Vorwort

Lieber Leser,

zunächst einmal gratuliere ich Ihnen zum Kauf dieses Buches. Wie schon der Titel "Gamedesigner's Workshop" aussagt, beschäftigt sich dieses Buch mit der Entwicklung und Programmierung von Spielen auf dem Atari 8-Bit Computer. Wenn Sie dieses Buch aufmerksam durcharbeiten, werden Sie am Ende in der Lage sein, selbst Spiele auf Ihrem Atari zu programmieren. Natürlich will ich Sie nicht nur mit trockener Theorie überschütten, sondern auch praktische Beispiele geben. So finden Sie zusätzlich noch einige interessante Spiele, deren Aufbau ausführlich erläutert wird. Um Ihnen das Entwickeln eigener Spiele zu vereinfachen, finden Sie außerdem einige Editor-Programme, mit denen sich Ihre Ideen besser verwirklichen lassen.

Ihnen sollte natürlich klar sein, daß ich in diesem Buch nicht alle Grundlagen der Programmierung in Basic vermitteln kann. Diese sollten Sie bereits beherrschen. Auch die Verwendung der zusätzlichen Turbobasic-Befehle sollte Ihnen bekannt sein. Hierzu befinden sich am Ende des Buches Literaturhinweise.

Sollten Sie trotz der Lektüre noch spezielle Fragen zur Spieleprogrammierung haben, stehe ich Ihnen gerne schriftlich (siehe Impressum) oder telefonisch unter 06181/87539 zur Verfügung.

Ich wünsche Ihnen viel Spaß beim Lesen und viel Erfolg mit Ihren eigenen Entwicklungen.

Ihr Kemal Ezcan

INHALT

INHALT

Was ist ein Computerspiel?

Bevor Sie sich an die Entwicklung eigener Spiele machen, sollten Sie sich zunächst Gedanken darüber machen, was denn ein Computerspiel eigentlich ist. Wenn Sie sich verschiedene Computerspiele ansehen, sei es nun Pac-Man, Donkey Kong, oder, um etwas mehr in unseren Bereich zu gehen, Zador, Tecno Ninja oder Zebu-Land, fällt auf, daß sie alle eine Gemeinsamkeit haben: Der Spieler muß versuchen, eine bestimmte Aufgabe zu lösen, wobei er Probleme verschiedenster Art bewältigen muß. Ein solches Problem könnte z.B. ein Zeitlimit, ein gefährliches Objekt (z.B. ein Monster, feindliches Raumschiff etc.), ein logisches Problem (Puzzle) oder ein Geschicklichkeitsproblem sein. Schafft der Spieler es, das Problem zu bewältigen, hat er gewonnen - zumindest für kurze Zeit, denn Computerspiele haben es so an sich, daß nach einer geschafften Aufgabe die nächste, schwierigere Aufgabe folgt. Ob nun das Monster gefährlicher, das Raumschiff schneller, die Zeit kürzer oder das Puzzle kopmplizierter wird, ist egal. Hauptsache, der Spieler sieht sich immer wieder neuen Herausforderungen entgegengestellt. Nur so kann ein Computerspiel auch über längeren Zeitraum hinweg Spaß bereiten, und das ist doch die Aufgabe eines Computerspieles, oder?

Computerspielogie

Was ist denn das?

Die Frage ist berechtigt, denn diesen Begriff gibt es gar nicht. Er steht hier für die etwas längere Überschrift "Verschiedene Arten der Computerspiele". Wie Sie sicherlich wissen, gibt es bereits eine Menge verschiedener Computerspiele. Wenn man sich diese Spiele aber genauer betrachtet, fällt auf, daß die meisten Spiele einigen wenigen grundsätzlichen Ideen zugrunde liegen. Diese Unterscheidung ist sehr wichtig, denn ein selbstentwickeltes Spiel sollte eine klare Idee haben. Hier also ein paar der häufigsten Arten von Computerspielen:

Actionspiele, teilsweise auch Ballerspiele genannt, ist die Art von Spielen, in denen man eine Figur, meist ein Raumschiff, steuert, mit dem man verschiedene andere Figuren, also die Feinde, abschießen muß. Das erste Computerspiel dieser Art war das bekannte Space Invaders. Dies wurde dann abgelöst durch Galaxian, daraus wurde dann Galaga usw. So nach und nach wurden die Feinde immer beweglicher, und inzwischen ist man bereits so weit, sich mitten im Kampfgeschehen zu befinden. Ein sehr gutes Beispiel dafür ist der Atari Klassiker Star Raiders.

Geschicklichkeitsspiele, meist in der Form von Leiter- oder Climb & Jump, Jump & Run usw. wie z.B. das legendäre Donkey Kong gibt es in sehr vielen Variationen. Bei dieser Art von Spiel sieht man das Spielgeschehen hauptsächlich von der Seite, d.h. was auf dem Bildschirm oben ist, soll auch in Wirklichkeit oben sein. Aus diesem Grunde unterliegen die Figuren in solchen Spielen meist den Schwerkraftgesetzen, woraus auch der Name folgt: Um nach oben zu kommen, muß man entweder klettern oder springen. Aufgabe bei einem Climb & Jump Spiel ist es entweder, einen bestimmten Punkt des Bildschirmes zu erreichen (z.B. den oberen oder rechten Rand), oder, bzw. und gewisse Gegenstände einzusammeln.

Eine andere Form der Geschicklichkeitsspiele sind die Labyrinthspiele. Das Spielgeschehen ist hier von oben zu sehen. Die Figuren in diesen Spielen werden allerdings nicht immer von oben dargestellt, da dies grafisch nicht ansprechend genug wäre, sondern sind von der Seite zu sehen. In der Realität würde das bedeuten, daß die Figuren auf dem Boden entlangrutschen, im Spiel sieht es aber realistisch und seltsamerweise nicht unlogisch aus. Auch bei den Labyrinthspielen muß man Gegenstände einsammeln. Ob dies nun Punkte (Pac Man) oder Goldstücke sind, hängt natürlich vom Spiel ab.

EINFÜHRUNG

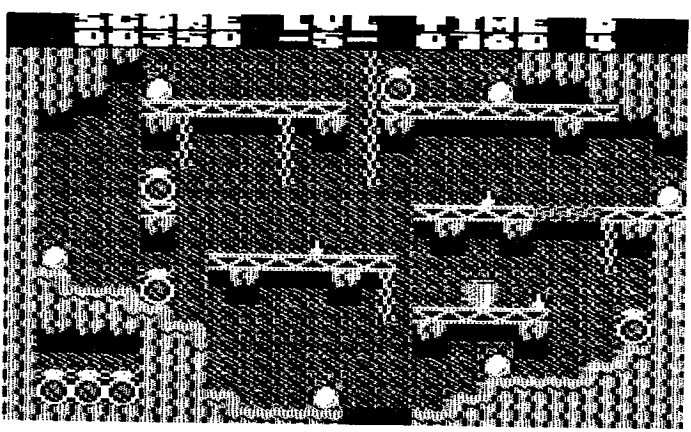
Vor einigen Jahren tauchte aus Russland eine neue Form der Spiele auf: Tetris. Dieses neuartige Spiel war das erste Glied in der heute sehr beliebten Kette der Denkspiele. Hier handelt es sich um eine Spielegattung, bei der es hauptsächlich darauf ankommt, eine Aufgabe durch logisches Denken möglichst schnell zu lösen. Ob es nun darauf ankommt, verschiedene Steine zusammenzusetzen, Formen zu ordnen oder Paare zu finden, die Denk- oder auch Puzzlespiele erfreuen sich immer größerer Beliebtheit. Das mag vielleicht daran liegen, daß diese Art von Spielen dem oftmals gefällten Urteil, Computerspiele verdummen, erfolgreich widerspricht. Gerade die Intelligenz wird bei diesen Spielen geschult, und so ist es nicht verwunderlich, daß immer mehr Erwachsene Gefallen an Denkspielen finden.

Ein uraltes Spielprinzip, das aber immer wieder Gefallen findet, sind die Strategiespiele. Bereits auf den PET-Computern von Commodore gab es ein Basic-Programm mit dem Namen Hamurabi. Man mußte durch Anbauen von Korn und entsprechender Verteilung dafür sorgen, daß das Volk zufrieden war. Nichts anderes finden wir heute in Spielen wir Kaiser II, natürlich in erweiterter Form. Neben diesen Handelsspielen gibt es auch Kriegsspiele, die eine Szenerie simulieren, die gespielt werden soll. Bekannt sind hier die SSI-Renner War In Russia und Battalion Commander. Im Gegensatz zu den bisher genannten Spielgenres läuft das Geschehen bei den Strategiespielen nicht in Echtzeit ab, sondern es werden Eingaben gemacht, deren Ergebnisse daraufhin angezeigt werden. Diese Spielgattung schließt also den Faktor Geschicklichkeit vollkommen aus. Umsetzungen von Brettspielen werden auch in diese Kategorie eingeordnet, denn vielen Brettspielen liegt eine kriegerische Handlung zugrunde. Bei Schach geht es schließlich auch darum, den Gegner durch strategisch geschicktes Handeln zu vernichten.

Zuguterletzt gibt es noch die sehr beliebten Simulationen, d.h. Spiele, die einen echten Vorgang, wie z.B. ein U-Boot (Silent Service) ein Flugzeug (F-15 Strike Eagle) oder ein Atomkraftwerk (Scram) simulieren. Hier kommt es darauf an, möglichst genau die Vorgänge der Realität nachzuahmen. Aufgabe des Spielers ist es dann, sich mit der gestellten Situation zurechtzufinden, also das U-Boot im Kampf zu lenken. Manche Simulationen sind sogar so realistisch, daß sie bei Behörden als Training für echte Situationen eingesetzt werden. Je realistischer die Simulation sein soll, dest schwieriger wird es natürlich, sie zu programmieren.

Der Vollständigkeit halber wollen wir auch noch die Adventures und Rollenspiele erwähnen. Diese Spielegattung ist allerdings ein Kapitel für sich, über das bereits viele andere Bücher sowie Serien, z.B. im ZONG-Magazin, existieren.

Interessant ist, daß die verschiedenen Spielegattungen nicht nur in ihrer reinen Form vorkommen, sondern beliebig gemischt werden können. So gibt es z.B. Actionadventures, Adventures und Rollenspiele mit Geschicklichkeitseinlagen, Strategiespiele mit Actionelementen usw. Der Fantasie sind hierbei keine Grenzen gesetzt. Wichtig ist nur, daß das fertige Spiel Spaß macht (wie immer).



Features

Unter Features versteht man die Eigenschaften, die ein Spiel aufzuweisen hat. Hiermit sind allerdings nicht nur die Details wie z.B. eine Highscoreliste gemeint, sondern auch die grundsätzlichen Eigenschaften.

Den Anfang macht die Grafik. Als Grafik bezeichnen wir bei einem Spiel das, was wir auf dem Bildschirm sehen. Wichtig zu wissen ist, daß eine gute Grafik noch lange kein gutes Spiel ausmacht! Nur wenn das Konzept stimmt, nützt die gute Grafik etwas. Diese Tatsache wird leider von vielen Spielemachern verkannt. Um gleich einige Begriffe, die später verwendet werden, zu erläutern: Als Playfield (dts. Spielfeld) bezeichnen wir das, was im Hintergrund ist. Dies kann also das Labyrinth sein, die Landschaft oder der Weltraum. Als Objekte oder Figuren bezeichnen wir die beweglichen Teile, also den eigenen Held, die Feinde usw.

Als zweites wäre der Sound zu nennen. Dieser ist aufzuteilen in Soundeffekte, also Geräusche, die das Geschehen macht, und Musik. Hier ist wichtig zu wissen, daß Musik zwar eine nette Untermalung ist, aber in keinem Fall die Geräusche ersetzen kann! Eine Aktion wirkt nur dann real, wenn sie auch die entsprechenden Geräusche von sich gibt.

Ein sehr wichtiger Punkt ist die Freundlichkeit eines Spieles. Ist ein Spiel zu leicht, macht es nach kurzer Zeit keinen Spaß mehr. Ist es zu schwer, ist man bald frustriert und gibt auf. Ein Spiel sollte dem Spieler immer wieder Erfolgserlebnisse und möglichst keinen Frust bringen. Um dies zu erreichen, sollte man sein Spiel möglichst anderen Leuten zum Ausprobieren vorlegen, denn sein eigenes Spiel beherrscht man meist besser als jeder andere.

Damit ein Spiel auch nach längerer Zeit noch Spaß macht, sollte es möglichst verschiedene Elemente oder Spielebenen (Levels) enthalten. Wer immer wieder das gleiche Labyrinth zu meistern hat, kennt es bald auswendig und hat keine Lust mehr dazu. Die Anzahl der Levels hängt natürlich davon ab, wie schnell diese zu meistern sind.

Last but not least sind natürlich die wichtigen Kleinigkeiten zu nennen: Eine Levelanwahl oder verschiedene Schwierigkeitsgrade, ein Zweispielermodus, Codewörter, eine Highscoreliste, ein Demomodus usw.

Welche der genannten Features enthalten sind, und in welcher Form diese vorliegen, ist einfach eine Frage der Zeit, die man in die Entwicklung investieren will, und wieviel Speicherplatz man zur Verfügung hat.

Grundsätzlich sollte man natürlich möglichst speichersparend programmieren, um wenigstens die Möglichkeit zu haben, weitere Features einzubinden.

Die Idee

Am Anfang eines neuen Spieles steht zunächst einmal die Idee. Ob es sich nun um eine völlig neue Idee oder ein bereits dagewesene handelt, ist eigentlich unwichtig. Wenn eine bereits existierende Idee abgewandelt wird, sollten Sie allerdings darauf achten, eine gewisse Eigenständigkeit mit einzubinden, denn die hundertste Pac-Man Version ohne irgendwelche Neuheiten interessiert niemanden. Mit ein paar kleinen Neuerungen, z.B. Teleportierfelder, Schlüssel und Türen oder ähnliches, gewinnt das Spiel wieder an Reiz. Aber nun von vorne: Notieren Sie sich alles, was Ihnen zu Ihrer Idee einfällt. Im einzelnen können das folgende Punkte sein:

Eine Vorgeschichte, wie es zu dem Spiel kam. Eine interessante Story, die das Spiel begründet, verleiht einem Spiel immer ein gewisses Etwas. Wenn man weiß, warum das Spiel so ist, wie es ist, kann man sich besser hineinversetzen. Dadurch hat man mehr Spaß am Spiel.

Machen Sie sich Skizzen, wie der Bildschirm aufgebaut sein soll, z.B. oben das Spielfeld, unten die Anzeigen für Punkte, Leben usw. Beim Entwerfen der Grafik kann dies natürlich noch abgewandelt werden.

Wichtig ist natürlich die Aufgabe des Spielers. Notieren Sie, was der Spieler tut, wie sich die Spielfigur bewegt, welche Schwierigkeiten es gibt usw. Auch hier besteht natürlich die Möglichkeit, später noch Änderungen anzubringen. Je genauer allerdings vorher das Konzept feststeht, desto einfacher wird es, dieses Konzept in ein Programm umzusetzen. Jede Änderung, die später gemacht wird, verkompliziert das Programm, wodurch es leicht unübersichtlich wird.

Feinde, Schwierigkeiten, Monster und alles, was den Spieler daran hindert, seine Aufgabe zu bewältigen, sollten notiert werden. Ebenso die verschiedenen Level oder Schwierigkeitsgrade. Verschiedene Level können sich sowohl von ihrer Grafik als auch von ihrem Aufbau oder sogar der zu lösenden Aufgabe unterscheiden. Dies bringt natürlich die meiste Abwechslung, bedeutet aber gleichzeitig einen großen Programmieraufwand. Ein gutes Beispiel dafür ist der Hit Necromancer, hier muß man in jedem der vier Level eine ganz neue Aufgabe erfüllen.

Die Umsetzung

Um die fertige Idee nun in ein Programm umzusetzen, müssen wir uns zunächst einmal für eine Programmiersprache entscheiden. Entscheidend bei der Wahl der Sprache ist als erstes, daß wir sie beherrschen müssen und zweitens die Geschwindigkeit, bzw. die Möglichkeiten, die wir mit der Sprache haben. Als Möglichkeiten stehen uns z.B. Basic, Turbobasic XL, Action, Quick und Assembler zur Verfügung. Von Basic sollte man grundsätzlich absehen, da es doch sehr langsam ist und außerdem sehr gut durch das Turbobasic XL ersetzt werden kann. Turbobasic XL ist schon schneller, bietet einen besseren Komfort, läßt sich strukturiert programmieren und bietet eine Menge zusätzlicher Befehle. Action und Quick sind sich ähnlich, da sie beide Pascal-orientiert sind. Action ist wesentlich schneller und flexibler, dafür bietet Quick einige zusätzliche Befehle. Die höchste Geschwindigkeit bietet natürlich Assembler, allerdings muß man hier einen wesentlich höheren Programmieraufwand in Kauf nehmen. Welche Sprache Sie für Ihr Programm wählen, hängt davon ab, welche Sie beherrschen und welche Ihnen zur Verfügung steht. Da Turbobasic XL am verbreitetsten ist, werden die hier besprochenen Beispiele in dieser Sprache abgefaßt. Für zeitkritische Operationen können spezielle Assembler-Unterroutinen verwendet werden. Da Turbobasic XL von jedem leicht erlernt werden kann, dürfte es bei Bedarf auch kein Problem sein, einzelne Routinen oder Programme in eine andere Sprache umzusetzen.

Zeichensatzdefinition

Die am häufigsten verwendete Methode der Darstellung von Grafik ist die Definition eines eigenen Zeichensatzes. Der vorgegebene Zeichensatz des Atari umfaßt neben den Buchstaben und Zahlen auch Sonder- und Kontrollzeichen, die allerdings erst durch bestimmte Tastenkombinationen sichtbar werden. Insgesamt gibt es 128 verschiedene Zeichen, die alle in einer 8*8 Bildpunkte großen Matrix aufgebaut sind.

Starten Sie von Programmdiskette 1 das Programm CHSETVIE.TB. Das Programm zeigt Ihnen in vergrößerter Darstellung das eingegebene Zeichen (einfach eine Taste drücken). So können Sie sich mit dem Aufbau eines Zeichens vertraut machen.

Glücklicherweise hat der Atari die angenehme Möglichkeit, diese Zeichen abzuändern, bzw. ihn dazu zu veranlassen, sich die Daten für den Aufbau der Zeichen aus einem anderen Speicherbereich zu holen. Wenn in diesen Speicherbereich dann ein anderer Zeichensatz geschrieben wurde, ist das Resultat die Darstellung von selbstdefinierten Zeichen, die zu allen möglichen Zwecken, wie z.B. Figuren und Landschaften für Spiele, verwendet werden können.

Die 8*8 Bildpunkte eines Zeichens werden im Speicher des Atari in Form von acht Bytes abgelegt. Acht nebeneinanderliegende Bildpunkte ergeben ein Byte, d.h. jeder Bildpunkt ist ein Bit, er kann entweder gesetzt oder nicht gesetzt sein. Jede Reihe von acht Bildpunkten werden als ein Byte zusammengefaßt, indem der erste Bildpunkt, wenn er gesetzt ist, mit dem Wert 128 multipliziert wird, der nächste mit 64, dann mit 32, 16, 8, 4, 2 und 1. Das Gesamtergebnis der Addition aller acht Bildpunkte ist also eine Zahl von 0-255. Starten Sie bitte von der Programmdiskette das Programm BYTEBAU.TB. Sie sehen die Berechnung der acht Bytes am Beispiel des Buchstaben A.

Da der gesamte Zeichensatz aus 128 verschiedenen Zeichen besteht, benötigen wir für die Unterbringung eines eigenen Zeichensatzes genau 128*8=1024 Bytes (1 Kilobyte). Die Reihenfolge, in der die Zeichendaten im Speicher abgelegt werden müssen, zeigt Ihnen das Programm INTCODE.TB. Die auf dem Bildschirm gezeigte Tabelle stellt die Internen Zeichencodes dar. Wie Sie sehen, ist das erste Zeichen das Leerzeichen, es nimmt im 1 Kilobyte langen Speicherbereich die ersten acht Bytes ein. Die nächsten acht Bytes ergeben das Ausrufezeichen usw. Die Adresse, die Sie zur Startadresse Ihres Zeichensatzes addieren müssen, um ein bestimmtes Zeichen zu definieren, ergibt sich aus dem internen Code*8. Angenommen, die Startadresse Ihres

Zeichensatzes läge bei 36864 (wie man auf diese Zahl kommt, erkläre ich später) und Sie wollen das A umdefinieren. Der interne Code des A ist 33, multipliziert mit 8 ergibt das 264: Die Daten für das A liegen also bei 36864+264 bis 36864+264+7 (insgesamt acht Byte). Geben Sie einmal folgende Zeile ein:

```
FOR I=36864+264 TO 36864+264+7:POKE I,RAND(256):NEXT I
```

Was passiert? Erst einmal garnichts, denn der Atari stellt nicht den Zeichensatz dar, der ab Adresse 36864 beginnt, sondern normalerweise immer den, der ab Adresse 57344 beginnt. Da sich dieser nicht ändern läßt, müssen wir auch unseren Zeichensatz bei einer anderen Adresse, nämlich 36864 ablegen. Um den Zeichensatz zu sehen, müssen wir dem Atari noch mitteilen, daß er gefälligst diesen, und nicht den ab 57344 darstellen soll. Die Adresse, in der steht, wo der sichtbare Zeichensatz beginnt, lautet 764. Mit einem POKE-Befehl in Adresse 756 teilen wir dem Atari mit, woher er sich seine Zeichensatzdaten holen soll. Da wir in 756 aber nur eine Zahl von 0-255 schreiben können, wird hier Seitenweise (eine Seite, meist Page genannt, besteht aus 256 Bytes) gezählt. Der Wert in 756 wird also mit 256 multipliziert und ergibt so die Startadresse für den Zeichensatz. Da die Zeichensatzdaten genau 1 Kilobyte lang sind, darf diese Startadresse auch nur an einer 1 Kilobyte Grenze liegen, d.h. sie muß durch 1024 teilbar sein. Für den Wert in 756 heißt das: Er muß durch vier teilbar sein.

Nun, 36864 geteilt durch 256 ergibt 144. Mit POKE 756,144 machen wir also den neuen Zeichensatz sichtbar. Oh, da verschwindet ja alles! Das liegt daran, daß wir bisher nur das A definiert haben, in den restlichen Speicherstellen unseres 1 Kilobyte Bereiches stehen nur Nullen. Wenn die anderen Zeichen auch sichtbar sein sollen, müssen wir weitere Daten in den Speicherbereich schreiben.

Zusammenfassung

Ein Zeichen besteht aus 8 Bytes. Der Zeichensatz besteht aus 128 Zeichen. Er hat damit eine Länge von 1024 Byte = 1 Kilobyte. Die Daten der Zeichen sind in der Reihenfolge der internen Codes abgelegt. Eine Page besteht aus 256 Bytes. In Adresse 756 steht die Pagenummer des auf dem Bildschirm sichtbaren Zeichensatzes. Der Original Zeichensatz beginnt auf Page 224.

Nun eine kleine Aufgabe: Nehmen Sie ein Blatt kariertes Papier und umrahmen Sie ein 8*8 Quadrate großes Feld. Dies stellt nun die Matrix für ein Zeichen dar. Entwerfen Sie eine Figur, z.B. ein Gesicht in dieser Matrix. Berechnen Sie

nun die acht Byte für Ihre Figur. Geben Sie nun folgendes Programm ein:

```
10 PAGE=144:CHS=PAGE*256
20 FOR I=0 TO 7
30 READ A
40 POKE CHS+33*8+I,A
50 NEXT I
60 DATA
```

In Zeile 60 tragen Sie nun bitte nach dem DATA Statement die acht Bytewerte durch Kommata getrennt ein. Für eine einfache Kugel könnte die Zeile z.B. so aussehen:

```
60 DATA 60,126,255,255,255,255,126,60
```

Starten Sie nun das Programm. Wie Sie sehen, passiert nichts. Das ist klar, denn in Adresse 756 steht immer noch die übliche 224. Geben Sie also

```
POKE 756,PAGE
```

ein und alle A's auf dem Bildschirm werden zu dem von Ihnen entworfenen Zeichen. Die Variablen PAGE und CHS werden verwendet, um die Zeichendaten jederzeit an eine andere Stelle legen zu können, ohne das gesamte Programm ändern zu müssen.

Nun können Sie natürlich beliebig viele Zeichen umdefinieren. Bitte versuchen Sie aber nicht, dies durch etliche FOR-NEXT Schleifen zu realisieren! Eine viel einfachere Lösung ist das Lesen der internen Codes. Schreiben Sie in jede DATA-Zeile vor die erste Zeichendate den internen Code des Zeichens, das umdefiniert werden soll. Folgendes Programm liest die Daten ein:

```
10 PAGE=144:CHS=PAGE*256
20 REPEAT
30 READ INTCODE
40 IF INTCODE<>-1
50 FOR I=0 TO 7
60 READ DATE
70 POKE CHS+INTCODE*8+I,DATE
80 NEXT I
90 ENDIF
100 UNTIL INTCODE=-1
```

Wichtig ist allerdings, daß nach der letzten DATA-Zeile eine zusätzliche DATA-Zeile folgt, die nur eine -1 enthält. Dies teilt dem Programm mit, daß

keine weiteren Zeichendaten mehr folgen.

Zeichensatzdaten können, wenn es sich um sehr viele umdefinierte Zeichen handelt, um die DATA-Zeilen zu sparen, auch auf Diskette abgelegt werden. Auf der Programmdiskette befindet sich das File ZONG.CHR. Dieses stellt den beim Auswahlprogramm sichtbaren Zeichensatz dar und besteht aus genau 1024 Bytes. Mit

```
10 PAGE=144:CHS=PAGE*256
20 OPEN #1,4,0,"D:ZONG.CHR"
20 BGET #1,CHS,1024
30 CLOSE#1
40 POKE 756,PAGE
```

kann der Zeichensatz geladen und dargestellt werden.

