

Spaltennummer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	Zahlen-codes
Wert	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	
Zeile 0																									
Zeile 1																									
Zeile 2																									
Zeile 3																									
Zeile 4																									
Zeile 5																									
Zeile 6																									
Zeile 7																									
Zeile 8																									
Zeile 9																									
Zeile 10																									
Zeile 11																									
Zeile 12																									
Zeile 13																									
Zeile 14																									
Zeile 15																									
Zeile 16																									
Zeile 17																									

Bild 6. Ein Sprite-Entwurfsblatt

Betrachten Sie nun das Sprite-Entwurfsblatt in Bild 6. Es enthält die erforderlichen 21 Zeilen zu je 24 Bildpunkten, wobei jede Zeile in drei Gruppen zu acht Bildpunkten eingeteilt und jeder Bildpunkt mit seinem Wert versehen ist. Zusätzlich enthält jede Zeile drei Felder, in welche die Zahlen-codes für die drei Gruppen zu jeweils acht Bildpunkten eingetragen werden können.

Da insgesamt 21 Zeilen zu je drei Zahlencodes vorhanden sind, wird ein Sprite also durch 63 Zahlen definiert, die Zeile für Zeile von oben nach unten und von links nach rechts in den C 64 eingegeben werden müssen. Um nun ein Sprite zu definieren, gehen Sie folgendermaßen vor:

1. Kopieren Sie sich das Sprite-Entwurfsblatt (Bild 6).
2. Entwerfen Sie ein Sprite und zeichnen Sie es ein, indem Sie für jeden darzustellenden Bildpunkt das entsprechende Kästchen ausfüllen.
3. Berechnen Sie für jede Zeile die Zahlen-codes für die drei Bildpunktgruppen und tragen Sie sie ein.
4. Geben Sie die Zahlen-codes in der richtigen Reihenfolge in den Computer ein.

Bild 7 stellt ein ausgefülltes Entwurfsblatt für ein kleines Sprite-Männchen dar. Lesen Sie erst weiter, wenn Ihnen der Aufbau dieses Entwurfsblattes vollständig klar ist und Sie die Berechnung der eingetragenen Werte nachvollziehen können.

Eilen Sie nun zum nächsten Kopierautomaten und machen Sie sich Kopien von dem Sprite-Entwurfsblatt. Entwerfen Sie anschließend einige Sprites. Sind Sie mit einem Entwurf zufrieden, so berechnen Sie bitte die entsprechenden 63 Zahlencodes. Sie benötigen die Daten im zweiten Kapitel.

Spaltennummer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	Zahlen-codes
Wert	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	
Zeile 0																									0 60 0
Zeile 1																									0 36 0
Zeile 2																									0 102 24
Zeile 3																									0 102 56
Zeile 4																									0 36 56
Zeile 5																									0 60 16
Zeile 6																									0 24 16
Zeile 7																									0 24 16
Zeile 8																									15 255 240
Zeile 9																									8 126 0
Zeile 10																									8 126 0
Zeile 11																									8 24 0
Zeile 12																									28 24 0
Zeile 13																									28 24 0
Zeile 14																									24 60 0
Zeile 15																									0 60 0
Zeile 16																									0 36 0
Zeile 17																									0 36 0
Zeile 18																									0 36 0
Zeile 19																									3 231 192
Zeile 20																									3 231 192

Bild 7. Beispiel für ein ausgefülltes Sprite-Entwurfsblatt

Ihr erstes Sprite-Programm

Beginnen Sie mit einem kleinen Programm zur Darstellung eines einfachen Sprites. Bild 8 zeigt den Programmaufbau, Listing 1 das Programm »Ein einfaches Sprite«.

Versuchen Sie, den Programmablauf zunächst anhand des Ablaufplanes zu verstehen. Geben Sie dann das Programm ein. Speichern Sie das Programm, ehe Sie es starten. Durch die Betätigung einer beliebigen Taste wird das Programm beendet.

```

1000 rem *** ein einfaches sprite ***      <045>
1010 :                                     <224>
1020 :                                     <234>
1030 rem ** ausgabe programmmeldung        <013>
1040 :                                     <000>
1050 print "[clr,8down]bitte warten";      <152>
1060 :                                     <020>
1070 :                                     <030>
1080 rem ** sprite-daten einlesen          <073>
1090 :                                     <050>
1100 for n = 896 to 958                    <018>
1110 :   poke n,255                        <198>
1120 next n                                <228>
1130 :                                     <090>
1140 :                                     <100>
1150 rem ** sprite-steuerung               <171>
1160 :                                     <120>
1170 print "[clr]"; :rem bilds.loeschen    <113>
1180 poke 2040,14 :rem datenzeiger         <111>
1190 :                                     <150>
1200 vic = 53248 :rem vic-baustein         <045>
1210 poke vic,170 :rem horizont. pos.      <122>
1220 poke vic+1,120:rem vertikale pos.     <248>
1230 poke vic+39,13:rem gruenes sprite    <197>
1240 poke vic+21,1 :rem sprite 0 ein       <003>
1250 :                                     <210>
1260 :                                     <220>
1270 rem ** prog.ende nach tastendruck    <045>
1280 :                                     <242>
1290 get kp$                               <125>
1300 if kp$ = "" then 1290                 <075>
1310 :                                     <016>
1320 :                                     <026>
1330 rem ** register zuruecksetzen         <073>
1340 :                                     <046>
1350 poke vic+21,0 :rem ruecksetzung       <021>
1360 poke vic+39,0 :rem in umgekehrter    <119>
1370 poke vic+1,0 :rem reihenfolge...     <247>
1380 poke vic,0                             <157>
1390 :                                     <096>
1400 end                                   <132>

```

Listing 1. Das Programm »Ein einfaches Sprite«

Der erste Programmteil besteht aus nur einer Zeile:

```
1050 PRINT CHR$(147):PRINT "BITTE WARTEN";
```

Dieser Basic-Befehl löscht den Bildschirm und druckt die Programmmeldung »BITTE WARTEN«, um dem Benutzer anzuzeigen, daß der C 64 mit der Arbeit begonnen hat, denn nichts ist beunruhigender als ein Programm, das nach dem Start keinerlei Aktivitäten zeigt.

Im zweiten Teil des Programms werden die 63 Sprite-Codes eingelesen:

```

1100 FOR N = 896 TO 958
1110 :   POKE N, 255
1120 NEXT N

```

Für dieses erste Programm wurde das einfachste Sprite ausgewählt, das möglich ist: alle Bildpunkte werden angezeigt. Daher haben alle 63 Codes den Wert 255. Der zweite Programmteil, die Schleife von 1100 bis 1120, schreibt diesen Wert nacheinander in die Speicherzellen 896 bis 958.

In Teil 3 des Programms wird die Hauptarbeit geleistet. Betrachten Sie zunächst die Zeilen 1170-1200:

```
1170 PRINT CHR$(147); :REM BILDS.LOESCHEN
1180 POKE 2040,14 :REM DATENZEIGER
1190 :
1200 VIC = 53248 :REM VIC-BAUSTEIN
```

Zeile 1170 löscht den Bildschirm. In Zeile 1180 wird dem Rechner dann mitgeteilt, wo die Sprite-Daten zu finden sind, nämlich in den Speicherzellen 896 bis 958.

Wie ist das möglich? Der C 64 kann maximal acht Sprites, die von 0 bis 7 numeriert sind, gleichzeitig auf dem Bildschirm darstellen. Erhält er zum Beispiel den Befehl, Sprite 0 anzuzeigen, so berechnet er die Adresse, ab der die Daten für Sprite 0 abgelegt sind, indem er den Wert aus Speicherstelle 2040 mit 64 multipliziert. Da das Programm in Zeile 1180 den Wert 14 in diese Zelle geschrieben hat, ergibt sich aus $14 \times 64 = 896$ die Anfangsadresse der Sprite-Daten.

In Zeile 1200 erfolgt die Zuweisung des Wertes 53248 an die Variable VIC. Um welche Variable handelt es sich dabei? Hierzu müssen wir etwas weiter ausholen.

Die erstaunlichen Grafikfähigkeiten des C 64 beruhen auf einem speziell entwickelten Grafikbaustein, dem 6567-VIC.II (Video Interface Chip II, eine Weiterentwicklung des 6560-VIC aus dem Computer VC 20), kurz VIC.II genannt. Der VIC.II steuert die Textausgabe (40 Zeichen in 25 Zeilen) ebenso wie die hochauflösende Grafik (200 Zeilen zu 320 Bildpunkten) und die Darstellung der acht Sprites. Nur wer noch die Anfänge der Computergrafik auf Heimcomputern miterlebt hat, kann ermessen, welchen Fortschritt der VIC.II darstellt.

Zur Steuerung des VIC.II enthält dieser 47 adressierbare Speicherzellen, auch Register genannt. Die Anfangsadresse ist 53248, die Endadresse 53294. Nähere Angaben über die Register finden Sie im Anhang Ihres C 64-Handbuchs.

Zeile 1200 weist also der Variablen VIC den Wert 53248 zu. Damit können Sie jedes der 47 VIC-Register einfach adressieren, indem Sie die Registernummer (0-46) zu dem Wert von VIC addieren. Dies geschieht in den Zeilen 1210-1240:

```
1210 POKE VIC,170 :REM HORIZONT. POS.
1220 POKE VIC+1,120 :REM VERITKALE POS.
1230 POKE VIC+39,13 :REM GRUENES SPRITE
1240 POKE VIC+21,1 :REM SPRITE 0 EIN
```

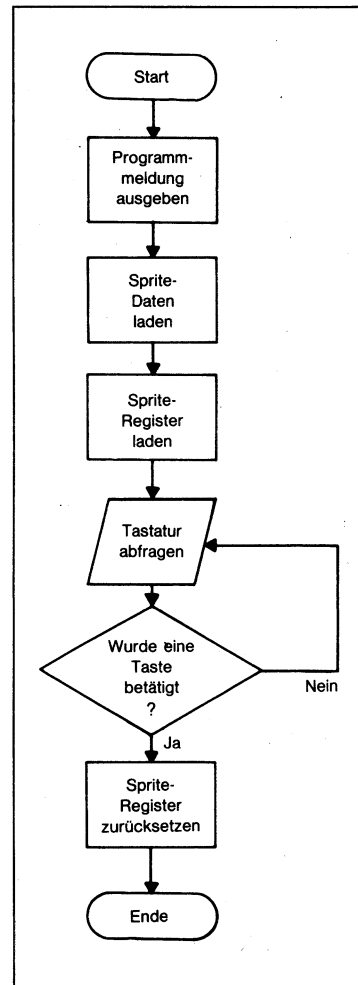
Register 0 bestimmt die horizontale Position von Sprite 0. In Zeile 1210 wird dieser Wert auf 170 gesetzt. Die vertikale Position des Sprites steht in Register 1. Es erhält in Zeile 1220 den Wert 120. Der Sprite-Anfang liegt damit nahe der Bildschirmmitte. Register 39 definiert die Farbe aller darzustellenden Bildpunkte von Sprite 0 und erhält den Wert 13 für Hellgrün (eine Liste der verfügbaren Farben und ihrer Codes finden Sie ebenfalls im Anhang des Handbuchs).

Sie haben nun die Sprite-Daten eingelesen, dem C 64 mitgeteilt, wo er diese Daten finden kann, die Sprite-Position zugeordnet und die Farbe der Bildpunkte festgelegt. Es fehlt nur noch die Anweisung zur Darstellung des Sprites. Diese Anweisung steht in Zeile 1240. Register 21 steuert die Anzeige der Sprites. Sobald es den Wert 1 erhält, wird Sprite 0 auf dem Bildschirm angezeigt.

Nun zum vierten Programmteil:

```
1290 GET KP$
1300 IF KP$ = "" THEN 1290
```

Zeile 1290 überprüft die Tastatur. In Zeile 1300 wird getestet, ob eine Taste betätigt worden ist. Wurde kein Zeichen eingegeben, verzweigt das Programm wieder nach Zeile 1290 und überprüft die Tastatur erneut. Wird schließlich eine Taste gedrückt, so beginnt das Programm mit dem fünften und letzten Programmteil.



**Bild 8. Zum besseren Verständnis:
Der Ablaufplan für das Programm
»Ein einfaches Sprite«**

Es ist allgemein angebracht, Dinge so zu hinterlassen, wie man sie angetroffen hat, insbesondere bei der Programmierung von Computern. In den Zeilen 1350-1380 werden die VIC-Register wieder auf 0 zurückgesetzt:

```
1350 POKE VIC+21,0
:REM RUECKSETZUNG
1360 POKE VIC+39,0
:REM IN UMGEKEHRTER
1370 POKE VIC+1,0
:REM REIHENFOLGE...
1380 POKE VIC,0
```

Beachten Sie, daß die Register, im Vergleich zur Wertzuweisung in den Zeilen 1210-1240, in umgekehrter Reihenfolge zurückgesetzt werden.

Sie lernen den Umgang mit Sprites am besten, indem Sie bereits vorhandene Programme testweise modifizieren und dann verfolgen, welche Änderungen auftreten. Hier sind einige Vorschläge für die Modifikation des Programms »Ein einfaches Sprite«:

- Ändern Sie den Sprite-Code in Zeile 1110.
- Ändern Sie die Positionsangaben in Zeile 1210 und 1220.
- Ändern Sie den Farbcode in Zeile 1230.

Bei allen Experimenten sollten Sie niemals das Original-Programm »Ein einfaches Sprite« auf der Kassette oder Diskette überschreiben, denn Sie werden es in den folgenden Abschnitten noch als Rahmenprogramm für weitere Versuche mit Sprites benötigen. Ansonsten können und sollen Sie mit diesem Programm ruhig alle Experimente anstellen, die Ihnen im Zusammenhang mit Sprites gerade in den Sinn kommen. Arbeiten Sie dabei ruhig mit Variablen und Programmschleifen. Beispielsweise könnten Sie in Zeile 1230 für die Sprite-Farbe die Variable F verwenden, die Sie vorher per INPUT einlesen:

```
»1230 INPUT F:POKE VIC+39,F«
```

Die Positionierung von Sprites

Bei der Positionierung von Sprites teilen Sie dem Computer mit, an welcher Bildschirmposition der linke obere Bildpunkt des Sprites angezeigt werden soll. Der C 64 kann 200 Zeilen mit 320 Bildpunkten darstellen. Mit Hilfe der VIC-Register ist jedoch eine horizontale Positionierung bis zum Wert 512 und eine vertikale Positionierung bis zum Wert 256 möglich. Auf diese Weise können Sie Sprites in den Bildschirm hinein- oder herausgleiten lassen.

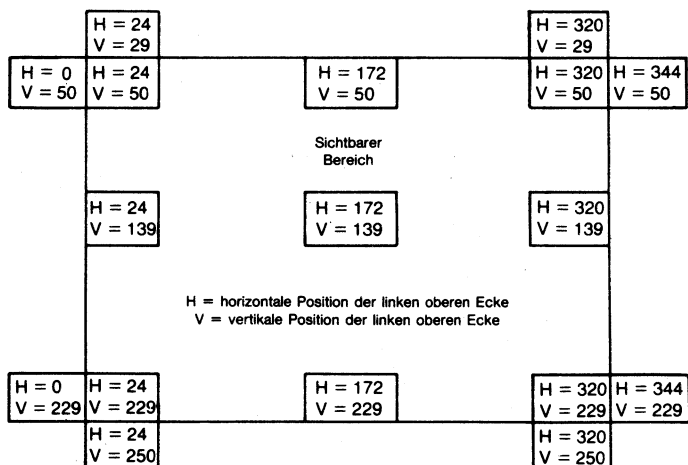


Bild 9. Einige wichtige horizontale und vertikale Bildschirmpositionen für Sprites in Normalgröße

In Bild 9 sind die Koordinaten einiger Randpositionen dargestellt. Sprites mit vertikalen Positionswerten kleiner als 30 liegen oberhalb des sichtbaren Bereichs. Sprites mit horizontalen Positionswerten zwischen 24 und 320 liegen dagegen immer innerhalb des Bildschirmbereichs.

Ein Sprite-JoJo

Laden Sie bitte das Programm »Ein einfaches Sprite« (Listing 1) und geben Sie dann die Zeilen aus Listing 2 ein. Dadurch werden einige Zeilen des Originalprogramms überschrieben und andere hinzugefügt. Benutzen Sie bitte genau die angegebenen Zeilennummern. Speichern und starten Sie dann das neue Programm.

```

1000 rem *** ein sprite-jojo          <176>
1220 poke vic+1,80 :rem vertikale pos <026>
1251 :                                <211>
1252 rem runter, dann hoch           <143>
1253 :                                <213>
1254 for vp = 80 to 200               <112>
1255 :   poke vic+1,vp                <170>
1256 next vp                          <177>
1257 :                                <217>
1258 for vp = 199 to 81 step -1       <051>
1259 :   poke vic+1,vp                <174>
1260 next vp                          <181>
1261 :                                <221>
1262 :                                <222>
1300 if kp$ = "" then 1254            <139>

```

Listing 2. Änderungen und Ergänzungen zur Umformung des Programms »Ein einfaches Sprite« in das Programm »Ein Sprite-JoJo«

Das Sprite bewegt sich wie ein JoJo auf und ab. Dieser Effekt wird durch die Zeilen 1254-1256 bewirkt:

```

1254 FOR VP = 80 TO 200
1255 : POKE VIC+1,VP
1256 NEXT VP

```

In dieser Schleife durchläuft die vertikale Position die Werte 80 bis 200, wodurch sich das Sprite nach unten bewegt. In der anschließenden Schleife 1258-1260 wird diese Bewegung durch die vertikalen Positionswerte 199 bis 81 dann wieder umgekehrt:

```

1258 FOR VP = 199 TO 81 STEP -1
1259 : POKE VIC+1,VP
1260 NEXT VP

```

Das Sprite bewegt sich wieder nach oben. Die neue Zeile 1300 bewirkt, daß dieses Auf und Ab bis zur Betätigung einer beliebigen Taste fortgesetzt wird.

```
1300 IF KP$ = "" THEN 1254
```

Die Definition von 512 horizontalen Positionen

Sicherlich haben Sie sich bereits gefragt, wie man eine horizontale Position größer als 255 definieren kann, obwohl ein Register nur Werte von 0 bis 255 aufnimmt.

