

AUSGABE



7

AMIGA

Markt & Technik

DM 16,-

OS 120,- / Sfr 16,-
Lit 16000 / hfl 21,- / dkr 75,-

SONDERHEFT

SPIELEPROGRAMMIERUNG

Know-how

- *Insider decken auf:
Die Geheimnisse
von Blitter, Copper & Co.*
- *Spieltheorie:
Mit der richtigen
Strategie zum Erfolg*

Projekte

- *Spiele-Top-Hits:
5 neue Power-Games*
- *Vektor-Grafik:
Die erste 3D-Vektor-
Library*

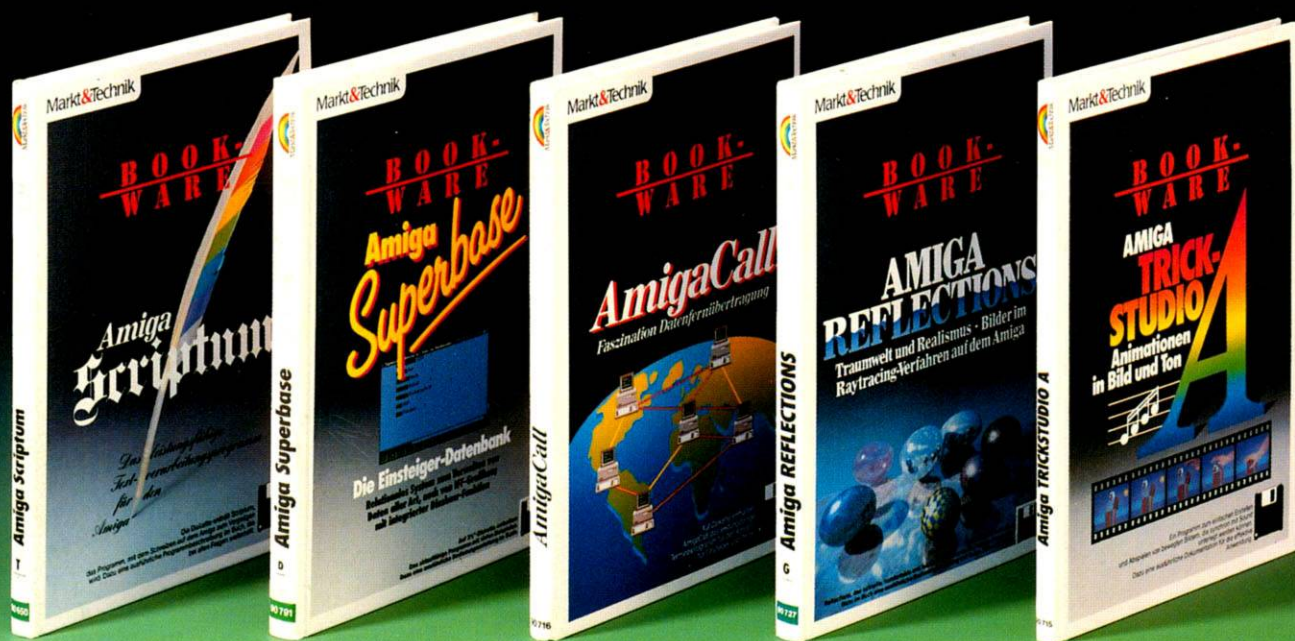
ALLE
PROGRAMME
AUCH AUF
DISKETTE
ERHÄLTICH



AMIGA

BOOK- WARE

Profi-Software unter 100 DM



R. Arbinger, I. Krüger

Scriptum

Das leistungsfähige Textverarbeitungsprogramm für den Amiga. Die Diskette enthält Scriptum, das Programm, mit dem Schreiben auf dem Amiga zum Vergnügen wird. Dazu eine ausführliche Programmbeschreibung im Buch, die bei allen Fragen weiterhilft.

1989, 141 Seiten,
inkl. Programmdiskette.

ISBN 3-89090-650-8

DM 79,-* (sFr 72,70*/öS 672,-*)

Precision Software

Amiga Superbase

Die Einsteiger-Datenbank. Relationales System zum Verwalten von Daten aller Art, auch von IFF-Grafiken; mit integrierter Diashow-Funktion.

1989, 176 Seiten,
inkl. Programmdiskette.

ISBN 3-89090-791-1,

DM 89,-* (sFr 81,90*/öS 757,-*)

Atlantis

AmigaCall

Treten Sie ein in die faszinierende Welt der Datenfernübertragung. Kommunizieren Sie über Mailboxen mit erfahrenen Computer-Anwendern, die Ihnen bei Ihren Problemen weiterhelfen können, oder Sie erhalten auf diesem Wege leistungsfähige Public-Domain-Software.

1988, 133 Seiten,
inklusive Programmdiskette.

ISBN 3-89090-716-4,

DM 99,-* (sFr 91,-*/öS 842,-*)

C. Fuchs

Reflections

Traumwelt und Realismus - Bilder im Raytracing-Verfahren auf dem Amiga. Dazu im Buch eine ausführ-

liche Bedienungsanleitung mit vielen

Tips und Tricks.

1989, 156 Seiten,
inklusive Programmdiskette.

ISBN 3-89090-727-X,

DM 98,-* (sFr 90,20*/öS 834,-*)

Atlantis

Trickstudio A

Animationen in Bild und Ton. Ein Programm zum einfachen Erstellen und Abspielen von bewegten Bildern, die synchron mit Sound unterlegt werden können. Dazu eine ausführliche Dokumentation für die effektive Anwendung.

1988, 86 Seiten,
inklusive Programmdiskette.

ISBN 3-89090-715-6,

DM 99,-* (sFr 91,-*/öS 842,-*)

In Vorbereitung:

N. Wirsing

Amiga Audio Entwickler-Paket

Dieses Buch macht Sie zum perfekten Amiga-Tontechniker.

Lieferbar 3. Quartal '89,

ca. 400 Seiten,

inkl. 2 Programmdisketten.

ISBN 3-89090-765-2,

ca. DM 98,-* (sFr 90,20*/öS 834,-*)

In Vorbereitung:

H. Knappe

Amiga Sounder

Der Amiga Sounder ist ein Komplett-paket für den Einstieg in die Welt der digitalen Klänge.

Lieferbar 3. Quartal 1989,

inkl. 2 Programmdisketten.

ISBN 3-89090-709-1,

ca. DM 98,-* (sFr 90,20*/öS 834,-*)

*Unverbindliche Preisempfehlung.

**Markt & Technik-Produkte
erhalten Sie bei Ihrem Buch-
oder Computerfachhändler**

INFO-COUPON

Bitte senden Sie mir Ihr Gesamtverzeichnis
mit 500 aktuellen Computerbüchern und Software.

Name _____

Straße _____

PLZ/Ort _____

Bitte ausschneiden und senden an: Markt & Technik Verlag AG,
Buchverlag, Frau Brosien, Hans-Pinsel-Str. 2, 8013 Haar

Ami SH 7

Game Over

● Tilt! Verdammtes \$%&? #ß)! Jetzt habe ich die Nase voll und schreib mein eigenes Spiel. Warum auch nicht?

● Schnell noch Assembler gelernt, ein paar Grafiken über den Bildschirm geblittert, das Betriebssystem abgeschaltet und Kopf voraus in den Supervisor-Modus.

● Und dann? Natürlich gehört es zum guten Ton, einige heiße Soundeffects eiskalt unterzumischen. Noch schnell die Copperliste manipulieren und einige Interrupts verbiegen. Wozu hat mein Amiga mindestens 4194304 einzelne Bits? Wer soll die setzen, wenn nicht ich? Devices und ähnlicher Schnickschnack – weg damit. Programmstruktur? Schön chaotisch – es braucht ja nicht jeder wissen, mit welchen Tricks ich arbeite.

● Somit haben wir alle Zutaten für ein perfektes Spiel. Garnieren wir das Ganze noch mit einem netten Vorspann und einem möglichst außerirdischen Namen, und schon ist der Erfolg programmiert.

● Aber fehlt da nicht noch was? Ach ja, die Spielidee – nach soviel »perfektem« Programmieren wird sich da sicher noch was ergeben.

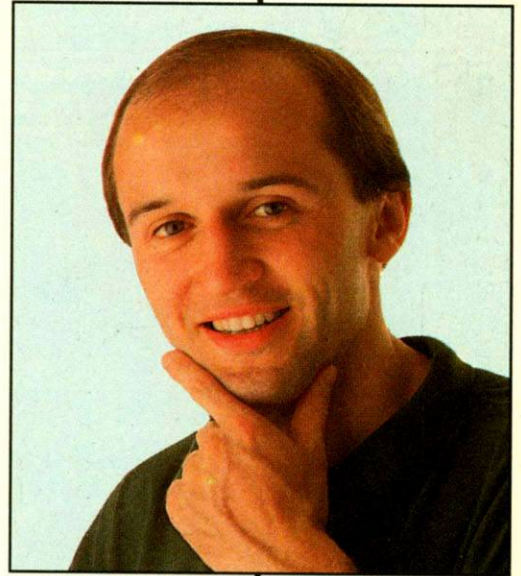
● Ist es da ein Wunder, daß noch immer die meisten Leute am liebsten Schach spielen? Warum wird dieses Spiel nicht langweilig, obwohl es seit Jahrtausenden »auf dem Markt« ist? Sehr einfach: Es lebt von einer genialen Idee.

● An dieser scheint es den meisten Computerspielen zu mangeln. Jedenfalls werden sie meist schon nach einigen Tagen nur noch von wenigen gespielt, gelten als fade und abgelutscht.

● Also tatsächlich »Game Over«? Sicher, es gibt einige rühmliche Ausnahmen. Dennoch treffen die oben genannten Merkmale – leider – auf die meisten Spiele zu.

● Machen Sie sich also zunächst auf die Suche nach einer guten Idee für Ihr Spiel. Sie werden für jeden Ansatz

auch eine programmtechnische Lösung finden. Das nötige Know-how finden Sie in diesem Heft. Die Ideen müssen Sie schon selbst haben.



Ihr
Klaus Sonnenleiter
(Redakteur)

K. Sonnenleiter

Projekt

6 Am Anfang war die Idee

Verfolgen Sie die Entstehung eines Spiels vom Entwurf bis zum fertigen Programm.

22 Auf dem Weg zu neuen Welten

Erleben Sie die Faszination der Vektor-Grafik. Hohe Geschwindigkeit, geringer Speicherverbrauch und schnelle Entwürfe sind die Stärke dieser Form der grafischen Berechnung.

Sprachen

28 Kreuzzüge

Es gibt eine unüberschaubare Zahl an Programmiersprachen. Welchem Guru sollen Sie folgen? Hier finden Sie »die« Lösung.

30 Brücke zwischen den Welten

C und Modula gelten unter Programmierern als zwei Welten. Diese müssen jedoch nicht unüberbrückbar bleiben. Frank Staudte erklärt wie's geht.

36 Ungleiche Brüder

Basic ist für fast jeden der Einstieg in die Programmierung des Amiga. Oft wird jedoch allzu schnell der Ruf nach einer anderen Programmiersprache laut. Jürgen Singer erleichtert Ihnen den Umstieg.

39 cc2lc - ganz einfach?

Aztec und der Lattice sind Compiler für dieselbe Sprache. Dennoch treten immer wieder Probleme bei der Konvertierung auf. Ralph Babel zeigt, wie Sie diese umgehen.

Know-how

44 Spiel mit Zahlen

Mathematische Spieltheorie ist eine unverzichtbare Grundlage für erfolgreiche Spieleprogrammierung. Lassen Sie sich von Jürgen Singer in die Welt der Zahlen und Spielzüge entführen.

53 Cola, Fanta, Sprite

Was sind Spiele ohne Sprites? Richtig, Suppe ohne Salz. Hier sehen Sie, wie Sie Hardwaresprites richtig einsetzen.

57 Sprites virtuell

Die Hardware des Amiga ist auf maximal acht Sprites festgelegt. Wenn das nicht ausreicht, müssen Sie auf VSprites zurückgreifen.

62 Blitters Objekte

Die flexibelste Form der Sprites sind die »BOBs«. Diese werden vom Blitter verwaltet und können daher leichter direkt beeinflusst werden.

66 Nimm zwei

Bewegte Grafiken mit statischen Komponenten werden mit nur einem Screen schnell zum Problem. Die Lösung ist einfach: Wir nehmen zwei und machen den oberen halb durchsichtig.

68 Flacker-Fixer

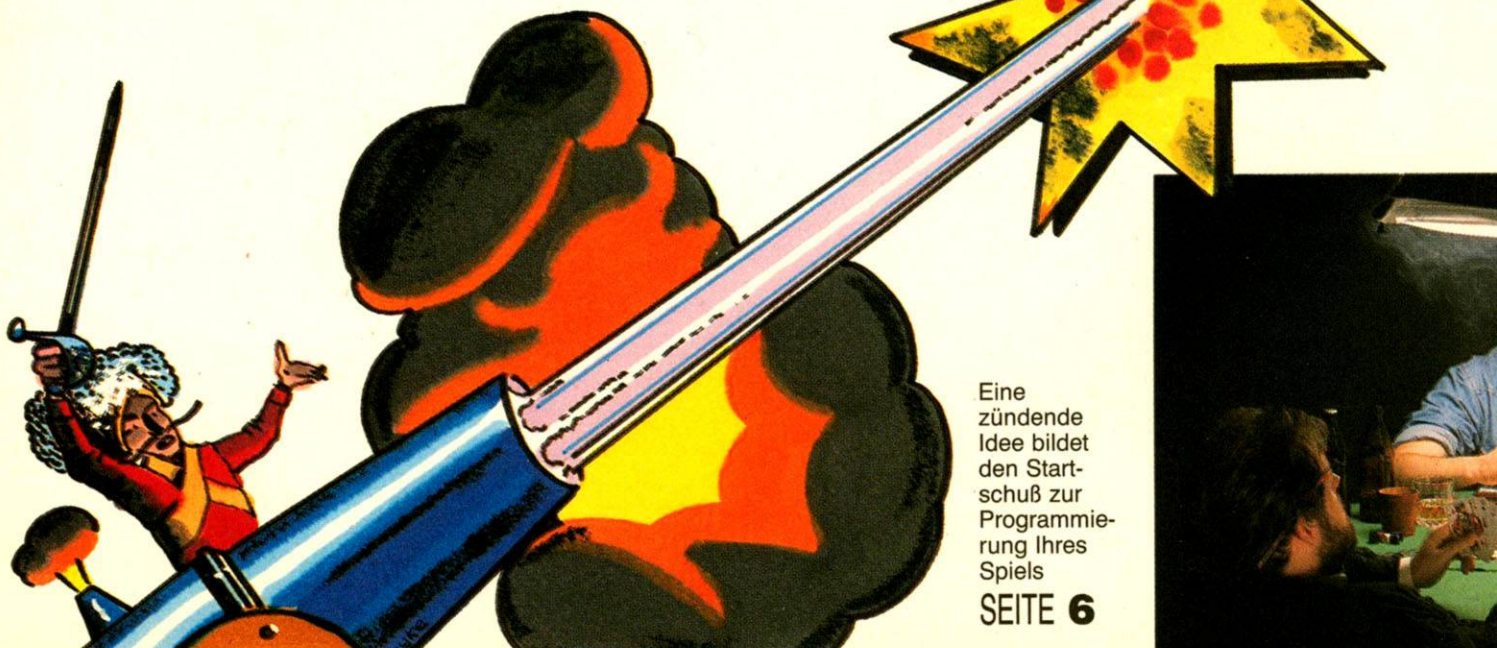
Bewegte Grafiken sind nur überzeugend, wenn sie fließend und ohne zu flackern über den Bildschirm gleiten. Double-Buffering macht's möglich.

72 Blitt-schnelle Grafik

Die Grafikfähigkeiten des Amiga werden vorwiegend einem einzigen Chip »angelastet«: dem Blitter. Hier erfahren Sie, wie Sie direkt auf diesen zugreifen.

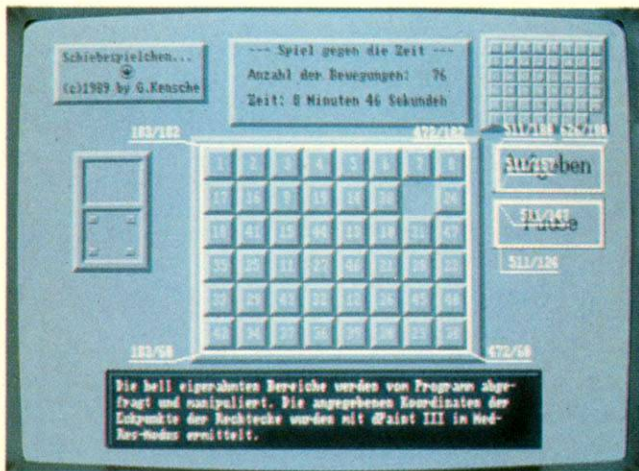
75 Kein Joy ohne Stick

Durch eine unsaubere oder langsame Joystickabfrage kann ein Spiel zum Wahnsinn treiben. Von Arno Gölzer erfahren Sie, wie Sie diesen »Lustknüppel« schnell und fehlerfrei abfragen.



Eine zündende Idee bildet den Startschuß zur Programmierung Ihres Spiels

SEITE 6



Schiebung ohne Ende – so einfach wie es aussieht, hat man gegen den Computer nicht gewonnen

SEITE 90

77 Grafik am laufenden Band

Betätigen Sie sich als Kulissenschieber. Mit »Superbitmaps« ist das kein Problem.

81 Intuition ausgebremsst

Für schnelle Grafikausgabe ist der Verwaltungsaufwand von Intuition eher im Weg. Also: Weg damit und ran an das eigene Display.

84 An der Coppercabana

16 Farben mit zwei Bitplanes – unmöglich sagen Sie? Kein Problem, wenn Sie die Hilfe des Copper bemühen.

Spiele

88 Der Sturz ins Bodenlose

»Wer raufsteigt, wird auch wieder runterfallen« – diese Weisheit ist bei »Fast Freddie« unverkennbar.

90 Schiebung

Zugegeben – das Spielprinzip ist nicht gerade neu. Aber lassen Sie sich von dieser gelungenen und reizvollen Umsetzung überraschen.

97 Zwilling gesucht

Sie haben ein gutes Gedächtnis? »Memory« wird Sie testen.



Statten Sie Ihr Spiel mit der richtigen Theorie aus – denn Erfolg ist kein Roulette-spiel

SEITE 44

107 Krieg den Kernen

»Reaktor« ist ein exzellentes Strategiespiel. Lösen Sie die Kettenreaktion aus, die Ihren Gegner vernichtet. Aber Vorsicht – er hat die Chance, sich zu rächen.

IFF

110 Pixelpracht leichtgemacht

Mit diesem IFF-Lademodul von Fridtjof Siebert laden Sie Ihre Grafiken problemlos und schnell.

121 Packen wir's an

Grafiken brauchen viel Platz und ebensoviel Zeit zum Laden. Abhilfe schafft ein guter IFF-Packer.

122 Der gute Ton zum Spiel

Eine »Ohrenweide« programmieren Sie mit diesen Soundmodulen in Modula-2.



»Fast Freddie« – das Spiel für Auf- und Absteiger, Umsteiger und Überflieger **SEITE 88**

133 Der Standard

Electronic Arts setzte mit dem Interchange File Format einen Standard zum Austausch von Daten. Hier finden Sie alles, was Sie wissen müssen.

Überblick

135 Spiele und kein Ende

Spiele sind das Thema für einen Computer wie den Amiga. Welche Sorten sind auf dem Markt? Lassen sich diese in Kategorien einteilen?

136 Von der Idee zum Erfolg

Vermarktungsstrategien für kommerzielle Spiele sind ein heißes Eisen. Martin Jobst hat Tips von erfahrenen Programmierern gesammelt.

Sonstiges

3 Editorial

138 Aus unserem Bücherregal

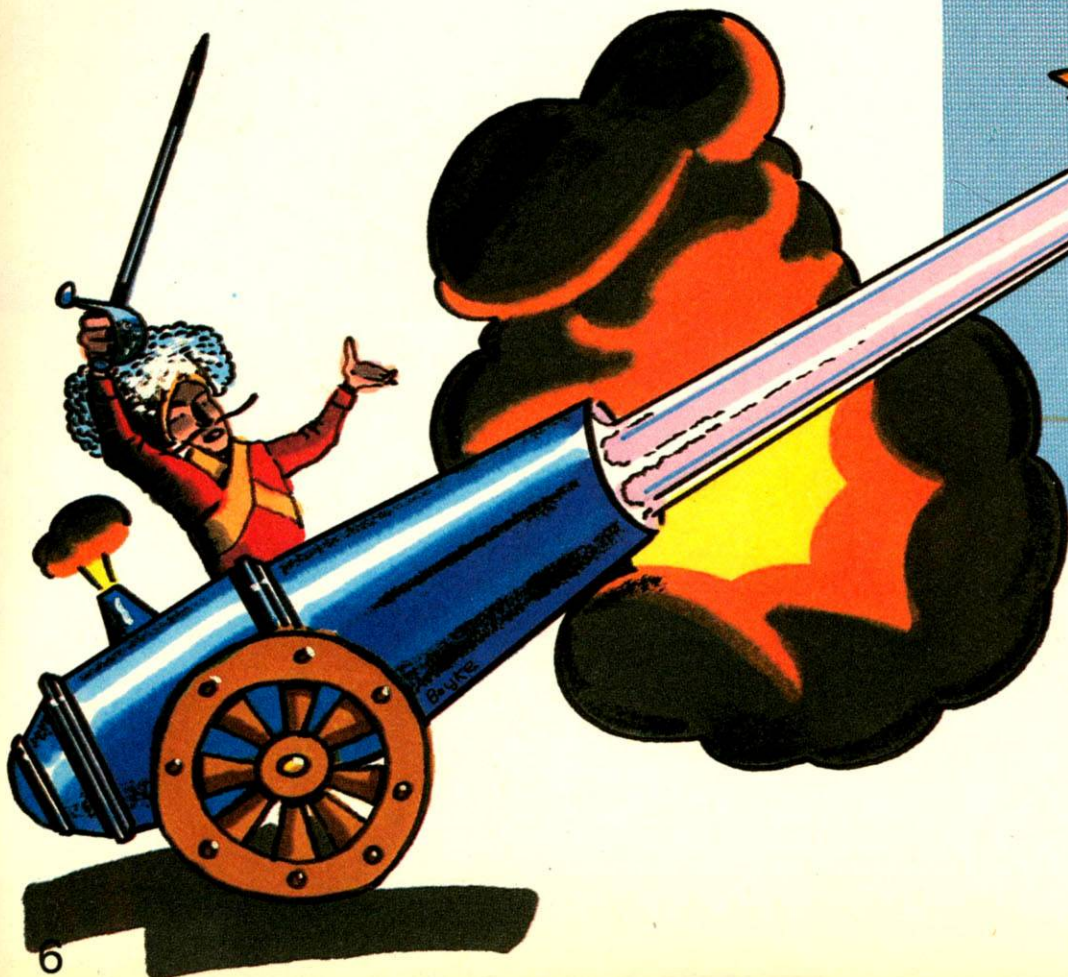
146 Impressum

Alle Programme aus Artikeln mit einem -Symbol finden Sie auch auf der Programm-service-Diskette zu diesem Sonderheft

Projektentwicklung

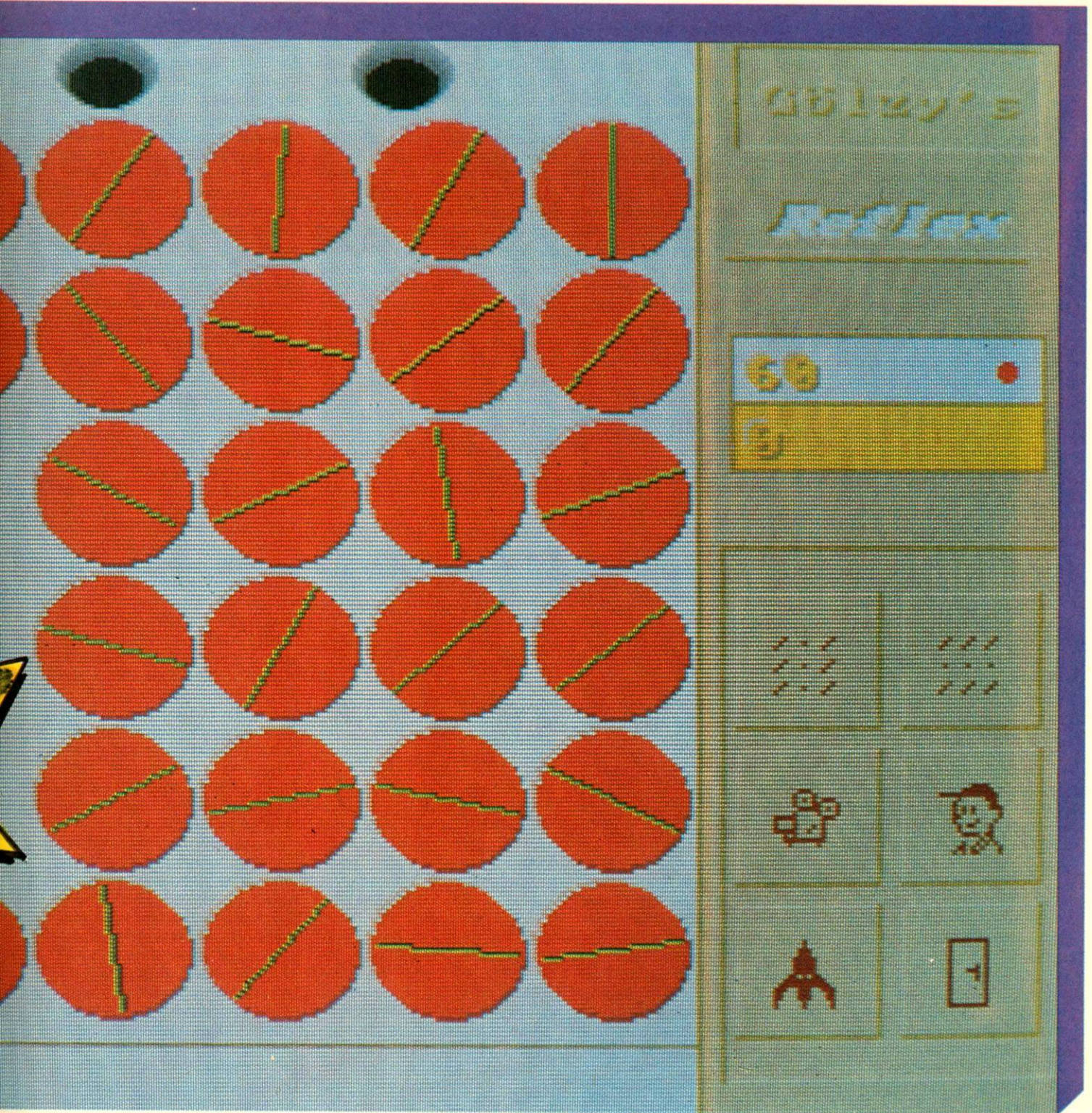
Am Anfang war die Idee

Wir zeigen Ihnen
anhand eines Beispiels, wie man
eine Spielidee in die
Wirklichkeit, sprich in ein
Programm, umsetzt.