Multicolor-Zeichensätze

Was mit einfarbigen Zeichensätzen gut aussieht, kann mit mehrfarbigen Zeichensätzen besser aussehen! Die Definition von mehrfarbigen Zeichensätzen ist ebenso einfach: Anstatt acht Punkten ist ein mehrfarbiges Zeichen nur vier Punkte breit. Beim Definieren werden immer zwei Bits zu einem farbigen Punkt zusammengefaßt. Ist das niedrigere Bit allein gesetzt, ergibt sich Farbe eins (Register 708), ist nur das höherwertige Bit gesetzt, ergibt sich Farbe zwei (Register 709). Sind beide Bits gesetzt, ergibt sich Farbe drei (Register 710). Es können also bis zu drei Farben gleichzeitig in einem Character dargestellt werden. Um Farbe vier (Register 711) darzustellen, muß ein Zeichen, das Farbe drei enthält, einfach invers auf den Bildschirm gebracht werden. Hierdurch erscheinen die als Farbe drei definierten Punkte in Farbe 4. Leider können daher nie Farbe drei und vier gleichzeitig in einem Zeichen zu sehen sein.

Die Darstellung der mehrfarbigen Zeichen kann in zwei Grafikmodi erfolgen: Modus 12 mit 40*24 Zeichen und Modus 13 mit 40*12 Zeichen. Modus 13 ist recht ungewöhnlich, da hier die Zeichen doppelt so hoch wie breit erscheinen.

Zum Abschluß noch das Wichtigste: Experimentieren Sie, definieren Sie eigene Zeichen, probieren Sie verschiedene Möglichkeiten aus. Aus Zeichensätzen läßt sich fast alles machen!

PM-Grafik

PM ist eine Abkürzung und steht für PLAYER-MISSILE. Ein Player (dts. Spieler) ist ein Objekt, das ähnlich wie ein Character aus acht Bildpunkten Breite, aber maximal 256 Bildpunkten Höhe aufgebaut ist. Eine Missile besteht aus zwei Punkten Breite und ebenfalls maximal 256 Punkten Höhe. Player und Missiles sind vom übrigen Playfield unabhängig, d.h. die Darstellung eines Players oder einer Missile zerstört nicht die an dieser Stelle auf dem Bildschirm befindliche Zeichen- oder Punktgrafik. Das Programm PMSHOW.TB zeigt vier Player und vier Missiles mit 256 Punkten Höhe. Auf dem Bildschirm sichtbar sind allerdings nicht alle 256 Punkte, da diese zum Teil außerhalb liegen.

Für Player-Missile Grafik haben wir die Wahl zwischen zwei verschiedenen Auflösungen. Die Objekte können wahlweise 128 oder 256 Bytes hoch sein, bei 128 Byte Höhe wird ein Byte doppelt so hoch dargestellt wie bei der 256 Byte Auflösung. Daraus folgt auch, daß der Speicherbereich, der für die PM's benötigt wird, nur halb so groß ist. Das Programm PMRAM.TB zeigt eine Grafik der Speicheraufteilung, PMBASE stellt die Startadresse dar, die ähnlich der Zeichensatz-Adresse berechnet wird.

Wie Sie sehen, benötigt jeder Player entweder 128 oder 256 Byte, während alle vier Missiles zusammen auch 128 oder 256 Byte einnehmen. Die beiden rechten Bits des Missile Speicherbereiches ergeben Missile 0, die nächsten beiden Bits Missile 1 usw. Ebenfalls aus der Grafik ersichtlich ist, daß die ersten 384 bzw. 768 Byte ungenutzt, also frei sind. Wenn Sie keine Missiles darstellen wollen, können Sie diesen Bereich z.B. als Zeichensatz Speicher verwenden (512 oder 1024 Bytes).

Wie programmieren wir nun die PM's? Als erstes müssen Sie dem Computer wieder die Startadresse mitteilen. Diese muß bei der 128 Byte Auflösung an einer 1 Kilobyte-, bei der 256 Byte Auflösung an einer 2 Kilobytegrenze liegen. Wie beim Zeichensatz gibt es auch hier ein Register, in das die Nummer der Page geschrieben wird, nämlich Adresse 54279. Mit POKE 54279,144 z.B. legen wir für die PM-Grafik den Speicherbereich ab 144*256 fest. Als nächstes müssen wir dem Atari die gewählte Auflösung mitteilen, indem wir entweder eine 46 für 128 Byte oder eine 62 für 256 Byte Auflösung in Register 559 schreiben. Aktiviert wird die PM-Grafik durch POKE 53277,3.

Schauen Sie sich einmal das Listing zum Programm PMSHOW.TB an. Als erstes legt das Programm die Pagenummer fest und berechnet daraufhin die Startadresse der Missiles. Die FOR-NEXT Schleife füllt nun den gesamten

PM-Bereich mit Zufallszahlen. Daraufhin wird Grafikmodus null eingeschaltet, gefolgt von der Prioritätsfestlegung in Register 623, dazu später mehr. Nun folgen die schon erwähnten POKEs für die Pagenummer und das Anschalten sowie die Auflösung. Die POKEs in 704-707 legen die Farben der vier Player fest. In 53248 bis 53251 stehen die horizontalen Positionen der Player.

Geben Sie bitte, wenn Sie die Player und Missiles auf dem Bildschirm sehen, folgende Zeile ein:

```
FOR I=0 TO 255:POKE 53248,I:NEXT I
```

Der erste Player wird sich nun von links nach rechts über den Bildschirm bewegen und dann verschwinden. Wie, Sie hätten lieber eine vernünftige Figur im Player? Dazu schreiben wir lieber ein neues Programm ...

```
10 PM=144:PMB=PM*256+1024
15 POKE 54279,PM:POKE 53277,3:POKE 559,62
20 FOR I=0 TO 255
30 POKE PMB+I,0
40 NEXT I
50 DO
60 ? "Bitte horizontale Position eingeben"
70 INPUT X
80 ? "Bitte vertikale Position eingeben"
90 INPUT Y
95 RESTORE
100 FOR I=0 TO 7
110 READ A
120 POKE PMB+Y+I,A
130 NEXT I
135 POKE 53248,X
140 LOOP
150 DATA 60,126,255,255,255,255,126,60
```

Nach Starten des Programmes können Sie die horizontale und vertikale Position des Players eingeben, woraufhin dieser dort erscheint. Die Werte für die Positionen dürfen dabei von 0-255 reichen, obwohl nicht alle diese Positionen auf dem Bildschirm sichtbar sind. Probieren Sie aus, wie groß, bzw. klein die Werte sein dürfen.

Bei Eingabe einer anderen vertikalen Position wird Ihnen auffallen, daß der alte Player nicht verschwindet, sondern nur ein zusätzlicher erscheint. Wie bereits erwähnt ist ein Player so hoch wie der ganze Bildschirm und kann somit auch als vertikale Wand benutzt werden. Machen Sie einfach Zeile 50

des Programmes zu Zeile 15 und der Player wird vor jedem neuen Setzen gelöscht.

Die Farben der Player können, ebenso wie die des restlichen Spielfeldes (Playfield), über Farbregister gewählt werden:

Register 704 für die Farbe von Player und Missile 0, 705 für PM-1, 706 für PM-2 und 707 für PM-3.

Weiterhin können wir für PM's die Priorität festlegen. Dies bedeuted nichts anderes, als "Wichtigkeit". Auf die PM-Grafik übertragen heißt das, daß man entscheiden kann, ob und welche Player "wichtig" sind, d.h. ob sie vor- oder hinter dem übrigen Spielfeld oder den anderen Playern sichtbar sind, wenn eine Überschneidung stattfindet. Das hierzu benutzte Register ist 623. Folgende Abstufungen sind möglich:

Bit 0 (Wert 1): Player 0-3, Playfield 0-3, Hintergrund.

Bit 1 (Wert 2): Player 0-1, Playfield 0-3, Player 2-3, Hintergrund

Bit 2 (Wert 4): Playfield 0-3, Player 0-3, Hintergrund

Bit 3 (Wert 8): PPlayfield 0-1, Player 0-3, Playfield 2-3, Hintergrund

Durch Schreiben des entsprechenden Wertes in das Prioritätsregister wird also die gezeigte Abstufung erzielt. Dies ist z.B. beim Erzeugen von 3D-Effekten wichtig, wenn ein Objekt hinter oder vor einem anderen stehen soll.

Mit dem Prioritätsregister sind allerdings noch weitere Dinge möglich. wird z.B. Bit 4 (Wert 16) gesetzt, so nehmen die vier Missiles anstatt der Farbe der einzelnen Player die Farbe von Playfield 3 (Farbregister 711) an. Hierdurch wird es möglich, die vier Missiles als fünften Player einzusetzen.

Will man mehrfarbige Player benutzen, ist Bit 5 (Wert 32) interessant. Ist dieses Bit gesetzt, entstehen bei sich überlappenden Playern Mischfarben. Auf diese Weise kann man mit nur zwei Playern insgesamt drei Farben erzeugen.

Eben gerade wurde erwähnt, daß man entscheiden kann, welcher Player bei Überlappungen vor dem anderen sichtbar ist. Eine solche Überlappung nennt man "Kollision". Diese Kollision kann ebenfalls abgefragt werden. Dies hat den Sinn, festzustellen, ob z.B. bei einem Spiel zwei Player zusammengestoßen sind, oder man mit einem Raumschiff den Felsen gerammt hat. Für jeden Player und jede Missile existiert ein Kollisionsregister, in dem steht, mit

welchem Player oder welchem Playfield der entsprechende Player kollidiert hat.

Die Register sind 53252-53255 für Player 0-3 mit Playfield Kollision, 53260-53263 für Player 0-3 mit Player Kollision, 53248-53251 für Missile 0-3 mit Playfield Kollision und 53256-53259 für Missile 0-3 mit Player Kollision.

Es fällt hier auf, daß Register vorhanden sind, die doppelt belegt sind. So zum Beispiel Register 53248, welches die horizontale Position von Player 0 und gleichzeitig Missile 0 mit Playfield Kollision ist. Dies hat den einfachen Grund, daß man die horizontale Position nur in das Register schreiben, aber nicht herauslesen kann, während man die Kollision nur lesen, aber nicht schreiben kann.

Bei Kollisionen werden folgende Bits gesetzt: Bit 0 für Kollision mit Player/Playfield/Missile 0, Bit 1 für Player/Playfield/Missile 1, Bit 2 für P/P/M 2 und Bit 3 für P/P/M 3.

Wichtig zu wissen ist, daß das Kollsionsregister die festgestellte Kollision so lange beinhaltet, bis es durch einen POKE 53278,0 gelöscht wird. Auch wenn keine Kollision mehr vorliegt, enthalten die Kollisionsregister noch die entsprechenden Werte, bis o.g. POKE ausgeführt wird. Adresse 53278 löscht alle Kollisionsregister gleichzeitig. Man erinnere sich: In die Kollisionsregister kann man nicht schreiben.

Und noch eine interssante Möglichkeit: Man kann entscheiden, in welcher Breite die Bits der Player auf dem Bildschirm erscheinen. Einfach, doppelt und vierfach stehen hier zur Auswahl. Die Werte für die Register sind: 0=Einfache Breite, 1=Doppelte Breite, 2=Vierfache Breite. Dies kann z.B. sinnvoll sein, wen ein Player als vertikale Wand oder besonders große Figur benutzt wird. Bei Breitschaltung aller vier Player ist es z.B. möglich, den gesamten Bildschirm nur mit vier Playern und Missiles zu füllen. Die Breite der Missiles ist durch ein einziges Register durch Setzen der Bitpaare wählbar, die der Player einzeln.

Hier noch eine Liste der Register für die PM-Grafik (S=Schreiben, L=Lesen) ...

53277 - Grafik Kontroll Register
54279 - Startadresse
559 - Direkter Speicher Zugriff
623 - Priorität

704 - Farbe Player & Missile 0
705 - Farbe Player & Missile 1
706 - Farbe Player & Missile 2

707 – Farbe Player & Missile 3

53248 (S) – Horizontale Position Player 0
53249 (S) – Horizontale Position Player 1
53250 (S) – Horizontale Position Player 2
53251 (S) – Horizontale Position Player 3
53252 (S) – Horizontale Position Missile 0
53253 (S) – Horizontale Position Missile 1
53254 (S) – Horizontale Position Missile 2
53255 (S) – Horizontale Position Missile 3
53256 (S) – Breite von Player 0
53257 (S) – Breite von Player 1
53258 (S) – Breite von Player 2
53259 (S) – Breite von Player 3
53260 (S) – Breite aller Missiles

53278 (S) – Kollision löschen
53248 (L) – Missile 0 mit Playfield Kollision
53249 (L) – Missile 1 mit Playfield Kollision
53250 (L) – Missile 2 mit Playfield Kollision
53251 (L) – Missile 3 mit Playfield Kollision
53252 (L) – Player 0 mit Playfield Kollision
53253 (L) – Player 1 mit Playfield Kollision
53254 (L) – Player 2 mit Playfield Kollision
53255 (L) – Player 3 mit Playfield Kollision
53256 (L) – Missile 0 mit Player Kollision
53257 (L) – Missile 1 mit Player Kollision
53258 (L) – Missile 2 mit Player Kollision
53259 (L) – Missile 3 mit Player Kollision
53260 (L) – Player 0 mit Player Kollision
53261 (L) – Player 1 mit Player Kollision
53262 (L) – Player 2 mit Player Kollision
53263 (L) – Player 3 mit Player Kollision

Um die einzelnen Möglichkeiten besser zu verstehen, schauen Sie sich einfach das Programm PMDEMO.TB auf der Programmdiskette an.

Animation

Als Animation bezeichnet man die Bewegung eines Objektes. Allerdings ist hier nicht das Von-der-Stelle-Bewegen des Objektes gemeint, sondern das Bewegen innerhalb des Objektes. Stellen wir uns ein Männchen von, das von der Seite in einer laufenden Position zu sehen ist. Wird dies nun über den Bildschirm bewegt, ohne animiert zu sein, entsteht der Eindruck, daß Männchen würde rutschen, da keine Laufbewegung vorhanden ist. Laden Sie bitte von der Programmdiskette das File ANIMAT1.TB. Nach Wahl von Menüpunkt eins sehen Sie das eben beschriebene. Menüpunkt 2 zeigt das gleiche Männchen in einer feineren Bewegung, allerdings immer noch ohne Animation. Auch dies wirkt steif und leblos.

Eine Animation wird einfach erzeugt, indem wir das Männchen in zwei verschiedenen Stellungen hintereinander auftauchen lassen. Diese bezeichnet man als Shapes. Menüpunkt 3 zeigt dies. Wird das Männchen nun noch zusätzlich von der Stelle bewegt, entsteht endlich eine lebhafte Bewegung. Dies zeigt Menüpunkt vier.

Die Echtheit einer Animation hängt hauptsächlich von der Anzahl der Shapes ab. Menüpunkt 5 zeigt das gleiche Männchen mit vier Shapes. Diese Animation wirkt natürlich noch lebendiger. Üblicherweise verwendet man bei den meisten Programmen zwei, drei oder vier Shapes. Die Anzahl der Shapes ist natürlich auch eine Frage des Speicherplatzes, da jedes Shape eine gewisse Anzahl Bytes einnimmt.

Die Programmierung einer Animation gestaltet sich ebenso einfach wie die Erklärung des Begriffes. Verwenden Sie einen Zeichensatz, genügt es, mehrere Zeichen mit den verschiedenen Shapes zu definieren. Diese müssen dann nur noch nacheinander gesetzt werden. Folgendes Programm zeigt dies in einer einfachen Weise:

```
10 DIM A$(2)
20 A$="HI":I=1
30 GRAPHICS 2
40 DO
50 POSITION 5,5
60 ? #6;A$(I,I)
70 I=(I=1)+1
80 PAUSE 20
90 LOOP
```

Die zweite Möglichkeit besteht in der Verwendung von mehreren Zeichensätzen. Hier erfolgt die Animation durch einfaches Umschalten zwischen den Zeichensätzen. Dies hat den Vorteil, daß ohne großen Rechenzeitverlust sogar der ganze Bildschirm animiert werden kann. Ein kleines Beispiel hierfür zeigt das folgende Programm, wobei allerdings keine Shapes, sondern der Inhalt des RAM verwendet wird:

```
10 DO
20 POKE 756,RAND(50)
30 PAUSE 10
40 LOOP
```

Als drittes bleibt noch die Animation von Player-Missiles. Hier gibt es entweder die Möglichkeit, immer verschiedene Datenblöcke in den Playerbereich zu kopieren, oder einfach die Player-Missile Startadresse zu ändern. Dies entspricht grundsätzlich dem Umschalten des Zeichensatzes. Beide Methoden benötigen relativ viel Rechenzeit, da entweder das neue Shape kopiert oder die Bewegung in beiden Speicherbereichen getätigt werden muß. Meist verwendet man hier das Kopieren von verschiedenen Datenblöcken, da die zweite Methode den doppelten RAM benötigt.

Titelbilder

Zu einem guten Spiel gehört auch ein gutes Titelbild. Als Titelbild bezeichnet man meist das Bild, das erscheint, bevor das Spiel gestartet wird und nachdem das Spiel beendet ist. In einem Titelbild sollten folgende Informationen enthalten sein:

Die zuletzt erreichte Punktzahl sowie der Highscore. Dies ist besonders wichtig, da man beim Spielen nicht immer Gelegenheit hat, auf die Punktzahl zu achten. Nach dem Spiel aber möchte man gerne seine Leistung sehen. Die Höchstpunktzahl kann dabei immer als Vergleich und Ansporn dienen.

Der Name des Spieles. Dieser wirkt natürlich am besten, wenn er ansprechend gestaltet ist, d.h. z.B. ziemlich groß, extra mehrfarbig oder mit Effekten wie Schatten o.Ä. versehen. Zusätzlich können noch Elemente des Spieles, z.B. der Held oder einige Monster auf der Schrift stehen. Solche Effekte verleihen dem Titelbild immer eine gewisse Eigenständigkeit.

Der Copyright Vermerk. Dieser sollte nicht zu groß sein und nicht aufgesetzt wirken. Dies erreicht man, indem man ihn schlicht und ohne Effekte gestaltet.

Hinweise zur Bedienung. Wie startet man das Spiel? Kann man verschiedene Level anwählen? Dies sollte klar und deutlich lesbar sein, aber nicht zu groß.

Am wichtigsten im Gesamtbild ist der Name des Spieles. Die anderen Informationen sind zwar auch wichtig, müssen aber immer dezent im Hintergrund bleiben.

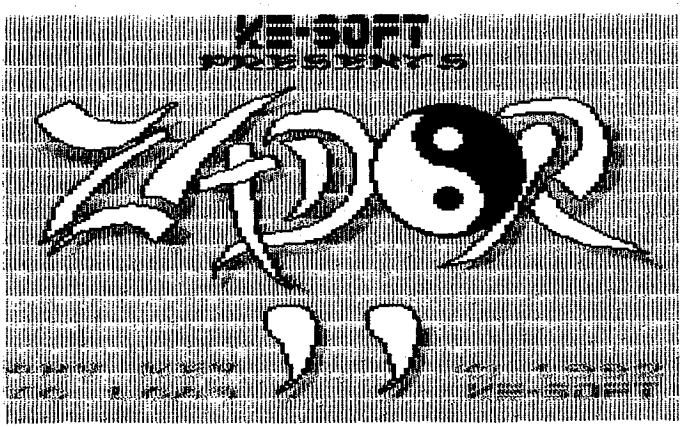
Eine ungefähre Aufteilung des Titelbildschirmes zeigt das Programm TITEL1.TB. Hier sehen Sie in verschieden großen Kästen die einzelnen Komponenten eines Titelbildes.

Zur Erstellung von Titelbildern gibt es mehrere Möglichkeiten: Entweder Sie entwerfen ein Titelbild mit einem Zeichenprogramm und setzen im Spiel die nötigen Informationen wie Score, Highscore usw. hinzu, oder Sie programmieren das Titelbild komplett.

Ein gezeichnetes Titelbild bietet natürlich wesentlich mehr künstlerische Gestaltungsfreiheit als ein programmiertes, ist aber dafür schwieriger zu erstellen. Ein weiterer Nachteil ist, daß das gezeichnete Titelbild relativ viel Speicherplatz in Anspruch nimmt und daher entweder immer wieder nachgeladen wird (lange Wartezeit) oder den Speicher permanent belegt.

Ein programmiertes Titelbild ist meist ganz einfach zu erstellen, da das Spiel selbst eine bestimmte Grafik benutzt, die man im Titelbild auch verwenden kann. Durch verschiedene Grafikmodi (siehe Extrakapitel) und eventuell Displaylist Interrupts läßt sich das Titelbild weiter auflockern.

Als weiteren Zusatz kann das Titelbild durch kleine Animationen oder Bewegungen sowie durch Musik verfeinert werden. Solange Ihr Spiel aber noch nicht fertig ist, sollten Sie sich jedoch mit einem provisorischen Titelbild begnügen. Das richtige Titelbild kann immer noch später hinzugefügt werden. Verwenden Sie den wertvollen Speicher lieber für ein gutes Spiel und nutzen den übrigen dann für das Titelbild, als umgekehrt.



Tastatur

Neben dem bekannten INPUT-Befehl gibt es mehrere Möglichkeiten, die Tastatur abzufragen. Meist wird nur ein einzelner Tastendruck benötigt, daher ist natürlich der INPUT-Befehl ungeeignet.

Mit Hilfe des GET-Befehles können wir auf einen Tastendruck warten, dessen Ergebnis dann in der folgenden Variable als ASCII-Wert gespeichert wird:

```
10 ? "Bitte eine Taste druecken."
20 GET A
30 ? "Sie haben ";CHR$(A);" getippt."
```

Die zweite Möglichkeit besteht darin, das Register 764 abzufragen. Wurde keine Taste gedrückt, befindet sich hier eine 255. Tippt man eine Taste, wird der Tastaturcode in 764 gespeichert und kann jederzeit abgefragt werden. Selbst wenn die Taste losgelassen wurde, bleibt der Wert in 764 erhalten, außer, Sie tippen eine andere Taste. Nach der Abfrage des Registers sollten Sie dessen Inhalt wieder auf 255 setzen. Mit Hilfe dieses Registers läßt sich die Tastatur sehr gut abfragen, ohne den Programmablauf zu unterbrechen:

```
10 DO
20 SOUND 0,RAND(100),10,6
30 IF PEEK(764)<>255 THEN ? "Taste gedrueckt.":POKE 764,255
40 LOOP
```

Der Nachteil dieses Registers besteht darin, daß hier weder der ASC-, noch der interne Code gespeichert werden. Der dort enthaltene Tastaturcode hat eine so seltsame Reihenfolge, daß man diese Methode meist nur für einzelne Tasten benutzt.

Beide Methoden lassen sich natürlich auch kombinieren. Dadurch ist es möglich, den Programmablauf fortzusetzen und trotzdem die getippte Taste zu erkennen:

```
10 DO
20 REPEAT
30 SOUND 0,RAND(100),10,6
40 UNTIL PEEK(764)<>255
50 GET A
60 ? "Taste ";CHR$(A);" getippt."
70 LOOP
```

Konsolentasten

Als Konsolentasten bezeichnen wir die Tasten START, SELECT und OPTION. Diese Tasten werden, schon wegen ihrer Bezeichnungen, oft zur Einstellung von verschiedenen Spielvariationen verwendet.

Die Abfrage der Konsolentasten gestaltet sich ganz einfach. Speicheradresse 53279 beinhaltet einen Wert, der die gerade gedrückten Tasten angibt. Eine Sieben als Ergebnis bedeuted, daß keine Taste gedrückt ist. Die START-Taste liefert eine Sechs, SELECT eine Fünf und OPTION eine Drei. Folgendes Programm zeigt die Anwendung:

```
10 DO
20 P=PEEK(53279)
30 IF P=6 THEN ? "START"
40 IF P=5 THEN ? "SELECT"
50 IF P=3 THEN ? "OPTION"
60 LOOP
```

Interessant ist noch, daß man auch abfragen kann, ob mehrere Tasten gleichzeitig gedrückt wurden. Lassen Sie sich einmal die Werte des Registers 53279 ausgeben, und Sie werden feststellen, daß START mit SELECT zusammen, START mit OPTION, SELECT mit OPTION und sogar alle drei Tasten zusammen unterschiedliche Werte produzieren. Solche Tastenkombinationen lassen sich sehr gut für geheime Cheats einsetzen ...

Joystick

Das bei Spielen am häufigsten verwendete Eingabegerägt ist der Joystick. Ein Joystick ist leicht zu handhaben und mit ihm läßt sich präzise und schnell steuern. Außerdem gibt es Joysticks für ganz wenig Geld überall zu kaufen.

Die Abfrage des Joysticks ist ebenso einfach, wie die der Konsolentasten: Die Funktion STICK(n) liefert die Joystickwerte. Probieren Sie einmal die Zeile

```
DO:? STICK(0):LOOP
```

aus. Hier sehen Sie auch gleich, welche Joystickposition welchen Wert liefert. Am besten, Sie notieren sich die Werte auf einen Zettel, den Sie in der Nähe Ihres Atari befestigen. Die Null in Klammern gibt übrigens den Joystickport an, der abgefragt wird. Mit STICK(1) können Sie demnach also den zweiten Joystick abfragen.

Der Feuerknopf des Joysticks hat einen eigenen Befehl: STRIG(n). Eine Null als Ergebnis heißt "Feuerknopf gedrückt", eine eins heißt "Feuerknopf nicht gedrückt".

CX-85 Zehnertastatur

Seit einiger Zeit macht ein neues Eingabemedium von sich reden: Die CX-85 Zehnertastatur. Diese externe Zehnertastatur läßt sich nicht nur bei der Eingabe von großen Datenmengen gut einsetzen, sondern eignet sich auch zur Steuerung von Spielen hervorragend. Das sieht man nicht zuletzt daran, daß immer mehr neue Spiele erscheinen, die auch die CX-85 als Bedienelement zulassen.

