Pseudo-Sprites auf dem VC 20

Der VC 20 kennt von Haus aus leider nicht die freibeweglichen Grafikobjekte des C 64, die sogenannten Sprites. Das bedeutet aber nicht, daß man auf die Vorteile der Sprites oder MOBs gänzlich verzichten muß.

Das Programm ist für den VC 20 mit 8 KByte Speichererweiterung konzipiert. Es läuft jedoch mit einigen Änderungen auch bei nur 3 KByte Speichererweiterung.

Vor dem Eintippen oder Laden muß man POKE 44,32:POKE 8192,0: NEW eingeben, womit der Basic-Anfang im Speicher auf die Adresse dezimal 8193 ($2001) erhöht wird. Somit ergibt sich folgende Speicheraufteilung:

4096 — 4607 Bildschirm
4608 — 8191 frei
8192 — 16383 (bei + 8 KByte) Basic-Programmspeicher
8192 — 24575 (bei + 16 KByte) Basic-Programmspeicher
8192 — 32767 (bei + 24 KByte) Basic-Programmspeicher

Der freie Bereich wird nun vollständig von dem Maschinenspracheprogramm gebraucht. Die Aufteilung des Speicherraums ist die folgende:

4608 — 5119	Sprite-Control-Block (SCB), wird später erklärt
5120 — 6143	freidefinierbarer Zeichensatz, die ersten 128 Zeichen, stehen zur freien Verfügung
6144 — 7167	freidefinierbarer Zeichensatz, die zweiten 128 Zeichen, werden vom Programm zur Erstellung der 9 Sprites gebraucht und stehen somit nicht zur freien Verfügung
7168 — 8191	Maschinenspracheprogramm, Beschreibung siehe Text

Die Pseudo-Sprites sollten eine Auflösung von 16 x 16 Punkten haben, das sind 256 Punkte oder 4 Zeichen im freidefinierbaren Zeichensatz (Bild 1). Damit aber ein 16 x 16 Punkte großes Zeichen jede Position auf dem Bildschirm einnehmen kann, braucht man eine 24 x 24 Punkte große Umdefinier-Matrix, in die das Zeichen hineinkopiert wird. Das Aussehen dieser Umdefinier-Matrix ist in Bild 2 zu sehen. Das Programm übernimmt nun die Aufgabe, das Zeichen in die Umdefinier-Matrix zu kopieren (Bild 3), in die richtige X-Position zu schieben (Bild 4), und dasselbe mit der Y-Position zu tun.

Außerdem werden die Zeichen, die später auf dem Bildschirm von den Sprites verdeckt werden, mit in die Umdefinier-Matrix hineinkopiert. So entsteht der Eindruck, daß die Sprites wirklich über die Zeichen wandern. Beim späteren Löschen werden die verdeckten Zeichen wieder hergestellt. Wie funktioniert das nun?

Im Speicher ab dezimal 4608 ist 9mal (für jedes Sprite einer) der sogenannte Sprite-Control-Block (SCB) eingerichtet. Er hat die Aufgabe, die momentane X- und Y-Position, die Farbe des Sprites, den Bildschirmmodus (gesetzt/gelöscht) des Sprites, die durch die Umdefinier-Matrix verdeckten 9 Zeichen und Farben zwischenzuspeichern:

Byte 0 — 8	Bildschirmcode der verdeckten Zeichen
Byte 9 — 17	Farbcode der verdeckten Zeichen
Byte 18	X-Position des Sprites
Byte 19	Y-Position des Sprites
Byte 20	Farbe des Sprites
Byte 21	Bildschirmmodus (gesetzt= $00/gelöscht=$FF)