Von Arno Gölzer

Geht es Ihnen nicht auch manchmal so? Man sitzt im Zug, geht spazieren oder liest gerade das neue Sonderheft und plötzlich – eine Idee! Eine Idee für ein neues Amiga-Spiel. Noch bevor das Computerzimmer, etwas außer Atem, erreicht ist, ist das Spiel schon deutlich vor Augen, doch kaum ist der Editor gela-



den, findet man keinen Anfang. Warum eigentlich?

Das Bild vor Augen ist das Wunschbild eines Spiels, man stellt es sich selbstverständlich mit der tollsten, blitzschnell animierten Grafik und einem Super-Stereosound vor. Wer jetzt in seinem Enthusiasmus planlos vorgeht, wird von der Problematik, die bei der Realisierung eines Spiels auftritt, förmlich erschlagen. Was jetzt fehlt

ist ein Konzept, welches das Problem der Realisierung in viele kleine, leicht überschaubare Teilprobleme zerlegt. Zunächst sollten Sie Ihre Idee verbalisieren und niederschreiben.

Bereits bei dieser Fixierung zerlegen Sie das Programm mehr oder weniger bewußt in Teilbereiche. Manchmal wissen Sie bereits, wie sich ein solcher Teilbereich in Ihre Lieblingspro-

grammiersprache umsetzen läßt. Notieren Sie sich die Lösung sofort und verweisen Sie im Text darauf. Stecken Sie Ihre Ziele ruhig hoch und nehmen Sie während der Beschreibung eines Bereichs keine Wertung vor! Killerphrasen, wie etwa »das schaffe ich nie« oder »kein Mensch würde das so machen«, sind verboten!

Halten Sie auch Ideen für Extras fest, die nichts mit dem ei-

gentlichen Spiel zu tun haben, wie zum Beispiel ein Modus für mehrere Spieler oder einen Entwurf für ein Titelbild. Sie werden feststellen, daß Sie nach dieser Beschreibung eine viel bessere Vorstellung von Ihrem Spiel bekommen werden. Neue Ideen fließen Ihnen zu, Sie inspirieren sich selbst. Einzelne Funktionen lassen sich bereits umsetzen, andere kann man zumindest, (vielleicht mit

Flußdiagrammen), näher umschreiben.

Bald wird eine erste spielbare Version Ihres selbstprogrammierten Spiels fertiggestellt sein. Diese wird dann verbessert und ausgeschmückt, bis das Programm Ihren Vorstellungen entspricht. Vielleicht sieht Ihr Projekt jetzt anders aus als zu Beginn vor dem geistigen Auge, aber – und das ist entscheidend – jetzt ist es real! Was hier in wenigen Zeilen beschrieben ist, dauert in Wirklichkeit natürlich einige Wochen und Monate, manchmal sogar Jahre. Wen wundert da der Groll der Programmierer gegen Raubkopierer, die sorglos die Früchte langer und harter Programmierarbeit verteilen...

Lassen Sie uns im Zeitraffer die Entwicklung eines Spiels am Beispiel Reflex nachvollziehen. Eine erste Beschreibung des Gedankenspiels könnte wie der Text im untenstehenden Kästchen formuliert sein – die Randnotizen sind in Klammern gesetzt.

Obwohl diese sehr kurze Beschreibung den Kern des Spiels nur grob beschreibt, lassen sich einige wichtige Informationen ablesen. Z.B., daß der Screen für Sprites vorbereitet sein muß. Ein erster Ansatzpunkt – wir beginnen mit dem Screen.

Überlegung: Erstellen wir über View ein eigenes Display oder nutzen wir einen Intuition-Screen? Der Text gibt Auskunft. Reflex erwartet viele Reaktionen vom Spieler, z. B. die Bewegung der Kanone oder Hindernisse drehen. Diese sind sehr einfach über das von Intuition zur Verfügung gestellte Message-System abzufragen. Darüber hinaus ist ein High-Score-Window gefordert. Wir brauchen Intuition also ohnehin, wenn wir den Programmieraufwand gering halten wollen. Und nicht zuletzt: Wir können uns

den »Verlust« des für Intuition notwendigen Speichers leisten, denn nach der Beschreibung dürfte Reflex auch auf einem Amiga mit der Grundversion von 512 KByte Arbeitsspeicher problemlos laufen. Also nutzen wir die einfache Handhabung der Intuitionroutinen und daher auch einen Intuition-Screen. Eine hohe Auflösung ist nicht unbedingt nötig, wir definieren einen LoRes-Screen mit schätzungsweise acht oder besser gleich 16 Farben. Falls noch mehr Farben nötig sind, ist das schnell geändert.

```
struct NewScreen nscr=
{ /* neuer LoRes-
Screen */
0,0,320,256,4,0,0,
SPRITES,CUSTOMSCREEN,
NULL,NULL,NULL,NULL
};
```

Bevor wir den neuen Screen öffnen, muß die Basisadresse der Intuition-Library bekannt sein. Das beste wird es sein, wenn wir alle diese Vorbereitungen (wie das Öffnen der Libraries, des Screens und der Windows etc.) in einer Funktion zusammenfassen: OpenW() (Listing 1) für »Open Work«.

Dies ist eine wichtige Information, die wir während der gesamten Entwicklungszeit für Reflex im Auge behalten müssen – gleich notieren!

Notiz: Vorbereitende Funktionsaufrufe in der Funktion OpenW() einordnen!

Apropos Windows – wie viele Fenster brauchen wir eigentlich? Laut Text sind es mindestens zwei: ein Spielfeld mit Bedienungselementen und ein High-Score-Window.

Überlegung: Der Spieler hat mit der geforderten Hilfsfunktion die Möglichkeit, Linien auf das Spielfeld zu zeichnen. Bei der Verwendung eines eigenen

Windows für das Spielfeld müssen wir hierbei nicht testen, ob sich eine Linie bereits außerhalb des Spielfeldes befindet, um sie dort zu »kappen«. Das übernimmt in diesem Fall das Betriebssystem für uns. Grund genug, um Spiel- und Bedienungsfeld zu trennen. Demnach benötigen wir vorläufig drei Windows. Widmen wir uns als erstes dem Spielfeld-Window (und schon ist wieder eine Aufgabe zerteilt!). Wir nutzen die gesamte Höhe des Screens. Wir möchten ein optisch ansprechendes quadratisches Spielfeld. Somit haben wir auch die Breite des Windows: Höhe = Breite = 256 Pixel.

```
struct NewWindow pwin=
{ /* Spielfeld */0,0,256,
256,0,1, MOUSEBUTTONS,
RMBTRAP,NULL,NULL,NULL,
NULL,NULL,0,0,0,0,
CUSTOMSCREEN
};
```

Das Window wird in OpenW() geöffnet. Vereinbaren wir doch gleich die Umkehrfunktion CloseW() (Listing 2). Darin schließen wir alle vom Betriebssystem angeforderten Quellen in umgekehrter Reihenfolge. Nach der Definition der GetMessage-Funktion, mit deren Hilfe wir die Intuition-Nachrichten empfangen, können wir einen Probestart wagen:

```
/*Reflex-Version 0.01*/
...
...
VOID main()
{
ULONG class;
BOOL Quit=0;
OpenW();
while(!Quit)
{
Wait(1«Play»->UserPort->mp_SigBit);
while(class=GetMessage
(DWin,0,0))
{
Quit=1;
}
}
CloseW();
}
```

In jeder Testversion wird die Versionsnummer erhöht und vermerkt. Kleine Änderungen erhöhen die Nummer im Hundertstelbereich, der Abschluß eines großen Teilgebiets im Zehntelbereich. Doch nach erfolgreicher Übersetzung, endlich der erste Probestart.

Enttäuscht wird man feststellen: Vom Window sieht man nur den Rahmen, und für das Be-

dienungsfeld bleibt daneben recht wenig Platz. Etwa ein Viertel der Screenbreite sollte das Bedienungsfeld schon messen. Das Beste ist es, beide Fenster gleich zu öffnen und dann deren Breite und Position zu optimieren. Eine Zeichnung auf kariertem Papier erleichtert diese Arbeit erheblich. Stimmen Position und Dimension der beiden Windows, erfolgt die Farbgebung. Auch sie ordnen wir in einer eigenen Funktion ein, denn nichts ändert man in einem Programm öfter als die Farben. Jetzt können die bisher leeren Window-Rahmen nach eigenem Geschmack gestaltet werden. Hierzu sind Grafikfunktionen und damit die Basisadresse der Graphics-Library nötig. Wir ergänzen OpenW() und CloseW() entsprechend. Der Aufbau des Reflex-Bildschirms übernimmt DrawScreen(). Das Spielfeld wird komplett weiß gefärbt und erhält einen einfarbigen dunklen Rahmen. An diesem Rahmen wird später der Kugelsprite reflektieren, der somit innerhalb der Spielfeldgrenzen bleibt. Das Bedienungsfeld färben wir dunkel mit einem hellen Rand links oben und einem dunklen Rand rechts unten. Damit erscheint das Feld reliefartig und von einer Lampe erhellt. Verschiedene Rahmen und die Schriften POut() (Listing 3) (erscheint aufgesetzt) und Pln() (Listing 3) (erscheint eingepreßt) verstärken den plastischen Eindruck.

Wichtig für das Gesamtbild ist, immer die Position der imaginären Lichtquelle im Auge zu behalten und die Farben der Rahmen, der Schriften oder anderer Bildschirmobjekte entsprechend zu setzen. Ein erneuter Testlauf stellt das verwöhnte Programmiererauge schon eher zufrieden. Die neue Version ist es sicher wert, zur Entspannung erst einmal Kaffee aufzusetzen und das Programm neben der Arbeitsdiskette auch auf einer anderen Diskette in einem eigenen Verzeichnis als Sicherheitskopie »zu verewigen«. Manchem wird dieser Aufwand als übertrieben erscheinen. Aber glauben Sie mir, wenn ich behaupte, daß sich dieser Aufwand lohnt. Denn wenn sich, aus welchen Gründen auch immer, Ihr Amiga dazu entschließt, Ihre Arbeitsdiskette nach eigenen Vorstellungen zu gestalten, haben Sie das Nachsehen. Nachdem der Kaffee endlich durchgelaufen ist, kann es mit der Gestal-

Erste Beschreibung vom 1.6.1989

Eine Kugel (Sprite) wird aus einer in X-Richtung beweglichen Kanone (Sprite, mit der Maus anklicken und bewegen) abgefeuert. Vor dem Ziel sind drehbare Hindernisse (Linien im COMPLEMENT-Modus gezeichnet) angebracht. Die Kugel soll in einer Art Kettenreaktion an möglichst vielen Hindernissen abprallen und dann ihr Ziel finden (Anzahl der getroffenen Hindernisse in die Bewertung eingehen lassen). Extras:

- Geräusche, z.B. beim Abprallen der Kugel an einem Hindernis (übernimmt Gölyz »SEED«)
- 2-Spieler-Option (umschalten über Gadgets)
- jederzeit einsehbare High-Score-Liste (eigenes Window)
- Hilfsfunktion, die zeigt, wie die Kugel an einer Barriere reflektiert wird (Linien im COMPLEMENT-Modus, gleiche Berechnung wie bei Kollision der Kugel)

tung des Spielfeldes weitergehen. Hier ist folgendes vorgegeben:

- a) Eine Kanone soll in X-Richtung beweglich sein. Wir bringen die Kanone am unteren Spielfeldrand an. Die Schußrichtung ist daher senkrecht nach oben.
- b) Ein Ziel muß sichtbar sein. Natürlich möchten wir kein Kampfspiel programmieren, dessen Spielziel es ist, möglichst viele kleine Männchen (oder Frauchen) abzuschießen. Ganz im Gegenteil: Das Ziel soll eine einfache Höhle oder Grube sein, in der der Spieler die Kugel versenken muß, bevor sie durch eine Explosion Schaden anrichten kann.
- c) Hindernisse oder Barrieren sollen das Ziel versperren.

Sie sehen, die Teilaufgaben werden kleiner und damit auch leichter überschaubar. Sie lassen sich so schnell und einfach erledigen, was durchaus motivierend wirkt. Wir arbeiten uns von vorne her durch: Punkt a) bereitet fast keine Probleme. Die Kanone ist mit einem Spri-teeditor oder auf einem karierten Blatt Papier schnell entworfen. Wenn wir gerade dabei sind, können wir auch gleich die Kugel entwerfen. Es fehlt dann nur noch die Routine, welche die Kanone bei einem Mausklick an den Pointer »klebt«. `MoveK()` (Listing 4) erledigt das für uns. Hier zeichnen wir auch eine »Stange«, auf der die Kanone befestigt ist.

Bevor es weitergeht, schnell noch ein geistiger Probelauf:

An der Unterkante des Spielfelds ist eine Stange angebracht. Darauf ist eine Kanone, in Stangenrichtung verschiebbar montiert. Linke Maustaste drücken... Kanone positionieren... Klick links und Schuß. Mist!

Beim Klick hat sich kein Schuß gelöst, sondern die Kanone hat sich bewegt!

Notiz: Schuß auslösen mit rechter Maustaste.

Noch ein Probelauf in Gedanken: Linke Maustaste drücken... Kanone positionieren... Klick rechts und Schuß. Jaaa! Man sieht die Kugel förmlich fliegen. Sie erreicht den oberen Bildschirmrand... reflektiert dort... kommt zurück... und trifft die Kanone!!!

Notiz: Kanone mittels einer Linie in der Farbe des Spielfeldrandes vom Spielfeld trennen.

Ein letzter Testflug zeigt, daß die Kugel sowohl am oberen Rand als auch an der Trennlinie abprallt. Nach entsprechenden Änderungen und Ergänzungen zeigt ein realer Probelauf, daß wir die while-Schleife in `main()` noch abändern müssen, denn gleich nach einem Mausklick endet Reflex in dieser Version. In `OpenW()` erhalten verschiedene Variablen die Signalbits der einzelnen Windows. Mittels `Wait()` erkennen wir, welches Window Nachrichten sendet. In der momentanen Version 0.04 beenden wir das Programm, sobald das Bedienungs-Window eine Nachricht liefert. Weiter geht es laut Plan mit Punkt b): Es geht um das Ziel. Sehr einfach, am oberen Rand des Spielfeldes zeichnen wir mittig ein Loch als Ziel. In der Höhe richten wir uns nach dem am unteren Bildschirmrand abgeteilten Kanonenbereich, um einen einigermaßen symmetrischen Bildschirmaufbau zu erreichen. Allerdings wird das Loch dann ziemlich klein und daher nicht einfach zu treffen. Was tun? Reflex soll ja nicht frustrierend, sondern entspannend wirken.

Klar! Wir zeichnen Löcher am besten gleich drei Stück.

Die Funktion `DrawHole()` (Listing 5) zeichnet ein Loch an der als Parameter gelieferten X-Position, so daß sich die Anzahl noch variieren läßt.

Geistiger Testflug: Kanone bewegen... Schuß... Treffer! Das Ziel wurde vor der Detonation der Kugel erreicht! Schade jedoch, daß die Explosion nicht zu sehen war. Von solchen kleinen Effekten »lebt« aber ein Spiel. Wir richten `DrawHole()` so ein, daß sie bei Bedarf auch ein »von innen erhelltes« Loch zeichnen kann. Ein abwechselndes Aufrufen der Funktion im Normal- und im Flushmodus erweckt den Eindruck einer unterirdischen Explosion, deren Blitze man durch die Höhlenöffnung sehen kann.

Notiz: Bei Treffer Explosion sichtbar machen, Soundeffekte könnten den Eindruck noch unterstützen.

Es fehlt noch Punkt c). Die Spielfeldmitte ist in ihrer ganzen Breite frei für Barrieren. In der Projekt-Beschreibung (im Kasten) finden wir für die Hindernisse den Hinweis »Linien im COMPLEMENT-Modus gezeichnet«. Also los: Wenn wir für eine Linie eine Breite von 30

Pixel annehmen und dabei berücksichtigen, daß der Spieler jede dieser Linien um den Mittelpunkt drehen kann, erreichen wir, daß auf dem Feld 42 Barrieren in sechs Reihen Platz finden. Eine Schleife in `DrawPlay()` (Listing 6) zeichnet uns die Hindernisse ein, und die `InitCyl()`-Funktion (Listing 7) sorgt für eine neue Ausgangsstellung der Barrieren. Ein Probe-start zeigt, daß das Spielfeld trotz der vielen Striche leer und wenig ansprechend wirkt. Also montieren wir die Hindernisse auf einzelne Sockel. Da die Linien später um ihren Mittelpunkt kreisförmig beweglich sind, wählen wir eine zylindrische Form. Bei der Konstruktion der Sockel berücksichtigen wir wieder die imaginäre Lampe in der linken oberen Screen-Ecke. Somit ist das Spielfeld schon fertig aufgebaut. Ein Manko hat es allerdings noch: Das Zeichnen der 42 Zylinder mit Barrieren dauert zu lange. Ein kleiner Kunstgriff erspart dem Spieler die Wartezeit: Wir zeichnen den Zylinder nur ein einziges Mal auf einen versteckten RastPort und kopieren ihn dann mittels einer Blitterfunktion auf das Spielfeld.

`MakeCyl()` (Listing 8) zeichnet einen roten und einen gelben Zylinder für Explosionen auf den RastPort eines Hilfs-windows. Die Funktion `DrawCyl()` (Listing 9) clippt die Grafik dann an die gewünschte Stelle im Spielfeld. Der Aufruf des neu übersetzten Programms bestätigt: Das Feld ist in einem Bruchteil einer Sekunde aufgebaut. Das Zeichnen der Barrieren im COMPLEMENT-Modus übernimmt ebenfalls `DrawCyl()`, allerdings indirekt, durch den Aufruf der Funktion `DrawBar()` (Listing 10). Um eine Barriere an der richtigen Position, in der richtigen Größe und im richtigen Winkel zeichnen zu können, benötigen wir die entsprechenden Daten für jede Barriere. Wir fassen sie, der Übersichtlichkeit wegen, in einer Struktur zusammen. So sollte man mit zusammengehörenden Daten immer verfahren. Dazu gehört natürlich die Beschreibung der einzelnen Daten in Form eines Kommentars.

Nur kommentierte Quellcodes lassen sich auch nach einem längeren Zeitraum noch problemlos lesen!

Nach der gewohnten Sicherheitskopie haben wir innerhalb unserer Programmentwicklung einen größeren Bereich, sprich den Spielfeldaufbau, abge-

schlossen – Grund genug, um die Versionsnummer auf 0.10 zu erhöhen. Womit geht es weiter? Stellen wir die verbleibenden Teilbereiche gegenüber und werten sie aus:

Ist es sinnvoll, mit der High-Score-Liste zu beginnen? Sicher nicht, wir könnten sie zwar anlegen, aber nicht testen, da Reflex noch keine Punkte errechnen kann. Sollten wir uns dem Bedienungsfeld widmen? Auch hier fehlen zu viele Informationen. Also bleiben wir beim Spielfeld und kümmern uns um die dort fehlenden Funktionen. Für das weitere Vorgehen stellen wir uns wieder einen Plan auf:

- a) Das Einstellen der Hindernisse ermöglichen
- b) Kugelsprite bewegen = Schuß
- c) Kollisionsabfrage und Reaktion an Hindernis und Spielfeldrand
- d) Bewertung, Punktevergabe

Bei der Aufstellung einer solchen Liste sollte man gleich auf eine logische Reihenfolge achten. Verständlicherweise muß vor der Formulierung der Kollisionsabfrage die Kugel beweglich sein, um diese auch testen zu können. Steht die Liste, so verfährt man wie gewohnt. Das erste Problem aufgreifen und zerlegen:

a) Die Drehung der Hindernisse soll mit der Maus erfolgen. Nichts ist für den Benutzer eines Programms ärgerlicher, als ständig zwischen Maus und Tastatur wechseln zu müssen. Da beide Maustasten bereits mit »Kanonenfunktionen« belegt sind, bleiben uns für die Auswertung zwei Möglichkeiten: Entweder wir lassen den Spieler über ein Modify-Gadget mitteilen, daß er die Barrieren drehen möchte, oder wir unterscheiden räumlich zwischen dem Bereich, in dem die Kanonen beweglich sind, und dem eigentlichen Spielbereich.

Die erste Methode ist für uns als Programmierer einfacher. Ist das Modify-Gadget selektiert und die linke Maustaste gedrückt, wissen wir, daß der Spieler den Winkel einer Barriere ändern möchte. Jedoch ist es für den Spieler umständlich, während des Spiels ständig Einstellungen vornehmen zu müssen. Also unterscheiden wir in der Funktion `CheckMouse()` (Listing 11) zwischen Kanonen- und Spielbereich. Außerdem prüfen wir dort, ob die linke oder die rechte Maustaste betätigt wurde. Wir unterscheiden vier mögliche Fälle:

- 1) Linke Taste, Kanonenbereich: Kanone bewegen
- 2) Rechte Taste, Kanonenbereich: Kanone abschießen
- 3) Linke Taste, Spielbereich: Hindernis drehen
- 4) Rechte Taste, Spielbereich: Hat vorerst keine Funktion.

Eine Funktion haben wir noch frei. Die Hilfsfunktion könnte hier gut passen. Stellen Sie sich vor, Sie richten gerade mit der linken Taste eine Barriere aus und können mit der rechten Taste gleich überprüfen, ob der Winkel in etwa stimmt.

Notiz: Rechte Maustaste im Spielbereich für Hilfsfunktion reservieren.

Sofort wird die Realisierung der Hilfsfunktion in unseren neu aufgestellten Plan über die weitere Vorgehensweise unter e) aufgenommen. Stellen wir die Betätigung der linken Maustaste im Spielbereich fest, rufen wir die Funktion `TurnBar()` (Listing 12) auf, welche den Winkel errechnet und das Hindernis, durch den Aufruf von `DrawBar()`, neu darstellt. Ein Testlauf zeigt uns eine Besonderheit: Klickt man auf den Drehpunkt der Linie, wird diese aufgrund der Berechnung als Punkt dargestellt! Eine Besonderheit, die besonders der Anfänger gerne nutzen wird, da er aus der Vielfalt der Hindernisse einige gezielt »ausschalten« kann. Man könnte sie sogar noch etwas ausbauen, indem man Funktionen zur Verfügung stellt, die ganze Hindernis-Spalten und -Reihen ausschalten.

Notiz: Funktionen zum Ausschalten von Hindernissen zur Verfügung stellen.

Sie sehen, während der Programmentwicklung ergeben sich völlig neue Ideen, die das Spiel bereichern können. Wir kommen zur Realisierung von Punkt b) unserer Liste. Die Spritedaten der Kugel liegen uns bereits vor, hier haben wir ja schon Vorarbeit geleistet. Sobald uns Intuition im Kanonenbereich die Betätigung der rechten Maustaste meldet, verzweigen wir nach `Schuss()`. Diese Funktion übernimmt alle notwendigen Voreinstellungen für das Bewegen der Kugel. Die eigentliche Bewegung übernimmt `MoveK()`, indem die Funktion zur aktuellen Spriteposition Geschwindigkeitsvektoren in X- und Y-Richtung addiert. Zu Beginn ist der X-Vektor

gleich null und der Y-Vektor negativ. Daraus resultiert eine Kugelbewegung nach oben.

Notiz: Eine Veränderung der Vektoren bewirkt einen Richtungswechsel (wichtig für die Reaktion nach der Kollision mit einem Hindernis oder dem Spielfeldrand).

Schnell mal übersetzen und einen Probestart wagen! Das Drehen der Hindernisse sieht nicht übel aus. Aber jetzt auf zum ersten Schuß:

Kanone positioniert... rechte Maustaste gedrückt, und schon fliegt die Kugel los - SUPER!... sie reflektiert am Rand oben... am Rand unten... und fliegt und reflektiert und fliegt und... und hört gar nicht mehr auf!

Natürlich: Die Verlangsamung und schließlich den Stillstand der Kugel müssen wir auch noch eingeben. Jetzt hilft nur noch ein Reset, um das Programm zu beenden. Das schnelle Übersetzen zwischen durch hat sich zeitlich nicht gelohnt! Wir ergänzen `MoveK()` durch einige Zeilen, welche durch die ständige Verminderung der Geschwindigkeitsvektoren die Kugel verlangsamen. Unter einem bestimmten Wert setzen wir die Vektoren gleich 0 - die Kugel bewegt sich nicht mehr - hier endet der mißglückte Versuch, das Ziel zu treffen.