Die Abfrage der CX-85 gestaltet sich ähnlich der des Joysticks. Auch die CX-85 liefert Joystickwerte. Allerdings ist hier ein kleiner Unterschied zu erkennen: Mit STICK(n) läßt sich jederzeit die zuletzt getippte Taste abfragen. Ist im Moment eine Taste gedrückt, liefert STRIG(n) den Wert null, ansonsten eine eins. Probieren Sie es aus und notieren Sie sich die erhaltenen Werte. Auffällig ist, daß die Tasten "0" und "ESCAPE" den gleichen Wert liefern. Warum? Nun, der Joystick liefert nur vier Bit als Signal, daher sind auch nur 16 verschiedene Kombinationen möglich. Da die CX-85 aber 17 Tasten besitzt, muß zwangsweise eine doppelt belegt sein.

Bildschirmaufbau

Der Aufbau des Bildschirmes ist eigentlich eine ganz einfache Sache. Zunächst sollte man mit Hilfe des GRAPHICS-Befehles in den gewünschten Grafikmodus gehen. Bei komplexeren Aufbauten kann auch eine eigene Displsylist (siehe Extrakapitel) erstellt werden.

Der Bildschirm sollte natürlich während des Spieles die wichtigsten Informationen wie z.B. die Levelnummer, Anzahl der Leben, übrige Zeit usw. enthalten. Meist werden diese in einer oder mehreren Kopf- oder Fußzeilen dargestellt.

Der Rest des Bildschirmes wird vom Spielgeschehen eingenommen. Der Aufbau dieses Teiles kann auf mehrere Weisen erfolgen. Am einfachsten ist es, durch mehrere PRINT, bzw. PRINT #6 Befehle das Playfield zu zeichnen. Besteht das Playfield aus einer einfacheren Konstruktion, kann man ebenso auf PLOT- und DRAWTO-Befehle zurückgreifen. Bei PLOT und DRAWTO im Textgrafikmodi bestimmt man mit COLOR den ASC-Wert des Zeichens, mit dem gezeichnet wird. Das Programm BILDAUFB.TB demonstriert dies.

Sind in Ihrem Spiel mehrere verschiedene Bildschirmaufbauten vorhanden, z.B. durch mehrere Level, so sollte eine Unterroutine den grundsätzlichen Aufbau übernehmen, während die verschiedenen Level extra bearbeitet werden. Näheres hierzu auch im Kapitel "Levelspeicherung".

Gemischte Grafikmodi

Gemischte Grafikmodi dienen dazu, das Bild aufzulockern. So kann man z.B. wichtigere Schriftzüge oder Elemente größer darstellen, als den Rest. Zur Programmierung ist es zunächst wichtig, zu wissen, wie das Ganze vor sich geht.

Der Bildschirmaufbau des Atari wird bestimmt durch die Display-List. Auf deutsch bedeuted das "Anzeige-Liste" und hat auch diese Bedeutung: In dieser Liste steht, aus welchen Grafikmodi der Bildschirm zusammengesetzt ist. Die Zusammensetzung erfolgt immer Zeilenweise, d.h. wir können theoretisch in jeder Zeile einen anderen Grafikmodus wählen.

Der Atari legt die Display-List an einer bestimmten Adresse ab, die wir durch DPEEK(560) ermitteln können. Die Display-List baut sich aus mehreren Arten von Daten auf: Sprunganweisungen, Leerzeilenbefehle, Grafikmodebefehle und Textmodebefehle.

Die Sprunganweisungen dienen dazu, dem Atari mitzuteilen, aus welchem RAM-Bereich er sich den in den folgenden Zeilen darzustellenden Bildschirminhalt holt. Leerzeilenbefehle erzeugen eine oder mehrere Blanklines, das sind leere Zeilen mit einer Höhe von einem Pixel. Der gesamte Bildschirm ist 192 Pixels hoch. Die Grafikmodebefehle erzeugen eine Zeile mit Pixelgrafik, die Textmodebefehle erzeugen eine Zeile mit Text.

Geben Sie einmal folgendes Programm ein:

```
10 GRAPHICS 0
20 DLIST=DPEEK(560)
30 FOR I=0 TO 31
40 ? PEEK(DLIST+I)
50 NEXT I
```

Dieses Programm zeigt uns den Aufbau der Display-List im Grafikmodus 0. Als Ergebnis müßten folgende Werte erscheinen:

112, 112, 112, 66, lb, hb, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 65, dl, dh

Die ersten drei Werte sind Leerzeilenanweisungen. Bei Leerzeilenanweisungen errechnet sich die Anzahl der Leerzeilen durch (Leerzeilen-1)*16. Für eine Leerzeile steht also eine Null, für vier Leerzeilen eine 48, für acht Leerzeilen

eine 112. Diese Leerzeilen sind oberhalb des Textbildschirmes zu sehen.

Der Wert 66 ist eine Sprunganweisung sowie eine Zeile des Grafikmodus 0. Für Grafikmodus 0 steht eine 2, die Sprunganweisung ist die 64. Die beiden Daten nach der Sprunganweisung geben in Form von Low- und Highbyte die Bildschirmadresse an. Diese kann ebenfalls mit DPEEK(88) ermittelt werden. Vergleichen Sie einmal das Ergebnis von DPEEK(88) und lb+hb*256.

Die nächsten Bytes stellen die übrigen 23 Textzeilen dar. Danach kommt wieder eine Sprunganweisung zum Anfang der Display-List. Der Wert dl+dh*256 ist also der gleiche, wie der Wert aus DPEEK(560).

Um nun die Display-List zu modifizieren, sollten Sie zunächst wissen, welche Werte welches Ergebnis liefern:

2: Grafik 0 Modus, 40 Zeichen pro Zeile, 24 Zeilen.
4: Grafik 12 Modus, 40 Zeichen pro Zeile, 24 Zeilen.
5: Grafik 13 Modus, 40 Zeichen pro Zeile, 12 Zeilen.
6: Grafik 1 Modus, 20 Zeichen pro Zeile, 24 Zeilen.
7: Grafik 2 Modus, 20 Zeichen pro Zeile, 12 Zeilen.
8: Grafik 3 Modus, 40 Punkte pro Zeile, 24 Zeilen.
10: Grafik 5 Modus, 80 Punkte pro Zeile, 48 Zeilen.
13: Grafik 7 Modus, 160 Punkte pro Zeile, 96 Zeilen.
14: Grafik 15 Modus, 160 Punkte pro Zeile, 192 Zeilen.
15: Grafik 8 Modus, 320 Punkte pro Zeile, 192 Zeilen.

Wenn wir eine vorhandene Display-List ändern wollen, sollten wir zunächst den Grafikmodus wählen, bei dem am wenigsten Modifikation nötig ist. Dies ist natürlich der Modus, der die meisten Zeilen so besitzt, wie wir sie haben wollen.

Die ersten drei Anweisungen der Display-List sind in allen Modi gleich. Diese ignorieren wir am besten, indem wir unsere DLIST-Variable auf DPEEK(560)+3 setzen.

Beim Ändern der Display-List müssen wir natürlich auf die Sprunganweisungen achten. Da, wo eine ist, muß sie auch bleiben, sonst sieht es schlecht aus. Ebenfalls sieht es schlecht aus, wenn wir z.B. durch Ersetzen von mehreren Grafik 0 Zeilen durch Grafik 2 Zeilen die Bildschirmhöhe überreizen. Probieren Sie das einfach selbst aus:

Sprunganweisung gekillt: DLIST=DPEEK(560)+3:POKE DLIST,2

Bildschirmhöhe überreizt: DLIST=DPEEK(560)+3:FOR I=5 to 10:POKE

```
DLIST+I,7:NEXT I
```

Experimentieren Sie einfach mal mit dem Ändern der Display-List. Wenn Sie noch nicht zurecht kommen, sehen Sie sich einfach die Programme DLIST1.TB, DLIST2.TB und DLIST3.TB an. Diese demonstrieren verschiedene Aufbauten und deren Programmierung.

Beim Darstellen von Text mit gemischten Grafikmodi ist allerdings darauf zu achten, daß eine Zeile mit nur zwanzig Bytes die Positionen verschiebt, da z.B. im Grafikmodus 0 jede Zeile 40 Bytes hat. Dies ist an den Beispielprogrammen sehr gut zu erkennen.

Darstellung und Bewegung

Die Darstellung von Grafik, bzw. Zeichensatzgrafik ist sehr stark abhängig vom Grafikmodus. Grundsätzlich gibt es aber drei verschiedene Methoden:

Mit Hilfe der Befehle POSITION x,y und PRINT, bzw. PRINT #6 ist es möglich, Zeichen oder Zeichenketten an einer bestimmten Stelle auf dem Bildschirm darzustellen. Diese Methode ist relativ langsam und daher nur sinnvoll, wenn längere Zeichenketten, wie z.B. Text dargestellt werden sollen.

Als zweites gibt es die Befehle COLOR, PLOT und DRAWTO. Damit lassen sich ebenfalls Zeichen auf dem Bildschirm darstellen. In Textmodi kann man mit COLOR den ASC-Wert des darzustellenden Zeichens wählen, mit PLOT und DRAWTO eines oder mehrere Zeichen darstellen. Diese Methode ist schneller und läßt sich für Spiele gut einsetzen.

Die dritte Möglichkeit ist das Schreiben der internen Codes direkt in den Bildschirmspeicher. Dazu muß vorher der Anfang des Bildschirmspeichers durch DPEEK(88) ermittelt werden. Die Adresse für eine x/y Position errechnet sich dann durch Startadresse+y*Zeichen pro Zeile+x. Der Wert, der in diese Adresse geschrieben wird, muß der interne Code des darzustellenden Zeichens sein. Diese Methode ist zweifellos die schnellste von allen, aber etwas schwieriger in der Anwendung.

In unseren Beispielen werden wir die COLOR-PLOT Methode vorziehen, da diese am leichtesten nachzuvollziehen ist. Für zeitkritische Dinge kann man immer noch die POKE-Methode verwenden.

Die Darstellungsart der Zeichen hängt nun vom Grafikmodus ab:

Grafikmodus 0 kann alle 128 verfügbaren Zeichen normal oder invers darstellen. Es gibt nur eine Farbe sowie den Hintergrund. Es stehen 24 Zeilen mit je 40 Zeichen zur Verfügung. Dieser Modus ist sehr hochauflösend, aber dafür nur einfarbig.

Grafikmodus 1 stellt nur die ASC-Werte 32-95 dar. Die Verwendung der anderen Werte bis 128 ergibt die Zeichen 32-95 mit einer anderen Farbe. Diese so erzielbaren zwei Farben lassen sich duch hinzuaddieren von 128 (also inverse Darstellung der Zeichen) erhöhen: Das Resultat sind die Farben 3 und 4. Es stehen ebenfalls 24 Zeilen, allerdings jeweils nur mit 20 Zeichen zur Verfügung, dafür kann jedes Zeichen eine von vier Farben annehmen.

Grafikmodus 2 ist identisch mit Modus 1, bietet allerdings eine Auflösung von 12 Zeilen mit je 20 Zeichen und ebenfalls vier Farben.

Grafikmodus 12 kann, wie Modus 0, alle 128 verfügbaren Zeichen darstellen. Da es sich hier um einen Multicolor-Modus handelt, sind alle Zeichen maximal dreifarbig. Die vierte Farbe wird erreicht, indem 128 zum ASC-Wert addiert wird, wodurch die Pixels in Farbe 3 des Zeichens zu Farbe 4 werden. Es stehen 24 Zeilen mit je 40 Zeichen zur Verfügung.

Grafikmodus 13 ist identisch mit 12, bietet allerdings eine Auflösung von 12 Zeilen mit je 40 Zeichen. Dies ist der einzige Modus, in dem ein Zeichen höher als breit dargestellt wird.

Wollen wir nun ein Zeichen auf dem Bildschirm bewegen, benötigen wir zwei Variablen, die die horizontale und vertikale Position festlegen. Nennen wir sie einfach x und y. Die linke obere Ecke des Bildschirmes ist Position 0,0. Nach rechts und unten werden die x-, bzw. y-Werte größer. Das Maximum ist je nach Modus unterschiedlich.

Zur Bewegung eines Zeichens muß einfach seine Position verändert und das Zeichen danach erneut gesetzt werden. Damit dies sinnvoll ist, integrieren wir einfach eine Joystickabfrage, die die Veränderung der x- und y-Werte übernimmt. Dies zeigt das Programm BEWE1.TB.

Hier fällt natürlich auf, daß bei der Bewegung des Zeichens eine Linie entsteht. Dies liegt einfach daran, daß vor der Bewegung das alte Zeichen nicht gelöscht wird. Das ändern wir einfach durch Hinzufügen einer Zeile, die mit Farbe 32 (also dem Leerzeichen) die Position belegt, sobald der Joystick bewegt wurde. Dies zeigt das Programm BEWE2.TB.

Soll eine solche Bewegung für ein Spiel Anwendung finden, ist es natürlich besser, für jede Bewegungsrichtung ein anderes Zeichen zu haben, damit die spätere Figur immer ein zur Bewegungsrichtung passendes Shape (Aussehen) annimmt. Dies zeigt das Programm BEWE3.TB.

Um nun noch zu verdeutlichen, wie das ganze mit umdefinierten Zeichen aussieht, habe ich das Programm BEWE4.TB als Abschluß geschrieben. Lassen Sie sich bitte nicht von der Zeile mit den seltsamen Zeichen verwirren, das sind nur die Zeichen-Daten, die auf diese Weise speichersparend untergebracht sind. Wie das geht, steht im Kapitel "Einsparung von Speicherplatz".

Wenn Sie nun Ihr eigenes Programm schreiben, das eine Figur nach Joystickabfrage bewegt, achten Sie bitte darauf, daß das Löschen der Figur

möglichst kurz vor dem erneuten Hinsetzen erfolgt, da sonst der Eindruck eines Flimmerns erzeugt wird.

Als weitere Erweiterung können Sie einmal probieren, eine Figur darzustellen, die aus mehreren Zeichen besteht. Hierbei müssen Sie natürlich jeweils mehrere Zeichen löschen und setzen. Achten Sie dann darauf, daß nur die Zeichen gelöscht werden, die nicht von der neuen Position überschrieben werden, das wäre nämlich doppelt gemoppelt und sieht ebenfalls nach Flimmern aus ...

Character-Animation

Nachdem wir nun fleißig Zeichen dargestellt und bewegt haben, fällt uns vielleicht auf, daß sich diese Zeichen zwar schnell, aber trotzdem immer ruckartig bewegen. Um diesem Übel vorzubeugen gibt es einen einfachen Trick, den man als Character-Animation bezeichnet. Er besteht einfach darin, mehrere Zeichen zu definieren, die das zu bewegende Zeichen in einer Position darstellen, die um einen halben oder viertel Schritt bewegt ist.

Starten Sie von der Programmdiskette das Programm CHRANIM.TB. Es demonstriert die Funktionsweise dieses Tricks anschaulich. Zuerst werden drei Zeichen umdefiniert: Das erste Zeichen stellt einen Ball dar. Die nächsten beiden Zeichen ergeben zusammengesetzt den gleichen Ball um einen halben Character bewegt. Im Grafikmodus zwei wird nun von links nach rechts immer der Ball, danach der halb bewegte Ball gezeichnet. Wie man sieht, reicht dieser eine Zwischenschritt aus, um ab einer gewissen Geschwindigkeit den Eindruck einer feinen Bewegung zu erzielen. Grundsätzlich gilt: Je schneller die Bewegung, desto gröber kann sie sein, ohne grob auszusehen. Daraus folgt, daß ein langsames Objekt wesentlich feiner bewegt werden sollte, als ein schnelles. Am Beispielprogramm sieht man deutlich, daß die langsame Geschwindigkeit nicht ausreicht, um den Eindruck einer feinen Bewegung entstehen zu lassen. Erst die schnellste Geschwindigkeit vermittelt diesen Eindruck realistisch. Experimentieren Sie einfach einmal mit einem oder mehreren Zwischenschritten.

Positionsberechnung

Wenn Sie ein Spiel programmieren möchten, das Zeichensatz- und Player-Missile Grafik gleichzeitig verwendet, ist es wichtig zu wissen, wie man Positionen von Zeichen in Playerpositionen umrechnen kann. Gehen wir einmal von der höheren Player-Auflösung (POKE 559,62) aus und nehmen an, Sie haben einen Player, der genau einen Graphics 0 Character groß ist, definiert. Der linke Bildschirmrand (horizontale Position 0) hat hier den Wert 48. Jedes Zeichen ist vier Player-Punkte breit. Demnach errechnet sich die horizontale Position für den Player mit 48+X*4. Für die vertikale Position gilt als oberer Rand 32, hier ist ein Zeichen allerdings acht Player-Punkte hoch. Die vertikale Position errechnet sich also aus 32+Y*8. Laden Sie bitte von der Programmdiskette das Programm POSBER1.TB. Es demonstriert diese Berechnung, indem Sie immer einen X- und einen Y-Wert eingeben sollen. An die Stelle dieser Zeichenposition wird nun der Player gesetzt.

Als nächstes folgt die zweite Möglichkeit: Die Umrechnung von Player- in Zeichenpositionen. Dies wird z.B. benötigt, wenn Sie einen Player als bewegliche Figur benutzen und anhand seiner Position feststellen wollen, an welcher Zeichenposition er sich befindet. Hierzu wird einfach die umgekehrte Berechnung angewandt: X-Position des Zeichens = (Playerposition-48)/4. Für Y ebenso: Y-Position des Zeichens = (Playerposition-32)/8. Das Programm POSBER2.TB zeigt dieses Prinzip. Sie können den Player per Joystick umher steuern, während das Programm den Character, über dem der Player nach dieser Formel steht, invers darstellt.

Auf diese Weise ist es z.B. möglich, bei einem Leiterspiel festzustellen, ob der Player vor einer Leiter steht oder nicht. Am einfachsten ist allerdings die folgende Methode: Bei des Festlegung der Playerposition wird gleichzeitig die Characterposition mit festgelegt. Bei jeder Bewegung des Players wird dann die Characterposition mit verändert. Z.B.: PLX=PLX+1:CX=CX+0.25 entspricht einer Bewegung um einen Punkt nach rechts, während sich die Zeichenposition um einen viertel Character ändert. Auch mit dieser Technik sollten Sie ein wenig experimentieren, bevor Sie sie anwenden.

Kollisionen

Unter einer Kollision verstehen wir einen Zusammenstoß bzw. die Überlappung von zwei Figuren in einem Spiel. Wie wir diesen Vorgang bei Player-Missile Grafik abfragen können, haben wir bereits gelernt. Bei Zeichensatzgrafik ist es fast genauso einfach, denn es gibt eine Funktion, mit dessen Hilfe wir den Inhalt des Bildschirmes an einer bestimmten Stelle erfragen können. Mit LOCATE 0,0,A sagt uns der Computer, was an Position 0,0 für eine Farbe, bzw. für ein ASC-Wert liegt. Das Ergebnis schreibt er in die Variable A. Natürlich können wir anstatt einer festen Position auch Variablen als Positionsangabe machen. Auch die Variable A kann durch andere ersetzt werden. So liefert z.B. LOCATE F,G,Z den Inhalt von Position F/G in die Variable Z. Die Locate-Funktion funktioniert übrigens in allen Grafikmodi. Ob das Ergebnis eine Farbe oder ein ASC-Wert ist, hängt vom Grafikmodus ab.

Sehen Sie sich am besten einmal das Programm LOCATE1.TB von der Programmdiskette an. Es baut zunächst per Zufall einen Bildschirm auf. Danach können Sie per Joystick ein "#"-Zeichen, welches sich in der Farbe von den anderen Zeichen unterscheidet (Sie erinnern sich: Kleinschrift/Großschrift/Invers?), umhersteuern. Wenn Sie an ein anderes Zeichen anstoßen, zeigt ihnen der Computer in der unteren Zeile das Zeichen sowie dessen ASC-Wert. Bewegen Sie sich auf freiem Raum, erscheint eine 32, da das Leerzeichen den ASC-Wert 32 hat.

In Ihren Spielen sollten Sie dann je nach Ergebnis des LOCATE-Befehles eine entsprechende Unterroutine aufrufen, die die Erledigung übernimmt. Z.B.: Gegenstand genommen, Spieler gestorben, Level geschafft usw.

Als zweite Möglichkeit, den Bildschirminhalt abzufragen, können Sie einfach den Inhalt des Bildschirmspeichers lesen. Mit DPEEK(88) erhalten Sie die Startadresse des Bildschirmspeichers. Die Adresse für eine X/Y-Position errechnet sich also aus DPEEK(88)+Y*Zeichen pro Zeile+X, im Grafikmodus 1 beispielsweise DPEEK(88)+Y*20+X. Das Ergebnis ist hier allerdings kein ASC-Wert, sondern der interne Code. Dieser ist identisch mit dem Wert, der durch POKE in die Bildschirmadresse geschrieben wird. Die Anwendung dieser Möglichkeit ist demnach nur sinnvoll, wenn sowohl das Setzen der Figuren als auch das Erkennen des Bildschirminhaltes direkt durch die Bildschirmadresse erfolgt. Das Programm LOCATE2.TB zeigt dies. Der Ablauf des Programmes ist übrigens identisch mit dem von LOCATE1.TB. Allerdings werden hier die internen Codes anstatt der ASC-Werte gezeigt.

Überlappungen

Wir wir abfragen, ob eine Figur an eine andere angestoßen ist, wissen wir inzwischen. Was aber, wenn eine Figur über eine andere, bzw. über verschiedene Hintergrundcharacters hinweg gehen soll, ohne diese zu verändern?

Auch dieses Problem ist ganz einfach zu lösen. Da wir vor der Bewegung mittels Locate oder Peek feststellen, welcher Character an der Stelle ist, wo sich die Figur hinbewegen soll, können wir diesen einfach in einer Zwischenvariable speichern. Wenn sich die Figur dann weiter bewegt, müssen wir nur noch diesen Character wieder an die Stelle setzen. Das Programm LOCATE3.TB erledigt dies. Hier wird wieder die Locate-Methode angewandt, es funktioniert natürlich genauso auch mit Peek und Poke. Anstatt an die Zeichen anzustoßen, kann man sich hier durch sie hindurch bewegen. Das Programm zeigt in der unteren Zeile an, welcher Character sich hinter der Figur befindet.

Wenn Sie sich das Programm ansehen, fällt auf, daß die Variable HINTER, in der der Hintergrundcharacter gespeichert wird, anfangs auf den Wert 32 gesetzt wird. Dies hat den einfachen Grund, daß das Leerzeichen den ASC-Wert 32 hat.

Bedingungen

Unter Bedingungen verstehen wir hier Dinge, die im Spiel passieren, bzw. nur passieren können, wenn bestimmte Vorraussetzungen erfüllt sind. Stellen wir uns vor, wir wollen eine Tür öffnen. Dies soll natürlich nur möglich sein, wenn wir vorher den passenden Schlüssel genommen haben. Wenn der Schlüssel berührt, also genommen, wird, erkennen wir durch eine Kollisionsabfrage (Locate oder Peek). Damit das Programm auch später, wenn wir die Tür öffnen wollen, noch weiß, daß wir den Schlüssel genommen haben, müssen wir dieses Ereignis speichern. Hierzu verwenden wir am günstigsten auch wieder eine Variable. Solche Variablen, die den Zustand einer Bedingung angeben, nennen wir FLAGs, auf deutsch Flaggen. Die Bezeichnung folgt aus der Tatsache, daß man an wichtigen Stellen zur Markierung Flaggen in den Boden steckt. Sehen wir uns das Programm FLAG1.TB an. Es handelt sich um ein einfaches Spiel. Zunächst werden vier Zeichen umdefiniert. Benötigt wird eine Spielfigur, eine Mauer, ein Schlüssel und eine Tür. Im Grafikmodus eins wird daraufhin ein Bildschirm mittels PRINT aufgebaut. Man startet in der linken oberen Ecke. Nun kann die Figur bewegt werden. Um zur rechten unteren Ecke zu gelangen, muß man durch mehrere Türen hindurch, was aber nur geht, wenn man jeweils vorher einen Schlüssel nimmt. Wenn man einen Schlüssel nehmen will, gibt es programmiertechnisch zwei Möglichkeiten: Entweder, man kann immer nur einen Schlüssel nehmen und muß dann damit eine Tür öffnen, bevor man einen weiteren nehmen kann. In diesem Fall muß man abfragen ob das KEYFLAG, also die Variable für den Schlüssel, Null (=kein Schlüssel genommen) ist. Wenn ja, darf er genommen werden und KEYFLAG wird auf eins gesetzt. Die andere Möglichkeit ist, mehrere Schlüssel tragen zu können. In diesem Fall muß man die Anzahl der getragenen Schlüssel speichern. Beim Nehmen eines Schlüssels also KEYFLAG=KEYFLAG+1, beim Benutzen KEYFLAG=KEYFLAG-1. Das Beispielprogramm verwendet die erste Möglichkeit. Die zweite Möglichkeit findet eher bei Spielen Anwendung, die mehr in Richtung Adventure gehen.

Finden im Spiel mehrere verschiedene Gegenstände Anwendung, werden einfach mehrere Variablen verwendet. Z.B. Bomben, Schlüssel, Diamanten usw. Lassen Sie Ihrer Fantasie freien Lauf.

Auf einen Sonderfall will ich allerdings noch eingehen: Stellen Sie sich vor, Sie müssen im Spiel verschiedene Gegenstände in einer bestimmten Reihenfolge sammeln. Es wäre Blödsinn, hier alle Bedingungen zu speichern. Wenn Sie z.B. die ASC-Werte 33, 56, 127, 254, 153 und 87 nacheinander einsammeln sollen, ist es einfacher, einen String zu verwenden, in dem nacheinander alle ASC-Werte in der richtigen Reihenfolge enthalten sind. Im Programm müssen

Sie dann nur noch einen Zeiger verwenden, der angibt, welcher Gegenstand als nächstes genommen werden muß. Also am Anfang der erste (ZEIGER=1). Wenn man dann den Gegenstand ASC(GEG$(ZEIGER,ZEIGER)) nimmt, wird der Zeiger um eins erhöht und das Programm reagiert nur noch auf den zweiten Gegenstand. Beispiel: Der Spieler soll die Buchstaben des Wortes ZONG in dieser Reihenfolge sammeln. Wir setzen GEG$="ZONG" und ZEIGER=1. Wenn dann Z (Der Character, der durch Locate festgestellt wird) = ASC(GEG$(ZEIGER,ZEIGER)), wird der Gegenstand genommen und ZEIGER=ZEIGER+1 erhöht. Das Programm FLAG2.TB zeigt dies. Hier kann allerdings vor Spielbeginn ein beliebiges Wort mit bis zu zwanzig Zeichen Länge eingegeben werden (bitte nur Großbuchstaben verwenden). Dieses Wort muß dann in dieser Reihenfolge gesammelt werden.