Die Lösung ist einfach. Der VIC.II benutzt zur Festlegung der horizontalen Position eines Sprites nämlich nicht nur ein, sondern zwei Register, wobei das zweite allerdings kein vollwertiges Register ist, da es nur die Werte 0 und 1 aufnehmen kann. Der Wert 1 bedeutet, daß eine horizontale Position größer als 255 angesteuert werden soll. In diesem Fall addiert der C 64 den Wert 256 zum aktuellen Wert des ersten Registers. Enthält also zum Beispiel das erste Register den Wert 33 und das zweite Register den Wert 1, so ergibt sich für die anzusteuende horizontale Position der Wert 298 (265 + 33). Enthält das zweite Register den Wert 0, so gilt nur der Wert des ersten Registers. Tabelle 1 zeigt einige Beispiele für horizontale Positionen zwischen 0 und 511.

1. Horizontales Register	2. Horizontales Register	Sprite Position
0	0	0
24	0	24
125	0	125
255	0	255
0	1	256
20	1	276
64	1	320
88	1	344
255	1	511

Tabelle 1. Einige Werte der horizontalen Positionsregister für Sprite-Positionen zwischen 0 und 511

Horizontale Bewegungen

Laden Sie bitte das Programm »Ein einfaches Sprite« (Listing 1), und fügen Sie die Zeilen aus Listing 3 hinzu. Speichern und starten Sie dann das neue Programm.

```

1000 rem *** horizontale bewegung *** <172>
1210 poke vic,64 :rem horizont. pos. <095>
1220 poke vic+1,139:rem vertikale pos. <068>
1251 :                                <211>
1252 rem rechts, dann links          <015>
1253 :                                <213>
1254 for hp = 64 to 280 step 2        <129>
1255 :   sf = (hp > 255)              <245>
1256 :   poke vic,hp + (sf * 256)     <097>
1257 :   poke vic+16, sf * (-1)      <138>
1258 next hp                          <067>
1259 :                                <219>
1260 for hp = 278 to 66 step-2        <099>
1261 :   sf = (hp > 255)              <251>
1262 :   poke vic,hp + (sf * 256)     <103>
1263 :   poke vic+16, sf * (-1)      <144>
1264 next hp                          <073>
1265 :                                <225>
1266 :                                <226>
1300 if kp$ = "" then 1254            <139>

```

Listing 3. Änderungen und Ergänzungen zur Umwandlung des Programms »ein einfaches Sprite« in das Programm »Horizontale Bewegung«

Ehe wir uns näher mit der Funktion der neuen Programmzeilen befassen, wollen wir noch einen kleinen Abstecher in die Welt der Logik unternehmen.

Bevor Sie ein Sprite an eine neue horizontale Position bewegen, müssen Sie immer folgende Aussage mit »wahr« oder »falsch« bewerten: Der neue Positionswert ist größer als 255. Von Ihrer Antwort hängt es ab, welcher Wert in das zweite horizontale Positionsregister geschrieben wird.

Das Commodore-Basic bewertet wahre Aussagen mit dem Wert -1 und falsche Aussagen mit dem Wert 0. Dieser Wahrheitswert kann einer Variablen zugeordnet werden. Dazu ein Beispiel:

```
100 LET WW = (5 > 3)
```

Da 5 größer ist als 3, ist die Aussage (5 > 3) wahr, und WW erhält den Wert -1. Ein weiteres Beispiel:

```
100 LET WW = (36 = 21)
```

In diesem Fall erhält WW den Wert 0, denn die Aussage (36 = 21) ist falsch.

Das Sprite soll sich nun nach rechts bewegen. Durch eine kleine Änderung der Zeilen 1210-1220 beginnt diese Bewegung an einer neuen Position:

```
1210 POKE VIC,64 :REM HORIZONT. POS.
1220 POKE VIC+1,139:REM VERTIKALE POS.
```

Betrachten Sie nun bitte die Zeilen 1254-1258:

```
1254 FOR HP = 64 TO 280 STEP 2
1255 : SF = (HP > 255)
1256 : POKE VIC,HP + (SF * 256)
1257 : POKE VIC+16, SF * (-1)
1258 NEXT HP
```

Die Zeilen 1254-1258 bilden eine Schleife, in der die horizontale Position des Sprites in Zweierschritten die Werte von 64 bis 280 durchläuft. Bei jedem Schleifendurchgang wird dabei in Zeile 1255 geprüft, ob die neue Position den Wert 255 überschritten hat. Jeweils nach dem Wahrheitswert der überprüften Aussage in Zeile 1255 erfolgt dann die Positionierung in Zeile 1256 durch Eintragung der entsprechenden Werte in die beiden horizontalen Register.

Hat zum Beispiel HP den Wert 125, dann erhält der Größtenfaktor SF in Zeile 1255 den Wert 0. Somit schreibt Zeile 1256 den Wert 125 + (0*256), also die Zahl 125, in das erste horizontale Register des Sprites. Das zweite horizontale Register erhält in Zeile 1257 den Wert -1*0=0. Beide Register enthalten damit die richtigen Werte für eine horizontale Position, die kleiner ist als 256.

Nun zu einer Position größer als 256. Hat HP beispielsweise den Wert 276, dann erhält SF in Zeile 1255 den Wert -1. Somit schreibt Zeile 1256 die Zahl 276+(-1*256) = 276-256 = 20 in das erste horizontale Register. Das zweite horizontale Register erhält in Zeile 1257 den Wert -1*-1=1. Wiederum enthalten beide Positionsregister somit die richtigen Werte.

Sollte Ihnen die Bedeutung der Zeilen 1245-1258 noch nicht ganz klar sein, so lesen Sie bitte noch einmal die letzten beiden Abschnitte, und überprüfen Sie die Formeln anhand einiger Werte aus Tabelle 1.

Betrachten Sie nun die Zeilen 1260-1264.

```
1260 FOR HP = 278 TO 66 STEP -2
1261 : SF = (HP > 255)
1262 : POKE VIC, HP + (SF * 256)
1263 : POKE VIC+16, SF * (-1)
1264 NEXT HP
```

Dieses Mal durchläuft die Schleife die horizontalen Positionswerte von 278 bis 66, wiederum in Zweierschritten. Das Sprite bewegt sich dadurch nach links. Die Zeilen 1261-1263 sind mit den Zeilen 1255-1257 identisch, da sowohl bei der Links- als auch bei der Rechtsbewegung dieselben horizontalen Register benutzt werden.

Schließlich haben Sie noch die Zeile 1300 verändert, um an den Anfang des für die horizontale Bewegung zuständigen Programnteils zu springen:

```
1300 IF KP$ = "" THEN 1254
```

Sprites vergrößern

Das Sprite in dem Programm »Ein einfaches Sprite« ist etwas langweilig. Um es interessanter zu gestalten, laden Sie bitte das Programm aus Listing 1 noch einmal und fügen die in Listing 4 dargestellten neuen Zeilen und Änderungen hinzu. Nach Abschluß dieser Arbeit speichern Sie das Programm (Sprite-Entwurf) dann wie üblich und starten es anschließend.

Anstelle des Quadrates erscheint jetzt die in Bild 7 entworfene und codierte Figur. Betrachten Sie dazu die neue Ladeschleife für die Sprite-Daten in den Zeilen 1100-1120:

```
1100 FOR N = 896 TO 958
1105 : READ SPDTA
1110 : POKE N, SPDTA
1120 NEXT N
```

Im alten Programm haben Sie in jede Speicherzelle den Wert 255 geschrieben, wodurch alle Bildpunkte sichtbar wurden. Nun benutzen Sie demgegenüber in Zeile 1005 eine READ-Anweisung, um die in DATA-Zeilen abgelegten Bildpunktdaten in die Variable SPDTA zu lesen und anschließend in den Speicher zu schreiben.

```
1000 rem *** sprite-entwurf *** <227>
1105 : read spdta <056>
1110 : poke n, spdta <088>
1121 : <081>
1122 data 0, 60, 0, 0, 36, 0 <104>
1123 data 0, 102, 24, 0, 102, 56 <195>
1124 data 0, 36, 56, 0, 60, 16 <223>
1125 data 0, 24, 16, 0, 24, 16 <189>
1126 data 15, 255, 240, 8, 126, 0 <118>
1127 data 8, 126, 0, 8, 24, 0 <037>
1128 data 28, 24, 0, 28, 24, 0 <196>
1129 data 24, 60, 0, 0, 60, 0 <020>
1130 data 0, 36, 0, 0, 36, 0 <145>
1131 data 0, 36, 0, 3, 231, 192 <216>
1132 data 3, 231, 192 <138>
1230 poke vic+39,1 :rem weisse farbe <242>
```

Listing 4. Änderung zum Programm »Sprite-Entwurf«

Betrachten Sie nun die elf DATA-Zeilen. Sie enthalten alle in Bild 7 berechneten Zahlencodes. Beachten Sie die Schreibweise der Daten: sie sind von links nach rechts, Zeile für Zeile, aus Bild 7 übernommen.

Schließlich wird durch die neue Version von Zeile 1230 das Sprite in weißer Farbe dargestellt, wodurch es besser sichtbar ist. Da sowohl die Technik eines Farbfernsehers als auch die des Commodore 64 nicht vollkommen ist, sind bestimmte Farben auf einigen Hintergrundfarben unterschiedlich gut zu erkennen. Sie müssen eine Weile experimentieren, um eine günstige Farbkombination zu finden.

Es gibt zwei Probleme im Zusammenhang mit dem letzten Programm. Erstens ist das Sprite zu klein, um in allen Details sichtbar zu sein, und zweitens ist dieser Entwurf nicht der Ihre.

Beginnen wir mit dem zweiten Problem. Zu Anfang dieses Kurses haben Sie mehrere Sprites entworfen und die 63 Zahlencodes für Ihren besten Entwurf berechnet. Benutzen Sie nun diese hart erarbeiteten Daten.

Laden Sie dazu das letzte Programm, »Sprite-Entwurf«, und lassen Sie sich die Zeilen 1122 bis 1132 ausgeben. Bewegen Sie dann den Cursor in die jeweilige Zeile und überschreiben Sie die alten Daten mit Ihren eigenen Bildpunktcodes.

Nun zum ersten Problem. Der VIC.II-Baustein ermöglicht sowohl eine horizontale als auch eine vertikale Vergrößerung von Sprites. In vergrößerten Sprites sind die Einzelheiten besser sichtbar. Fügen Sie daher dem Programm »Sprite-Entwurf« die folgenden Zeilen hinzu und speichern Sie es anschließend.

```
1000 REM *** EIN GROSSES SPRITE ***
1233 POKE VIC+23,1 :REM VERLAENGERN
1236 POKE VIC+29,1 :REM VERBREITERN
1353 POKE VIC+29,0
1356 POKE VIC+23,0
```

Sobald Sie das Programm gestartet haben, werden alle Einzelheiten Ihres Entwurfes sichtbar. Klopfen Sie sich auf die Schulter, ehe Sie sich näher mit der Vergrößerung von Sprites befassen.

Ein Sprite kann sowohl in doppelter Breite als auch in doppelter Höhe dargestellt werden. Das 30. VIC.II-Register mit der Adresse VIC+29 steuert die horizontale Vergrößerung aller acht Sprites. Der Wert 1 in diesem Register bewirkt, daß Sprite 0 in doppelter Breite ausgegeben wird. Enthält das Register den Wert 0, so hat Sprite 0 seine normale Breite.

Das 24. VIC.II-Register mit der Adresse VIC+23 steuert die vertikale Vergrößerung der acht Sprites. Wenn Sie eine 1 in dieses Register schreiben, verdoppelt sich die Höhe von Sprite 0. Enthält das Register eine 0, wird Sprite 0 in normaler Höhe ausgegeben.

Die Werte in den horizontalen und vertikalen Registern definieren auch für ein vergrößertes Sprite die Position seiner linken Ecke auf dem Bildschirm. Bild 10 zeigt einige wichtige Bildschirmpositionen für vergrößerte Sprites. Vergleichen Sie dieses Schema mit der Darstellung in Bild 9.

In den Zeilen 1233 und 1236 des Programms »Ein großes Sprite« wird der Wert 1 in beide Vergrößerungsregister geschrieben:

```
1233 POKE VIC+23,1: REM VERLAENGERN
1236 POKE VIC+29,1: REM VERBREITERN
```

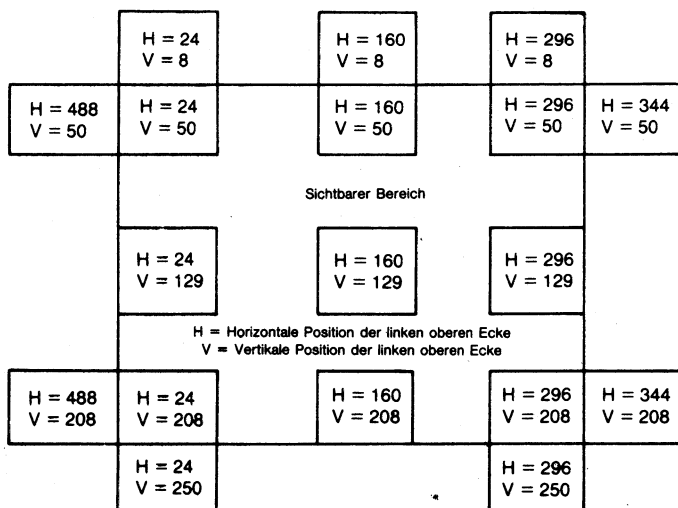


Bild 10. Einige Positionswerte für Sprites doppelter Größe

Dadurch vergrößert sich Sprite 0 in beide Richtungen. Am Ende des Programms, in den Zeilen 1353 und 1356, wird es durch den Wert 0 dann wieder auf seine Normalgröße zurückgesetzt:

```
1353 POKE VIC+29,0
1356 POKE VIC+23,0
```

Zusammenfassung

Im ersten Kapitel haben Sie gelernt:

- was Bildpunkte und was Sprites sind,
- wie Sie Ihr eigenes Sprite entwerfen und die zugehörigen 63 Zahlencodes berechnen können,
- wie die Sprite-Daten zu laden und die VIC.II-Register zu definieren sind, um ein einfaches Sprite auf dem Bildschirm darzustellen,
- wie die Position, Farbe und Größe eines Sprites anzugeben ist,
- wie ein Sprite in horizontaler und in vertikaler Richtung zu bewegen ist.

Nun haben Sie Gelegenheit, Ihr bisher erworbenes Wissen zu überprüfen. Vervollständigen Sie bitte die Sätze im Selbsttest und programmieren Sie die in den Programmierübungen geforderten Ergänzungen und Änderungen. Benutzen Sie dann die in diesem Kapitel vorgestellten Techniken und Hinweise, um eigene Programme zu schreiben. Nur Übung macht den Meister!

Selbsttest

Die Antworten zu diesem Selbsttest finden Sie im Anschluß an den Übungsteil.

1. Ein Sprite besteht aus einem Muster von 504, das über den Bildschirm bewegt werden kann.
2. Zur Codierung eines Sprite-Musters wird jede der 21 Zeilen in drei Gruppen zu je Bildpunkten aufgeteilt.
3. Um ein Sprite sichtbar zu machen, müssen Sie bestimmte Werte in ein des -Bausteins schreiben.
4. Die Positionsangabe für ein Sprite bestimmt genau genommen nur die Position der Ecke des Sprite-Musters
5. Durch eine Veränderung des Positionswertes bewegt sich ein Sprite auf oder ab.
6. Um die horizontale Position eines Sprites festzulegen, müssen Sie Positionsregister benutzen, da es horizontale Positionen gibt.
7. Durch die folgende Basic-Anweisung erhält die Variable TV den Wert :
10 LET TV = (17 < 5)
8. Modifizieren Sie die Zeile 1230 des Programms »Sprite-Entwurf« derart, daß das Sprite in gelber Farbe dargestellt wird.
1230
9. Ein Sprite kann in und in Richtung oder in beiden Richtungen vergrößert werden.

Programmierübungen

Alle Aufgaben können durch Erweiterung oder Änderung des Programms »Ein einfaches Sprite« (Listing 1) gelöst werden, aber auch durch Neuprogrammierung. Selbstverständlich stellt auch jedes andere lauffähige Programm, daß den gleichen Zweck erfüllt, eine Lösung dar.

1. Veranlassen Sie das Sprite, sich auf einer rechteckigen Bahn zu bewegen.
2. Verändern Sie das Programm so, das sich hin und wieder die Farbe des Sprites ändert.
3. Schreiben Sie eine Programmschleife, die das Sprite nacheinander in vier Größen darstellt: normal, verbreitert, verlängert und in beide Richtungen vergrößert.

Antworten zum Selbsttest

Hier sind die wohl zutreffendsten Antworten. Aber sicherlich sind auch andere sinnvolle Antworten möglich.

1. Bildpunkte
2. acht
3. Register, VIC.II
4. linken oberen
5. vertikalen
6. zwei, 512
7. null
8. 1230 POKE VIC+39,7 : REM GELBE FARBE
9. horizontaler, vertikaler (vertikaler, horizontaler)

Lösungsvorschläge für die Programmierübungen

Die folgenden drei Lösungsvorschläge basieren auf dem Programm »Ein einfaches Sprite« aus Listing 1. Dargestellt sind die Zeilen, die geändert oder hinzugefügt werden müssen, um die Aufgabe zu erfüllen.