Diese Stelle markieren wir uns im Listing, da hier noch eine entsprechende Reaktion folgen muß. Wie müßte diese Reaktion aussehen? Wir haben festgelegt, daß die Kugel das Zielloch erreichen muß, bevor sie Schaden anrichtet. Das heißt, sie muß in diesem Fall, da das Ziel nicht erreicht wurde, etwas anrichten. Wir setzen zu Testzwecken einfach ein `DisplayBeep()` ein.

Notiz: Das Ziel wurde nicht erreicht: Löschen der Kugel und Zerstörung einiger Zylinder - der Spieler kann immer weniger Punkte erreichen, da die Zahl der Hindernisse abnimmt - daraus folgt: Spielende nach Verlust des letzten Zylinders.

Das bringt uns auch gleich zu dem nächsten Punkt: die Abfrage unserer Ziele. Befindet sich die Kugel im Bereich der Löcher am oberen Spielfeldrand, kontrollieren wir innerhalb `MoveK()`, mittels eines Aufrufs der `CheckH()`-Funktion (Listing

13), ob sich der Kugelsprite über einer Farbe befindet, die für das Zeichnen der Löcher benutzt wurde. Ist dies der Fall, leitet `CheckH()` die Kugel ins Zentrum des Lochs, um sie dort zu löschen - was für den Spieler wie ein Fallen der Kugel in ein tiefes Loch aussieht.

Ein Blick in unsere Notizen erinnert uns: Hier ist eine (unterirdische) Explosion vorgesehen. Die for-Schleife innerhalb `CheckH()` erzeugt sie - das Loch wird so gezeichnet, als sei es durch Blitze erleuchtet. Der Vermerk wird nur teilweise gestrichen, er enthält noch die Anregung »Soundeffekt«.

Damit können wir Punkt b) unserer Liste abschließen und uns auf die Kollisionsabfrage, Punkt c), konzentrieren. Wir müssen, um den Lauf der Kugel genau kontrollieren zu können, vor jeder Bewegung der Kugel prüfen, ob sich an der neuen Kugel-Position ein Hindernis oder der Feldrand befindet. Der Funktionsaufruf `CheckB()` (Listing 14) ist hierfür vorgesehen. Dort erfolgt auch die Neuberechnung der Geschwindigkeitsvektoren, die ja laut Notiz die Richtung der Kugel ändert. Jetzt dürfte bei einem erneuten Probelauf nichts mehr schiefgehen. Die Kugel sollte am Feldrand und an den Hindernissen korrekt reflektieren. Sie müßte ständig langsamer werden und schließlich mit einem Flush (`DisplayBeep()`) oder, falls der Schuß gut war, mit einer Explosion in einem der Löcher verschwinden. Mal sehen... zuerst jedoch eine Sicherheitskopie. Und frischen Kaffee aufsetzen. Ein Test der neuesten Reflex-Version zeigt noch einige Fehler auf, aber auch das gehört zum Programmieralltag und läßt uns völlig cool bleiben. Wir wissen ja, in welchen Routinen der Fehler steckt, und nach einiger Zeit kann erneut kompiliert werden. Danach ist auch der Kaffee fertig, und einem erneuten Test steht nichts mehr im Wege.

Diesmal läuft das Programm fehlerfrei, zumindest programmtechnisch. Aber ein ganz anderer »Fehler« macht sich bemerkbar: Hat man einmal beim Spielen eine gute Einstellung der Hindernisse gefunden und die Kanone richtig positioniert, könnte man beliebig viele Punkte machen und sämtliche High-Score-Listen sprengen, indem man nichts an der erprobten Einstellung ändert und immer wieder von derselben Stelle aus feuert.

Auch daran muß man sich gewöhnen. Manchmal treten Situationen auf, mit denen man vorher nicht gerechnet hatte. An solchen Stellen darf man die Flinte nicht ins Korn werfen. Wenn sich in einer verfahrenen Situation scheinbar keine Lösung anbietet, schalten Sie den Rechner aus und versuchen es am nächsten Tag erneut. Sie werden dann sicher eine annehmbare Lösung dafür finden. Wie sieht die Lösung für unser Problem aus?

Man könnte die Stellung der Hindernisse oder der Kanone nach jedem Schuß verändern, das wäre jedoch sicher nicht die Lösung, um den Spieler an den Bildschirm zu fesseln. Es gibt (mindestens) eine andere Lösung: Die Kugel soll »unter der Wucht des Aufpralls« das erste Hindernis zerstören, und schon ist die ganze schöne Einstellung dahin - schade, schade, schade!

Natürlich muß diese Explosion zu sehen sein. Wir stellen den Zylinder gelb dar, warten kurz und löschen ihn mit der Funktion `KillCyl()` (Listing 15). In einer vierfach verschachtelten Schleife lassen wir die »Trümmer« kreisförmig vom Explosionsherd davonschweben. Die Daten für die Explosion berechnen wir aus Gründen der Ausführungsgeschwindigkeit in `InitData()` (Listing 16) vor. Ein Test zeigt uns, daß zwar die Explosion recht gut aussieht, aber das Spiel in der Zeit, in der sie abläuft, stillsteht. Glücklicherweise ist der Amiga multitaskingfähig. Das ist die Lösung! Spiel und Explosion lassen sich durch den Start eines eigenen Prozesses, mit Hilfe der Funktion `CreateFunctionProc()`, scheinbar gleichzeitig abarbeiten. Die Prozeß-Funktion, sie heißt `TFunc()` (Listing 17), ist zur Verdeutlichung des Sub-Prozesses als einzige Funktion hinter `main()` angeordnet. Die Kommunikation mit dem Sub-Prozeß findet über den unbenutzten User-Port des Hilfswindows statt. Empfängt der Prozeß die Nachricht `FIRSTCOLLISION`, so startet er die Explosion. Wenn Sie sich über diese Technik genauer informieren wollen, empfehlen wir entweder ein genaues Studium des Listings oder einen Blick in das nächste AMIGA-Magazin, das Ende Oktober am Kiosk liegt. Sie finden dort einen Artikel, der ausführlich auf dieses Thema eingeht. Könnte man diese Explosion nicht auch zur Eliminierung der drei Zylinder beim

Nichterreichen des Ziels einsetzen? Das ist sicher möglich, würde aber sicher mit der Zeit langweilig wirken. Da muß eine neue Idee her. Die Lösung: Die Zylinder zerbröckeln durch den Druck der Explosion.

RectFill() füllt den Bereich der drei nächsten Zylinder, die wir mit GetCylNr() (Listing 18) ermitteln, mit kleinen Rechtecken in der Farbe des Spielfeldes aus. Die Positionen der Rechtecke sollen zufällig verteilt sein. Die Ermittlung der Zufallszahlen (wir benötigen gleich 500 Stück!) dauert natürlich wieder einige Zeit, so daß das Füllen sehr langsam geschieht. Also müssen wir die Zufallszahlen vorberechnen. Eine weitere Aufgabe für InitData(). Nach einem sehr ausge dehnten Punkt c) soll laut unserer Liste die Umsetzung der Bewertung eines Versuchs folgen. Dies ist eine einfache Aufgabe: Eine globale Variable wird bei jeder Kollision mit einem Hindernis um eins inkrementiert. Wurde die Kugel versenkt, errechnet sich die Punktzahl in CheckH() aus dem Produkt der Kollisionsanzahl und der Punkte für das Zielloch. Das mittlere Loch wird mit zehn und die äußeren mit fünf Punkten bewertet. Da später einmal zwei Spieler teilnehmen können, sichern wir die Punktzahl gleich in einem Array. Die Variable »Player« enthält die Spielernummer -1 und kann so als Index für dieses Array dienen. Die Punktezahl für den jeweiligen Spieler geben wir auf dem Bedienungsfeld aus. Am besten gleich über dem Gadget zum Umschalten der Teilnehmerzahl. Die Ausgabe der Punkte muß sicher später noch angepaßt und damit öfter verändert werden. Wir richten deshalb hierfür eine eigene Funktion mit Namen SetP() (Listing 19) ein.

Somit sind fast alle Punkte unserer Liste abgearbeitet. Es fehlt nur der zusätzlich eingefügte Punkt e), der die Hilfsfunktion betrifft. Unsere Notizen erinnern uns: Der Spieler aktiviert die Hilfsfunktion mit der rechten Maustaste. Es sollen dann Linien im COMPLETE-Modus erscheinen, die zeigen, wie die Kugel an einem Hindernis abprallt. Dafür können wir die gleichen Berechnungen wie in CheckB() verwenden. Sie sehen, in diesem Fall enthalten die Notizen sogar die Lösung der Vorgabe! Wir kopieren die entsprechenden Zeilen aus CheckB() in eine neue Funktion namens Help()

(Listing 20). Beim Testen der Funktion fällt auf, daß unsere Überlegung, das eigene Spielfeld-Window betreffend, richtig war. Lange Linien, die eigentlich über das Spielfeld ragen würden, werden ab dem Spielfeldrand automatisch nicht mehr dargestellt. Alle Punkte der Liste sind jetzt abgehakt, ein großer Bereich unseres Spiels ist fertiggestellt. Es wird Zeit die Notizen auszuwerten. Einige Punkte sind noch nicht berücksichtigt:

- a) Soundeffekte für Explosionen einbauen
- b) Wenn alle Zylinder zerstört sind, ist Spielende
- c) Hilfsfunktion zum Ausschalten von Hindernissen

Sofort ergibt sich eine neue Liste, die das weitere Vorgehen vorschreibt. Die Daten für die Soundeffekte sind, mit Hilfe des Soundeffekte-Editors aus Sonderheft Nummer 4, in kurzer Zeit erstellt. Wir definieren eine Funktion mit Namen PlaySound() (Listing 21), die uns alle Geräusche abspielt. Für jedes Ereignis, an der betreffenden Stelle im Quellcode durch Kommentare gekennzeichnet, rufen wir die Funktion mit den entsprechenden Parametern auf. Es ist sinnvoll, Kommentare an noch zu ergänzenden Stellen oder solchen, die an etwas erinnern sollen, besonders zu kennzeichnen. Etwa so: /* */. Sie müssen dann, um alle wichtigen Stellen im Quellcode zu finden, nur nach dieser Zeichenfolge suchen.

Was das Spielende betrifft, so müssen wir nur nach jedem Schuß (Funktion Schuss()) (Listing 22) prüfen, ob noch Zylinder übrig sind. Dies geschieht durch den Aufruf der GetCylNr()-Funktion. Liefert sie den Wert -1, verzweigt das Programm nach GameOver() (Listing 23). Diese Funktion stellt das Ergebnis grafisch dar. Später soll sich der Spieler, bei einem besonders guten Ergebnis, in die High-Score-Liste eintragen können.

Notiz: High-Score-Liste bei Spielende darstellen und eventuell ergänzen lassen.

Nach einer Bestätigung baut DrawPlay() (Listing 6) das Spielfeld neu auf. Diese Funktion könnten wir auch einsetzen, wenn der Spieler während des Spiels noch einmal von vorne beginnen möchte.

Notiz: Funktion »Neustart« einbauen.

Diese Funktion und auch die zum Ausschalten der Hindernisse sollten über Gadgets im Bedienungsfeld zu erreichen sein und nicht über schwer einprägsame Tastenkombinationen. Der Spieler muß dann nicht ständig zwischen Maus und Tastatur wechseln. Wieviele Gadgets sind eigentlich insgesamt anzubringen?

Laut unserer Notizen sind es genau sieben: Zwei Gadgets zum Umschalten der Spielerzahl, auf denen auch die Punktzahl des jeweiligen Spielers erscheint, zwei Gadgets zum Ausschalten einer Reihe, zwei zum Ausschalten einer Spalte von Hindernissen und ein Neustart-Gadget. Hinzu kommt noch eines zum Verlassen des Spiels. Weiter könnte man noch einen Trainer einbauen, der, sobald er aktiv ist, eine Spielsituation beliebig oft wiederholt. Bei dem Entwurf des Bedienungsfeldes hilft eine Skizze auf kariertem Blatt weiter. Die Breite und Höhe des Windows sind bekannt. Zeichnen Sie die Bedienungselemente an den gewünschten Positionen in ein Koordinatensystem ein und geben Sie den Achsen die Einheit »Pixel«. Im Beispiel macht die ungerade Gadget-Anzahl Probleme. Sie lassen sich einfach nicht optisch ansprechend positionieren. Vielleicht sollte man eine Funktion streichen? Nein, besser noch eine einfügen.

Notiz: Funktion Replay (wiederholt den letzten Schuß) einbauen.

Nachdem die Gadgets nun endlich an den richtigen Positionen montiert sind, werden sie mit Borders versehen. Dabei berücksichtigen wir natürlich auch wieder unsere Lichtquelle. Links oben erhält das Gadget einen hellen, an den gegenüberliegenden Seiten einen dunklen Rand. Es erscheint aufgesetzt. Wählt nun der Spieler ein Gadget an, vertauschen wir die Ränder. Der rechte und der untere Rand scheinen erhellt. Als Bediener hat man dadurch den Eindruck, als könne man das Gadget tatsächlich niederdrücken wie einen Knopf. Weiter sollen die Gadgets Images erhalten, kleine Bilder also, die auf die Gadgetfunktion hindeuten. Danach geht es an die Realisierung der

einzelnen Funktionen. Wie besprochen steht am Anfang das Aufstellen einer Liste. In diesem Fall ist sie sogar, in Form des Bedienungsfeldes, auf dem Bildschirm sichtbar.

Die Funktion CheckGadgets() (Listing 24) reagiert auf die Anwahl der Gadgets. Als Unterscheidung dient deren Kennnummer. Auf jede der acht möglichen Nummern muß korrekt reagiert werden. Die Umschaltung der Spieleranzahl erfolgt nach der Wahl eines Gadgets mit der Nummer null oder eins. Der empfangenen Kennnummer entsprechend erhält die Variable »PlayerNr« die Zahl der Mitspieler -1. Eine andere Variable, sie heißt »Player«, gibt an, wer am Zug ist. Sie wird in Schuss() umgeschaltet. SetP() zeichnet in Abhängigkeit des Wertes dieser Variable einen Punkt über Gadget 0 oder 1. So erkennen die Spieler, wer gerade am Schuß ist.

Die Funktionen zum Ausschalten der Hindernisse sind ebenfalls schnell installiert. Sie werden bei der Wahl der Gadgets mit den Nummern zwei und drei aktiviert. Ist eines der Gadgets (oder beide) selektiert, schaltet CheckMouse() (Listing 11) je nach Wunsch eine Hindernisspalte oder -reihe aus. Hierzu reicht der Aufruf von TurnBar() mit den Drehpunkten der Linien als Parameter. Ein Probelauf zeigt, daß man sehr leicht vergißt, die Hilfsfunktion wieder auszuschaalten. Wurden Hindernisse ungewollt ausgeschaltet, sind alle sorgsam vorgenommenen Einstellungen verloren.

Ahnhilfe: Eine Rotfärbung des Hintergrunds zeigt den speziellen Modus an und warnt den Spieler vor unüberlegten Tastendrücken. Das Gadget mit der Kennnummer vier soll die Replay-Funktion aktivieren. Bei jedem Aufruf füllt die Funktion Schuss() ein Array mit den Ausgangsdaten des gerade erfolgten Versuchs. Die alte Spielszene kann, mit Hilfe dieser Daten, bei der Wahl des Replay-Schalters rekonstruiert und der Schuß wiederholt werden. Die Trainerfunktion soll dem Spieler ermöglichen, jede Spielsituation beliebig oft zu wiederholen. Natürlich erhält er in diesem Modus keine Punkte. Zur Verwirklichung des Trainers können wir die Erfahrungen, welche wir bei der Entwicklung von Replay gemacht haben, verwenden. Der Spieler darf nach der Aktivierung des Trainerschalters an Kanone und

SuperbaseTM

FÜR JEDEN GELDBEUTEL

DM89,-*

ca. (sFr 81,90 / öS 757,-)

BOOK-
WARE

AMIGA
Superbase

Amiga Superbase (Bookware)

Wegen seiner Bedienerfreundlichkeit die optimale Einsteiger-Datenbank. Per Mausclick erstellen Sie Ihre Datenmaske. Aufnahme, Auswahl und Ausgabe der Daten erfolgen über ein Bedienerfeld, das dem eines Videorecorders sehr ähnlich ist. Selbstverständlich können Sie Amiga-Grafiken ebenso verwalten und anzeigen lassen wie Zahlen und Texte - Briefmarkensammler zum Beispiel könnten somit die digitalisierten Abbilder ihrer Schätze mit den zugehörigen Daten auf den Monitor bringen. Das System ist relational; Sie können also Einträge verschiedener Datenbanken miteinander verknüpfen. Die Daten lassen sich als Formular oder Liste auf dem Bildschirm oder Drucker ausgeben. Vorbei ist also die Zeit des Chaos in Platten-, Dia- oder anderen Sortimenten!

Hardware-Anforderungen: Amiga 500, 1000, 2000 mit mindestens 512 Kbyte Arbeitsspeicher.

1989, 188 Seiten, inkl. Programmdiskette,

ISBN 3-89090-791-1, Bestell-Nr. 90791

DM199,-*

(sFr 179,- / öS 1990,-)

AMIGA
**Superbase
2**

Superbase 2 Amiga (deutsch)

Wegen seiner Verkaufszahlen der Renner unter den Datenbanken, für den Hausgebrauch ebenso geeignet wie für das Büro. Daten und Bilder lassen sich mit diesem mächtigen, relationalen Datenbanksystem fast spielerisch verwalten. Eine neue Form der Lagerhaltung ist somit beispielsweise möglich: Nicht mehr nur die Daten, sondern auch ein digitalisiertes Bild jedes Artikels erscheinen auf dem Monitor. Bis zu 16 Millionen Datensätze pro Datei und eine unbegrenzte Anzahl geöffneter Dateien sind erlaubt. Die Ein- oder Ausgabe erfolgt in Listen oder Formularen, die Sie sich am Bildschirm erstellen können. Weitere Leistungsmerkmale: Textverarbeitungsprogramm integriert, Serienbrieffunktion, speicherbare Masken, Etikettendruck u.v.m.

Hardware-Anforderungen: Amiga 500, 1000, 2000 mit mind. 512 Kbyte RAM (empfohlen 1 Mbyte).

Bestell-Nr. 54110

Upgrade von Superbase 2 auf Superbase Professional, 51672U, DM 199,-* (sFr 179,- / öS 1990,-)

DM399,-*

(sFr 359,- / öS 3690,-)

AMIGA
**Superbase
PROFESSIONAL**

Superbase Professional

Neben den Funktionen von Superbase 2 bietet Ihnen diese Professional-Version zwei erhebliche Vorteile: zum einen den Formular-Editor, mit dem Sie sich grafische Ein- und Ausgabemasken in grandioser Qualität schaffen können, und zum anderen beinhaltet sie eine Programmiermöglichkeit. Die Programmiersprache DML ist zwar einfach erlernbar wie Basic, unterstützt die Möglichkeiten von Superbase jedoch vollständig. Somit sind der Flexibilität der Datenbank keine Grenzen gesetzt, sei es bei der individuellen Bedienung, sei es bei der Möglichkeit, Daten zu selektieren, darzustellen und auf dem Drucker auszugeben.

Hardware-Anforderungen: Amiga 500, 1000, 2000 mit mindestens 512 Kbyte RAM

(empfohlen: 1 Mbyte RAM, Festplatte), Bestell-Nr. 51672

Upgrade Superbase Professional auf Superbase Professional Entwickler-Paket, Bestell-Nr. 54136U, gegen Einsendung der Originaldiskette DM 199,-* (sFr 179,- / öS 1990,-)

DM599,-*

(sFr 539,- / öS 5690,-)

AMIGA
**Superbase
PROFESSIONAL
ENTWICKLER-PAKET**

Superbase Professional Entwickler-Paket

Superbase Professional Entwickler-Paket ist die konsequente Weiterentwicklung der herausragenden Datenbank Superbase Professional. Im Anwenderteil wurden die Tastaturbedienung, die Einbaumöglichkeit von Hilfefenstern, die Import-/Export-Funktion, der Formulareditor u.v.m. erheblich erweitert. dBase, Lotus u.a. Dateien können nun komfortabel eingelesen werden. Ein leistungsstarkes DFÜ-Programm wurde integriert, um die Datenfernübertragung zu erleichtern. Für die Programmierer stehen nun weitere Befehle zur Verfügung, die zum Beispiel Transaktionen von Daten möglich machen. Das mitgelieferte Runtime-Modul ermöglicht es Entwicklern, ihre Anwendungen auch denen zugänglich zu machen, die das Superbase-Professional-Entwickler-Paket nicht besitzen.

Hardware-Anforderungen: Amiga 500, 1000, 2000 mit mindestens 512 Kbyte RAM (empfohlen: 1 Mbyte RAM, Festplatte). Lieferbar 2. Quartal 1989, Bestell-Nr. 54136

*Unverbindliche Preisempfehlung

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 37, 6300 Zug, Telefon (042) 440550.

ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 587 1393-0.


Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Hindernissen beliebige Einstellungen vornehmen und den Schuß abgeben. Dann bauen wir, wie in Replay, die letzte Spielszene wieder auf. Die noch fehlenden Gadgets für Neustart und Programmende sind ebenfalls sehr schnell berücksichtigt, so daß sich auch die Installation unseres Bedienungsfeldes dem Ende nähert. Bei jeder sichtbaren Ergänzung des Reflex-Bildschirms tritt ein negativer Effekt immer deutlicher hervor: Der Aufbau des Screens nimmt immer mehr Zeit in Anspruch. Man kann das Erscheinen der einzelnen Bildschirmelemente mit bloßem Auge beobachten, was sehr unprofessionell wirkt. Was tun? Wir können doch kein Element weglassen! Man könnte z.B. den Bildschirmaufbau mit einer Titelgrafik verdecken – aber es geht noch einfacher: Wir schalten, für die Zeit des Aufbaus, den Bildschirm mit OFF_DISPLAY ab.

Damit ist wieder ein großes Reflex-Kapitel erledigt. Nach unseren Notizen bleibt nur noch der Einbau einer High-Score-Liste übrig. Sie soll auf einem eigenen Window untergebracht und jederzeit einsehbar sein. Ein eigenes Window ist schnell geöffnet, wir platzieren es hinter dem Bedienungsfeld. WriteScore() (Listing 25) listet darauf alle Siegernamen in unserer POut()-Schrift (Listing 3). Durch eine bestimmte Aktivität des Spielers soll das Score-Window nach vorne klappen. Nur – durch welche? Hier hilft uns nur ein wenig Fantasie weiter. Wie stellt man z.B. fest, wie viele Bälle ein Spieler im Ziel »versenkt« hat? Klar, man schaut im Zielloch nach. Na also. Das ist doch eine Lösung: Ein B(K)lick in eines unserer Ziele soll die Liste sichtbar machen. Jetzt müssen wir nur noch dafür sorgen, daß sich

ein Spieler bei einer besonders hohen Punktzahl in die Liste eintragen kann. Die Funktion Score() (Listing 26) sortiert die erreichte Punktzahl in die Liste ein. An der entsprechenden Position erscheint ein String-Gadget und erwartet die Eingabe des glücklichen Siegers. Eine richtige Bestenliste muß natürlich auf Diskette verewigt werden.

SaveScore() (Listing 27) übernimmt diese Aufgabe – und zwar so, daß die Liste nicht einfach mit einem Editor geändert werden kann. Beim Programmstart soll die Liste geladen werden – in InitData() paßt der Aufruf der LoadScore()-Funktion (Listing 28) am besten. Wir richten sie so ein, daß sie, falls sie eine Änderung bemerkt, alle Punktezahlen auf Null setzt.

Es ist kaum zu glauben: Alle Forderungen sind erfüllt, und alle Notizen wurden berücksichtigt. Bei einem ausführlichen Test der »0.99 Version« werden Kleinigkeiten geändert und noch einige Ideen verwirklicht. Zum Schluß fehlt, neben einem Gadget für die Workbench, nur noch die Anleitung des neuen Spiels. Dabei sollte man beachten, daß sie sauber gegliedert ist. Der Anwender sollte auf Anhieb finden, was er sucht. Verwenden Sie im Text keine Fachbegriffe. Formulieren Sie einfach und klar. Unterstützen Sie den Text eventuell mit Zeichnungen oder Bildschirmfotos. Sie finden die Anleitung zu Reflex im nebenstehenden Kasten. pe

Autor Arno Gölzer ist ein Mann der ersten Stunde und schreibt schon einige Jahre für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Herrn Gölzer, Hans-Pinsel-Straße 2, 8013 Haar bei München).

Anleitung zu Reflex

1. Allgemeines
2. Starten des Programms
3. Der Bildschirm

4. Der Schuß
5. Wertung
6. Tips

1. Allgemeines

Reflex ist ein neuartiges Schieß-Spiel, bei dem es nicht darum geht, irgendwelche imaginären Gegner niederzumähen, sondern eine Kugel durch einen geschickten Schuß in einem »Loch« zu versenken, bevor sie Schaden anrichtet.

Dabei muß der Spieler die Bahn der Kugel vor dem Schuß genau abschätzen, denn er muß dafür sorgen, daß die Kugel auf ihrem Weg an möglichst vielen »Reflektorschildern« abprallt. Reflex kann man alleine oder zu zweit spielen.

2. Starten des Programms

Starten Sie Reflex aus dem CLI, indem Sie den Programmnamen, gefolgt von <RETURN>, eingeben oder einfach per Mausklick von der Workbench. Nach wenigen Sekunden erscheint der Bildschirm von Reflex. Nach einem Mausklick auf dem Titel-Fenster kann das Match beginnen.