Haben Sie alle Buchstaben gesammelt, gibt das Programm einen ERROR aus, weil die Bedingung, ob das Spiel geschafft ist oder nicht, noch nicht enthalten ist. Dies ist das Thema des nächsten Kapitels.

Aufgaben

Wie wir im letzten Kapitel gesehen haben, muß unser Spielprogramm erkennen können, wann wir eine Aufgabe gelöst haben. Hierzu gibt es viele Möglichkeiten, abhängig von der Art des Spieles:

Bei Spielen wir DONKEY KONG hat man die Aufgabe, eine bestimmte Position des Bildschirmes zu erreichen. Wenn die Variablen X/Y die Position des Spielers darstellen, kann man also abfragen ob X=... AND Y=..., wenn eine ganz bestimmte Position erreicht werden soll. Soll man z.B. den oberen Rand erreichen, wird Y=0 abgefragt. Natürlich läßt sich jede Position oder jeder Positionsbereich abfragen, so z.B. die Mitte des Bildschirmes in Modus 1: IF X>7 AND X<12 AND Y>9 AND Y<13 THEN ...

Anstatt einer bestimmten Position kann man auch ein bestimmtes Zeichen erreichen müssen. Dies wird mit dem üblichen LOCATE oder PEEK abgefragt.

Bei Spielen wie z.B. Pac Man muß man eine bestimmte Anzahl Gegenstände (bzw. Punkte bei Pac Man) einsammeln. Hierzu wird zu Beginn die Anzahl in einer Variable festgelegt. Beim Sammeln eines Gegenstandes wird diese Variable um eins erniedrigt. Ist sie auf Null, hat man es geschafft.

Beim letzten Beispielprogramm des letzten Kapitels hatte man es geschafft, wenn alle Gegenstände gesammelt waren. Man könnte hier ebenfalls einen Zähler einfügen. Einfacher wäre es allerdings, wenn man abfragt, ob der LETZTE Gegenstand genommen ist. Bei Aufgaben, deren Erfüllung nur unter bestimmten Vorraussetzungen möglich ist, müssen diese Vorraussetzung nicht mehr extra abgefragt werden. Hat man den letzten Gegenstand genommen, muß nicht mehr abgefragt werden, ob man die vorherigen bereits genommen hat, wenn das Programm das Nehmen des letzten Gegenstandes sowieso nur zuläßt, wenn die vorherigen in der richtigen Reihenfolge genommen wurden.

Bei Kombinationen von verschiedenen Bedingungen muß man unterscheiden, ob diese voneinander abhängig sind. Ist dies der Fall, braucht nur die letzte abgefragt zu werden, ansonsten muß dies bei allen erfolgen.

AND

NOW...

Einsparung von Speicherplatz

Um Speicherplatz zu sparen, gibt es eine Menge Möglichkeiten. Als Speicherplatz ist hier allerdings nur der Speicher gemeint, den das Programm im Rechner belegt. Daraus folgt, daß die erste Möglichkeit die Datenablage auf Diskette ist. Tips hierzu entnehmen Sie bitte diesem Kapitel. Auch die strukturierte Programmierung sowie ML-Routinen können Speicherplatz sparen. Auch diesen Themen habe ich ein extra Kapitel gewidmet.

Hier aber ein paar Möglichkeiten, Speicherplatz einzusparen:

In einem Programm benötigt eine feste Zahl sieben Bytes. Kommt eine bestimmte Zahl mehrmals vor, so belegt sie jeweils sieben Bytes. Häufigstes Beispiel hierfür sind die Zahlen null bis drei. Aber auch viele andere Zahlen kommen oftmals vor. Das hängt natürlich ganz vom jeweiligen Programm ab.

Um diesen Speicher zu sparen, definieren Sie bitte am Anfang Ihres Programmes die Zahlen, die mehr als dreimal vorkommen, als Variablen und ersetzen dann diese Zahlen im Programm durch die entsprechenden Variablen. Also z.B. alle "128" durch "Q128". Wählen Sie die Variablennamen möglichst passend, um Verwechslungen zu vermeiden. Die Zahlen null bis drei sind sogar vom Turbobasic aus schon fest definiert, hängen Sie an sie einfach ein "%" an.

Eine weitere Methode, um Platz für Variablen einzusparen, ist die Definition von Konstanten. Diese Möglichkeit wird in normalem Turbobasic leider nicht geboten, also sind wir gezwungen, sie zu simulieren. Eine Konstante ist eine Variable, die stets einen festen Wert hat. Die Variable "Q128" aus obigem Beispiel, ist bereits eine Konstante. Derartige Konstanten können Sie bei einem Programm, das später compiliert werden soll, durch Schreiben in Speicheradressen ersetzen. Die Page 6 (ab 1536) sowie Page 4 (ab 1024), sofern Sie keine Kassettenoperationen durchführen, sind hierfür frei. Anstatt PAGE=144:CHS=PAGE*256 können Sie DPOKE 1536,144*256 schreiben. Mit diesem DPOKE-Befehl schreiben Sie gleichzeitig die Adressen 1536 und 1537. Möchten Sie nun PAGE wissen, schreiben Sie einfach PEEK(1537), bei CHS einfach DPEEK(1536). Achten Sie darauf, dies nur einzusetzen, wenn es sich um konstante Variablen handelt, denn die Addier- und Subrahier-Operationen mit diesen Adressen fressen den gesparten Speicher wieder.

Wird im Programm ein umdefinierter Zeichensatz verwendet, kann man diesen auch platzsparend ablegen. Wenn der Zeichensatz nicht auf Diskette gespeichert werden darf, z.B. weil das Programm eingetippt werden soll,

NO. 1

bietet sich folgende Möglichkeit an:

Im Normalfall besteht die Zeichendefinition aus eine READ- und POKE-Schleife sowie vielen DATA-Zeilen mit den Zeichensatzdaten. Dies können Sie durch einen MOVE ADR("..."),CHS+X,Y ersetzen. Nehmen wir an, Sie haben zehn verschiedene Zeichen definiert. "X" stellt hier das Startzeichen*8 dar, Y die Anzahl der Zeichen*8. Anstatt der Punkte werden hier die Daten in Form von ASC-Codes eingesetzt. Das "Einlesen" der Daten erfolgt dadurch auch wesentlich schneller.

Stellen wir uns vor, wir haben ein Spiel programmiert, in dem die Zeichensatzgrafik relativ komplex ist. Das eigene Männchen kann also über zehn verschiedene Hintergrundzeichen wandern, fünf verschiedene Gegenstände nehmen, an zehn verschiedenen tödlichen Gegenständen sterben und es gibt zwanzig verschiedene Wand-Zeichen, durch die man nicht hindurch gehen kann. Es wäre Wahnsinn, jetzt im Programm alle Zeichen einzeln abzufragen: IF Z=3 OR Z=53 OR Z=47 OR Z=164 OR Z=... usw. die Zeilen nähmen kein Ende. Um sich dies zu ersparen gibt es einen einfachen Trick:

Notieren Sie sich, welche ASC-Codes welche Funktion haben. Schreiben Sie danach ein kleines Programm, das einen String mit 256 Zeichen Länge dimensioniert. Legen Sie für jede Funktion eine Zahl fest. Z.B. 0=Hintergrund, 1=Gegenstand, 2=Monster, 3=Mauer usw. Schreiben Sie nun an jede Stelle des Strings die entsprechenden Ziffern. Soll z.B. ASC-Code 24 ein Hintergrund sein, schreiben Sie LOC$(25,25)=CHR$(0). Die 25 entsteht, weil die ASC-Codes von 0-255 gehen, während die String-Positionen von 1-256 gehen. Sie müssen also jeweils eins hinzuzählen. Auf diese Weise schreiben Sie an alle Stellen des Strings, dessen ASC-Wert eine Funktion haben soll, das entsprechende Zeichen. Am besten geht dies mit einer kleinen Unterroutine. Im Programm muß nun nach dem LOCATE X,Y,Z noch ein Z1=ASC(LOC$(Z+1,Z+1)) eingesetzt werden. Um festzustellen, welche Funktion der Character hat, müssen Sie jetzt nur noch Z1 auf 0, 1, 2 oder 3 prüfen. Auch diese Methode spart nicht nur Speicher, sondern auch Zeit.

Mit solchen Tabellen in Strings lassen sich übrigens sehr viele Probleme lösen. Um z.B. eine schnelle Kreisroutine zu programmieren, speichern Sie einfach vorher die benötigten SINUS-Werte in einen String ab und greifen dann darauf zu.

THE LARCH

Geschwindigkeitssteigerung

Im letzten Kapitel haben wir bereits einige Möglichkeiten kennengelernt, die nicht nur Speicherplatz sparen, sondern auch die Geschwindigkeit eines Programmes erhöhen. Im allgemeinen erhöhen alle Methoden, die den Speicheranspruch verringern, auch die Geschwindigkeit, weil ein kürzeres Programm einfach schneller abläuft, als ein langes.

Eine Methode, um die Geschwindigkeit in Spielen zu steigern, ist die bereits erwähnte, anstatt PLOT oder PRINT lieber POKE zu verwenden. Im allgemeinen ist ein POKE-Befehl immer schneller, als ein spezieller Basic-Befehl, da dieser immer erst in diverse POKEs umgesetzt werden muß. Versuchen Sie also, lieber mehrere POKEs anstatt eines SETCOLOR-, PLOT-, PRINT-, SOUND- usw. Befehles einzusetzen.

Ein sehr interessanter Befehl in Turbobasic ist auch der MOVE-Befehl. Mit ihm können Sie viele Dinge sehr schnell erledigen. Anstatt PRINT "blablabla" können Sie auch einen MOVE-Befehl anwenden. Ein Player kann gesetzt werden, alle Farbregister können superschnell verändert werden, sogar ein einfaches Scrolling ist damit möglich. Experimentieren Sie damit!

Levelspeicherung

Für die Speicherung verschiedener Level im Programm gibt es mehrere Möglichkeiten.

Besteht ein Level nur aus anderen Daten für Monster, Zeitlimit usw., kann man einfach für jeden Level eine DATA-Zeile angeben, die die entsprechenden Informationen enthält.

Besteht jeder Level aus einem anderen Bildschirmaufbau, kann man diesen ebenfalls in DATA-Zeilen speichern, die dann eingelesen und auf den Bildschirm gebracht werden.

Weiterhin gibt es mehrere Möglichkeiten, um viele Level in wenig Raum unterzubringen:

Wenn mehrere Level gleiche Elemente enthalten, kann man verschiedene Unterroutinen anfertigen, die bestimmte Elemente zeichnen. Ein Level besteht dann nur noch aus einer DATA-Zeile, die angibt, welches Element wohin gesetzt wird. Dies zeigt das Programm LEVEL1.TB.

Bestehen die Level nur aus wenigen Linien, können diese auch durch PLOT und DRAWTO gezeichnet werden. Auch hier genügen ein- oder mehrere DATA-Zeilen. Als Beispiel hier das Programm LEVEL2.TB. Hier wurde ein Interpreter geschrieben, der die DATA-Zeilen in PLOT- und DRAWTO-Befehle umwandelt.

Platzsparend ist auch die Verwendung von "Blöcken". Dies ist ähnlich dem Element-Prinzip (siehe oben). Hier werden einfach verschiedene Blöcke definiert, aus denen der Bildschirm zusammengesetzt werden kann. Diese werden ebenfalls per DATA-Zeilen eingelesen. Beispielprogramm LEVEL3.TB.

Datenablage auf Diskette

Wenn Ihr Programm nicht unbedingt als Listing eintippbar sein soll, ist es sinnvoll, möglichst viele der Daten, die vom Programm benötigt werden, auf Diskette abzulegen. Dies kürzt einerseits das Programm und steigert andererseits die Geschwindigkeit. Außerdem wird dadurch Platz für neue Features geschaffen.

Doch was kann man nun alles auf Diskette ablegen? Als erstes die verschiedenen Level des Spieles. Wenn diese mit dem Grafikeditor (siehe TOOLS) erstellt werden, liegen sie auch direkt im richtigen Format vor. Dieses kann dann einfach eingelesen werden. Das Beispielprogramm DISK1.TB erledigt diese Aufgabe. Nach Eingabe des Grafikmodes und des Filenamens liest das Programm das entsprechende File ein und stellt es als Bildschirm dar.

Weiterhin kann man alle Daten, die das Programm doppelt belegt, auf Diskette ablegen. Doppelt belegt sind Daten, wenn sie einerseits im Programm stehen, und vom Programm auch noch an eine andere Stelle im Speicher geschrieben werden. DATA-Zeilen sind z.B. immer doppelt belegt. Abspeichern lassen sich unter anderem:

- Zeichensatz Daten
- Maschinenroutinen Daten
- Locate-String Daten
- Musikdaten
- Highscoreliste Daten

uvm.

Achten Sie beim Ablegen von Daten auf Diskette allerdings darauf, daß Ihr Programm nicht zu oft oder zu lange nachlädt, da dies den Spielfluß behindern würde.

ML-Routinen

ML ist hier die Abkürzung für Machine-Language und heißt auf deutsch Maschinensprache. Gemeint sind Assembler-Unterroutinen, die den Programmfluß bei zeitkritischen Operationen antreiben sollen.

Für den Einsatz von ML-Routinen ist es wichtig, erst einmal zu überlegen, ob und wo eine Assembler-Routine nötig ist. Wer jetzt z.B. anfängt, seine Highscoreliste als Assemblerroutine zu programmieren, kann sich gleich begraben. Derartige Dinge lassen sich mit dem Basic-Befehlssatz weitaus günstiger erledigen. Assembler-Routinen sollte man dort einsetzen, wo sie benötigt werden: Bei zeitkritischen Operationen. Soll z.B. in einem Boulderdash-Spiel der ganze Bildschirm nach Steinen durchsucht werden, die sich dann nach unten bewegen sollen, ist hierfür eine ML-Routine unersetzlich.

ML-Routinen kann man aber nicht nur aus Geschwindigkeits-, sondern auch aus anderen Gründen einsetzen. Soll eine bestimmte Sache z.B. parallel zum Spielgeschehen ablaufen, ist eine Vertikal-Blank ML-Routine sehr günstig. Bestes Beispiel ist hier z.B. die Hintergrundmusik oder das Umschalten von mehreren Zeichensätzen.

Sollten Sie Assembler beherrschen, dürfte es für Sie kein Problem sein, entsprechende Routinen zu programmieren. Ich empfehle Ihnen aber auf keinen Fall, das gesamte Spiel im Assembler zu schreiben, außer, wenn Sie zu viel Zeit haben. Der gezielte Einsatz von ML-Routinen in TB-Programmen ist dagegen viel praktischer und vor allem einfacher. Das Ergebnis ist das gleiche.

Für diejenigen, die Assembler nicht beherrschen, empfehle ich den Assembler-Kurs in ZONG zum Erlernen oder die Verwendung von vorgefertigten Routinen. Solche finden Sie in Kürze in einer neuen Serie in ZONG.

Strukturierte Programmierung

Wer kennt es nicht, das Programm, das nach der 10. Überarbeitung in ein totales Chaos verwandelt wurde, bei dem man erst nach langem Suchen einen Fehler finden kann?

Um dies zu vermeiden, sollten Sie von Anfang an strukturiert programmieren. Hierzu stellt Ihnen das Turbobasic einige sehr nützliche Befehle zur Verfügung. Als erstes einmal die beiden Minuszeichen, die eine ganze Zeile ergeben. Damit lassen sich Programmabschnitte gezielt voneinander trennen.

Als nächstes sollten Sie die Befehle GOTO und GOSUB aus Ihrem Programm streichen. Ersetzen Sie GOSUB durch EXEC und Prozeduren, ersetzen Sie GOTO durch strukturierte Schleifen. Das Turbobasic stellt Ihnen hierzu REPEAT-UNTIL, WHILE-WEND sowie DO-LOOP zur Verfügung.

Arbeiten Sie mit Prozeduren. Anstatt einer Hauptschleife, die zweihundert Zeilen lang ist, weil Sie alle Bewegungen, Zeitlimits und Sounds beinhaltet, sollten Sie diese in viele kleine Prozeduren aufteilen. Nicht nur eine Prozedur, die mehrmals aufgerufen wird, erfüllt einen Zweck. Prozeduren, die nur einmal aufgerufen werden, tragen viel zur Übersichtlichkeit des Programmes bei.

Überarbeiten Sie Ihr Programm: Die IF-ELSE-ENDIF Bedingung des Turbobasic macht Ihr Programm bei richtigem Einsatz viel übersichtlicher und kürzer.

Versuchen Sie, für strukturierte Befehle immer eine ganze Programmzeile zu reservieren. Also kein IF, ELSE, ENDIF, REPEAT, UNTIL, DO, LOOP, WHILE, WEND usw. zusammen mit anderen Befehlen in einer Zeile. Der Interpreter dankt es ihnen, indem er die Programmzeilen nach ihrer Struktur einrückt und das Programm dadurch viel lesbarer wird.

Ein Programm ist nur strukturiert genug, wenn Sie es nach vier Jahren noch einmal hervorkramen können und wieder durchblicken, ohne sich durch ein Chaos durchzuwuseln.

Effekte

Als erstes beginnen wir mit der Synthetisierung von Klängen. Grundsätzlich sollten Sie sich darüber im klaren sein, welchen Sound Sie umsetzen möchten, bevor Sie sich daran setzen. Es reicht hier allerdings nicht aus, zu wissen, daß es sich z.B. um eine Explosion o.Ä. handelt, sondern der gewünschte Klang sollte zur Verfügung stehen. Eine Möglichkeit hierzu ist eine Platte, Kassette usw, auf der sich irgendwo der gewünschte Klang befindet. Eine andere Möglichkeit ist die eigene Vorstellungskraft oder das eigene Mundwerk, welches viele Klänge sehr realistisch wiedergeben kann.

Nun folgt die erste schwierige Aufgabe: Der Klang muß genau analysiert werden. Dies erfolgt nach folgenden Gesichtspunkten:

- Grundklang
- Lautstärkenverlauf
- Frequenz, d.h. Tonhöhenverlauf

Dies erfordert natürlich ein gebildetes Gehör oder entsprechende Übung.

Der eigentliche Grundklang beschreibt beim ATARI-Soundbefehl einfach die Distortion, d.h. die Verzerrung der Rechteckfrequenz, wobei ein Klavier eindeutig mit Distortion zehn und eine Explosion mit Distortion acht ablaufen müßte.

Den Lautstärkenverlauf sollte man sich in einem Diagramm notieren, wobei die X-Achse die Zeit und die Y-Achse die Lautstärke darstellt. Mit dem Frequenzverlauf sieht es ganz ähnlich aus, denn auch hier ist ein Diagramm sinnvoll. Man sollte sich auf alle Fälle über eine Grundfrequenz, von der der Klang dann entsprechend abweicht, im klaren sein.

Bei sehr komplexen Geräuschen kann man zusätzlich zur Lautstärke und Frequenz während des Klanges auch noch die Distortion verändern. Bei derartigen Experimenten ist allerdings einiges Tüfteln notwendig.

Eine einfache Möglichkeit, gute Soundeffekte zu erstellen, liegt darin, den Soundeditor zu benutzen. Wie das vor sich geht, entnehmen Sie einfach der Beschreibung.

Natürlich können Sie auch durch Experimentieren auf gute Soundeffekte kommen. Das Programm SOUND.TB zeigt Ihnen einige Effekte, jeweils im Programm durch Kommentare beschrieben, die mit einfachen Mitteln erzeugt

werden können.

Where they

make

the watches ...

Musik

Bevor wir anfangen, Musik auf den Computer umzusetzen, müssen wir klarstellen, daß dies sehr schwierig ist, sofern wir nicht in irgendeiner Form musikalische Kenntnisse haben. Haben wir die nicht, können wir sie uns trotzdem noch aneignen. In Musikgeschäften gibt es vielerlei Lehrbücher. Man sollte Grundkenntnisse des Notenlesens haben, außerdem irgendein Instrument, um seine Lieder auszuprobieren.

Nachdem wir uns nun für ein Lied entschieden haben, das wir umsetzen möchten, benötigen wir zunächst einmal Noten davon. Musikalische unter Ihnen können sich diese einfach selbst schreiben, die anderen sollten sie sich irgendwo (siehe oben) kaufen, bzw. leihen.

Am einfachsten gestaltet sich die Einbindung eines Musikstückes, wenn wir einen der vielen Musikeditoren verwenden. Soll dies nicht geschehen, müssen wir selbst eines programmieren.

Als nächstes ist es wichtig zu wissen, daß wir für ein Computerlied ein anderes Arrangement benötigen. Ersteinmal sollten wir uns entscheiden, wiviele Tongeneratoren wir für die Musik opfern wollen und wieviel Speicherplatz vorhanden ist.

Im Falle eines Tongenerators sollte dieser für die Melodie verwendet werden. Bei zweien kommt wahlweise Schlagzeug oder eine Begleit- oder Bassstimme hinzu. Der dritte und vierte kann ebenso belegt werden. Natürlich kann man auch einen Tongenerator für einen Halleffekt opfern. Dies hängt nun wieder ganz vom Musikstück und Ihrem Geschmack ab.

Bei der Programmierung Ihrer Musik sollten Sie die Hauptschleife auf den kleinsten Notenwert aufbauen. Längere Töne werden dann einfach gehalten, indem eine Date dies veranlaßt. Das Programm MUSIK1.TB veranschaulicht dieses Prinzip.

Um sich das Eingeben von vielen vielen Daten zu ersparen, müssen wir nun versuchen, gleiche Teile des Musikstückes in Sequenzen abzulegen, die dann immer aufgerufen werden. Diese Sequenzen sollten am besten immer einen Takt lang sein und für jeden Tongenerator einzeln in einem String gespeichert werden. Demnach muß das Abspielprogramm pro Tongenerator und Takt immer nur eine Date lesen, die angibt, welcher Takt gespielt wird. Auf diesem Prinzip beruhen auch die meisten Musikeditoren. Hierdurch wird es möglich, ein sehr langes Musikstück mit wenig Programmier- und Speicheraufwand zu

realisieren.

Laden Sie von der Programmdiskette einfach das File MUSIK2.TB. Das Musikstück aus dem ersten Beispiel wurde hier etwas aufwendiger umgesetzt.

Die Initialisierung der Strings für Drums, Bass und Sequenz erfolgt in Zeile 10-40. Die Drums spielen drei verschiedene Sounds: Bassdrum mit Frequenz 100 und Distortion 12 bei sehr kurzer Hüllkurve, Snaredrum mit Frequenz 20 und Distortion acht bei kurzer Hüllkurve und Hihat mit Frequenz eins und Distortion acht bei sehr kurzer Hüllkurve. Für jeden Takt zählt ein Counter, der die Stelle in den Begleitstimmen angibt, von 1-16. Da die Melodie in Distortion zwei gespielt werden soll, muß das Audiocontrolregister-Bit sieben gesetzt werden. Die Hauptschleife beginnt dann in Zeile 60 und reicht bis Zeile 250. Als erstes wird ein Wert aus dem Bassnotenstring gelesen und interpretiert, d.h., wenn er ungleich Null ist, wird er als Tonhöhe übernommen und die Lautstärke neu gesetzt. Wenn er gleich eins ist, wird die Lautstärke auf Null gesetzt, damit die Töne an bestimmten Stellen ausklingen. Bei den Drums läuft es ähnlich, allerdings gibt es hier drei Möglichkeiten (d.h. Instrumente), die auch jeweils andere Distortion-, Hüllkurven- und Lautstärkenwerte haben.

Für die Sequenz- und die Melodiestimme gilt das gleiche, allerdings werden die Daten für die Melodiestimme aus den DATA-Zeilen gelesen.

Die nun folgende FOR-NEXT Schleife wird sechsmal durchlaufen und aktualisiert die jeweiligen Werte in den Distortion- und Lautstärke-Registern. Außerdem werden die Lautstärkenwerte entsprechend dekrementiert. Die IF-THEN-ELSE Abfragen sind notwendig, um eine gleichmäßige Bearbeitung, unabhängig vom Ausgang der Entscheidung, zu gewährleisten. Dies ist natürlich bei einem Musikstück extrem wichtig, da es sich sehr komisch anhören würde, wenn die Musik je nach Tondichte langsamer oder schneller werden würde.

Probieren Sie einfach aus, das Programm zu ändern, ein anderes Musikstück daraus zu machen und dann einmal ein eigenes Programm zu schreiben.

Levelanwahl

Ein wichtiges Feature bei Spielen, die viele Level bieten, ist die Levelanwahl. Wenn wir ein Spiel mit 50 Leveln haben, die alle einzeln schon sehr schwierig zu schaffen sind, ist es einfach frustrierend, immer wieder von vorne anfangen zu müssen.

Für eine Levelanwahl bieten sich mehrere Möglichkeiten: Entweder kann jeder Level einzeln oder nur bestimmte Levelgruppen, z.B. jeder fünfte oder jeder zehnte angewählt werden.

Für die Wahlmöglichkeit gibt es wieder mehrere Möglichkeiten: Man kann z.B. von Anfang an jeden Level anwählen können oder nur die anwählen dürfen, die man bereits einmal erreicht oder geschafft hat.

Um dies zu ermöglichen, gibt es wieder mehrere Möglichkeiten: Entweder, man speichert auf Diskette ab, welcher Level bereits erreicht wurde, oder man vergibt Codewörter, die man vor Spielbeginn eingeben kann. Es ist auch möglich, den Spieler dazu zu zwingen, bei jedem Neuladen von vorne anfangen zu müssen.