1. Laden Sie das Programm »Ein einfaches Sprite« und geben Sie die folgenden Zeilen ein (Programm »Rechteckbewegung«):

```
1000 REM *** RECHTECKBEWEGUNG ***
1210 POKE VIC,82 :REM HORIZONT. POS.
1220 POKE VIC+1,100 :REM VERTIKALE POS.
1243 REM ** BEWEGUNG NACH RECHTS, UNTEN,
1244 REM LINKS, DANN AUSGANGSPUNKT
1246 REM (HORIZONTALE DREIERSPRUNGE
1247 REM ZUR ANPASS. AN VERT.GESCH.)
1249 FOR HP = 84 TO 261 STEP 3 :REM -> R
1250 : SF = (HP > R 255)
1251 : POKE VIC, HP + (SF * 256)
1252 : POKE VIC+16, SF * (-1)
1253 NEXT HP
1255 FOR VP = 101 TO 179 :REM RUNTER
1256 : POKE VIC+1, VP
1257 NEXT VP
1259 FOR HP = 258 TO 87 STEP -3 :REM <E-
1260 : SF = (HP > R 255)
1261 : POKE VIC, HP + (SF * 256)
1262 : POKE VIC+16, SF * (-1)
1263 NEXT HP
1265 FOR VP = 178 TO 100 STEP -1 : REM
1266 : POKE VIC+1, VP
1267 NEXT VP
1300 IF KP$ = "" THEN 1249
```

2. Laden Sie das Programm »Ein einfaches Sprite« und geben Sie die folgenden Zeilen ein (Programm »Farbwechsel«):

```
1000 REM *** FARBEWECHSEL ***
1071 REM ** ANFANGSFARBE FESTLEGEN
```

```
1073 OC = 13 :REM START MIT GRUEN
1230 POKE VIC+39,OC :REM ANFANGSFARBE
1252 REM ** FARBEN WECHSELN
1254 FOR DELAY = 1 TO 500 : NEXT
1255 NC = 0 OC + 1 :REM FARBE NEU
1256 IF NC=16 THEN NC=0 :REM FARBE 1-15
1257 POKE VIC+39,NC :REM ALTE FARBE
1258 OC = NC :REM NEUE FARBE
1300 IF KP$ = "" THEN 1254
```

3. Laden Sie das Programm »Ein einfaches Sprite« und geben Sie die folgenden Zeilen ein (Programm »Größenwechsel«):

```
1000 REM *** GROESSENWECHSEL ***
1252 REM ** HORIZONTALE VERGROESSERUNG
1254 FOR PAUSE = 1 TO 400 : NEXT
1255 POKE VIC+29,1
1258 REM ** HORIZONTAL VERKLEINERN UND
1259 REM VERTIKAL VERGROESSERN
1261 FOR PAUSE = 1 TO 400 : NEXT
1262 POKE VIC+29,0
1263 POKE VIC+23,1
1266 REM ** HORIZONTALE VERGROESSERUNG
1268 FOR PAUSE = 1 TO 400 : NEXT
1269 POKE VIC+29,1
1272 REM ** HORIZONTAL VERKLEINERN UND
1273 REM VERTIKAL VERKLEINERN
1275 FOR PAUSE = 1 TO 400 : NEXT
1276 POKE VIC+29,0
1277 POKE VIC+23,0
1280 REM ** PROGR.ENDE MIT TASTENDRUCK
1282 IF KP$ = "" THEN 1254
```

Kapitel 2: Mehrere Sprites

In diesem Kapitel lernen Sie, mehrere Sprites gleichzeitig auf dem Bildschirm darzustellen, aus den Daten eines Sprites mehrere identische oder unterschiedlich große Sprites zu erzeugen und zwei Sprites gleichzeitig über den Bildschirm gleiten zu lassen.

Sprite-Kopien

Der C 64 erlaubt die Darstellung mehrerer Sprites, auch wenn nur ein Satz von Sprite-Codes vorhanden ist. Wenn Sie ein solches Sprite in gleicher Größe an unterschiedlichen Bildschirmpositionen abbilden, sehen Sie vier Kopien desselben Sprites.

Listing 5 zeigt ein entsprechendes Programm, »Einfache Kopien«, das vier Kopien des in Bild 11 dargestellten Sprites erzeugt.

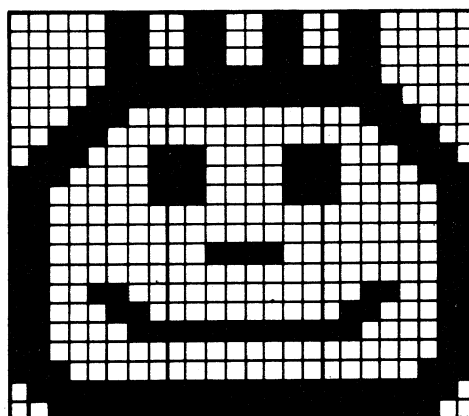


Bild 11.
Ein einfaches
Sprite für den
Kopiervorgang

Das Programm entspricht in seinem Aufbau dem Programm »Ein einfaches Sprite« aus Kapitel 1, mit dem Unterschied, daß jetzt Datenanzeiger, Positionsangaben und Farben für vier Sprites vorhanden sind. Geben Sie das Programm ein, speichern und starten Sie es.

In Kapitel 1 haben Sie gelernt, daß Adresse 2040 dazu dient, dem Rechner mitzuteilen, an welcher Speicherstelle sich die Pixelcodes für Sprite 0 befinden. Die entsprechenden Zeiger für die Sprites 1-7 stehen in den Adressen 2041 bis 2047 (Tabelle 2).

Adresse	2040	2041	2042	2043	2044	2045	2046	2047
Zeigt auf Pixel-Daten für Sprite- Nummer	0	1	2	3	4	5	6	7

Tabelle 2. Die Speicherzellen der Sprite-Datenzeiger

Betrachten wir die wichtigen Abschnitte des Programms (Listing 8) einmal genauer. Die Zeilen 1000-1310 kommen Ihnen sicherlich bekannt vor. Die Rückmeldung des Programms wird ausgegeben, die Bildpunktdaten werden in die Speicherzellen 896-958 gelesen, der Bildschirm wird gelöscht, und die Variable VIC erhält die Anfangsadresse des VIC-II-Bausteins.

Der erste Unterschied zum alten Programm liegt in den Zeilen 1330 bis 1360:

```
1330 POKE 2040,14 :REM DATENZEIGER 0
1340 POKE 2041,14 :REM DATENZEIGER 1
1350 POKE 2042,14 :REM DATENZEIGER 2
1360 POKE 2043,14 :REM DATENZEIGER 3
```

Statt eines Sprites sind vier Sprites darzustellen, wobei der Rechner jedesmal dieselben 63 Werte aus den Speicherzellen ab Adresse $(14 \cdot 64) = 896$ lesen soll. In den Zeilen 1380-1460 wird jedem Sprite eine horizontale und eine vertikale Bildschirmposition zugeordnet:

```
1380 POKE VIC,98 :REM HORZNTL.POS. 0
1390 POKE VIC+2,246 :REM HORZNTL.POS. 1
1400 POKE VIC+4,98 :REM HORZNTL.POS. 1
1410 POKE VIC+6,246 :REM HORZNTL.POS. 3
1420 :
1430 POKE VIC+1,95 :REM VERTKAL.POS. 0
1440 POKE VIC+3,95 :REM VERTKAL.POS. 1
1450 POKE VIC+5,184 :REM VERTKAL.POS. 2
1460 POKE VIC+7,184 :REM VERTKAL.POS. 3
```

Die Adresse VIC (53248) ist das Register für die horizontale Position von Sprite 0, und VIC+1 (53249) das Register für die vertikale Position dieses Sprites. Entsprechend sind die anderen 14 Register den Sprites 1 bis 7 zugeordnet. VIC+2 (53250) ist das horizontale und VIC+3 (53251) das vertikale Register für Sprite 1 und am Ende steht VIC+15 (53263), das vertikale Register für Sprite 7.

Als aufmerksamer Leser werden Sie jetzt fragen: Wo bleibt das zweite horizontale Register für jedes Sprite? Wie Sie aus Kapitel 1 wissen, können diese Register nur den Wert 0 oder 1 enthalten. Alle acht finden daher in einer Speicherstelle Platz, in der Adresse VIC+16 (53264). Sie werden diese Register später genauer kennenlernen.

Farben zuordnen und Sprites anzeigen

Die Zuordnung der Farben erfolgt in den Zeilen 1480-1510:

```
1000 rem *** einfache kopien *** <209>
1010 : <224>
1020 : <234>
1030 rem ** ausgabe programmierung <013>
1040 : <000>
1050 print "(clr,8down)bitte warten" <034>
1060 : <020>
1070 : <030>
1080 rem ** sprite-daten laden <212>
1090 : <050>
1100 for n = 896 to 958 <018>
1110 : read spdta <061>
1120 : poke n, spdta <098>
1130 next n <238>
1140 : <100>
1150 data 6,102,96,6,102,96 <148>
1160 data 6,102,96,7,255,224 <122>
1170 data 15,255,240,28,0,56 <157>
1180 data 56,0,28,113,195,142 <193>
1190 data 225,195,135,193,195,131 <090>
1200 data 192,0,3,192,0,3 <164>
1210 data 192,60,3,192,0,3 <061>
1220 data 204,0,51,198,0,99 <101>
1230 data 195,255,195,192,0,3 <116>
1240 data 224,0,7,127,255,254 <004>
1250 data 63,255,252 <168>
1260 : <220>
1270 : <230>
1280 rem ** sprite-register vorbereiten <000>
1290 : <252>
1300 print "(clr)" :rem schirm loesch <079>
1310 vic = 53248 :rem grafikbaustein <125>
1320 : <026>
1330 poke 2040,14 :rem datenzeiger 0 <031>
1340 poke 2041,14 :rem datenzeiger 1 <233>
1350 poke 2042,14 :rem datenzeiger 2 <180>
1360 poke 2043,14 :rem datenzeiger 3 <126>
1370 : <076>
1380 poke vic,98 :rem horzntl.pos. 0 <111>
1390 poke vic+2,246 :rem horzntl.pos. 1 <061>
1400 poke vic+4,98 :rem horzntl.pos. 2 <065>
1410 poke vic+6,246 :rem horzntl.pos. 3 <091>
1420 : <126>
1430 poke vic+1,95 :rem vertkal.pos. 0 <180>
1440 poke vic+3,95 :rem vertkal.pos. 1 <193>
1450 poke vic+5,184 :rem vertkal.pos. 2 <202>
1460 poke vic+7,184 :rem vertkal.pos. 3 <217>
1470 : <176>
1480 poke vic+39,1 :rem 0 ist weiss <064>
1490 poke vic+40,3 :rem 1 ist hellblau <089>
1500 poke vic+41,5 :rem 2 ist gruen <015>
1510 poke vic+42,7 :rem 3 ist gelb <205>
1520 : <226>
1530 poke vic+21,15 :rem sprites 0-3 an <180>
1540 : <248>
1550 : <002>
1560 rem ** prog.ende mit tastendruck <071>
1570 : <022>
1580 get kp$ <161>
1590 if kp$ = "" then 1580 <119>
1600 : <052>
1620 rem ** ruecksetzen des registers <207>
1630 : <082>
1640 poke vic+21,0 <140>
1650 : <102>
1660 end <138>
```

Listing 5. Ausdruck des Programms »Einfache Kopien«

```
1480 POKE VIC+39,1 :REM 0 IST WEISS
1490 POKE VIC+40,3 :REM 1 IST HELLBLAU
1500 POKE VIC+41,5 :REM 2 IST GRUEN
1510 POKE VIC+42,7 :REM 3 IST GELB
```

Wie Sie sicherlich vermutet haben, enthalten die acht aufeinanderfolgenden Register VIC+39 (53287) bis VIC+46 (53294) die Farbcodes für die einzelnen Sprites. Zeile 1530 sorgt schließlich für die Ausgabe der Sprites 0, 1, 2 und 3 auf dem Bildschirm:

```
1530 POKE VIC+21,15 :REM SPRITES 0-3 AN
```

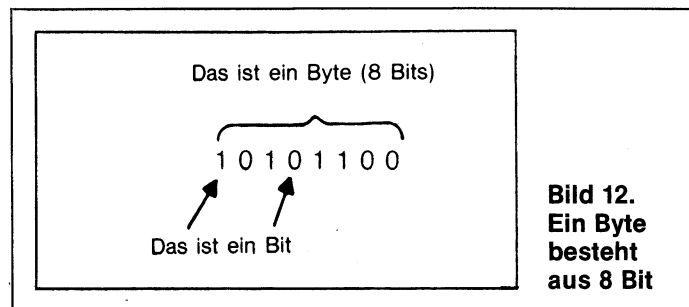
Welcher Zusammenhang besteht aber zwischen der Zahl 15 und den Nummern 0, 1, 2 und 3 der Sprites? Zur Beantwortung dieser Frage ist ein Ausflug in die Welt der Bits und Bytes erforderlich.

Bits und Bytes

Vielleicht haben Sie sich schon bei der Einteilung der Bildpunktzeilen eines Sprites in Gruppen zu je acht Bildpunkten gefragt, warum es gerade acht Bildpunkte sein müssen und nicht neun, zehn oder 24.

Der Grund liegt in der Art und Weise, in der Computer wie der C 64 Zahlen verarbeiten. Der Rechner kennt eigentlich nur die beiden Werte 0 und 1. Ein Zahlensystem, in dem alle Zahlen nur durch diese beiden Werte ausgedrückt werden, nennt man Binärsystem.

Die beiden binären Zeichen 0 und 1 heißen »Bits«. Eine Gruppe von acht Bits bezeichnet man als ein »Byte«. Jede der Speicherzellen des C 64, auch die Register, können jeweils ein Byte aufnehmen. Die Bilder 12 und 13 zeigen die Darstellung einiger Bytes. Mit 8 Bit lassen sich die Zahlen von 0 bis 255 darstellen.



Viele der VIC-II-Register ermöglichen die gleichzeitige Steuerung aller acht Sprites, da jeweils eines der 8 Bit pro Register einem Sprite zugeordnet ist und somit eine Art Miniaturregister darstellt.

So ist zum Beispiel das Register VIC+21 (53269) das Zentralregister zur Steuerung der Sprite-Anzeige. Jedes Bit dieses Registers ist einem Sprite zugeordnet und bestimmt, ob das Sprite sichtbar ist oder nicht.

Zahl →	255	240	128	127	60	15	1	0
Byte-Schreibweise →	11111111	11110000	10000000	01111111	00111100	00001111	00000001	00000000

Bild 13. In einem Byte können die Dezimalwerte (Basis 10) von 0 bis 255 in binärer Form (8 Bit) dargestellt werden

Die 8 Bit eines Bytes werden von rechts nach links durchnumeriert, erhalten also die Nummern 0 bis 7. In Register VIC+21 steuert Bit 0 das Sprite 0, Bit 1 das Sprite 1 und so weiter. Um ein bestimmtes Sprite sichtbar werden zu lassen, genügt es, das entsprechende Bit auf den Wert 1 zu setzen. Um es wieder verschwinden zu lassen, setzen Sie das Bit wieder auf 0.

Um also die vier Sprites 0 bis 3 anzuzeigen, muß in das Anzeige-Zentralregister VIC+21 eine Zahl geschrieben werden, deren Binärwert viermal die 1 (für die Bits 0, 1, 2 und 3) und viermal die 0 (für die Bits 4 bis 7) enthält. Sie können dazu ebenfalls die Tabelle 1 benutzen.

Bild 14 zeigt ein Byte mit den nummerierten Bits, ihren Codes sowie den beiden möglichen Bitwerten. Um den richtigen Zahlenwert zur Steuerung der Sprites zu erhalten, tragen Sie zunächst für jedes Sprite, das sichtbar werden soll, eine 1 an

der entsprechenden Bitposition ein und für jedes Byte, das unsichtbar bleiben soll, eine 0. Die Addition der Codes aller Bits, deren Wert 1 ist, ergibt dann die gesuchte Zahl, deren binäre Darstellung die Reihe der eingetragenen 0/1-Werte ist.

Einige Beispiele dazu finden Sie in Bild 15. Die Tabelle enthält auch den im Programm »Einfache Kopien« benutzten Wert. Es sollen dort die Sprites 0-3 sichtbar gemacht werden. Dazu müssen Sie die Bits 0 bis 3 auf 1 setzen. Die Addition der entsprechenden Bitcodes ergibt $(8 + 4 + 2 + 1) = 15$. Damit ist das Geheimnis der Zahl 15 in Zeile 1530 gelüftet.

Bit-Code →	128	64	32	16	8	4	2	1
Ein beliebiges Byte →	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0	1 or 0
Bit-Nummer →	7	6	5	4	3	2	1	0

Bild 14. Ein Byte mit seinen nummerierten 8 Bit, den Bit-Codes und den möglichen Bitwerten

Sichtbare Sprites	Unsichtbare Sprites	Registerbyte (Jedes Bit steuert das Sprite mit der gleichen Nummer)	Wert für den POKE-Befehl
-	0-7	Bit-Code: 128 64 32 16 8 4 2 1 Bit-Nummer: 7 6 5 4 3 2 1 0	0
0	1-7	Bc: 128 64 32 16 8 4 2 1 Bn: 0 0 0 0 0 0 0 1	1 = 1
0,1	2-7	Bc: 128 64 32 16 8 4 2 1 Bn: 0 0 0 0 0 0 1 1	1+2 = 3
0-3	4-7	Bc: 128 64 32 16 8 4 2 1 Bn: 0 0 0 0 1 1 1 1	1+2+4+8 = 15
0,2, 4,6	1,3, 5,7	Bc: 128 64 32 16 8 4 2 1 Bn: 0 1 0 1 0 1 0 1	1+4+16+64 = 85
0-7	-	Bc: 128 64 32 16 8 4 2 1 Bn: 1 1 1 1 1 1 1 1	1+2+4+8+16+32+64+128 = 255

Bild 15. Beispiel zur Steuerung der Sprites

Die restlichen Zeilen des Programms »Einfache Kopien« sind Ihnen bereits bekannt. In den Zeilen 1580-1590 wartet der Rechner auf einen Tastendruck. Sobald eine Taste betätigt wird, setzt Zeile 1640 die Sprite-Register zurück. Dabei zeigt sich eine kleine Besonderheit: nicht jedes Register muß zurückgesetzt werden.

Welche Register sollten Sie wieder auf 0 setzen? Zunächst das Zentralregister VIC+21, falls alle Sprites wieder verschwinden sollen. Wenn Sie ein Sprite verbreitert oder verlängert haben, ist es angebracht, auch die entsprechenden Register VIC+23 und VIC+29 wieder auf 0 zu setzen. Dadurch vermeiden Sie, daß ein Sprite ungewollt in vergrößerter Form dargestellt wird.