3. Der Bildschirm

Der Reflex-Bildschirm teilt sich in ein Spielfeld und ein Bedienungsfeld auf.

a) Spielfeld

Am oberen Spielfeldrand finden Sie die drei »Ziellöcher«. Ein »Blick« in ein solches Loch (Klick mit der linken Maustaste – Taste gedrückt lassen!) zeigt Ihnen anhand einer Bestenliste, wie viele Kugeln bisher versenkt wurden. Es schließen sich sechs Reihen mit je sieben roten Zylindern an. Auf jedem Zylinder ist ein Reflektorschild montiert. Sie sehen ihn aus Ihrer Perspektive als grünen Strich. Er läßt sich mit der linken Maustaste in einem Lager bewegen, welches sich im Mittelpunkt des Zylinders befindet. Drücken Sie hierzu auf einem Ende des Schildes die linke Maustaste – gedrückt lassen – und bewegen Sie den Pointer um den Mittelpunkt des roten Zylinders. Sie können fast jeden beliebigen Winkel einstellen, nur an einer Stelle (bei jedem Lager verschieden) ist das Lager beschädigt. Dort kann das Schild nicht positioniert werden. Die Barriere läßt sich, wenn sie einmal im Weg ist, »hochkant« stellen. Klicken Sie hierzu etwa in die Mitte des Zylinders oder nutzen Sie einen der vorgesehenen Funktionsschalter (siehe unter b). Aber Vorsicht: Auch an dem senkrechten Schild, der vom Spieler nun wie ein grüner Punkt aussieht, kann die Kugel reflektieren!

Wenn es Ihnen zu mühsam erscheint, die Zylindermitte zu treffen, hier ein Tip: Bewegen Sie den Pointer mit gedrückter linker Maustaste am grünen Schild entlang zur Mitte hin – oder nutzen Sie einen der hierfür vorgesehenen Funktionsschalter (siehe unter (b)). Am unteren Spielfeldrand ist eine Stange angebracht, auf der eine Kanone längs der Stangenachse beweglich montiert ist. Positionieren Sie die Kanone mit der linken Maustaste: Taste drücken – gedrückt lassen – und Kanone seitlich verschieben.

b) Bedienungsfeld

In der oberen Feldhälfte befindet sich die Spielstandsanzeige für beide Spieler. Die Anzeige für Spieler zwei ist direkt nach dem Start von Reflex noch nicht aktiv, d.h., Sie befinden sich im Ein-Spieler-Modus. Ein Klick mit der linken Maustaste auf das untere, farblich unterlegte Anzeigefeld aktiviert den Zwei-Spieler-Modus. Ein roter Punkt zeigt an, welcher Spieler gerade am »Drücker« ist. Klicken Sie in die obere Anzeige, um in den Ein-Spieler-Modus zurückzukehren.

Ein zweiter Klick auf das jeweilige Display setzt die bisher erreichte Punktzahl auf Null. Weiter unten sind im Bedienungsfeld noch sechs Schalter angebracht. Die beiden oberen davon helfen Ihnen, die Reflektorschilder hochkant zu stellen. Ist der linke Schalter aktiviert, wird die senkrechte Reihe, die Sie nun mit der linken Maustaste anwählen, ausgerichtet. Der rechte Schalter funktioniert analog für waagerechte Reihen. Sind eine oder beide Schalter aktiv, färbt sich der Hintergrund des Spielfeldes als Warnung rot, so daß Sie eingestellte Reflektoren nicht ungewollt verändern. Schalten Sie, wenn Sie die gewünschten Schilder eingestellt haben, den Modus aus, indem Sie die Schalter ein zweites Mal betätigen, oder aber, indem Sie die rechte Maustaste betätigen. Der Hintergrund nimmt dann wieder die alte Farbe an.

Um ein Schild wieder in die horizontale Lage zu versetzen, klicken Sie einfach auf den betreffenden Zylinder. Ein Klick mit der linken Maustaste auf den Schalter mit dem Projektor wiederholt den letzten Schuß. Sie können ihn sich so mehrmals anschauen und, falls er gut gelungen war, Ihren Gegner durch unablässiges Vorzeigen an den Rand der Verzeufung bringen.

Rechts daneben wartet Ihr Trainer auf seinen Einsatz. Aktivieren Sie ihn, können Sie eine Spielsituation beliebig oft trainieren. Nach dem Ausschalten des Trainers wird der alte Bildschirm wieder aufgebaut. Mit der Wahl des Raketenymbols starten Sie ein neues Spiel.

Die Tür auf dem letzten Schalter führt ins CLI bzw. zur Workbench, je nachdem, von wo Sie Reflex gestartet haben. Falls Sie sich in die Bestenliste eintragen konnten, wird das Programm vor dem Programmende, die Liste speichern.

4. Der Schuß

Drücken Sie im Bereich der Kanone die rechte Maustaste, so lösen Sie einen Schuß aus. Der erste Zylinder explodiert sofort unter der Wucht des Aufpralls. Manchmal gelingt es der Kugel, ein Hindernis zu

überspringen, aber in der Regel wird die Kugel bei einer Kollision mit einem Reflektorschild in eine andere Bahn gelenkt. Dabei gilt: Einfallswinkel der Kugel ist gleich dem Ausfallswinkel. Haben Sie Schwierigkeiten, sich den Verlauf der Kugel vorzustellen, drücken Sie an der Stelle des Schildes, wo Sie die Kollision vermuten, die rechte Maustaste – gedrückt lassen – und bewegen Sie den Pointer in Richtung Kanone. Jetzt erkennen Sie deutlich das oben beschriebene Gesetz der Winkel. Sie erkennen, wie die Kugel am Hindernis abprallt und können den nächsten Kollisionspunkt abschätzen. Lassen Sie sich von dort aus, wie beschrieben, den weiteren Bahnverlauf zeigen usw. ... möglicherweise bis zum Ziel. Beachten Sie jedoch, daß diese Meßeinrichtung nur korrekt funktioniert, wenn Sie den Kollisionspunkt mit der linken Maustaste so nah wie möglich an den grünen Schild legen. Die Kugel wird während des Fluges immer langsamer. Berührt sie, wegen der auf sie wirkenden Gravitation, den Spielfeldboden, explodiert sie und zerstört dabei die drei nächstgelegenen Zylinder mit ihren Schildern. Ihr Ziel ist es also, eines der drei Löcher zu treffen, so daß die Kugel »unterirdisch« detoniert und keinen Schaden anrichtet.

5. Wertung

Wichtig für die Wertung ist die Anzahl der Kollisionen während des Fluges der Kugel. Diese Anzahl wird beim »Versenken« der Kugel in ein äußeres Loch mit fünf und, falls das mittlere Loch getroffen wurde, mit zehn multipliziert.

Trifft man ein Loch, ohne daß die Kugel zuvor mit einem Schild zusammenprallte, erhält man demnach zwar keine Punkte, hat aber immerhin noch den Vorteil, daß keine Zylinder zerstört werden. Spielende ist, wenn alle Zylinder zerstört sind.

Im Zwei-Spieler-Modus darf ein Spieler solange schießen, wie er Punkte erzielt. Erst bei einer »Niete« erfolgt ein Spielerwechsel. Ein roter Punkt in der Spielstandanzeige gibt an, wer gerade am Schuß ist. Haben Sie besonders viele Punkte erreicht, können Sie sich mit fünf Buchstaben (oder Zeichen) in eine Bestenliste eintragen. Schauen Sie sich die bestehende Liste an, indem Sie die linke Maustaste betätigen – gedrückt lassen – während sich der Pointer über einem der drei »Löcher« befindet.

6. Tips

Reflex erscheint, zumindest am Anfang, als ein relativ schwieriges Spiel. Deshalb finden Sie hier ein paar Tips:

- Möglichst bald einen breiten Durchgang nach oben schaffen
- Versuchen Sie sich am Anfang mit nur wenigen Kollisionen. Schalten Sie die anderen Schilder aus (hochkant stellen).
- Nutzen Sie es aus, daß der erste Zylinder zerstört wird: Befindet sich z. B. vor dem mittleren Loch nur noch ein Zylinder, stellen Sie das Schild wenn möglich waagrecht und lösen Sie einen Schuß in Richtung Loch aus. Die Kugel reflektiert am Schild, dieser explodiert, und die Kugel bewegt sich nach unten. Am unteren Spielfeldrand reflektiert sie wiederum und fliegt den gleichen Weg zurück und, da nun kein Hindernis mehr stört, geradewegs ins Zielloch. Aber mehr soll jetzt nicht verraten werden – versuchen Sie sich selbst!!!

```
VOID OpenW()
/* Vorbereitungen wie Screen und Fenster öffnen usw */
if(!(IntuitionBase=(struct IntuitionBase *)
  OpenLibrary("intuition.library",0))) CloseW();
if(!(GfxBase=(struct GfxBase *)
  OpenLibrary("graphics.library",0))) CloseW();
if(!(MathTransBase=(struct MathTransBase *)
  OpenLibrary("mathtrans.library",0))) CloseW();
if(!(Screen=OpenScreen(&nsr))) CloseW();
OFF_DISPLAY;
ShowTitle(Screen,0);
VP = &Screen->Viewport;
SCRIP=&Screen->RastPort;
SetColor();
if(!(SWin=OpenWindow(&score))) CloseW();
SCBit=1<<SWin->UserPort->mp_SigBit;
SRP=SWin->RPort;
if(!(Cyly=OpenWindow(&cwin))) CloseW();
CBit=1<<Cyly->UserPort->mp_SigBit;
CRP=Cyly->RPort;
```

Listing 1. OpenW() übernimmt das Öffnen aller Libraries und Screens für Sie

```
if(!(Bedi=OpenWindow(&bwin))) CloseW();
SBit=1<<Bedi->UserPort->mp_SigBit;
BRP=Bedi->RPort;
if(!(Play=OpenWindow(&pwin))) CloseW();
PBit=1<<Play->UserPort->mp_SigBit;
PRP=Play->RPort;
if(!(Intro=OpenWindow(&iwin))) CloseW();
IBit=1<<Intro->UserPort->mp_SigBit;
IRP=Intro->RPort;
if(!(Process=CreateFunctionProc("TFunc",0,TFunc,8000)))
  CloseW();
Cyly->UserPort->mp_SigTask=FindTask("TFunc");
if(!(Bitplane=AllocRaster(100,100))) CloseW();
InitTmpRas(&Raster,Bitplane,RASSIZE(100,100));
IRP->TmpRas=PRP->TmpRas=BRP->TmpRas=CRP->TmpRas=&Raster;
InitArea(&AInfo,Buffer,2);
IRP->AreaInfo=PRP->AreaInfo=BRP->AreaInfo=CRP->AreaInfo=
  &AInfo;
SNr[0]=GetSprite(&Kanone,3);
SNr[1]=GetSprite(&Kugel,4);
InitRequester(&DReq);
DReq.LeftEdge=DReq.TopEdge=DReq.Width=DReq.
Height=DReq.BackFill=2;
SoundOn();
InitData();
}
```

Listing 1. (Schluß)

```
VOID CloseW()
/* zum Programmende alle Quellen schließen */
SHORT i;

if(SData) FreeMem(SData,1024);
if(!SDev) CloseDevice(&Sound);
if(SPort) DeletePort(SPort,sizeof(struct MsgPort));
for(i=0;i<2;i++) FreeSprite(SNr[i]);
if(Bitplane) FreeRaster(Bitplane,100,100);
if(Process) CloseProcess();
if(Intro) CloseWindow(Intro);
if(Cyly) CloseWindow(Cyly);
if(SWin) CloseWindow(SWin);
if(Play) CloseWindow(Play);
if(Bedi) CloseWindow(Bedi);
if(Screen) CloseScreen(Screen);
if(MathTransBase) CloseLibrary(MathTransBase);
if(GfxBase) CloseLibrary(GfxBase);
if(IntuitionBase) CloseLibrary(IntuitionBase);
ON_DISPLAY;
exit(0);
}
```

Listing 2. CloseW() erledigt die Aufräumarbeiten bei Beendigung des Programms

```
VOID POut(RP,col,text,xpos,ypos)
struct RastPort *RP;
STRPTR text;
SHORT col,xpos,ypos;
/* Textausgabe Outlay */
Print(RP,text,3,xpos-1,ypos-1);
Print(RP,text,4,xpos+1,ypos+1);
Print(RP,text,col,xpos,ypos);
}

VOID PIn(RP,text,xpos,ypos)
struct RastPort *RP;
STRPTR text;
SHORT xpos,ypos;
/* Textausgabe Inlay */
SetSoftStyle(RP,FSF_ITALIC|FSF_BOLD,AskSoftStyle(RP));
Print(RP,text,4,xpos-1,ypos-1);
Print(RP,text,3,xpos+1,ypos+1);
Print(RP,text,1,xpos,ypos);
Line(RP,3,xpos+2,ypos+10,xpos+8*strlen(text)+2,ypos+10);
Line(RP,4,xpos-2,ypos+12,xpos+8*strlen(text)-2,ypos+12);
```



```
SetSoftStyle(RP,FS_NORMAL,AskSoftStyle(RP));
```

Listing 3. POut() und Pin() geben den Gadgets das Aussehen von realen Bedientasten

```
VOID MoveK(x)
SHORT x;
/* Kanone nach Mauspointer bewegen */
REGISTER SHORT i;

if(x< 9) x= 9;
if(x>233) x=233;
if(x==LKPos) return;
SetDrMd(PRp,JAM1);
do{
    for(i=0;i<3;i++) Line(PRp,SCol[i],1,249+1,239,249+1);
    Block(PRp,1,x-4,248,x+3,252);
    MoveSprite(VP,&Kanone,x-10,235);
    while(((PMouseX<9)|| (PMouseX>233))&&(x==PMouseX))&&LEFTDOWN);
    if(PMouseY>232) x=PMouseX;
    if(x< 9) x= 9;
    if(x>233) x=233;
}while(LEFTDOWN);
LKPos=x;
```

Listing 4. MoveK() bringt das Geschütz in Stellung

```
VOID DrawHole(x,f)
SHORT x,f;
/* Löcher zeichnen: f=-1: Flush || f= 1: Hole */
REGISTER SHORT i;

SetDrMd(PRp,JAM1);
for(i=0;i<4;i++){
    SetAPen(PRp,12+(i*f));
    AreaEllipse(PRp,x-1,12+1,10-1,9-1);
    AreaEnd(PRp);
}
```

Listing 5. DrawHole() zeichnet die Ziellöcher auf den Screen

```
VOID DrawPlay(clear)
BOOL clear;
/* Spielfeld neuzeichnen */
REGISTER SHORT i;

if(clear) Clear();
else{
    Block(PRp,1,1,24,239,230);
    MoveK(RKPos?RKPos:122);
}
for(i=0;i<BNr;i++) DrawCyl(i,1);
SetP();
```

Listing 6. Die Reflektoren setzt DrawPlay() ins Bild

```
VOID InitCyl()
/* Ausgangsstellung Zylinder mit Barrieren */
REGISTER SHORT i,j,k,nr;
FLOAT x,y,w;

NewRandom();
for(i=0;i<Braw;i++){
    for(j=0;j<Bcol;j++){
        nr=i*Bcol+j;
        for(k=0;k<4;k++) Bar[nr].bar[k]=DBar[k];
        Bar[nr].BMx=18+j*34;
```

```
Bar[nr].BMy=40+i*34;
Bar[nr].VW=99;
Bar[nr].Off=0;
TurnBar(0,nr,Bar[nr].BMx+randx[nr],Bar[nr].BMy+randy[nr]);
x=(FLOAT)(Bar[nr].bar[2]-Bar[nr].bar[0]);
y=(FLOAT)(Bar[nr].bar[3]-Bar[nr].bar[1]);
w=RAD*atan(y/(x+NO5));
Bar[nr].BW=(w<0)?180+w:w;
Bar[nr].VW=rand()/8192; /* 4 Winkel */
}
EnableReplay=0;
```

Listing 7. Für lange Spielfreude sorgt InitCyl()

```
VOID MakeCyl()
/* Vorzeichen eines roten und eines gelben
Zylinders im Hilfswindow */
REGISTER SHORT i; /* i==0 Zylinder rot || i==1 Zylinder gelb */

SetRast(CRp,1);
SetDrMd(CRp,JAM1);
for(i=0;i<2;i++){
    SetAPen(CRp,5+i*8);
    DrawCircle(CRp,15+i*35,15,15);
    SetAPen(CRp,6+i*8);
    AreaEllipse(CRp,17+i*35,17,15,15);
    AreaEnd(CRp);
    SetAPen(CRp,7+i*8);
    AreaEllipse(CRp,16+i*35,16,15,15);
    AreaEnd(CRp);
}
```

Listing 8. MakeCyl() arbeitet im Hintergrund, um einen reibungslosen Ablauf zu garantieren

```
VOID DrawCyl(nr,bar)
SHORT nr,bar;
/* roter Zylinder ins Spielfeld clippen */
if(Bar[nr].Off) return;
ClipBlit(CRp,0,0,PRp,Bar[nr].BMx-16,Bar[nr].BMy-16,34,34,0xC0);
if(bar)
    DrawBar(nr, Bar[nr].bar[0],Bar[nr].bar[1],
            Bar[nr].bar[2],Bar[nr].bar[3]);
```

Listing 9. Die mit MakeCyl() berechneten Zylinder bringt DrawCyl() ins Bild

```
VOID DrawBar(nr,ax,ay,ex,ey)
SHORT nr;
/* Barriere zeichnen */
SetDrMd(PRp,JAM1|COMPLEMENT);
Move(PRp,ax+Bar[nr].BMx,ay+Bar[nr].BMy);
Draw(PRp,ex+Bar[nr].BMx,ey+Bar[nr].BMy);
```

Listing 10. Das Zeichnen der Barrieren ist die Aufgabe von DrawBar()

```
VOID CheckMouse(code)
USHORT code;
/* Reaktion auf Mausklick */
REGISTER SHORT i,mx,my,nr,klick=0,col;

mx=PMouseX;
my=PMouseY;
if(code==SELECTDOWN){
    col=ReadPixel(PRp,mx,my);
    if((my<40)&&(col>8)&&(col<13)) Score(0);
```



```

else{
if(VERI_ON){
nr=GetCylNr(mx,-1);
for(i=0;i<Braw;i++,nr+=Bool)
if(!Bar[nr].Off)
TurnBar(1,nr,Bar[nr].BMx,Bar[nr].BMy);
}
if(HORI_ON){
nr=GetCylNr(-1,my);
for(i=nr;i<nr+Bcol;i++)
if(!Bar[i].Off)
TurnBar(1,i,Bar[i].BMx,Bar[i].BMy);
}
if((!VERI_ON)&&(!HORI_ON)){
SetRGB4(VP,0,15,11,7);
if(my>232) MoveK(mx);
else CheckCyl(mx,my);
}
}
}
else if(code==MENUDOWN){
for(i=2;i<4;i++){
if(Gad[i].Flags&SELECTED){
Klick=1;
Gad[i].Flags=GADGHIMAGE;
RefreshGList(&Gad[i],Bedi,NULL,1);
}
}
if(!Klick){
if(my>232) Schuss();
else Help(mx,my);
}
else if((!VERI_ON)&&(!HORI_ON)) SetRGB4(VP,0,15,11,7);
}
}

```

Listing 11. Die Maus im Griff mit CheckMouse()

```

VOID TurnBar(draw,nr,x,y)
SHORT draw,nr,x,y;
/* Winkel einer Barrier-Linie berechnen und
Barriere darstellen */
REGISTER SHORT i,ntrn=0;
SHORT ax=Bar[nr].bar[0],ay=Bar[nr].bar[1],
ex=Bar[nr].bar[2],ey=Bar[nr].bar[3];
FLOAT xp,yp,diffx,diffy,hyp,sinus=0.0,cosinus=0.0;

diffx=(FLOAT)(x-Bar[nr].BMx);
diffy=(FLOAT)(Bar[nr].BMy-y);
hyp=sqrt((diffx*diffx)+(diffy*diffy))+NO5;
if(hyp>30) return;
if(hyp<5) diffx=diffy=0;
if(fabs(diffx)+fabs(diffy)){
switch(Bar[nr].VW){ /* welcher Winkel ist verboten ? */
case 0:{ /* ! */
ntrn=((SHORT)(diffx>-2)&&(SHORT)(diffx<2));
break;
}
case 1:{ /* - */
ntrn=((SHORT)(diffy>-2)&&(SHORT)(diffy<2));
break;
}
case 2:{ /* \ */
i=(SHORT)diffx+(SHORT)diffy;
if((i>-2)&&(i<2)) ntrn=1;
break;
}
case 3:{ /* / */
i=(SHORT)diffx-(SHORT)diffy;
if((i>-2)&&(i<2)) ntrn=1;
break;
}
default:break;
}
if(ntrn) return; /* verbotener Winkel ! */
sinus=diffy/hyp;
cosinus=diffx/hyp;
for(i=0;i<4;i+=2){

```

```

xp=(FLOAT)DBar[i];
yp=(FLOAT)DBar[i+1];
Bar[nr].bar[i] =(SHORT)( xp*cosinus+yp*sinus);
Bar[nr].bar[i+1]=(SHORT)(-xp*sinus+yp*cosinus);
}
if(draw&&((ax!=Bar[nr].bar[0])|| (ay!=Bar[nr].bar[1])||
(ex!=Bar[nr].bar[2])|| (ey!=Bar[nr].bar[3])))
DrawBar(nr,ax,ay,ex,ey); /* löschen, dann zeichnen: */
DrawBar(nr,Bar[nr].bar[0],Bar[nr].bar[1],
Bar[nr].bar[2],Bar[nr].bar[3]);
}
}

```

Listing 12. TurnBar() hat den Dreh raus

```

SHORT CheckH()
/* ist die Kugel versenkt? */
SHORT n,p,i,x,col,stop=0;
FLOAT sx,sy;

col=ReadPixel(PRp,(SHORT)xpos,(SHORT)ypos);
if((col>8)&&(col<13)){
if((xpos>=50.0)&&(xpos<=70.0)) stop=x=60;
else if((xpos>=110.0)&&(xpos<=130.0)) stop=x=120;
else if((xpos>=170.0)&&(xpos<=190.0)) stop=x=180;
}
if(stop){
p=((x==120)?10:5);
while(((SHORT)xpos!=x)||((SHORT)ypos!=16)){
sx=(xpos<(FLOAT)x)?1.0:-1.0;
sy=(ypos<16.0)?1.0:-1.0;
if((SHORT)xpos!=x) xpos+=(xv=sx*(.005+fabs(xv)));
if((SHORT)ypos!=16) ypos+=(yv=sy*(.005+fabs(yv)));
MoveSprite(VP,&Kugel,(SHORT)xpos-14,(SHORT)ypos-2);
}
Kugel_OFF;
if(!ReplayOn&&!TRAINER_ON){
LPkte=p*Koll;
Pkte[Player]+=LPkte;
}
Delay(10);
for(i=0;i<5;i++){
n=rand()/1000; /* Manx RandomNumber */
DrawHole(x,1);
PlaySound(&SDData[512],10239,32);
Delay(n);
DrawHole(x,-1);
}
}
return(stop);
}

```

Listing 13. Über Erfolg oder Mißerfolg entscheidet CheckH()

```

VOID CheckB()
/* Kugel-Kollision an Barriere oder Feldrand checken */
SHORT col,nr,kf=0;
FLOAT h,aw,nw;

if((col=ReadPixel(PRp,(SHORT)xpos,(SHORT)ypos))==8){
if((nr=GetCylNr((SHORT)xpos,(SHORT)ypos))>=0){
if(!Koll) PutProcessMsg(FIRSTCOLLISION,nr);
else if(KFlag<1) PlaySound(&SDData[256],3839,16);
Koll++;
KFlag++;
xpos-=xv;
ypos-=yv;
if(KFlag>5){
KFlag=0;
xpos+=xv<0?1.0:-1.0;
ypos+=yv<0?1.0:-1.0;
}
aw=RAD*atan(yv/(xv+NO5));
nw=2*Bar[nr].BW-(xv<.0?180+aw:aw);
h=sqrt(yv*yv+xv*xv);
yv=(h*sin(nw/RAD));

```



```

        xv=(h*cos(nw/RAD));
        xpos+=(2*xv);
        ypos+=(2*yv);
    }
    else{ /* Feldrand */
        if((xpos>240)|| (xpos<1)){
            kf=1;
            if(KFlag<1) PlaySound(&SData[256],1839,16);
            xv=-xv;
        }
        if((ypos>230)|| (ypos<1)){
            kf=1;
            if(KFlag<1) PlaySound(&SData[256],1839,16);
            yv=-yv;
        }
        KFlag=kf;
    }
}

```

Listing 14. Bei Kollisionen ist CheckB() gefragt

```

VOID KillCyl(nr)
    SHORT nr;
/* Zylinder löschen */
Bar[nr].Off=1;
Block(PRp,1,Bar[nr].BMx-16,Bar[nr].BMy-16,
Bar[nr].BMx+16,Bar[nr].BMy+16);

```

Listing 15. Mit KillCyl() werden Zylinder gelöscht

```

VOID InitData()
/* Initialisierung und Vorbereitung verschiedener Daten */
REGISTER SHORT i,j,n,x,y,inc;

srand(VBeamPos()); /* Start für Manx RandomNumber */
if(!(SData=AllocMem(1024,MEMF_CHIP))) CloseW();
for(i=0;i<1024;i++) SData[i]=SD[i]; /* Sound-Daten auf LONG */
for(i=START,n=0;n<KREISE;i+=INCREMENT,n++)
{ /* Explosion berechnen */
    inc=1/PUNKTE;
    for(x=j=0;j<PUNKTE;x+=inc,j++){
        y=(SHORT)sqrt((FLOAT)(i*i*x*x));
        Exp[n][j][0]= x;
        Exp[n][j][1]= y;
        Exp[n][j][2]= y;
        Exp[n][j][3]=-x;
        Exp[n][j][4]=-x;
        Exp[n][j][5]=-y;
        Exp[n][j][6]=-y;
        Exp[n][j][7]= x;
    }
}
LoadScore();

```

Listing 16. InitDat sorgt für Tempo durch Vorausberechnung

```

VOID TFunc()
/* SubProcess: zuständig für Explosion und Geräusche */
struct IntuiMessage *mes;
LONG z;
ULONG class;
USHORT code;
REGISTER SHORT i,j,k,l,x,y,n;
BOOL ende=0;

geta4();
while(!ende){
    Wait(CBit);
    while(class=GetMessage(Cyly,&code,0)){
        switch(class){

```

```

        case FIRSTCOLLISION:{
            ExCyl(code);
            Delay(4);
            PlaySound(&SData[768],12979,32);
            Bar[code].Off=1;
            KillCyl(code);
            for(i=0;i<KREISE;i++){
                for(j=0;j<2;j++){
                    if(j) Delay(4);
                    for(k=0;k<PUNKTE;k++){
                        for(l=0;l<KOOXY;l+=2){
                            SetDrMd(PRp,JAM11 COMPLEMENT);
                            x=Bar[code].BMx+Exp[i][k][l];
                            y=Bar[code].BMy+Exp[i][k][l+1];
                            Move(PRp,x-1,y);
                            Draw(PRp,x+1,y);
                            Move(PRp,x,y-1);
                            Draw(PRp,x,y+1);
                        }
                    }
                }
            }
            break;
        }
        case NOPOINT:{
            PlaySound(&SData[0],12031,64);
            break;
        }
        case SOUND_ON:{
            n=-1;
            do{
                n=((n+1)>127)?0:n+1;
                PlaySound(&SData[256],839+Drums[n]*1000,32);
                Delay(6);
                class=GetMessage(Cyly,0,0);
            }while(class!=SOUND_OFF);
            break;
        }
        case CLOSETASK:{
            ende=1;
            break;
        }
        default: break;
    }
}

```

Listing 17. Nutzen Sie die Fähigkeiten des Multitasking mit TFunk()

```

SHORT GetCylNr(mx,my)
    SHORT mx,my;
/* Nummer des nächsten (bezüglich der mx/my Position)
Zylinders ermitteln */
REGISTER SHORT i;
SHORT nr=-1,w=0,min=1000;

for(i=0;i<BNr;i++){
    if((my<0)|| (mx<0)|| (Bar[i].Off==0)){
        w=ABS(Bar[i].BMx-mx)+ABS(Bar[i].BMy-my);
        if(w<=min){
            min=w;
            nr=i;
        }
    }
}
return(nr);

```

Listing 18. GetCylNr() lokalisiert die von der Explosion betroffenen Zylinder aus

```

VOID SetP()
/* Punktzahlen darstellen */
REGISTER SHORT i;

```


Deluxe Paint III

Als die Bilder laufen lernten



DELUXE PAINT

DPaint II war bisher das leistungsfähigste Malprogramm auf dem Amiga. Diese Zeiten sind glücklicherweise vorbei, denn nun gibt es Deluxe Paint III! Ob Sie Zeichentrick-Pionier im eigenen Heimkino sind oder fernsehgerechte Vorspanne für Ihre Videofilme erzeugen wollen, ob Sie Grafik beruflich benötigen

oder als Hobbymaler auf den Spuren der großen Meister wandeln: Mit Deluxe Paint III eröffnen sich Ihnen Möglichkeiten, die Sie bisher für unmöglich gehalten haben. Was das Programm so alles kann?