Die Programmierung ist natürlich ganz einfach. Man benötigt nur einen Zähler, der die höchste erreichte Levelnummer angibt. Im Titelbild kann man dann diesen Level per Joystick oder Tastatur einfach anwählen.

Codes

Die Codes sind eine Möglichkeit der Levelanwahl, die immer mehr Beliebtheit findet. Codes haben einfach den Vorteil, daß man jederzeit in jedem beliebigen Level einsteigen kann, sofern man ihn bereits einmal erreicht hat.

Die Programmierung von Codes ist natürlich ebenso einfach: Im Programm oder auf der Diskette wird eine Tabelle abgelegt, die die verschiedenen Codes enthält. Beim Erreichen eines Levels wird der entsprechende Code auf dem Bildschirm angezeigt. Vor Spielbeginn kann man durch Eingabe eines Codes, der dann mit der Tabelle verglichen wird, einen Level anwählen.

Für die Verwendung von Codes ist es wichtig zu wissen, daß dies sehr vielen Spielern die Möglichkeit zum Schummeln bietet, da sie einfach mit einem Diskmonitor die Codes entdecken können. Um dies zu vermeiden, sollte man versuchen, seine Codes auf irgendeine Art zu verschlüsseln. Am einfachsten ist es, eine bestimmte Zahl zum ASC-Code hinzuzuzählen, schwieriger wird es, wenn man zu jedem Code einen anderen ASC-Wert addiert, oder noch besser zu jedem Buchstaben des Codes.

Im Übrigen sollte ein Code nicht kürzer als vier Buchstaben/Ziffern sein, damit man ihn nicht auch zufällig herausbekommen kann. Codes, die länger als sechs oder sieben Buchstaben/Ziffern sind, frustrieren natürlich bei der Eingabe.

Highscoreliste

Eine Highscoreliste ist für ein Spiel ein wichtiges Feature, wenn es nicht durch viele Level oder ein Adventureelement dazu reizt, weiter zu kommen. Doch auch wenn dies der Fall ist, ist eine Highscoreliste immer ein gern gesehenes Feature, da man seine Bestleistungen immer verewigt sieht.

Die kleinste Version einer Highscoreliste besteht darin, einfach den höchsten erreichten Score abzuspeichern. Hierzu wird einfach nach Spielende eine Variable HIGHSCORE (oder so ähnlich) mit dem aktuellen Score verglichen und übernommen, wenn der Score größer ist.

Als erste Zugabe kann man den Highscore auf Diskette abspeichern. Dies kann einfach durch ein OPEN, PRINT auf Diskette und CLOSE erfolgen. Natürlich darf man nicht vergessen, den Highscore beim Programmstart einzuladen.

Die nächste Zugabe ist die Eingabe eines Namens. Hierzu kann man entweder ein einfaches INPUT oder eine GET-Schleife verwenden, besser ist es natürlich, einen schön gestylten Bildschirm, der alle Zeichen enthält, zu entwerfen, auf dem man dann per Cursor einige eintragen kann. Dies benötigt natürlich wieder mehr Speicher.

Die umfangreichste Highscoreliste ist dann die, bei der mehrere Plätze vorhanden sind. Im Mindestfall sind dies fünf, meist sind es zehn, in seltenen Fällen sogar bis zu 100. Dies hängt ganz vom Geschmack des Programmierers und dem freien Speicher ab.

Eine universelle Routine für eine Highscoreliste finden Sie unter dem Namen HIGH.TB. Diese können Sie je nach Bedarf abändern:

ANZ gibt die Anzahl der Plätze der Liste an. SLN ist die Anzahl der Stellen für die Punktezahl und EIN die Anzahl der Stellen für den Namen. Zeile 66 zeigt die Tabelle auf dem Bildschirm, Zeile 70 setzt eine Punktezahl. In den Zeilen 1000-1050 kann dann die "Namen eintragen"-Routine stehen. 10050-10055 initialisiert die Tabelle, was entfallen kann, wenn die Tabelle von Diskette nachgeladen wird (Zeile 10060-10090). Zeilen 11160-11180 speichern die Tabelle ab.

Dieses Programm sollten Sie nun nach Ihren Wünschen abändern, bzw. verbessern und können es dann in Ihren eigenen Programmen verwenden.

Actionspiele

Bei der Programmierung von Actionspielen ist der wichtigste Punkt, auf den man achten muß, der Spielfluß. Da bei Actionspielen meist geschossen wird, gilt es, die vielen Objekte, die bewegt werden müssen, in einer annehmbaren Geschwindigkeit zu animieren. Dies dürfte in Turbobasic nur unter Anwendung von Tricks möglich sein. Als Tricks stehen z.B. zur Verfügung:

Feindliche Objekte können mit einer einfachen Logik bewegt werden. Diese ist in der Abarbeitung sehr schnell.

Feindliche Objekte können abwechselnd bewegt werden. Also: Das eigene Objekt, dann der erste Feind, dann das eigene Objekt, dann der zweite Feind usw.

Als weitere Möglichkeit bietet sich natürlich der Einsatz von ML-Routinen an. Wie man die einzelnen Probleme nun löst, hängt natürlich wieder vom Spiel ab, das man programmieren möchte.

Ein schönes Beispiel für ein Actionspiel ist SABOTAGE. Hier wirden neben dem eigenen Männchen nur noch der feindliche Schuß bewegt. Schießt man selbst, bleibt der Spielfluß kurz stehen, was aber kaum auffällt, da dies sehr schnell geht. Der feindliche Schutzschirm sowie die Roboter werden nicht direkt bewegt, sondern nur gesetzt/gelöscht. Trotzdem entsteht hier ein actionreiches Spielgeschehen.

Climb & Jump Spiele

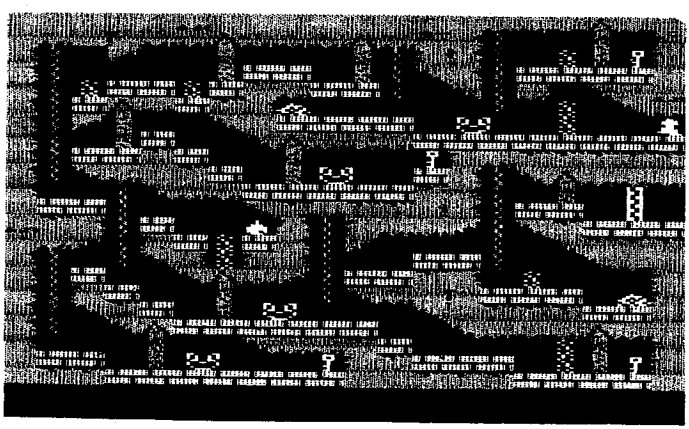
Climb & Jump Spiele sind dadurch characterisiert, daß man das Spielgeschehen von der Seite sieht. Daraus folgt, daß man beim Entwurf der Grafik umdenken muß.

Wichtig ist bei der Programmierung, daß die Bewegung noch relativ schnell vonstatten gehen muß, da dies den Spielfluß und gleichzeitig den Spielspaß bremsen würde.

Sie sollten ebenfalls darauf achten, die Steuerung für den Spieler möglichst einfach zu gestalten. Es ist wichtig, daß die einzelnen Funktionen, wie z.B. Springen, schnell und problemlos ausgeführt werden können.

Die verschiedenen Level Ihres Spieles sollten sich möglichst durch einige Aspekte voneinander unterscheiden: Grafik, Aufgabe, Feinde, Aufbau. Zwanzig Level mit der gleichen Grafik und den gleichen Feinden sowie der gleichen Aufgabe durchzuspielen macht nur in den seltendsten Fällen Spaß.

Ein schönes Beispiel für Climb & Jump Spiele ist GET UP. Das Spiel gewinnt dadurch an Reiz, daß es einerseits sehr schnell abläuft und man andererseits zu zweit spielen kann. Gegen einen menschlichen Gegner zu spielen, bietet immer einen gewissen Reiz. Die verschiedenen Level werden hier per Zufall ausgewählt, ein Spiel gleicht also nie dem vorherigen.

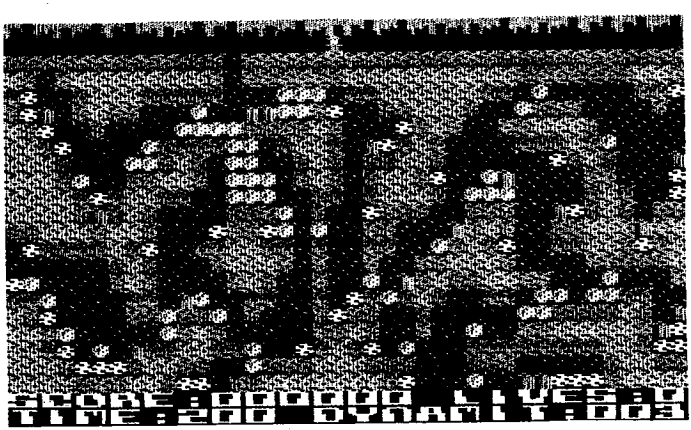


Labyrinthspiele

Wenn Sie ein Labyrinthspiel programmieren, sollten Sie sich ein wirklich gutes Konzept dazu überlegen. Noch eine Pacman Variante ist zwar schön, lockt aber niemanden hinter dem Ofen hervor. Das Original Pacman war ein wirklich gutes Spiel. Dies lag aber hauptsächlich daran, daß es überaus flüssig spielbar war. Ein paar neue Spielelemente und das Spiel hätte den zehnfachen Reiz. Schlüssel und Türen, Gegenstände, die man benutzen kann, Blöcke, die man verschieben kann usw. sind inzwischen gängige Elemente für Labyrinthspiele.

Das Spiel ZAUBERWALD II ist genau genommen ein Labyrinthspiel. Der Bildschirmaufbau sieht zwar nicht sehr nach einem Labyrinth aus, aber der Spielablauf ist eindeutig: Fresse einen Gegenstand ohne das Monster zu berühren. Als Gag kann man hier das Monster beschießen und bestimmte Mauern aufbrechen.

STONE MINE II sieht schon eher nach einem Labyrinth aus. Aber auch hier wurde als Gag so einiges eingebaut: Man kann Steine wegsprengen, Dynamit sammeln und darf bestimmte Wände nur begrenzt berühren. Manche Wände kann man sogar durch Berühren beseitigen, allerdings auch nur begrenzt. In diesem Spiel sind bereits einige Denkspielelemente eingebaut.

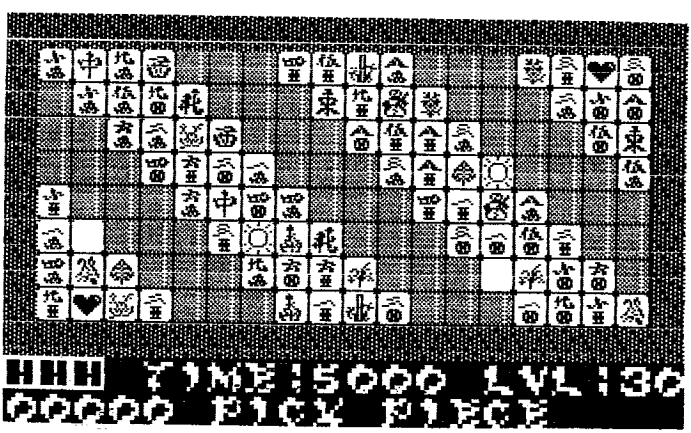


Denkspiele

Wie bereits bei den Labyrinthspielen erwähnt, finden in letzter Zeit die Denkspiele immer größeren Anklang. Für ein Denkspiel ist es einfach wichtig, ein gutes Konzept zu haben, daß interessante Elemente bietet. Verschiebbare Blöcke, Drehtüren, Löcher, Schlüssel usw. sind die üblichen Elemente. Das heißt nicht, daß diese bereits langweilig sind. Im Gegenteil: Auf die richtige Kombination kommt es an.

Als zweiter wichtiger Punkt ist zu nennen, daß die Grafik dieser Denkspiele, da sie meist abstrakt ist, möglichst ansprechend oder sogar comicmäßig gestaltet wird. Übersichtlichkeit ist hier sehr wichtig, da sonst der Denkprozess sehr behindert wird.

Die Spiele LEAP, MATCH und DIAMONDS sind Vertreter dieser Kategorie. Jedes der Spiele hat ein komplett anderes Konzept, das etwas von den üblichen Mischungen aus Labyrinth- und Denkspielen abweicht. Wichtig sind hierbei nicht unbedingt eine superschnelle Animation und hervorragende Soundeffekte, sondern eine herausfordernde Aufgabe.



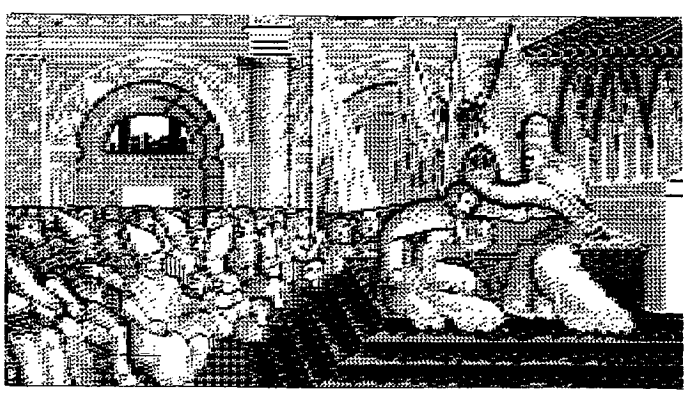
Strategiespiele

Wenn Sie ein Strategiespiel programmieren wollen, ist hier die schwierigste Aufgabe das Konzept. Hierbei müssen Sie nämlich schon vorher wissen, welche Faktoren welche Spielelemente beeinflussen. Die Programmierung ist eigentlich nicht sehr schwierig, besonders, da hier keine richtige Animation und kaum zeitkritische Operationen vorkommen. Sie sollten darauf achten, daß die Beeinflussung der verschiedenen Faktoren einigermaßen realistisch ist.

Fangen Sie doch einfach mal mit dem typischen HAMURABI-Spiel an. Sie sind der Herrscher, können Korn ernten und an die Leute verteilen und müssen so versuchen, die Leute bei Laune zu halten. Nun können Sie anfangen, mit den verschiedenen Faktoren zu experimentieren. Was passiert, wenn die Leute nicht satt werden, wieviel Korn benötigen Sie usw. Klappt das alles, können Sie weitere Faktoren einbauen: Armeen, Soldaten, Land, Fabriken usw. Noch eine schöne Grafik hinzu und schon haben wir ein Spiel wie Kaiser.

Das Spiel WUMPUS ist ein Spiel der Gattung, wenn auch ein etwas obskures. Durch taktisches Planen und Auswerten der gegebenen Informationen gilt es, die Aufgabe zu lösen. Hier wurde zusätzlich noch Grafik eingebaut, die das Spielgeschehen realistischer macht.

Beim Spiel STERNENHAUFEN handelt es sich eher um einen typischen Vertreter. Hier müssen Raumschiffe gebaut und Planeten erobert werden. Schauen Sie sich einfach einmal beide Programme an und Sie werden den Aufbau leicht erkennen können.



Simulationen

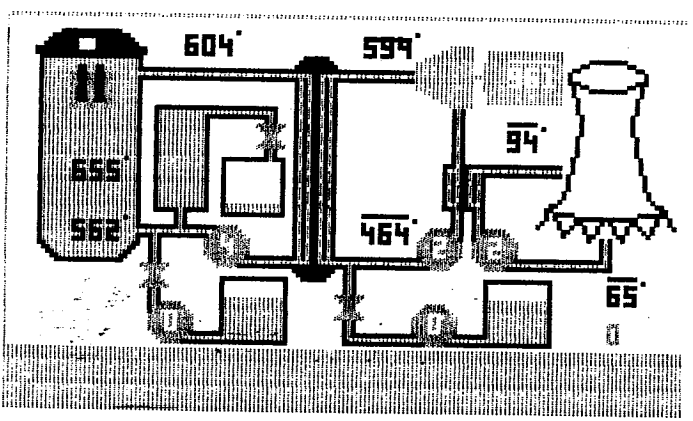
Die Simulationen sind die am schwierigsten zu realisierende Spielegattung. Der Programmaufbau entspricht in etwa dem eines Strategiespieles, allerdings kommt es hier gleichzeitig noch auf eine einigermaßen schnelle Grafik an.

Um diese Grafik zu erhalten, kann man natürlich auf verschiedene komplexe Faktoren verzichten, allerdings wird dadurch das Spiel wieder mehr zu einem Actionspiel. Es kommt also darauf an, die richtige Mischung zu finden.

Wie bei den Strategiespielen ist es auch hier wichtig, die Faktoren möglichst realistisch zu berechnen und umzusetzen.

Die typischen Vertreter der Simulationsspiele sind die Flugsimulatoren, U-Boot Simulationen, Auto (Renn) Simulationen. Diese können natürlich auch jederzeit mit Elementen wie Kampf kombiniert werden.

Das Spiel FLUG 747 gehört zu diesen typischen Gattungen. Man soll versuchen, die Maschine sicher zu landen. Wenn Sie sich das Programm ansehen, wird Ihnen auffallen, wieviele Berechnungen allein hierzu nötig sind.



PM-Animator

Der PM-Animator ist ein sehr nützliches Tool, das es Ihnen ermöglicht, mehrfarbige Player selbst zu erstellen und in eigene Programme einzubinden. Er bietet die Möglichkeit, viele verschiedene Shapes zu bearbeiten, hat komfortable Funktionen wie Spiegeln, Verschieben usw. und bietet sogar die Möglichkeit, die Player zu animieren.

Die Bedienung des Programmes entnehmen Sie am besten der HELP-Funktion. Funktionen können über die Cursortasten (ohne Control) angewählt werden, RETURN löst eine Funktion aus.

Widmen wir uns nun der Einbindung in eigene Programme: Ein Shape des PM-Animators besteht aus zwei Playern, die übereinandergelegt werden, um drei Farben zu erzeugen. Um ein solches Shape in eigene Programme einzubinden, muß das Shape einfach von Diskette geladen werden, um dann in den entsprechenden RAM-Bereich kopiert zu werden. Dies geht z.B., indem Sie einen String mit Länge 64 dimensionieren, diesen zunächst mit Leerzeichen füllen und dann mittels

```
OPEN #1,4,0,"name.ext":BGET #1,ADR(..$):CLOSE #1
```

64 Shapedaten einladen. Nachdem danach die PM-Grafik initialisiert ist, kann man das Shape mit

```
MOVE ADR(...$),PMBASE+YPOS,32:MOVE ADR(...$),PMBASE+YPOS+256,32
```

auf dem Bildschirm erscheinen lassen.

Grafikeditor

Mit diesem Grafikeditor ist es möglich, bis zu vier verschiedene Zeichensätze gleichzeitig zu erstellen und mit diesen beliebige Spielfelder zusammenzustellen. Zur Vereinfachung können Blöcke aus bis zu vier Zeichen gleichzeitig editiert werden.

Die Auswahl der einzelnen Punkte der Menüs erfolgt durch Tippen der in Klammern stehenden Zahl. Die Steuerung des Cursors erfolgt über Joystick.

Im Hauptmenü haben Sie folgende Wahlmöglichkeiten:

(1) Modewahl
(2) Auswahlscreeneditor
(3) Charactereditor
(4) Screeneditor
(5) Disk Menü

Modewahl

Diese wird nach Starten des Programmes automatisch aufgerufen. In den folgenden Grafikmodi können Sie arbeiten:

(1) Modus 0: 40*24 Zeichen, Zeichen einfarbig, 2 Farben (1*Zeichenfarbe + 1*Hintergrundfarbe)

(2) Modus 1: 20*24 Zeichen, Zeichen einfarbig, 5 Farben (4*Zeichenfarbe + 1*Hintergrundfarbe)

(3) Modus 2: 20*12 Zeichen, Zeichen einfarbig, 5 Farben (4*Zeichenfarbe + 1*Hintergrundfarbe)

(4) Modus 12: 40*24 Zeichen, Zeichen vierfarbig, 5 Farben (4*Zeichenfarbe + 1*Hintergrundfarbe)

(5) Modus 13: 40*12 Zeichen, Zeichen vierfarbig, 5 Farben (4*Zeichenfarbe + 1*Hintergrundfarbe)

Auswahlscreeneditor

Im Auswahlscreeneditor, sowie im Screen- und im Zeichensatzeditor können Sie per Tastatur folgende Funktionen auslösen:

0-4: Zeichensatz umschalten (0=normaler Zeichensatz)

C + 0-4: Farbeeinstellung mit Hilfe des Joysticks vornehmen (Knopf=Beenden)

S + 0-4: Blockgröβe einstellen:

Size 1: Ein Zeichen. Size 2: Zwei Zeichen nebeneinander. Size 3: Zwei Zeichen übereinander. Size 4: Vier Zeichen, zwei nebeneinander und zwei übereinander.

Durch die Blockgröβe legen Sie nicht nur fest, wieviel Zeichen Sie auf einmal editieren oder setzen möchten, sondern auch, um wieviele Zeichen sich Ihr Cursor bewegt. Z.B. Size 3: Nach rechts und links bewegt er sich in Einer-, nach unten und oben in Zweier-Schritten.

Der Auswahlscreeneditor dient dazu, aus dem ASCII-Zeichensatz diejenigen Zeichen auszuwählen (und dementsprechend zusammenzusetzen), die für den jeweiligen Zeichensatz (bzw. für das jeweilige Spiel o.Ä.) benötigt werden. Dies hat den Vorteil, daβ man die Zeichen beim Editieren des Zeichensatzes oder der Spielfeldes nicht extra aus dem ASCII-Zeichensatz aussuchen muβ, sondern nur die wirklich benötigten schon richtig zusammengefaβt vor sich sieht.

Im Klartext: Der Auswahlscreen soll die Zeichen (oder Blöcke) zeigen, die umdefiniert werden, bzw. im Spielfeld vorkommen. Beim Zeichensatz- und Screeneditor können Sie die Zeichen bzw. Blöcke nur aus dem Auswahlscreen auswählen.

Der Auswahlscreeneditor besteht aus zwei Bildschirmen: Dem ASCII-Screen und dem Auswahlscreen.

Im ASCII-Screen können Sie mit der Taste P dem kompletten ASCII-Screen in den Auswahlscreen kopieren. Möchten Sie nur ein einzelnes Zeichen auswählen, fahren Sie mit dem Cursor auf das entsprechende Zeichen und drücken den Feuerknopf. Sie gelangen damit automatisch wieder in den Auswahlscreen.

Im Auswahlscreen können Sie Ihr Zeichen mit Hilfe des Joysticks an die Stelle setzten, die Sie möchten (Feuerknopf drücken zum Setzen des Zeichens). Mit "+" gelangen Sie wieder in den ASCII-Screen, mit "<" wird der komplette

Auswahlscreen gelöscht. Tippen Sie "I", um das Zeichen invers zu setzen.

Hat man den Auswahlscreen fertig zusammengestellt, gelangen Sie durch Drücken der RETURN-Taste wieder in das Hauptmenü zurück. Ein Beispiel-Auswahlscreen für Size 4 befindet sich mit auf der Diskette.

Zeichensatzeditor

Zuerst können Sie aus dem Auswahlscreen das umzudefinierende Zeichen (oder den Block) wählen. Das Anwählen geschieht durch Drücken des Feuerknopfes. Sie gelangen damit automatisch in den Editor. Die Punkte im Zeichen können nun durch den Feuerknopf gesetzt werden. Hat dieser Punkt bereits eine Farbe, erhält er nun die nächste Farbe. Möchten Sie das Editieren beenden, können Sie dies mit RETURN und gelangen wieder in den Auswahlscreen.

Dort ist es möglich Zeichen in andere Zeichen hineinzukopieren. Dazu tippt man "G" um das entsprechende Zeichen zu nehmen (get) und "P" (put) um den Inhalt dieses Zeichens (oder Blockes) in den Inhalt des ausgwählten Zeichens (oder Blockes) zu kopieren. Achtung: Kopieren Sie den Inhalt eines Zeichens in ein Leerzeichen, ist dieses anschließend gefüllt!

Sie können natürlich auch den Inhalt eines Zeichens (oder Blockes) von einen Zeichsatz in einen anderen kopieren. Dazu müssen Sie nur vor Tippen von "P" in diesen Zeichensatz wechseln (1-4). Zum Kopieren des gesamten Zeichensatzes in einen anderen tippt man einfach "M" (für "Move") und die Nummer des Zeichensatzes, in den man den momentanen Zeichensatz kopieren möchte. Achtung, der vorherige Zeichensatz wird dann natürlich überschrieben! Mit RETURN gelangt man wieder in das Hauptmenü.

Sreeneditor

Hier kann man die im Auswahlscreen befindlichen Zeichen und Blöcke zu einem Spielfeld zusammensetzen. Auch hier empfielt es sich, mit den Blöcken (Size 1-4) entsprechend oft zu arbeiten. Vom Spielfeldeditor gelangt man in den Auswahlscreen durch Tippen von "*". Hier nehmen Sie mit dem Feuerknopf das entsprechende Zeichen/Block. Sie gelangen damit automatisch wieder in den Screen-/Spielfeldeditor. Dort setzten Sie Ihren Block mit dem Feuerknopf an die entsprechenden Stellen. Beachten Sie die jeweilige Schrittweite Ihre Blockgröße. Zudem können Sie die Blockgröße (Size)

jederzeit verändern, ohne daß sie anschließend das Zeichen neu anwählen müssen. Das Programm behält sich immer alle vier möglichen Größen, sodaß Sie auch mal erst ein Zeichen und später vier Zeichen setzen können, wenn Sie wollen. Tippen Sie "F", wird der gesamte Bildschirm mit dem ausgewählten Block in der entsprechenden Schrittweite gefüllt!