Die Basisadresse des SCB errechnet sich somit aus der Formel Basisadresse = 4608 + Spritenummer x 22. Das Zwischenspeichern und die Auswertung der Parameter übernimmt natürlich das Maschinenprogramm. Über den SCB werden auch im nachfolgend beschriebenen Programm »Sprite-Definer« die Sprites initialisiert und deren Farbe festgelegt. Da nur die oberen 128 Zeichen des Zeichensatzes für die Sprites verwendet werden, hat man eine ausreichende Anzahl von noch frei definierbaren Zeichen, nämlich genau 128, zur Verfügung. Außerdem kommen jeweils 13 Zeichen, nämlich 4 für das Sprite und 9 für die Umdefinier-Matrix hinzu, wenn man auf ein Sprite verzichtet. Die Matrixen werden im Speicher so abgelegt:

128 — 131	Grundmatrix Sprite 0
132 — 140	Umdef.-Matrix Sprite 0
141 — 144	Grundmatrix Sprite 1
145 — 153	Umdef.-Matrix Sprite 1

Konkret wird das Programm (Listing 1) nun folgendermaßen bedient: Vor dem Laden oder Eingeben wird POKE 44,32:POKE 8192,0:NEW eingetippt. Ist nun das Maschinen-

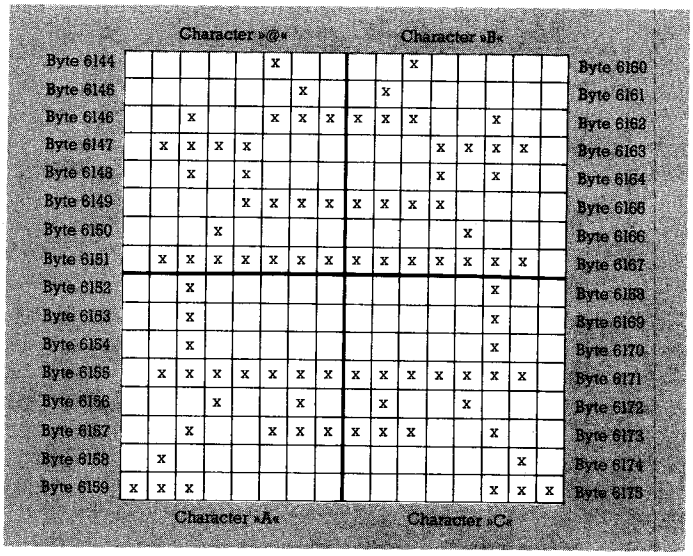

Bild 1. Die Speicheraufteilung in der Grundmatrix des Sprites Nummer 0. Das höchstwertige Bit eines Bytes steht links.

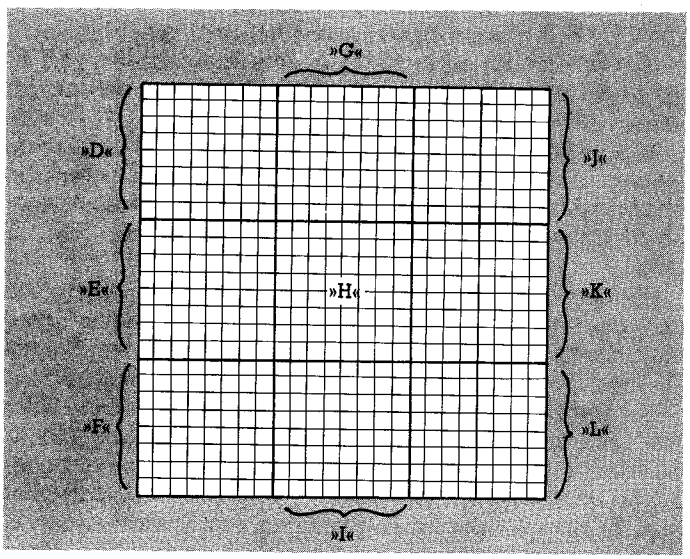

Bild 2. Die Undefinier-Matrix hat 24 x 24 Punkte und ist notwendig, um Sprites auch punktweise verschieben zu können

spracheprogramm im Speicher, kann es mit einem Monitorprogramm auch noch einmal abgespeichert werden. Später muß man es nur noch mit LOAD »name«, 1,1 laden.