- Alle Funktionen von Deluxe Paint II
- Extra Halfbrite-Modus für 64 Farben
- Beliebige Definition und Handhabung von Pinseln
- Spezielle Maleffekte wie Schattieren, Verschmieren, Weichzeichnen, Farbverlauf, Sprühdose
- Neue Füllfunktionen
- Exaktes Zeichnen geometrischer Formen
- Schnelle Perspektive-Funktion für 3-D-Grafiken
- Alle Amiga-Zeichensätze sowie

- spezielle Color-Fonts nutzbar
- Overscan-Unterstützung
- Filme aus Einzelbildern erstellen
- Pinselanimation: Jeder Teil eines Filmes kann als Pinsel ausgeschnitten werden, und ist selbst wieder ein kleiner Film
- Bewegen-Option: Jeder Pinsel kann in jede beliebige Richtung bewegt und dreidimensional gedreht werden, ja sogar bei der Bewegung »Spuren« hinterlassen. DPaint III macht daraus automatisch einen Film.
- 1 Mbyte RAM reicht für kreative Animationen
- ANIM-Speicherformat – Kompatibilität zu anderen Animationsprogrammen

Markt & Technik
Zeitschriften · Bücher
Software · Schulung



Hardware-Anforderungen:
Amiga mit mindestens 1 Mbyte RAM.

Deluxe Paint III deutsch

Bestell-Nr.: 54138

DM 249,--*

(sFr 225,--*/öS 2490,--*)

**Update von Deluxe Paint II
auf Deluxe Paint III**

(gegen Einsendung der Original-
diskette und Verrechnungsscheck)

Bestell-Nr. 54138U

DM 99,--* (sFr 89,--*/öS 990,--*)

Für alle Amiga-Einsteiger:

Deluxe Paint II deutsch

Bestell-Nr.: 54140

jetzt **DM 149,--***

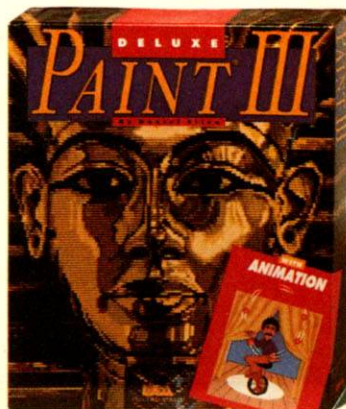
(sFr 135,--*/öS 1490,--*)

Deluxe Paint III Demo

Demo-Versionen erhalten Sie gegen
Vorauskauf direkt beim Verlag.

Bestell-Nr. W718

DM 15,--



**Markt&Technik-Produkte
erhalten Sie bei Ihrem Buch-
oder Computerfachhändler**

INFO-COUPON

Bitte senden Sie mir weitere Informationen
zu Deluxe Paint III

Name

Straße

PLZ/Ort

Bitte ausschneiden und senden an: Markt&Technik Verlag AG,
Buchverlag, Frau Brosien, Hans-Pinsel-Str. 2, 8013 Haar

Ami SH 7

*Unverbindliche Preisempfehlung


```

TEXT p[2][10];

SetOpen(BRp,4)
for(i=0;i<2;i++){
    Block(BRp,1+(1-PlayerNr)*i*13,4,73+i*15,71,88+i*15);
    sprintf(p[i],"%d",Pkte[i]);
    POut(BRp,14,p[i],7,78+i*15);
}
SetAPen(BRp,7);
AreaEllipse(BRp,63,81+Player*15,2,2);
AreaEnd(BRp);
BNDRYOFF(BRp);
}

```

Listing 19. Die Ausgabe der Punkte erfolgt über SetP()

```

VOID Help(mx,my)
SHORT mx,my;
{ /* Bestimmung des Ausfallwinkels per Maus */
SHORT nr,fx=mx,fy=my;
FLOAT h,aw,nw,xv,yv;

if((nr=GetCylNr(fx,fy))<0) return;
do{
    xv=(FLOAT)(fx-mx);
    yv=(FLOAT)(fy-my);
    aw=RAD*atan(yv/(xv+NO5));
    nw=2*Bar[nr].BW-(xv<.0?180+aw:aw);
    h=sqrt(yv*yv+xv*xv);
    yv=(h*sin(nw/RAD));
    xv=(h*cos(nw/RAD));
    DrawHLine(mx,my,fx,fy,(SHORT)xv+fx,(SHORT)yv+fy);
    while((mx==PMouseX)&&(my==PMouseY)&&RIGHTDOWN);
    DrawHLine(mx,my,fx,fy,(SHORT)xv+fx,(SHORT)yv+fy);
    mx=PMouseX;
    my=PMouseY;
} while(RIGHTDOWN);
}

```

Listing 20. Eine große Hilfe für die Flugbahnberechnung ist Help()

```

VOID PlaySound(Data,Period,Volume)
UBYTE *Data;
UWORD Period;
SHORT Volume;
{ /* Geräusch (Data) abspielen */
if(!CheckIO(&Sound)) AbortIO(&Sound);
Sound.ioa_Data=Data;
Sound.ioa_Period=Period;
Sound.ioa_Volume=Volume;
BeginIO(&Sound);
}

```

Listing 21. Für richtige Spielhallenatmosphäre sorgt PlaySound()

```

VOID Schuss()
{ /* ein Schuß aus der Kanone abfeuern */
REGISTER SHORT i;

SLEEP;
EnableReplay=1;
RKPos=LKPos;
for(i=0;i<BNr;i++) RBar[i]=Bar[i];
LPkte=Koll=0; /* Anzahl Punkte/Kollisionen auf 0 */
xv=0.0; /* Geschwindigkeitsvektoren */
yv=-.99;
xpos=(FLOAT)(Kanone.x+9); /* Startposition der Kugel X,Y */
ypos=230.0;
while(MKugel()==0); /* warten solange die Kugel läuft */
while(GetMessage(Play,0,0));
if(TRAINER_ON){

```

```

Delay(50);
for(i=0;i<BNr;i++) Bar[i]=TBar[i];
DrawPlay(0);
}
Kugel_OFF;
if(!LPkte&&!ReplayOn&&!TRAINER_ON){
    if(PlayerNr) Player=(Player==0)?1:0;
    else Player=0;
}
WAKE;
SetP();
if(GetCylNr((SHORT)xpos,(SHORT)ypos)<0) GameOver();
}

```

Listing 22. Schuss() bringt Action ins Spiel

```

VOID GameOver()
{ /* Spielende: Auswertung */
SetOpen(PRp,9);
Block(PRp,12,60, 50,180,100);
Block(PRp, 2, 2,105,238,230);
BNDRYOFF(PRp);
POut(PRp,7," Game Over ",68,70);
Score(1);
NewScore=0;
DrawPlay(1);
}

```

Listing 23. Das bittere Ende: GameOver()

```

BOOL CheckGadgets(id)
SHORT id;
{ /* Reaktion auf Gadgetwahl */
REGISTER SHORT i;
BOOL Quit=0;

switch(id){
    case 0: /* ein Spieler */
    case 1: /* zwei Spieler */
        if(PlayerNr==id) Pkte[id]=0;
        else Player=PlayerNr=id;
        SetP();
        break;

    case 2: /* Horizontale nullen */
    case 3: /* Vertikale nullen */
        if((!HORI_ON)&&!VERI_ON) SetRGB4(VP,0,15,11,7);
        else SetRGB4(VP,0,15,0,0);
        break;

    case 4: /* Replay */
        if(!EnableReplay) break;
        SLEEP;
        ReplayOn=1;
        for(i=0;i<BNr;i++) Bar[i]=RBar[i]; /* Replay-Bild */
        DrawPlay(0);
        WAKE;
        Delay(50);
        Schuss();
        ReplayOn=0;
        break;

    case 5: /* Trainer */
        if(TRAINER_ON) for(i=0;i<BNr;i++) TBar[i]=Bar[i];
        break;

    case 6: /* NewGame */
        ReplayOn=LKPos=NewScore=0;
        for(i=2;i<6;i++) Gad[2].Flags=GADGHIMAGE;
        RefreshGList(&Gad[2],Bedi,NULL,4);
        DrawPlay(1);
        break;

    case 7: /* Ende */
        Quit=1;
        break;
}
}

```



```

        default: break;
    }
    return(Quit);
}

```

Listing 24. Treffen Sie Ihre Wahl mit CheckGadgets()

```

VOID WriteScore()
/* Score neuschreiben */
REGISTER SHORT i;

SetRast(SRp,2);
Line(SRp,4,0, 25,75, 25);
Line(SRp,3,0,233,75,233);
POut(SRp,7, "HighScore",1,10);
for(i=0;i<SIEGER;i++){
    POut(SRp, 1,&Sieger[i][0], 1,32+i*10);
    POut(SRp,15,&Punkte[i][0],43,32+i*10);
}
POut(SRp,7, "* Klick *",1,40+i*10);
}

```

Listing 25. Die glorreichen 20 auf einen Blick mit WriteScore()

```

VOID Score(check)
    BOOL check;
/* Score darstellen und eventuell Name eintragen lassen */
LONG bit;
REGISTER SHORT i,st=-1,snr;
USHORT code,id;
BOOL player=0;
TEXT Kommentar[2][31];

PutProcessMsg(SOUND_ON,0);
ActivateWindow(SWin);
if(check){
    player=1;
    if(PlayerNr) snr=Pkte[1]>Pkte[0]?1:0;
    else snr=0;
    for(i=SIEGER-1;i>=0;
    i--) if(Pkte[snr]>=atoi(Punkte[i])) st=i;
}
if(st>=0){
    ScoreOn=NewScore=1;
    for(i=SIEGER-1;i>=st;i--){
        strcpy(&Sieger[i+1][0],&Sieger[i][0]);
        strcpy(&Punkte[i+1][0],&Punkte[i][0]);
    }
    sprintf(&Punkte[st][0],"%4d",Pkte[snr]);
    WriteScore();
    Sieger[st][0]='\0';
    SInfo.Buffer=&Sieger[st][0];
    SGad.TopEdge=32+st*10;
    RefreshGadgets(&SGad,SWin,NULL);
    WindowToFront(SWin);
    ActivateGadget(&SGad,SWin,NULL);
    if(PlayerNr)
        sprintf(Kommentar[0],"%d Punkte für Spieler %d!",
            Pkte[snr],snr+1);
    else
        sprintf(Kommentar[0],"Sie erzielten %d Punkte!",
            Pkte[snr]);
    sprintf(Kommentar[1],"Das bedeutet: Platz %d !",st+1);
    Pin(PRp, " Ein S U P E R - Spiel !!! ",10,130);
    for(i=0;i<2;i++) POut(PRp,4,Kommentar[i],15,170+i*15);
    do{
        bit=Wait(PBit|SCBit);
        if(bit&PBit) win=Play;
        else win=SWin;
        GetMessage(win,&code,&id);
        if(id==8){
            id=0;
            SGad.TopEdge=-10;
            WriteScore();
        }
    }while(code!=SELECTUP);
}

```

```

SaveScore();
}
else{
    if(player){
        if(PlayerNr)
            sprintf(Kommentar[0],"%d Punkte für Spieler %d!",
                Pkte[snr],snr+1);
        else
            sprintf(Kommentar[0],"Sie erzielten %d Punkte!",
                Pkte[snr]);
        Pin(PRp, " Ein wirklich gutes Spiel ! ",10,130);
        POut(PRp,4,Kommentar[0],15,170);
        POut(PRp,4, "Trainieren Sie weiter so!",15,185);
    }
    SGad.TopEdge=-10;
    WindowToFront(SWin);
    do{
        bit=Wait(PBit|SCBit);
        if(bit&PBit) win=Play;
        else win=SWin;
        GetMessage(win,&code,0);
    }while(code!=SELECTUP);
}
WindowToBack(SWin);
PutProcessMsg(SOUND_OFF,0);
WriteScore();
}

```

Listing 26. Sind Sie würdig für die High-Score-Liste? Score() gibt Ihnen die Antwort.

```

VOID SaveScore()
/* Bestenliste (wenn möglich) speichern */
REGISTER SHORT i;
FILE *file;

if(NewScore&&(file=fopen(SCORE,"w"))){
    for(i=0;i<SIEGER;i++){
        fwrite(&Sieger[i][0],sizeof(Sieger[i]),1,file);
        KPkte[i]=atoi(&Punkte[i][0]);
        fwrite(&Punkte[i][0],sizeof(Punkte[i]),1,file);
        fwrite(&KPkte[i],sizeof(SHORT),1,file);
    }
    fclose(file);
    ScoreOn=NewScore=0;
}

```

Listing 27. SaveScore() verewigt Ihre Taten auf Diskette

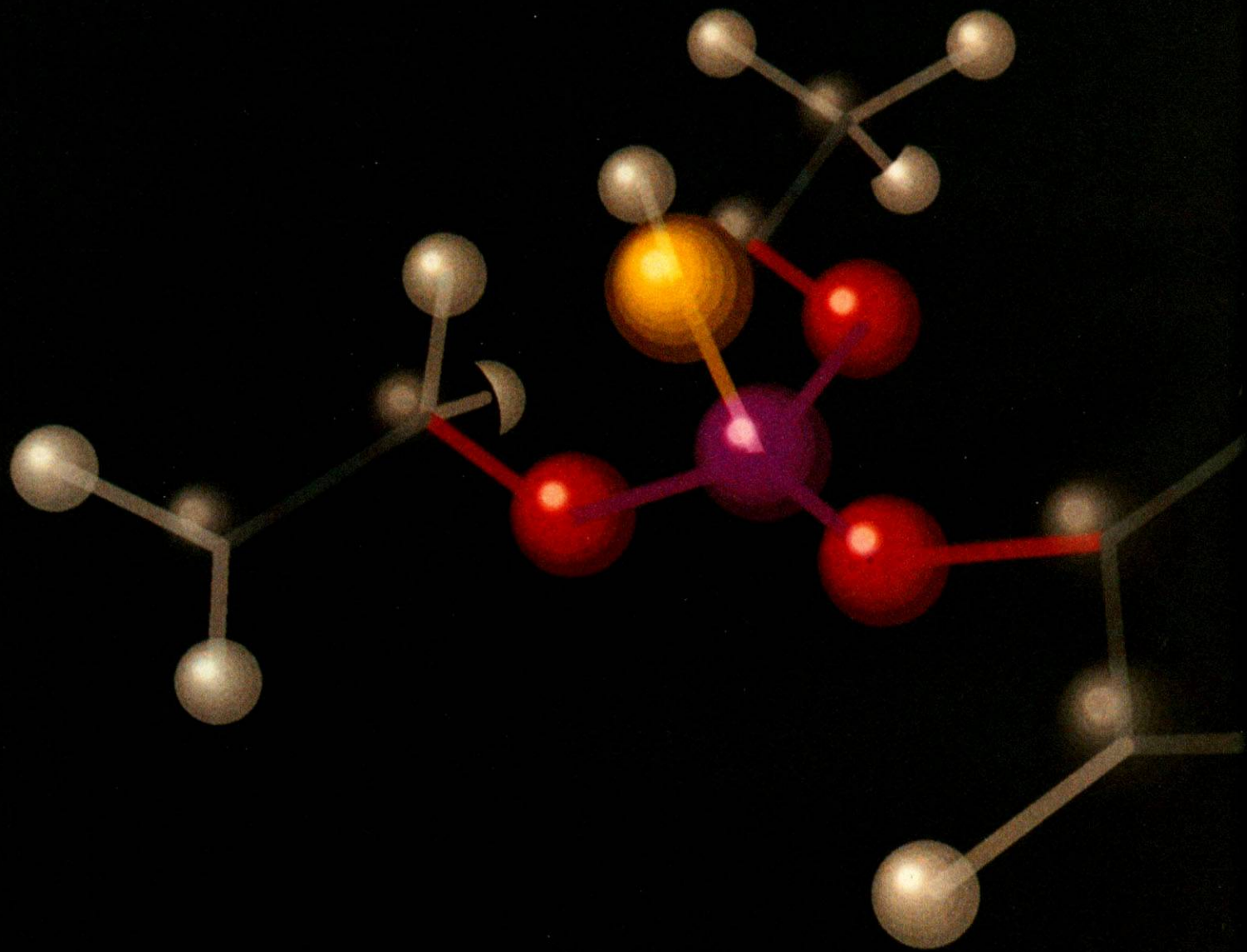
```

VOID LoadScore()
/* Bestenliste laden */
REGISTER SHORT i;
FILE *file;

if(file=fopen(SCORE,"r")){
    for(i=0;i<SIEGER;i++){
        fread(&Sieger[i][0],sizeof(Sieger[i]),1,file);
        fread(&Punkte[i][0],sizeof(Punkte[i]),1,file);
        fread(&KPkte[i],sizeof(SHORT),1,file);
        if(KPkte[i]!=atoi(&Punkte[i][0])){
            strcpy(&Punkte[i][0]," 0");
            NewScore=1;
        }
        Sieger[i][5]=Punkte[i][4]='\0';
    }
    fclose(file);
}
else for(i=0;i<SIEGER;i++){
    strcpy(&Sieger[i][0],"- ");
    strcpy(&Punkte[i][0]," 0");
}
}

```

Listing 28. Erinnerung aus vergangenen Tagen mit LoadScore()

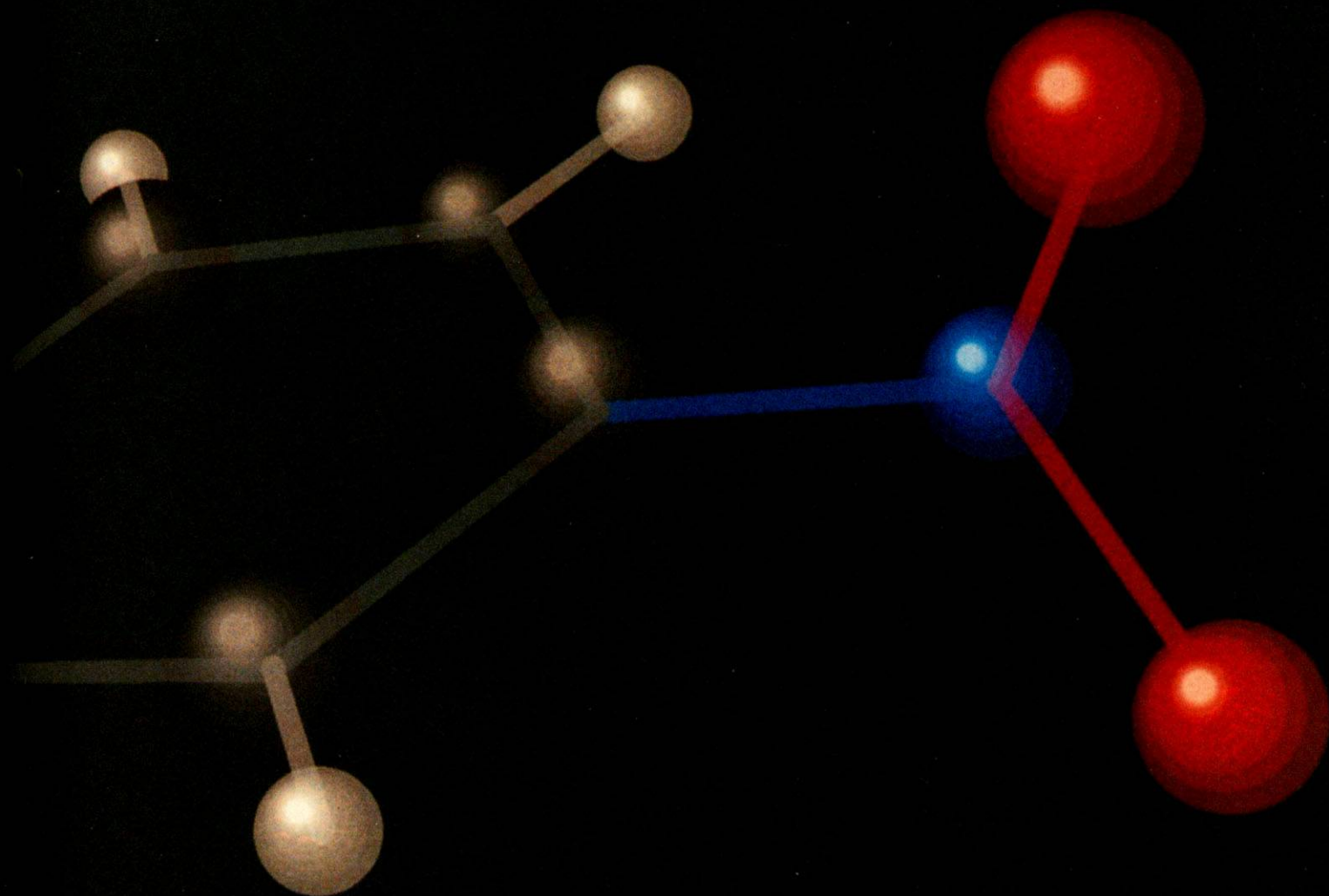


Bibliothek für Echtzeit-Vektorgrafik

AUF DEM WEG ZU

NEUEN WELTEN

Rasante Flüge
durch das All.
Raumschiffe in
Richtung Zukunft.
Ein neues Gefühl
für Zeit und Raum.
Das Erleben neuer
Dimensionen.
Keine Utopie,
sondern Realität:
Vektorgrafik
macht's möglich.



**Von Heiko Schlichting,
Dirk Mehren und Martin Jobst**

Viele tolle Spiele für den Amiga faszinieren mit animierter Vektorgrafik: Elite, Starglider oder Starwars. Sicher hatten Sie auch schon einmal den Wunsch, Ihre Programme mit solcher Grafik auszustatten. Wenn Sie bis jetzt nicht wußten wie, bieten wir Ihnen eine optimale Lösung: die C-Bibliothek »VG«. Mit Hilfe der eingebauten, schnellen Grafikroutinen können Sie Ihre C-Programme mit schneller Vektorgrafik versehen. Die Anwendung ist denkbar einfach.

Für viele Zwecke reicht die zweidimensionale Darstellung der Amiga-Systembibliotheken nicht aus, gerade Spiele gewinnen durch dreidimensionale

Grafik an Reiz. Die effektive Darstellung räumlicher Objekte ist die Domäne der Vektorgrafik. Sie erlaubt die schnelle Berechnung sowie die räumliche Darstellung und Bewegung aller Körper, die sich aus Linien aufbauen lassen.

