Profi-Sprites

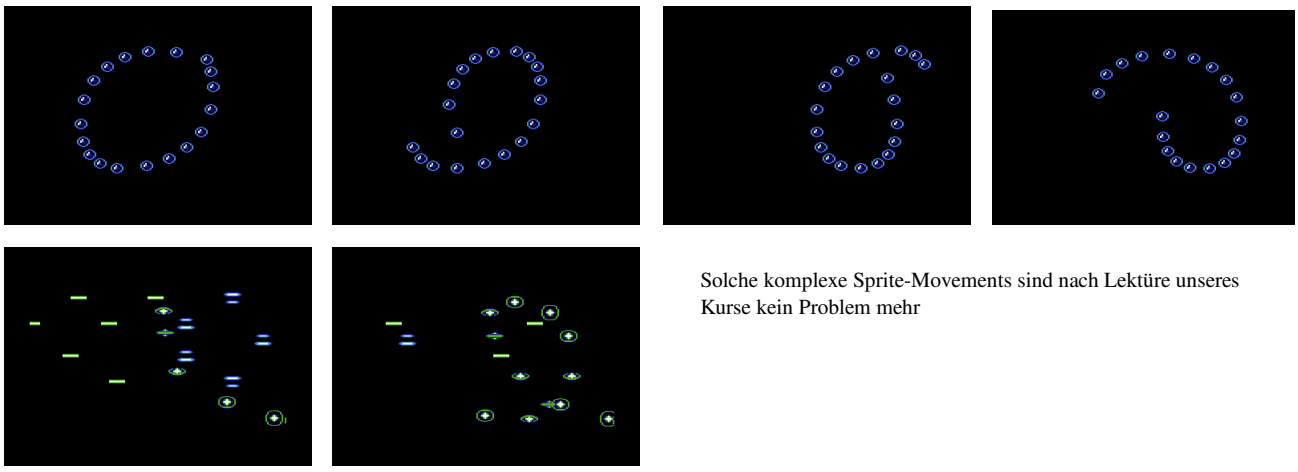
Mit unserem Sprite-Kurs werden Sie zum König unter den Multiplexern. Sie programmieren eine universell einsetzbare Sprite-Routine; bei der großer Wert auf Flexibilität und Schnelligkeit gelegt wurde. Selbst komplizierteste Sprite-Movements sind ein Kinderspiel.

von Ingo Kusch

Wer das Spiel “EON" kennt, hat sicher schon einmal die fantastischen Sprite-Routinen dieses Shoot'em-Up-Games bewundert. Der Autor zeichnet übrigens auch für unseren Kurs verantwortlich. Eines gleich noch vorweg: der Mini-Kurs beschreibt eine sehr leistungsfähige Multiplexer-Routine, die für Profis keine Schwierigkeit ist. Neueinsteiger, die sich mit den Themen Assembler und Sprites noch nicht hundertprozentig auskennen, sollten sich zuerst die Grundlagen aneignen und erst dann den Kurs an gehen.