Die Bedienung:
Sind die Sprites definiert, muß dem Maschinenspracheprogramm mitgeteilt werden, wo der Zeichensatz liegt, den es verwalten soll. Das geschieht mit den Befehlen POKE 677, Lowbyte:POKE 678, Highbyte, in unserem Fall also POKE 677,0:POKE 678,20, da der Zeichensatz auf der Adresse 5120 beginnt.

Soll nun ein Sprite auf den Bildschirm, muß zuerst einmal in Adresse 683 die Spritenummer gePOKEt werden (Achtung, keine Zahl über 8 angeben, da sich das Programm dann selbst zerstören könnte). Schließlich werden in Adresse 673 die X-Koordinate (maximal 159) und 674 die Y-Koordinate (maximal 167) gesetzt. Dann kann das Programm mit SYS 8021 sofort aufgerufen und auf dem Bildschirm das Sprite betrachtet werden, vorausgesetzt man hat vorher mit POKE 36869,205 auf den freidefinierbaren Zeichensatz geschaltet.

Wird nun das Sprite auf eine andere Position gesetzt, so verschwindet es vollständig von der alten Position, und die Zeichen, die auf diesem Platz waren, erscheinen wieder mit ihrer alten Farbe. Will man aber das Sprite ganz vom Bildschirm löschen, POKEt man wieder in 683 die Spritenummer und ruft das Maschinenspracheprogramm diesmal mit SYS 8099 auf. PRINT »CLR/HOME« sollte man nicht verwenden, da im SCB noch die alten Bildschirmzeichen gespeichert sind und beim nächsten Setzen wieder auf ihren alten Plätzen auf dem Bildschirm erscheinen würden.

Der Sprite-Generator

Nun zum Programm »Sprite-Definer« (Listing 2).

Dieses Programm ist ein Sprite-Generator in Basic, der bei der Erstellung von Sprites recht hilfreich sein kann. Das Programm verdeutlicht auch, wie die Definition der Sprites und die Bedienung des Maschinenspracheprogramms erfolgt.

Obwohl sich das Programm fast von selbst erklärt, hier doch einige kurze Erläuterungen:

Startet man das Programm mit RUN, erscheint als erstes die Begrüßung und die Aufforderung »Bitte warten!«. Das Programm kopiert nämlich jetzt den Zeichensatz aus dem ROM ins RAM, was in Basic naturgemäß etwas dauert.

Jedesmal, wenn man in einem Menüteil eine Eingabe gemacht hat, wird man »Richtig?« gefragt. Tippt man hier für N (Nein), so kann die Eingabe wiederholt werden. Drückt man aber den Linkspfeil, so kommt man wieder ins Hauptmenü.

Die Tastenbelegung im Editiermodus:

Cursor-Tasten	**Cursor-Bewegungen**
Leertaste	**Punkt setzen**
Delete-Taste	**Punkt löschen**
CLR/HOME	**Gitter löschen**
RETURN	**Modus beenden mit Änderung des Sprites, vorher aber Abfrage**
Linkspfeil	**Modus beenden, aber ohne Änderung des Sprites**
I-Taste	**Sprite invertieren**

Bei der Funktion »Weiter« kommt man in ein zweites Menü, das weitere Funktionen zur Verfügung stellt. Aus diesem Menü gelangt man mit »zurück« wieder ins Hauptmenü. Beim Speichern werden die Sprites als reiner Speicherauszug auf Kassette gebracht, so daß das Laden im Prinzip auch mit LOAD »name«,1,1 möglich ist.