Ist beispielsweise beim Raytracing-Verfahren das Ziel die realitätsnahe Darstellung, bietet Vektorgrafik die Möglichkeit, schnell einfache Objekte zu konstruieren. Da Vektorgrafik nur einen Bruchteil der Rechenleistung eines Raytracing-Bildes benötigt, kann eine Bewegung auch in Echtzeit berechnet und dargestellt werden. Was bedeutet, daß die einzelnen Bilder der Animation nicht im voraus, sondern Bild für Bild während des Ablaufs »live«, also in Echtzeit berechnet wird. Dazu sind jedoch sehr aufwendige mathematische Berechnungen nötig. Mit Hilfe der Matrizenrechnung können diese

aber übersichtlich und schnell durchgeführt werden.

Vektorgrafik oder »VG« arbeitet objektorientiert, wobei ein Objekt aus einzelnen Punkten besteht, die Sie räumlich relativ zueinander setzen, verbinden und frei im Raum bewegen können. »VG« ist die Weiterentwicklung eines Moduls, das Strukturen von Molekülen darstellen kann.

Um Ihnen den Gebrauch von Vektorgrafik möglichst einfach zu machen, sind alle nötigen Funktionen so geschrieben, daß Sie Objekte leicht kreieren und in eigenen Programmen benutzen können. Ein Beispielprogramm finden Sie in Listing 3.

»VG« erledigt alle komplizierten mathematischen Berechnungen mit wenigen Befehlen, deshalb ist es nicht nötig, daß Sie die mathematischen Grundlagen allzu genau kennen. Um uns auf wesentlichere Dinge zu ►

beschränken, verweisen wir für weitergehende Informationen auf ein beliebiges Mathematikbuch, das sich mit linearer Algebra befaßt. Nachfolgend finden Sie eine Beschreibung aller Funktionen von »VG«, die Sie außerdem in Tabelle 1 als Übersicht finden. Den Source-Code von »VG« sehen Sie in Listing 1, das zugehörige Include-File »VG.h« zeigt Listing 2. Nebenstehende Kästen zeigen Ihnen, was Sie beim Compilieren und Linken des Programms beachten sollten.

VGopen:

Diese Funktion muß immer als erste aufgerufen werden, da sie die nötigen Amiga-Bibliotheken (»Intuition«, »Graphics«, »Mathfp« und »Mathtrans«) bereitstellt. Außerdem reserviert sie den benötigten Speicher und öffnet einen Screen in der gewünschten Auflösung. Die Farben werden auf voreingestellte Standardwerte gesetzt, Sie können sie aber nachträglich Ihren Bedürfnissen anpassen.

Mit den Parametern »AufsgX«, »AufsgY« und »Bitplanes« bestimmen Sie den Typ des Screens. Mögliche Werte für die horizontale Auflösung sind 320 oder 640, für die vertikale 200, 256, 400 oder 512. Der Screen kann bis zu fünf Bitplanes erhalten. Bedenken Sie aber, daß jede zusätzliche Bitplane neben der Farbenpracht auch Rechenzeit kostet. Die Anzahl der für die Linien zur Verfügung stehenden Farben können Sie mit der Formel »Farben=2^{Bitplanes-1}« berechnen.

Damit Sie sich nicht um die interne Verarbeitung der Koordinaten kümmern müssen, benötigt das Programm die maximale Anzahl der später verfügbaren Punkte und Linien. Diese Werte werden in »MaxLines« und »MaxPoints« an die Funktion übergeben. Sie sollten diese Werte ruhig etwas größer dimensionieren, um einen Puffer zu haben. Die einzige Begrenzung ist der zur Verfügung stehende Speicherplatz.

Mit dem letzten Parameter von »VGopen« hat es eine besondere Bewandnis. Da »VGopen« einen eigenen Screen öffnet und Sie deshalb Fehlermeldungen nicht sehen können, gibt es die Möglichkeit, diese in ein Textfile schreiben zu lassen. Es genügt, wenn Sie dem Parameter »ErrFile« einen Filenamen samt Pfad übergeben.

In unserem Beispiel haben wir die Ausgabe in das File

»RAM:SYSPRINT« gesendet. Später folgende Fehlermeldungen werden dem bestehenden File hinzugefügt. Damit Sie die Übersicht bewahren, werden den Fehlern auch Datum und Uhrzeit sowie die Funktion, die den Fehler signalisierte, mitprotokolliert. Auf diese Weise können Fehler später besser nachvollzogen werden. Wenn Sie »ErrFile« auf Null setzen, können Sie die Fehlerausgabe unterbinden. Eine Ausgabe nach »stderr« (normalerweise das aufrufende CLI) erfolgt in jedem Fall. Eine genaue Beschreibung aller möglichen Fehlermeldungen samt ihren Ursachen finden Sie in Tabelle 2.

Screen gesetzt. Diese können mit »VGcolor« geändert werden. Die ersten beiden Parameter bestimmen Anzahl und Position der Farbe(n) in der Palette. Verändern Sie Farben, die bei den gewählten Bitplanes nicht genutzt werden, so zeigen sich keine Auswirkungen auf die Darstellung. Der letzte Parameter ist ein Zeiger auf eine Farbtabelle vom Typ »VGColor« (siehe Tabelle 3, dort werden die vom Include-File »VG.h« vordefinierten Strukturen erklärt).

VGpoint:

Diese Funktion definiert einen Punkt in der Darstellung. Übergeben Sie ihr die Objekt-

Linker-Optionen

Das File »VG.O« sowie ein entsprechendes weiteres Programm wie »VGTest.O« werden zusammen mit der »m32.lib« und der »c32.lib« gelinkt. Die »m32.lib« ist die Mathematik-Library für 32 Bit Integer. Die »c32.lib« enthält die Aufrufe für die Amiga- und Aztec-Funktionen. Wichtig ist beim Aufruf die Einhaltung der Reihenfolge: zuerst die »O«-Files, dann die »m32.lib« und die »c32.lib«.

Die verwendeten Optionen sind »-o file« und »-Lname«. Die »-o file«-Option erlaubt Ihnen, den Namen des ausführbaren Files anzugeben. »-Lname« durchsucht die angegebene Library nach benötigten Modulen.

Anweisungen:

In VG.O VGTEST.O -Lm32 -Lc32 -o VGTEST

Compiler-Optionen

Die Files können mit dem Aztec C68K 3.6a compiliert werden. Verwenden Sie dabei folgende Compileroptionen:

-e140 +1

»-e140« vergrößert den »Expression-Space«. Diese Option wird beispielsweise für mathematische Ausdrücke benötigt. Man erweitert damit die mögliche Länge von Ausdrücken, die der Compiler verarbeiten kann.

»+1«: interne Variablen sind nicht mehr 16, sondern 32 Bit groß

Anweisungen:

cc -e140 +1 VG.C

cc +1 VGTEST.C

VGclose:

Diese Funktion beendet die Vektorgrafikdarstellung. Der reservierte Speicher wird wieder freigegeben, der Screen und die Bibliotheken geschlossen. Mit dem Parameter »end« können Sie bestimmen, ob Sie Ihr Programm mit dieser Funktion verlassen möchten (FALSE= Programm beenden). Da die in den meisten Fällen nicht gewünscht wird, sollten Sie diese Funktion mit VGclose(TRUE); aufrufen, woraufhin nur die Vektorgrafikfunktion beendet wird.

VGcolor:

Beim Aufruf von »VGopen« werden bereits voreingestellte Farbwerte für den neuen

nummer (1-32767) und ein Feld mit den drei Koordinaten. Dieses Feld sollte vom Typ »union VGzahl MeineKoordinaten[VG-DIM]« sein. Sie können die X-Koordinate dann mit »MeineKoordinaten[0].f=(float)0.0234« zuweisen.

Farbenpracht

Die Y- und Z-Koordinate können Sie entsprechend zuweisen, setzen Sie nur entsprechend [1] bzw. [2] ein. Dabei ist wichtig, daß Sie Variablen vom Typ »union VGzahl« als float-Variablen ansprechen. Sie müssen den Suffix »f« bei den Variablen angeben. Der Wertebereich der Koordinaten reicht unabhängig von der Auflösung

immer von -1.0 bis +1.0. Dies gilt für die horizontale wie für die vertikale Achse. Der Ursprung eines von »VG« definier-

Funktionen

Return-Wert	Funktion
void	VGopen (AufsgX, AufsgY, int MaxLines, MaxPoints, ErrFile)
BitPlanes,	
void	VGclose (end)
BOOL	VGcolor (Pos, Anzahl, Colors)
void	VGpoint (Obj, Koord)
void	VGline (Num, Anfang, Ende, Color)
void	VGeraseobj (PointObj)
void	VGeraseline (LineNum)
BOOL	VGreaddata (FileName)
void	VGrotate (RMatrix, Xangle, Yangle, Zangle, clear)
void	VGtranslate (RMatrix, E1, E2, E3, length, clear)
void	VGglobalscale (RMatrix, scalefactor, clear)
void	VGlocalscale (RMatrix, E, scalefactor, clear)
void	VGtransform (Obj, Matrix)
void	VGdisplay (clear)

Tabelle 1. Alle von »VG« zur Ver-

ten Koordinatenkreuzes liegt in der Mitte des Bildschirms.

Die Objektnummer eines Punktes kann von Ihnen frei ge-

und Parameter

Parameter-Typ
int int
unsigned int unsigned int char
BOOL
unsigned int unsigned int struct
int union
int int int short
int
int
char
union (siehe Text) int int itn BOOL
union (siehe Text) float float float float BOOL
union (siehe Text) float BOOL
union (siehe Text) foat float float BOOL
int union
BOOL

fügung gestellten Funktionen

wählt werden. Sie ermöglicht, daß Sie später Teilobjekte bewegen, drehen oder vergrößern können. Dies erreichen Sie, indem Sie mehreren Punkten die gleiche Objektnummer zuweisen und diese dann gemeinsam - als einen Körper ansprechen. Die Objektnummer 0 ist reserviert und bezeichnet die gesamten Punkte. Geben Sie daher nie einem Punkt oder einer Linie die Objektnummer 0.

VGline:

Zwei mit »VGPoint« definierte Punkte können durch eine Linie miteinander verbunden werden. Sie übergeben der Funktion »VGline« dazu eine Objektnummer, das heißt die Linie wird zu einem eigenen Objekt, oder Teil eines solchen. Dies erweist sich für Bewegungsvorgänge als günstig.

Start- und Endpunkt können Sie nicht mittels der Objektnummer angeben, vielmehr ergeben sich deren Nummern aus der Reihenfolge, wie Sie die Punkte mit »VGpoint« definieren. Der erste Punkt, den Sie bestimmen, erhält die Nummer 0, der zweite die Nummer 1 usw. Verwechseln Sie diese Nummerierung nicht mit den Objektnummern, die Sie bei den anderen Funktionen angeben müssen.

VGeraseobj:

Mit »VGeraseobj« können Sie ein bestimmtes Objekt löschen. Sie übergeben die Objektnummer und alle mit dieser Nummer bezeichneten Punkte und Linien werden gelöscht.

VGeraseline:

Diese Funktion löscht ausschließlich Linien eines Objekts. Sie brauchen dazu nur die Objektnummer der Linie(n) anzugeben.

VGreaddata:

Wollen Sie in einem Programm kein festes Objekt definieren, sondern die Daten aus einem File einlesen, haben Sie mit dieser Funktion die Möglichkeit dazu. Das Datenfile muß dazu in einem ganz bestimmten Format vorliegen:

Anzahl der Punkte
Anzahl der Linien
Objektnummer des Punktes X-, Y-, Z-Koordinate
Objektnummer des Punktes X-, Y-, Z-Koordinate
...
Anfangspunkt/Endpunkt/
Farbe der Linie
Anfangspunkt/Endpunkt/
Farbe der Linie
...

Das File kann mit einem beliebigen Texteditor erstellt werden. Der einzige Parameter, der an die Funktion »VGreaddata« übergeben wird, ist der vollständige Name Ihres Datenfiles. Ein Beispiel finden Sie in Tabelle 4.

Sie haben außerdem die Möglichkeit, mit den Funktionen »VGpoint« und »VGLine« ein eigenes Modul zum Einlesen von Objektdaten zu erstellen.

VGrotate:

Nun kommt Bewegung ins Spiel. Mit »VGrotate« haben Sie die Möglichkeit, eine zuvor deklarierte Matrix mit der Abbildungsmatrix für Rotationen um die X-, Y- und Z-Achse zu initialisieren.

Bevor die einzelnen Parameter erläutert werden, sei darauf hingewiesen, daß für diese Abbildungsmatrix ein Feld in der Form

```
union VGzahl MeineMatrix
[VGDIM][VGDIM]
```

reserviert werden muß. Ein Zeiger auf diese Matrix stellt dann auch den ersten Parameter der Funktion »VGrotate« dar. Weiterhin können Sie die verschiedenen Drehwinkel um die drei Achsen des Koordinatensystems angeben. Die Werte von »Xangle«, »Yangle« und »Zangle« werden im Gradmaß erwartet und müssen Integerwerte im Bereich von -359 bis 359 sein. Es ist möglich, jedoch nicht not-

wendig, ein Objekt um alle drei Achsen rotieren zu lassen. Wenn nur eine Matrix erstellt werden soll, die beispielsweise das Objekt um die X-Achse dreht, setzen Sie »Yangle« und »Zangle« auf 0 Grad.

Drehmomente

Der letzte Parameter der Funktion ist vom Type Bool und für die besondere Möglichkeit der Kombination einzelner Bewegungsmatrizen vorgesehen. Wird an dieser Stelle »TRUE« angegeben, so beachtet »VGrotate« die alten Werte Ihrer Matrix nicht und überschreibt sie einfach. Im Falle eines »FALSE« wird jedoch die erzeugte Rotationsmatrix der vorhandenen angehängt. Dies hat zur Folge, daß bei Ausführung beide Abbildungen (Bewegungen) simultan durchgeführt werden. Wie viele Abbildungsmatrizen Sie deklarieren, initialisieren und verwalten, liegt in Ihrem eigenen Ermessen.

Dazu ein Beispiel:

```
VGrotate(MeineMatrix,
0,2,-15,TRUE)
```

Damit speichern Sie die reine Rotationsmatrix mit den Werten für eine gleichzeitige Drehung um die Y-Achse mit 2 Grad und um die Z-Achse um -15 Grad (345 Grad) in »MeineMatrix«.

Fehlermeldungen

»OpenLibrary (Intuition) ERROR.«

Die Intuitionlibrary konnte nicht geöffnet werden, ein gravierender Fehler, der normalerweise nicht vorkommen sollte. Bei seinem Auftreten ist ein Systemabsturz nicht zu vermeiden.

»OpenLibrary (Gfx) ERROR.«

Die Grafik-Library wurde nicht gefunden. Auch dieser Fehler tritt im Normalfall nicht auf, da diese Library Bestandteil des ROMs ist und deshalb immer verfügbar ist.

»OpenLibrary (MathFFP) ERROR.«

Die Mathematik-Library konnte nicht gefunden werden. Auch diese Bibliothek ist Teil des ROMs und sollte immer vorhanden sein. Prüfen Sie bei Auftreten dieses Fehlers, ob die Version Ihres Kickstart kleiner als 1.2 ist, oder sie eine unerlaubt modifizierte Kickstart-Version benutzen.

»OpenLibrary (MathTrans) ERROR.«

Die Mathematik-Library für transzendente Funktionen konnte nicht gefunden oder geöffnet werden. Dies ist der Fall, wenn sich die »math-trans.library« nicht im »LIBS:«-Ordner befindet. In diesem Fall sollten Sie die Library von der Workbenchdiskette in Ihren aktuellen »LIBS:«-Ordner kopieren.

»OpenScreen ERROR.«

Der für die Grafik nötige Screen konnte nicht geöffnet werden. Entweder reichte das nötige Chip-Memory nicht aus, oder Sie versuchten auf einem NTSC-Amiga einen Screen mit PAL-Auflösung zu öffnen.

Tabelle 2. Alle Fehlermeldungen von »VG«

»Out of Memory ERROR.« Der Speicherplatz reicht nicht aus. Verkleinern Sie in diesem Fall die Werte von »MaxLines« und »MaxPoints« der Funktion »VGOpen« oder wählen Sie eine niedrigere Auflösung mit weniger Farben.
»Invalid Viewresolution (X) ERROR.« Der Wert für die horizontale Auflösung bei »VGOpen« war nicht zulässig. Benutzen Sie nur die Werte 320 oder 640. Es wird in diesem Fall die voreingestellte Auflösung von 320 Pixeln automatisch gewählt.
»Invalid Viewresolution (Y) ERROR.« Der Wert für die vertikale Auflösung, den Sie als Parameter der Funktion »VGOpen« angegeben haben, war nicht erlaubt. Verwenden Sie nur die Werte 200, 256, 400 oder 512. (Default: 200)
»Bitplane ERROR.« Der Wert für die Anzahl der verwendeten Bitplanes darf nur im Bereich von 1 bis 5 liegen (1-31 Farben + Hintergrundfarbe). Bei einem Fehler werden zunächst zwei Bitplanes benutzt.
»Nothing to do: LOGIC ERROR.« Sie haben angegeben, daß Sie keinen einzigen Punkt und keine Linie definieren wollen. Was wollen Sie dann mit diesem Programm? Abhilfe: Verwenden Sie sinnvolle Werte bei »MaxLines« und »MaxPoints«.
»Requires VGopen ERROR.« Alle Funktionen erwarten, daß Sie zunächst die Funktion »VGOpen« aufrufen. Das haben Sie in diesem Fall wohl vergessen und sollten dies nachholen. Rufen Sie »VGclose« auf, so sollte die nächste Funktion, die Sie aufrufen, ebenfalls »VGopen« sein.
»Out of RGB bounds ERROR.« Die Farben dürfen nur mit Werten von 0 bis 15 definiert werden. Verwenden Sie Werte, die nicht in diesem Bereich liegen, so werden die Werte als »MODULO 16« interpretiert.
»Out of Colortable ERROR.« Die von Ihnen definierte Farbpalette kann maximal 32 Farben aufnehmen. Dies ist unabhängig von der Anzahl der tatsächlich benutzten Bitplanes. Sie versuchten aber mehr als 32 Farben zu definieren. Beachten Sie bitte auch, daß die Farben von 0 bis 31 numeriert werden.
»Second VGopen without VGclose ERROR.« Bevor Sie »VGopen« ein zweites Mal in einem Programm aufrufen können, müssen Sie mit der Funktion »VGclose« den Speicher freigeben und den alten Screen schließen. Abhilfe: fügen Sie entweder ein »VGclose« ein oder überprüfen Sie, ob nicht »VGopen« unabsichtlich mehrfach hintereinander aufgerufen wurde.
»Number of Lines > MaxLines ERROR.« Sie versuchten mehr Linien zu definieren, als Sie in »VGopen« als Maximum angegeben haben. Vergrößern Sie den Wert von »MaxLines«, wenn Sie genug Speicherplatz haben.
»Number of Points > MaxPoints ERROR.« Sie versuchten mehr Punkte zu definieren, als Sie in »VGopen« als Maximum angegeben haben. Vergrößern Sie den Wert von »MaxPoints«.
»Line with undefined Point ERROR.« Linien können nur zwischen existierenden Punkten gezogen werden. Sie sprachen aber eine Punktnummer an, die Sie nicht definiert haben.
»File not found, can't open it ERROR.« Die Funktion »VGreaddata« kann das angegebene File nicht öffnen. Überprüfen Sie Pfad- oder Filenamen.
»Out of Line/Point-Memory ERROR.« Sie versuchen mehr Punkte oder Linien als zulässig einzulesen. Vergrößern Sie die Werte von »MaxLines« und »MaxPoints« oder prüfen Sie, ob es sich bei dem File um ein zulässiges Datenfile handelt.

Tabelle 2. (Fortsetzung)

VGtranslate:

Um ein Objekt zu verschieben, wird die Funktion »VGtranslate« benötigt. Sie erzeugt die Parallelverschiebungsmatrix, die bei Anwendung alle angeählten Punkte um einen von Ihnen angegebenen Vektor verschiebt. Der erste Parameter ist wie zuvor bei »VGrotate« der Zeiger auf eine Matrix. Die nächsten drei Werte, die übergeben werden, stellen die drei Komponenten des Vektors (X,Y,Z) dar und sollten – müssen aber nicht – im Wertebereich [-1.0, +1.0] sein.

Als weiterer Parameter folgt die Verschiebungslänge vom Typ »FLOAT«. Durch den Wechsel des Vorzeichens kann die

Richtung der Verschiebung umgekehrt werden. Der letzte Wert, der an »VGtranslate« übergeben wird, entspricht in seiner Funktion dem letzten Parameter von »VGrotate«.

Bei unserem Beispiel

```
VGtranslate(MeineMatrix,
0.5,-0.2,0.0,-0.1,FALSE)
```

wird eine Verschiebungsmatrix erzeugt, die um -0.05 in X-Richtung und 0.02 in Y-Richtung

Einer für alle

verschiebt. Diese Matrix wird auf die schon bestehende Rotationsmatrix (siehe oben) in »MeineMatrix« aufmultipliziert. Auf diese Weise können Sie ei-

Strukturen von »VG.h«

```
union VGzahl {
    float f;
    int i;
};
```

»Union VGzahl« enthält eine von »VG« benutzte Koordinate. Sie benötigen diesen Typ für die Definition einer Bewegungsmatrix. Eine Matrix können Sie folgendermaßen definieren:
union VGzahl MeineMatrix[VGDIM][VGDIM];

Dadurch haben Sie eine Matrix mit dem Namen »MeineMatrix« definiert, die alle von »VG« erzeugten Bewegungen ausführen kann. (Der Wert von »VGDIM« ist 4.)

```
struct VGLine {
    int    Number;
    int    StartPoint;
    int    EndPoint;
    short  Color;
};
```

Diese Struktur definiert eine Linie in »VG«. Dieser Typ wird vorwiegend intern benötigt, Sie können diese aber auch verwenden, um auf einfache Weise Linien zu definieren. Die Strukturelemente enthalten die Objekt-, Startpunkt- und Endpunktnummer sowie die Farbe. Beachten Sie, daß die Funktion »VGLine« die Strukturelemente einzeln benötigt und Sie diese Struktur nicht direkt übergeben können. Eine Anwendung sehen Sie in unserem Beispielpogramm (Listing 3).

```
struct VGPoint {
    int    Object;
    union VGzahl Koord[VGDIM];
};
```

Entspricht weitestgehend der Struktur »VGLine«. Die Strukturelemente sind in diesem Fall aber Objektnummer und Koordinaten des Punktes. Beachten Sie, daß der Wert von »VGDIM« 4 ist, Sie aber nur Punkte im dreidimensionalen Raum definieren können. Die vierte Koordinate wird von »VG« für interne Berechnungen verwendet, bei der Definition eines Punktes wird der angegebene Wert ignoriert und immer 1.0 dafür eingesetzt.

```
struct VGColor {
    unsigned short R,G,B;
};
```

Diese Struktur definiert eine Farbe der Farbpalette. Die Strukturelemente sind die Farbanteile in Rot, Grün und Blau. Der zulässige Bereich für diese Werte sind 0 bis 15. Mit dieser Struktur wird eine Farbpalette an die Funktion »VGcolor« übergeben.

Tabelle 3. Die in »VG.h« (Listing 2) vordefinierten Strukturen

ne kombinierte Dreh- und Verschiebewegung erzeugen.

VGglobalscale:

Auch diese Funktion erzeugt eine spezielle Abbildungsmatrix, jedoch eine für die allseitige Dehnung. Mit ihr lassen sich einzelne oder alle Objekte vergrößern bzw. verkleinern. Diese Funktion benötigt folgende Parameter: »RMatrix« ist wieder der Zeiger auf eine Ihrer Matrizen. »scalefactor« – vom Typ »FLOAT« – ist der Dehnungsfaktor, mit dem Sie bestimmen, ob vergrößert (Werte größer als 1.0) oder verkleinert (Werte kleiner als 1.0) werden soll. Die Bedeutung von »clear« entspricht der bei »VGrotate« und »VGtranslate«.

VGlocalscale:

Alle Parameter dieser Funktion entsprechen denen von »VGglobalscale«. Zusätzlich muß jedoch noch ein Zeiger auf einen Richtungsvektor (mit drei Komponenten X,Y,Z) vom Typ »FLOAT« übergeben werden. Der Unterschied zu der Funktion »VGglobalscale« liegt darin, daß durch den Vektor eine gerichtete Dehnung erzeugt wird. Stellen Sie sich dazu Ihr Objekt aus elastischem Material bestehend vor, das an zwei gegenüberliegenden Seiten festgehalten und auseinandergezogen wird. Dazu ein Beispiel:

Beispiel:
float MeinVektor[3];

VGlocalscale
(MeineMatrix,MeinVektor,
0.055,TRUE)

VGtransform:

Damit können Sie alle zuvor erzeugten Abbildungsmatrizen auf Ihre Objekte anwenden. Als erster Parameter müssen Sie die Objektnummer, die vom Typ »Integer« sein muß, übergeben. Wollen Sie alle Objekte gleichzeitig ansprechen, muß dieser Parameter gleich Null sein. Als zweites wird ein Zeiger auf die anzuwendende Matrix übergeben. Die Funktion »VGtransform« verändert die Punktkoordinaten der einzelnen Objekte, nicht jedoch die übergebene Matrix.

Durch mehrfachen Aufruf mit derselben Matrix kann so eine Bewegung erzeugt werden. Zu beachten ist, daß diese Funktion die Objekte nicht sofort darstellt, was den Vorteil hat, daß Sie ein Objekt an eine bestimmte Position bewegen können, ohne es zu zeigen.

Daten

36	36			4	0.2	-0.5	0.1	15	16	1
1	-0.8	-0.5	-0.1	4	0.8	-0.5	0.1	16	17	1
1	-0.6	-0.5	-0.1	4	0.8	0.5	0.1	17	10	1
1	-0.2	-0.5	-0.1	4	0.2	0.5	0.1	10	11	1
1	0	-0.5	-0.1	4	0.35	-0.35	0.1	11	7	1
1	-0.4	0.5	-0.1	4	0.8	-0.35	0.1	18	19	3
1	-0.4	0.1	-0.1	4	0.5	0	0.1	19	23	3
2	0.2	-0.5	-0.1	4	0.8	0	0.1	23	20	3
2	0.8	-0.5	-0.1	4	0.5	0.15	0.1	20	21	3
2	0.8	0.5	-0.1	4	0.65	0.15	0.1	21	22	3
2	0.2	0.5	-0.1	4	0.65	0.35	0.1	22	18	3
2	0.35	-0.35	-0.1	4	0.35	0.35	0.1	24	25	3
2	0.8	-0.35	-0.1	0	1	1		24	27	3
2	0.5	0	-0.1	1	5	1		26	27	3
2	0.8	0	-0.1	5	2	1		26	31	3
2	0.5	0.15	-0.1	2	3	1		31	30	3
2	0.65	0.15	-0.1	3	4	1		30	32	3
2	0.65	0.35	-0.1	4	0	1		32	33	3
2	0.35	0.35	-0.1	6	7	1		33	34	3
3	-0.8	-0.5	0.1	6	9	1		34	35	3
3	-0.6	-0.5	0.1	8	9	1		35	28	3
3	-0.2	-0.5	0.1	8	13	1		28	29	3
3	0	-0.5	0.1	13	12	1		29	25	3
3	-0.4	0.5	0.1	12	14	1				
3	-0.4	0.1	0.1	14	15	1				

VGdisplay:

Um die durch »VGtransform« hervorgerufene Positionsveränderung der Objekte sichtbar zu machen, wird »VGdisplay« benötigt. Diese Funktion hat nur einen Parameter vom Typ Bool, der angibt, ob der Bildschirm gelöscht werden soll (TRUE) oder nicht (FALSE).

Zum Abschluß der Funktionen von »VG« noch einige Worte zum Koordinatensystem. Der auf dem Bildschirm sichtbare Teil des Koordinatensystems ist normiert und liegt im X-Intervall von -1.0 (links) bis 1.0 (rechts) sowie im Y-Intervall von -1.0 (oben) bis 1.0 (unten). Die Ko-

zeugten Bewegungen nicht direkt umgesetzt: die Objektbewegung ist auf die zuvor erzeugten, »objektunabhängigen« Matrizen angewiesen. Die ermöglichen auch die Zusammenfassung von beliebig vielen Matrizen zu einer Bewegungs-bibliothek. Diese können Sie dann auf verschiedene Objekte anwenden.

Sie werden sehen, »VG« wird sehr bald zu einem Werkzeug

werden, das Sie nicht mehr missen wollen. Nutzen Sie seine fantastischen Fähigkeiten und begeben Sie sich auf den Weg in neue Welten... so

Heiko Schlichting studiert Biochemie in Berlin und arbeitet als freier Mitarbeiter für Zeitschriften des Markt & Technik Verlages. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Heiko Schlichting, Hans-Pinsel-Straße 2, 8013 Haar).

Zum Koordinatensystem

ordinaten der Punkte sollten sich in diesem Bereich befinden.

Um Ihnen die Anwendung von »VG« zu erleichtern, finden Sie in Listing 3 das Demoprogramm »VGTEST.C«. Dieses Demo benötigt die Daten in Tabelle 4 als ASCII-File. Geben Sie bitte diese mit einem Editor ein (die Zahlen jeweils mit einem Leerzeichen voneinander getrennt, am Ende jeder Zeile ein RETURN).

Im ersten Teil des Programms wird ein Schriftzug in einer vorgegebenen Form bewegt. Im zweiten Abschnitt kann die Animation mit Hilfe der Maus beeinflusst werden. Dabei werden die mit der Maus er-

```

/*****
|   VG - Vector Graphics   |
|*****|
|   Heiko Schlichting     |
|   Dirk Mehren           |
|   Berlin / Cuxhaven     |
\*****/

#include "vg.h"

ULONG mask=0;
struct VGColor VGDefaultColors[32] = {
    0x0,0x0,0x0,0xf,0xf,0xf,0xf,0xf,0x0,0x0,0xf,0x0,
    0x0,0x0,0xf,0x9,0x9,0x9,0xf,0xf,0x0,0x0,0x6,0xd,
    0xc,0x1,0xf,0xc,0x8,0x0,0x0,0xd,0xb,0xf,0x9,0x0,
    0xc,0xc,0xc,0x6,0xf,0xc,0xd,0x0,0x0,0x0,0xb,0xb,
    0xf,0xa,0xc,0x6,0x1,0xf,0xf,0xd,0x0,0x8,0xc,0x0,
    0x9,0x1,0xf,0x1,0xf,0xb,0xf,0x8,0x0,0x2,0xc,0x0,
    0xd,0xb,0x9,0x6,0xc,0xc,0xa,0x8,0x7,0xb,0xf,0x0,
    0x0,0xb,0x1,0x7,0x7,0x3,0x3,0x3,0x5,0x5,0x5 };

struct NewScreen VGnewscreen = { 0,0,320,200,2,1,2,
    NULL,CUSTOMSCREEN_SCREENQUIET,
    NULL,(UBYTE *)" ",NULL,NULL };

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

```

Listing 1.
Mit »VG« erwecken Sie Ihre Spiele zum Leben
Fortsetzung auf Seite 140.

KREUZ

Von Klaus Sonnenleiter

Lassen Sie sich nicht von dem verkohlen, was andere Leute auf ihre Fahnen geschrieben haben. Für Ihre individuellen Anforderungen brauchen Sie speziell angepaßte Lösungen. Auch wenn Sie immer wieder von selbsternannten Gurus anderes zu hören bekommen, gibt es keine Universalsprache, die für alle Aufgaben geeignet ist.

Natürlich ist es verlockend, einem eindeutigen Wegweiser, beispielsweise dem Rat eines guten Freundes oder einer Fachzeitschrift, zu folgen. Bei der babylonischen Sprachverwirrung im Bereich der Programmiersprachen ist meist jede kleine Hilfe ein willkommener Richtungsanzeiger. Andernfalls fürchtet man, sich allzu schnell zu verirren, im Dschungel der diversen Pascal- und Modula-, C- und Assembler-, Lisp- und Prolog-, Fortran- und Basic-Dialekte.

Doch was dem Einsteiger recht ist, ist dem Profi billig. Erbitterte Grabenkämpfe herrschen zwischen Verfechtern aller Programmiersprachen. Verstehen Sie sich auch als Glaubenskrieger unter der Fahne »Ihrer« Programmiersprache? Das Computer-Nirwana werden Sie so nicht erreichen. Ehr-

licherweise müssen wir eingestehen, daß auch wir den Weg dorthin nicht kennen, aber eines ist gewiß: Den Gurus Glauben zu schenken, ist der sicherste Weg in den Absturz.

Jeder Programmierer hat seine Vorlieben und sich in einer Fremdsprache auszudrücken, bedeutet meist mehr Mühen als die Mängel der »eigenen« Sprache zu umgehen.

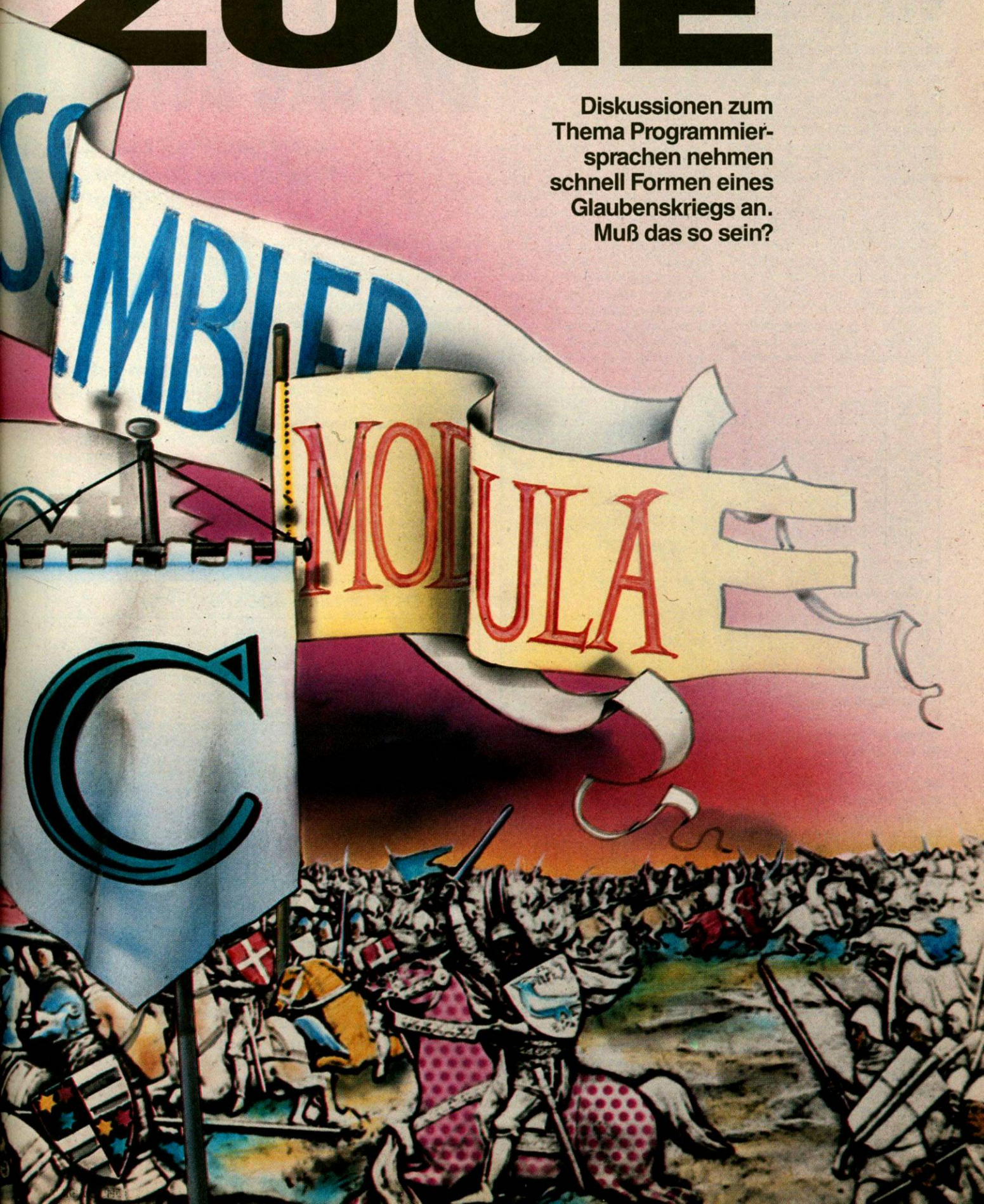
Dennoch: Oft genug ist es ärgerlich, wenn Sie Listings nur deshalb nicht verwenden können, weil sie in einer Sprache geschrieben sind, die Sie nicht verstehen.

Wir wollen Ihnen auf den folgenden Seiten Hinweise geben, wie Sie diese Schwierigkeit umgehen können. Sie finden zu den wichtigsten Programmiersprachen (C, Modula und Basic) wertvolle Richtlinien, mit denen Sie Programme dieser Sprachen in »Ihre« konvertieren können. Natürlich können diese nie perfekt sein. Aber sie sind der erste Schritt auf dem Weg zum Verständnis des jeweils anderen Lagers. so



ZÜGE

Diskussionen zum
Thema Programmier-
sprachen nehmen
schnell Formen eines
Glaubenskriegs an.
Muß das so sein?



Modula und C sind die wichtigsten auf dem Amiga verfügbaren Hochsprachen. In vielen Büchern und Zeitschriften finden Sie Programme, die in einer dieser Sprachen veröffentlicht sind. Natürlich brauchen Sie nicht in beiden Sprachen zu programmieren, um Nutzen aus diesen Programmen zu ziehen. Besser ist es, wenn Sie weiterhin in »Ihrer« Sprache programmieren. Sie sollten dann allerdings die jeweils andere Sprache lesen können. Wie Sie Programme zwischen diesen Welten austauschen, erfahren Sie im folgenden Artikel.

Bei der Programmierung des Amiga sind einige Grundkenntnisse in C fast lebensnotwendig. Modula ist wegen seiner steigenden Beliebtheit immer

cecode angegeben werden muß, was bei C nicht der Fall ist. Dieser Unterschied ist noch relativ unbedeutend, bei den weiteren handelt es sich dagegen um konzeptionelle Unterschiede.

Direkt nach dem Modulnamen steht in Modula die Import-Liste, die dazu dient, dem Compiler die sogenannten externen Referenzen (Namen, die er in anderen Files als dem aktuellen Sourcecode findet) zu erklären. In C wird hier für die Pre-Prozessoranweisung »#include« benutzt. Der Unterschied ist, daß es sich bei »include« um ein Einbinden von »Textbausteinen« in den Quelltext handelt, während in Modula nur die Referenzen definiert werden und erst beim Linken der – vorher schon compilierte – Code eingebunden wird.

In der Praxis heißt das, daß aus den beiden Modula-Zeilen

Modula	C
MODULE ProgrammName; FROM BibName IMPORT Proc1, Const1, Typ1; TYPE NeuerType = AnyType; CONST MyConst = Irgendwas; VAR MyVariable: Type; PROCEDURE MyProc (TestMe:Type1):Type2; BEGIN <Procedure-Code>; END MyProc; BEGIN <Programm-Code>; END ProgrammName.	- #include <directory/filename.h> typedef NeuerTyp IrgendeinTyp #define MyConst = Irgendwas; Type MyVariable; Type2 MyProc(TestMe) Type1 TestMe; { <Procedure-Code>; } <Programm-Code>; }

Tabelle 1. Eine Gegenüberstellung der Programmskelette für C und Modula

häufiger als Source-Code in Büchern und Artikeln zu finden.

Leider sind die C- und Modula-Compiler für den Amiga nicht objekt-kompatibel. Das bedeutet, daß keine Möglichkeit besteht, Programmteile, die in Modula formuliert sind, mit C-Modulen zu verbinden. Da die beiden Hochsprachen sich von der Struktur her sehr ähnlich sind, kann man ein Programm der einen Sprache aber relativ leicht verstehen und in die andere übertragen. Sie müssen dazu nur die wesentlichen Unterschiede der beiden Sprachen kennen.

Der erste kleine Unterschied, der bei der direkten Gegenüberstellung (siehe Tabelle 1) auffällt, ist, daß bei Modula der Name des Programms im Sour-

FROM Exec IMPORT Node;
FROM Dos IMPORT old
File;

folgende C-Zeilen werden:

#include <exec/nodes.h>
#include <dos/dos.h>

Wenn Sie wissen wollen, welche Struktur in welchem »Header« zu finden ist, würde ich jedem Programmierer das »ROM-Kernel-Reference-Manual: Libraries & Devices« vom Addison-Wesley-Verlag ans Herz legen: Dort sind im Anhang alle Include-Files aufgeführt. Die Endung nach dem Filenamen sagt aus, um was für eine Art von Quellcode es sich handelt:

»H« bezeichnet ein sogenanntes »Header-File«, wie es

Brücke zwisch

Programme von einer Sprache ist oft nicht ganz einfach. Wir für die Konvertierung

am häufigsten für die Bibliotheken benutzt wird. Der Inhalt der »H«-Files wird in Modula in den entsprechenden Bibliotheksmodulen (DEFINITION MODULE) definiert. In Listing 1 finden Sie ein Beispiel für Bibliotheksmodule in beiden Sprachen.

Mit »C« kann ein komplettes C-Programm als Source-Code eingebunden werden. Es handelt sich dabei meist um Routinen, die häufig benötigt werden. Modula bietet dieses Feature nicht, man kann dies aber mit jedem guten Editor selbst machen.

Viel interessanter ist aber die dritte Möglichkeit: In Files mit der Endung »O« sind vorcompi-

schied zwischen Groß- und Kleinschreibung. In C werden Konstanten mit einer weiteren Pre-Prozessoranweisung (»#define«) dem Compiler zugänglich gemacht.

Zuweisungen = Gleichmacherei?

Wie Typen von Modula nach C oder zurück übersetzt werden, entnehmen Sie Tabelle 2. Bei jedem übersetzten Typ sollten Sie sich jedoch Gedanken dazu machen, wofür er verwendet wird und ob er von den entsprechenden Prozeduren als

C-Typ	Modula-2-Typ
int	INTEGER
long int	LONGINT
unsigned int	CARDINAL
unsigned long int	LONGCARD
Byte	Exec.Byte
UByte	Exec.UByte
float	REAL / FFP
/* AMIGA C-Typen */	
WORD	CARDINAL
LONG	LONGINT
ULONG	LONGCARD
BYTE	BYTE
APTR	ADDRESS
BPTR	BPTR erst ab M2c V3.2 enthalten
BOOL	BOOLEAN

Tabelle 2. Die wichtigsten Typen von C und Modula gegenübergestellt

lierte Routinen enthalten. In Modula muß dazu ein DEFINITION und IMPLEMENTATION MODULE erzeugt werden.

Konstanten werden in Modula im Deklarationsteil (nach IMPORT, vor BEGIN des Hauptprogramms) definiert. Das Schlüsselwort »CONST« sagt dem Compiler, daß nun folgende Namen Konstanten mit dem dahinterstehenden Wert sind. Ebenso wie bei den Variablen macht Modula einen Unter-

Übergabeparameter akzeptiert wird.

Eine Zuweisung (Var1 »ist gleich« Var2) unterscheidet sich in den beiden Programmiersprachen kaum:

C Var1 = Var2;
MODULA Var1 := Var2;

Dies steht im Gegensatz zu einem Vergleich. Letztere werden etwas weiter unten bei den Schleifen erklärt.

hen Modula und C

en den Welten

in die andere zu übersetzen,
vermitteln Ihnen das Know-how
zwischen Modula und C.

Ein wichtiges Strukturelement beider Sprachen sind Zeiger auf Variablen. Eine Deklaration als Zeiger, beispielsweise auf eine Screen-Struktur, hätte folgendes Aussehen:

```
C      struct Screen .
        *MyScreen;
MODULA MyScreen:
        POINTER TO
        Screen;
oder   MyScreen:
        Intuition.ScreenPtr;
```

Die zweite Version in Modula ist möglich, da man auch Typen aus einer Bibliothek importieren kann, die schon als Zeiger (in unserem Fall auf einen Screen) definiert sind.

Die Adresse einer Variable wird folgendermaßen ermittelt:

```
Modula Addr := ADR(Var);
        wobei ADR() aus
        dem Standard-
        Modul SYSTEM
        importiert werden
        muß.
C      Addr = &Var;
```

Die Initialisierung von komplexen Strukturen ist in C denkbar einfach gelöst worden (die einzelnen Felder bei der Definition initialisiert – siehe unten).

```
struct NewScreen
MyNewScreen =
{
0,
0,
100,
200,
}
main()
{...}
```

In Modula dagegen bleibt keine andere Lösung, als die Struktur in einer eigenen Prozedur zu initialisieren. Dazu verwenden Sie am einfachsten folgende Schreibweise:

```
...
WITH MyScreen DO
leftEdge := 0; top
```

```
Edge := 0; width :
= 100;
height := 200;
...
END
...
```

Die Modula-Anweisung
YourScreen := MyScreen;
weist der Verbund-Variable YourScreen vom Typ »Intuition.Screen« den Inhalt der Variablen zu, auf die der Zeiger »MyScreen« zeigt. Das Dach hinter MyScreen sorgt dafür, daß nicht der Zeiger selbst, sondern die entsprechende Variable angesprochen wird (Dereferenzierung). Dies erreichen Sie in C folgendermaßen:

```
YourScreen = *MyScreen;
```

Man kann die beiden Ausdrücke auch umdrehen, (*MyScreen) := YourScreen, solange die Typen der Variablen übereinstimmen.

Mut zum neuen Typ

In Modula wandeln Sie eine Variable vom Typ ADDRESS mit folgender Zeile in den Typ POINTER TO Screen um:

```
MyScreenPointer :=
ScreenPtr(MyAddress);
```

wobei »MyScreenPointer« vom Typ ScreenPtr ist, und dieser importiert werden sollte.

Das Ganze sieht in C etwas anders aus:

```
MyScreenPtr = (struct
Screen*)MyAddress;
```

Das gleiche gilt auch von einer Sorte Zeiger zur anderen. Bedenken Sie jedoch, daß dies nur bei zuweisungskompatiblen Typen gilt, beispielsweise Zeigern oder wie INTEGER (int) zu LONGINT (long). Für alle anderen Konvertierungen gibt es in Modula die »Holzhammer«-

Printf()-Operator	InOut-Prozedur	Bedeutung
\n	WriteLn();	Beginnt eine neue Zeile.
\t	Write(ASCII.tab);	Zum nächsten Tabulator.
\r	Write(ASCII.cr);	Geht zum Anfang der aktuellen Zeile (carriage return)
\b	Write(ASCII.bs);	Löscht ein Zeichen (Backspace)
\f	Write(ASCII.ff);	Neue Seite. (Form Feed)
Das Modul ASCII stellt dem Modula-2 Programmierer alle (Steuer)Zeichen zur Verfügung. So können Sie alle Feinheiten von »C« simulieren.		
printf()-Formelement	InOut-Prozedur	Zu lesender Wert
%c	Read()/ Write()	Einzelner Buchstabe (CHAR).
%d	ReadInt()/ WriteInt()	Zahl ((LONG)INTEGER).
%s	ReadString()/ WriteString()	String (ARRAY OF CHAR).
%o	ReadOktal()/ WriteOktal()	Oktal-Zahl (INTEGER).
%x	ReadHex()/ WriteHex()	HexaZahl (INTEGER).
%u	ReadCard()/ WriteCard()	Zahlen ohne Vorzeichen ((LONG)CARDINAL)
Folgende Prozeduren stammen aus den Modulen LongRealInOut / FFPInOut		
%f	ReadReal()/ WriteReal()	Gleitpunktzahlen (REAL / FFP)
%e	dto.	dto.
Ein »l« als Zusatz in C bei %d, %u, %o, %x führt zu einer Long-Zahl. In Modula-2 sind diese zuweisungskompatibel. Das heißt es wird keine eigene Prozedur benötigt.		

Tabelle 3. »printf()« oder »WriteString«?

Methode mit Hilfe der Prozedur »CAST()«, die aus SYSTEM importiert werden kann.

In C gibt es nur eine Möglichkeit der Umwandlung:

```
(typ)(Variable)
```

also

```
VarTypLong =
(long)(VarAndererTyp)
```

Die Systembibliotheken sind unbestreitbar eines der wichtigsten Hilfsmittel bei der Programmierung des Amiga. Aber gerade hier herrscht zwischen Modula und C ein großer Unterschied:

Um in C auf eine Bibliotheks-routine zugreifen zu können, muß die Library geöffnet werden (dies gilt für alle außer der Exec-Library). Deshalb findet man oft am Anfang von Programmen den Aufruf wie

```
MyGfxBase = ( struct
GfxBase *)
OpenLibrary( "graphics.
library", 0)
```

oder ähnliches. Darum muß sich ein M2Amiga-Programmierer nicht kümmern. Diese

Arbeit besorgt das sogenannte »ARTS« (Amiga Run-Time System, das Laufzeit-System des Compilers). Wer unbedingt möchte, kann dies natürlich auch in Modula selbst übernehmen.

Jede Bibliotheksroutine, die Sie in Modula benutzen möchten, importieren Sie vorher mit der »FROM...IMPORT...«-Anweisung. Ein Beispiel für Bibliotheksaufrufe finden Sie in Listing 2.

Modula und C benutzen für einen Bibliotheksaufruf gelegentlich unterschiedliche Typen (sowohl für die Parameter als auch für den Rückgabewert). Dies kann bei der Konvertierung sehr ärgerlich werden, zumal Ihnen kein anderer Weg bleibt, als jedesmal im Compiler-Handbuch nachzuschlagen, welcher Typ der richtige ist.

Einige Routinen sind in Modula anders definiert als in C. So erhalten Sie als Rückgabewert für DoIO() in C einen Wert, der angibt, ob ein Fehler aufgetreten ist. Eine Variable »error« ist aber auch im Verbund der

Variable des Requests enthalten, den Sie ausführen (Request.error). Darum wurde in Modula dieser Rückgabewert weggelassen. Auf diese Feinheiten sollten Sie bei einer Übertragung sehr genau achten. Es erspart Ihnen viel Ärger.

Bei der Ein- und Ausgabe verwenden beide Sprachen grundlegend unterschiedliche Konzepte: printf() und scanf() sind im Standard der Sprache C enthalten. Modula kennt hierfür keinen vergleichbaren Standard. Bei allen Modula-Entwicklungssystemen (auch auf anderen Computern) wird es stattdessen die Module »InOut« und »FileSystem« geben. InOut behandelt die Bildschirm-Ein-/Ausgabe und FileSystem erledigt die Kommunikation mit externen Speichermedien.

Dabei gilt es, einen wichtigen Unterschied zu beachten: printf() und scanf() können alle Daten-Typen (Buchstaben, Zahlen, etc.) schreiben bzw. lesen, Sie geben den Typ einfach als Argument an. In Modula sind in InOut verschiedene Prozeduren für jeden Zweck vorhanden. Welche Angabe in printf()/scanf() mit welcher Write/Read-Procedure zu ersetzen ist, erkennen Sie aus Tabelle 3.

Ein häufiges Konvertierungsproblem ist das interne Format von Zeichenketten (Strings). Variablen vom Typ String sind in Modula als ARRAY OF CHAR definiert (parallel dazu in C: »char[]«). Sowohl in C als auch in Modula werden Strings automatisch mit einem abschließenden Null-Zeichen (0C) begrenzt.