Außerdem ist ein Animationmodus eingebaut. Hierbei können Sie wählen, in welcher Reihenfolge der Screen in den einzelnen Zeichensätzen (1-4) dargestellt werden soll. Dazu tippen Sie zuerst CONTROL-M. Nun können Sie bis zu zehn Ziffern (1-4) tippen. Mit RETURN können Sie eine Sequenz abschließen, wenn Sie kürzer als zehn Folgen lang sein soll. Um eine Animation von vier Zeichensätzen hintereinander zu erzeugen, wird also folgendes getippt:

<CONTROL-M> <1> <2> <3> <4> <RETURN>

Das Abspielen der Animation erfolgt durch Tippen von COTROL-A. Beim Abspielen kann die Geschwindigkeit per Joystick links/rechts in kleinen, per Joystick hoch/runter in großen Schritten beeinflußt werden. Ein Druck auf den Feuerknopf stoppt die Animation.

Wie immer gelangt man mit RETURN wieder in das Hauptmenü zurück.

Diskmenü

Die Directory bietet die Möglichkeit, die Einträge nach Extender gefiltert auszugeben.

(1) .PFD (Playfield = Spielfeld) zeigt Ihnen alle Screens an, die sich auf der Diskette befinden.

(2) .CHR (Character = Zeichensatz) zeigt Ihnen alle Zeichensätze an, die sich auf der Diskette befinden.

(3) .AWS (Auswahlscreen) zeigt Ihnen alle Auswahlscreens an, die sich auf der Diskette befinden.

(4) .* zeigt Ihnen alle Files an, die sich auf der Diskette befinden.

Beim Laden oder Speichern der Spielfelder, Zeichensätze oder Auswahlscreens ist es immer nur möglich, 8 Buchstaben für den Filenamen anzugeben, der Extender wird automatisch gewählt (s.o.).

Geben Sie keinen Filenamen an, wird der Punkt nicht mehr ausgeführt, sondern abgebrochen.

Beim Laden/Speichern von Zeichensätzen müssen Sie anschließend noch zusätzlich die Zeichensatznummer angeben, in die der zu ladende Zeichensatz geladen, bzw. welcher gespeichert werden soll.

Beim Speichern des Spielfeldes werden in die ersten fünf Bytes des Files die Farben abgespeichert. Zudem wird nach dem Laden und Speichern eines Spielfeldes der damit aktuelle Name des Files im Kopf des Diskmenüs angezeigt!

Mit (8) gelangen Sie wieder in das Hauptmenü.

Einbindung

Die Einbindung in eigene Programme gestaltet sich natürlich auch sehr einfach. Ein Zeichensatz wird in Form von 1024 Bytes abgespeichert. Diese können mit

```
OPEN #1,4,0,"D:name.CHR":BGET #1,chset,1024:CLOSE#1
```

geladen werden. Ein Playfield besteht aus fünf Farbbytes sowie der Anzahl der Zeichen pro Zeile * Anzahl der Zeilen an Zeichendaten. Für Grafikmodus 0 oder 12 also z.B. 960. Diese können mit

```
OPEN #1,4,0,"D:name.PFD":BGET #1,708,5:BGET #1,DPEEK(88),anz:CLOSE #1
```

geladen werden.

Soundeditor

Der Soundeditor dient dazu, auf einfache Weise relativ komplexe Sounds zu erzeugen, die ebenso einfach in eigene Programme eingebaut werden können.

Jeder Sound des Soundeditors baut sich aus drei Tabellen auf: Lautstärkentabelle (Volumetable), Verzerrungstabelle (Distortiontable) und Frequenzmodulationstabelle (Thmod). Weiterhin können Sie die Werte der Grundtonhöhe, Modulationstiefe und Geschwindigkeit einstellen. Beim Spielen eines Sound wird von der Grundtonhöhe ausgegangen. Nun werden die drei Tabellen gleichzeitig abgearbeitet, wodurch ein relativ komplexer Sound entstehen kann. Durch die Distortion-Tabelle ist es sogar möglich, zusammengesetzte Sounds zu erstellen. Ein Beispiel wäre eine fallende Bombe mit Aufschlag und Explosion.

Die Bedienung ist ganz einfach: In der oberen Bildschirmzeile steht der Name der Tabelle, die gerade bearbeitet wird. Mit dem Joystick wird der Cursor in der Tabelle bewegt, mit dem Feuerknopf wird ein Wert gesetzt. Das Wechseln zwischen den einzelnen Tabellen erfolgt durch Tippen der Taste "*". Die Volumetable hat oben die Einstellung "laut", unten "leise". Bei der Distortion gilt oben der Wert 14, unten 0. Die Pitchmodulationstable beeinflußt die Töne durch oben = tieferer Ton, unten = höherer Ton. Durch Tippen der Taste "G" und anschließendem Steuern mit dem Joystick wird die Grundtonhöhe eingestellt. Der Vorgang wird per Knopfdruck beendet. Mit der Modulationstiefe, d.h. dem Wert, um den die Tonhöhe beeinflußt wird, sowie der Geschwindigkeit wird in gleicher Weise verfahren. Hierzu dienen die Tasten "M" und "S". Die Taste "F" füllt die gesamte Tabelle in der aktuellen Cursorhöhe. Mit "T" für Test wurd der Sound abgespielt.

Der Soundeditor bietet die Möglichkeit, bis zu 50 Sounds zu editieren. Die Nummer des aktuellen Sounds kann durch Tippen von "N" (siehe oben) angewählt werden. Mit "D" gelangen Sie ins Disketten Menü, welches nach Auflisten der auf der Diskette befindlichen Soundfiles die Möglichkeit zum Speichern oder Laden von Sounds bietet. Sowohl beim Speichern als auch beim Laden ist die Nummer des Soundfiles anzugeben (Taste 0-9). Das Soundfile wird auf Diskette unter dem Namen SOUNDx.DAT abgelegt. Wichtig: Beim Speichern von Sounds wird nur bis zur aktuellen Soundnummer abgespeichert. Mit "Q" für Quit gelangen Sie in den Editor zurück.

Nun zum Einbinden der Sounds in eigene Programme: Dies ist recht einfach, denn auf der Diskette befindet sich eine Turbobasic-Prozedur, mit in eigene Programme eingebunden werden kann (mit LIST abspeichern und mit ENTER

dazuladen). Sie beginnt mit Zeile 10 und kann bei Bedarf RENUMberd werden. Das Soundfile muß bei Programmstart in einen String, der die Länge der Anzahl der Sounds * 83 hat und mit Leerzeichen gefüllt wurde, durch

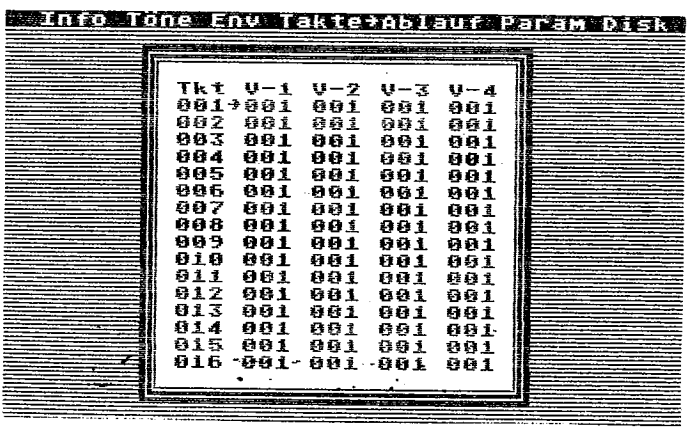
OPEN #1,4,0,"name.ext":BGET #1,adr(..$),anz*83:CLOSE #1

eingeladen werden.

Der Aufruf der Routine erfolgt dann einfach durch EXEC PLAY. Die Nummer des abzuspielenden Sounds (1-50) wird durch die Variable SNDNUM vorher festgelegt. Auch der Tongenerator (0-3), der den Sound spielen soll, kann festgelegt werden: Die Variable TG bestimmt diesen. Da die Sounds eventuell die Tabelle nicht bis zum Ende füllen und man auch keine Pause in Kauf nehmen möchte, kann man das Programm veranlassen, nur einen Teile des Sounds abzuspielen. Die Variable WEITE gibt an, bis zu welcher Stufe der Tabelle der Sound gespielt werden soll. Dieser Wert muß im Bereich 0-39 liegen. Die Prozedur benutzt außerdem noch folgende Variablen: PARA, TH, MUL, WTM, A, I. Diese sollten im übrigen Programm nicht vorkommen. Ein Aufruf kann also z.B. durch

SNDNUM=1:WEITE=10:TG=0:EXEC PLAY

erfolgen.



Sabotage

Die Regierung eines unbekannten Staates hat beschlossen, das geheime Hauptquartier der Dr. Bamuse zu serstören. Zu diesem Zweck hat sie dort eine große Zeitbombe plaziert. Da Dr. Bamuse sehr hellhörig ist, hatte er die Eindringlinge sofort enttarnt und das Zünden der Bombe damit verhindert.

Aus diesem Grund werden Sie als Spezialagent beauftragt, in das Hauptquartier eunzudringen und die Bombe zu zünden.

Das Quartier besteht aus sechs Räumen, die jeweils eine Verteidigungskanone gegen Fremdlinge sowie 10 Robo-Wachtürme aufweisen. Die Wachtürme können einfach abgeschossen werden. Solange allerdings die Kanone noch aktiv ist, entstehen laufend neue Wachtürme. Die Kanone besitzt ein undurchdringliches Schutzschild, welches nur kurz ausgeschaltet ist, wenn Sie schießt. Nur dann kann die Kanone zerstört werden. Solange allerdings noch mehr als fünf Wachtürme vorhanden sind, erscheint die Kanone nach kurzer Zeit wieder.

Schaffen Sie es, alle Türme sowie die Kanone zu eliminieren, öffnet sich der Raum und Sie können sich an den nächsten machen.

Im letzten Raum steht dann die Zeitbombe, die durch Anschießen aktiviert wird. Der Countdown zählt von 99 bis Null runter. In dieser Zeit müssen Sie wieder zurück zum Anfang des ersten Raumes gelangt sein. Vorsicht: Der sechste Raum öffnet sich erst, wenn Türme und Kanone eliminiert wurden, die übrigen Räume sind auf dem Rückweg offen. Daher erst alle Türme und die Kanone eliminieren, bevor Sie die Zetbombe aktivieren!

Die Steuerung erfolgt über einen Joystick in Port eins. Zu Beginn des Spieles stehen vier Leben zur Verfügung. Wenn Sie getroffen werden oder gegen einen Turm oder eine Wand rennen, verlieren Sie ein Leben. Sollten Sie alle Leben verlieren oder die Bombe explodieren, bevor Sie den Ausgang erreicht haben, ist das Spiel beendet.

Devil Sky

Ein Flugzeugunfall gehört wohl eher zu den unangenehmeren Dingen des Lebens. Besonders, wenn man selbst im Flugzeug sitzt.

Zum Glück hat irgendein Flieger Fallschirme erfunden, mit denen man sich retten kann. Leider haben diese den Nachteil, daß sie sich leicht vom Wind abtreiben lassen, und die umliegenden Felswände sehen überhaupt nicht freundlich aus. Natürlich sollte man auch nicht auf der verseuchten Wiese eines nahen Chemiewerkes landen und sich auch nicht von einem UFO kidnappen lassen (immer gesetzt den Fall, man hat nicht vergessen, früh genug aus dem Flugzeug auszusteigen)!

Alle oben genannten Dinge können Ihnen passieren, es sei denn Sie sind geschickt genug und landen auf der sich manchmal bewegenden Bodenplattform.

Nach Starten des Spieles mittels der START-Taste kommt von links ziemlich schnell ein Flugzeug angeschossen und zerschellt kurz darauf an der gegenüberliegenden Felswand. Inzwischen sollten Sie den Feuerknopf gedrückt haben, um abzuspringen, sonst ZONG!

Wenn alles geklappt hat, fällt Ihr Männchen nun sehr schnell nach unten. Bevor es jedoch eine zu harte Bekanntschaft mit dem Boden macht (ZONG), sollten Sie mittels nach oben Steuern des Joysticks den Fallschirm öffnen. In diesem Fall können Sie durch Steuern nach rechts oder links das Männchen auf seinem Weg nach unten gezielt auf die Plattform bewegen.

Achtung: Je später der Fallschirm geöffnet wird, desto mehr Punkte ergatteren Sie und desto weniger werden Sie vom Wind abgetrieben. Wird der Fallschirm allerdings zu spät geöffnet, bringt dies gar keine Punkte (ZONG)!

Zum Thema Wind: Aufpassen sollte man schon auf ihn. Seine Richtung und Stärke sind am unteren Bildrand durch ein grünes Quadrat dargestellt.

Sollten Sie (auf eine völlig unerklärliche Weise) gegen einen Felsen oder ein UFO stoßen: ... (Na was wohl)!

Vor Spielbeginn kann per SELECT-Taste der Schwierigkeitsgrad eingestellt werden:

A: Stehende Plattform, kein UFO

B: Bewegliche Plattform, kein UFO
C: Stehende Plattform, mit UFO
D: Surprise, Surprise ...

Nach einigen erfolgreichen Landungen fängt der Felsen auch noch an, sich selbstständig zu machen.

Zum Programm ist zu sagen, daß man hier die Kombination von Charactergrafik mit Player-Missiles anschaulich demonstriert bekommt.

Get Up

Stellen Sie sich vor, Sie gingen zum Klo, und auf einmal wäre es weg! Nichts anderes ist unseren beiden Helden mit einem bösen Zauberer passiert. Schwuppdiwupp zauberte er ihnen einfach das ersehnte Örtchen weg und versetzte sie in ein verücktes Labyrinth, an dessen Ende die Klo-Tür wartet. Damit die Helden es nicht so leicht haben, hat der Zauberer noch ein paar Hindernisse hinzugezaubert, die die ganze Sache so richtig schön heiß werden lassen.

Get Up wird mit zwei Spielern und zwei Joysticks gespielt. Beide müssen versuchen, vor dem anderen das Klo zu erreichen. Falls ein Spieler in eine Flamme rennt oder aus dem Bildschirm fällt bzw. springt, muß er leider wieder am Startpunkt anfangen. Man hat dabei unendlich viele Leben.

Der Spieler, der als erstes das Örtchen erreicht, bekommt zehn Punkte gutgeschrieben. Als zusätzliche Verdienstmöglichkeit liegen noch Herzen herum, die jeweils zwei Punkte einbringen. Das Spiel ist beendet, wenn ein Spieler nach Beendigung eines Screens die vorher festgelegte Punktzahl erreicht oder überschritten hat.

Nach jedem vollendeten Screen zeigt das Programm eine Zwischenbilanz der erreichten Punkte an. Sobald die START-Taste gedrückt wird, geht es dann weiter.

Beim Titelbild wird die Anzahl der zu erreichenden Punkte mit SELECT eingestellt, wobei Werte von 10-90 gewählt werden können. Die START-Taste läßt das Spiel beginnen.

Die Figuren werden mit den Joysticks nach links, rechts, oben oder unten gesteuert. Mit dem Feuerknopf springt die Figur in die zuletzt gegangene Richtung.

Das Spiel ist eigentlich ein typisches Climb & Jump Game. Zusätzlich bietet es einen besonderen Reiz, gegen einen menschlichen Gegner zu spielen. Wie man hier sieht, kann man auch mit Zeichensatzgrafik im Modus eins gute Ergebnisse erzielen.

Golddiver

Es gab schon immer waghalsige Typen, die ihr Leben riskieren, nur um an ein paar Schätze heranzukommen ...

Einen von diesen verrückten Abenteurern steuern Sie in diesem Spiel, denn Sie haben sich aufgemacht, in den Tiefen der Meere nach Gold zu suchen!

An Gold mangelt es da unten wahrlich nicht, aber leider gibt es da noch so einen Typen, der sich einen Spaß daraus macht, ständig Wasserbomben zu werfen.

Sie müssen nun also versuchen, alle Goldnuggets einzusammeln, die herumliegen, um dann wieder zum sicherern Startpunkt zurückzukehren. Hierbei dürfen Sie sich nicht von einer Wasserbombe erwischen lassen, die dieser fiese Typ im Boot über der Wasseroberfläche ständig herunter wirft!

Bei der Aktion sollten Sie sich beeilen, da Ihnen sonst die Luft (Anzeige: OXYGEN) ausgeht!

Der Taucher wird mit dem Joystick gesteuert. Eine ganze Menge Meerlandschaften (und Wasserbomben) stehen zu Erkundung bereit ...

An diesem Spiel kann man gut die Methode der Levelspeicherung auf Diskette erkennen. Man sieht ebenfalls, daß auch mehrere Objekte (bis zu vier Bomben, das Boot und das eigene Männchen) in einer annehmbaren Geschwindigkeit bewegt werden können.

Scromber

Alarm! Bombendrohung im Wolkenkratzer! Sie sind die letzte Rettung! Ihre Aufgabe ist es, in jedem Stockwerk des Wolkenkratzers zehn Bomben zu entschärfen und dann in Richtung des nächsten Stockwerkes zu entschwinden, bevor die Zeitbomben explodieren.

Zu diesem Zweck steuern Sie einen Entschärfungskreisel auf dem dreidimensional dargestellten Stockwerk umher. Die Bomben müssen durch Berührung entschärft werden, bevor die Zeit abläuft.

Achtung: Kreiseln Sie nicht zu nahe am Rand, da Sie sonst herunterfallen!

Für den Notfall steht Ihnen eine Atombremse zur Verfügung, die den Kreisel augenblicklich anhält. Diese funktioniert allerdings nur drei mal! Nach Bewältigung eines Stockwerkes erhalten Sie eine zusätzliche Bremse als Bonus.

Der Kreisel wird mit dem Joystick gesteuert, wobei zu beachten ist, daß dieser beim Steuern nach oben auf dem Bildschirm diagonal wandert (bedingt durch den 3D-Effekt). Die Atombremse wird durch den Feuerknopf ausgelöst.

Werden Sie es schaffen, alle neun Stockwerke zu reinigen und so die Bewohner zu retten?

Dieses Spiel ist ein gutes Beispiel für die Grafikmöglichkeiten im kleinen Mehrfarbenmodus. Der 3D-Effekt ist relativ einfach zu programmieren, aber sieht hervorragend aus. Auch hier wurde Zeichensatz- und PM-Grafik gut miteinander kombiniert.

Zauberwald II

Sie steuern den mutigen Zauberer Mörc in den geheimnisvollen Zauberwald von Thor und müssen versuchen, den Schatz mit Zauberkugeln zu bewerfen.

Daran werden Sie allerdings von einem höchst aufdringlichen Monster gehindert, das seinen Schatz aufmerksam bewacht. Ein Berührung mit dem Monster führt zum Verlust eines Lebens.

Wenn Sie das Monster mit Zauberkugeln bewerfen, ist es davon so schockiert, daß es eine Weile in Ohnmacht fällt.

Es besteht natürlich auch die Möglichkeit, sich hinter einem Baum zu verstecken, aber das kostet nur wertvolle Zeit. Die verbleibende Zeit ist am Hintergrundton zu erkennen.

Noch ein kleines Problem: Der Schatz liegt natürlich nicht einfach so in der Gegend herum, sondern ist meist in einer Höhle versteckt. Zum Glück sind manche Steine weich genug, um sie mit Zauberkugeln zerstören zu können. Doch Achtung: Der Vorrat an Zauberkugeln ist pro Screen auf 100 beschränkt.

Mörc wird mit dem Joystick in Port eins gesteuert, per Feuerknopf werden Zauberkugeln geworfen.

Tip: Die Screends wurden mit dem Grafikeditor entworfen. Die Screens können damit erweitert und verändert werden, da sicher der Auswahlscreen auf der Diskette befindet!

Stone Mine II

In diesem Spiel steuert man einen mutigen Mann, der versuchen muß, in unterirdischen Höhlen Gold zu sammeln. Hierbei stellen sich ihm folgende Gefahren in den Weg:

- Blaue Mauern, die einfach undurchdringlich sind.

- Rote Mauern, die (pro Level) nur zehn Mal berührt werden dürfen, da sonst ein Leben verloren ist.

- Grüne Mauern, die sich bei Berührung selbst sprengen. Dies funktioniert aber nur fünf Mal und führt beim sechsten Mal zum Lebensverlust.

- Radioaktive Steine, die weggesprengt werden können, aber nicht berührt werden dürfen.

Auf seinem Weg findet der Held Dynamitstangen, die eingesammelt werden können. Gesprengt wird dann mit Feuerknopf und Richtungsangabe.

Das Spiel hat nicht besonders viele Screens (aber schaffen Sie die erst einmal). Es ist allerdings ganz einfach, eigene zu entwerfen. Alles, was Sie dazu benötigen, ist der mitgelieferte Grafikeditor. Auf der Programmdiskette befinden sich nämlich Screens, Zeichensatz sowie Auswahlscreen. Wählen Sie Grafikmodus (4) und los geht's. Die oberen und unteren Zeilen müssen freigelassen werden. Das Programm zählt das vorhandene Gold automatisch. Speichern Sie die Screens dann hinter den Originalleveln ab. Bis zu 26 Level (A–Z) sind möglich. Die entworfenen Level senden Sie uns bitte zu, damit wir sie veröffentlichen können. Wie bei jeder Einsendung winkt natürlich eine Gutschrift als Belohnung.

Leap

Bei Leap handelt es sich um ein Spiel, das höchstwahrscheinlich jeder von uns (wenn auch in einer anderen Form) schon einmal gespielt hat. Bei Spielbeginn sieht man ein Brett, auf dem 4*8 Kugeln angeordnet sind. Ziel des Spieles ist es, jeweils mit einer Kugel über eine andere in ein freies Feld zu springen, um so die übersprungene Kugel wegzunehmen. Es gilt, möglichst viele Kugeln wegzunehmen. Wer es schafft, nur och eine einzige Kugel übrigzubehalten, darf sich als Leaper betrachten.

Die Gelb dargestellte Kugel dient als Cursor, der mit dem Joystick bewegt werden kann. Per Knopfdruck wird eine Kugel ausgewählt. Auf die Frage "Choose Direction" wird per Joystick in die gewünschte Sprungrichtung gesteuert. Soll nicht gesprungen werden, kann nochmals der Feuerknopf gedrükt werden. Nach erfolgtem Sprung kann die nächste Kugel angewählt werden. Sind keine Sprungmöglichkeiten mehr vorhanden, kann das Spiel per SELECT-Taste abgebrochen werden.

Match

Nach Einladen des Spieles fragt der Computer zunächst nach dem Schwierigkeitsgrad. Dieser geht von "Sehr leicht" bis "unmöglich" und ist einfach durch Tippen einer Taste von 1-6 anzugeben.

Das Spielfeld besteht aus 19*11 zunächst leeren Feldern. Der Computer wählt nun per Zufall ein Feld und wechselt die Farbe dieses und der acht Nachbarfelder nach folgender Reihenfolge: Schwarz -> Rot -> Blau -> Hellblau -> Schwarz.

Dies wiederholt er je nach Schwierigkeitsgrad 20 bis 120 mal.

Aufgabe des Spielers ist es nun, das Spielfeld wieder in den ursprünglichen Zustand zu versetzen. Hierzu wird ein Cursor mit dem Joystick gesteuert, während durch Drücken des Feuerknopfes die Felder gewechselt werden. Je weniger Züge benötigt werden, desto besser.

Bei Spielende zeigt der Computer die vom Spieler gebrauchten Züge sowie die anfangs gemachten Züge an. Ein gutes Ergebnis ist erzielt worden, wenn die Anzahl der gebrauchten Züge weniger als das Dreifache der vom Computer gemachten Züge beträgt.

Diamonds

Bei Diamonds handelt es sich um ein wirklich schwieriges Strategiespiel. Nach Starten des Programmes wird zunächst nach der Anzahl der Diamanten gefragt, was gleichbedeutend mit dem Schwierigkeitsgrad ist. Hierzu ist auf der Tastatur einfach eine Nummer von 2-8 zu tippen.

Nun erscheinen irgendwo in dem Labyrinth die gewählte Anzahl an Diamanten sowie gleichviele Löcher.

Aufgabe ist es nun, jeden Diamanten in eines der Löcher zu befördern. Das Problem dabei ist, daß sich bei einer Bewegung alle Diamanten gleichzeitig bewegen (sofern möglich). Außerdem bleibt ein Diamant NICHT in einem Loch liegen. Erst, wenn alle Diamanten gleichzeitig in einem Loch liegen, ist das Spiel geschafft ...

Tip: Erstmal mit zwei Diamanten probieren, um das Prinzip zu verstehen, dann weiter hocharbeiten.

Die Bewegung der Diamanten erfolgt per Joystick in Port 0.

Wumpus

Bei diesem Spiel geht es darum, in einer 6*6 Räume großen Höhle drei Monster aufzuspüren und zu erlegen. Hierbei gibt es allerdings eine Menge Probleme:

Da es in der Höhle dunkel ist, kann man die Monster natürlich auch nicht sehen. Befindet sich ein Monster in einer benachbarten Höhle, so erscheint am unteren Bildschirmrand die Nachricht "ACHTUNG, MONSTER!". Die Berührung mit einem Monster ist natürlich tödlich!

Neben den Monstern, die sich übrigens nicht von der Stelle bewegen, sind in der Höhle noch einige Fallgruben sowie eine Fledermaus versteckt. In eine Fallgrube zu fallen ist ebenfalls tödlich. Vor Fallgruben wird allerdings ebenfalls am unteren Bildschirmrand gewarnt. Die Berührung mit der Fledermaus versetzt den Spieler in einen zufällig ausgewählten raum, in dem sich NICHTS befindet!

Damit Sie nicht in die Versuchung kommen, wild um sich zu ballern, haben Sie für die drei Monster nur vier Schuß. Um zu schießen wird erst der Knopf gedrückt und dann in eine durch Fragezeichen dargestellte Richtung gesteuert. Hat man keine Schüsse, ist das Spiel beendet! Also: Sparsam sein.

Für jeden Raum, der zum erstenmal betreten wird, bekommt man 200 Punkte gutgeschrieben. Sie sollten allerdings nicht unnötig herumlaufen, da jede Bewegung etwas von der Zeit abzieht. Abgelaufene Zeit bedeutet: GAME OVER!

Haben Sie es geschafft, alle drei Monster zu erlegen, bekommen Sie die verbliebene Zeit und die nicht verbrauchten Schüsse als Bonus.

