Reise durch die Wunderwelt der Grafik

Nachdem wir sie in den ersten vier Folgen kaum erwähnt haben, sind sie heute dran: die Sprites. Sie müssen einfach in jeder Reise durch die Grafik unseres C 64 auftauchen.

Sprite heißt auf deutsch soviel wie »Kobold«, »Gespenst«. Wie richtige Kobolde können sie sowohl in Dornröschens Schloß (also im Bit-Map-Modus) als auch außerhalb (nämlich im Normalmodus) über den Bildschirm geistern — anscheinend dabei allen bisher gelernten Regeln über die Grafik widersprechen. Wir werden in dieser Folge zwar lernen, mit ihnen umzugehen, sogar sie zu beherrschen — aber ihre genaue Funktion und Herkunft wird weiterhin im dunkeln bleiben: Meines Wissens gibt es noch kein Listing des Sprite-unterstützenden Maschinenprogramms, das wohl im tiefsten Dunkel des VIC-II-Chip verborgen liegt: Denn der VIC-II-Chip belegt ja die Speicherplätze 53248 bis 54271. Im erreichbaren Teil dieses Zauberschlosses (53248 bis 53294) liegen die bisher viel von uns begangenen 47 Register (siehe 1. Folge, Tabelle 1), aber wo ist die Geheimtür zu den anderen 978 Bytes? Alles recht romantisch, werden Sie sagen. Nun — ganz so mysteriös ist die Sache nun auch wieder nicht, wie uns der Name Sprite = Kobold, Gespenst einreden will. Gehörig entschleiert wird das Geheimnis schon durch den anderen englischen Ausdruck für diese Dinger: MOBs. Das bedeutet: Movable Object Blocks, also bewegliche Blöcke von Objekten (Bildern, Darstellungen). Sie werden sehen, wenn wir mit dem MOBs umgehen können, ist das verbleibende Geheimnis eigentlich keines mehr, sondern verwandelt sich nur noch in eine Herausforderung für einen Maschinensprache-Programmierer.

Wir werden zunächst einmal schöpferisch tätig und erschaffen ein Sprite. Der erste Schritt dazu ist kreativ: Wie soll das Ding aussehen? Da bietet sich ja zum Beispiel der aus der Magie bekannte Drudenfuß an, auch Pentagramm genannt, weil wir ja schließlich diese Sprite-Geister bannen wollen (siehe Bild 1).

Jetzt müssen wir diese Zeichnung in eine Form bringen, die unser Computer versteht, also in Bytes. Wie auch schon bei den Buchstaben und der hochauflösenden Grafik sind hier wieder gesetzte und gelöschte Bits in den Bytes die Anzeige für »Punkt sichtbar« oder »Punkt nicht sichtbar«. Das im VIC-II-Chip organisierte Sprite-Programm nimmt die Bytes in der im Bild 2 gezeigten Anordnung wahr.

Den so gebildeten Block verwaltet es in genau dieser Anordnung als ein Sprite. Deswegen müssen wir uns nun die Mühe machen, unser Pentagramm in so ein Bit-Raster ein-

Zeile	Byte	binär	dez.	Zeile	Byte	binär	dez.
	0	00000000	4		33	00000001	1
1	1	00011000	24	12	34	10000001	129
	2	00000000	0		35	10000000	128
	3	00000000	0		36	00000001	1
2	4	00011000	24	13	37	11000011	195
	5	00000000	0		38	10000000	128
	6	00000000	0		39	00000001	1
3	7	00111100	60	14	40	10111101	189
	8	00000000	0		41	10000000	128
	9	00000000	0		42	00000011	3
4	10	00100100	36	15	43	00011000	24
	11	00000000	0		44	11000000	192
	12	00000000	0		45	00000011	3
5	13	01100110	102	16	46	00111100	60
	14	00000000	0		47	11000000	192
	15	00000000	0		48	00000110	6
6	16	01100110	102	17	49	11000011	195
	17	00000000	0		50	01100000	96
	18	00000000	0		51	00000111	7
7	19	01100110	102	18	52	10000001	129
	20	00000000	0		53	11100000	224
	21	00111111	63		54	00000110	6
8	22	11111111	255	19	55	00000000	0
	23	11111100	252		56	01100000	96
	24	00011000	24		57	00001100	12
9	25	01000010	66	20	58	00000000	0
	26	00011000	24		59	00110000	48
	27	00001100	12		60	00000000	0
10	28	11000011	195	21	61	00000000	0
	29	00110000	48		62	00000000	0
	30	00000110	6				
11	31	10000001	129				
	32	01100000	96				

Tabelle 1. Kennzahlen für die Berechnung von Sprites

elt Teil 5

zufügen. Das ist in Bild 3 geschehen.

Nun muß dieses Bild in einen Zahlencode übersetzt werden. Überall dort, wo ein Bitfeld ausgefüllt ist, steht bei der Binärzahl eine 1, sonst eine Null. Demnach ergeben sich die Kennzahlen in der Tabelle 1.

Dieser Wust an Zahlen legt also unser Sprite fest. Diese doch recht aufwendige Rechnerei und Planerei ist, wenn sie aufmerkamer Leser von Computerliteratur sind — meist nicht mehr nötig. Obwohl es sicher gut ist, das Planen eines MOBs auch von Hand zu beherrschen (wie wir jetzt!), kann man sich die Arbeit doch mächtig erleichtern durch Eintippen eines der vielen Sprite-Editor-Programme, die es in fast allen Fachzeitschriften wie Sand am Meer gibt. Weil hier nicht der 1001. Sprite-Editor abgedruckt werden soll, dient das anliegende Programm »Sprity« anderen Zwecken. (Ein nettes kurzes Listing von H. Kunz finden Sie zum Beispiel in der Zeitschrift Computer persönlich Nr. 21, 1983 auf Seite 120). Jetzt müssen wir noch dafür sorgen, daß der C 64 diese Zahlen irgendwo zugreifbar hat, mit anderen Worten: Sie müssen in den Speicher gePOKEd werden. Im allgemeinen verwendet man dazu eine kleine FOR-NEXT-Schleife in der die in DATA-Zeilen abgelegten Zahlen gelesen und eingePOKEd werden. Wohin packt man die Kennzahlen? Im Prinzip kann man sie überall — wo sie nicht gerade lebenswichtige Computerfunktionen oder Basicprogramme stören — unterbringen. Es gibt lediglich zwei Dinge, die zu beachten sind:

a) Die Startadresse muß durch 64 glatt teilbar sein. (Zum Beispiel 896 = 14 mal 64 etc.)

b) Aus Gründen, auf die wir noch zu sprechen kommen werden, sollten die Sprite-Daten im gleichen 16 KByte-Speicherabschnitt (siehe vorangegangene Folge) abgelegt werden, in dem sich das Video-RAM befindet, welches bei der Sprite-Nutzung angeschaltet ist.

Damit gibt es im Prinzip 256 Orte pro Speicherabschnitt, in denen die Sprite-Daten gespeichert werden können. Allerdings sind einige Stellen zum Beispiel im Abschnitt 0 (mit dem normalen Bildschirm) bei der Verwendung von nur wenigen Sprites besonders bevorzugt, weil man keinen Basicspeicher wegnimmt und deshalb auch keine Schutz-POKEs nötig sind:

1) 704 — 767 ungenutzte Adressen
2) 832 — 895 Kassettenpuffer
3) 896 — 959 Kassettenpuffer
4) 960 — 1023 restlicher Kassettenpuffer und freier Platz.

Für weitere MOBs muß dann Basicspeicher verwendet und dieser dann vor dem Überschreiben durch Programmtext, Variablen oder Strings — wie in Folge 2 gezeigt — geschützt werden. Dem Erfindungsreichtum sind allerdings keine Grenzen gesetzt. So könnte man beispielsweise (siehe vorangegangene Folge) den Bildschirm an den oberen Rand des Abschnittes 2 verschieben und darunter oder darüber Sprite-Daten ablegen. Man muß dann zwar auch den Speicherschutz einPOKEn, hat aber trotzdem meistens noch mehr als genug Basicspeicherplatz. Wenn man nur vier Sprites gleichzeitig verwendet, insgesamt aber mehrere definiert hat, kann man sie alle im 4 KByte-Bereich ab $C000 abspeichern und bei Bedarf den Sprite, der dran ist, mit einer kleinen FOR-NEXT-Schleife in einen der vier Speicherbereiche (704 .... und so weiter) umladen. Sicher fallen Ihnen noch mehr Möglichkeiten ein. Wir werden uns im nachfolgenden einfach mit drei Sprites begnügen und uns nur im normalen Abschnitt 0 bewegen.

Legen wir also nun zunächst unser Pentagramm in den Bereich 704 bis 767 und nach 832 bis 895:

```
10 FOR I=0 TO 62:READ A:POKE
704+I,A:POKE 832+I,A:NEXT I
30 DATA hier werden jetzt unsere 63
Kennzahlen eingegeben.
bis 60 DATA
```

Sprite gerade noch voll sichtbar		Sprite gerade nicht mehr sichtbar	
links oben	rechts oben	links oben	rechts oben
X = 24 Y = 50	X = 320 Y = 50	X = 0 oder/und Y = 29	X = 344 oder/und Y = 29
links unten	rechts unten	links unten	rechts unten
X = 24 Y = 229	X = 320 Y = 229	X = 0 oder/und Y = 250	X = 344 oder/und Y = 250

Tabelle 2. Grenzposition normaler Sprites

Sprite No.	Register
0	53287
1	53288
2	53289
.	.
.	.
.	.
7	53294

Tabelle 3. Zuordnung der Sprite-Farben-Register

Bit-Paar	Farbherkunft
00	durchsichtig
01	Sprite Mehrfarbregister 53285
10	normale Sprite-Farben-Register (53287 bis 53294)
11	Sprite Merhfarbregister 52286

Tabelle 4. Herkunft der Bitpaar-Farben im Multicolor-Modus

Sprite gerade noch voll sichtbar		Sprite gerade nicht mehr sichtbar	
links oben	rechts oben	links oben	rechts oben
X = 24 Y = 50	X = 296 Y = 50	X nicht möglich nur: Y = 8	X = 344 oder/und Y = 8
links unten	rechts unten	links unten	rechts unten
X = 24 Y = 208	X = 926 Y = 208	X nicht möglich nur: Y = 250	X = 344 oder/und Y = 250

Tabelle 5. Grenzposition doppelt gedehnter Sprites

Noch ignoriert der C 64 unsere Sprites völlig. Es interessiert ihn überhaupt nicht, was wir an Arbeit zum Füllen seiner Speicherbauches aufgewendet haben. Wir müssen ihm noch mitteilen, wo der VIC-II-Chip die Sprite-Daten finden kann. Weil dieser Chip dazu eingerichtet ist, gleichzeitig 8 MOBs zu verwalten, gibt es acht Speicherzellen, die sogenannte Sprite-Zeiger enthalten.

Na, wo ist er denn? Sprite-Zeiger

Sie befinden sich immer im gleichen 1 KByte-Bereich, in dem auch der Bildschirmspeicher liegt. Weil dieser Video-RAM nur 1000 Bytes benötigt, sind oberhalb desselben noch 24 Bytes frei, von denen die obersten 8 als Sprite-Zeiger dienen. Wenn also im Normalfall der Bildschirmspeicher von 1024 bis 2023 geht, dann liegen die Sprite-Zeiger von 2040 bis 2047. Verschiebt man den Bildschirm, dann werden die Sprite-Zeiger mit verschoben. In diese Sprite-Zeiger-Bytes POKEn wir die Zahlen ein, die mit 64 multipliziert die Startadresse der Sprite-Daten ergeben. In unserem Beispiel also: 704/64 = 11 und 832/64 = 13.

Wenn wir diese Sprite-Zeiger eingeben, nehmen wir automatisch gleichzeitig auch die Numerierung vor. Dabei gehört zu
Sprite Nr. 0 der Sprite-Zeiger 2040
Sprite Nr. 1 der Sprite-Zeiger 2041
Sprite Nr. 2 der Sprite-Zeiger 2042
und so weiter.

Nennen wir also einfach den Sprite, dessen Daten von 704 bis 767 liegen, Nummer 0 und den anderen Nummer 1, dann müssen wir also eingeben:
70 POKE 2040,11:POKE 2041,13
Damit sind die Sprite-Zeiger gesetzt.

Anschalten der Sprites

Wenn Sie jetzt freudig RUN (RETURN) eingetippt haben, um unsere Drudenfüsse zu sehen, werden Sie ein langes Gesicht gemacht haben: Nix zu sehen! Aber das ist wie bei einer Lampe: Sie haben den Lampenschirm, die Glühbirne, den Stecker eingesteckt und sie leuchtet nicht! Deswegen schalten wir sie jetzt ein, die Sprites. Der Schalter dafür sitzt im Sprite-Kontrollregister 53269 (siehe Folge 1 Tabelle der VIC-II-Chip-Register). Dort gibt es für jedes MOB ein Bit. Also Sprite Nummer 0 entspricht Bit Nummer 0 und so weiter. Ein Sprite ist eingeschaltet, wenn sein Bit auf 1 gesetzt ist. Wie man einzelne Bits setzt oder löscht, kennen wir noch aus der zweiten Folge. Spielen wir das hier nochmal an unserem Beispiel durch: Wir wollen Sprite Nummer 0 und Sprite Nummer 1 einschalten, müssen also die Bits 0 und 1 auf den Wert 1 setzen. Da gab es doch die OR-Funktion und eine sogenannte Maske:

XXXXXXXX irgendein binärer Inhalt von 53269
OR 00000011 Maske (= dezimal 3)
XXXXXX11 Ergebnis (Bits 0 und 1 sind = 1)

Das ergäbe die Programmzeile:
80 POKE 53269,PEEK(53269) OR 3

Will man einzelne MOBs mit der Nummer N einschalten, dann empfiehlt sich folgender Befehl:
POKE 53269,PEEK(53269) OR (2↑N)

	Abschnitt 1	Abschnitt 2	Abschnitt 3
Zeile 1	Byte 0	Byte 1	Byte 2
Zeile 2	Byte 3	Byte 4	Byte 5
.	.	.	.
.	.	.	.
Zeile 20	Byte 57	Byte 58	Byte 59
Zeile 21	Byte 60	Byte 61	Byte 62

Bild 2. Die Sprite-Organisation im VIC-II-Chip

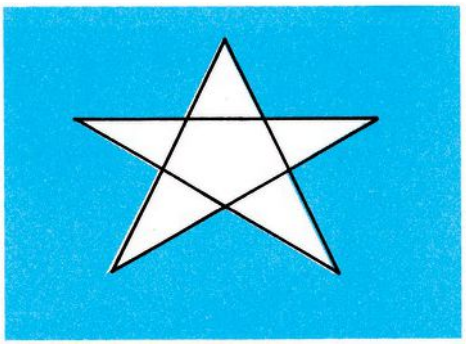

Bild 1. Ein Drudenfuß oder Pentagramm

Das Ausschalten geschieht durch die AND-Funktion. Jedes Bit, das mit 0 AND-verknüpft wird, wird dadurch gelöscht. Wenn wir also Sprite Nummer 1 ausschalten wollen, verwenden wir wieder eine Maske:

XXXXXX11 PEEK(53269), Sprite 0 und 1 eingeschaltet
AND 11111101 Maske (dezimal = 253)
XXXXXX01 Ergebnis: Bit 1 = 0, Sprite 1 ausgeschaltet,
Bit 0 = 1, Sprite 0 eingeschaltet.

Allgemein kann man einzelne Sprites also abschalten mit POKE 53269, PEEK(53269) AND (255-2↑N), wobei wieder N die Sprite-Nummer ist.

Ach, Ihre Geduld wird schon auf eine harte Probe gestellt. Wenn Sie nämlich bis jetzt alle Programmzeilen brav eingegeben und gestartet haben, sehen Sie immer noch kein Sprite! Aber Sie müssen dem MOB noch sagen, wo er erscheinen soll!

Schon wieder ein Koordinatensystem: Ort der Sprites

Vor den Erfolg haben auch die Commodore-Softwareplaner den Schweiß gesetzt! Denn nicht genug damit, daß wir den Bildschirm schon im Normalmodus in X-Richtung, in Y-Richtung in 25 Positionen und im Bit-Map-Modus in X-Richtung in 320, in Y-Richtung in 200 Positionen aufgeteilt finden, jetzt kommt noch eine Einteilung, die sogar noch über den sichtbaren Bildschirm hinausreicht! In Bild 4 sehen wir diese Aufteilung in 512 horizontale und 256 vertikale Koordinaten.

Für den Ort eines Sprites ist — wie in Bild angedeutet — die linke obere Ecke des Spritedefinitionsfeldes entscheidend. Also auch dann, wenn diese Ecke (wie in unserem Pentagramm) unsichtbar. So hat das im Bild gezeigte MOB die X-Koordinate 128 und die Y - Koordinate 120. Diese X- und Y-Werte muß man nun in die zur Sprite-Nummer

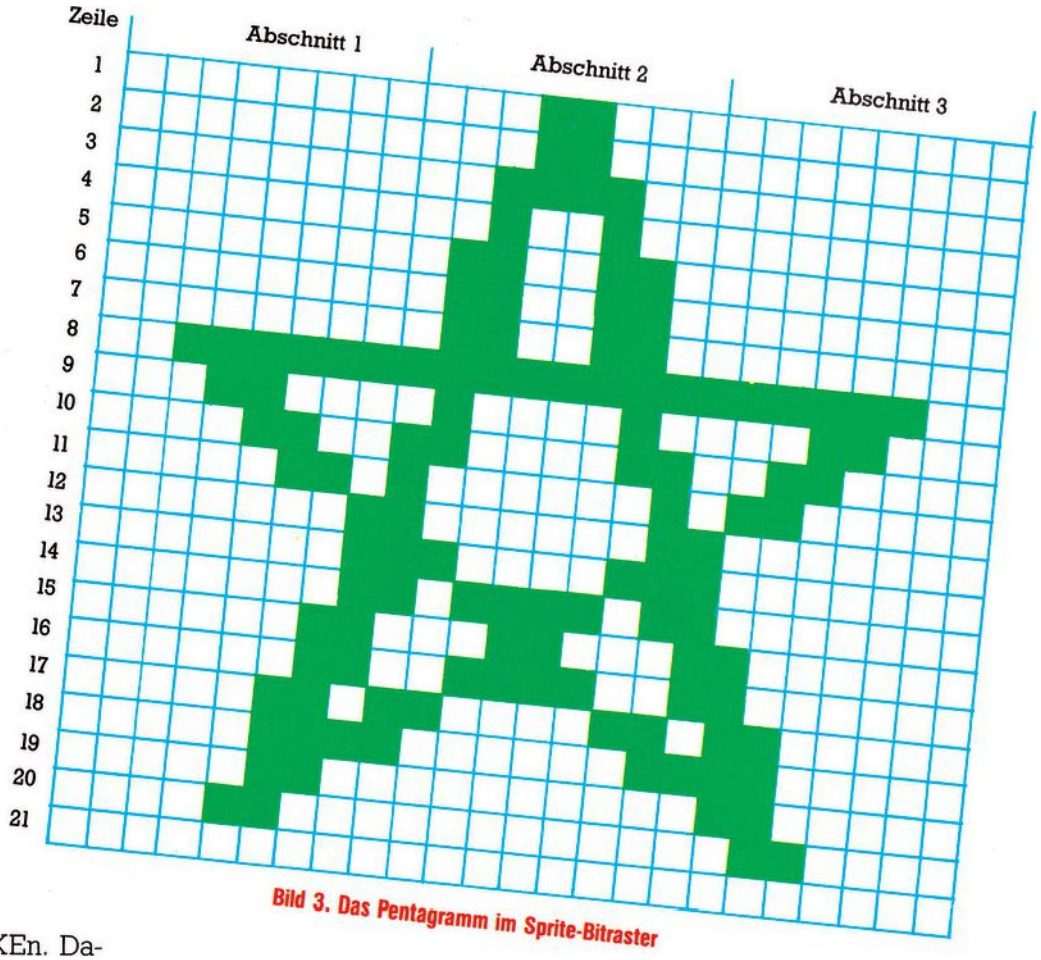

Bild 3. Das Pentagramm im Sprite-Bitraster

gehörigen Register einPOKEn. Dabei handelt es sich um folgende Speicherstellen:
X-Position von Sprite 0 : 53248
Y-Position von Sprite 0 : 53249

X-Position von Sprite 1 : 53250
Y-Position von Sprite 1 : 53251

und so weiter bis
X-Position von Sprite 7 : 53262
Y-Position von Sprite 7 : 53263

Um nun also endlich unser Sprite Nummer 0 sichtbar zu machen, geben wir ein:
100 POKE 53248,128:POKE 53249,120
So! Jetzt tippen Sie ein RUN (RETURN) und endlich: Da ist unser Pentagramm. Gefällt es Ihnen? Sprite 1, das zweite Pentagramm, soll am rechten Bildschirmrand auftauchen, also ungefähr bei X = 266 und bei Y = 130. Deswegen müßten wir aber in die entsprechende X-Positions-Speicherstelle für Sprite 1 eine Zahl größer als 255 einPOKEn! Wenn Sie's versuchen, meldet der Computer natürlich einen ILLEGAL QUANTITY ERROR. Geht also nicht! Anscheinend war das bisher noch nicht verzwickt genug. Jetzt müssen wir nämlich mal wieder das Hexadezimalsystem bemühen (die Sechzehnfingerlinge, erinnern Sie sich: Folge 2).

Dezimal 266 ist hexadezimal $010A. Jetzt zerteilen wir diese Zahl wieder in das LSB und das MSB:
$ 01 0A
MSB LSB
=1 =0A
und rechnen wieder zurück ins Dezimalsystem:
MSB = 1
LSB = 10

Rechnen Sie mal nach: Das MSB kann auch bei der höchsten X-Position nie größer als 1 werden. Es gibt also nur zwei Fälle: Ist die X-Position größer als 256, dann ist das MSB 1, sonst ist es 0.

Deswegen speichert man die MSBs aller 8 Sprites in nur einem Byte. Ebenso wie beim Kontrollregister für das An- und Ausschalten gehört auch hier zu jedem Sprite ein Bit, also: Bit 1 gehört zu Sprite 1 und so weiter. Wenn nun also die X-Koordinate von Sprite 1 266 ist, dann ergibt die Aufspaltung, wie oben gezeigt, MSB = 1 und LSB = 10. Das Register für die MSBs ist Speicherstelle 53264. Die 10 (das LSB) wird also in die normale Speicherstelle für die X-Position gePOKEt. Das MSB (= 1) ist also Bit 1 (weil Sprite 1) von Speicherstelle 53264. Hier ist also wieder eine OR-Operation nötig. Die Y-Position wird ganz normal eingegeben. Es ergibt sich also die Programmzeile:
110 POKE 53250,10:POKE 53264, PEEK(53264)OR(2↑1):
POKE 53251,130

Wenn Sie jetzt einen Schwarzweiß-Monitor haben, sehen Sie nach RUN (RETURN) trotzdem nur unser Sprite Nummer 0. Farbmonitor-Eigner bewundern schon jetzt Sprite Nummer 1. Warum, das wird uns gleich noch beschäftigen.

Welche Koordinaten eines Sprites sind eigentlich möglich, und was sieht man dann? Um das zu erforschen, bauen wir uns ein Primitiv-Sprite:
20 FOR I = 0TO 62:POKE I+896, 255:NEXT I:POKE 2042,14:
REM SPRITE NR. 2

Dann schalten wir ein:
90 POKE 53269,PEEK(53269) OR (12)
und bauen eine Abfrage ein für die Position:
130 INPUT"SPRITE 2:X,Y=";X,Y:IF X=—1 THEN END

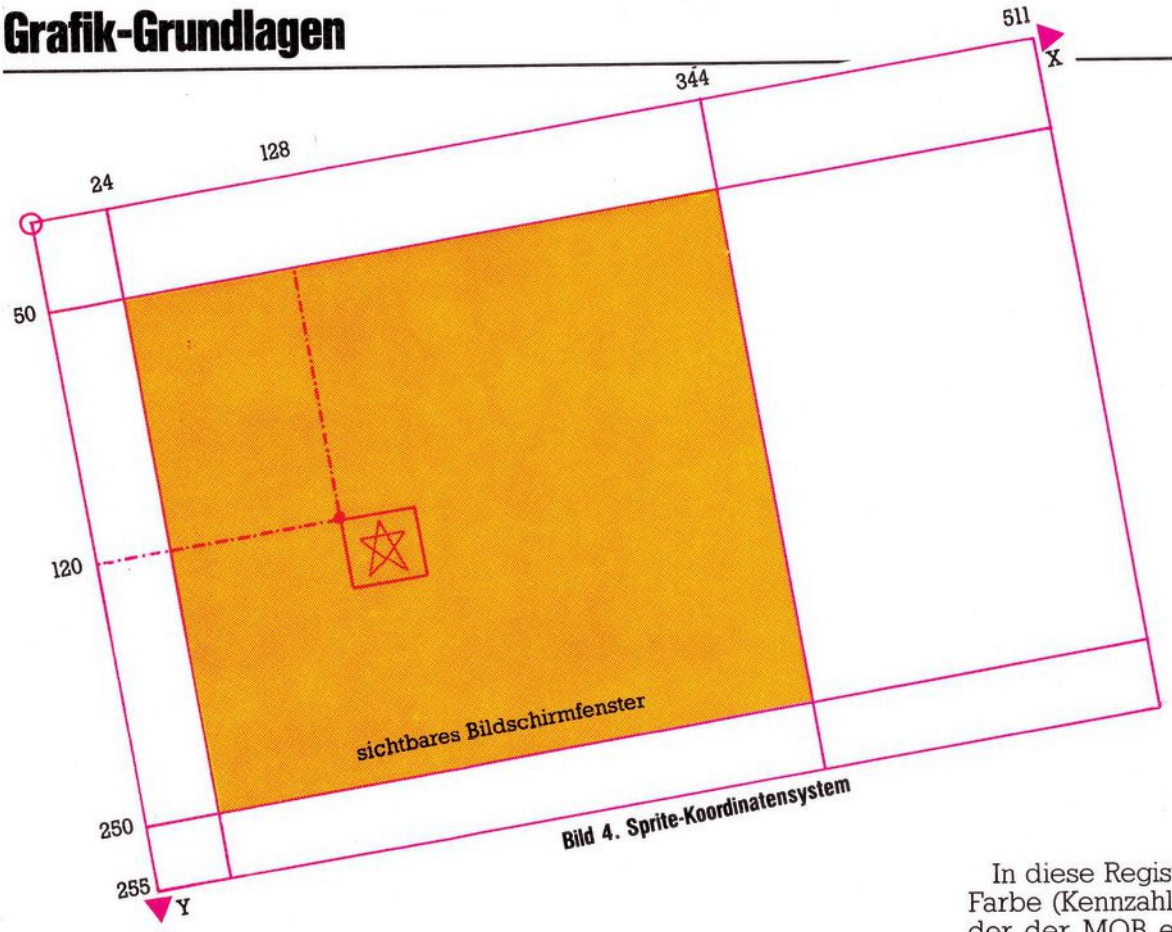

Bild 4. Sprite-Koordinatensystem

Für die X-Position soll der Computer noch die Umrechnung in LSB und MSB für uns durchführen:
140 X1=INT(X/256):X2=X—256*X1
Dann POKEn wir diese Koordinaten ein:
150 POKE 53252,X2:POKE 5253,Y: IF X1=1 THEN POKE 53264,PEEK (53264)OR(212)

Wenn die X-Koordinaten kleiner als 256 sind, also das MSB = 0 ist, muß es natürlich gelöscht werden:
160 IF X1=0 THEN POKE 53264,PEEK(53264)AND(255—2127)

Schließlich lassen wir uns außer dem Sprite Nummer 2 auch noch die eingegebenen Koordinaten zeigen und kehren zur Abfrage zurück:
170PRINTCHR$(147), "x="x, "y="y: GOTO 130

Wenn Sie jetzt starten, können Sie alle möglichen Positionen für X und Y eingeben. Versuchen Sie doch mal X = 265, Y = 130. Da ist auch für die Schwarzweiß-Seher (ich bin auch einer) unser Sprite Nummer 1 zu erkennen. Warum, dazu kommen wir noch. Wenn Sie genug ausprobiert haben, dann geben Sie für X jetzt —1 und irgendeinen Y - Wert ein und das Programm ist beendet. Vermutlich haben auch Sie festgestellt, daß man unseren Testsprite ganz allmählich über den sichtbaren Bildschirmrand hinauswandern lassen kann. Es ergeben sich so die Grenzkoordinaten in Tabelle 2:

Die in der Tabelle 2 angegebenen Werte gelten allerdings nur für die normalen Sprites. Also gibt es auch noch anormale Sprites? Ja, die gibt's auch. Aber wir wollen der Reihe nach vorgehen. Wir können jetzt Sprites entwerfen, Sprite-Zeiger setzen, MOBs an- und wieder abschalten und sie an die richtige Stelle setzen.

Mal wieder Farbe: Diesmal die von Sprites.

Eines haben wir bisher total vergessen: Welche Farbe soll unser Sprite haben und wie geben wir sie ihm? Auch hierfür gibt es natürlich wieder Register, die Sprite-Farben-Register und zwar die in der Tabelle 3.

In diese Register POKEt man die Farbe (Kennzahlen 0 bis 15) ein, in der der MOB erscheinen soll. Testen Sie mal: Ändern wir Zeile 130. Dort soll in der IF - THEN-Anweisung nach THEN anstelle von END jetzt stehen 180. Diese Zeile 180 schaltet das vorerst ausgediente Sprite 2 aus:
180 POKE 53269,PEEK(53269) AND (255—212)

Damit eröffnen wir uns die Farbeingabemöglichkeit mit:
190 PRINT CHR$(147):INPUT"SPRITE 0,SPRITE 1 FARBEN";
F1,F2:IF F1=-1THEN210

Schließlich POKEn wir die Farben in die zu den Sprites gehörigen Register:
200 POKE 53287,F1:POKE53288, F2:GOTO190
210 END

Jetzt können Sie, bis Sie schließlich für F1 mal —1 eingeben, allerlei Farben durchprobieren. Wir erkennen nun auch, daß bislang Sprite 1 für

```
READY.

10  :   REM**************************************************************
20  :   REM**                    S P R I T Y                             **
30  :   REM**                                                            **
40  :   REM**       MANFRED WALTER THOMA  2102 HAMBURG 93  ERNASTRASSE 10**
50  :   REM**                                                            **
60  :   REM**************************************************************
70 POKE53280,0:POKE53281,0:POKE646,1:PRINTCHR$(147);
75  :   REM**************************************************************
80  :   REM**                  DATEN FUER DIE DREI SPRITES               **
90  :   REM**************************************************************
100 DATA0,0,0,0,0,0,0,0,0,0,0,0,14,0,0,17,0,0,32,128,0,64,64,0,155,32,0,128
110 DATA32,0,132,32,0,132,32,0,145,32,0,78,64,0,32,128,0,31,0,0,0,0,0,0,0,0
120 DATA0,0,0,0,0,0,0
130 DATA0,0,0,0,0,0,0,0,0,0,0,0,14,0,0,17,0,0,32,128,0,64,64,0,155,32,0,128
140 DATA32,0,132,32,0,132,32,0,128,32,0,78,64,0,64,64,0,32,128,0,31,0,0,0,0,0
150 DATA0,0,0,0,0,0,0,0
160 DATA0,0,0,0,0,0,0,0,0,0,0,0,14,0,0,17,0,0,32,128,0,64,64,0,155,32,0,128
170 DATA32,0,132,32,0,132,32,0,128,32,0,78,64,0,81,64,0,64,64,0,32,128,0,31,0
180 DATA0,0,0,0,0,0,0,0,0
```

Schwarzweiß-Seher nicht erkennbar war, weil seine Farbe keinen Kontrast zur Hintergrundfarbe gebildet hat.

Wem's noch nicht bunt genug war bisher, der hat auch hier bei den Sprites die Möglichkeit, den Mehrfarben-Modus zu verwenden. Während im bisher gebrauchten Modus jedes Sprite-Definitions-Bit entweder 0 (=Hintergrundfarbe) oder 1 (Farbe des zum Sprite gehörigen Sprite-Farb-Registers) sein konnte, zählen — wie auch sonst im Mehrfarbenmodus — wieder Bit-Paare. Dabei stammt dann die jeweilige Farbe aus den in Tabelle 4 angegebenen Registern.

Unser Drudenfuß sieht somit dann aus wie in Bild 5 gezeigt.

Das wollen wir noch einmal auf dem Bildschirm ansehen. Wir schreiben ab Zeile 190 neu:

```
190 PRINT CHR$(147):INPUT"MOB1,
MOB2,MULTCOL1,MULTCOL2";F1,
F2,F3,F4:IFF1=-1THEN 220
200 POKE 53287,F1:POKE53288,F2:
POKE 53285,F3:POKE 53286,F4
220 END
```

So läuft natürlich noch nichts Neues, denn der Mehrfarben-Modus muß noch angeschaltet werden. Auch dazu gibt es wieder ein Register: 53276. Wie bei einigen anderen Sprite-Registern gehört auch wieder zu jedem MOB das entsprechende Bit, also zu Sprite 1 das Bit 1 und so weiter. Wenn dieses Bit auf 1 gesetzt ist, ist für das dazugehörige Sprite der Mehrfarb-Modus angeschaltet. Man kann sie also einzeln oder zusammen — ganz wie's beliebt — im Normal-Modus oder im Muticolor-Modus betrachten. Wir schalten Sprite 0 und Sprite 1 in den Mehrfarb-Modus mit Zeile

```
210 POKE 53276,PEEK(53276)OR3:
GOTO 190
```

Will man nur Sprite Nummer N umschalten, dann verwendet man wie gehabt: POKE 53276, PEEK(53276)OR(2↑N).

Das Zurückschalten in den Normalmodus geschieht dann durch Löschen der entsprechenden Bits: POKE 53276,PEEK(53276)AND (255—2↑N).

Auch hier habe ich das Programm so gebaut, daß man durch Eingabe von —1 und irgendwelchen drei anderen Zahlen aussteigen kann. Im Verlauf des Probierens werden Sie bestimmt gemerkt haben, daß man Sprites, die für den Mehrfarben-Modus gedacht sind, speziell konstruieren sollte unter Berücksichtigung der Bit-Paar-Zusammenstellungen. Unser Drudenfuß gefällt mir im Normal-Modus jedenfalls besser.

Deswegen schalten wir in Zeile 220 lieber den Mehrfarben-Modus aus:

```
220 POKE 53276,PEEK(53276) AND
252
```

Anormale Sprites? Gequetschte und gezerrte MOBs

Die normalen Sprites bestehen aus 24 x 21 Bildpunkten und haben auf dem Bildschirm eine Ausdehnung von zirka drei Zeilen mal drei Spalten. Für einige Effekte ist es ganz nett, sie in ihrer Größe verändern zu können. Genau das ist möglich, und zwar in X-Richtung, in Y-Richtung oder in beide Richtungen gleichzeitig. Das Merkwürdige an diese Sache ist — außer dem manchmal recht verzerrten Aussehen —, daß die gleiche 24 mal 21 Pixel abgebildet werden, nur jedes Pixel ist vergrößert. Wenn sowohl in X- als auch in Y-Richtung vergrößert

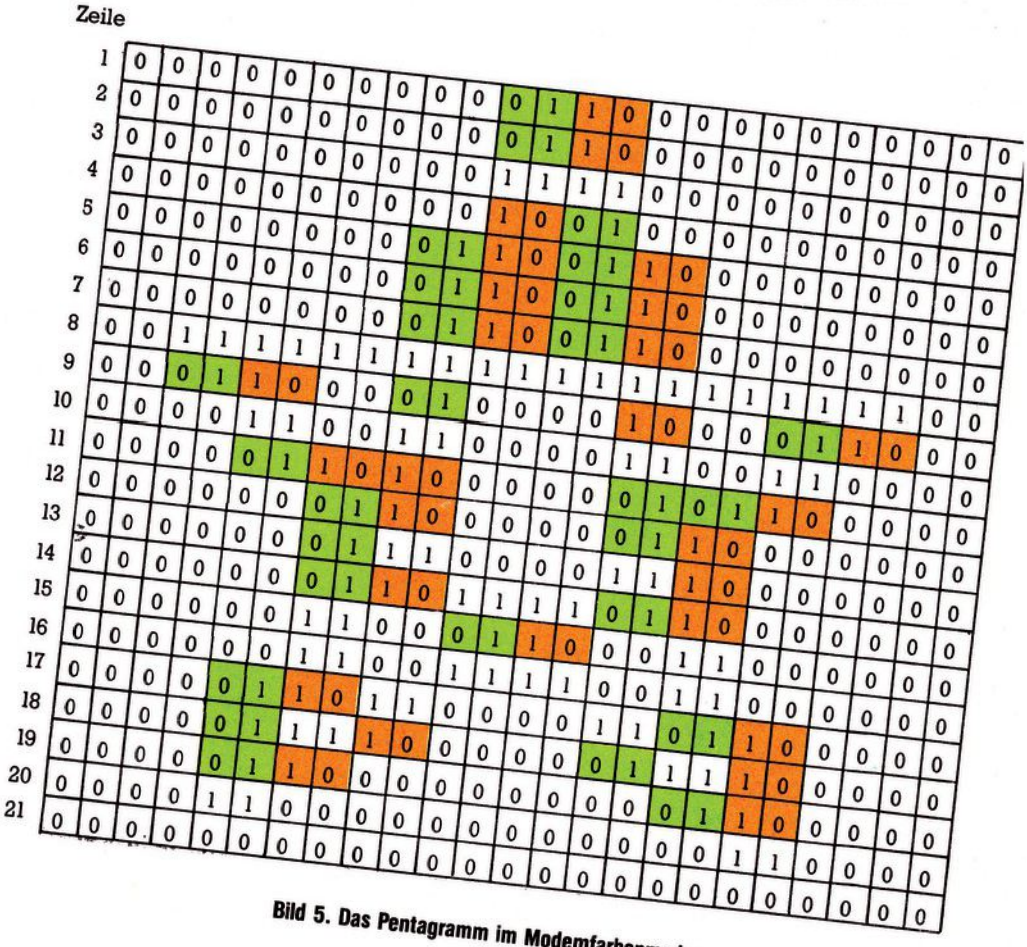

Bild 5. Das Pentagramm im Modemfarbenmodus

```
190 :  REM*************************************************************
200 :  REM**              EINLESEN DER SPRITES UND TITELBILD           **
210 :  REM*************************************************************
220 PRINTTAB(12)"S  P  R  I  T  Y"
230 PRINTTAB(12)"================"
240 PRINT:PRINT:PRINT:PRINT
250 PRINTTAB(19)"BY"
260 PRINTTAB(12)"MANFRED W. THOMA"
270 PRINTTAB(12)"2102  HAMBURG 93"
280 PRINTTAB(12)" ERNASTRASSE 18"
290 FORI=13*64TO13*64+62:READX:POKEI,X:NEXT
300 POKE2040,13:                    REM** SPRITE 0 AUS BLOCK 13      **
310 FORI=14*64TO14*64+62:READX:POKEI,X:NEXT
320 POKE2041,14:                    REM** SPRITE 1 AUS BLOCK 14      **
330 FORI=15*64TO15*64+62:READX:POKEI,X:NEXT
340 POKE2042,15:                    REM** SPRITE 2 AUS BLOCK 15      **
350 V=53248:                        REM** STARTADRESSE DES VIC 2 CHIP**
360 POKE V+39,1:                    REM** FARBE FUER SPRITE 0        **
370 POKE V+40,1:                    REM** FARBE FUER SPRITE 1        **
380 POKE V+41,1:                    REM** FARBE FUER SPRITE 2        **
```

Das Programm »Sprity«

wurde, ist einfach jeder Bildpunkt viermal so groß wie vorher. Auch hier geschieht das natürlich wieder über Kontrollregister, von denen jedes Bit wieder zu einem Sprite gleicher Nummer gehört. Die Verdoppelung in X-Richtung wird durch eine 1 im zum Sprite gehörigen Bit des Registers 53277, die in Y-Richtung auf die gleiche Weise im Register 53271 geschaltet. Wenn wir also unser Sprite Nummer 0 in X-Richtung verdoppeln wollen, dann geben wir noch ein:

230 POKE 53277,PEEK(53277)OR1: FOR I=0TO2000:NEXT I

Dann sehen wir uns das nach einer kleinen Pause noch in Y-Richtung an:

240 POKE 53277,PEEK(53277)AND 254:POKE53271,PEEK(53271)OR1: FOR I=0 TO 2000:NEXT I

Jetzt vergrößern wir noch in beide Richtungen:

250 POKE 53277,PEEK(53277) OR 1

Wie Sie sich denken können, stimmt jetzt die Positionierung dieser vergrößerten Sprites auf dem sichtbaren Bildschirmteil nicht mehr. Um das zu testen, bemühen wir wieder unser Testsprite Nummer 2 und vergrößern in beide Richtungen:

120 POKE 53277,PEEK(53277)OR4: POKE53271,PEEK(53271)OR4

Wenn wir nun das Programm mit RUN (RETURN) starten, taucht unser Test-Sprite in vergrößterter Form auf und wir können es wieder an verschiedene Orte auf dem Bildschirm packen. Für dieses in beide Richtungen verdoppelte MOB findet man dann die Grenzwerte in Tabelle 5.

Will man also ein solchermaßen vergrößertes Sprite langsam aus dem Bildschirm ziehen lassen, dann ist das nicht nach links möglich, weil selbst bei X=0 der Sprite noch teilweise sichtbar ist.

Jetzt wissen wir eigentlich fast alles, was mit dem einzelnen Sprite zusammenhängt. Wenn Sie ein sich veränderndes Sprite darstellen wollen, so ist das zum Beispiel möglich, indem alle zu zeigenden Bewegungsphasen als Spritemuster im Speicher abgelegt werden und dann per Programm der Sprite-Zeiger auf den jeweils aktuellen Bewegungszustand umgeschaltet wird. So könnte man wie in Bild 6 die Abläufe A bis G im Speicher ablegen und zum Beispiel den Sprite-Zeiger für Sprite 3 zuerst auf das Muster A, dann nach entsprechender Verzögerung auf Muster B, dann C, D, E, F, G und schließlich wieder A richten. Wenn die einzelnen Spritemuster gut ge-

```
390 POKE V+23,7:                              REM** SPRITES 0-2 IN X-RICHTUNG  **
400 POKE V+29,7:                              REM** SPRITES 0-2 IN Y-RICHTUNG  **
410 POKEV,160:POKEV+1,160:                    REM** X-Y-KOORDINATEN SPRITE 0   **
420 POKEV+2,160:POKEV+3,160:                  REM** X-Y-KOORDINATEN SPRITE 1   **
430 POKEV+4,160:POKEV+5,160:                  REM** X-Y-KOORDINATEN SPRITE 2   **
440 FOR I=1 TO10
450 POKEV+21,1:FOR II=1TO300:NEXT:            REM** EINSCHALTEN NUR SPRITE 0   **
460 POKEV+21,2:FOR II=1TO200:NEXT:            REM** EINSCHALTEN NUR SPRITE 1   **
470 POKEV+21,4:FOR II=1TO100:NEXT:            REM** EINSCHALTEN NUR SPRITE 2   **
480 NEXTI
490 POKEV+21,0:                               REM** AUSSCHALTEN ALLER SPRITES  **
500 PRINTCHR$(147)
510 DR$="|     ←|"
520 POKE 650,128:                             REM**  REPEAT FUER ALLE TASTEN   **
530 : REM**********************************************************************
540 : REM**                    TASTENBELEGUNGSERKLAERUNG                      **
550 : REM**********************************************************************
560 PRINTCHR$(147);
570 PRINTTAB(9)"BEWEGUNGSSIMULATION"
580 PRINTTAB(9)"====================="
590 PRINT:PRINT"┌─────┐"
600 PRINT"| F1 |   BEWEGEN SPRITE 0 IN +X RICHTUNG";
610 PRINT"├─────┤
620 PRINT"| F3 |   BEWEGEN SPRITE 0 IN -X RICHTUNG";
630 PRINT"├─────┤
640 PRINT"| F5 |   BEWEGEN SPRITE 0 IN +Y RICHTUNG";
650 PRINT"├─────┤
660 PRINT"| F7 |   BEWEGEN SPRITE 0 IN -Y RICHTUNG";
670 PRINT"└─────┘
680 PRINT:PRINT"┌─────┐"
690 PRINT"| F2 |   BEWEGEN SPRITE 1 IN +X RICHTUNG";
700 PRINT"├─────┤"
710 PRINT"| F4 |   BEWEGEN SPRITE 1 IN -X RICHTUNG";
720 PRINT"├─────┤
730 PRINT"| F6 |   BEWEGEN SPRITE 1 IN +Y RICHTUNG";
740 PRINT"├─────┤
750 PRINT"| F8 |   BEWEGEN SPRITE 1 IN -Y RICHTUNG";
760 PRINT"└─────┘
770 PRINT:PRINTTAB(11)"WEITER BITTE TASTE"
780 GETA$:IFA$=""THEN780
790 PRINT"■■VERKLEINERN DER SPRITE"
800 PRINT"NACH DRUECKEN DER TASTE■< V >■KANN MIT DEN";
810 PRINT"FUNKTIONSTASTEN IN +X,-X UND +Y,-Y  VER-";
820 PRINT"KLEINERT WERDEN."
830 PRINT"■■FARBEN
840 PRINT"NACH DRUECKEN DER TASTE■< F >■KANN MIT DER";
850 PRINT"TASTE >F1< DIE FARBE FUER SPRITE0 GEAEN-";
860 PRINT"DERT WERDEN.MIT >F2< FARBE FUER SPRITE1.";
870 PRINT"■■HINTERGRUNDFARBE"
880 PRINT"DURCH DRUECKEN D. TASTE■< H >■WIRD DIE"
890 PRINT"BILDSCHIRMFARBE GEAENDERT."
900 PRINT"■■ENDE"
910 PRINT"PROGRAMMENDE MIT  TASTE■< E >■"
920 PRINT"■■SPRITE EIN/AUS"
930 PRINT"NACH DRUECKEN DER TASTE■< 0 >■KANN MIT DEN";
940 PRINT"FUNKTIONSTASTEN >F1< UND >F2<SPRITE0 UND";
950 PRINT"SPRITE1 EIN- BZW. AUSGESCHALTET WERDEN"
960 PRINT"■            ■WEITER BITTE TASTE"
970 GETA$:IFA$=""THEN970
980 PRINT CHR$(147);
990 PRINTSPC(33)" ■X-SP.0■";
1000 PRINTSPC(33)"|  50 ←";
1010 PRINTSPC(33)" ■Y-SP.0■";
1020 PRINTSPC(33)"| 150 ←";
1030 PRINTSPC(33)" ■X-SP.1■";
1040 PRINTSPC(33)"| 150 ←";
1050 PRINTSPC(33)" ■Y-SP.1■";
1060 PRINTSPC(33)"| 150 ←";
1070 PRINTSPC(33)" ■FARBE0■";
1080 PRINTSPC(33)"|   1 ←";
1090 PRINTSPC(33)" ■FARBE1■";
1100 PRINTSPC(33)"|   1 ←";
1110 PRINTSPC(33)" ■X-DEHN■";
1120 PRINTSPC(33)"|   3 ←";
1130 PRINTSPC(33)" ■Y-DEHN■";
1140 PRINTSPC(33)"|   3 ←";
1150 PRINTSPC(33)" ■SCHIRM■";
1160 PRINTSPC(33)"|   0 ←";
1170 PRINTSPC(33)" ──────";
1180 PRINTSPC(33)" ■ON/OFF■";
1190 PRINTSPC(33)" ■S0:■ ON";
1200 PRINTSPC(33)" ■S1:■ ON";
1210 PRINT"■              S P R I T Y            ";CHR$(19)
1220 POKEV,50:
1230 POKEV+2,150
1240 POKEV+21,3
1250 A=PEEK(V+30):                            REM** LESEN/LOESCHEN KOLLISONREG**
1260 : REM**********************************************************************
1270 : REM**                    TASTATUR ABFRAGE (GET)                         **
1280 : REM**********************************************************************
1290 GETA$:IFA$=""THEN1290
1300 IF A$="E"THENPRINTCHR$(147):POKEV+21,0:END
1310 IF A$="H"THEN 1730:                      REM** AENDERN DER BILDS.FARBE    **
1320 IF A$="V"THEN 1990:                      REM** SPRITES KLEINER/GROESSER   **
1330 IF A$="F"THEN 1830:                      REM** SPRITES FARBE              **
1340 IF A$="0"THEN 2210:                      REM** SPRITES EIN/AUS            **
1350 A=ASC(A$)-132:                           REM** UMRECHNEN FUNKTIONSTASTEN  **
1360 IFA<0 OR A>8THEN1290:                    REM** KEINE FUNKTIONSTASTE       **
1370 ON A GOSUB  1490,1520,1550,1580,1610,1640,1670,1700
1380 : REM**********************************************************************
1390 : REM**                    KOLLISIONS ABFRAGE                             **
1400 : REM**********************************************************************
```

Das Programm »Sprity« (Fortsetzung)

```
1410 IF PEEK(V+30)=0 THEN 1290
1420 PRINT"[S]       !!! KOLLISION !!!"
1430 FOR I=0 TO 100:NEXT
1440 PRINT"[S]                        "
1450 GOTO 1290
1460 : REM**************************************************************************
1470 : REM**                    SPRITE BEWEGEN X/Y-KOORD.                         **
1480 : REM**************************************************************************
1490 WE=PEEK(V):IFWE=255 THEN RETURN
1500 WE=WE+1:POKEV,WE
1510 Z=1:GOSUB2370:RETURN
1520 WE=PEEK(V):IFWE=1THEN RETURN
1530 WE=WE-1:POKEV,WE
1540 Z=1:GOSUB2370:RETURN
1550 WE=PEEK(V+1):IFWE=1THEN RETURN
1560 WE=WE-1:POKEV+1,WE
1570 Z=3:GOSUB2370:RETURN
1580 WE=PEEK(V+1):IFWE=198THEN RETURN
1590 WE=WE+1:POKEV+1,WE
1600 Z=3:GOSUB2370:RETURN
1610 WE=PEEK(V+2):IFWE=255 THEN RETURN
1620 WE=WE+1:POKEV+2,WE
1630 Z=5:GOSUB2370:RETURN
1640 WE=PEEK(V+2):IFWE=1THEN RETURN
1650 WE=WE-1:POKEV+2,WE
1660 Z=5:GOSUB2370:RETURN
1670 WE=PEEK(V+3):IFWE=1THEN RETURN
1680 WE=WE-1:POKEV+3,WE
1690 Z=7:GOSUB2370:RETURN
1700 WE=PEEK(V+3):IFWE=198THEN RETURN
1710 WE=WE+1:POKEV+3,WE
1720 Z=7:GOSUB2370:RETURN
1730 : REM**************************************************************************
1740 : REM**                    HINTERGRUNDFARBE (BILDS.)                         **
1750 : REM**************************************************************************
1760 HF=PEEK(53280)
1770 IF HF=255 THEN POKE53280,0:POKE53281,0:WE=0:Z=17:GOSUB2370:GOTO 1290
1780 POKE53280,HF+1:POKE53281,HF+1
1790 WE=HF+1-240:Z=17:GOSUB2370
1800 GOTO 1290
1810 :
1820 :
1830 : REM**************************************************************************
1840 : REM**                         SPRITE  FARBEN                               **
1850 : REM**************************************************************************
1860 FOR I=1 TO50
1870 GETA$
1880 IFA$=CHR$(133) OR A$=CHR$(137)THEN 1910:REM** PRUEFEN  >F1<  ODER  >F2< **
1890 NEXT I
1900 GOTO 1290
1910 F1=PEEK(V+40):F2=PEEK(V+39):          REM** AKTUELLE FARBEN SPRITE 0+1**
1920 IF A$=CHR$(137) THEN 1960
1930 IF F2=255THEN POKEV+39,0:WE=0:Z=9:GOSUB2370:GOTO1290
1940 POKE V+39,F2+1:WE=F2+1-240:Z=9:GOSUB2370
1950 GOTO1290
1960 IF F1=255THEN POKEV+40,0:WE=0:Z=11:GOSUB2370:GOTO1290
1970 POKE V+40,F1+1:WE=F1+1-240:Z=11:GOSUB2370
1980 GOTO1290
1990 : REM**************************************************************************
2000 : REM**                      VERKLEINERN DER SPRITES                         **
2010 : REM**************************************************************************
2020 FOR I=1 TO 100
2030 GETA$:IF A$=""THEN2060
2040 A=ASC(A$)-132
2050 IF A>0 AND A<9 THEN 2070
2060 NEXT I:GOTO 1290
2070 DX=PEEK(V+29):DY=PEEK(V+23)
2080 ON A GOTO 2100,2110,2120,2130,2140,2150,2160,2170
2090 GOTO1260
2100 DX=DX OR 1:GOTO2180
2110 DX=DX AND 254:GOTO2180
2120 DY=DY OR 1:GOTO2190
2130 DY=DY AND 254:GOTO2190
2140 DX=DX OR 2:GOTO2180
2150 DX=DX AND 253:GOTO2180
2160 DY=DY OR 2:GOTO2190
2170 DY=DY AND 253:GOTO2190
2180 POKE V+29,DX:WE=DX-4:Z=13:GOSUB 2370:GOTO1290
2190 POKE V+23,DY:WE=DY-4:Z=15:GOSUB 2370:GOTO1290
2200 GOTO1290
2210 : REM**************************************************************************
2220 : REM**                      SPRITES EIN/AUS SCHALTEN                        **
2230 : REM**************************************************************************
2240 FOR I=1 TO 100
2250 GETA$:IF A$=""THEN2280
2260 IFA$=CHR$(133)THEN 2290
2270 IFA$=CHR$(137)THEN 2310
2280 NEXT I: GOTO 1290
2290 IF(PEEK(V+21)AND1)=1THENPOKEV+21,PEEK(V+21)-1:GOTO2330:REM*BIT 0 GESETZT ?*
2300 POKEV+21,PEEK(V+21)+1:GOTO2340
2310 IF(PEEK(V+21)AND2)=2THENPOKEV+21,PEEK(V+21)-2:GOTO2350:REM*BIT 1 GESETZT ?*
2320 POKEV+21,PEEK(V+21)+2:GOTO2360
2330 POKE211,37:POKE214,20:SYS58640:PRINT"OFF":GOTO1290
2340 POKE211,37:POKE214,20:SYS58640:PRINT" ON":GOTO1290
2350 POKE211,37:POKE214,21:SYS58640:PRINT"OFF":GOTO1290
2360 POKE211,37:POKE214,21:SYS58640:PRINT" ON":GOTO1290
2370 :REM**************************************************************************
2380 :REM**                        CURSOR SETZEN (X-Y)                           **
2390 :REM**************************************************************************
2400 POKE211,33:POKE214,Z:SYS58640:PRINTLEFT$(DR$,9+LEN(STR$(WE)));WE:RETURN
2410 :
READY.
```

Das Programm »Sprity« (Schluß)

macht und die Verzögerungsschleifen richtig abgestimmt sind, kann so eine Art Zeichentrickfilm ablaufen, der nun auch noch durch die anderen bisher gelernten Sprite-Eigenheiten (Vergrößern, Position, Farbe etc.) veränderbar ist. Wie Sie unschwer erkennen, sind Ihrer Phantasie keine Grenzen gesetzt, und ich würde mich freuen, von Ihnen mal so einen witzigen Trickablauf sehen zu können. Auf einem ähnlichen Prinzip basiert auch ein Programm von Hans Grigat in Happy-Computer, Ausgabe Nummer 11 (1983), Seite 99 ff. Überhaupt lohnt es sich, sich dieses Programm mal genau anzusehen, weil hier die anfangs erwähnte Möglichkeit der Sprite-Daten-Verschiebung genutzt wurde.

Wer hat Vorfahrt? Prioritäten

Wir wollen uns jetzt noch um die Beziehung von Sprites zu ihrer Umwelt (also zu anderen Sprites und/oder zu Zeichen auf dem Bildschirm) kümmern. Sie erinnern sich vielleicht an unseren Versuch, unser kleines Test-Sprite über den Bildschirm zu bewegen und an ein Ergebnis davon, nämlich, daß auch Schwarzweiß-Sehern plötzlich bei Eingabe der abgedruckten Bildschirmposition das Sprite Nummer 1 sichtbar war. Wenn also — wie in diesem Fall — zwei Sprites sich überdecken, welches von beiden wird dann gezeigt? Siehe dazu das Bild 7, auf dem die Situation abgebildet ist. Wir sehen da: an den Stellen, wo Sprite 1 Bits mit dem Wert 0 vorliegen hat, ist Sprite 2 zu sehen. Wo aber der Bitwert des Sprite 1 gleich 1 ist, wird Sprite 2 durch Sprite 1 verdeckt. Das MOB 1 hat eine höhere Priorität als MOB 2. Der VIC-II-Chip organisiert die Prioritäten von Sprites untereinander also in folgender Weise: Höchste Priorität hat Sprite 0, dann folgt Sprite 1, Sprite 2 und so weiter bis zum Schlußlicht Sprite 7.

Wenn Sie also Programme planen, in denen Sprites aneinander vorbeiziehen sollen, denken Sie daran, daß an diese Vorfahrtsregelung nichts zu ändern ist. Unter Umständen muß man dann die Sprite-Zeiger umwechseln, um die Prioritäten umzukehren: Man macht dann beispielsweise für den Augenblick der Überschneidung aus Sprite 7 zum Beispiel Sprite 1 und umgekehrt.

Eine andere Vorfahrtsregelung gilt, wenn Sprites und Bildschirmzei-

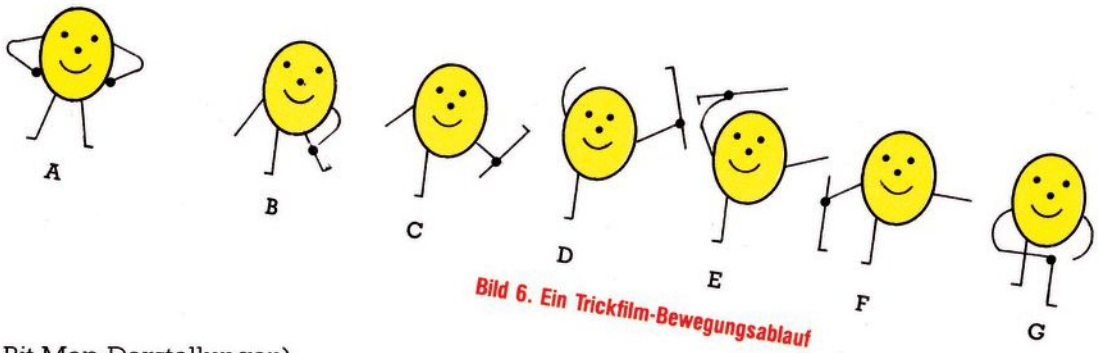

Bild 6. Ein Trickfilm-Bewegungsablauf

len (oder Bit-Map-Darstellungen) aufeinandertreffen. Hier haben wir ein Kontrollregister zur Hand (mal wieder eines!), Speicherstelle 53275, wo wieder jedem Sprite ein Bit entspricht (also Bit 0 entspricht Sprite 0 und so weiter). Wenn nun dieses Bit den Wert 0 hat, steht das dazugehörige MOB vor den Bildschirmdarstellungen, andernfalls verkrümelt es sich dahinter.

Das wollen wir uns mal ansehen! Starten Sie das Programm nochmal und wenn Sie an die Farbabfrage kommen, geben Sie dem Sprite 0 die Farbe 0 (= schwarz). Beenden Sie das Programm in der angegebenen Weise und wenn sich READY gemeldet hat. LISTen Sie das ganze. Jetzt steht unser schwarzes Pentagramm vor dem Listing. Geben sie nun im Direktmodus ein:
POKE 53275,1 (RETURN). Siehe da: Sprite 0 versteckt sich hinter dem Listing.

Wir können also zusammenfassen: Überall dort, wo auf dem Bildschirm (durch Buchstaben oder andere Zeichen) ein Bit gesetzt ist, verschwindet dahinter ein gegebenenfalls vorhandener Sprite-Bildpunkt (wenn natürlich das zum Sprite gehörige Bit im Register 53275 gesetzt ist). Nur dort, wo auf dem Bildschirm ein Bit nicht gesetzt ist (also 0 ist), sieht man einen dort vorhandenen Sprite-Bildpunkt.

Etwas komplizierter liegen die Verhältnisse, wenn wir das MOB im Mehrfarben-Modus vorliegen haben. Hier werden nämlich auch Bildschirmbitpaare 01 (von Zeichen oder Bit-Map-Darstellungen) genauso behandelt wie Bitpaare 00. Das heißt, auch an solchen Stellen wird ein eventuell vorhandenes Sprite-Bitpaar dargestellt.

Zusammenstöße: Kollisionsregister

Erfreulicherweise haben die Software-Planer des C 64 auch zwei Möglichkeiten vorgesehen, Zusammenstöße in Registern abzufragen. Weil es zwei Sorten von Zusammenstöße gibt (Sprite kollidiert mit Sprite und Sprite kollidiert mit Zeichen), kann man zwei Register abfragen.

Da hätten wir zunächst das Sprite-Sprite-Kollisions-Register 53278. Auch hier gehört zu jedem Bit ein Sprite, wie bei den anderen Kontroll-Registern. Von einem Zusammenstoß spricht man in Sprite-Kreisen

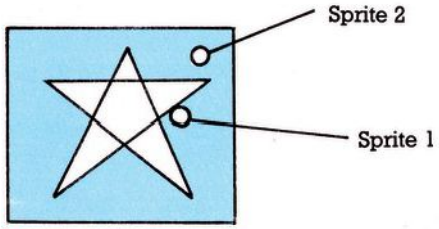

Bild 7. Sprite 1 überdeckt Sprite 2

immer dann, wenn undurchsichtige Teile der MOBs aufeinandertreffen. Überlappen sich nur die Teile, deren Bits bei der Sprite-Definition auf 0 gesetzt wurden, so zählt das nicht als Kollision. Die Bits der Sprites, die in den Zusammenstoß verwickelt sind, werden auf 1 gesetzt, zum Beispiel erzeugt ein Zusammenstoß von Sprite 0 mit Sprite 1 folgenden Inhalt des Registers 53278: 00000011 = dezimal 3.

Wenn in einem Supercrash acht Sprites kollidieren, steht im Sprite-Sprite-Kollisionsregister 255. Übrigens werden auch Zusammenstöße registriert, die außerhalb des sichtbaren Bildschirms liegen (siehe Bild 4).

Wenn ein Sprite mit Bildschirmdarstellungen (Zeichen und so weiter) zusammenstößt, wird in Register 53279 das zu ihm gehörende Bit gesetzt. Stößt also Sprite Nummer 1 mit zum Beispiel einem A zusammen, dann liest man aus dem Register 53279: 00000010 = dezimal 2. Apropos lesen: Die Register 53278 und 53279 sind so gestaltet, daß sie nach dem Herauslesen (PEEKen) gelöscht sind! Deswegen empfiehlt es sich, wenn man diese Inhalte danach noch braucht, sie in Variablen abzulegen. Auch bei dieser Sorte Zusammenstößen zählen nur diejenigen, bei denen undurchsichtige Sprite-Teile kollidieren. Wenn übrigens Bildschirminhalte horizontal aus dem sichtbaren Bereich »herausgescrollt« worden sind (zum »Scrollen« kommen wir in der nächsten Folge), und dabei ein Zusammenstoß mit einem Sprite geschieht, wird ebenfalls ein Bit in 53279 gesetzt.

Manfred Thoma hat das anliegende Programm »Sprity« geschrieben, mit dem Sie einige Sprite-Eigenschaften und die dazugehörigen Registerveränderungen beobachtet werden können.

Obwohl es zu den MOBs noch einiges zu sagen gäbe (wie kommen sie auf den Bildschirm, schnelles Steuern und so weiter), soll das Thema hiermit abgeschlossen sein. Ich denke, daß wir in dieser Folge die kleinen Kobolde schon weitgehend entzaubern konnten mit Hilfe unserer Pentagramme. Zu den Sprites gibt es mehr Literatur als zu vielen anderen C 64-Themen. Hier eine kleine Auswahl:

— Zwei Artikel wurden schon genannt (von H. Grigat und H. Kunz)
— Herbert Kunz hat auch schon etwas über schnelles Bewegen von MOBs geschrieben im 64'er, Ausgabe 4/83 auf Seite 70 f.
— Einen Überblick vor allem über die Anwendung von Sprites in Spielen geben Schneider und Ebert in den Bänden 1 und 3 des Commodore 64-Buches. Erschienen 1984 in Haar bei München im Markt & Technik Verlag. Diese beiden Bände sind auch von den sehr gut überschaubaren Programmbeispielen her zu empfehlen.

(Heimo Ponnath/aa)