Den durch das vorangegangene Programm erzeugten vier Sprites lag zwar der gleiche Satz von Zahlencodes zugrunde, sie wurden jedoch in unterschiedlichen Farben dargestellt.

Weitere Unterschiede lassen sich durch Verlängern oder Verbreitern erzeugen.

Modifizierte Sprite-Kopien

Wie Sie aus Kapitel 1 wissen, steuert das Register VIC+29 die Verbreiterung der Sprites. Jedem Bit des Registers ist ein Sprite zugeordnet. Um zum Beispiel die Sprites 2 und 3 zu verbreitern, müssen Sie Bit 2 und 3 des Registers auf 1 setzen. Mit Hilfe von Bild 14 finden Sie für die in das Register einzugebende Zahl den Wert $(8 + 4) = 12$. Dieser Wert ist in Tabelle 3 dargestellt.

Bit-Code →	128	64	32	16	8	4	2	1
Bit →	0	0	0	0	1	1	0	0
Bit-Nummer →	7	6	5	4	3	2	1	0

$8 + 4 = 12$

Tabelle 3. Die Eingabe des Wertes 12 in das horizontale Vergrößerungsregister setzt Bit 2 und 3 auf 1 und bewirkt die horizontale Vergrößerung der Sprites 2 und 3

Das Register VIC+23 steuert in gleicher Weise die Verlängerung von Sprites. Um beispielsweise Sprite 1 und 3 zu verlängern, müssen Sie Bit 1 und 3 auf 1 setzen. Der entsprechende Zahlenwert für das Register ist $(8 + 2) = 10$. Er ist in Tabelle 4 dargestellt.

Bit-Code →	128	64	32	16	8	4	2	1
Bit →	0	0	0	0	1	0	1	0
Bit-Nummer →	7	6	5	4	3	2	1	0

$8 + 2 = 10$

Tabelle 4. Die Eingabe des Wertes 10 in das vertikale Vergrößerungsregister setzt Bit 1 und 3 auf 1 und bewirkt die vertikale Vergrößerung der Sprites 1 und 3

Sie besitzen nun das nötige Rüstzeug, um das Programm »Einfache Kopien« zu modifizieren. Laden Sie es und fügen Sie die Zeilen aus Listing 9 hinzu. Damit erhalten Sie das Programm »Modifizierte Kopien«. Speichern und starten Sie das neue Programm.

Der Bildschirm zeigt nun vier Sprites, die alle aus derselben Datengruppe erzeugt werden und die sich doch alle voneinander unterscheiden. Die neuen Zeilen 1400, 1410, 1440 und 1460 verteilen die Sprites auf dem Bildschirm, und die beiden Zeilen 1522 und 1524 sorgen für eine Vergrößerung entsprechend den beiden genannten Beispielen:

```

1000 rem*** unterschiedliche kopien ***      <108>
1400 poke vic+4,86 :rem horzntl.pos. 2      <055>
1410 poke vic+6,243 :rem horzntl.pos. 3      <067>
1440 poke vic+3,85 :rem vertkal.pos. 1        <191>
1460 poke vic+7,174 :rem vertkal.pos. 3       <213>
1522 poke vic+23,10 :rem 1 & 3 schmal        <197>
1524 poke vic+29,12 :rem 2 & 3 breit          <046>
1526 :                                         <232>
1642 poke vic+23,0                             <144>
1644 poke vic+29,0                             <152>

```

Listing 6. Änderungen und Erweiterungen zur Umwandlung des Programms »Einfache Kopien« in das Programm »Unterschiedliche Kopien«

```

1522 POKE VIC+23,10 :REM 1 & 3 SCHMAL
1524 POKE VIC+29,12 :REM 2 & 3 BREIT

```

Sprite 0 bleibt unverändert. Sprite 1 wird verlängert und Sprite 2 verbreitert. Sprite 3 wird sowohl verlängert als auch verbreitert. Ein Tastendruck beendet das Programm und setzt das Vergrößerungsregister auf 0 zurück.

In den meisten Fällen werden Sie Sprites unterschiedlicher Struktur benötigen. Dazu müssen Sie für jedes Sprite eine eigene Datengruppe (Bildpunktcodes) laden, und es stellt sich die Frage, wo die jeweils 63 Zahlen am besten unterzubringen sind.

Wenn Sie nur drei oder weniger Sprites darstellen wollen, stehen folgende Speicherbereiche zur Verfügung: 832-894, 896-958, 960-1022. Dieser gesamte Bereich ist der Puffer für den Kassettenrekorder und wird während des Programmlaufes nicht benutzt. Falls Sie den Kassettenpuffer benutzen, müssen die Datenzeiger der entsprechenden Sprites (2040-2047) die Anfangsadresse des jeweiligen Datenblocks, geteilt durch 64, enthalten. Für den Kassettenpuffer sind dies die Werte 13, 14 und 15.

Für die Daten von mehr als drei Sprites empfiehlt sich der Speicherbereich ab Adresse 12288. Tabelle 5 zeigt die jeweiligen Anfangsadressen pro Block sowie die zugehörigen Werte für die Datenzeiger. Dem fortgeschrittenen Systemprogrammierer des C 64 stehen zwar noch weitere Speicherbereiche zur Verfügung, deren Manipulation ist jedoch nicht Thema dieses Buches.

63-Byte Speicherbereich →	12288 — 12350	12352 — 12414	12416 — 12478	12480 — 12542	12544 — 12606	12608 — 12670	12672 — 12734	12736 — 12798
Datenzeiger auf →	192	193	194	195	196	197	198	199

Tabelle 5. Die für die Speicherung von Sprite-Daten geeigneten Bereiche und die entsprechenden Werte der Datenzeiger

Zwei unterschiedliche Sprites

Worin wird sich ein Programm, das zwei unterschiedliche Sprites auf dem Bildschirm darstellt, von dem Programm »Sprite-Entwurf« aus Kapitel 1 unterscheiden?

Erstens muß es statt einem zwei Datenblöcke laden, zweitens die Datenzeiger in Adresse 2040 und 2041 auf die Speicherbereiche mit den geladenen Daten richten, und drittens muß es die entsprechenden Werte für die Position, Größe und Farbe jedes Sprites in die VIC-II-Register schreiben. Schließlich sind noch beide Sprites sichtbar zu machen.

Bild 16 zeigt zwei neue Sprite-Entwürfe. Listing 7 enthält das Programm »Ein Sprite-Paar«, das die beiden Figuren auf dem Bildschirm darstellt. Geben Sie das Programm ein und speichern Sie es. Anschließend können Sie das Programm starten und Ihren Vorstellungen entsprechend modifizieren.

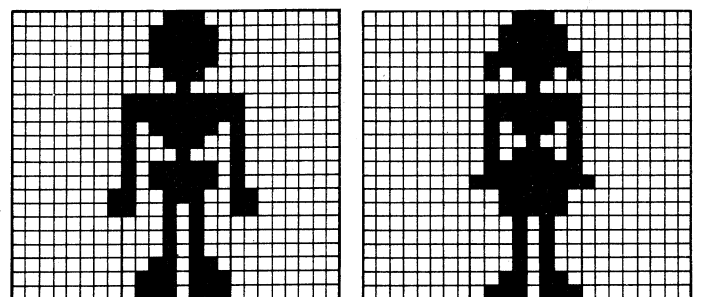


Bild 16. Die zwei neuen Sprites für »Ein Sprite-Paar«

```

1000 rem *** ein sprite-paar ***
1010 :
1020 :
1030 rem ** ausgabe programmmeldung
1040 :
1050 print "{clr,8down}bitte warten";
1060 :
1070 :
1080 rem ** sprite-daten laden
1090 :
1100 for n = 896 to 958 :rem 1.sprite
1105 : read spdta
1110 : poke n, spdta
1130 next n
1140 print". "; :rem pktausgb
1150 :
1160 for n = 960 to 1022 :rem 2.sprite
1165 : read spdta
1170 : poke n, spdta 0
1170 next n
1200 print". "; :rem pktausgb
1210 :
1220 data 0, 28, 0, 0, 62, 0
1230 data 0, 62, 0, 0, 62, 0
1240 data 0, 28, 0, 0, 8, 0
1250 data 0,255,128, 0,255,128
1260 data 0,190,128, 0,156,128
1270 data 0,136,128, 0,190,128
1280 data 0,190,128, 1,156,192
1290 data 1,148,192, 0, 20, 0
1300 data 0, 20, 0, 0, 20, 0
1310 data 0, 54, 0, 0,119, 0
1320 data 0,119, 0
1330 :
1340 data 0, 28, 0, 0, 62, 0
1350 data 0, 62, 0, 0,127, 0
1360 data 0, 93, 0, 0, 8, 0
1370 data 0,127, 0, 0,127, 0
1380 data 0, 93, 0, 0, 73, 0
1390 data 0, 93, 0, 0,127, 0
1400 data 0,255,128, 0, 62, 0
1410 data 0, 62, 0, 0, 20, 0
1420 data 0, 20, 0, 0, 20, 0
1430 data 0, 20, 0, 0, 54, 0
1440 data 0,119, 0
1450 :
1460 :
1470 rem ** sprite-register vorbereiten
1480 :
1490 print "{clr}" :rem schirm loesch
.
1500 vic = 53248 :rem grafikbaustein
1510 :
1520 poke 2040,14 :rem datenzeiger 0
1530 poke 2041,15 :rem datenzeiger 1
1540 :
1550 poke vic,124 :rem horzntl.pos. 0
1560 poke vic+2,173 :rem horzntl.pos. 1
1570 poke vic+1,150 :rem vertkal.pos. 0
1580 poke vic+3,150 :rem vertkal.pos. 1
1590 :
1600 poke vic+39,3 :rem 0 ist hellblau
1610 poke vic+40,7 :rem 1 ist gelb
1620 :
1630 poke vic+23,3 :rem beide sprites
1640 poke vic+29,3 :rem dopp. groesse
1650 :
1660 poke vic+21,3 :rem beide sichtb.
1670 :
1680 :
1690 rem ** prog.ende mit tastendruck
1700 :
1710 get kp$
1720 if kp$ = "" then 1710
1730 :
1740 :
1750 rem ** ruecksetzen des registers
1760 :
1770 poke vic+21,0 :rem sprites aus
1780 poke vic+29,0 :rem und wieder
1790 poke vic+23,0 :rem normalgroesse
1800 :
1810 end

```

Listing 7. Ausdruck des Programms »Ein Sprite-Paar«

Das Programm enthält keine neuen Anwendungen. Zeile 1050 löscht den Bildschirm und gibt die Programmmeldung aus. Zwei Schleifen laden dann die jeweils 63 Daten der beiden Sprites. Der erste Datenblock wird in den Speicherbereich 896-958 geschrieben. Das Programm signalisiert den Abschluß dieses Ladevorgangs, indem es in Zeile 1140 einen Punkt hinter die Programmmeldung setzt. Ein weiterer Punkt wird in Zeile 1200 ausgegeben, nachdem der zweite Datenblock in den Bereich 960-1022 geschrieben worden ist. Die Daten beider Sprites sind übrigens mit Hilfe des Entwurfsblattes aus Bild 6 berechnet worden.

Das Programm setzt dann die Datenzeiger und VIC-II-Register. Um die Einzelheiten beider Sprites besser erkennbar zu machen, werden beide in doppelter Größe dargestellt. Zeile 1660 setzt die Bits 0 und 1 des Registers VIC+21 auf 1 und macht die beiden Sprites 0 und 1 damit sichtbar. Sollten Sie sich nicht mehr klar darüber sein, warum dazu der Wert 3 benutzt wird, so lesen Sie bitte noch einmal das Kapitel 1.

Die Zeilen 1700-1710 warten auf den üblichen Tastendruck, um das Programm zu beenden, und die Zeilen 1770-1790 setzen die Anzeige- und die Vergrößerungsregister auf den Wert 0 zurück.

Die gleichzeitige Bewegung mehrerer Sprites

Es gibt viele Methoden, um mehrere Sprites zu gleicher Zeit über den Bildschirm zu bewegen. Einige sind einfach zu programmieren, andere schwierig. Einige benötigen viel Speicherplatz, andere wenig. Einige ermöglichen nur geradlinige, andere beliebige Bewegungen. Einige erzeugen schnelle und zusammenhängende, andere nur langsame und ruckartige Bewegungen. Einige sind sehr leicht nachzuvollziehen, andere sind trickreich und schwer verständlich.

Das Beispiel in diesem Kapitel ist ein Mittelding. Es ist nicht zu schwierig, liefert aber trotzdem ein beachtliches Resultat. Um die beiden Sprites zu veranlassen, sich im Kreis herumzujagen, laden Sie bitte zuerst das Programm »Ein Sprite-Paar« und geben dann die in Listing 8 dargestellten Zeilen zusätzlich ein. Speichern und starten Sie dann das neue Programm.

Die beiden Sprites bewegen sich in diesem Programm auf einer quadratischen Bahn in der Bildschirmmitte. Die Seitenlänge dieses Quadrates beträgt 100 Bildpunkte. Seine Eckpunkte sind somit in jeder Richtung 50 Bildpunkte vom Mittelpunkt entfernt. Damit ein Sprite doppelter Größe in der Bildschirmmitte dargestellt werden kann, muß seine horizontale Position 160 und seine vertikale Position 129 betragen. Durch Addition beziehungsweise Subtraktion von 50 wird das Sprite also an einen der Eckpunkte des Bahnquadrates versetzt. Die Positionswerte für die vier Eckpunkte sind in Bild 17 dargestellt.

Sprite 0 beginnt seinen Weg in der linken oberen Ecke und Sprite 1 gegenüber in der rechten unteren Ecke. Sprite 0 bewegt sich nach unten, nach rechts, nach oben und dann nach links. Sprite 1 bewegt sich nach oben, nach links, nach unten und dann nach rechts. Betrachten Sie bitte Bild 18. Es zeigt vier unterschiedliche Bahnpositionen für die beiden Sprites. Bei Bild 1 handelt es sich um die Startposition. Die Pfeile geben die Bewegungsrichtung an. Beachten Sie, daß beide Sprites stets auf gegenüberliegenden Bahnen laufen, jedoch immer in entgegengesetzter Richtung. Diese Bewegungssymmetrie erleichtert die Programmierung erheblich.

Auch bei diesem Beispiel sollten Sie wieder eigene Experimente und Modifikationen am Programm vornehmen und deren Wirkung beobachten.

```

1000 rem *** ein sprite-jagd ***          <161>
1550 poke vic,110 :rem horzntl.pos. 0    <141>
1560 poke vic+2,210 :rem horzntl.pos. 1   <173>
1570 poke vic+1,79 :rem vertkal.pos. 0    <078>
1580 poke vic+3,179 :rem vertkal.pos. 1   <109>
1662 :                                   <114>
1664 :                                   <116>
1666 rem ** sprite-bewegung vorbereiten   <071>
1667 :                                   <119>
1668 d0 = 1 : d1 = -1                    <240>
1670 :                                   <122>
1672 :                                   <124>
1674 rem ** vertikale bewegung            <058>
1676 :                                   <128>
1678 for move = 1 to 100                  <052>
1680 :   poke vic+1, peek(vic+1) + d0     <205>
1682 :   poke vic+3, peek(vic+3) + d1     <227>
1684 :   get kp$                           <092>
1686 :   if kp$ = "" then 1690             <041>
1688 :   move=100 : keypress = -1          <009>
1690 next move                            <041>
1692 :                                   <144>
1694 :                                   <146>
1696 rem ** bei tastendruck abbrechen    <068>
1698 :                                   <150>
1700 if keypress then 1750                <199>
1702 :                                   <154>
1704 :                                   <156>
1706 rem ** horizontale bewegung          <097>
1708 :                                   <160>
1710 for move = 1 to 100                  <084>
1712 :   poke vic, peek(vic) + d0          <222>
1714 :   poke vic+2, peek(vic+2) + d1     <001>
1716 :   get kp$                           <124>
1718 :   if kp$ = "" then 1722             <248>
1720 :   move=100 : keypress = -1          <041>
1722 next move                            <073>
1724 :                                   <176>
1726 :                                   <178>
1728 rem ** bei tastendruck abbrechen    <100>
1730 :                                   <182>
1732 if keypress then 1750                <231>
1734 :                                   <186>
1736 :                                   <188>
1738 rem ** richtung umgekehrt und los    <031>
1740 :                                   <192>
1742 d0 = -d0 : d1 = -d1                  <222>
1744 goto 1670                            <224>
1746 :                                   <198>
1748 :                                   <200>

```

Listing 8. Änderungen und Ergänzungen zur Umwandlung des Programms »Ein Sprite-Paar« in »Sprite-Jagd«

Programmierung der Richtungen und Bewegungen

Die Ausgangsstellung der Sprites wird in Zeile 1550-1580 festgelegt:

```

1550 POKE VIC,110 :REM HORIZONTAL 0
1560 POKE VIC+2,210 :REM HORIZONTAL 1
1570 POKE VIC+1,79 :REM VERTIKAL 0
1580 POKE VIC+3,179 :REM VERTIKAL 1

```

Wie bereits beschrieben, startet Sprite 0 in der linken oberen und Sprite 1 in der rechten unteren Ecke.