Sternenhaufen

Bei Sternenhaufen handelt es sich um eine Strategiespiel, bei dem es darum geht, möglichst viele Sterne zu erobern. Die genaue Anleitung befindet sich im Programm.

Flug 747

Im Titelbild können Sie per SELECT und OPTION die Spielstärke auswählen und eventuell die Instrumente abschalten. Per START und Feuerknopf beginnt das Spiel.

Auf dem Bildschrim erscheint eine Sicht auf die Landebahn. Links und rechts befindet sich eine Gradeinteilung. Von Teilstrich zu Teilstrich sind es 5 Grad Neigungswinkel. Der optimale Neigungswinkel beträgt 6 Grad, durch einen Strich neben der Gradanzeige angedeutet.

Die Instrumententafel gliedert sich wie folgt:

ALTD: Flughöhe

RANGE: Entfernung bis zur Landebahn (in der letzten Phase: Entfernung bis zum Landebahnende)

HDG: Links/Rechtsabweichung von der Ideallinie

V: Geschwindigkeit

T: Benötigte Zeit

F: Treibstoff (sollte nach der Landung möglichst genau 1000 sein)

$: Flugzustandsanzeige

L: Fahrgestell (0=ausgefahren)

Steuerung per Joystick mit gedrücktem Feuerknopf:

Links: Geschwindigkeit verringern (höchstens auf 40)

Rechts: Geschwindigkeit vergrößern (höchstens auf 400)

Oben: Fahrgestell ausfahren (erst in der letzten Phase möglich)

Unten: Autopilot einschalten (hält Flugzeug auf Kurs, gibt aber weniger Punkte)

Die Fluganzeige:

"^": Flugzeug zu niedrig, Joystick nach hinten ziehen.

"=": Flugzeug zu hoch, Joystick nach vorne drücken.

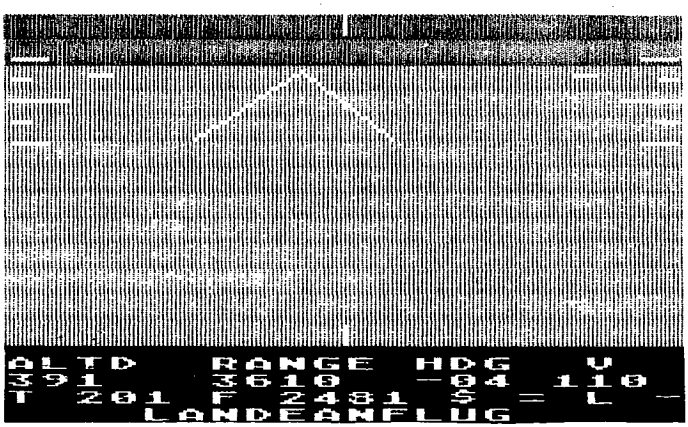
":": Flugzeug ist auf der Ideallinie.

Vor der Schlußphase muß die Geschwindigkeit auf 150 verringgert werden.

Die Landung erfolgt am Ende ungefähr so:

Fahrgestell ausfahren. Geschwindigkeit, Höhe und Anflugwinkel verringern, bis die Instrumentenanzeige grün wird (ACHTUNG: Solange die Anzeige nicht grün ist, muß die Geschwindigkeit größer 0 sein). Geschwindigkeit auf 0 bringen. Warten, bis das Flugzeug zu stehen kommt.

Insgesamt können mehr als 100000 Punkte erreicht werden. Dies klappt allerdings nur wenn der Treibstoff genau 1000 ist, der Autopilot nicht eingeschaltet wurde, die Instrumente nicht benutzt wurden, die Spielstärke 3 ist und die Landung schnell von statten ging.



ZONG	*** INT/ASC-Code Tabelle ***								ZONG
Z	INT-D/H	ASC-D/H	Z	INT-D/H	ASC-D/H	Z	INT-D/H	ASC-D/H	Z	INT-D/H	ASC-D/H
	000	$00 032 $20	@	032	$20 064 $40	♥	064	$40 000 $00	♦	096	$60 096 $60
!	001	$01 033 $21	A	033	$21 065 $41	├	065	$41 001 $01	a	097	$61 097 $61
"	002	$02 034 $22	B	034	$22 066 $42	│	066	$42 002 $02	b	098	$62 098 $62
#	003	$03 035 $23	C	035	$23 067 $43	┘	067	$43 003 $03	c	099	$63 099 $63
$	004	$04 036 $24	D	036	$24 068 $44	┤	068	$44 004 $04	d	100	$64 100 $64
%	005	$05 037 $25	E	037	$25 069 $45	┐	069	$45 005 $05	e	101	$65 101 $65
&	006	$06 038 $26	F	038	$26 070 $46	/	070	$46 006 $06	f	102	$66 102 $66
'	007	$07 039 $27	G	039	$27 071 $47	\	071	$47 007 $07	g	103	$67 103 $67
(	008	$08 040 $28	H	040	$28 072 $48	◢	072	$48 008 $08	h	104	$68 104 $68
)	009	$09 041 $29	I	041	$29 073 $49	▘	073	$49 009 $09	i	105	$69 105 $69
*	010	$0A 042 $2A	J	042	$2A 074 $4A	◣	074	$4A 010 $0A	j	106	$6A 106 $6A
+	011	$0B 043 $2B	K	043	$2B 075 $4B	▝	075	$4B 011 $0B	k	107	$6B 107 $6B
,	012	$0C 044 $2C	L	044	$2C 076 $4C	▗	076	$4C 012 $0C	l	108	$6C 108 $6C
-	013	$0D 045 $2D	M	045	$2D 077 $4D	▔	077	$4D 013 $0D	m	109	$6D 109 $6D
.	014	$0E 046 $2E	N	046	$2E 078 $4E	_	078	$4E 014 $0E	n	110	$6E 110 $6E
/	015	$0F 047 $2F	O	047	$2F 079 $4F	▖	079	$4F 015 $0F	o	111	$6F 111 $6F
0	016	$10 048 $30	P	048	$30 080 $50	┼	080	$50 016 $10	p	112	$70 112 $70
1	017	$11 049 $31	Q	049	$31 081 $51	┌	081	$51 017 $11	q	113	$71 113 $71
2	018	$12 050 $32	R	050	$32 082 $52	─	082	$52 018 $12	r	114	$72 114 $72
3	019	$13 051 $33	S	051	$33 083 $53	┼	083	$53 019 $13	s	115	$73 115 $73
4	020	$14 052 $34	T	052	$34 084 $54	●	084	$54 020 $14	t	116	$74 116 $74
5	021	$15 053 $35	U	053	$35 085 $55	■	085	$55 021 $15	u	117	$75 117 $75
6	022	$16 054 $36	V	054	$36 086 $56	▏	086	$56 022 $16	v	118	$76 118 $76
7	023	$17 055 $37	W	055	$37 087 $57	┬	087	$57 023 $17	w	119	$77 119 $77
8	024	$18 056 $38	X	056	$38 088 $58	┴	088	$58 024 $18	x	120	$78 120 $78
9	025	$19 057 $39	Y	057	$39 089 $59	▌	089	$59 025 $19	y	121	$79 121 $79
:	026	$1A 058 $3A	Z	058	$3A 090 $5A	└	090	$5A 026 $1A	z	122	$7A 122 $7A
;	027	$1B 059 $3B	[	059	$3B 091 $5B	▒	091	$5B 027 $1B	♠	123	$7B 123 $7B
<	028	$1C 060 $3C	\	060	$3C 092 $5C	↑	092	$5C 028 $1C	|	124	$7C 124 $7C
=	029	$1D 061 $3D	]	061	$3D 093 $5D	↓	093	$5D 029 $1D	¶	125	$7D 125 $7D
>	030	$1E 062 $3E	^	062	$3E 094 $5E	←	094	$5E 030 $1E	◀	126	$7E 126 $7E
?	031	$1F 063 $3F	_	063	$3F 095 $5F	→	095	$5F 031 $1F	▶	127	$7F 127 $7F

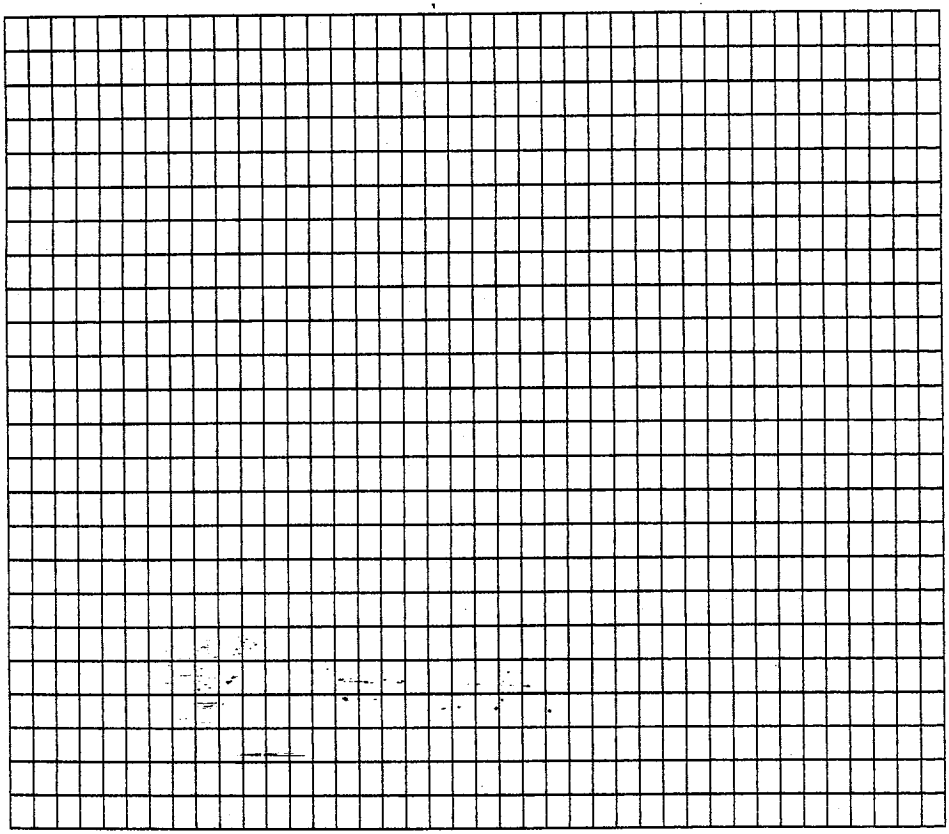
Grafikmodi-Übersicht

Hier finden Sie eine Übersicht der Grafikmodi von 0–15, jeweils mit Angaben der Grafikart, der Auflösung mit Positionen sowie der Farben und einem Rasterbeispiel. Bei den Grafikmodi ist jeweils das Textfenster ausgeschaltet, über Basic muß zum Grafikmodus also 16 addiert werden.

Grafikmodus 0, Zeichensatzgrafik mit einfarbigen Zeichen

Auflösung: Horizontal 40 (0–39), Vertikal 24 (0–23)

Farben: 3, Zeichenfarbe (709), Hintergrund (710), Umrahmung (712)

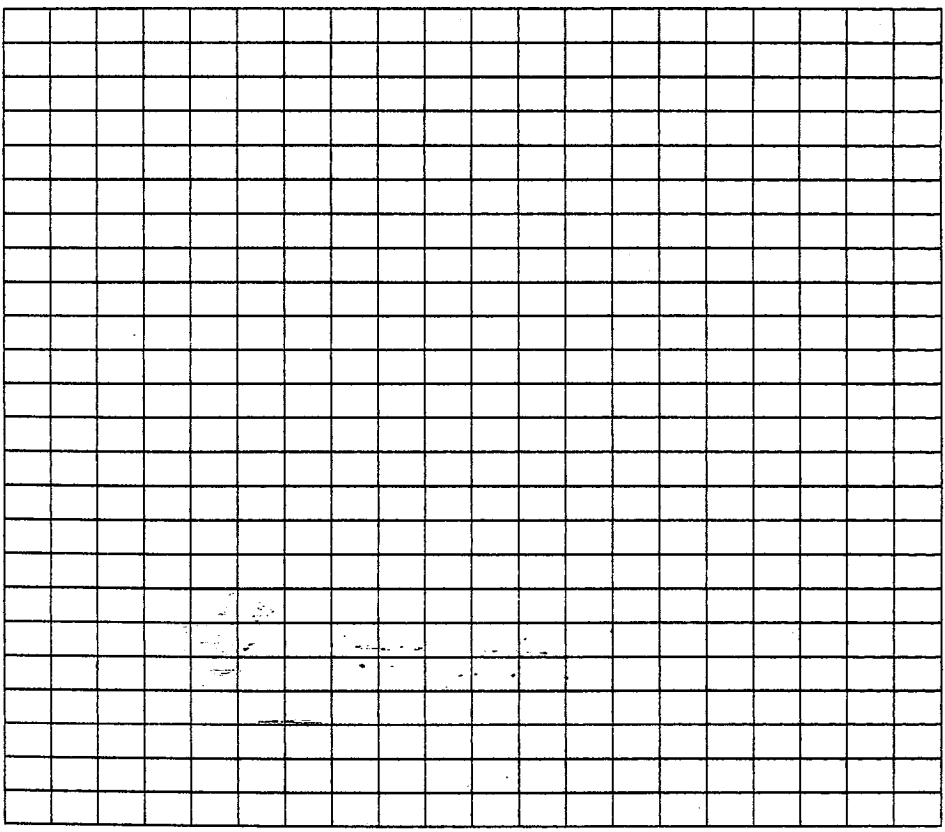


ANHANG

Grafikmodus 1, Zeichensatzgrafik mit einfarbigen Zeichen

Auflösung: Horizontal 20 (0–19), Vertikal 24 (0–23)

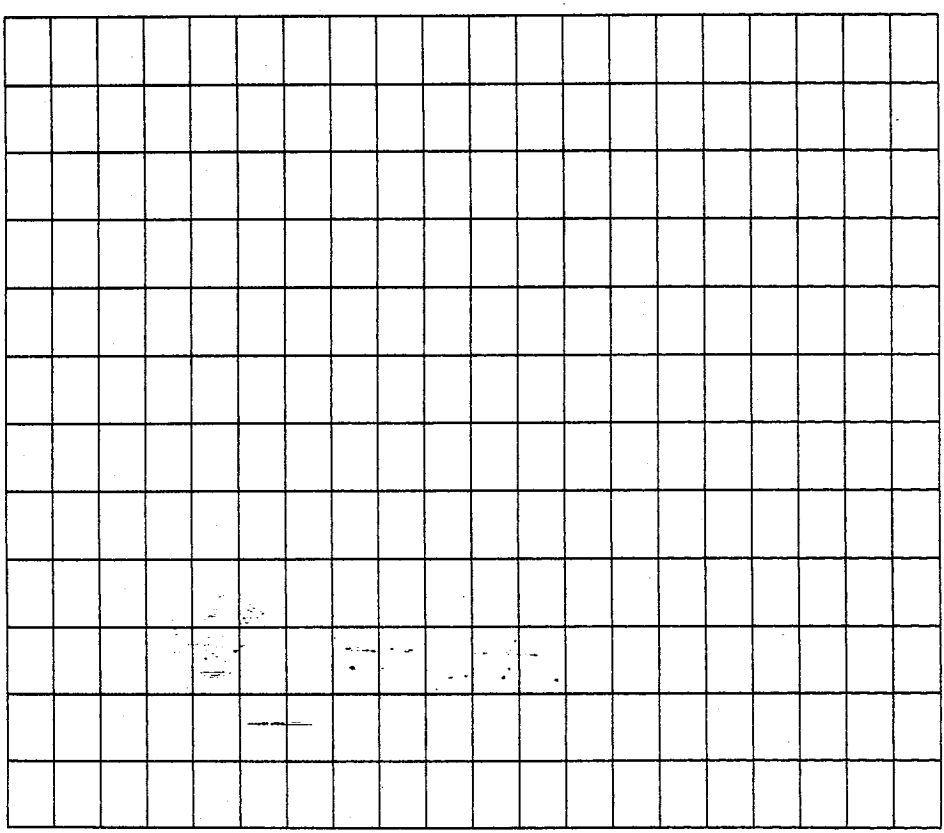
Farben: 5, Zeichenfarben (708–711), Hintergrund (712)



Grafikmodus 2, Zeichensatzgrafik mit einfarbigen Zeichen

Auflösung: Horizontal 20 (0–19), Vertikal 12 (0–11)

Farben: 5, Zeichenfarben (708–711), Hintergrund (712)

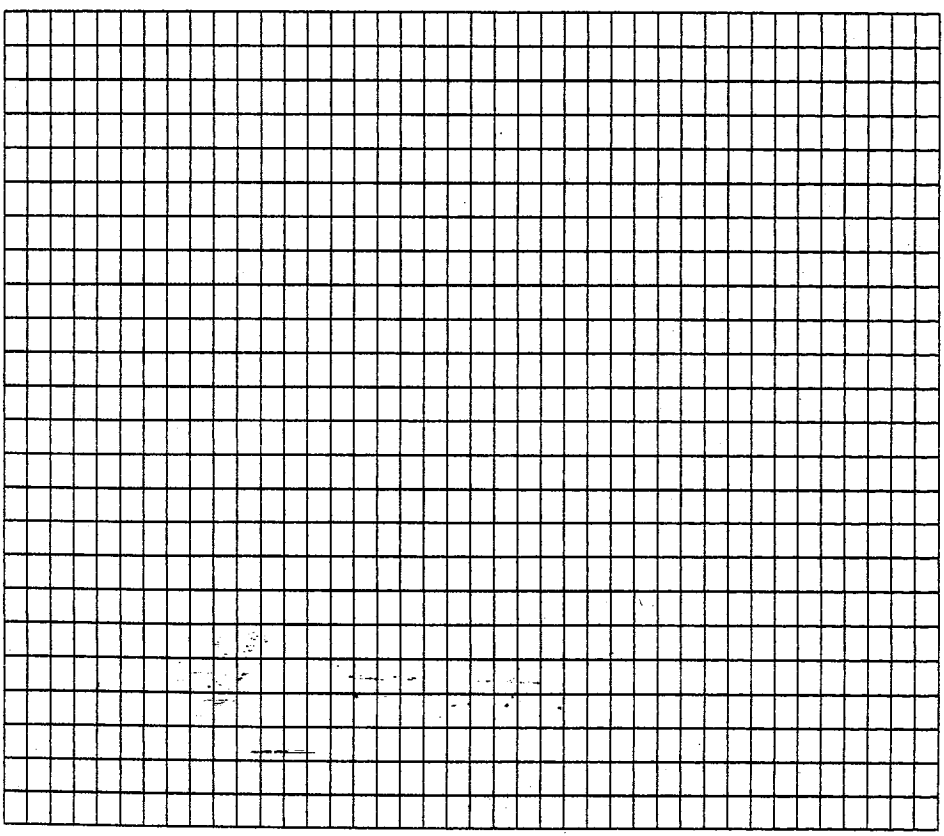


ANHANG

Grafikmodus 3, Pixelgrafik

Auflösung: Horizontal 40 (0–39), Vertikal 24 (0–23)

Farben: 4, Pixelfarben (708–710), Hintergrund (712)

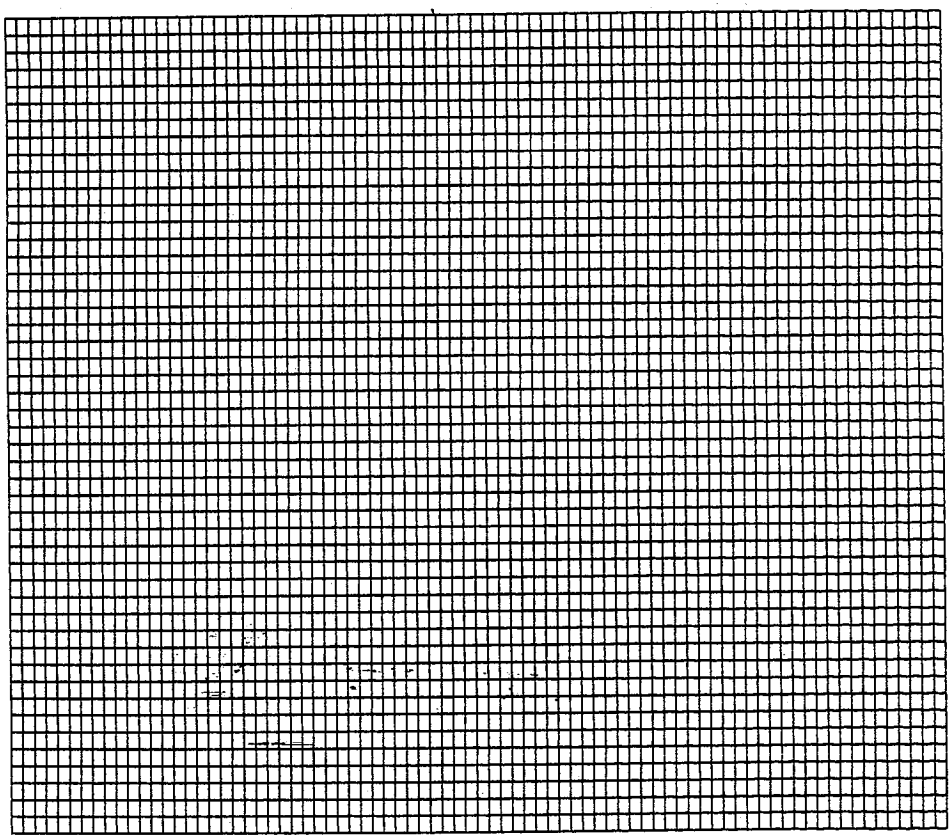


ANHANG

Grafikmodus 4, Pixelgrafik

Auflösung: Horizontal 80 (0–79), Vertikal 48 (0–47)

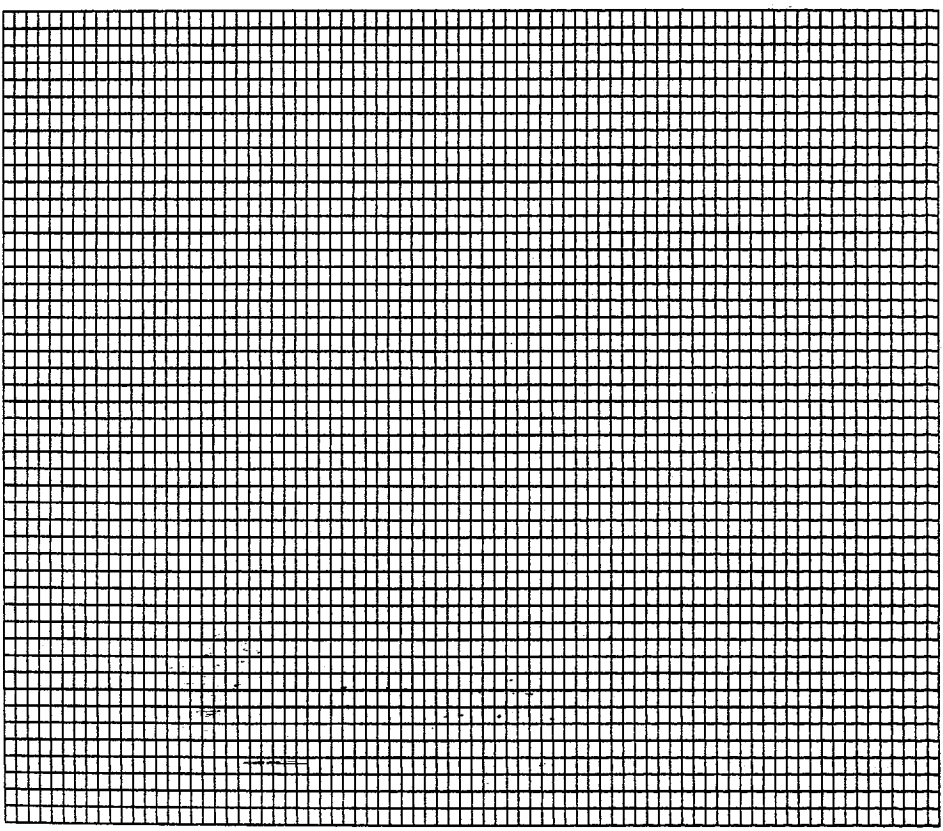
Farben: 2, Pixelfarbe (708), Hintergrund (712)



Grafikmodus 5, Pixelgrafik

Auflösung: Horizontal 80 (0–79), Vertikal 48 (0–47)

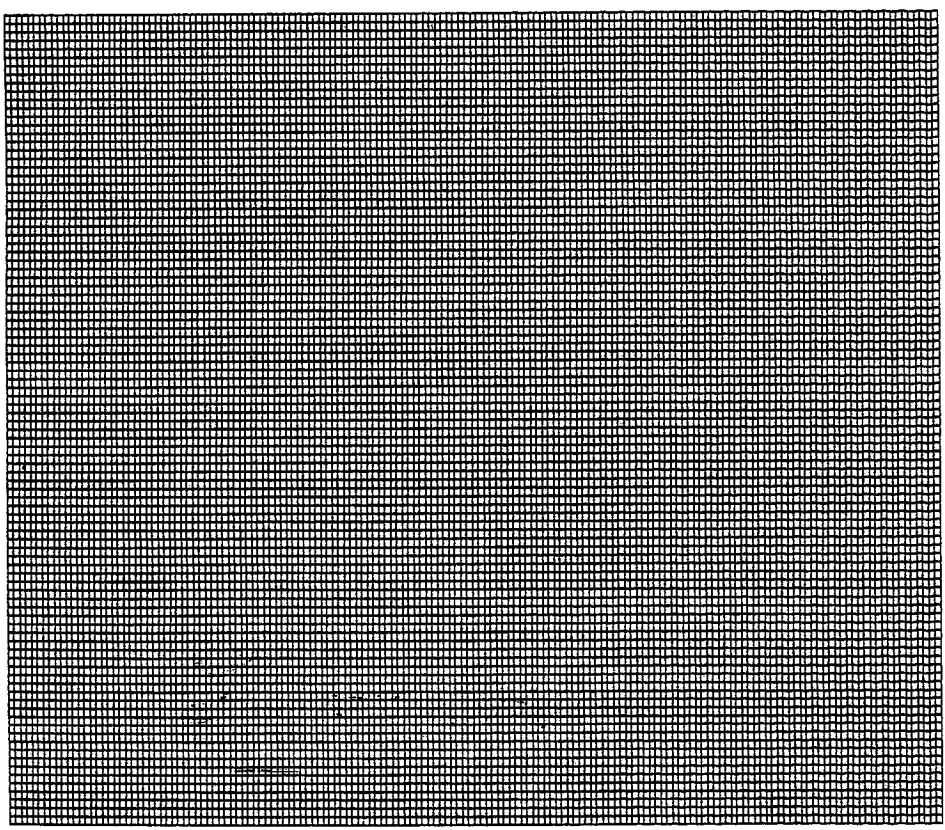
Farben: 4, Pixelfarben (708–710), Hintergrund (712)