Die MOVE-Routine

Die MOVE-Routine besitzt eine Reihe von Grundfunktionen, die für umfangreiche Sprite-Bewegungsabläufen (kurz: MVE) unerläßlich sind. Hierzu gehört zunächst die Animation von Sprites. Der Großteil aller auf den Bildschirm gebrachten Sprites ist in irgendeiner Form animiert. Hierzu nutzt MOVE die Register für jedes Sprite, die die Animation übernehmen. Im Register “ANITAB” befindet sich die Nummer der durchzuführenden Animation. Diese Nummer zeigt auf eine Tabelle: ‘•ANITAB”, die die Adressen der jeweiligen Animationstabellen und die Geschwindigkeiten enthält, mit denen diese ablaufen sollen. Die Animationstabelle, deren Adresse in ANITAB zu finden ist, enthält nun die Blocknummern der einzelnen Animationsschritte und am Ende als Pointer eine “0". MOVE holt sich nun mit Hilfe der Animationsnummer die Animationsgeschwindigkeit aus ANITAB und die erste Block-Nummer aus der Animationstabelle. Nach der Pause holt sie sich die nächste Block-Nummer usw. bis sie auf ein Nullbyte stößt. Sobald es gefunden ist, beginnt die Animation wieder von vorne, sofern in der Animationsnummer nicht das Bit S40 gesetzt war. Da jeder Eintrag in der besagten ANITAB vier Bytes enthält, kann es also bei einer 8-Bit-Nummer nur 64 verschieden aufgeführte Animationstabellen geben (64*4=256). Daher wurde das S40 Bit als Controlbit für einmalig ablaufende Animationen eingeführt. Ist dieses Bit in der Animationsnummer gesetzt, überprüft MOVE am Ende dieser Animation, ob es sich um eine Explosion handelte. die als "bevorzugte" Animation die Position "0" in der ANITAB reserviert bekam. War es eine Explosion. wird das Sprite gelöscht. War es dagegen keine, wird der nächste Bewegungsablauf initialisiert (s. MINIT)
Die beiden weiteren Register übernehmen zum einen eine Pointerfunktion (ANICO) und zum anderen eine Zählerfunktion (ANITCO). Die nächste Routine übernimmt die linearen Bewegungen in X/Y-Richtungen. Sie verwendet die Register DELTAX und DELTAY, welche, sofern sie ungleich null sind, den Wert der jeweiligen Schrittweite in X/Y-Richtung enthalten. Dieser Wert besteht aus zwei Hälften, die jede für sich auf eine Tabelle: "DELTB” weisen, welche die tatsächlichen Additionswerte enthält. Handelt es sich dabei um Halfbytes zwischen $01 - $07, findet man in der DELTB größer werdende positive Schrittweiten vor. Sind es hingegen Haltbytes zwischen $09 - $0f, so findet man größer werdende negative Schrittweiten. Für die Halfbytes $00 - $08 findet man jeweils den Wert 0: Vor jedem neuen Einsprung in die MOVE-Routine wird ein Register gesetzt, das abwechselnd mal das obere, mal das untere Haltbyte für das MVE heranzieht. Dies hat den Vorteil, das ein Sprite auch mit 3.5 Pixeln pro Screendurchlauf bewegt werden kann! Nämlich einmal 3 Pixel und beim nächsten mal 4 Pixel, dann wieder 3 Pixel usw.. Die beschleunigten Bewegungen benötigen bereits sechs Register um alle Funktionen zu erfüllen: Hierbei wollen wir zunächst die X-Komponente betrachten. Befindet sich im ACCTX ein Wert ungleich null, wird eine Beschleunigung vorausgesetzt. Jedes mal wenn das Verzögerungsregister ACCT1 gleich null ist. wird es wieder restauriert und die eigentliche Beschleunigung im Register DELTAX durchgeführt, das im Falle einer Beschleunigung nur noch für diese reserviert ist und nicht mehr für die gewöhnliche lineare MVEs.
DELTAX enthält hierbei als oberes Halfbyte das Ziel, das die Beschleunigung erreichen soll und im unteren Halfbyte den Momentan- (Start-)wert, der bereits erreicht ist: Sobald diese beiden Werte identisch sind, ist die Beschleunigung beendet und das Sprite bewegt sich mit der zuletzt erreichten, konstanten Geschwindigkeit vorwärts. Analog dazu gilt das gleiche für eine beschleunigte Bewegung in Y- Richtung; wobei die Register hier ACCTY, ACCT2 und DELTAY heißen. Eine weitere Option bietet das DELAY-Register, das eine Art Countdown durchführt, sobald man einen Wert ungleich null hineinschreibt. Wird nach der gewünschten Zeit wieder der Wert Null erreicht, gibt MOVE das Kommando den nächsten Bewegungsablauf zu initialisieren. Gerade DELAY bietet hierdurch unzählige Möglichkeiten MVEs exakt und einfach abzustimmen.



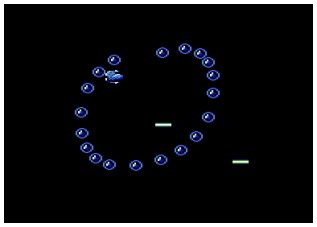
Solche komplexe Sprite-Movements sind nach Lektüre unseres Kurse kein Problem mehr

Sprites sind das A und O von Spielen.