Das Programm benutzt zur Steuerung der Bewegung zwei Variable, D0 für Sprite 0 und D1 für Sprite 1. Beide werden in Zeile 1668 mit dem Wert 1 initialisiert:

```
1668 D0 = 1: D1 = -1
```

Die Bewegung der Sprites ergibt sich durch Addition der beiden Variablen zu den entsprechenden Positionsregistern. Wenn Sie eine positive Zahl zu dem jeweiligen vertikalen Positionswert addieren, so erhöht sich der Registerwert und das Sprite bewegt sich auf dem Bildschirm nach unten. Bei

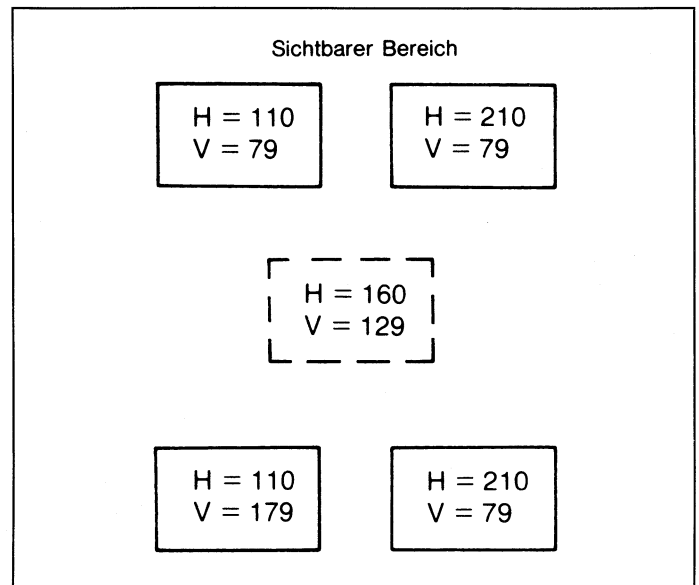


Bild 17. Eckpunkte der quadratischen Bahn der Sprites

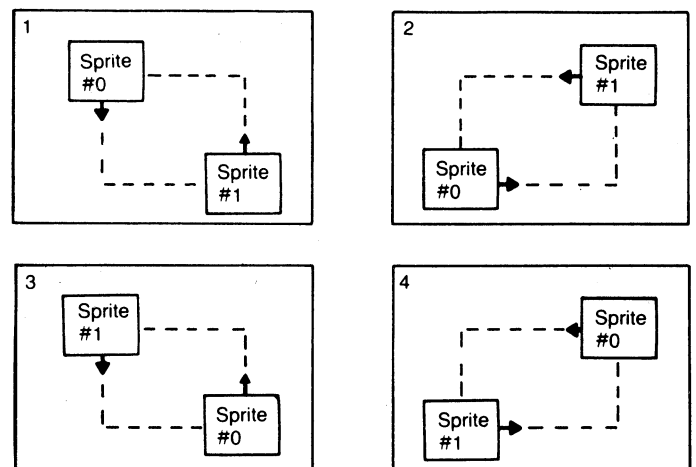


Bild 18. Die vier Eckpositionen der beiden Sprites auf ihrem Bahnquadrat (siehe Programm »Sprite-Jagd«)

der Addition negativer Werte ist es umgekehrt, der Registerwert verkleinert sich und das Sprite bewegt sich nach oben. Entsprechend bewirkt die Addition positiver Zahlen zum horizontalen Positionswert eine Bewegung nach rechts, die Addition negativer Werte eine Bewegung nach links.

Zeile 1678-1690 steuert die vertikalen Bewegungen beider Sprites:

```

1678 FOR MOVE = 1 TO 100
1680 :   POKE VIC+1, PEEK(VIC+1) + D0
1682 :   POKE VIC+3, PEEK(VIC+3) + D1
1684 :   GET KP$
1686 :   IF KP$ = "" THEN 1690
1688 :   MOVE = 100 : KEYPRESS = -1
1690 NEXT MOVE

```

Die Zeilen 1678 und 1690 erzeugen eine Schleife, die hundertmal durchlaufen wird. Hundertmal deshalb, weil jede Bahnseite aus 100 Bildpunkten besteht und das Sprite um jeweils einen Bildpunkt weiterbewegt werden soll. Bei jedem Durchlauf wird in Zeile 1680 der Wert 1 der Bewegungsvariablen für Sprite 0 zum vertikalen Positionswert dieses Sprites addiert. In gleicher Weise erfolgt in Zeile 1682 die Addition der Bewegungsvariablen von Sprite 1 zu dessen vertikalem Positionsregister.

Die Zeilen 1686 bis 1688 stellen eine Erweiterung gegenüber den bisherigen Bewegungsprogrammen dar. Die Tastatur wird nun nach jeder Bewegung (um einen Bildpunkt) abgefragt und nicht mehr erst nach Abarbeitung der gesamten Schleife. Die Abfrage erfolgt in Zeile 1686. Wurde keine Taste betätigt, wird Zeile 1688 übersprungen und das Programm fortgesetzt. Ist jedoch eine Taste gedrückt worden, so erhält die Schleifenvariable MOVE den Wert 100 und die Variable KEYPRESS den Wert -1. Die Erhöhung der Schleifenvariablen auf ihren Maximalwert (100) bewirkt, daß die Schleife nicht erneut durchlaufen wird. Dieses ist in Basic der beste Weg, um eine Schleife vorzeitig zu beenden. KEYPRESS wird auf -1 gesetzt, da -1 der Wahrheitswert für eine logisch wahre Bedingung ist. Wir haben dieses Thema in Kapitel 1 behandelt, bitte informieren Sie sich dort, wenn Ihnen etwas unklar ist.

Zeile 1700 steuert den weiteren Programmablauf:

```
1700 IF KEYPRESS THEN 1750
```

Enthält KEYPRESS den Wert 0 (Wahrheitswert für » keine Taste gedrückt«), so beginnt das Programm mit der Schleife für die horizontale Bewegung. Wurde dagegen eine Taste betätigt, so enthält KEYPRESS den Wert -1 und das Programm endet mit dem Programmteil zur Rücksetzung der Register ab Zeile 1750.

Die Abfrage eines Wahrheitswertes bezeichnet man übrigens nach dem englischen Mathematiker Boole als Booleschen Test und die logischen Werte WAHR und FALSCH als Boolesche Variable.

Die Zeilen 1710-1722 bilden die Schleife für die horizontale Bewegung:

```
1710 FOR MOVE = 1 TO 100
1712 : POKE VIC, PEEK (VIC) + D0
1714 : POKE VIC+2, PEEK (VIC+2) + D1
1716 : GET KP$
1718 : IF KP$ = "" THEN 1722
1720 : MOVE = 100 : KEYPRESS = -1
1722 NEXT MOVE
```

Diese Schleife entspricht fast vollständig der Schleife für die vertikale Bewegung. Der einzige Unterschied ist, daß die Bewegungsvariablen nicht zu den vertikalen, sondern zu den horizontalen Registern addiert werden.

Zeile 1732 prüft wieder, ob im vorangegangenen Durchgang eine Taste gedrückt worden ist:

```
1732 IF KEYPRESS THEN 1750
```

Wenn ja, endet das Programm mit der Rücksetzung der Register. Anderenfalls beginnt es mit der Änderung der Bewegungsrichtung.

Richtungsänderung beim Bahndurchlauf

Führen Sie sich noch einmal den Weg von Sprite 0 vor Augen. Es muß sich nach unten, nach rechts, nach oben und schließlich nach links bewegen. Anders ausgedrückt: vertikale Bewegung, horizontale Bewegung, vertikale Bewegung, horizontale Bewegung. Die Schleifen für die erste vertikale und horizontale Bewegung haben Sie bereits programmiert. Benötigen Sie nun noch zwei weitere Schleifen?

Nein, Sie können darauf verzichten. Es genügt, die Bewegungsrichtung von Sprite 0 umzukehren und dann die vorhandene horizontale und vertikale Schleife noch einmal zu durchlaufen. Der bisherige Wert der Bewegungsvariablen D0

war 1. Durch Multiplikation mit -1 wird er zu -1. Damit bewegt sich Sprite 0 in der vertikalen Schleife nach oben und in der horizontalen Schleife nach links. Ebenso läßt sich die Bewegungsrichtung von Sprite 1 ändern. Nach einer Multiplikation von D1 mit -1 bewegt es sich in der vertikalen Schleife nach unten und in der horizontalen Schleife nach rechts. Nach der Multiplikation beider Bewegungsvariablen mit -1 muß das Programm also nach Zeile 1678 zurückspringen und die Bewegungsschleifen noch einmal durchlaufen.

Die Zeilen 1742-1744 sorgen für die Bewegungsänderung:

```
1742 D0 = -D0 : D1 = -D1
1744 GOTO 1678
```

Sobald das Programm die Zeile 1742 erneut erreicht, erfolgt wieder ein Richtungswechsel, denn die beiden Richtungswerte werden erneut umgekehrt, das heißt, auf ihren vorangegangenen Wert zurückgesetzt. Da sich beide Sprites in diesem Moment wieder in ihren Ausgangspositionen befinden (Sprite 0 im linken oberen, Sprite 1 im rechten unteren Eckpunkt der Bahn), beginnt die Jagd erneut.

Sie können nun mit dem Programm experimentieren. Wie muß die Änderung für eine Dreiecksbahn aussehen? Lassen sich auch vier Sprites auf der quadratischen Bahn bewegen? Wodurch ergeben sich Spiralbahnen in den Bildmittelpunkt und dann wieder heraus? Bedenken Sie bei größeren Änderungen, daß es wichtig ist, zunächst ein Konzept zu entwickeln und sich dann erst an den Rechner zu setzen.

Zusammenfassung

In diesem Kapitel haben Sie sich mit einigen Techniken zur Steuerung mehrerer Sprites befaßt. Sie haben gelernt:

- unter Benutzung derselben 63 Sprite-Codes mehrere Sprite-Kopien auf dem Bildschirm darzustellen,
- wie mit Hilfe einzelner Bits in einigen VIC-II-Registern bestimmte Sprites gesteuert werden,
- wo die Datenblöcke für mehrere Sprites gespeichert werden können und wie die Datenzeiger in 2040-2047 zu setzen sind,
- auf welche Weise mehrere Sprites gleichzeitig zu bewegen sind,
- wie eine Programmschleife verlassen und der Programmablauf mit Hilfe von Booleschen Variablen gesteuert werden kann.

Im folgenden Kapitel lernen Sie weitere Tricks zur Programmierung von Sprites kennen. Vorher sollten Sie jedoch noch Ihr in diesem Kapitel erworbenes Wissen überprüfen.

Selbsttest

1. Speicherzelle 2045 enthält normalerweise den Datenzeiger für Sprite
2. Eine Gruppe von 8 Bit wird genannt.
3. Mit einem Byte können die Dezimalzahlen bis dargestellt werden.
4. Um die Sprites 2, 4 und 7 sichtbar zu machen, müssen Sie den Dezimalwert in das Register VIC+21 schreiben.
5. Um die Sprites 0, 3 und 4 zu verlängern, müssen Sie den Wert in Register VIC+23 schreiben.
6. Die Daten zur Darstellung von acht Sprites werden am besten ab Adresse gespeichert.

- Um ein in beide Richtungen vergrößertes Sprite in Höhe der Bildschirmmitte darzustellen, müssen Sie den Wert in das entsprechende vertikale Positionsregister schreiben.
- Wird der horizontale Positionswert eines Sprites vergrößert, so bewegt sich das Sprite nach
- Variable, deren Zahlenwert den logischen Wert WAHR oder FALSCH repräsentiert, heißen logische oder Variable.
- Um den Wert einer Variablen abwechselnd auf -1 und 1 zu setzen, muß diese jeweils mit der Zahl multipliziert werden.

Programmierungsübungen

- Modifizieren Sie das Programm »Einfache Kopien« derart, daß vier weitere Kopien erscheinen, und zwar in jeder Bildschirmecke eine.
- Ändern Sie das Programm »Sprite-Jagd«, so daß sich das Sprite-Paar im Uhrzeigersinn bewegt.
- Ändern Sie das Programm »Sprite-Jagd«, so daß zwei weibliche Sprites zwei männliche Sprites umherjagen.

Antworten zum Selbsttest

Die folgenden Antworten dürften wiederum die zutreffendsten sein. Möglicherweise finden Sie jedoch weitere.

- 5
- Byte
- 255
- 148
- 25
- 12288
- 129
- rechts
- Boolesche
- 1

Lösungsvorschläge für die Programmierungsübungen

Die Lösungen bestehen aus zusätzlichen oder geänderten Zeilen für die in diesem Kapitel vorgestellten Programme. Dabei handelt es sich nur um Vorschläge. Anderslautende Lösungen sind ebenfalls richtig, wenn sie den Anforderungen entsprechen.

- Laden Sie das Programm »Einfache Kopie« und geben Sie folgende Zeilen ein:

```
1000 REM *** ACHT KOPIEN ***
1362 POKE 2044,14 :REM DATENZEIGER 4
1364 POKE 2045,14 :REM DATENZEIGER 5
1366 POKE 2046,14 :REM DATENZEIGER 6
1368 POKE 2047,14 :REM DATENZEIGER 7
1412 POKE VIC+8,24 :REM HORZNTL.POS. 4
1414 POKE VIC+10,64 :REM HORZNTL.POS. 5
1416 POKE VIC+12,24 :REM HORZNTL.POS. 6
1418 POKE VIC+14,64 :REM HORZNTL.POS. 7
1419 POKE VIC+16,160:REM 5&7 MIT 2.HR
1462 POKE VIC+9,50 :REM VERTKAL.POS. 4
1464 POKE VIC+11,50 :REM VERTKAL.POS. 5
1466 POKE VIC+13,299:REM VERTKAL.POS. 6
1468 POKE VIC+15,229:REM VERTKAL.POS. 7
1512 POKE VIC+43,7 :REM 4 IST GELB
1514 POKE VIC+44,5 :REM 5 IST GRUEN
```

```
1516 POKE VIC+45,3 :REM 6 IST HELLBLAU
1518 POKE VIC+46,1 :REM 7 IST WEISS
1530 POKE VIC+21,255:REM SPRITES 0-7 AN
```

- Laden Sie das Programm »Sprite-Jagd« und geben Sie folgende Zeilen ein:

```
1000 REM *** IM UHRZEIGERSINN ***
1674 REM ** HORIZONTALE BEWEGUNG
1680 : POKE VIC, PEEK(VIC) + D0
1682 : POKE VIC+2, PEEK(VIC+2) + D1
1706 REM ** VERTIKALE BEWEGUNG
1712 : POKE VIC+1, PEEK(VIC+1) + D0
1714 : POKE VIC+3, PEEK(VIC+3) + D1
```

- Laden Sie das Programm »Sprite-Jagd« und geben Sie folgende Zeilen ein:

```
1000 REM *** PAARWEISE JAGD ***
1530 POKE 2041,14 :REM DATENZEIGER 1
1533 POKE 2042,15 :REM DATENZEIGER 2
1536 POKE 2043,15 :REM DATENZEIGER 3
1563 POKE VIC+4,110 :REM HORIZONTAL 2
1566 POKE VIC-6,210 :REM HORIZONTAL 3
1568 :
1583 POKE VIC+5,179 :REM VERTIKAL 2
1586 POKE VIC+7,79 :REM VERTIKAL 3
1613 POKE VIC+41,1 :REM 2 IST WEISS
1616 POKE VIC+42,5 :REM 3 IST GRUEN
1630 POKE VIC+23,15 :REM ALLE 4 SPRITES
1640 POKE VIC+29,15 :REM DOPPELT GROSS
1660 POKE VIC+21,15 :REM ALLE 4 SPR. AN
1674 REM ** BEWEGUNG AUF EINER SEITE
1681 : POKE VIC+4, PEEK(VIC+4) + D0
1683 : POKE VIC+6, PEEK(VIC+6) + D1
1706 REM ** BEWEGUNG AUF ANDERER SEITE
1713 : POKE VIC+5, PEEK(VIC+5) + D1
1715 : POKE VIC+7, PEEK(VIC+7) + D0
1753 : POKE VIC+7, PEEK(VIC+7) + D0
```

Kapitel 3: Weitere Sprite-Tricks

Interessiert es Sie, wie mehrfarbige Sprites erzeugt werden? In diesem Kapitel werden Sie Näheres darüber erfahren. Außerdem lernen Sie, Sprites übereinander hinweg zu bewegen. Schließlich werden Sie sogar, mit Hilfe mehrerer sich überlagernder Grundfiguren, eine kleine Zirkusnummer programmieren. Nebenbei gibt es weitere Experimente mit Bits, Bytes und Sprites.

Ein normales Sprite ist durch 63 Bytes definiert, die ab einer bestimmten Adresse im Speicher abgelegt sind. Jedes Byte besteht aus acht Bits, und jedes Bit steuert einen Bildpunkt (sichtbar oder unsichtbar). Ein Sprite besteht somit aus $(63 * 8) = 504$ Bildpunkten.

Hat das einem bestimmten Bildpunkt zugeordnete Bit den Wert 1, so wird der Bildpunkt in der im Farbregister des Sprites codierten Farbe ausgegeben. Ist der Wert 0, so wird der Bildpunkt in der Farbe des Bildschirms angezeigt, das heißt, er ist unsichtbar.

Im Normalfall gibt es für einen Bildpunkt also nur zwei Zustände: sichtbar oder unsichtbar. Der C64 bietet jedoch noch eine weitere Möglichkeit, den sogenannten Mehrfarbenmodus. In diesem Modus werden zwei Bits (ein Bitpaar) benutzt, um eine Farbe zu bestimmen.

Zwei Bits lassen sich in vier unterschiedlichen Bitmustern darstellen (Bild 19). Damit können vier Farben für die beiden dem Bitpaar zugeordneten Bildpunkte definiert werden.

Selbstverständlich sind beide Bildpunkte von gleicher Farbe. Am einfachsten ist es, die beiden Bildpunkte als einen großen Bildpunkt doppelter Breite zu betrachten. Bei der Benutzung dieser Möglichkeit bestimmt dann ein Byte die Farbe für vier Doppelbildpunkte, und eine Sprite-Zeile besteht statt aus 24 Bildpunkten nun aus 12 Doppelbildpunkten. Die horizontale Auflösung eines mehrfarbigen Sprites ist also nur halb so hoch wie die eines normalen Sprites.

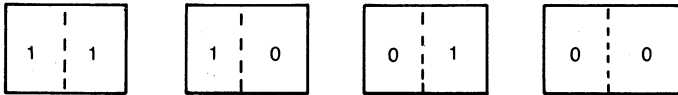


Bild 19. Mit zwei Bit können vier unterschiedliche Bitmuster dargestellt werden

Halbe Auflösung bedeutet, daß nicht mehr jeder einzelne Bildpunkt sichtbar oder unsichtbar gemacht werden kann, sondern nur noch Bildpunktpaare. Zur Wiederholung: Der Grund dafür ist, daß zwei Bits benötigt werden, um eine Farbe zu definieren, und ein Byte somit nur noch vier Doppelbildpunkte steuern kann (Bild 20). Eine Sprite-Reihe besteht jetzt also aus 12 Doppelbildpunkten. Die Gesamtgröße des Sprites wird dadurch nicht verändert. Ein mehrfarbiges Sprite besteht somit aus $(63 * 4)$ 252 Doppelbildpunkten und wird nach wie vor durch 63 Datenbytes definiert.

