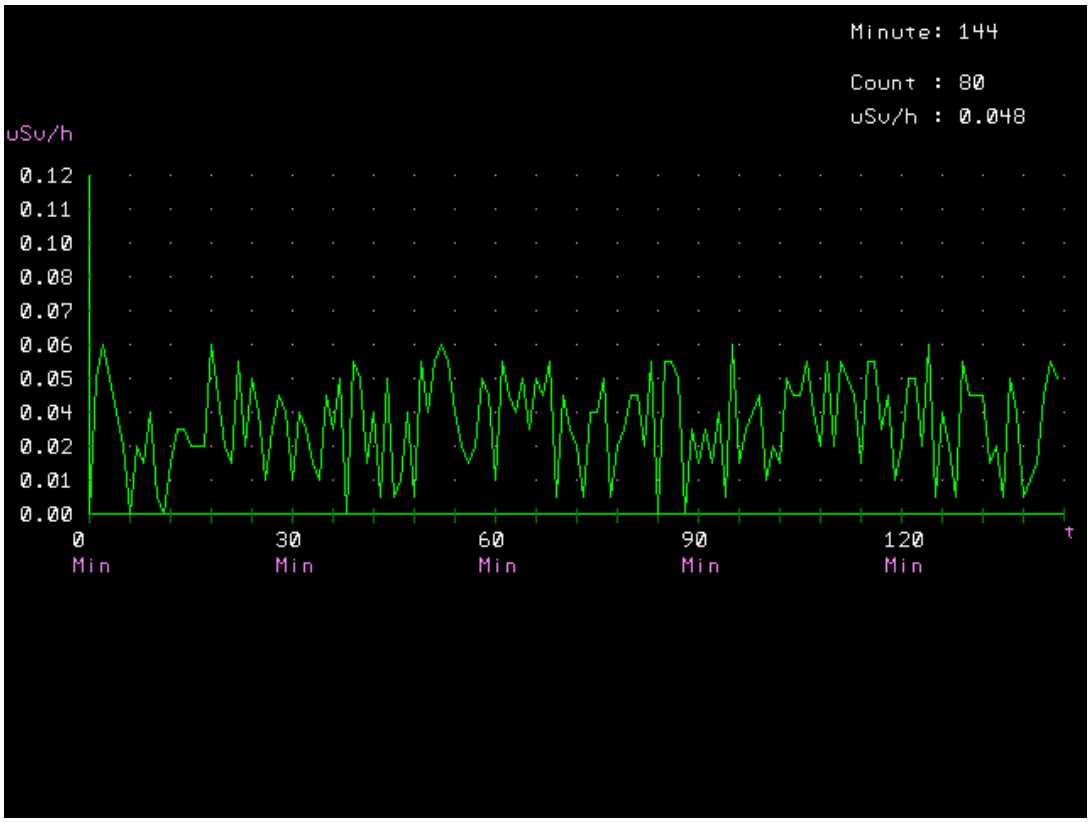
1. Durchlauf: 0.012
2. Durchlauf: 0.024
3. Durchlauf: 0.036
4. Durchlauf: 0.048
5. Durchlauf: 0.060
6. Durchlauf: 0.072
7. Durchlauf: 0.084
8. Durchlauf: 0.096
9. Durchlauf: 0.108
10. Durchlauf: 0.120

Durch die -20‘er Schritte erfolgt die Darstellung tatsächlich natürlich in umgekehrter Reihenfolge. Wird das Programm getestet, stellt sich jedoch heraus, dass die Werte *scheinbar* abweichen. Das begründet sich in der gewählten Darstellungsform:

```
Text 1,yScale,Str$(uScaleIndex,2,2),,1,1,RGB(white)
```

*Str$(uScaleIndex,**2,2**)* besagt, dass auf zwei Nachkommastellen *gerundet* wird. Selbstverständlich kann dieser Wert beliebig verändert werden. Nach dem letzten Durchlauf wird noch die Variable *uScaleIndex* auf 0 zurückgesetzt, sodass sie neu ausgelöst werden kann.

Das nun vollständige Liniendiagramm mit automatischer Skalierung von Graph und Skaleneinteilung sieht wie folgt aus, zunächst ohne extreme Messwerte:



Und hier mit „Extremwerte“ und der gewünschten automatischen Anpassung; Skaleneinteilung sowie Graph sind angepasst: