

Sprites

Nach soviel grauer Theorie kommt nun wieder Bewegung ins Spiel!

In diesem Kapitel geht es um die Programmierung von Sprites. Das sind kleine, bewegliche Objekte, deren Aussehen Sie innerhalb bestimmter Grenzen frei gestalten können und die sich dann beispielsweise für Spiele verwenden lassen.

Wir werden die Sprite-Programmierung zunächst in BASIC durchführen, um die grundlegenden Abläufe kennenzulernen. Aber keine Sorge, für jedes BASIC-Programm werden wir immer das entsprechende Assembler-Gegenstück erstellen.

Sprites werden vom Commodore 64 bereits seitens der Hardware unterstützt und das vereinfacht die Programmierung erheblich. Es werden standardmäßig 8 Sprites unterstützt, doch es sind durch Anwendung spezieller Techniken auch mehr Sprites möglich.

Es gibt einfarbige Sprites und mehrfarbige Sprites, wobei wir uns zunächst mit den einfarbigen Sprites beschäftigen wollen.

Einfarbige Sprites können eine von 16 Farben annehmen und maximal 24 Pixel breit bzw. maximal 21 Pixel hoch sein. Ein Sprite besteht also insgesamt aus 504 Punkten.

Bevor wir mit einem Sprite arbeiten können, müssen wir erst einmal wissen, wie es aussehen soll.

Doch wie sagt man dem C64, wie man sich das Sprite vorstellt?

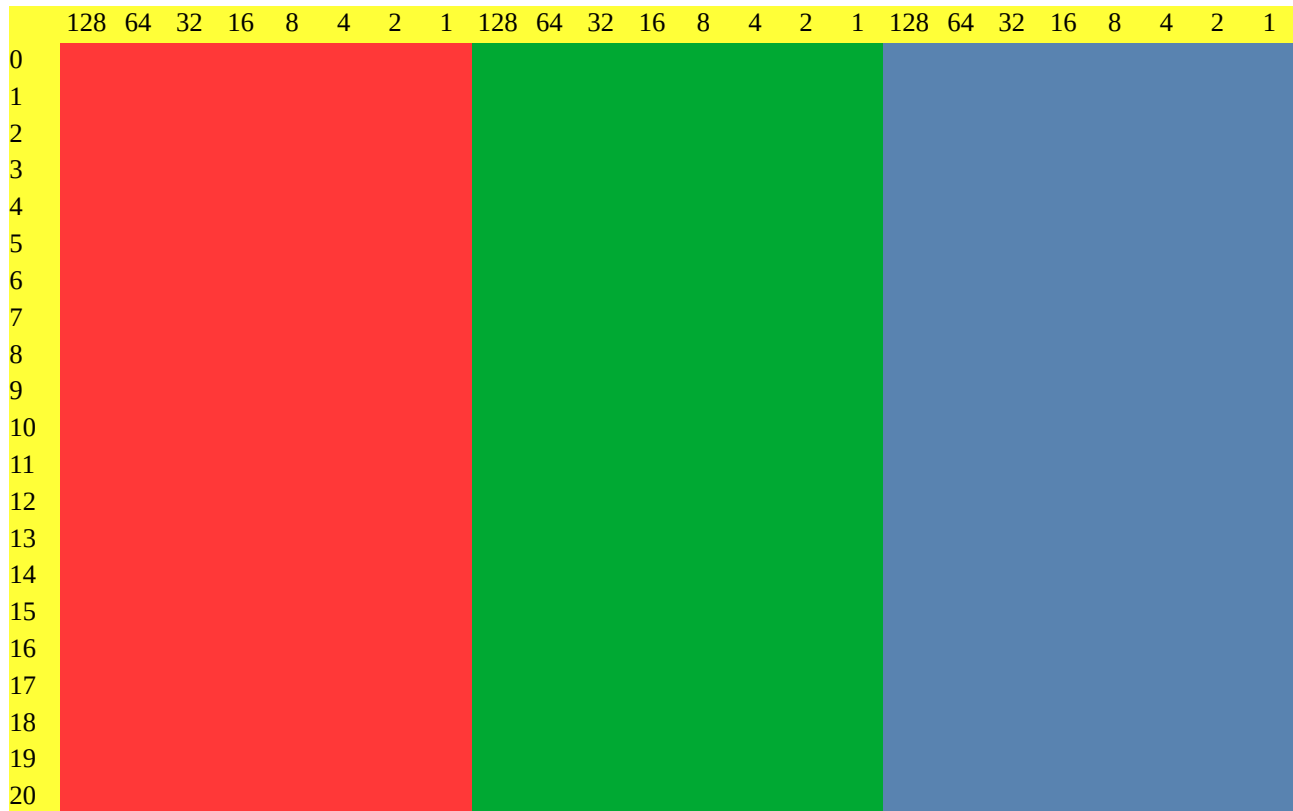
Dazu zeichnet man sich zunächst beispielsweise auf kariertem Papier einen Raster mit 24 Spalten und 21 Zeilen auf, wobei jede Zelle des Rasters einem der 504 Pixel des Sprites entspricht.

Das Sprite hat eine horizontale Auflösung von 24 Pixel und wenn man jedem Pixel ein Bit zuordnet, dann benötigen wir 3 Bytes ($3 \times 8 \text{ Bit}$ für 24 Pixel) um eine Zeile aus unserem Raster speichern zu können.

In vertikaler Richtung beträgt die Auflösung 21 Pixel, d.h. wir benötigen insgesamt ($3 \times 21 \text{ Bytes} = 63 \text{ Bytes}$) um das Aussehen unseres Sprites festzulegen.

Ein Block mit Spritedaten muss jedoch 64 Bytes umfassen, daher folgt auf das letzte Byte noch ein Platzhalter-Byte zum nächsten Block.

Unser Sprite-Raster sieht folgendermaßen aus:



Jede Zeile besteht wie gesagt aufgrund der horizontalen 24 Pixel aus 3 Bytes, der rote Bereich entspricht dem ersten, der grüne Bereich dem zweiten und der blaue Bereich dem dritten Byte in jeder Zeile.

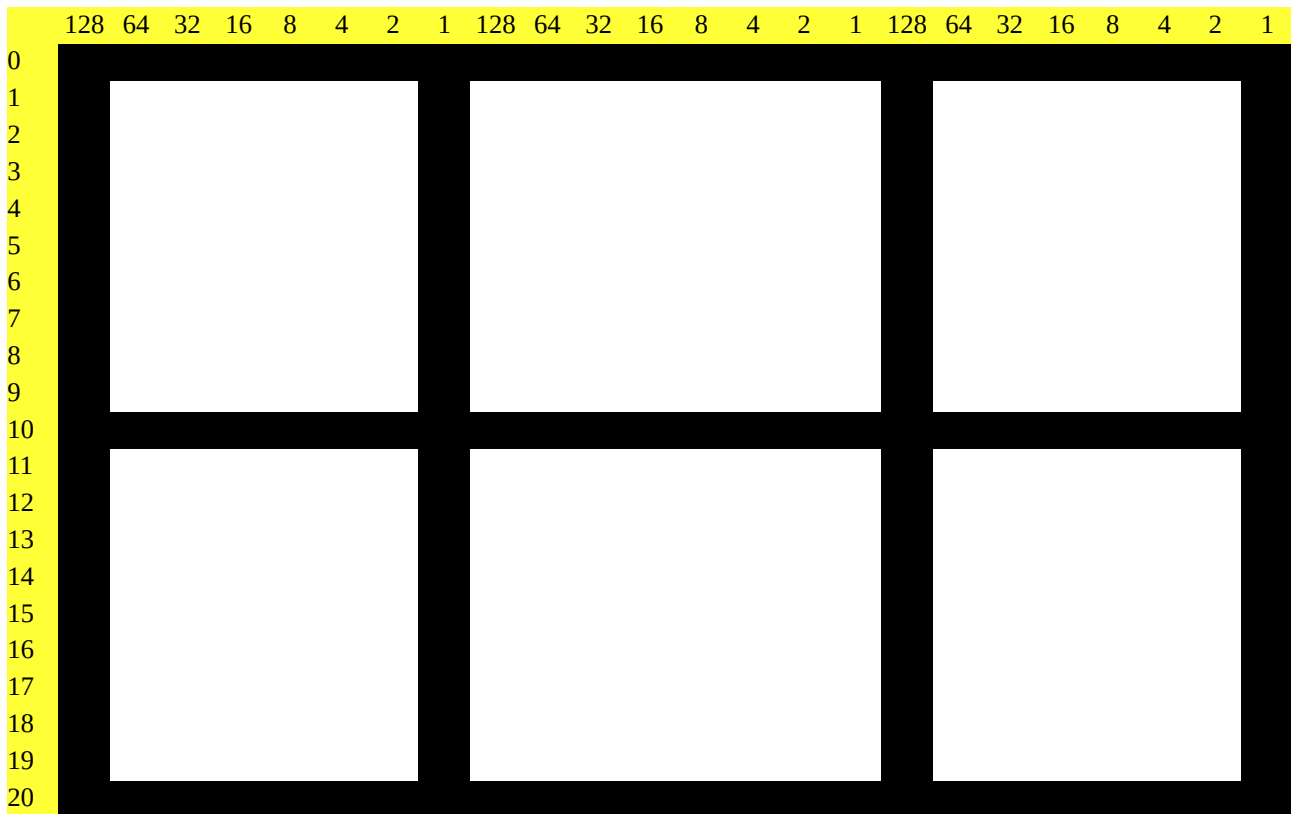
Über jedes Bit der drei Bytes schreiben wir die jeweilige Wertigkeit an der Stelle.

Diese beginnt jeweils mit 128 (2^7) und endet jeweils mit 1 (2^0)

Nehmen wir nun einen leeren Raster und zeichnen uns Pixel für Pixel ein einfach gehaltenes Sprite.

Wir füllen alle Stellen im Raster, an die wir einen Pixel setzen wollen.

Zeichnen Sie folgende einfache Form in den Raster. Jede ausgefüllte Rasterzelle entspricht einem gesetzten Pixel (das Bit hat also den Wert 1). An den weißen Stellen haben wir keinen Pixel gesetzt (das Bit hat also den Wert 0), d.h. hier scheint der Hintergrund durch.



Sehen wir uns das erste Byte in der ersten Zeile an, hier haben wir an jeder Bitposition eine ausgefüllte Zelle, also eine 1. Dies entspricht der binären Zahl %11111111 (hexadezimal \$FF bzw. dezimal 255)

Beim zweiten und dritten Byte ist ebenfalls an jeder Bitposition eine 1, d.h. wir haben auch hier den binären Wert %11111111 (hexadezimal \$FF bzw. dezimal 255)

Unsere erste Zeile wird also durch die drei Bytes 255,255,255 beschrieben.

Gehen wir nun zum ersten Byte in der zweiten Zeile.

Hier haben wir an den Bitpositionen 7 und 0 eine 1 stehen, d.h. wir haben hier die binäre Zahl %10000001 (hexadezimal \$81 bzw. dezimal 129)

Im zweiten Byte haben wir keine gesetzten Bits, d.h. wir haben hier den binären Wert %00000000 (hexadezimal \$00 bzw. dezimal 0)

Das dritte Byte entspricht dem ersten Byte, auch hier haben wir den binären Wert %10000001 (hexadezimal \$81 bzw. dezimal 129)

Die zweite Zeile wird also durch die drei Bytes 129,0,129 beschrieben.

Das setzen wir nun fort bis zur letzten Zeile und erhalten insgesamt folgende Zahlenwerte für die 21 Zeilen:

| Erstes Byte | Zweites Byte | Drittes Byte |
|-------------|--------------|--------------|
| 255 | 255 | 255 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 255 | 255 | 255 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 129 | 0 | 129 |
| 255 | 255 | 255 |

Soweit so gut. Aber wo speichern wir diese Zahlen nun ab? Da wir wie gesagt zunächst in BASIC programmieren wollen, legen wir die Zahlen in DATA-Zeilen ab.

Wir beginnen mit hohen Zeilennummern, da wir davor später noch weiteren BASIC-Code einfügen wollen.

```

READY.
1000 REM DATEN FUER SPRITE 0
1010 DATA 255,255,255
1020 DATA 129,0,129
1030 DATA 129,0,129
1040 DATA 129,0,129
1050 DATA 129,0,129
1060 DATA 129,0,129
1070 DATA 129,0,129
1080 DATA 129,0,129
1090 DATA 129,0,129
1100 DATA 129,0,129
1110 DATA 255,255,255
1120 DATA 129,0,129
1130 DATA 129,0,129
1140 DATA 129,0,129
1150 DATA 129,0,129
1160 DATA 129,0,129
1170 DATA 129,0,129
1180 DATA 129,0,129
1190 DATA 129,0,129
1200 DATA 129,0,129
1210 DATA 255,255,255

```

Nun müssen wir diese Daten an einem passenden Platz im Speicher ablegen.
Aber wo? Und wie sagen wir dann dem C64 wo wir die Daten für unser Sprite abgelegt haben?

Kümmern wir uns zuerst darum, wo wir unsere Daten im Speicher ablegen.

Sprite-Daten können wir nicht an jeder beliebigen Stelle im Speicher ablegen. Der Speicherbereich, den wir uns aussuchen, muss zwei Kriterien erfüllen:

- Er muss an einer durch 64 teilbaren Adresse beginnen
- Er muss 63 Byte durchgehend frei nutzbaren Platz bieten, denn wir dürfen unsere Sprite-Daten natürlich nicht in einen Speicherbereich schreiben, der bereits für andere Daten genutzt wird. 63 Byte deswegen, weil das 64. Byte nur als Platzhalter zum nächsten Block dient und nicht in die Spritedaten einfließt.

Durch 64 teilbare Adressen gibt es ja viele, aber wir müssen in dem Speicherbereich auch alle unsere 63 Bytes unterbringen können, ohne dabei andere Daten zu überschreiben.

Es hilft uns nichts, wenn die Adresse durch 64 teilbar ist, wir aber nur vielleicht 15 Bytes nutzen können, weil ab dem 16. Byte vielleicht bereits andere Daten folgen, die nicht überschrieben werden dürfen.

Glücklicherweise gibt es einige solcher frei verfügbaren Bereiche, welche diese Kriterien erfüllen und die wir daher zur Ablage unserer Sprite-Daten nutzen können.

Doch alles schön der Reihe nach.

Warum muss der Speicherbereich an einer durch 64 teilbaren Adresse beginnen?

Der Grund ist folgender:

Die 8 Speicherstellen von 2040 bis 2047 haben in Bezug auf Sprites eine wichtige Bedeutung.

Jede dieser 8 Speicherstellen ist einem Sprite zugeordnet, Speicherstelle 2040 ist Sprite 0 zugeordnet, Speicherstelle 2041 ist Sprite 1 zugeordnet, bis hin zur Speicherstelle 2047, welche Sprite 7 zugeordnet ist.

Jede dieser Speicherstellen enthält eine Blocknummer zwischen 0 und 255.

Diese Blocknummer multipliziert mit 64 ergibt dann jene Speicheradresse, die den Beginn des Speicherbereichs darstellt, in welchem wir die 63 Bytes Daten für unser Sprite ablegen.

Spielen wir das mal anhand der Speicherstelle 2040 durch, d.h. mit jener Speicherstelle, welche die Blocknummer für die Daten von Sprite 0 enthält.

Angenommen, sie enthielte die Blocknummer 0, dann würden die Spritedaten an Adresse $0 * 64 = 0$ beginnen. Diesen Block können wir jedoch nicht benutzen, denn wenn wir auf der Seite <https://www.c64-wiki.de/wiki/Zeropage> einen Blick auf die Belegung der Zeropage werfen, dann sehen wir, dass der Bereich von Adresse 0-63 bereits von anderen wichtigen Daten genutzt wird.

Probieren wir es mit Blocknummer 1, das wären dann die Adressen ab Adresse $1 * 64$, also Adresse 64. Tja, laut den Informationen auf der oben genannten Seite ist Block 1 leider auch schon vergeben.

Das geht leider weiter bis inklusive Block 10, also den Adressen 640 – 703.

Den Bereich mit der Blocknummer 11, also der Bereich von Adresse 704 bis 767, können wir jedoch für die Ablage unserer Spritedaten nutzen, da er nicht benutzt wird.

| | | | |
|---------------|-----------|--|---|
| \$2C0 - \$2FF | 704 - 767 | | Platz für Spritedatenblock 11, da nicht genutzt |
|---------------|-----------|--|---|

Um es gleich vorweg zu nehmen:

Auch die Blöcke mit den Nummern 13, 14 und 15 können wir für unsere Spritedaten nutzen.

| | | | |
|---------------|------------|--|--|
| \$340 - \$37F | 832 - 895 | | Platz für Spritedatenblock 13 (nur bei Nichtnutzung des Datasetten-/Kassettenpuffers!) |
| \$380 - \$3BF | 896 - 959 | | Platz für Spritedatenblock 14 (nur bei Nichtnutzung des Datasetten-/Kassettenpuffers!) |
| \$3C0 - \$3FF | 960 - 1023 | | Platz für Spritedatenblock 15 (nur bei Nichtnutzung des Datasetten-/Kassettenpuffers!) |

Es gibt noch einiges anzumerken in Bezug auf die Blocknummern, doch das würde an dieser Stelle nur verwirren. Am Ende des Kapitels werde ich dies nachholen.

Festlegen der Blocknummer für die Spritedaten

Gut, dann nehmen wir doch für die Daten unseres Sprites gleich den ersten Block, den wir gefunden haben, also den mit der Nummer 11.

Wir fügen also folgende Zeile hinzu:

```
10 POKE 2040,11
```

Dadurch weiß der C64, dass die Daten für das Sprite 0 in Block 11 liegen, also ab der Speicheradresse 704 ($11 * 64$) zu finden sind.

Doch das ist erst die halbe Miete, denn bis jetzt stehen unsere Spritedaten nur in den DATA-Zeilen und noch nicht in dem Speicherblock 11.

Das Kopieren führen wir mittels folgender Schleife durch:

```
20 FOR I=0 TO 62  
30 READ A  
40 POKE 704+I,A  
50 NEXT I
```

Nun müssen wir noch eine ganze Reihe bestimmter Speicherstellen verändern, damit unser Sprite in der gewünschten Form auf dem Bildschirm angezeigt wird.

Typ des Sprites festlegen (einfarbig oder mehrfarbig)

In der Speicherstelle 53276 ist jedes der 8 Bits mit einem Sprite verbunden, Bit 0 mit Sprite 0 bis hin zu Bit 7, welches mit Sprite 7 verbunden ist. Setzt man ein Bit auf den Wert 0, dann gibt man dadurch an, dass es sich bei dem Sprite, welches mit diesem Bit verbunden ist, um ein einfarbiges Sprite handelt. Setzt man den Wert hingegen auf den Wert 1, dann gibt man dadurch an, dass es sich um ein mehrfarbiges Sprite handelt.

Da wir uns aktuell mit den einfarbigen Sprites beschäftigen, setzen wir das Bit an der Position 0 auf den Wert 0. Dadurch wird das Sprite 0 als einfarbig markiert.

Hier kommt uns nun unser Wissen über logische Verknüpfungen entgegen, denn wir müssen hier das Bit 0 auf den Wert 0 setzen.

Dazu brauchen wir folgende UND-Verknüpfung:

```
60 POKE 53276,PEEK(53276) AND (NOT (1))
```

Farbe des Sprites festlegen

Die Speicherstellen von 53287 bis 53294 enthalten die Farben für die 8 Sprites (falls es sich um einfarbige Sprites handelt)

Wir wählen für Sprite 0 die Farbe Weiß, also müssen wir den Wert 1 in die Speicherstelle 53287 schreiben.

```
70 POKE 53287,1
```

Festlegen der Spriteposition

Die Position eines Sprites wird durch eine Pixelposition in horizontaler und durch eine Pixelposition in vertikaler Richtung angegeben. In horizontaler Richtung (X) sind Werte von 0 bis

511 möglich und in vertikaler Richtung (Y) sind es Werte zwischen 0 und 255, wobei die Position X=0, Y=0 in der linken oberen Ecke des Bildschirms liegt.

Hier ist jedoch wirklich die linke obere Ecke des gesamten Bildschirms inklusive Rahmen gemeint, nicht die linke, obere Ecke des Ausgabebereichs, in dem beispielsweise die Textausgaben erfolgen.

Für die X-Koordinaten der 8 Sprites sind die Speicherstellen 53248, 53250, 53252, 53254, 53256, 53258, 53260 und 53262 zuständig, im Falle von Sprite 0 müssen wir die X-Koordinate also in der Speicherstelle 53248 ablegen.

Um das Sprite an den linken Rand des sichtbaren Bereichs zu positionieren, ist nicht, wie vielleicht vermutet, der Wert 0 erforderlich, sondern der Wert 24.

Die Einstellung der X-Koordinate führen wir mit dem Befehl

```
80 POKE 53248,24
```

durch.

Nun müssen wir uns noch um die Y-Koordinate kümmern.

Für die Y-Koordinaten der 8 Sprites sind die Speicherstellen 53249, 53251, 53253, 53255, 53257, 53259, 53261 und 53263 zuständig, im Falle von Sprite 0 müssen wir die Y-Koordinate also in der Speicherstelle 53249 ablegen.

Der Wert für den obersten Rand des sichtbaren Bereichs lautet 50.

Die Einstellung der Y-Koordinate für diese Position führen wir mit dem Befehl

```
90 POKE 53249,50
```

durch.

Festlegen der Sprite-Priorität in Bezug auf den Hintergrund

Dazu brauchen wir die Speicherstelle 53275. Auch diese Speicherstelle folgt dem bereits beschriebenen Schema und enthält für jedes Sprite ein eigenes Bit.

Enthält dieses Bit den Wert 0, dann hat das Sprite eine höhere Priorität als der Hintergrund und wird daher vor dem Hintergrund dargestellt. Enthält das jeweilige Bit jedoch den Wert 1, dann hat der Hintergrund höhere Priorität und das Sprite wird hinter dem Hintergrund dargestellt.

Wir entscheiden uns dafür, das Sprite vor dem Hintergrund darzustellen und setzen daher das Bit 0 auf den Wert 0.

```
100 POKE 53275,PEEK(53275) AND (NOT(1))
```

Sprite aktivieren

Nun müssen wir unser Sprite nur noch einschalten, damit es auch auf dem Bildschirm angezeigt wird.

In der Speicherstelle 53269 ist jedes der 8 Bits mit einem Sprite verbunden, Bit 0 mit Sprite 0 bis hin zu Bit 7, welches mit Sprite 7 verbunden ist. Setzt man ein Bit auf den Wert 1, dann wird das Sprite, das mit diesem Bit verbunden ist, angezeigt. Setzt man es umgekehrt auf den Wert 0, dann verschwindet das jeweilige Sprite.

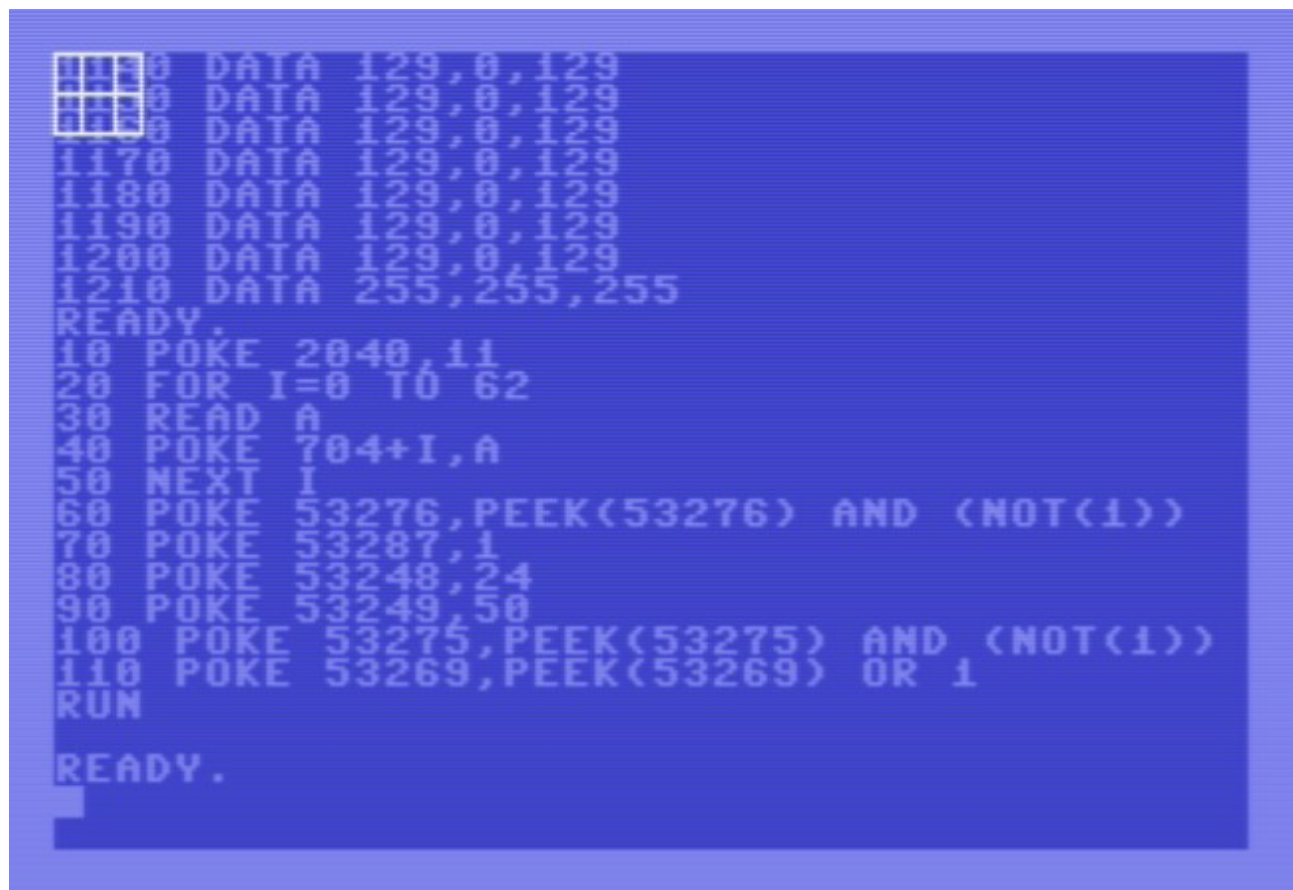
Wichtig:

Durch Aktivieren des Sprites wird das Sprite zwar grundsätzlich sichtbar gemacht, das bedeutet jedoch nicht, dass es sich gerade auch im sichtbaren Bereich auf dem Bildschirm befindet. Es kann je nach Koordinate beispielsweise vom Rahmen teilweise oder ganz verdeckt werden.

Setzen wir also Bit 0 in dieser Speicherstelle auf den Wert 1:

```
110 POKE 53269,PEEK(53269) OR 1
```

Wenn wir das Programm nun mit RUN starten, dann sollte folgendes zu sehen sein.



```
10 DATA 129,0,129
20 DATA 129,0,129
30 DATA 129,0,129
40 DATA 129,0,129
50 DATA 129,0,129
60 DATA 129,0,129
70 DATA 255,255,255
80 READY.
90 POKE 2040,11
100 FOR I=0 TO 62
110 READ A
120 POKE 704+I,A
130 NEXT I
140 POKE 53276,PEEK(53276) AND (NOT(1))
150 POKE 53287,1
160 POKE 53248,24
170 POKE 53249,50
180 POKE 53275,PEEK(53275) AND (NOT(1))
190 POKE 53269,PEEK(53269) OR 1
200 RUN
210 READY.
```

Es hat also soweit alles funktioniert und wir haben unser erstes Sprite auf dem Bildschirm dargestellt.

Wählen wir doch mal eine andere Farbe, z.B. Gelb (Farbcode 7) und geben gleich im Direktmodus den Befehl

```
POKE 53287,7
```

ein.

Das Sprite sollte nun in gelb angezeigt werden.

Lassen wir unser Sprite mal verschwinden? Aber sicher, das funktioniert mit dem Befehl

```
POKE 53269,PEEK(53269) AND (NOT(1))
```

Das Sprite sollte nun verschwunden sein.

Sichtbar machen können wir es wieder mit dem Befehl

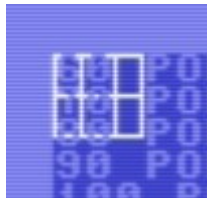
```
POKE 53269,PEEK(53269) OR 1
```

Das Sprite sollte nun wieder zu sehen sein.

Legen wir es doch mal hinter den Hintergrund, dazu ist der Befehl

```
POKE 53275,PEEK(53275) OR 1
```

nötig.



Nun befinden sich die BASIC-Zeilennummern im Vordergrund und überdecken das Sprite an manchen Stellen.

Hier zum Vergleich die vorherige Anzeige, bei der das Sprite im Vordergrund liegt und die Zeilennummern an manchen Stellen verdeckt.



Experimentieren wir nun ein wenig mit der Position des Sprites.

Verändern wir doch mal die X-Koordinate auf den Wert 100, was über den Befehl

```
POKE 53248,100
```

möglich ist.



Wichtig:

Wenn wir für unser Sprite eine X-Koordinate größer als 255 wählen, dann müssen wir hier einen anderen Weg einschlagen, denn in einem Byte kann man ja nur Werte zwischen 0 und 255 ablegen.

Hier sehen Sie die Position bei einer X-Koordinate von 255, also die höchstmögliche X-Koordinate, die in der Speicherstelle 53248 möglich ist.

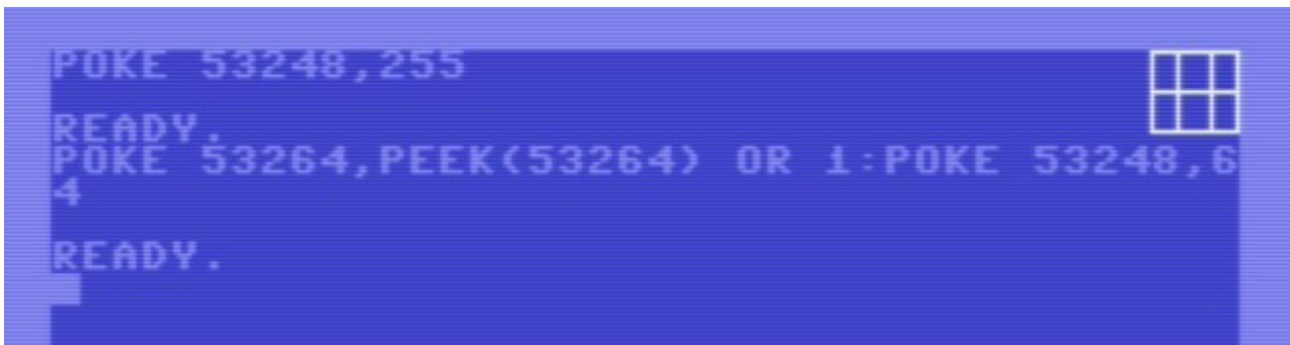


Wollen wir das Sprite an den rechten Rand des sichtbaren Bereichs positionieren, also auf die Position 320, dann müssen wir die Speicherstelle 53264 zu Hilfe nehmen.

Auch diese Speicherstelle enthält nach dem bereits erwähnten Schema für jedes Sprite ein eigenes Bit. Dieses Bit dient als zusätzliches Bit für die Darstellung von X-Koordinaten, welche größer als 255 sind und hat in Bezug auf die X-Koordinate die Wertigkeit 256.

Wir wollen das Sprite auf die X-Koordinate 320 setzen, d.h. wir müssen das Bit 0 (für das Sprite 0) in der Speicherstelle 53264 auf den Wert 1 setzen und den Rest, also was vom Wert 256 noch auf den Wert 320 fehlt, schreiben wir wie gehabt in die Speicherstelle 53248.

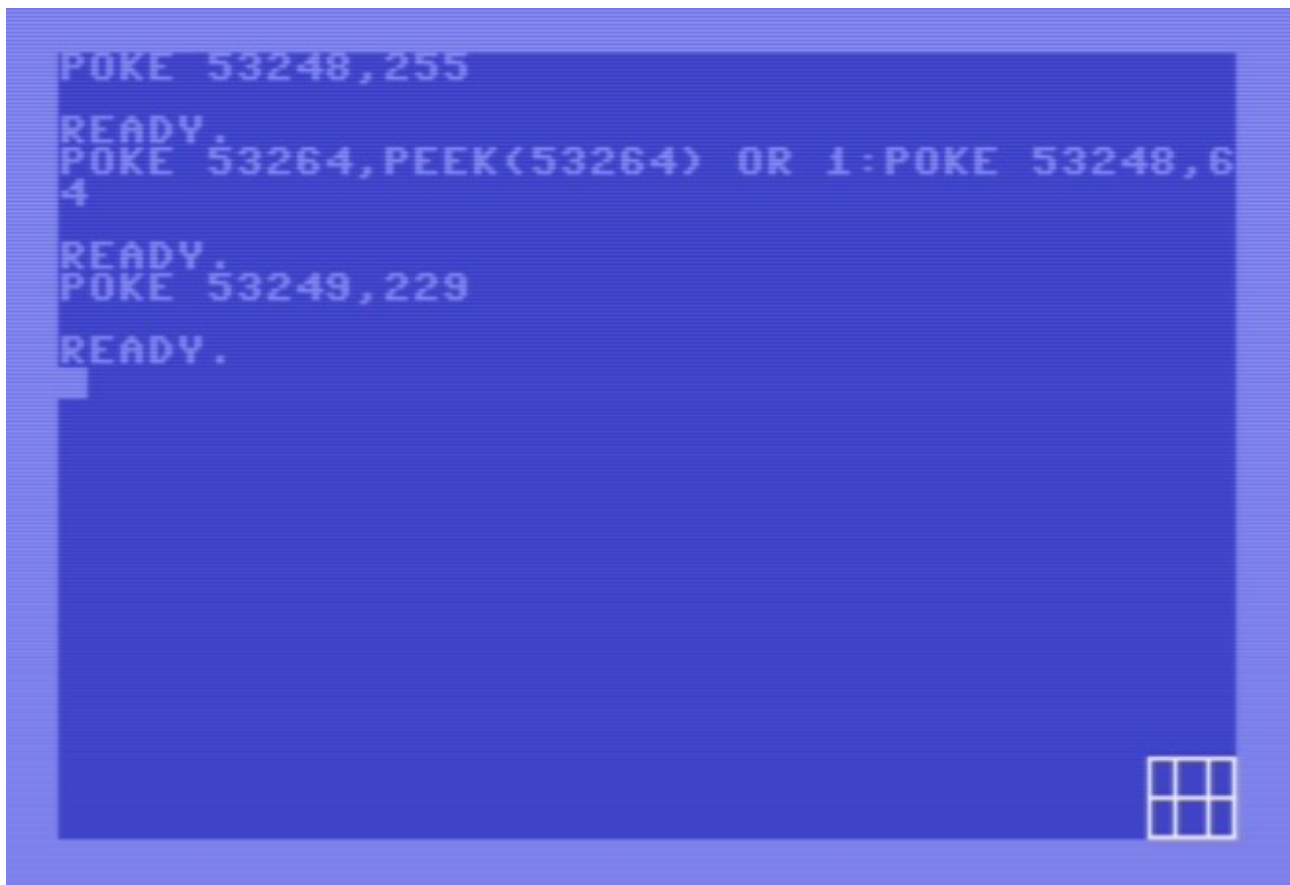
```
POKE 53264,PEEK(53264) OR 1:POKE 53248,64
```



Wichtig ist hier, dass wir das Bit 0 in Speicherstelle 53264 wieder auf 0 setzen, wenn wir die X-Koordinate auf einen Wert zwischen 0 und 255 setzen wollen.

Dies funktioniert mit dem Befehl `POKE 53264,PEEK(53264) AND (NOT(1))`

Nun verschieben wir noch mit dem Befehl `POKE 53249,229` das Sprite an den unteren Rand des sichtbaren Bildschirmbereichs.



Wir haben auch die Möglichkeit, das Sprite sowohl in horizontaler als auch in vertikaler Richtung zu vergrößern. Die Auflösung wird dadurch nicht verdoppelt, das Sprite wird nur doppelt so breit oder hoch dargestellt.

Für eine horizontale Vergrößerung ist die Speicherstelle 53277 zuständig. Auch hier ist jedes Sprite mit einem eigenen Bit vertreten. Setzt man es auf den Wert 1, so wird das Sprite in horizontaler Richtung verdoppelt. Setzt man es umgekehrt auf den Wert 0, so wird das Sprite in Normalgröße angezeigt.

Hier eine Vergrößerung des Sprites in horizontaler Richtung:



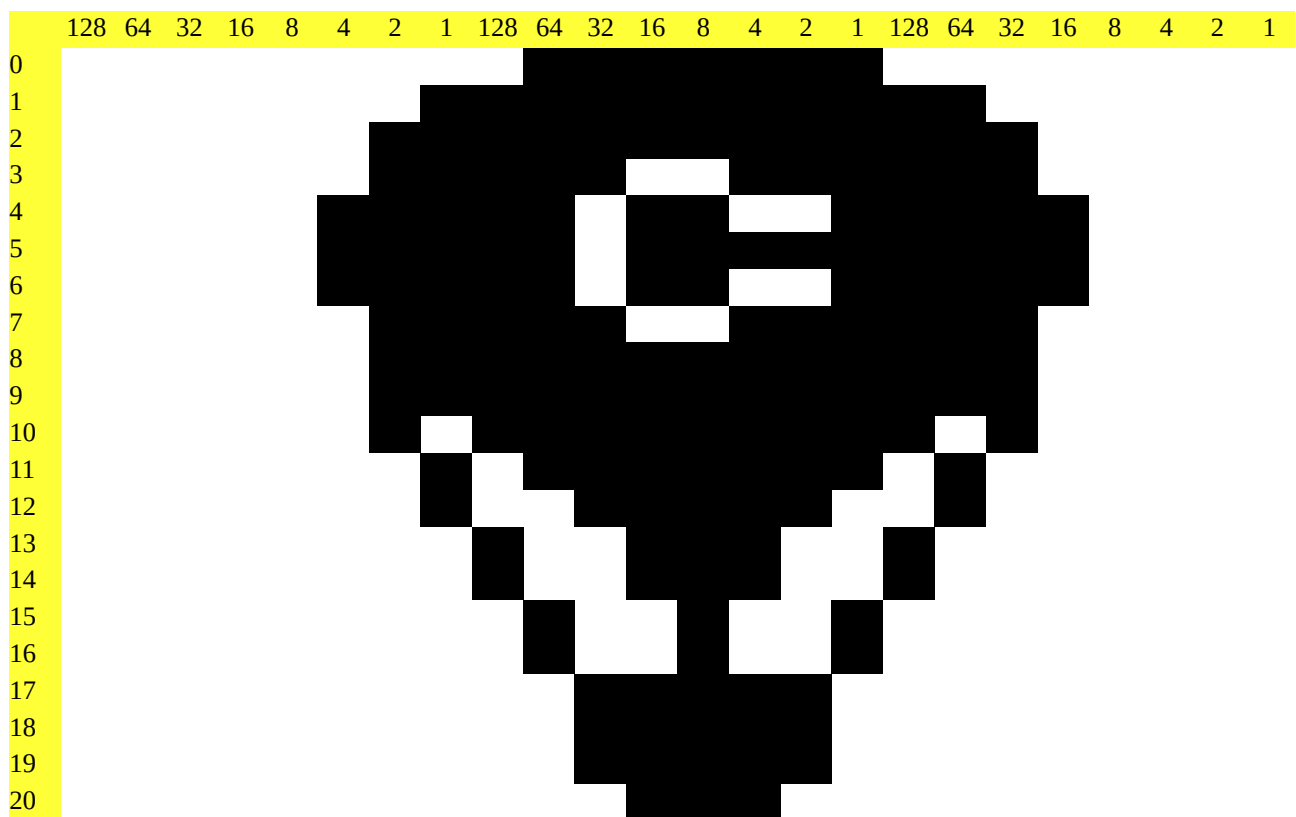
Für eine vertikale Vergrößerung ist die Speicherstelle 53271 zuständig. Auch hier ist wieder jedes Sprite mit einem eigenen Bit vertreten. Setzt man es auf den Wert 1, so wird das Sprite in vertikaler Richtung verdoppelt. Setzt man es umgekehrt auf den Wert 0, so wird das Sprite in Normalgröße angezeigt.

Hier eine Vergrößerung des Sprites in vertikaler Richtung:



Fügen wir nun ein weiteres Sprite hinzu.

Aus dem Handbuch des C64 kennen Sie sicherlich diesen Ballon, der dort als Beispiel für die Sprite-Programmierung verwendet wird. Bauen wir uns diesen doch mal nach.



Damit Sie sich nicht die Mühe machen brauchen, habe ich hier gleich die Tabelle mit den entsprechenden Bytes für Sie.

| Erstes Byte | Zweites Byte | Drittes Byte |
|-------------|--------------|--------------|
| 0 | 127 | 0 |
| 1 | 255 | 192 |
| 3 | 255 | 224 |
| 3 | 231 | 224 |
| 7 | 217 | 240 |
| 7 | 223 | 240 |
| 7 | 217 | 240 |
| 3 | 231 | 224 |
| 3 | 255 | 224 |
| 3 | 255 | 224 |
| 2 | 255 | 160 |
| 1 | 127 | 64 |
| 1 | 62 | 64 |
| 0 | 156 | 128 |
| 0 | 156 | 128 |
| 0 | 73 | 0 |
| 0 | 73 | 0 |
| 0 | 62 | 0 |
| 0 | 62 | 0 |
| 0 | 62 | 0 |
| 0 | 28 | 0 |

Wie bei unserem ersten Sprite legen wir diese Daten wieder in DATA-Zeilen ab.

```

1210 DATA 255,255,255
READY.
1220 REM DATEN FUER SPRITE 1
1230 DATA 0,127,0
1240 DATA 1,255,192
1250 DATA 3,255,224
1260 DATA 3,231,224
1270 DATA 7,217,240
1280 DATA 7,223,240
1290 DATA 7,217,240
1300 DATA 3,231,224
1310 DATA 3,255,224
1320 DATA 3,255,224
1330 DATA 2,255,160
1340 DATA 1,127,64
1350 DATA 1,62,64
1360 DATA 0,156,128
1370 DATA 0,156,128
1380 DATA 0,73,0
1390 DATA 0,73,0
1400 DATA 0,62,0
1410 DATA 0,62,0
1420 DATA 0,62,0
1430 DATA 0,28,0

```

Dann führen wir exakt dieselben Schritte durch, die wir auch beim ersten Sprite durchgeführt haben.

Als Speicherort werden wir für unser zweites Sprite den Block Nummer 13 verwenden, denn wie bereits erwähnt, stehen die Blöcke 13, 14 und 15 zur freien Verfügung.

Wir ergänzen also folgende Zeile:

```
120 POKE 2041,13
```

Da wir dieses mal ja Sprite 1 meinen, müssen wir hier die Speicherstelle 2041 verwenden.

Nun kopieren wir die Spritedaten in den Block Nummer 13, dieser hat die Startadresse 832.

```
130 FOR I=0 TO 62  
140 READ A  
150 POKE 832+I,A  
160 NEXT I
```

Typ des Sprites festlegen (einfarbig oder mehrfarbig)

Da es sich wieder um ein einfarbigen Sprite handelt, setzen wir das Bit an der Position 1 auf den Wert 0. Dadurch wird das Sprite 1 als einfarbig markiert.

Dazu brauchen wir folgende UND-Verknüpfung:

```
170 POKE 53276,PEEK(53276) AND (NOT (2))
```

Farbe des Sprites festlegen

Wir wählen für Sprite 1 die Farbe Gelb, also müssen wir den Wert 7 in die Speicherstelle 53288 schreiben.

```
180 POKE 53288,7
```

Festlegen der Spriteposition

Nehmen wir für dieses Sprite die X-Koordinate 160 und die Y-Koordinate 140.

```
190 POKE 53250,160  
200 POKE 53251,140
```

Festlegen der Sprite-Priorität in Bezug auf den Hintergrund

Wir entscheiden uns auch bei diesem Sprite dafür, dass das Sprite vor dem Hintergrund dargestellt wird und wählen daher den Wert 0 für das Bit 1 in der Speicherstelle 53275.

```
210 POKE 53275,PEEK(53275) AND (NOT(2))
```

Sprite aktivieren

Setzen wir also Bit 1 in der Speicherstelle 53269 auf den Wert 1.

```
220 POKE 53269,PEEK(53269) OR 2
```

Nun starten wir das Programm mit RUN und es wird nun auch das zweite Sprite angezeigt.

```
100  FOR I=0 TO 62
110  READ A
120  POKE 704+I,A
130  NEXT I
140  POKE 53276,PEEK(53276) AND (NOT(1))
150  POKE 53287,1
160  POKE 53248,24
170  POKE 53249,50
180  POKE 53275,PEEK(53275) AND (NOT(1))
190  POKE 53269,PEEK(53269) OR 1
200  POKE 2041,13
210  FOR I=0 TO 62
220  READ A
230  POKE 832+I,A
240  POKE 53276,PEEK(53276) AND (NOT(2))
250  POKE 53288,7
260  POKE 53250,160
270  POKE 53251,140
280  POKE 53275,PEEK(53275) AND (NOT(2))
290  POKE 53269,PEEK(53269) OR 2
300  REM DATEN FUER SPRITE 0
READY.
```



Nun bauen wir uns noch ein drittes Sprite, das Zeichnen ist dieses mal sehr einfach, da es die komplette Fläche ausfüllt.

| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|----|-----|----|----|----|---|---|---|---|-----|----|----|----|---|---|---|---|-----|----|----|----|---|---|---|---|
| 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | | | | | | | | | | | | | | | |

Das macht die Ermittlung der Werte für die DATA-Zeilen natürlich auch recht einfach.
Wir benötigen für die 21 DATA-Zeilen immer denselben Inhalt 255,255,255.



So, nun wollen wir das alles mal in Assembler umsetzen. Logische Verknüpfungen sind hier sehr stark vertreten. Wenn Sie diesbezüglich fit sind, sollten Sie keine Probleme damit haben, den Code nachvollziehen zu können.

Ich habe die drei 64 Byte Blöcke für die Spritedaten ab Adresse \$1500 abgelegt. Rechts unten sehen Sie den Wert \$00 welcher den Block für Sprite 2 abschließt. Damit Sie diese vielen Zahlen und auch das Assembler-Programm nicht abtippen müssen, habe ich es auf der Diskette zur Verfügung gestellt.



Hier der Code für Sprite 0:

| | | | | | |
|--------|----|----|----|-----|--------|
| ,15CC0 | A9 | 0B | | LDA | #0B |
| ,15CC2 | 8D | F8 | 07 | STA | 07F8 |
| ,15CC5 | A2 | 00 | | LDX | #00 |
| ,15CC7 | BD | 00 | 15 | LDA | 1500,X |
| ,15CCA | 9D | C0 | 02 | STA | 02C0,X |
| ,15CCD | E8 | | | INX | |
| ,15CCE | E0 | 40 | | CPX | #40 |
| ,15CD0 | D0 | F5 | | BNE | 15C7 |
| ,15CD2 | A9 | 01 | | LDA | #01 |
| ,15CD4 | 49 | FF | | EOR | #FF |
| ,15CD6 | 2D | 1C | D0 | AND | D01C |
| ,15CD9 | 8D | 1C | D0 | STA | D01C |
| ,15CDC | A9 | 01 | | LDA | #01 |
| ,15CDE | 8D | 27 | D0 | STA | D027 |
| ,15FE1 | A9 | 18 | | LDA | #18 |
| ,15FE3 | 8D | 00 | D0 | STA | D000 |
| ,15FE6 | A9 | 32 | | LDA | #32 |
| ,15FE8 | 8D | 01 | D0 | STA | D001 |
| ,15FEB | A9 | 01 | | LDA | #01 |
| ,15FED | 49 | FF | | EOR | #FF |
| ,15FEF | 2D | 1B | D0 | AND | D01B |
| ,15FF2 | 8D | 1B | D0 | STA | D01B |
| ,15FF5 | A9 | 01 | | LDA | #01 |
| ,15FF7 | 0D | 15 | D0 | ORA | D015 |
| ,15FFA | 8D | 15 | D0 | STA | D015 |

Hier der Code für Sprite 1:

| | | | | | |
|--------|----|----|----|-----|--------|
| ,15FFD | A9 | 0D | | LDA | #0D |
| ,15FFF | 8D | F9 | 07 | STA | 07F9 |
| ,16002 | A2 | 00 | | LDX | #00 |
| ,16004 | BD | 40 | 15 | LDA | 1540,X |
| ,16007 | 9D | 40 | 03 | STA | 0340,X |
| ,1600A | E8 | | | INX | |
| ,1600B | E0 | 40 | | CPX | #40 |
| ,1600D | D0 | F5 | | BNE | 1604 |
| ,1600F | A9 | 02 | | LDA | #02 |
| ,16111 | 49 | FF | | EOR | #FF |
| ,16113 | 2D | 1C | D0 | AND | D01C |
| ,16116 | 8D | 1C | D0 | STA | D01C |
| ,16119 | A9 | 07 | | LDA | #07 |
| ,1611B | 8D | 28 | D0 | STA | D028 |
| ,1611E | A9 | A0 | | LDA | #A0 |
| ,16220 | 8D | 02 | D0 | STA | D002 |
| ,16223 | A9 | 8C | | LDA | #8C |
| ,16225 | 8D | 03 | D0 | STA | D003 |
| ,16228 | A9 | 02 | | LDA | #02 |
| ,1622A | 49 | FF | | EOR | #FF |
| ,1622C | 2D | 1B | D0 | AND | D01B |
| ,1622F | 8D | 1B | D0 | STA | D01B |
| ,16332 | A9 | 02 | | LDA | #02 |
| ,16334 | 0D | 15 | D0 | ORA | D015 |
| ,16337 | 8D | 15 | D0 | STA | D015 |

Hier der Code für Sprite 2:

| | | | | | |
|-------|----|----|----|-----|--------|
| ,163A | A9 | 0E | | LDA | #0E |
| ,163C | 8D | FA | 07 | STA | 07FA |
| ,163F | A2 | 00 | | LDX | #00 |
| ,1641 | BD | 80 | 15 | LDA | 1580,X |
| ,1644 | 9D | 80 | 03 | STA | 0380,X |
| ,1647 | E8 | | | INX | |
| ,1648 | E0 | 40 | | CPX | #40 |
| ,164A | D0 | F5 | | BNE | 1641 |
| ,164C | A9 | 04 | | LDA | #04 |
| ,164E | 49 | FF | | EOR | #FF |
| ,1650 | 2D | 1C | D0 | AND | D01C |
| ,1653 | 8D | 1C | D0 | STA | D01C |
| ,1656 | A9 | 03 | | LDA | #03 |
| ,1658 | 8D | 29 | D0 | STA | D029 |
| ,165B | A9 | 64 | | LDA | #64 |
| ,165D | 8D | 04 | D0 | STA | D004 |
| ,1660 | 8D | 05 | D0 | STA | D005 |
| ,1663 | A9 | 04 | | LDA | #04 |
| ,1665 | 0D | 15 | D0 | ORA | D015 |
| ,1668 | 8D | 15 | D0 | STA | D015 |
| ,166B | 60 | | | RTS | |

Wechseln Sie mit X zurück nach BASIC und starten das Programm mit SYS 5568.

Als erstes fällt sofort der enorme Unterschied in der Geschwindigkeit auf. Die Sprites werden nach der Eingabe von SYS 5568 augenblicklich angezeigt. Bei der BASIC-Version dauerte das um einiges länger.

Ich werde den Code anhand von Sprite 0 erklären, der Code für Sprite 1 und Sprite 2 ist bis auf die unterschiedlichen Einstellungen für Blocknummer, Farbe, Position usw. identisch.

\$15C0 - \$15C2

```
LDA #0B  
STA $07FB
```

Ist das Gegenstück zu

```
POKE 2040,11
```

\$15C5 - \$15D0

```
LDX #00  
LDA 1500,X  
STA 02C0,X  
INX  
CPX #40  
BNE zu LDA Befehl
```

Eine Schleife, welche die 63 Bytes an Daten für Sprite 0, die sich ab Adresse \$1500 im Speicher befinden, in den Block 11 (Startadresse \$02C0, dezimal 704) umkopiert.

Sie ist das Gegenstück zur BASIC-Schleife, welche die Daten aus den DATA-Zeilen in den Block 11 umkopiert.

\$15D2 - \$15D9 (Einstellung dass Sprite 0 einfärbig ist)

```
LDA #01  
EOR #FF  
AND D01C  
STA D01C
```

Ist das Gegenstück zu

```
POKE 53276,PEEK(53276) AND (NOT(1))
```

In Assembler sind hier die beiden Zahlen vertauscht, d.h. zuerst wird der Akkumulator mit dem Wert \$01 geladen, dann durch Anwendung von EOR #FF das Einerkomplement gebildet, dieses dann durch AND mit dem Inhalt der Speicherstelle D01C (dezimal 53276) verknüpft und durch STA D01C wird das Ergebnis wieder zurück in die Speicherstelle D01C geschrieben.

In BASIC würde man dies so schreiben:

```
POKE 53276,(NOT (1)) AND PEEK(53276)
```

\$15DC - \$15DE (Farbe Weiß einstellen)

```
LDA #01  
STA D027
```

Ist das Gegenstück zu

```
POKE 53287,1
```

\$15E1 - \$15E3 (X-Koordinate auf den Wert 24 einstellen)

```
LDA #18  
STA D000
```

Ist das Gegenstück zu

```
POKE 53248,24
```

\$15E6 - \$15E8 (Y-Koordinate auf den Wert 50 einstellen)

```
LDA #32  
STA D001
```

Ist das Gegenstück zu

```
POKE 53249,50
```

\$15EB - \$15F2 (Einstellen, dass Sprite vor dem Hintergrund liegt)

```
LDA #01  
EOR #FF  
AND D01B  
STA D01B
```

Ist das Gegenstück zu

```
POKE 53275,PEEK(53275) AND (NOT(1))
```

Die AND-Verknüpfung funktioniert analog wie vorhin bei der Speicherstelle 53276.

\$15F5 - \$15FA (Sprite aktivieren)

```
LDA #01  
ORA D015  
STA D015
```

Ist das Gegenstück zu

```
POKE 53269,PEEK(53269) OR 1
```

Hier ist es ähnlich wie vorhin bei der AND-Verknüpfung. Die beiden Zahlen sind vertauscht, d.h. zuerst wird der Akkumulator mit dem Wert 1 geladen, dieser dann durch OR mit dem Inhalt der Speicherstelle D015 (dezimal 53269) verknüpft und durch STA D015 wird das Ergebnis wieder zurück in die Speicherstelle D015 geschrieben.

Sprite-Priorität

Ich habe Sprite 0 auf die Position 87,87 verschoben und Sprite 1 auf die Position 110,110 damit man die Prioritäten der Sprites erkennen kann.



Hier sieht man, dass Sprite 2 (Quadrat) sowohl von Sprite 0 (Gitter) als auch von Sprite 1 (Ballon) überdeckt wird.

Die weißen Linien des Gitters überdecken die türkisen Stellen des Quadrats.

Das liegt daran, dass Sprite 0 die höchste Priorität hat. Das geht weiter bis Sprite 7 mit der niedrigsten Priorität.

Die weißen Linien des Gitters würden auch den Ballon stellenweise überdecken, weil die Priorität des Gitters höher ist.

Je niedriger die Nummer des Sprites ist, desto höher ist die Priorität gegenüber den Sprites mit höheren Nummern.

Diese Prioritäten kann man nicht verändern. Sprite 0 wird also immer die höchste und Sprite 7 immer die niedrigste Priorität haben. Es ist also nicht möglich, beispielsweise Sprite 2 eine höhere Priorität als Sprite 1 zu geben.

Die Priorität der Sprites untereinander ist nicht zu verwechseln mit der Priorität, welche die einzelnen Sprites in Bezug auf den Hintergrund haben.

Auf dem Bild sieht man, dass alle Sprites den BASIC-Code verdecken, weil wir dies bei jedem Sprite über die Speicherstelle 53275 so eingestellt haben.

Mehrfarbige Sprites

Bei den mehrfarbigen Sprites reduziert sich die horizontale Auflösung auf die Hälfte, also auf 12 Pixel, da jeweils zwei Bits für die Farbeinstellung eines Pixels gebraucht werden. Hierbei wird in den beiden Bits jedoch nicht direkt eine Farbe angegeben (dafür wären 4 Bits nötig wenn man alle 16 Farben abbilden will), sondern die beiden Bits bilden eine Bitkombination, welche folgende mögliche Werte darstellen kann:

| Bitkombination | Farbinformation |
|----------------|--|
| 00 | Hintergrund (transparent) |
| 10 | Spritefarbe (Register 53287 - 53294) |
| 01 | Mehrfarbenregister #0 (Register 53285) |
| 11 | Mehrfarbenregister #1 (Register 53286) |

Hier ein Beispiellaster für ein mehrfarbiges Sprite:

| | 128/64 | 32/16 | 8/4 | 2/1 | 128/64 | 32/16 | 8/4 | 2/1 | 128/64 | 32/16 | 8/4 | 2/1 |
|----|--------|-------|-----|-----|--------|-------|-----|-----|--------|-------|-----|-----|
| 0 | 00 | 00 | 00 | 00 | 10 | 10 | 10 | 10 | 00 | 00 | 00 | 00 |
| 1 | 00 | 00 | 00 | 10 | 10 | 10 | 10 | 10 | 10 | 00 | 00 | 00 |
| 2 | 00 | 00 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 00 | 00 |
| 3 | 00 | 00 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 00 | 00 |
| 4 | 00 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 00 |
| 5 | 00 | 10 | 10 | 11 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 00 |
| 6 | 00 | 10 | 11 | 11 | 11 | 10 | 10 | 11 | 11 | 11 | 10 | 00 |
| 7 | 10 | 10 | 11 | 11 | 11 | 10 | 10 | 11 | 11 | 11 | 10 | 10 |
| 8 | 10 | 10 | 11 | 01 | 11 | 10 | 10 | 11 | 01 | 11 | 10 | 10 |
| 9 | 10 | 10 | 11 | 01 | 11 | 10 | 10 | 11 | 01 | 11 | 10 | 10 |
| 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 |
| 11 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 12 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 13 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 14 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 15 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 16 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| 17 | 10 | 10 | 00 | 10 | 10 | 00 | 10 | 10 | 10 | 00 | 10 | 10 |
| 18 | 10 | 10 | 00 | 10 | 10 | 00 | 00 | 10 | 10 | 00 | 10 | 10 |
| 19 | 10 | 00 | 00 | 00 | 10 | 00 | 00 | 10 | 00 | 00 | 00 | 10 |
| 20 | 10 | 00 | 00 | 00 | 10 | 00 | 00 | 10 | 00 | 00 | 00 | 10 |

Enthält eine Zelle die Bitkombination 00, dann enthält das Sprite an dieser Stelle keinen Pixel, d.h. hier scheint der Hintergrund durch (in dem Beispiel sind dies die schwarzen Zellen).

Enthält eine Zelle die Bitkombination 10, dann enthält der Pixel an dieser Stelle jene Farbe, welche in dem Farbregister, welches dem Sprite zugeordnet ist, hinterlegt ist. Dies sind dieselben Register wie bei den einfärbigen Sprites, d.h. Register 53287 enthält die Farbinformation für Sprite 0,

Register 53288 enthält die Farbinformation für Sprite 1 usw. bis hin zum Register 53294, welches die Farbinformation für Sprite 7 enthält. In dem Beispiel sind das die grünen Zellen.

Enthält eine Zelle die Bitkombination 01, dann enthält das Sprite an dieser Stelle jene Farbe, welche in dem Mehrfarbenregister #0 (53285) hinterlegt ist. In dem Beispiel sind das die blauen Zellen.

Enthält eine Zelle die Bitkombination 11, dann enthält das Sprite an dieser Stelle jene Farbe, welche in dem Mehrfarbenregister #1 (53286) hinterlegt ist. In dem Beispiel sind das die weißen Zellen.

Die Farben, die in diesen beiden Registern hinterlegt sind, gelten für alle Sprites.

Das heißt, wenn zwei Sprites an derselben Position die Bitkombination 01 oder 11 enthalten, dann haben sie an dieser Stelle auch dieselbe Farbe. Je nach Bitkombination entweder jene aus dem Register 53285 (01) oder jene aus dem Register 53286 (11)

Die Pixel haben im Vergleich zu einfarbigen Sprites jedoch die doppelte Breite, d.h. die sichtbare Breite des Sprites ändert sich trotzdem nicht (24 einfach breite Pixel sind gleich breit wie 12 doppelt breite Pixel)

In horizontaler Richtung haben wir nun nur mehr 12 Zellen, in der vertikalen Richtung hat sich nichts verändert, d.h. es sind hier nach wie vor 21 Zeilen.

Jede Zelle repräsentiert nun jedoch 2 Bits, die jeweils eine der oben genannten Kombinationen annehmen können.

Bei den einfarbigen Sprites entsprach jede Zelle einem Bit (Pixel oder kein Pixel)

Das heißt also, dass wir insgesamt trotzdem wieder auf 24 Bits, also 3 Bytes kommen, weil ja jede der 12 Zellen 2 Bits beinhaltet.

Eine Zeile ist also nach wie vor durch drei Bytes definiert, nur der Inhalt der Bytes wird bei mehrfarbigen Sprites anders interpretiert.

Für die obige Figur lauten die Byte-Werte für die Zeilen:

| Erstes Byte | Zweites Byte | Drittes Byte |
|-------------|--------------|--------------|
| 0 | 170 | 0 |
| 2 | 170 | 128 |
| 10 | 170 | 160 |
| 10 | 170 | 160 |
| 42 | 170 | 168 |
| 43 | 170 | 232 |
| 47 | 235 | 248 |
| 175 | 235 | 250 |
| 173 | 235 | 122 |
| 173 | 235 | 122 |
| 171 | 170 | 234 |
| 170 | 170 | 170 |
| 170 | 170 | 170 |
| 170 | 170 | 170 |
| 170 | 170 | 170 |
| 170 | 170 | 170 |
| 170 | 170 | 170 |
| 162 | 138 | 138 |
| 162 | 130 | 138 |
| 128 | 130 | 2 |
| 128 | 130 | 2 |

Gut, dann erstellen wir doch gleich mal ein Programm mit diesem mehrfarbigen Sprite.
Zuerst erfassen wir die Daten aus der Tabelle in DATA-Zeilen:

```

1000 REM DATEN FUER SPRITE
1010 DATA 0,170,0
1020 DATA 2,170,128
1030 DATA 10,170,160
1040 DATA 10,170,160
1050 DATA 42,170,168
1060 DATA 43,170,232
1070 DATA 47,235,248
1080 DATA 175,235,250
1090 DATA 173,235,122
1100 DATA 173,235,122
1110 DATA 171,170,234
1120 DATA 170,170,170
1130 DATA 170,170,170
1140 DATA 170,170,170
1150 DATA 170,170,170
1160 DATA 170,170,170
1170 DATA 170,170,170
1180 DATA 162,138,138
1190 DATA 162,130,138
1200 DATA 128,130,2
1210 DATA 128,130,2
READY.

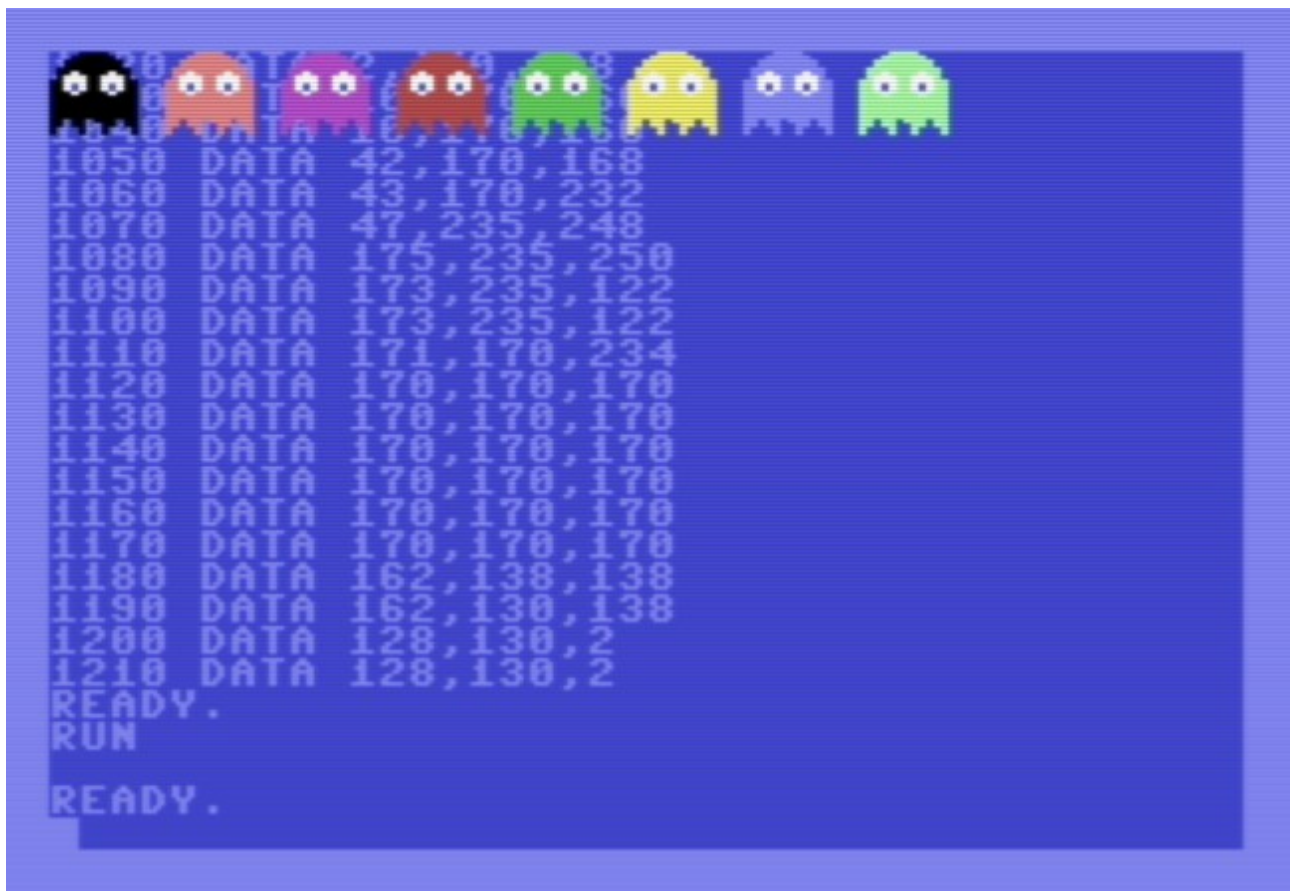
```

Was soll unser Programm machen?

Es soll 8 Sprites, welche sich nur in der Farbgebung unterscheiden, am oberen Rand des Bildschirms anzeigen. Da sich die Sprites nur farblich unterscheiden, ist es nicht nötig, für jedes Sprite eigene Spritedaten zu definieren, sondern wir können für alle Sprites denselben Datenblock nutzen (hier Block Nummer 11) und das individuelle Aussehen der Sprites über die Farbgebung steuern.

Dann sollen sich diese 8 Sprites vom oberen zum unteren Rand des Bildschirms bewegen und danach wieder zurück in die Ausgangsposition am oberen Rand.

Hier die 8 Geister am Ausgangspunkt bzw. nachdem sie wieder an den oberen Bildschirmrand zurückgekehrt sind:



Gehen wir das BASIC-Programm nun Schritt für Schritt durch.

```
LIST 10-170
10 REM BLOCKNUMMER EINTRAGEN
20 FOR I=2040 TO 2047
30 POKE I,11
40 NEXT I
50 REM SPRITEDATEN IN BLOCK 11 KOPIEREN
60 FOR I=0 TO 62
70 READ A
80 POKE 704+I,A
90 NEXT I
100 REM ALLE SPRITES SIND MEHRFARBIG
110 POKE 53276,255
120 REM ALLE SPRITES VOR HINTERGRUND
130 POKE 53275,0
140 REM ERSTE GEMEINSAME FARBE
150 POKE 53285,6
160 REM ZWEITE GEMEINSAME FARBE
170 POKE 53286,1
READY.
```

Zeile 10 – 40

Hier wird für jedes Sprite dieselbe Blocknummer (11) eingetragen, da ja jedes Sprite gleich aussieht und die Unterschiede nur in der Farbgebung bestehen.

Zeile 50 – 90

Hier werden die Spritedaten aus den DATA-Zeilen in den Speicherblock 11 kopiert.

Zeile 100 – 110

Da alle Sprites mehrfarbig sind, können wir in Speicherstelle 53276 alle 8 Bits auf den Wert 1 setzen, also den Wert 255 dort eintragen.

Zeile 120 – 130

Alle Sprites sollen sich vor dem Hintergrund befinden, also höhere Priorität als der Hintergrund haben. Daher setzen wir alle 8 Bits auf den Wert 0 und tragen den Wert 0 in die Speicherstelle 53275 ein.

Zeile 140 – 150

Hier wird die erste Farbe, welchen allen Sprites gemeinsam ist, eingestellt. In unserem Beispiel hier ist das die Farbe Blau (Farbcode 6). Dies ist die Augenfarbe der Geister.

Zeile 160 – 170

Hier wird die zweite Farbe, welchen allen Sprites gemeinsam ist, eingestellt. In unserem Beispiel hier ist das die Farbe Weiß (Farbcode 1). Dies ist „das Weiße“ im Auge der Geister.

Zeile 180 – 330

Hier wird die individuelle Farben für jedes der 8 Sprites eingestellt.

```
LIST 180-330
180 REM FARBE FUER SPRITE 0
190 POKE 53287,0
200 REM FARBE FUER SPRITE 1
210 POKE 53288,10
220 REM FARBE FUER SPRITE 2
230 POKE 53289,4
240 REM FARBE FUER SPRITE 3
250 POKE 53290,2
260 REM FARBE FUER SPRITE 4
270 POKE 53291,5
280 REM FARBE FUER SPRITE 5
290 POKE 53292,7
300 REM FARBE FUER SPRITE 6
310 POKE 53293,14
320 REM FARBE FUER SPRITE 7
330 POKE 53294,13

READY.
```

Zeile 340 – 490

Hier werden sowohl die X- als auch die Y – Koordinate für jedes der 8 Sprites eingestellt. Zu Beginn befinden sich alle 8 Sprites am oberen Bildschirmrand, daher ist zu Beginn die Y – Koordinate bei allen Sprites gleich.


```
LIST 340-510
```

```
340 REM POSITION VON SPRITE 0
350 POKE 53248,24:POKE 53249,50
360 REM POSITION VON SPRITE 1
370 POKE 53250,55:POKE 53251,50
380 REM POSITION VON SPRITE 2
390 POKE 53252,86:POKE 53253,50
400 REM POSITION VON SPRITE 3
410 POKE 53254,117:POKE 53255,50
420 REM POSITION VON SPRITE 4
430 POKE 53256,148:POKE 53257,50
440 REM POSITION VON SPRITE 5
450 POKE 53258,179:POKE 53259,50
460 REM POSITION VON SPRITE 6
470 POKE 53260,210:POKE 53261,50
480 REM POSITION VON SPRITE 7
490 POKE 53262,241:POKE 53263,50
500 REM ALLE SPRITES AKTIVIEREN
510 POKE 53269,255
```

```
READY.
```

Zeile 500 – 510

Hier werden alle Sprites aktiviert, also auf den Positionen, die wir eingestellt haben, angezeigt. Da wir alle 8 Sprites aktivieren wollen, ist es nicht nötig, jedes einzelne zu aktivieren, sondern wir können alle Sprites in einem Schwung sichtbar machen, in dem wir alle 8 Bits in der Speicherstelle 53269 auf den Wert 1 setzen, also den Wert 255 dorthin schreiben.

Zeile 520 – 790

Hier findet die Bewegung der Sprites nach unten bzw. anschließend wieder nach oben statt.

Die Werte für die Y – Koordinate werden jeweils in einer Schleife durchlaufen und durch das Unterprogramm ab Zeile 700 werden die aktuellen Y - Koordinaten für jedes Sprite aktualisiert.


```

LIST 520-790
520 REM SPRITES NACH UNTEN BEWEGEN
530 FOR Y=51 TO 229
540 GOSUB 700
550 NEXT Y
560 REM SPRITES NACH OBEN BEWEGEN
570 FOR Y=228 TO 50 STEP-1
580 GOSUB 700
590 NEXT Y
600 END
700 REM Y KOORDINATEN SETZEN
710 POKE 53249,Y
720 POKE 53251,Y
730 POKE 53253,Y
740 POKE 53255,Y
750 POKE 53257,Y
760 POKE 53259,Y
770 POKE 53261,Y
780 POKE 53263,Y
790 RETURN
READY.

```

Nun wird es wieder spannend, denn wir wollen die Assembler-Version dieses BASIC-Programms in Angriff nehmen.

Ok, was müssen wir als Erstes machen? Richtig, wir müssen zunächst mal die Daten für unser Sprite im Speicher ablegen und von dort dann in den Speicherblock 11 kopieren.

Hier die Spritedaten im Speicher ab Adresse \$1500:

```

.M 1500 1540
:1500 00 AA 00 02 AA 80 0A AA
:1508 A0 0A AA A0 2A AA A8 2B
:1510 AA E8 2F EB F8 AF EB FA
:1518 AD EB 7A AD EB 7A AB AA
:1520 EA AA AA AA AA AA AA AA
:1528 AA AA AA AA AA AA AA AA
:1530 AA AA AA A2 8A 8A A2 82
:1538 8A 80 82 02 80 82 02 00
.■

```

Assembler-Gegenstück für die Zeilen 10 – 40 (Blocknummer 11 für alle Sprites einstellen)

```

.D 1540
,1540 A9 0B LDA #0B
,1542 A2 00 LDX #00
,1544 9D F8 07 STA 07F8,X
,1547 E8 INX
,1548 E0 08 CPX #08
,154A D0 F8 BNE 1544

```

Assembler-Gegenstück für die Zeilen 50 – 90 (Spritedaten in Block 11 kopieren)

```
,154C A2 00 LDX #00
,154E BD 00 15 LDA 1500,X
,1551 9D C0 02 STA 02C0,X
,1554 E8 INX
,1555 E0 3F CPX #3F
,1557 D0 F5 BNE 154E
```

Assembler-Gegenstück für die Zeilen 100 – 110 (Alle 8 Sprites sind mehrfarbig)

```
,1559 A9 FF BNE 154E
,155B 8D 1C D0 LDA #FF
,155D 8D 1C D0 STA D01C
```

Assembler-Gegenstück für die Zeilen 120 – 130 (Alle 8 Sprites vor dem Hintergrund)

```
,155E A9 00 LDA #00
,1560 8D 1B D0 STA D01B
```

Assembler-Gegenstück für die Zeilen 140 – 150 (Erste gemeinsame Farbe einstellen)

```
,1563 A9 06 LDA #06
,1565 8D 25 D0 STA D025
```

Assembler-Gegenstück für die Zeilen 160 – 170 (Zweite gemeinsame Farbe einstellen)

```
,1568 A9 01 LDA #01
,156A 8D 26 D0 STA D026
```

Assembler-Gegenstück für die Zeilen 180 – 330 (Individuelle Farben der Sprites einstellen)

```
,156D A9 00 LDA #00
,156F 8D 27 D0 STA D027
,1572 A9 0A LDA #0A
,1574 8D 28 D0 STA D028
,1577 A9 04 LDA #04
,1579 8D 29 D0 STA D029
,157C A9 02 LDA #02
,157E 8D 2A D0 STA D02A
,1581 A9 05 LDA #05
,1583 8D 2B D0 STA D02B
,1586 A9 07 LDA #07
,1588 8D 2C D0 STA D02C
,158B A9 0E LDA #0E
,158D 8D 2D D0 STA D02D
,1590 A9 0D LDA #0D
,1592 8D 2E D0 STA D02E
```

Assembler-Gegenstück für die Zeilen 340 – 490 (X- und Y – Koordinaten der Sprites einstellen)

```
,1595F 8D 18 D0 LDA #18
,1597 8D 00 D0 STA D000
,1599A 8D 32 D0 LDA #32
,1599C 8D 01 D0 STA D001
,1599F 8D 37 D0 LDA #37
,15A1 8D 02 D0 STA D002
,15A4 8D 32 D0 LDA #32
,15A6 8D 03 D0 STA D003
,15A9 8D 56 D0 LDA #56
,15AB 8D 04 D0 STA D004
,15AE 8D 32 D0 LDA #32
,15B0 8D 05 D0 STA D005
,15B3 8D 75 D0 LDA #75
,15B5 8D 06 D0 STA D006
,15B8 8D 32 D0 LDA #32
,15BA 8D 07 D0 STA D007
,15BD 8D 94 D0 LDA #94
,15BF 8D 08 D0 STA D008
,15C2 8D 32 D0 LDA #32
,15C4 8D 09 D0 STA D009
,15C7 8D B3 D0 LDA #B3
,15C9 8D 0A D0 STA D00A
,15CC 8D 32 D0 LDA #32
,15CE 8D 0B D0 STA D00B
```

```
,15D1 8D 0C D0 STA D00C
,15D3 8D 0C D0 STA D00C
,15D6 8D 32 D0 LDA #32
,15D8 8D 0D D0 STA D00D
,15DB 8D F1 D0 LDA #F1
,15DD 8D 0E D0 STA D00E
,15E0 8D 32 D0 LDA #32
,15E2 8D 0F D0 STA D00F
,15E5 8D FF D0 LDA #FF
```


Assembler-Gegenstück für die Zeilen 500 – 510 (Alle 8 Sprites aktivieren)

```
,15E5 A9 FF D0 LDA #FF
,15E7 8D 15 D0 STA D015
,15EA 4C 0E 16 JMP 160E
```

Der JMP – Befehl an der Adresse \$15EA dient dazu, die beiden folgenden Unterprogramme zu überspringen. Ansonsten würde der Code ja ausgeführt werden.

Zwischen Adresse \$15ED und \$1605 sehen Sie das Assembler-Gegenstück zum BASIC-Unterprogramm von Zeile 700 bis 790.

```
,15E7 8D 15 D0 STA D015
,15EA 4C 0E 16 JMP 160E
-----
,15ED 8C 01 D0 STY D001
,15F0 8C 03 D0 STY D003
,15F3 8C 05 D0 STY D005
,15F6 8C 07 D0 STY D007
,15F9 8C 09 D0 STY D009
,15FC 8C 0B D0 STY D00B
,15FF 8C 0D D0 STY D00D
,1602 8C 0F D0 STY D00F
,1605 60 RTS
-----
,1606 A2 00 LDX #00
,1608 E8 INX
,1609 E0 FF CPX #FF
,160B D0 FB BNE 1608
,160D 60 RTS
-----
```

Zwischen Adresse \$1606 und \$160D liegt ein Unterprogramm, das nur die Aufgabe hat eine Warteschleife zwischen jedem Bewegungsschritt einzulegen. Ansonsten würde man die Bewegung gar nicht als solche wahrnehmen, weil das Assembler-Programm im Vergleich zum BASIC-Programm um ein Vielfaches schneller läuft.

Abschließend noch die Assembler-Gegenstücke zu den beiden Schleifen, welche die Bewegung nach unten bzw. nach oben bewirken.

Zuerst wird das Y Register mit dem Startwert 51 geladen. Dann werden die Sprites durch den Befehl JSR 15ED an der neuen Position angezeigt und durch den Befehl JSR 1606 eine Warteschleife durchlaufen. Sobald diese durchlaufen ist, wird der Inhalt des Y Registers um 1 erhöht und der nächste Durchlauf beginnt. Die Schleife endet, sobald der Inhalt des Y Registers den Wert 229 erreicht hat.

Dann wird das Y Register mit dem Wert 228 (hexadezimal \$E4) geladen und die Bewegung nach oben wird durch laufende Verringerung des Inhalts des Y Registers durchgeführt.

| | | | | | |
|-------|----|----|----|-----|------|
| ,160E | A0 | 33 | | LDY | #33 |
| ,1610 | 20 | ED | 15 | JSR | 15ED |
| ,1613 | 20 | 06 | 16 | JSR | 1606 |
| ,1616 | C8 | | | INY | |
| ,1617 | C0 | E6 | | CPY | #E6 |
| ,1619 | D0 | F5 | | BNE | 1610 |
| ,161B | A0 | E4 | | LDY | #E4 |
| ,161D | 20 | ED | 15 | JSR | 15ED |
| ,1620 | 20 | 06 | 16 | JSR | 1606 |
| ,1623 | 88 | | | DEY | |
| ,1624 | C0 | 31 | | CPY | #31 |
| ,1626 | D0 | F5 | | BNE | 161D |
| ,1628 | 60 | | | RTS | |

Sobald im Y Register der Wert 50 unterschritten wird, endet die Schleife und die Geister sind wieder am oberen Bildschirmrand angelangt.