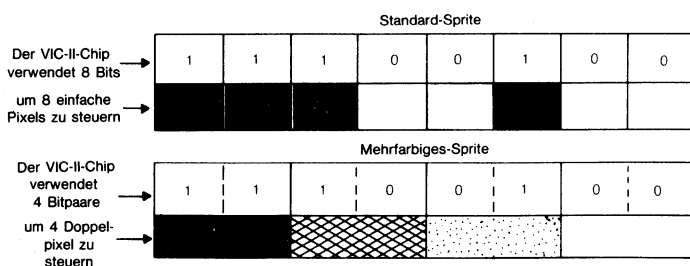


Bild 20. In einem mehrfarbigen Sprite wird jeweils ein Doppelpunkt durch ein Bitpaar gesteuert

Der Mehrfarbenmodus

Welche Farben stehen für mehrfarbige Sprites zur Verfügung? Das Bitpaar 00 definiert die Farbe des Bildschirms, die ja bekanntlich im Register VIC+33 (53281) codiert ist. Das Bitpaar 01 bedeutet, daß der Farbcode aus dem Mehrfarbenregister 0 mit der Adresse VIC+37 gelesen wird. Ein Bitpaar 10 veranlaßt den C64, das normale Farbregister des Sprites zu benutzen. Wie Sie sich erinnern, besitzt jedes Sprite sein eigenes Farbregister im Bereich VIC+39 bis VIC + 46. Das Bitpaar 11 schließlich verweist auf den Farbcode im Mehrfarbenregister 1 mit der Adresse VIC + 38. Alle Zuordnungen sind in Tabelle 6 noch einmal zusammengefaßt.

Auf welche Weise wird dem C64 nun mitgeteilt, daß er ein bestimmtes Sprite mehrfarbig darstellen soll? Das Steuerregister dafür ist VIC+28. Wie üblich, ist jedem Bit ein Sprite zugeordnet, Bit 0 steuert Sprite 1, und so weiter. Ist ein Bit dieses Registers auf 1 gesetzt, so wird das entsprechende

Bit-paar	Beschreibung	Adresse
0 0	Bildschirmfarbe	VC+33 (53281)
0 1	Sprite- Mehrfarbenregister 0	VC+37 (53285)
1 0	Sprite-Farbbregister	eines der Register VC+39 - VC+46 (53287 - 53294)
1 1	Sprite- Mehrfarbenregister 1	VC+38 (53286)

Tabelle 6. Das einem Doppelbildpunkt zugeordnete Bitpaar bestimmt das Farbcode-Register

Sprite mehrfarbig dargestellt. Ein Rücksetzen des Bits bewirkt wieder die normale Darstellung des Sprites.

Der Entwurf eines mehrfarbigen Sprites

Falls Sie nicht zu den wenigen Spezialisten gehören, die ein mehrfarbiges Sprite im Kopf entwerfen können, benutzen Sie bitte das Entwurfsblatt aus Bild 21. Es entspricht weitgehend dem Entwurfsblatt für einfarbige Sprites aus Kapitel 1. Wie dieses enthält es 21 Zeilen, die Bitwerte und drei Spalten für die Codezahlen. Allerdings sind nur 12 Spalten vorhanden, da in mehrfarbigen Sprites ja nur Doppelbildpunkte angesprochen werden können.

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

Hintergrundfarbe (transparent)

0	0
---	---

Mehrfarben-register 0

0	1
---	---

Spalte-Farbe

1	0
---	---

Mehrfarben-register 1

1	1
---	---

Bild 21. Ein Entwurfsblatt für mehrfarbige Sprites

Wie wird dieses Entwurfsblatt benutzt? Betrachten Sie dazu Bild 22, in dem ein ausgefülltes Entwurfsblatt dargestellt ist.

Spaltennummer	0	1	2	3	4	5	6	7	8	9	10	11	Zahlen-Codes				
Wert	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	
Zeile 0																	1 85 64
Zeile 1																	1 85 64
Zeile 2																	1 20 64
Zeile 3																	1 20 64
Zeile 4																	1 85 64
Zeile 5																	1 20 64
Zeile 6																	1 65 64
Zeile 7																	1 85 64
Zeile 8																	0 60 0
Zeile 9																	0 60 0
Zeile 10																	62 170 188
Zeile 11																	62 170 188
Zeile 12																	48 170 12
Zeile 13																	16 170 4
Zeile 14																	20 130 20
Zeile 15																	20 130 20
Zeile 16																	16 195 4
Zeile 17																	0 195 0
Zeile 18																	0 65 0
Zeile 19																	1 65 64
Zeile 20																	1 65 64

Hintergrundfarbe
(transparent)
0 0

Mehrfarben-
register 0
0 1

Sprite-
Farbe
1 1 0

Mehrfarben-
register
1 1 1

Bild 22. Ein ausgefülltes Entwurfsblatt

Denken Sie sich zunächst für jede Farbe, die Sie benutzen wollen, ein Muster aus, und tragen Sie dieses in die Farbkästchen am unteren Rand des Blattes ein. Sie können auch Farbstifte verwenden. Die Farbe des Bildschirmes wird am besten durch ein weißes Kästchen dargestellt.

Nun setzen Sie das Sprite aus Doppelkästchen zusammen, wobei ein Doppelkästchen einem Doppelbildpunkt entspricht. Liegt die Form des Sprites fest, füllen Sie die Doppelkästchen mit dem entsprechenden Farbmuster, am besten eine Farbe nach der anderen. Anschließend beginnen Sie dann mit der Berechnung der Zahlencodes.

Dazu addieren Sie die in der zweiten Kopfzeile des Entwurfsblattes dargestellten Bitcodes aller mit einem Farbmu-

ster ausgefüllten Einzelkästchen (!) jeder Gruppe (jeder Zahlencode gilt ja für eine Gruppe von 8 Bildpunkten bzw. 4 Doppelbildpunkten), wenn das entsprechende Bit in den Farbkästchen den Wert 1 hat. Diese Addition entspricht der Addition bei einfarbigen Sprites, nur werden diesmal Doppelkästchen betrachtet, deren Einzelkästchen bei der Berechnung unberücksichtigt bleiben, wenn über dem entsprechenden Einzelkästchen des Farbdoppelkästchens der Bitwert 0 steht.

Die errechneten Zahlencodes tragen Sie wie üblich in die am rechten Rand des Entwurfsblattes dafür vorgesehenen Spalten ein.

Fertigen Sie sich mehrere Kopien des Entwurfsblattes an, und zeichnen Sie eigene Entwürfe. Vergewissern Sie sich, daß Sie die Berechnung der Zahlencodes in Bild 22 verstanden haben, und berechnen Sie dann die Codes für Ihren eigenen Entwurf. Sie benötigen diese Daten im folgenden Kapitel.

Ein Programm zur Darstellung mehrfarbiger Sprites

Listing 9 zeigt das Programm »Vierfarben-Sprite«. Es erzeugt das in Bild 22 gezeichnete Sprite. Die Betätigung einer beliebigen Taste beendet das Programm.

Das Programm entspricht weitgehend den bisherigen Sprite-Programmen, mit Ausnahme der Zeilen 1400-1440:

```
1400 POKE VIC+28,1 :REM MEHRFARBMODUS
1410 POKE VIC+33,0 :REM HGRUND SCHWARZ
1420 POKE VIC+37,7 :REM MFREG 0 GELB
1430 POKE VIC+39,5 :REM SPRITE 0 GRUEN
1440 POKE VIC+38,6 :REM MFREG 1 BLAU
```

Zeile 1400 schreibt in das Auswahlregister den Wert 1, so daß Sprite 0 mehrfarbig dargestellt wird. Die Zeilen 1410 bis 1440 bestimmen dann die zu benutzenden Farben: Schwarz, definiert durch das Bitpaar 00; Gelb, definiert durch das Bitpaar 01; Grün, definiert durch das Bitpaar 10; und Blau, definiert durch das Bitpaar 11.

Am Ende des Programms wird das Mehrfarben-Sprite-Register MFSR zurückgesetzt:

```
1580 POKE VIC+28,0 :REM MEHRFARB. AUS
```

Die Benutzung eines normalen Sprites als mehrfarbiges Sprite ergibt eine sehr eigenartige Darstellung. Sie können sich davon überzeugen, indem Sie eines der vorangegangenen Sprite-Programme laden und die Zeilen 1410-1440 hinzufügen.

Geben Sie das Programm »Vierfarben-Sprite« ein, speichern und starten Sie es anschließend. Versuchen Sie, die Farbwerte in den Zeilen 1410-1440 zu modifizieren und andere Kombinationen zu finden.

Nachdem Sie diese Experimente beendet haben, ist es Zeit, Ihre eigenen Daten zu testen. Ersetzen Sie die Bildpunktdaten in den Zeilen 1150-1250 des Programms »Vierfarben-Sprite« durch die von Ihnen im letzten Abschnitt errechneten Werte, und starten Sie das Programm anschließend erneut. Entspricht das Ergebnis Ihren Erwartungen? Manchmal muß man etwas herumbasteln, ehe die Daten mit dem gewünschten Bild übereinstimmen.

Sprites unterschiedlicher Prioritäten

Wenn sich Sprites über den Bildschirm bewegen, kann es vorkommen, daß sie miteinander kollidieren und sich überschneiden. In diesem Fall bestimmt der C 64, in welcher

```
1000 rem *** vierfarben-sprite *** <054>
1010 : <224>
1020 : <234>
1030 rem ** ausgabe programmierung <013>
1040 : <000>
1050 print "{clr,8down}bitte warten" <034>
1060 : <020>
1070 : <030>
1080 rem ** sprite-daten laden <212>
1090 : <050>
1100 for n = 896 to 958 <018>
1110 : read spdta <061>
1120 : poke n, spdta <098>
1130 next n <238>
1140 : <100>
1150 data 1, 85, 64, 1, 85, 64 <234>
1160 data 1, 20, 64, 1, 20, 64 <240>
1170 data 1, 85, 64, 1, 20, 64 <252>
1180 data 1, 65, 64, 1, 85, 64 <199>
1190 data 0, 60, 0, 0, 60, 0 <028>
1200 data 62,170,188, 62,170,188 <190>
1210 data 48,170, 12, 16,170, 4 <203>
1220 data 20,130, 20, 20,130, 20 <180>
1230 data 16,195, 4, 0,195, 0 <184>
1240 data 0, 65, 0, 1, 65, 64 <106>
1250 data 1, 65, 64 <010>
1260 : <220>
1270 : <230>
1280 rem ** sprite-register vorbereiten <000>
1290 print "{clr}" :rem schirm loesch <069>
1310 poke 2040,14 :rem datenzeiger <243>
1320 vic = 53248 :rem grafikbaustein <135>
1330 : <036>
1340 poke vic,160 :rem horiz.position <007>
1350 poke vic+1,129 :rem vertk.position <227>
1360 : <066>
1370 poke vic+23,1 :rem verlaengerung <026>
1380 poke vic+29,1 :rem verbreiterung <235>
1390 : <096>
1400 poke vic+28,1 :rem mehrfarbmodus <190>
1410 poke vic+33,0 :rem hgrund schwarz <070>
1420 poke vic+37,7 :rem mfreg 0 gelb <182>
1430 poke vic+39,5 :rem sprite 0 gruen <078>
1440 poke vic+38,6 :rem mfreg 1 blau <119>
1450 : <156>
1460 poke vic+21,1 :rem sprite 0 ein <225>
1470 : <176>
1480 : <186>
1490 rem ** prog.ende mit tastendruck <255>
1500 : <206>
1510 get kp$ : <089>
1520 if kp$ = "" then 1510 <191>
1530 : <236>
1540 : <248>
1550 rem ** ruecksetzen des registers <137>
1560 : <012>
1570 poke vic+21,0 :rem sprite aus <133>
1580 poke vic+28,0 :rem mehrfarb.aus <233>
1590 poke vic+29,0 :rem verbreit.aus <011>
1600 : <052>
1610 : <062>
1620 end <098>
```

Listing 9. Ausdruck des Programms »Vierfarben-Sprite«

Rangfolge die Sprites dargestellt werden, das heißt, welches der beiden Sprites verdeckt und welches angezeigt wird. Die höchste Priorität hat immer Sprite 0, die niedrigste Sprite 7. Bei einer Überlagerung von Sprite 0 und Sprite 7 wird also Sprite 0 dargestellt. Ebenso hat zum Beispiel Sprite 4 eine höhere Priorität als Sprite 5. Bild 23 zeigt die Rangfolge der Sprites.

Bei der vollständigen oder teilweisen Überlagerung zweier Sprites werden alle Bildpunkte des Sprites mit der höheren Priorität, die unsichtbar, also in der Farbe des Bildschirms dargestellt werden, sozusagen transparent, so daß die entsprechenden Teile des darunterliegenden Sprites wie durch ein Fenster sichtbar sind.

Listing 10 zeigt das Programm »Sprite-Überlagerung«. Geben Sie es ein, speichern und starten Sie es.

»Sprite Überlagerung« stellt vier gleichartige Sprites auf dem Bildschirm dar und verschiebt sie dann so lange überein-

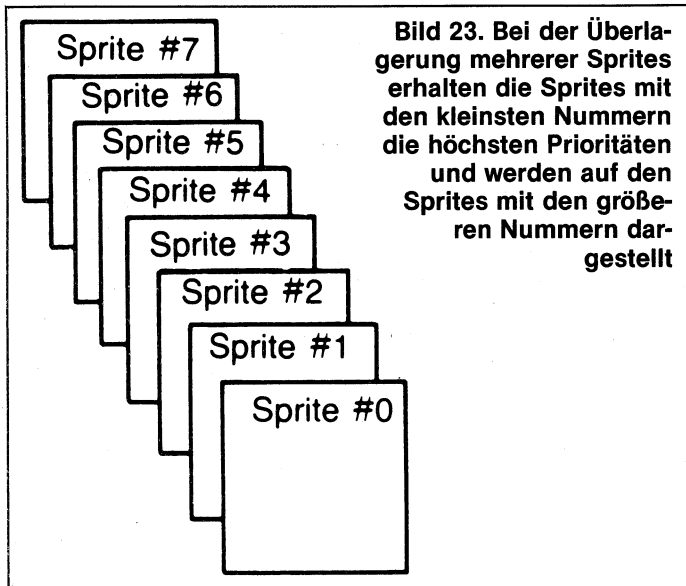


Bild 23. Bei der Überlagerung mehrerer Sprites erhalten die Sprites mit den kleinsten Nummern die höchsten Prioritäten und werden auf den Sprites mit den größten Nummern dargestellt

ander, bis eine Taste gedrückt wird. Beachten Sie, wie bei der Überlagerung Teile der unteren Sprites durch die transparenten Bereiche der Sprites 0 und 1 hindurch sichtbar werden.

Abgesehen von der Überlagerung von Sprites ist in dem Programm auch interessant, auf welche Weise die Gestalt der Sprites definiert und der Bewegungsablauf programmiert wird.

Die Zahlencodes für das Sprite im Programm »Sprite-Überlagerung« werden durch eine Programmschleife generiert:

```
1100 FOR N = 832 TO 894
1110 : POKE N, 60
1120 NEXT N
```

Eine ähnliche Technik haben Sie bereits im Programm »ein einfaches Sprite« benutzt. Dort enthielt der POKE-Befehl den Wert 255, und jeder Bildpunkt des Sprites wurde sichtbar. Im vorliegenden Fall bewirkt die Zahl 60, daß nur die mittleren vier von jeweils 8 Bildpunkten sichtbar werden. Dieser Effekt ist in Bild 24 dargestellt. Enthält jede Sprite-Zeile drei Muster dieser Art, so besteht das Sprite aus drei vertikal verlaufenden Streifen.