Grafikmodus 6, Pixelgrafik

Auflösung: Horizontal 160 (0-159), Vertikal 96 (0-95)

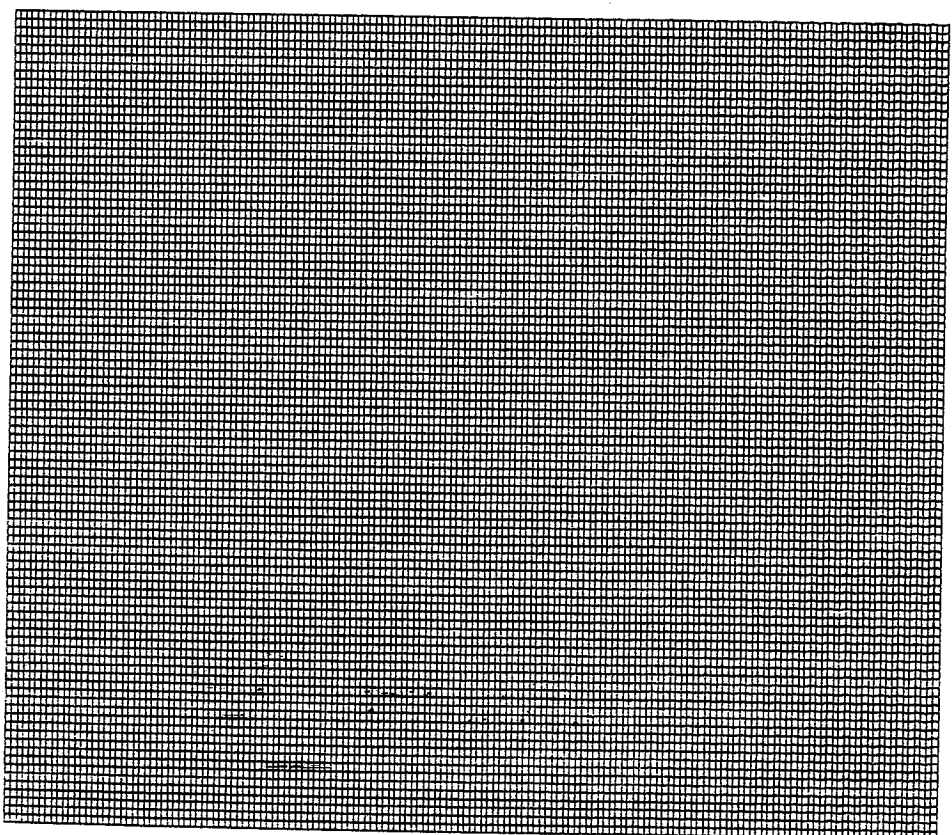
Farben: 2, Pixelfarbe (708), Hintergrund (712)



Grafikmodus 7, Pixelgrafik

Auflösung: Horizontal 160 (0-159), Vertikal 96 (0-95)

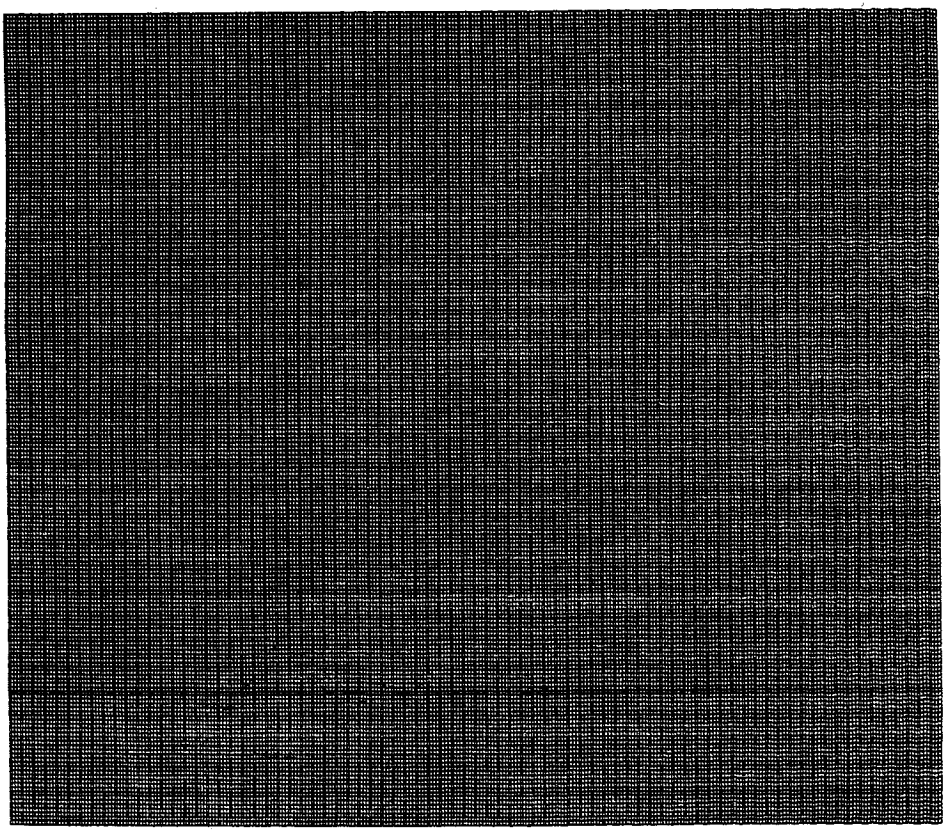
Farben: 4, Pixelfarben (708-710), Hintergrund (712)



Grafikmodus 8, Pixelgrafik

Auflösung: Horizontal 320 (0-319), Vertikal 192 (0-191)

Farben: 2, Pixelfarbe (708), Hintergrund (712)

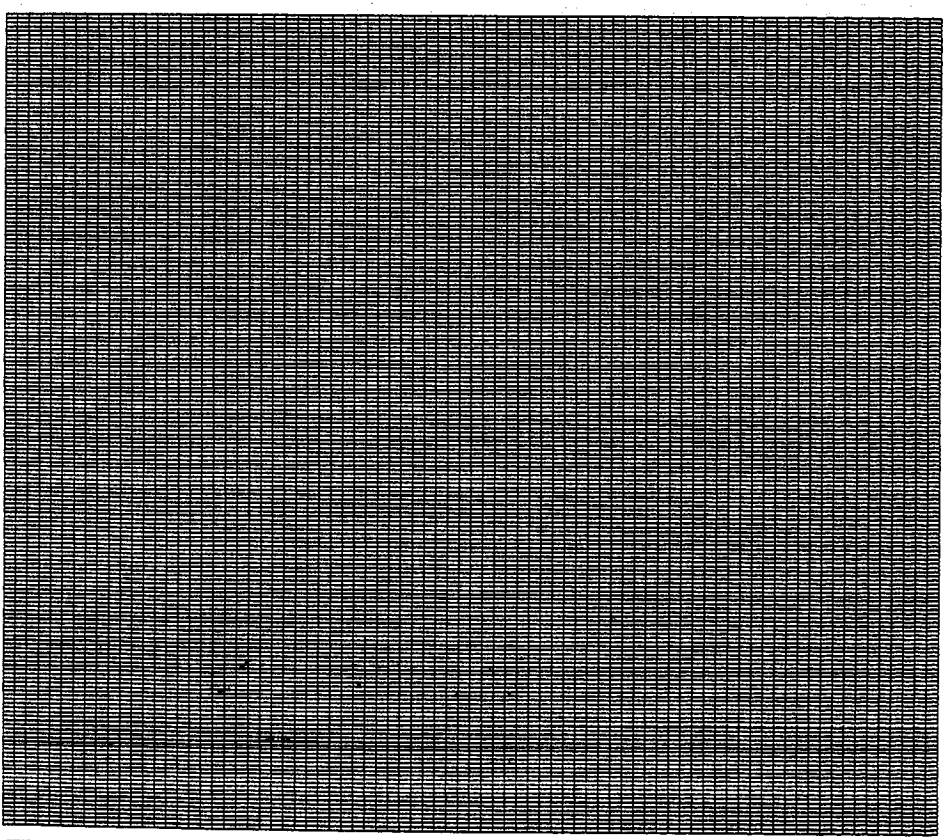


ANHANG

Grafikmodus 9, Pixelgrafik

Auflösung: Horizontal 80 (0–79), Vertikal 192 (0–191)

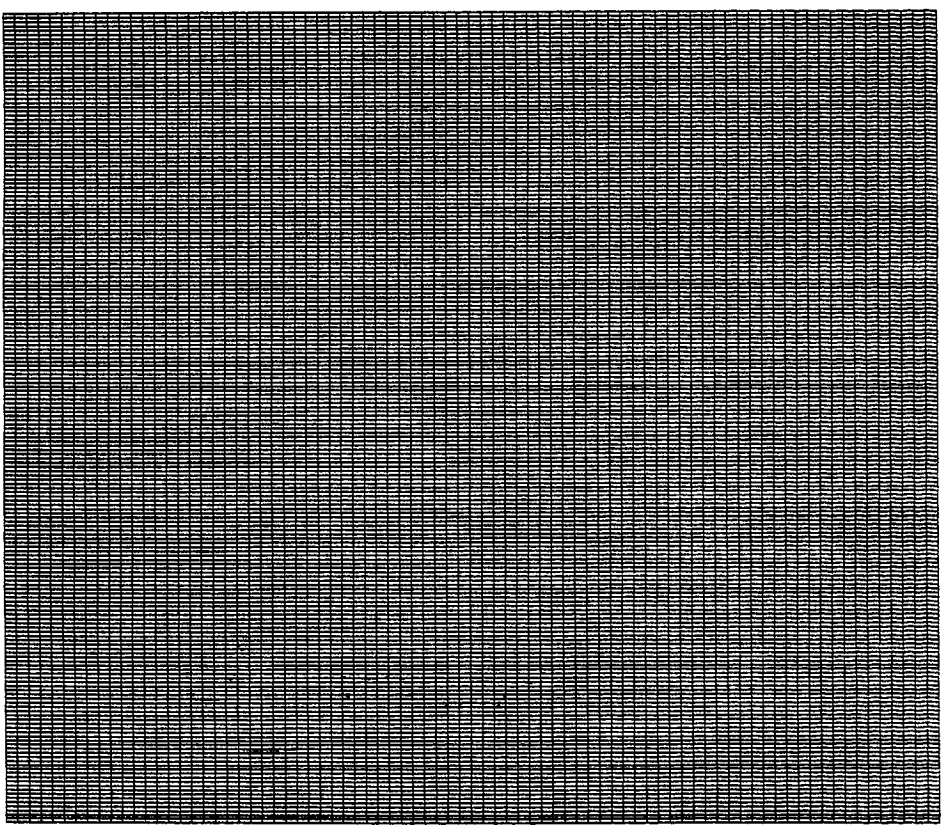
Farben: 1 (712) mit 16 verschiedenen Helligkeiten



Grafikmodus 10, Pixelgrafik

Auflösung: Horizontal 80 (0–79), Vertikal 192 (0–191)

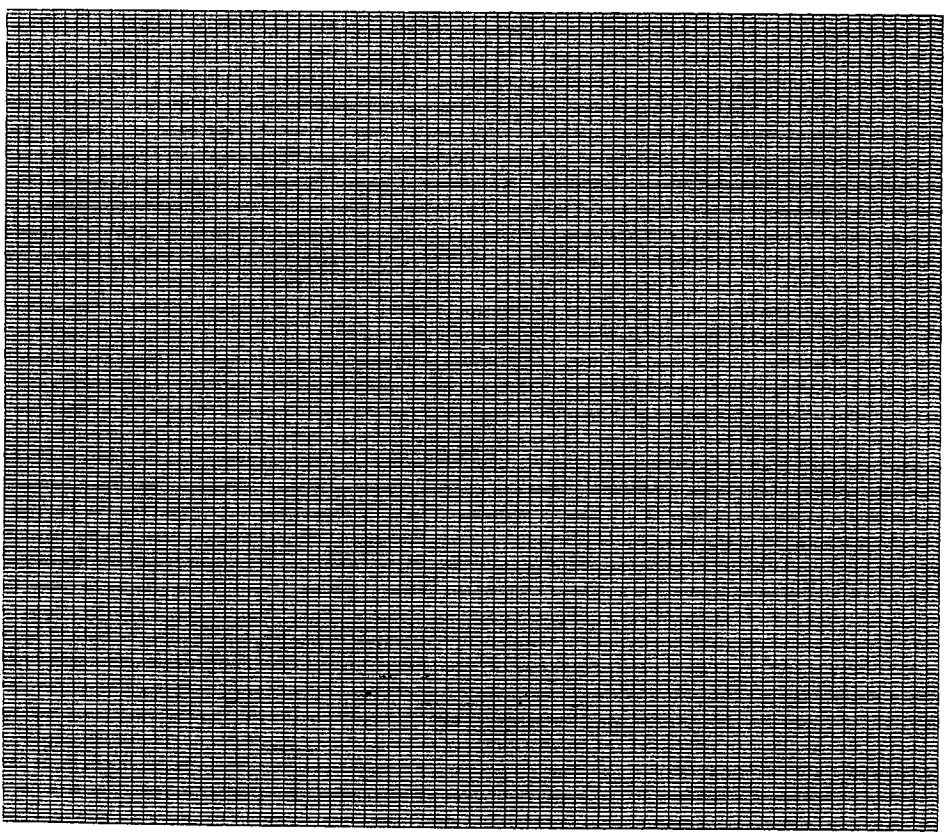
Farben: 16, Pixelfarben (704–712)



Grafikmodus 11, Pixelgrafik

Auflösung: Horizontal 80 (0–79), Vertikal 192 (0–191)

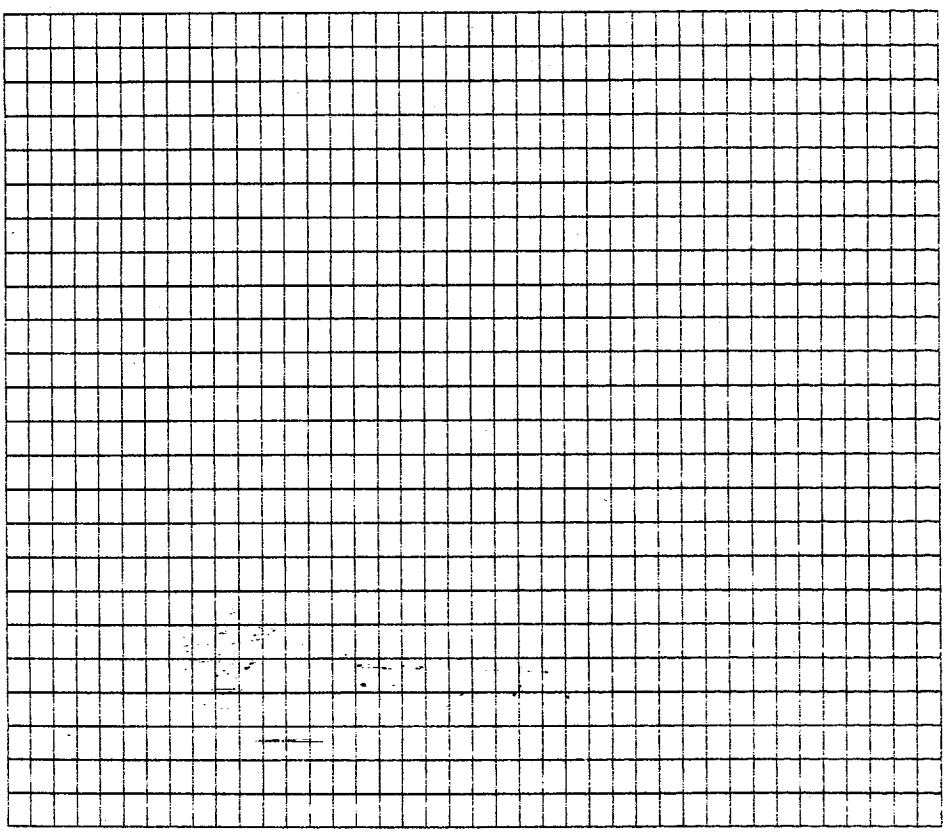
Farben: 16 verschiedene (fest)



Grafikmodus 12, Zeichensatzgrafik mit mehrfarbigen Zeichen

Auflösung: Horizontal 40 (0-39), Vertikal 24 (0-23)

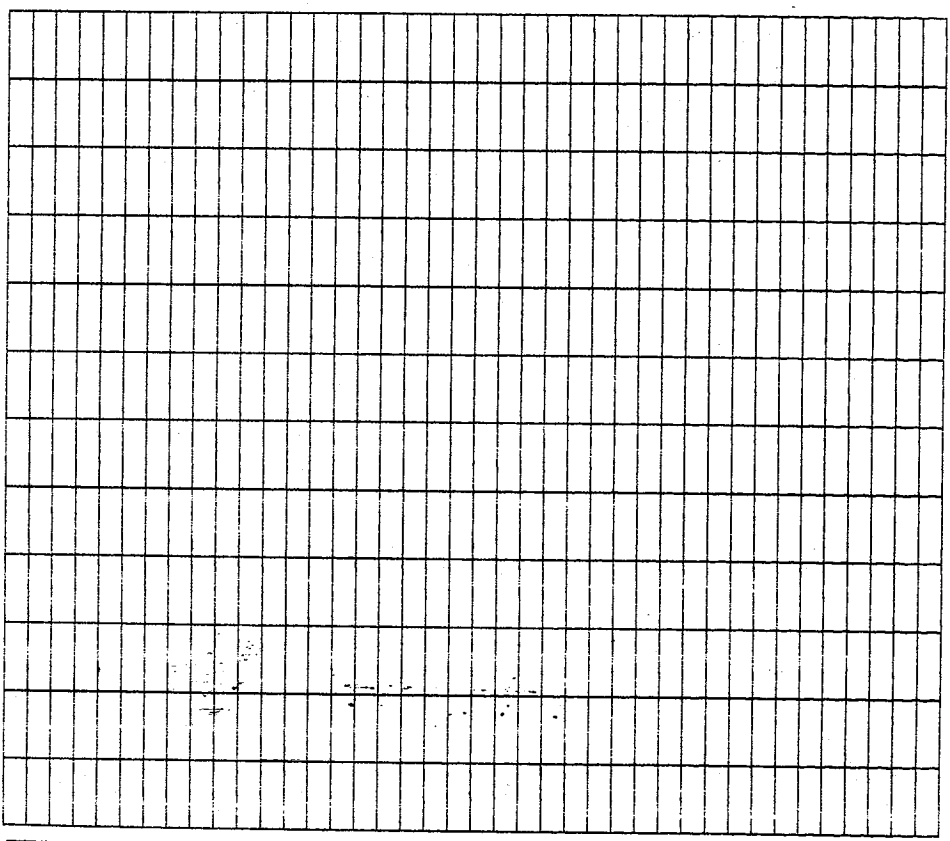
Farben: 5, Zeichenfarben (708-711), Hintergrund (712)



Grafikmodus 13, Zeichensatzgrafik mit mehrfarbigen Zeichen

Auflösung: Horizontal 40 (0–39), Vertikal 12 (0–11)

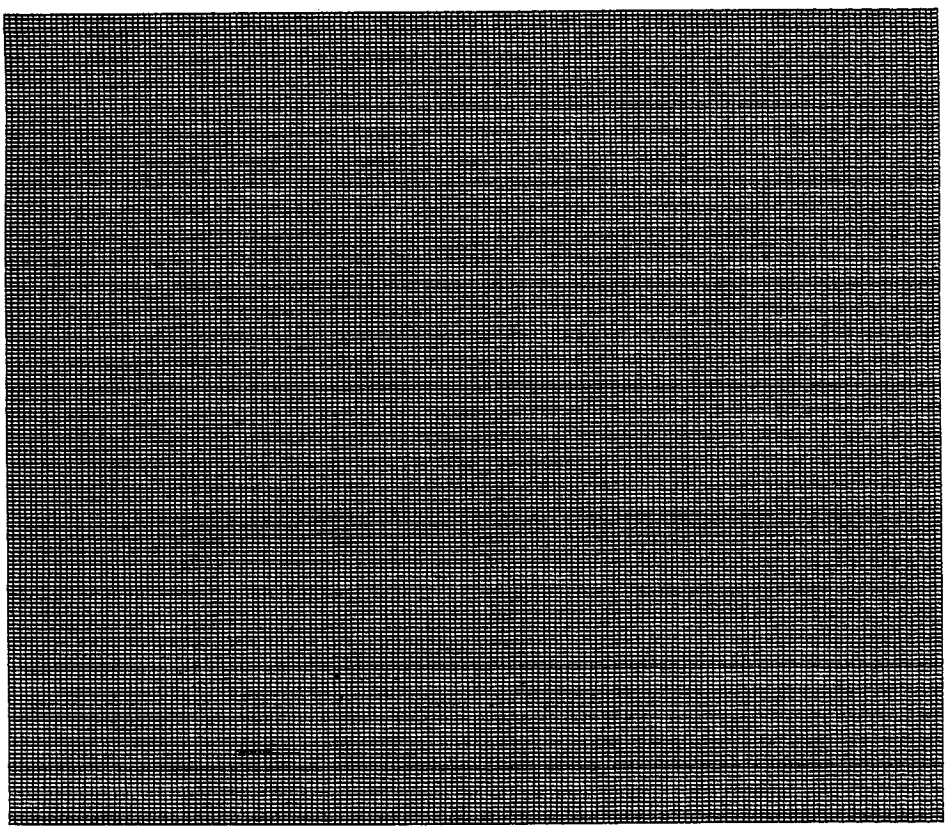
Farben: 5, Zeichenfarben (708–711), Hintergrund (712)



Grafikmodus 14, Pixelgrafik

Auflösung: Horizontal 160 (0-159), Vertikal 192 (0-191)

Farben: 2, Pixelfarbe (708), Hintergrund (712)

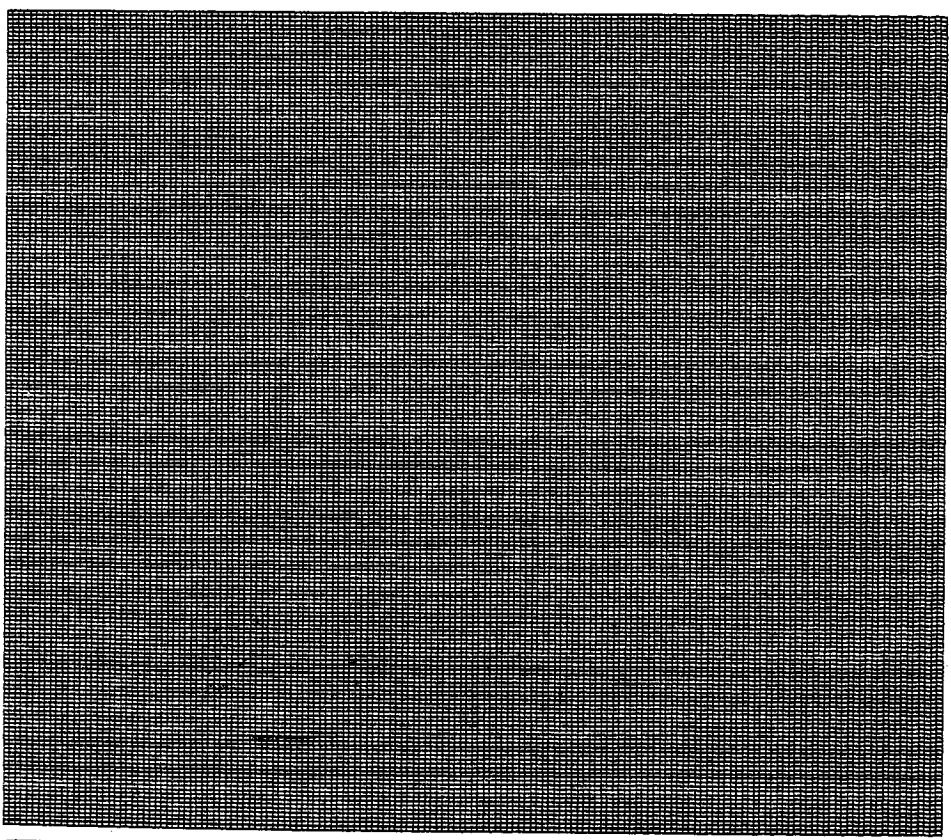


ANHANG

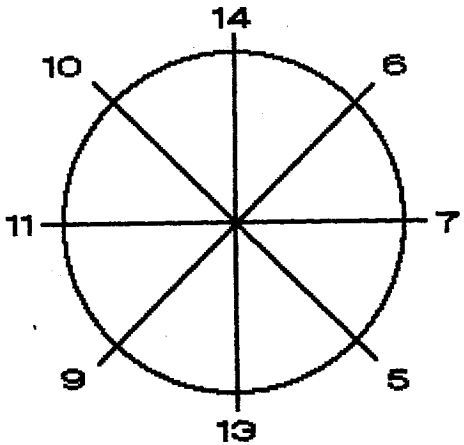
Grafikmodus 15, Pixelgrafik

Auflösung: Horizontal 160 (0-159), Vertikal 192 (0-191)

Farben: 4, Pixelfarben (708-710), Hintergrund (712)



Joystick-Werte



CX-85 Werte