Sehr schöne Anwendungen bietet die Sinus-Funktion. Mit ihr lassen sich alle möglichen Ellipsen, Kreise, Kreisabschnitte, Eiformen und sonstiges auf Basis einer Sinus-Tabelle erzeugen. Die Ansteuerung ist einfach. Die Register SINXLO/SINXHI enthalten die Adresse, an der die Sinustabelle zu finden ist. SINX1 enthält den Pointer, SINX2 den maximal anzunehmenden Wert und SINX3 entsprechend das Minimum, das nach Erreichen des Maximums angenommen wird. Hieraus ergibt sich, daß eine Sinustabelle, die analog zu den gewöhnlichen DELTAX/ DELTAY-Werten aus Halfbytes besteht, nur maximal 256 Bytes lang sein darf. In der Praxis ist das allerdings kein Problem, da sie in aller Regel nicht annähernd so lang sind: Die Entscheidung, ob die obere oder untere Hälfte des jeweiligen Halfbytes verwendet wird, fällt bei jedem Sprite individuell im Bit $10 des CONTROl-Registers, das automatisch umgeschaltet wird. Aus dem Zusammenspiel der X- und Y-Komponente des jeweils festgelegten Sinus ergibt sich später, welche Figur durchflogen wird. Sind z.B. beide Komponenten aus einer Sinustabelle und 90 Grad versetzt, ergibt sich ein Kreis. Sind sie um 120 Grad versetzt, ergibt sich eine diagonale Ellipse. Verwendet man unterschiedliche Sinustabellen, kommen Ellipsen dabei heraus (bei gleicher Größe der Sinustabelle, hier ist nur der Radius unterschiedlich) oder auch völlig Unvorhergesehenes (bei unterschiedlicher Größe der Sinustabellen). Die interessanteste Option bieten ohne Zweifel die Register SPECLO & SPECHI. Befindet sich in diesen Registern die Startadresse eines beliebigen Programmes, springt MOVE dorthin und arbeitet diese Routine ab. Hier ist schlicht alles möglich: Ob es sich um Routinen handelt, die von Zeit zu Zeit einen Schuß initialisieren, Sterne über den Screen fliegen lassen oder eine Rakete individuell ins Ziel steuern, alles ist machbar, sofern es in der zur Verfügung stehenden Zeit durchgeführt werden kann. MOVE erledigt übrigens auch die für den Multiplexer notwendigen Sortieralgorithmen. Wer sich insbesondere für Einzelheiten über den Sortieralgorithmus, Multiplexer usw. interessiert, kann sich gerne an den Autor wenden.

Die MINIT-Routine

Diese Routine übernimmt die gesamte aufwendige Initialisierung der einzelnen MVE-Steps mit Hilfe sogenannter Tracks, die das jeweils komplette MVE enthalten. Aktiviert wird MINIT durch Setzen der gewünschten Tracknummer im INITTB-Register des ausgewählten Sprites. Die Tracknummer dient als Pointer auf die "TRKTAB", die die Adresse des jeweiligen Trackbeginns enthält. Wird ein Track neu initialisiert, d.h. ist das Sprite bei Beginn des INIT noch nicht eingeschaltet, erwartet MINIT zunächst eine Positionsangabe in der Form **(X-Pos/2), (Y-Pos)**
Die X-Position wird durch zwei geteilt, damit man das MSB nicht extra mitschleppen muß. Ist hierbei die X-Position gleich null, wird angenommen, daß das Sprite bereits positioniert ist (etwa durch MORE/s.u.) und überspringt diese ersten beiden Bytes, die übrigens nie fehlen dürfen. Daraufhin erwartet MINIT zwingend ein Statusbyte, das den weiteren INIT steuert. War das Sprite zu Beginn des INlTs bereits eingeschaltet, nimmt MINIT an, daß sich das Sprite innerhalb eines noch nicht beendeten Tracks befindet und versucht diesen fortzuführen. Die Register TRKLO. TRKHI und TRKPO müssen auf dieses Statusbyte zeigen. Stößt MINIT dabei auf ein Statusbyte gleich Null, wird die Initialisierung beendet.



Die MORE-Routine lechzt förmlich nach Erläuterung.
Das Statusbyte von MINIT wird von rechts nach links analysiert, das bedeutet:
Die Daten werden in der gleichen Reihenfolge wie die Bits erwartet. Also zuerst die Daten von Bit $01 (wenn es gleich 1 ist), dann die Daten von Bit $02 usw. bis man zuletzt die Daten von Bit %80 folgen läßt, die der Nummer des gewünschten Sinus entsprechen würden (sofern es gleich 1 ist) . Hat man dieses recht simple System einmal begriffen, ist es leicht mit Hilfe der unzähligen Kontrollmechanismen (DELAY, ANIT8 usw.) die kompliziertesten MVEs anzulegen. Bleibt noch, die Syntax der erwähnten Tabellen zu erläutern. An den Punkten, wo die vollständige Parameterübergabe durch die Tracktabelle zu umfangreich ist, weicht MINIT auf Extratabellen aus, die die notwendigen Daten enthalten, im Track aber nur als Byte-Pointer erwähnt werden müssen. Hierzu zählen die ACCTAB. ANITAB, SINTAB2 und SINTAB. Die Pointer des Tracks weisen hierbei stets auf einen gesamten Eintrag, der in jedem Fall vier Bytes enthält.

ACCTAB enthält alle Daten für eine aus X- und Y-Komponenten bestehende beschleunigte Bewegung:
1. Byte: Oberes Nibble Ziel, unteres Nibble Start der Beschleunigung in X-Richtung (s. MOVE)
2. Byte: Pause zwischen den einzelnen Schritten.
3. Byte: Ziel/Start für Y-Beschleunigung.
4. Byte: Pause für Y-Schritte.

ANITAB enthält die Animationsdaten:
1. und 2.Byte: Adresse der Animationstabeile.
3. Byte: Pause zwischen den Animationsschritten.
4. Byte: unbenutzt.

SINTAB2 enthält die Daten für den Start einer Sinusbewegung.
1. Byte: Nummer des Sinus in X-Richtung (dient als Zeiger auf SINTAB)
2. Byte: Beginn des X-Sinus
3. Byte: Nummer des Y-Sinus (ebenso Zeiger auf SINTAB)
4. Byte: Beginn des Y-Sinus

SINTAB enthält zuletzt die Daten die eine Sinustabelle charakterisieren:
1. und 2. Byte: Adresse der Sinustabelle
3. Byte: Ende der Tabelle
4. Byte: Beginn der Tabelle (häufig einfach Null)

Während der ersten Versuche noch nicht notwendig, später hauptsächlich zur Speicherersparnis gedacht, wurden die JUMP- und EINSCHUB-Routinen in MINIT implementiert. Aufgerufen werden beide dadurch, daß MINIT ein Statusbyte $FF findet. Ist das darauffolgende Byte auch $FF, wird ein JUMP ausgeführt. Die JUMP-Routine holt sich aus der Tracktabelle die nächsten zwei Bytes und verwendet diese als Vektor in eine andere Tracktabelle, wobei ganz wichtig ist, daß dieser Vektor auch auf ein Statusbyte zielt (sinnvoll ist es in jedem Fall, die Statusbytes nur in binärer Form darzustellen). EINSCHUB wird durch ein dem $FF folgenden $FE aktiviert, holt sich die nächsten beiden Bytes als Zeiger auf einen anderen Track, führt die dort gefundene Sequenz (Statusbyte und danach die entsprechenden Daten) aus und kehrt in den alten Track zurück.

Special-init-Routinen

Als erster Punkt ist zu bemerken, daß bei erstmaligem INIT (Sprite beim Beginn von MINIT aus!) das A-Register beim Einsprung in das Special-Init-Programm die Nummer des Tracks, der gerade initialisiert werden soll, enthält. Hierdurch lassen sich z.B. Sprites, die unterschiedlich positioniert werden müssen, einfach mehrfach in der TRKTAB untereinander aufführen und mit einem Special-Init-Programm anschließend je nach Nummer des aufgerufenen Tracks einfach die entsprechende Position aus einer Tabelle holen, anstatt den gleichen Track mit unterschiedlichen Startpositionen zigmal untereinander zu kopieren (Speicherplatzverschwendung)! Um die vielfältigen Möglichkeiten anzudeuten, sind drei verschiedene MORE-Initialisierungsprogramme angeführt, die jedes für sich völlig andere Aufgaben zu erfüllen haben. Bei der Erstellung von MORE stand die Darstellung von großen Gegnern im Vordergrund. Dabei stellt sich das Problem, die Sprites relativ zu einem beliebigen ersten Sprite zu positionieren, so daß alles zu einer Einheit verschmilzt. Weiterhin mußten alle diese Sprites nachvollziehbar miteinander in Kontakt stehen, damit sie später gemeinsam zerstört werden können.

Die MORE-Routine liefert beides. In der Tracktabelle übergibt man zusammen mit der Einsprungsadresse den Zeiger auf die MORETB, in der wiederum die Adressen der MOREtracks stehen. Aus diesem MOREtrack holt sich MORE nun die Nummer der Tracks, die die neuen Sprites haben sollen, sowie die Positionen relativ zum Anfangssprite. Ebenso wird im ZWISCH-Register die Nummer des Ausgangssprites vermerkt und im ZWISCH- Register die Nummer des Vorgängersprites, das zuvor initialisiert wurde. Hiermit kann man im Falle einer gemeinsamen Vernichtung von hinten nach vorne alles zerstören.

Die MORE2-Routine war hauptsächlich für Einzelinitialisierungen wie Schüsse, Bomben u.a. gedacht. Sie holt sich aus der Tracktabelle nur die Nummer des zu startenden Tracks und gibt ihn an die MINIT-Routine weiter. Nachdem sich die Geschwaderinitialisierung durch MORE2 und MORE als zu umständlich und zu speicherintensiv erwies, wurde

die MORE3-Routine geboren: die wird als Special-Init-Programm aufgerufen und übernimmt dann als Specialprogramm von MOVE die Neuinitialisierungen. Man übergibt dabei in der Tracktabelle zusätzlich zur Einsprungadresse noch drei Bytes: Pausendauer, Anzahl der Inits, Nummer des Tracks. Den Rest nimmt Ihnen die Routine ab.

Alle diese Beispiele sollen nur ein Anreiz sein, selbständig über mögliche neue Anwendungen nachzudenken und diese auszuarbeiten. Die Routinen geben Ihnen das Gerüst für eigenständige, kreative Arbeit im Umgang mit Sprites.