Mit Hilfe einer Programmschleife lassen sich viele Sprites erzeugen. Testen Sie zum Beispiel folgende Zeilen:

```
1100 FOR N = 832 TO 894 STEP 2
1110 : POKE N, 255 : POKE N+1, 0
```

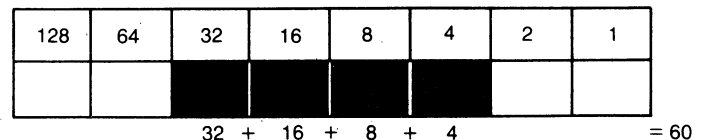


Bild 24. Der Zahlencode 60 für ein Sprite bewirkt, daß die vier mittleren der acht Bildpunkte jeder Gruppe sichtbar werden

```
1000 rem *** sprite-ueberlagerung ***      <162>
1010 :                                     <224>
1020 :                                     <234>
1030 rem ** ausgabe programmmeldung      <013>
1040 :                                     <000>
1050 print "{clr,8down}bitte warten"      <034>
1060 :                                     <020>
1070 :                                     <030>
1080 rem ** sprite-daten laden            <212>
1090 :                                     <050>
1100 for n = 832 to 894                    <124>
1110 :   poke n, 60                        <016>
1120 next n                                <228>
1130 :                                     <090>
1140 :                                     <100>
1150 rem ** sprite-register vorbereiten    <124>
1160 :                                     <120>
1170 print "{clr}" :rem schirm loesch     <203>
1180 vic = 53248 :rem grafikbaustein      <249>
1190 :                                     <150>
1200 poke 2040,13 :rem datenzeiger 1      <025>
1210 poke 2041,13 :rem datenzeiger 2      <228>
1220 poke 2042,13 :rem datenzeiger 3      <174>
1230 poke 2043,13 :rem datenzeiger 4      <121>
1240 :                                     <200>
1250 poke vic,226 :rem horzntl.pos. 0      <235>
1260 poke vic+2,94 :rem horzntl.pos. 1     <160>
1270 poke vic+4,144 :rem horzntl.pos. 2    <182>
1280 poke vic+6,176 :rem horzntl.pos. 3    <227>
1290 :                                     <252>
1300 poke vic+1,140 :rem vertal.pos. 0     <109>
1310 poke vic+3,118 :rem vertal.pos. 1     <174>
1320 poke vic+5,190 :rem vertal.pos. 2     <155>
1330 poke vic+7,68 :rem vertal.pos. 3      <150>
1340 :                                     <046>
1350 poke vic+39,7 :rem 0 ist gelb          <083>
1360 poke vic+40,5 :rem 1 ist gruen         <098>
1370 poke vic+41,3 :rem 2 ist hellblau     <002>
1380 poke vic+42,1 :rem 3 ist weiss         <182>
1390 :                                     <096>
1400 poke vic+23,15 :rem alle sprites      <080>
1410 poke vic+29,15 :rem dopp. groesse     <108>
1420 :                                     <126>
1430 poke vic+21,15 :rem sprite 0-3 an     <182>
1440 :                                     <146>
1450 :                                     <156>
1460 rem ** beweg.register vorbereiten     <144>
1470 rem   und anf.schritte zuweisen      <208>
1480 :                                     <186>

1490 mr(0) = vic : mr(1) = vic+2          <154>
1500 mr(2) = vic+5 : mr(3) = vic+7        <041>
1510 :                                     <216>
1520 mv(0) = -1 : mv(1) = 1               <201>
1530 mv(2) = -1 : mv(3) = 1               <147>
1540 :                                     <248>
1550 df = -1 :rem -1=n.innen 0=n.aussen    <205>
1560 :                                     <012>
1570 :                                     <022>
1580 rem ** sprite bewegen                 <059>
1590 :                                     <042>
1600 for count = 1 to 200                 <007>
1610 :   sprnum = int((count-1)/50)        <150>
1620 :   if df then sprnum = 3 - sprnum     <144>
1630 :   reg = mr(sprnum)                  <245>
1640 :   move = mv(sprnum)                  <140>
1650 :   poke reg, peek(reg) + move        <084>
1660 :   get kp$                           <068>
1670 :   if kp$ = "" then 1690              <025>
1680 :   count = 200 : keypress = -1       <009>
1690 next count                           <037>
1700 :                                     <152>
1710 :                                     <162>
1720 rem ** prog.end nach tastendruck    <246>
1730 :                                     <182>
1740 if keypress then 1900                 <239>
1750 :                                     <202>
1760 :                                     <212>
1770 rem ** pause, dann richtung um-     <069>
1780 rem   kehren und von vorn            <182>
1790 :                                     <242>
1800 for delay = 1 to 400 : next delay    <212>
1810 for sprnum = 0 to 3                   <022>
1820 :   mv(sprnum) = -1 * mv(sprnum)      <102>
1830 next sprnum                           <177>
1840 df = -1 - df                          <018>
1850 goto 1600                             <108>
1860 :                                     <058>
1870 :                                     <068>
1880 rem ** register zuruecksetzen        <115>
1890 :                                     <088>
1900 poke vic+21,0                         <146>
1910 poke vic+29,0                         <164>
1920 poke vic+23,0                         <168>
1930 :                                     <128>
1940 end                                    <164>
```

Listing 10. Ausdruck des Programms »Sprite-Überlagerung«

Sicherlich können Sie mit Hilfe dieser Technik noch eine Vielzahl weiterer Sprites erfinden. Starten Sie die neue Version Ihres Programms, und lassen Sie sich nicht hypnotisieren. Versuchen Sie, herauszufinden, wie viele weitere Sprites Sie durch den geschickten Einsatz von Programmschleifen erzeugen können.

Der Bewegungsablauf

Beim Start des Programmes »Sprite-Überlagerung« befinden sich die vier Sprites in den in Bild 25 dargestellten Grundpositionen. Nacheinander bewegen sie sich dann auf die Bildmitte zu, legen dort eine Pause ein, wandern anschließend auf ihre Ausgangspositionen zurück und beginnen den Reigen nach einer weiteren Pause erneut.

Für die Programmierung von Bewegungsabläufen ist es von Vorteil, die auszuführende Bewegung auf gleichartige Abläufe hin zu untersuchen. Im vorliegenden Fall muß jedes Sprite nur zwei Bewegungen ausführen: sich in den Bildmittelpunkt bewegen und auf gleicher Bahn zurückkehren. Daher kann ein Programmsegment, das diese Bewegungen steuert, von allen vier Sprites benutzt werden. Sind diese Bewegungsvariablen als Feldvariable definiert, so sind die Bewegungen besonders einfach zu programmieren: für das entsprechende Sprite muß als Index nur die entsprechende Sprite-Nummer angegeben werden.

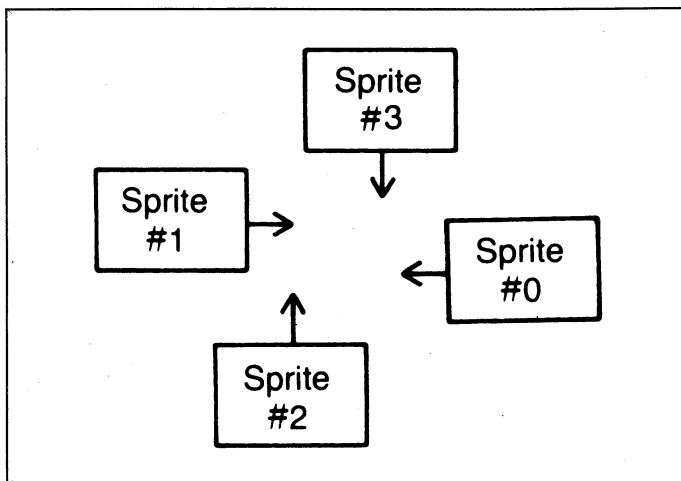


Bild 25. Die Ausgangsposition der vier Sprites aus dem Programm »Sprite-Überlagerung«. Diese Pfeile geben die Bewegungsrichtung nach dem Programmstart an.

Eine weitere Vereinfachung ergibt sich aus der Tatsache, daß die beiden Bewegungen, die ein Sprite auszuführen hat, sich nur in der Richtung unterscheiden. Nach Abschluß einer Bewegung muß also nur der Wert der Bewegungsvariablen umgekehrt werden. Damit ist dann im Prinzip die Steuerung aller vier Sprites durch ein Programmsegment möglich. Nur die Feinheiten müssen noch genauer ausgearbeitet werden (ein berühmtes letztes Wort vieler Programmierer).

Da ein Sprite sich entweder nur horizontal oder nur vertikal bewegt, ist ein Positionsregister pro Sprite ausreichend. In den Zeilen 1490-1500 werden die entsprechenden Registeradressen an vier Variable übergeben:

```
1490 MR(0) = VIC :MR(1) = VIC+2
1500 MR(2) = VIC+5 :MR(3) = VIC+7
```

Betrachten Sie noch einmal Bild 25. Sprite 0 und 1 bewegen sich horizontal, Sprite 2 und 3 vertikal. Entspre-

chend diesen Richtungen wurden die Positionsregister in den Zeilen 1490-1500 ausgewählt.

In den Zeilen 1520-1530 werden die Bewegungsvariablen definiert:

```
1520 MV(0) = -1 : MV(1) = 1
1530 MV(2) = -1 : MV(3) = 1
```

Der jeweilige Wert der Richtungsvariablen, addiert zum aktuellen Positionswert, ergibt die neue Position des entsprechenden Sprites. Betrachten Sie dazu noch einmal das Bild 25. Die Pfeile kennzeichnen die Richtung der Sprites zu Beginn des Programms. Sprite 3 zum Beispiel beginnt mit einer Bewegung nach unten. Nach jedem Schritt erhöht sich sein Positionswert, und genau diese Vergrößerung des Wertes wird in Zeile 1530 vorgenommen. Um die Bewegungsrichtung umzukehren, genügt es, den Wert der Variablen MV(3) mit -1 zu multiplizieren. Dann nämlich verkleinert sich der vertikale Positionswert bei jedem Schritt um 1, und Sprite 3 bewegt sich wieder hinauf.

Es ist noch eine weitere Einzelheit zu berücksichtigen: die Reihenfolge der Sprites. Bei der Bewegung nach innen (auf den Mittelpunkt zu) soll die Bewegung in der Reihenfolge 3, 2, 1, 0 erfolgen, bei der Bewegung nach außen (vom Mittelpunkt weg) dagegen in der Reihenfolge 0, 1, 2, 3. Zeile 1550 initialisiert eine Variable DF, mit der die Reihenfolge gesteuert wird:

```
1550 DF = -1 :REM -1=N.INNEN 0=N.AUSSEN
```

Die allgemeine Bewegungsschleife

Die Zeilen 1600-1690 bilden die Bewegungsschleife:

```
1600 FOR COUNT = 1 TO 200
1610 : SPRNUM = INT((COUNT-1)/50)
1620 : IF DF THEN SPRNUM = 3 - SPRNUM
1630 : REG = MR(SPRNUM)
1640 : MOVE = MV(SPRNUM)
1650 : POKE REG, PEEK(REG) + MOVE
1660 : GET KP$
1670 : IF KP$ = "" THEN 1690
1680 : COUNT = 200 : KEYPRESS = -1
1690 NEXT COUNT
```

Die Zeilen 1600 und 1690 definieren eine Schleife, die 200mal durchlaufen wird, falls Sie nicht zwischenzeitlich eine beliebige Taste betätigen. Die Schleifenvariable hat den Endwert 200, da jedes der vier Sprites einen Weg von 50 Bildpunkten zurückzulegen hat ($4 * 50 = 200$).

Die Zeilen 1610 und 1620 bestimmen, welches Sprite bewegt werden soll, und speichern die entsprechende Sprite-Nummer in der Variablen SPRNUM. Bei der Bewegung nach innen hat DF den Wert -1, und SPRNUM nimmt nacheinander die Werte 3, 2, 1 und 0 an. Bei der Bewegung nach außen, wenn DF den Wert 0 hat, erhält SPRNUM die Werte 0, 1, 2 und 3.

Zeile 1630 wählt, entsprechend dem Wert von SPRNUM, das Positionsregister, und Zeile 1640 die Schrittweite. In Zeile 1650 wird dann die Schrittweite zum alten Positionswert addiert.

Zeile 1660 überprüft die Tastatur. Wurde eine Taste betätigt, beendet Zeile 1670 die Schleife. Sie haben diese Technik schon im vorhergehenden Abschnitt kennengelernt. Nachdem die Sprites ihre Bewegung nach innen oder außen beendet haben, muß die Bewegungsrichtung umgekehrt werden:

```

1000 rem *** jongleur ***
1010 :
1020 :
1030 rem ** ausgabe programmmeldung
1040 :
1050 print "{clr,8down}bitte warten";
1060 :
1070 :
1080 rem ** sprite-daten laden
1090 :
1100 for n = 832 to 1023
1110 : read spdta
1120 : if spdta = -1 then print ". "; :
      goto 1140
1130 : poke n, spdta
1140 next n
1150 :
1160 data 0, 16, 0, 0, 0, 0
1170 data 1, 0, 128, 0, 0, 0
1180 data 0, 0, 0, 0, 120, 0
1190 data 4, 120, 16, 0, 120, 0
1200 data 0, 120, 0, 12, 24, 0
1210 data 15, 255, 16, 0, 61, 128
1220 data 0, 60, 176, 4, 24, 240
1230 data 0, 61, 0, 0, 60, 0
1240 data 0, 36, 0, 0, 36, 0
1250 data 0, 36, 0, 0, 36, 0
1260 data 0, 102, 0, -1
1270 :
1280 data 0, 8, 0, 0, 64, 0
1290 data 0, 0, 0, 0, 0, 64
1300 data 0, 0, 0, 4, 60, 0
1310 data 0, 60, 0, 0, 60, 0
1320 data 0, 60, 16, 8, 24, 0
1330 data 13, 255, 0, 15, 61, 48
1340 data 0, 61, 240, 0, 24, 0
1350 data 0, 190, 0, 0, 60, 0
1360 data 0, 36, 0, 0, 36, 0
1370 data 0, 36, 0, 0, 36, 0
1380 data 0, 102, 0, -1
1390 :
1400 data 0, 32, 0, 0, 2, 0
1410 data 0, 0, 0, 2, 0, 0
1420 data 0, 0, 32, 0, 60, 0
1430 data 0, 60, 0, 0, 60, 0
1440 data 8, 60, 0, 0, 24, 16
1450 data 0, 255, 152, 1, 188, 248
1460 data 13, 60, 0, 15, 24, 64
1470 data 0, 60, 0, 0, 60, 0
1480 data 0, 36, 0, 0, 36, 0
1490 data 0, 36, 0, 0, 38, 0
1500 data 0, 96, 0, -1
1510 :
1520 :
1530 rem ** sprite register vorbereiten
1540 :
1550 print "{clr}" : rem schirm loesche
      :
1560 vic = 53248 : rem grafikbaustein
1570 :
1580 poke 2040, 13 : rem datenzeiger 0
1590 poke vic, 160 : rem horzntl.pos. 0
1600 poke vic+1, 129 : rem vertkal.pos. 0
1610 poke vic+39, 1 : rem 0 ist weiss
1620 poke vic+29, 1 : rem sprite 0 hat
1630 poke vic+23, 1 : rem dopp. groesse
1640 poke vic+21, 1 : rem sprite 0 ein
1650 :
1660 :
1670 rem ** jonglieren beginnt
1680 :
1690 image = peek (2040) + 1
1700 if image = 16 then image = 13
1710 poke 2040, image
1720 :
1730 for delay = 1 to 30 : next delay
1740 :
1750 :
1760 rem ** prog.ende nach tastendruck
1770 :
1780 get kp$
1790 if kp$ = "" then 1690
1810 poke vic+21, 0 : rem sprites aus
1820 poke vic+29, 0 : rem und wieder
1830 poke vic+23, 0 : rem normalgroesse
1840 :
1850 end

```

Listing 11. Ausdruck des Programms »Jongleur«

```

1810 FOR SPRNUM = 0 TO 3
1820 : MV(SPRNUM) = -1 * MV(SPRNUM)
1830 NEXT SPRNUM
1840 DF = -1 - DF
1850 GOTO 1600

```

Zunächst wird die Bewegungsrichtung jedes Sprites durch Multiplikation mit -1 umgekehrt, dann, in Zeile 1840, der Variablenwert für die Reihenfolge angepaßt. 0 wird zu -1, und -1 zu 0. Zeile 1850 schließlich übergibt die Steuerung wieder an die Bewegungsschleife, die in Zeile 1600 beginnt. Auf diese Weise bewegen sich die Sprites vor und zurück, bis ein Tastendruck ihren Tanz unterbricht.

Die vorliegende Bewegungsschleife bietet eine Vielzahl von Variationsmöglichkeiten. Versuchen Sie, andere Bewegungsformen zu realisieren. Dazu einige Vorschläge:

- * Bewegen Sie zwei Sprites gleichzeitig.
- * Bewegen Sie die Sprites bei der Bewegung nach außen auf neue Ausgangspositionen zu.
- * Überlagern Sie die Sprites in der Bildmitte vollständig.

Trickbilder

In allen bisherigen Programmen wurde die Bewegung der Sprites durch eine Veränderung des jeweiligen Positionswertes bewirkt. Das Sprite selbst änderte dabei seine Form nicht. Diesen Effekt können Sie jedoch hervorrufen, wenn Sie mehrere unterschiedliche Sprites laden und dann nacheinander an derselben Bildschirmposition darstellen, indem Sie den Datenzeiger in einer Schleife nacheinander auf die Datenblöcke zeigen lassen. Erfolgt die Darstellung der einzelnen Sprites schnell genug, so entsteht, ähnlich wie beim Trickfilm, der Eindruck einer ständig die Form wechselnden Figur.

Bild 26 zeigt drei Einzelbilder, die in schnellem Wechsel übereinandergeblendet werden sollen. Sie stellen einen Jongleur in drei verschiedenen Stellungen dar. Die Bewegungsfolge beginnt nach dem dritten Bild wieder mit dem ersten. Der Entwurf einer solchen Bildfolge gelingt meistens nicht auf Anhieb. Entwerfen Sie zunächst die Einzelbilder, wobei jedes Bild die Fortsetzung des vorangegangenen sein sollte. Die Bildpunktdaten berechnen Sie wie immer mit Hilfe der Entwurfsblätter. Sie benötigen diese Daten im folgenden Abschnitt. Hier einige Themen: ein springender Ball, ein zwinkerndes Auge, ein lachendes Gesicht, ein blinkender Stern, eine sich verlängernde und verkürzende Linie, ein Blizzard.

Listing 11 enthält das Programm »Jongleur«. Es stellt die in Bild 26 gezeigten Bilder in der beschriebenen Weise auf dem Bildschirm dar.

Die Zeilen 1100-1130 generieren die Sprite-Daten. Zeile 1120 enthält eine interessante Einzelheit:

```
1120 : IF SPDTA = -1 THEN PRINT ". "; : GOTO 1140
```

Ein Sprite wird durch 63 Datenbytes definiert. Wie in Kapitel 2 beschrieben, zeigt der Datenzeiger jedoch immer auf einen Block von jeweils 64 Bytes. Liegen die Datenblöcke direkt hintereinander, so lassen sich die Daten mit einer einzigen Schleife nacheinander laden, wenn jeder Datenblock mit einem vierundsechzigsten Füllbyte aufgefüllt wird. Geben Sie nun diesem Füllbyte einen Wert, der in einem Datenblock normalerweise nicht vorkommt, so kann das Programm das Ende jedes Datenblocks erkennen und eine entsprechende Meldung ausgeben. Im vorliegenden Programm hat das Füllbyte den Wert -1.

Die drei Datenblöcke der Sprite-Bilder haben die Adressen 832-894, 896-958 und 960-1022. Durch Division der An-

fangsadresse jedes Blocks durch 64 ergeben sich die Werte 13, 14 und 15 für die Datenzeiger. Das Programm führt die Überlagerung der Bilder aus, indem es dem Datenzeiger für Sprite 0 (Adresse 2040) in einer Schleife nacheinander die Werte 13, 14 und 15 zuordnet und diese Schleife ständig wiederholt.

In den Zeilen 1580-1640 werden die Sprite-Register wie gewohnt initialisiert. Da der Datenzeiger für Sprite 0 hier den Wert 13 erhält, beginnt die Vorstellung mit dem im Datenblock 832-894 gespeicherten Bild.