Sicherlich kann das Maschinenspracheprogramm noch weiter verbessert werden. So wäre zum Beispiel eine Spritesteuerung per Interrupt durchaus denkbar. Leider funktioniert das Maschinenprogramm nicht mit den üblichen Grafikmodulen, da diese den Bildschirminhalt auch mit dem freidefinierbaren Zeichensatz aufbauen. Sollen Sprites auch miteinander oder übereinander dargestellt werden, dann muß das Setzen und Löschen nach folgender Reihenfolge durchgeführt werden, da es sonst zu Schwierigkeiten mit dem SCB kommen kann:
Sprite 0 setzen, Sprite 1 setzen,..., Sprite n setzen. Hiernach die Berechnungen für die neuen Positionen durchführen.
Sprite n löschen,.Sprite n-1 löschen,..., Sprite 0 löschen. Danach Vorgang von oben wiederholen.

Noch eins zum »Sprite-Definer«: Die erste REM-Zeile muß auf jeden Fall mit 16 Sternchen eingegeben werden, da sich das Programm später mit POKEs selbst verändert und andernfalls, wäre die REM-Zeile kürzer, die folgende Zeile in Mitleidenschaft ziehen würde. Doch nun wünsche ich allen, die das Programm eintippen, viel Spaß und vielleicht ein bißchen C 64-Feeling.

(Markus Leberecht/ev)

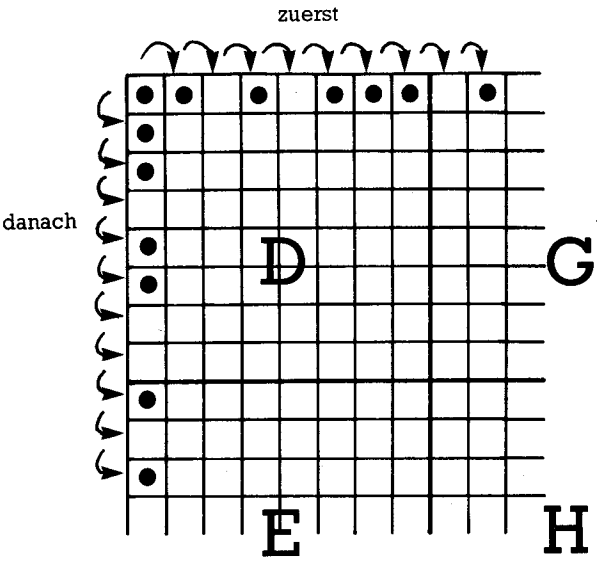

Bild 4. Zur punktgenauen Justierung wird das Sprite nach dem Kopieren in die Umdefinier-Matrix noch horizontal und vertikal verschoben

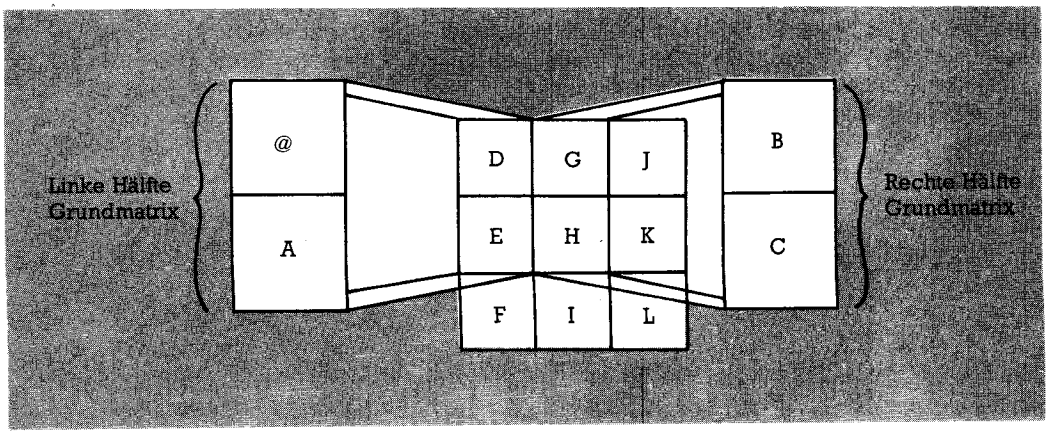

Bild 3. Hineinkopieren der Grundmatrix in die Umdefinier-Matrix