Die Zeilen 1690-1710 tauschen die Bilder aus:

```
1690 IMAGE = PEEK (2040) +1
1700 IF IMAGE = 16 THEN IMAGE =13
1710 POKE 2040, IMAGE
```

In Zeile 1690 wird der Wert des aktuellen Datenzeigers um 1 erhöht. Hat er den Wert 16 erreicht, setzt Zeile 1700 ihn auf 13 zurück. In Zeile 1710 erfolgt die Eintragung des Wertes in die entsprechende Speicherzelle. Der Zeiger durchläuft somit die Werte 13, 14, 15 und beginnt dann erneut mit 13.

Zeile 1730 ist nur eine Verzögerungsschleife. Durch Veränderung des Schleifenendwertes wird der Jongleur schneller oder langsamer.

Die Zeilen 1780-1790 enthalten schließlich die gewohnte Tastaturabfrage. Wurde keine Taste betätigt, so wird die Steuerung an Zeile 1690 übergeben und das nächste Bild angezeigt. Anderenfalls endet das Programm nach Rücksetzung der Datenzeiger.

Ersetzen Sie im Programm »Jongleur« nun die Daten in den

Zeilen 1160-1500 durch die Codes für Ihre eigenen Trickbilder, und starten Sie das Programm. Stimmt der Ablauf mit Ihren Vorstellungen überein? Experimentieren Sie mit der Reihenfolge der Einzeldarstellungen, ändern Sie den Endwert der Verzögerungsschleife, benutzen Sie andere Daten. Durch derartige Experimente lernen Sie die Anwendung der beschriebenen Programmtechnik.

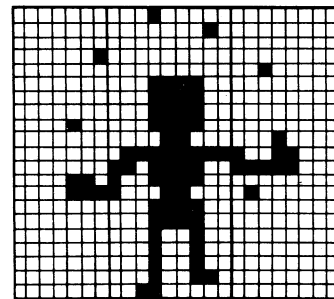
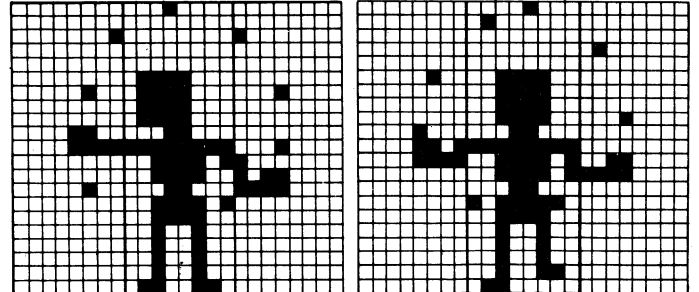


Bild 26.
Anfangsbild
nach dem
Start des
Programms

```

1000 rem *** lustige kollision ***
1010 :
1020 :
1030 rem ** sprite-daten laden
1040 :
1050 print "{clr}"
1060 for n = 832 to 958
1070 : read spdta
1080 : poke n, spdta
1090 next n
1100 :
1110 data 0,255, 0, 1,129,128
1120 data 3, 0,192, 3, 0,192
1130 data 3, 0,192, 6,102, 96
1140 data 60,102, 60, 96, 0, 6
1150 data 192, 0, 3,192,102, 3
1160 data 198, 60, 99, 99, 0,198
1170 data 113,129,142, 28,195, 56
1180 data 12,195, 48, 12,102, 48
1190 data 6, 60, 96, 6, 0, 96
1200 data 3,129,192, 0,195, 0
1210 data 0,126, 0, 0
1220 :
1230 data 0, 0, 0, 0, 16, 0
1240 data 0, 56, 0, 0, 84, 0
1250 data 0, 16, 0, 2, 16,128
1260 data 1, 17, 0, 0,146, 0
1270 data 16, 84, 16, 32, 56, 8
1280 data 127,255,252, 32, 56, 8
1290 data 16, 84, 16, 0,146, 0
1300 data 1, 17, 0, 2, 16,128
1310 data 0, 16, 0, 0, 84, 0
1320 data 0, 56, 0, 0, 16, 0
1330 data 0, 0, 0
1340 :
1350 :
1360 rem ** sprites vorbereiten und ein
1370 :
1380 vic = 53248 :rem grafikbaustein
1390 poke vic+33,0 :rem hgrnd schwarz
1400 :
1410 poke 2040,13 :rem datenzeiger 0
1420 poke 2041,14 :rem datenzeiger 1
1430 :
1440 poke vic,120 :rem horzntl.pos. 0
1450 poke vic+2,160 :rem horzntl.pos. 1
1460 poke vic+1,138 :rem vertkal.pos. 0
1470 poke vic+3,126 :rem vertkal.pos. 1
1480 :
1490 poke vic+39,3 :rem 0 ist hellblau
1500 poke vic+40,7 :rem 1 ist gelb
1510 poke vic+29,2 :rem nur 1 ist von
1520 poke vic+23,2 :rem dopp. groesse
1530 :
1540 poke vic+21,3 :rem beide sichtb.
1550 :
1560 :
1570 rem ** bewegung sprite 0
1580 :
1590 jr = peek (56320) :rem eingang 2
1600 if (jr and 16) = 0 then 1870
1610 hd = sgn(jr and 4) - sgn(jr and 8)
1620 vd = sgn(jr and 1) - sgn(jr and 2)
1630 :
1640 poke vic, peek(vic) + hd
1650 poke vic+1, peek(vic+1) + vd
1660 :
1670 :
1680 rem ** zurueck, wenn keine kollis.
1690 :
1700 if peek(vic+30) = 0 then 1490
1710 :
1720 :
1730 rem ** kollision : 1 wird weiss
und 0 blinkt bunt
1740 :
1750 poke vic+40, 1
1760 :
1770 hue = peek(vic+39) and 15
1780 hue = hue + 1
1790 if hue = 8 then hue = 1
1800 poke vic+39, hue
1810 :
1820 goto 1590
1830 :
1840 :
1850 rem ** ausgangswert und ende
1860 :
1870 poke vic+21,0
1880 poke vic+29,0
1890 poke vic+23,0
1900 :
1910 end

```

Listing 12. Das Programm »Lustige Kollision«

Sprite-Kollisionen

Es ist für viele Anwendungen nützlich zu wissen, wann bestimmte Objekte auf dem Bildschirm miteinander kollidieren. Bei den früheren Heimcomputern war dies sehr schwierig festzustellen. Der Commodore 64 enthält jedoch Schaltkreise, die in der Lage sind, solche Kollisionen zu erkennen.

Die Kollision zweier Sprites wird in Register VIC+30 (Adresse 53278) registriert. Jedes Bit dieses Register ist einem Sprite zugeordnet. Ist ein Sprite in eine Kollision mit einem anderen Sprite verwickelt, erhält sein Bit den Wert 1. Kollidiert zum Beispiel Sprite 2 mit Sprite 7, so werden die Bits 2 und 7 des Registers VIC+30 auf 1 gesetzt. Eine Rücksetzung dieser Bits auf 0 erfolgt nicht dadurch, daß beide Sprites sich wieder voneinander entfernen, sondern nur durch einen PEEK-Befehl, mit dem das Register VIC+30 gelesen wird.

Das Register VIC+31 (Adresse 53279) registriert Kollisionen zwischen Sprites und Hintergrundobjekten. Hintergrundobjekte sind entweder Textzeichen (Textmodus) oder Teile einer hochauflösenden Grafik (Grafikmodus). Wiederum ist jedem Sprite ein Bit des Registers zugeordnet. Das Bit wird bei einer Kollision auf 1 gesetzt. Kollidiert zum Beispiel Sprite 5 im Textmodus mit einem Textzeichen, so erhält Bit 5 in Adresse VIC+31 den Wert 1. Das Bit behält diesen Wert, bis das Register mit einem PEEK-Befehl gelesen wird.

Listing 12 enthält das Programm »Lustige Kollision«. Es enthält Beispiele zum Lesen der Steuerknüppelregister und zur Registrierung der Kollisionen zweier Sprites. Geben Sie das Programm ein, speichern und starten Sie es. Daraufhin werden die in Bild 27 dargestellten Sprites sichtbar. Benutzen Sie einen an Eingang 2 angeschlossenen Steuerknüppel, um das Gesicht auf der Wetterfahne zu bewegen. Beachten Sie, was geschieht, wenn die beiden Sprites kollidieren. Ein Druck auf den Feuerknopf beendet das Programm.

Die Zeilen 1050-1090 laden die Daten für beide Sprites, die Zeilen 1380-1540 initialisieren die erforderlichen VIC-Register und aktivieren die Sprites.

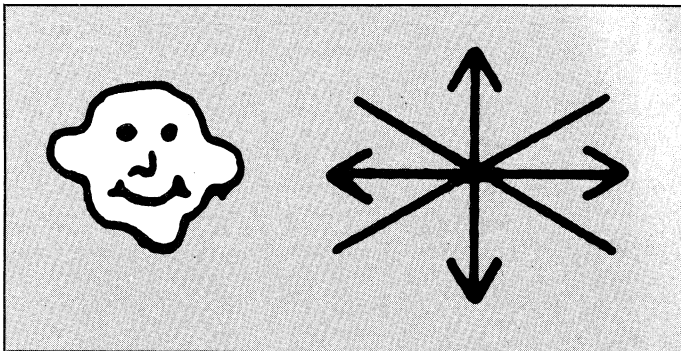


Bild 27. Anfangsbild nach dem Start des Programms »Lustige Kollision«

Nun folgt der Hauptteil des Programms. Zeile 1590 liest den Wert des E/A-Registers mit der Adresse 56320. Wie Sie vielleicht wissen, ist diese Adresse dem Steuerknüppel in Eingang 2 zugeordnet. In Zeile 1600 wird eine AND-Funktion benutzt, um festzustellen, ob der Feuerknopf gedrückt ist. Ist das der Fall, so endet das Programm mit der Rücksetzungs-routine ab Zeile 1870.

Die Zeilen 1610-1620 bestimmen aus dem Wert in Register 56320 die auszuführende horizontale und vertikale Bewegung, und zwar auf eine schnelle und trickreiche Weise. Mit Hilfe der AND-Funktion ergeben sich die Werte der den einzelnen Schaltern zugeordneten Bits. Die SGN-Funktion liefert entweder den Wert 0 oder den Wert 1, je nachdem, ob

der Wert in Klammern gleich oder größer als 0 ist. Je nach Stellung des Steuerknüppels erhält HD, die Variable für die horizontale Bewegung, den Wert -1, 0 oder 1. Die gleichen Werte kann VD, die Variable für die vertikale Bewegung, annehmen. Beide Bewegungsvariablen werden dann benutzt, um die neue Position für Sprite 0 festzulegen.

Zeile 1700 überprüft das Register, in dem die Kollision zweier Sprites registriert wird. Findet keine Kollision statt, springt das Programm zurück, aktiviert wieder die alten Sprite-Farben und fragt erneut das Steuerknüppelregister ab. Ist dagegen eine Kollision erfolgt, so werden in den Zeilen 1750-1800 erst die Sprite-Farben geändert, ehe das Programm zur Abfrage des Steuerknüppels springt.

Zusammenfassung

In diesem Kapitel haben Sie gelernt:

- die VIC-II-Register so zu programmieren, daß ein Sprite in vier Farben dargestellt wird,
- ein mehrfarbiges Sprite zu entwerfen,
- zu erkennen, welche Sprites bei einer Überlagerung sichtbar bleiben,
- weitere Einzelheiten, um mehrere Sprites gleichzeitig zu bewegen,
- mit Hilfe eines durch eine Programmschleife gesteuerten Datenzeigers eine Tricksequenz aus mehreren Sprites zu erzeugen.
- Sprite-Kollisionen abzufragen

Ein Kurs dieses Umfangs kann Ihnen nur Anfangskenntnisse in der Programmierung von Sprites vermitteln. Durch Experimente mit den vorgestellten und mit eigenen Programmen werden Sie Ihre Kenntnisse jedoch schnell vertiefen.

Selbsttest

1. Im Mehrfarbenmodus wird ein Doppelbildpunkt durch zwei Bits definiert. Damit lassen sich Farben darstellen.
2. Da Sprites im Mehrfarbenmodus nur eine Breite von 12 Doppelbildpunkten besitzen, ist ihre Auflösung kleiner als die eines einfarbigen Sprites.
3. Welche Sprites werden durch den Befehl POKE VIC+28,15 im Mehrfarbenmodus dargestellt?
4. Welches Sprite hat bei der Überschneidung mehrerer Sprites die höchste Priorität?
5. Welches Aussehen haben die Sprites, die sich im Programm »Sprite-Überlagerung« durch folgende Programmzeilen ergeben:

```
1100 FOR N = 832 TO 894 STEP 3
1105 : POKE N, 225
1110 : POKE N+1, 195
1115 : POKE N+2, 135
```

6. Betrachten Sie die Zeilen 1610-1620 im Programm »Sprite-Überlagerung«. Welchen Wert erhält SPRNUM in den Zeilen 1610 und 1620, wenn COUNT den Wert 120 und DF den Wert 0 hat?
7. Wie viele Punkte (.) werden neben der Programmierung im Programm »Jongleur« auf dem Bildschirm ausgegeben, während die Sprite-Daten gelesen werden?

8. Welche Wirkung hat eine Änderung des Endwertes der Verzögerungsschleife von 30 auf 100 im Programm »Jongleur«?

Programmierübungen

1. Modifizieren Sie das Programm »Vierfarben-Sprite« derart, daß aus den vorhandenen Sprite-Codes ein zweites Sprite erzeugt und ebenfalls im Mehrfarbenmodus angezeigt wird.
2. Modifizieren Sie das Programm »Sprite-Ueberlagerung« derart, daß sich die vier Sprites in der Bildschirmmitte vollständig überlagern.
3. Ändern Sie das Programm »Jongleur«, so daß sich die Kugeln des Jongleurs zunächst im Uhrzeigersinn bewegen, nach einer Weile die Richtung ändern, sich dann wieder rechts herum bewegen, und so weiter.

Antworten zum Selbsttest

1. vier
2. horizontale
3. 0, 1, 2 und 3
4. 0
5. Jedes Sprite besteht aus vier vertikalen Streifen (Bild 28).
6. 2
7. 3
8. Die Bewegungen verlangsamen sich.

Lösungsvorschläge für die Programmierübungen

Die folgenden Lösungen sind Änderungen oder Erweiterungen der in diesem Kapitel besprochenen Programme.

1. Laden Sie das Programm »Vierfarben-Sprite«, und geben Sie die folgenden Zeilen ein:

```
1000 REM *** ZWEI VIERFARBEN-SPRITES ***
1315 POKE 2041,14 :REM ZEIGER SPRITE1
1353 POKE VIC+2,160 :REM SPRITE 1 HPOS.
1356 POKE VIC+3,69 :REM SPRITE 1 VPOS.
1370 POKE VIC+23,3 :REM VERLAENGERN
1380 POKE VIC+29,3 :REM VERBREITERN
1400 POKE VIC+28,3 :REM 0+1 MEHRFARBEN
1435 POKE VIC+40,2 :REM SPRITE 1 ROT
1460 POKE VIC+21,3 :REM SPRITE 0+1 EIN
```

2. Laden Sie das Programm »Sprite-Überlagerung« und geben Sie die folgenden Zeilen ein:

```
1000 REM *** TOTALUEBERLAGERUNG ***
1250 POKE VIC,210 :REM HORZNTL.POS. 0
1260 POKE VIC+2,110 :REM HORZNTL.POS. 1
1270 POKE VIC+4,160 :REM HORZNTL.POS. 2
1280 POKE VIC+6,160 :REM HORZNTL.POS. 3
1300 POKE VIC+1,129 :REM VERTKAL.POS. 0
1310 POKE VIC+3,129 :REM VERTKAL.POS. 1
1320 POKE VIC+5,179 :REM VERTKAL.POS. 2
1330 POKE VIC+7,79 :REM VERTKAL.POS. 3
```

3. Laden Sie das Programm »Jongleur«, und geben Sie die folgenden Zeilen ein:

```
1000 REM *** RECHTS-LINKS-JONGLEUR ***
1655 JUGDIR = 1 :REM IM UHRZEIGERS.
1690 IMAGE = PEEK (2040) + JUGDIR
1705 IF IMAGE = 12 THEN IMAGE = 15
1712 :
1715 COUNT = COUNT + 1
1718 IF INT (COUNT/27) = COUNT/27
    THEN JUGDIR = -JUGDIR : COUNT = 0
```

Damit sind wir am Ende unseres Sprite-Kurses angelangt. Was Ihnen jetzt noch zu tun bleibt, ist das hier Gelernte in praktische Programme umzusetzen, etwa in Spiele. Natürlich können dabei noch Schwierigkeiten auftreten, aber mit diesem Kurs haben Sie alles nötige Wissen über Sprites an der Hand. (Stan Krute/ev)

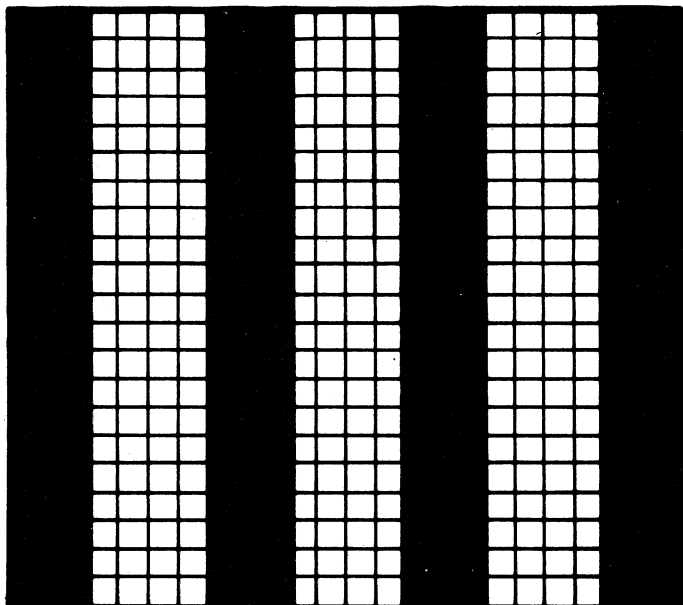


Bild 28. Die in Aufgabe 5 des Selbsttests genannten Programmzeilen ergeben dieses Sprite