Hier beginnt jedoch der Unterschied: Da man in C ein Feld auch »offen« deklarieren kann – ohne anzugeben, wie viele Einträge das Feld maximal haben darf – ist das abschließende Null-Zeichen immer Bestandteil des Strings.

In Modula muß jedoch ein Feld immer begrenzt sein (einzige Ausnahme: Parameterübergabe). Wenn ein Feld mit 79 Einträgen definiert und auch mit 79 Elementen »bis zum Rand« gefüllt ist, hat der Compiler keinen Platz mehr, um das »0C« anzuhängen.

Wozu der Aufstand, werden Sie sich fragen. Wenn in einem C-Programm eine Variable »offen« deklariert wird, und es ist nicht ersichtlich, wieviel Einträge maximal eingesetzt werden, so bleibt Ihnen in Modula nur der Weg über einen generischen Datentyp. Solche sind in

Modula nicht vorgesehen, es existieren jedoch Implementationen, beispielsweise auf den AMOK-Disketten.

Außerdem sind die Zugriffe auf einige Betriebssystem-Routinen entsprechend angepaßt. Ein gutes Beispiel ist die OpenDevice()-Funktion. Während in C als Device-Name ein beliebiger String akzeptiert wird, muß in Modula hier die Adresse dieses Strings angegeben werden. Auf die gleiche Weise arbeiten alle Systemfunktionen, denen ein String als Parameter übergeben werden muß.

Von Zweigen und Schleifen

Die einfachste Form einer Programmverzweigung ist das »IF«-Statement. Sie werden in jeder Programmiersprache ein ähnliches Element finden. Die Konvertierung ist daher denkbar einfach.

Aus der C-Verzweigung:

```
if (wert1 == wert2)
do_something();
else do_something_
else();
```

werden die Modula-Zeilen:

```
IF (wert1 = wert2) THEN
do_something() END;
```

Soll das C-Programm mehrere Anweisungen nach der Bedingung enthalten, so muß ein Block gebildet werden. Dies erreichen Sie auf folgende Weise:

```
if (wert1 == wert2)
{ /* Block beginnen! */
do_this();
do_that();
} /* Block beenden */
else
{ /* Block beginnen */
this_else();
that_else();
} /* Block beenden */
```

Derselbe Block (einschließlich ELSE-Zweig) sieht in Modula nicht viel anders aus:

```
IF Wert1 = Wert2 THEN
do_something();
(* Egal, ob eine oder
mehrere Zeilen! *)
ELSE
do_something_else();
(* Egal, ob eine oder
mehrere Zeilen! *)
END;
```

Sie können das »else«-Statement selbstverständlich auch weglassen.

Die Bedingungen einer Verzweigung (oder einer Schleife)

können in beiden Sprachen auch mit Hilfe eines logischen Operators verknüpft werden.

In Modula:

```
IF ((Bedingung1) AND
(Bedingung2) OR
(Bedingung3)) THEN
DoIt(); END;
```

In C:

```
if (bedingung1 &&
bedingung2 ||
bedingung3) do_it();
```

Sie schreiben also in Modula die Wirkungen der Operatoren aus, während in C die Kürzel benutzt werden. Nicht zuletzt ist diese Schreibweise verantwortlich für den schlechten Ruf von C in bezug auf die Lesbarkeit.

Bitte beachten Sie: Mit »Bedingung« meinen wir immer das Ergebnis einer logischen Operation: »Wert1 >= Wert2« oder ähnliches.

Nein, Nein, Nein (oder doch?)

Um das Gegenteil einer Aussage zu erhalten, kann in C ein Ausrufezeichen (!) vor die Klammer mit der Bedingung gesetzt werden.

```
if !(bedingung) do_it;
```

In Modula erreichen Sie den gleichen Effekt durch das Wort »NOT«.

```
IF NOT (Bedingung) THEN
DoIt() END;
```

Für ständig sich wiederholende Aufgaben verwendet man im allgemeinen eine Schleife. Um beispielsweise 30-mal eine Zahl zu erhöhen, schreiben Sie folgendes:

In Modula

```
FOR i:=0 TO 29 BY 1 DO
INC(Zahl); (* erhöhe
Zahl um eins. *)
END;
```

In C

```
for (i=0; i<29; i++)
Zahl = Zahl +1;
```

Die Anweisung »BY 1« in Modula ist nicht notwendig, da 1 immer als Schrittweite voreingestellt ist. In C müssen Sie dagegen stets die Schrittweite angeben.

Bei der for-Schleife in C gelten die gleichen Regeln, wie beim if-Statement: Soll nach der Schleifenanweisung nur eine einzige Zeile erscheinen, so brauchen Sie dafür keinen Block anzulegen. Bedenken Sie aber, daß Ihr Sourcecode durch einen Block übersichtlicher wird.

```
/* Sub-Routine: IncludeMe */
MyExternProc() /* Compilieren, NICHT linken */
{
/* kopieren: Z.B. RAM:IncludeMe.o */
<PrgCode>;
}

/* Hauptprogramm */
#include "RAM:IncludeMe.o"

main()
{
MyExternProc();
}

(* Definition des zu importierenden Files *)
DEFINITION MODULE IncludeMe;
PROCEDURE MyExternProc();
END IncludeMe.

(* Implementation des zu importierenden Files *)
IMPLEMENTATION MODULE IncludeMe;
PROCEDURE MyExternProc();
BEGIN
<Procedure-Code>;
END MyExternProc;
END IncludeMe.

(* Hauptprogramm *)
MODULE HauptProgramm;
FROM IncludeMe IMPORT MyExternProc;
BEGIN
MyExternProc();
END HauptProgramm.
```

Listing 1. »Import-Export« versus »Include«

AMIGA
SONDERHEFT

PROGRAMM-SERVICE

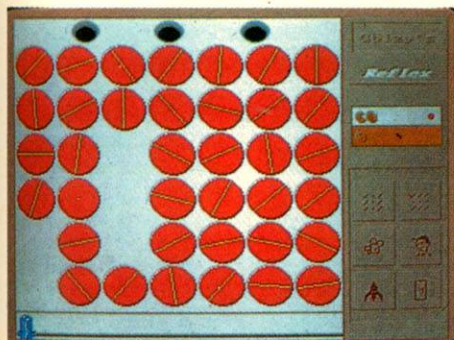
Direkt bestellen statt abtippen!

Die aktuelle Diskette zum Heft:

Amiga Sonderheft 7:

Reflex (siehe Abb. unten)

Das neueste Projekt von Arno Gölzer. »Reflex« ist der Nachfolger der beliebten Billardumsetzung. Lassen Sie sich von einer einzigartigen Spielidee und der tollen grafischen Umsetzung begeistern. *Beschreibung auf Seite 6ff.*



Noch mehr starke Spiele...

Ob Action- oder Strategiespiel - hier finden Sie alles, was das Spielerherz begehrt.

»Fast Freddie«

Die Beschreibung dieses Actionspiels finden Sie auf Seite 86



»Memory«

Eine interessante grafische Umsetzung für Leute mit gutem Gedächtnis. *Beschreibung auf Seite 97*

»Reactor«

Das Strategiespiel, das auch Atomkraftgegner fesseln wird.

Beschreibung auf Seite 107

»Schiebespiel«

Strategisches Denken und Tüfteln ist bei diesem Spiel das A und O.

Beschreibung auf Seite 88

Vector-Grafik-Library

Nutzen Sie die ganze Faszination der Vektor-Grafik. Mit dieser Bibliothek bauen Sie Vector-Grafiken auf einfache und schnelle Weise in Ihre Programme ein. Der Vorteil: Es wird nur wenig Speicherplatz und Rechenzeit verbraucht.

Beschreibung auf Seite 22ff.

Zwei Disketten

Bestell-Nr. 45907

DM 34,90*

sFr 29,50*/öS 349,-*


Markt&Technik
Zeitschriften · Bücher
Software · Schulung

**Weitere Angebote
auf der Rückseite!**

AMIGA PROGRAMMSERVICE

Sie suchen hilfreiche Utilities und professionelle Anwendungen für Ihren Computer? Sie wünschen sich gute Software zu vernünftigen Preisen? Hier finden Sie beides! Unser stetig wachsendes Sortiment enthält interessante Listing-Software für alle gängigen Computertypen. Jeden Monat erweitert sich unser aktuelles Angebot um eine weitere interessante Programmsammlung für jeweils einen Computertyp. Bei Fragen zu Bestellung und Versand der Programmservice-Disketten wählen Sie bitte:
Telefon (089) 46 13-232.

Bestellungen bitte nur gegen Vorankasse an: Markt & Technik Verlag AG, Buch- und Software-Verlag, Hans-Pinsel-Straße 2, D-8013 Haar, Telefon (089) 46 13-0. Schweiz: Markt & Technik Vertriebs AG, Kollerstrasse 37, CH-6300 Zug, Telefon (042) 440 550. Österreich: Markt & Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 587 1393-0; Microcomput-ique, E. Schiller, Göglstraße 17, A-3500 Krems, Telefon (02732) 74193; MES-Versand, Postfach 15, A-3485 Haitzendorf; Bücherzentrum Meidling, Schönbrunner Straße 261, A-1120 Wien, Telefon (0222) 833196. Bestellungen aus anderen Ländern bitte nur schriftlich an: Markt & Technik Verlag AG, Abt. Buchvertrieb, Hans-Pinsel-Straße 2, D-8013 Haar. Nur gegen Bezahlung der Rechnung im voraus.

Bitte verwenden Sie für Ihre Bestellung und Überweisung die beige-heftete Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungsscheck mit Ihrer Bestellung. Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

Amiga-Sonderheft 4: C und Assembler

Raytracer: berechnet fantastische 3-D-Bilder. Die Grafikfähigkeiten des Amiga werden eindrucksvoll genutzt.

Sound-Effekte-Editor: Zeichnen Sie beliebige Geräusche mit der Maus. Der Clou: Die Sounds werden als C-Sourcecode gespeichert und sind leicht in eigene Programme zu übernehmen.

Programmierertools: Jeweils eine mausgesteuerte Benutzeroberfläche für Lattice- und Aztec-Compiler. Programmieren wird ebenso komfortabel wie das Arbeiten mit der Workbench. Eine Shell hilft bei der Vereinfachung umständlicher CLI-Operationen.

Weiterhin befinden sich auf der Diskette alle Programme komplett mit den jeweiligen Quelltexten, die im Inhaltsverzeichnis des Amiga-Sonderhefts 4 mit einem Diskettensymbol gekennzeichnet sind.

3 1/2"-Diskette für Amiga

Bestell-Nr. 45904 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga-Sonderheft 3: Basic, Spiele

Broker: Erleben Sie die Faszination der Börse hautnah. Diese Simulation für 2 bis 4 Spieler ist einzigartig. Der Autor setzt seine fundierten Kenntnisse in spannendes Spielgeschehen um.

Ping-Pong: Dieses Sportspiel bringt Wettkampfstimmung ins Wohnzimmer. Dreidimensionale Darstellung, realistische Soundeffekte und rasante Ballwechsel führen zu lang anhaltendem Spielspaß.

Anpfiff: Als Manager in der Fußball-Bundesliga führen Sie Ihr Lieblingsteam durch die Saison. Zusätzliche Spiele im UEFA-Cup verhelfen Ihrer Mannschaft zu Ruhm und Ihnen zu vielen Manager-Punkten.

Basic-Routinen: Die Basic-Kurse im Sonderheft 3 bieten zahlreiche, hilfreiche Routinen. Alle dort vorgestellten Programme finden Sie auch auf dieser Programmservice-Diskette.

Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des Amiga-Sonderhefts 3 mit einem Diskettensymbol gekennzeichnet sind.

3 1/2"-Diskette für Amiga

Bestell-Nr. 45903 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga-Sonderheft 2: Grafik, Anwendung

Objekt-Editor: Animierte Figuren, beispielsweise für eigene Spiele, entwickeln Sie mit diesem Editor auf komfortable Weise. Sogar mit Deluxe Paint erstellte Pinsel lassen sich einlesen.

Haushaltsbuch: Mit diesem hervorragenden Anwendungsprogramm verwalten Sie alle Einnahmen und Ausgaben auf übersichtliche Weise. Eine Monats- oder Jahresstatistik zeigt, in welchen Bereichen Sie zukünftig sparen können. Jetzt haben Sie Ihre Finanzen im Griff.

Keyboard-Master: Lernen Sie im Zehn-Finger-System zu tippen. Mit diesem didaktisch ausgereiften Programm ist dies kein Problem. Für Programmierer sind sogar Speziallektionen mit wichtigen Sonderzeichen vorhanden.

FastLoadCopy: Dieses Tool bringt den DIR-Befehl auf Trab. Nach der »Operation« wird das Inhaltsverzeichnis einer Diskette im D-Zug-Tempo eingelesen. Zusätzlich kopiert das Programm Disketten und versieht diese mit dem schnellen Directory.

Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des Amiga-Sonderhefts 2 mit einem Diskettensymbol gekennzeichnet sind.

3 1/2"-Diskette für Amiga

Bestell-Nr. 45802 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga-Sonderheft 6:

CONTEXT: Maßgeschneidert für Ihre Texte. Traumhafte Geschwindigkeit dank Systemroutinen, einfache Bedienung und sehr gut strukturierte Bedienoberfläche. Das sind

Merkmale von Context – die Textverarbeitung nach Maß. Beschreibung auf Seite 90ff.

Kontenführung: Keine Ahnung, warum Sie noch immer nicht Millionär sind? Dann brauchen Sie das Programm Kontenführung! Beschreibung auf Seite 85ff.

AMSpool: Machen Sie ihrem Drucker Beine! Beschreibung auf Seite 137ff.

Waste Hunter: Dieser Packer schafft Platz auf Ihren Disketten. Beschreibung auf Seite 140ff.

Weiterhin befinden sich auf der Diskette alle Programme, die im Inhaltsverzeichnis des Amiga-Sonderhefts 6 mit einem Diskettensymbol gekennzeichnet sind.

Zwei Disketten

Bestell-Nr. 45906 **DM 34,90*** sFr 29,50*/öS 349,-*

Die Wiederbelebung für die C64-Peripherie

Viele Amiga-Besitzer haben noch einen C64 mit Peripheriegeräten zu Hause stehen. Mit ein bißchen Hard- und Software können Sie diese zu neuem Leben erwecken und Ihre Daten so weiterbenutzen. Dabei ist die Bedienung wirklich einfach.

Der fertig aufgebaute IEC-Handler erlaubt es, alle C64-Geräte wie die Floppy 1541 oder 1571, Commodore-MPS-Drucker und natürlich auch den C64 (zur Datenübertragung) am Amiga zu betreiben.

Das Gesamtpaket besteht aus der fertig aufgebauten Platine mit Verbindungskabel, der Treibersoftware auf 3 1/2"-Diskette sowie einer entsprechenden Dokumentation.

Bestell-Nr. 39101

DM 79,-* sFr 71,-*/öS 790,-*

Amiga 3/88 Bildschirmfüllende Boot-Bilder mit allen Extras

BootGirl: Fantastische Bilder sofort nach dem Reset. Bis zu 32 Farben mit Color-Cycling. Die Bilder können auch bildschirmfüllend ohne Rand sein. Ein absolutes Muß für jeden Amiga-Besitzer.

CassCover: Selbstgedruckte Kassettenhüllen geben Ihnen den richtigen Überblick. Einfache Bedienung macht das Eingeben und Ausdrucken zur wahren Freude.

Command: Das Programm ermöglicht die Steuerung des Aztec-C-Compilers mit der Maus. Keine langen Eingaben per Tastatur, sondern ein einziger Mausklick startet nun die Übersetzung.

VideoText: Ein unentbehrliches Werkzeug für alle Video-Fans, die ihren eigenen Vorspann mit dem Amiga generieren wollen. Laufbänder, verschiedene Schriften und IFF-Bilder sind nur einige Stichpunkte, die das Programm so interessant machen.

3 1/2"-Diskette für Amiga

Bestell-Nr. 48803 **DM 29,90*** sFr 24,90*/öS 299,-*

* Unverbindliche Preisempfehlung

Wichtig: Mit den Gutscheinen aus dem »Super-Software-Scheckheft« zu DM 149,- können Sie Software-Disketten Ihrer Wahl aus dem Preiswerte Super-Software-Angebot im Wert von DM 180,- bestellen – egal, ob diese DM 19,90, DM 29,90 oder DM 89,- kosten. Sie sparen DM 30,-! Das Super-Software-Angebot finden Sie in den Zeitschriften **Computer Persönlich, PC Magazin Plus, Amiga-Magazin, Amiga-Sonderheft, 64'er-Magazin, 64'er-Sonderheft, ST-Magazin, PC Magazin, Happy-Computer.**

Übrigens: Die Gutscheine können Sie auch übertragen oder verschenken!

Das Scheckheft können Sie per Verrechnungsscheck oder mit der eingeklebten Zahlkarte direkt beim Verlag bestellen. Kennwort: »Super-Software-Scheckheft«, Bestell-Nr. W156

P.S.: Bei Bestellungen bitte kein Bargeld einschicken!

Bei While- und Repeat-Schleifen beschränken sich die Unterschiede auf die eben genannten. Auf eine genauere Behandlung kann daher an dieser Stelle verzichtet werden.

Bitgefummel

In C-Listings werden Sie oftmals Makros mit der Pre-Processor-Anweisung »#define« finden, die ungefähr folgendes Aussehen haben:

```
#define Mal2(x)
(x < 1)
```

Die C-Befehle »>>« und »<<« werden in Modula mit der Standard-Funktion SHIFT() aus dem Modul SYSTEM implementiert, die zwei Übergabe-Parameter verlangt:

Als ersten Wert übergeben Sie die Variable, die bitweise verschoben (»geSHIFTet«) werden soll (der Typ ist dabei unbedeutend, Records sind jedoch nicht erlaubt). Der zweite Wert ist ein Longint-Wert. Er gibt die Anzahl Bits an, um die die übergebene Variable verschoben werden soll. Ist dieser Wert positiv, so wird nach links (»<<«) verschoben, während ein negativer Wert nach rechts (»>>«) verschiebt.

Sind die bisher genannten »kleinen Unterschiede« noch relativ einfach von der einen in die andere Programmiersprache zu übertragen, so gibt es doch auch Besonderheiten. Werden diese »Spezialitäten« verwendet, so ist meist keine direkte Übersetzung möglich. Es führt kein Weg daran vorbei, die Gedanken des Programmierers (in Form der Programmstruktur) nachzuvollziehen und in eigene Ideen umzusetzen. Im folgenden finden Sie die wichtigsten Eigenheiten kurz erklärt.

In Modula gibt es das Konzept der »TermProcedure«. Es handelt sich dabei um eine Prozedur, die automatisch am Ende des Programmes oder auch bei einem unvorhergesehenen Ende (beispielsweise beim Auftreten eines Fehler) aufgerufen wird. Sinnvoll ist es vor allem, hier eine Prozedur anzugeben, die eventuell geöffnete Screens und Windows aufräumt und alle Ressourcen wieder freigibt. Diesen Service sucht der C-Programmierer vergeblich!

Die einzige Chance, dieses Konzept wenigstens annähernd nach C zu übertragen, ist, die Prozedur als eine normale C-Routine zu deklarieren, und in jedem nur erdenklichen

Falle eines möglichen Fehlers anzuspringen. Eine TermProcedure sollte aber so »intelligent« sein, daß sie zuerst abfragt, ob ein Screen offen ist, oder Speicher schon belegt war, bevor sie diesen freigibt.

```
if !(MyScreen == NULL)
  CloseScreen(MyScreen);
```

Der komplette Ablauf in Modula:

```
PROCEDURE MyCleanUp();
BEGIN
  IF NOT (MyScreen =
  NIL) THEN CloseScreen
  (MyScreen); END;
END MyCleanUp;
```

```
BEGIN (* Hauptprogramm *)
  TermProcedure
  (MyCleanUp);
  ...
  <Screen öffnen>;
  ...
  (*Programmende, ohne
  den Screen zu schließen
  (übernimmt MyCleanUp)
  *)
END .
```

Wenn aber eine TermProcedure für andere Zwecke als zum »Aufräumen« benutzt wird, kann es sehr schwierig werden. Da hilft nur die eigene Kreativität, es gibt keine Tips!

Hingegen kann es einem Modula-Programmierer in einem C-Programm passieren, daß er vor der Deklaration einer Variablen das Schlüsselwort »register« findet.

Dies weist den C-Compiler an, eine Variable nach Möglichkeit in ein Register zu legen, was die Ablaufgeschwindigkeit des Programms erheblich erhöhen kann. In Modula besteht keine Chance, denselben Effekt zu erreichen!

In C kann ein String nicht einfach einem anderen zugewiesen werden. Er muß Buchstabe für Buchstabe »kopiert« werden. Dafür existieren Routinen, die dem Programmierer diese Arbeit abnehmen (»strcpy«). In Modula ist das unter genau definierten Umständen möglich (gleiche Länge und gleicher Indexbereich). Beispielsweise ist

```
MeinName := "Frank
  Staudte";
```

in Modula durchaus zulässig. Sie sollten aber auch in Modula besser die Routinen aus dem Modul Strings benutzen, um ein sicheres Gelingen zu garantieren.

In C können Sie Feld-Variablen benutzen, um Speicher beispielsweise für einen Puffer zu reservieren. Dies ist in Modula wegen der strengen Typenkontrolle nicht möglich. Es

bleibt Ihnen kein anderer Weg, als den Speicher mit einer der Alloc-Funktionen anzufordern (aus den Modulen Heap oder Exec). Als Größe des zu allozierenden Speichers geben Sie die Größe der vorgesehenen Variable an (beispielsweise derselben Variable, die in C benutzt wird):

```
Buffer := SIZE(Typ);
```

Modula kennt das Konzept der »gebundenen« Parameter bei der Variablenübergabe an eine Prozedur (sogenannte »VAR«-Parameter). Das bedeutet, daß der Prozedur die »Erlaubnis« erteilt wird, den entsprechenden Wert im Hauptprogramm zu verändern. Wird also eine Variable global definiert, und dann einer Prozedur als VAR-Parameter übergeben, so ändert sich bei jeder Veränderung innerhalb der Prozedur auch der globale Wert.

Dies ist in C nicht so einfach möglich, da C ausschließlich Werteparameter kennt. Will man das gleiche Konzept in C realisieren, so muß der übergebene Wert als Zeiger (Pointer) auf seinen Typ deklariert werden (sowohl lokal, als auch in der Routine!). Dann wird in der Prozedur mittels Dereferenzierung (siehe oben) auf die eigentliche Variable zugegriffen und deren Wert geändert.

Wie Sie sicherlich bemerkt haben, ist Modula für den Leser eines Programmes etwas leichter verständlich, während in C der kürzere Tippweg möglich ist. Wir möchten jedem C-Programmierer ans Herz legen, auf Statements wie »x += 1;« zu verzichten, falls er Wert darauf legt, daß irgendjemand seinen Source-Code entziffern kann.

Jedem interessierten Programmierer sei an dieser Stelle nahegelegt, sich ein Buch mit einer kompletten Sprachdefinition der jeweils »fremden« Sprache zuzulegen. Diese sind relativ günstig, und enthalten die wirklich benötigten Informationen.

Natürlich wird die Konvertierung zwischen verschiedenen Programmiersprachen auch mit den hier genannten Tips noch nicht zum »Spaziergang«. Wir hoffen aber, Ihnen damit beim Verständnis von Fremdsprachen etwas geholfen zu haben. so

Frank Staudte ist einer der Gründer des Amiga Modula Klub Stuttgart (AMOK). Sie können ihn über unsere Verlagsanschrift erreichen (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Herrn Frank Staudte, Hans-Pinsel-Str. 2, 8013 Haar).

```
MODULE OpenMe;

FROM SYSTEM IMPORT ADR;
FROM Dos IMPORT Open, oldFile, FileHandlePtr;

VAR
  MyFile: FileHandlePtr;

BEGIN
  MyFile := Open(ADR("SYS:S/Startup-Sequence"),oldFile);
END OpenMe.

/* Dieses Programm soll die DOS-Routine Open() aufrufen */
/* Dies ist ein Demo, kein lauffähiges Programm */

#include <exec/types.h>
#include <dos/dos.h>
#include <dos/dosextens.h>

struct DosBase *DosBase;

main()
{
  struct FileHandle *MyFile;

  DosBase = (struct DosBase *) OpenLibrary(DOSNAME,0);
  if (DosBase == NULL) exit();

  MyFile = Open("SYS:S/Startup-Sequence",MODE_OLDFILE);
}
```

Listing 2. Das Ansprechen der Systembibliotheken sieht in beiden Sprachen sehr ähnlich aus

Allen Programmiersprachen gemeinsam sind gewisse Programmstrukturen, die es erlauben, einen das gestellte Problem lösenden Algorithmus unabhängig von der in der Implementation gewählten Sprache zu formulieren. Die einfachste dieser Strukturen ist die Wertzuweisung (Variable a soll den Wert b erhalten). Daneben gibt es einfache Verzweigungen (meist an Schlüsselwörtern wie »IF« und »ELSE« zu erkennen), abweisende und nicht abweisende Schleifen (»REPEAT«, »WHILE«, »FOR« oder »LOOP«) und Fallunterscheidungen (»SWITCH« oder »CASE«).

Leider sind nicht alle dieser Strukturen in allen Programmiersprachen durch explizite Befehle implementiert, sondern es wird manchmal nötig, diese Strukturen durch Kombination anderer Befehle nachzubilden. Als Beispiel hierfür kann die nicht-abweisende Schleife dienen (siehe Tabelle 1 Punkt 3), die zwar in C vorhanden ist, in Basic aber mit Hilfe eines Labels und einem bedingten Rücksprung simuliert werden muß. Auch wenn sich bei der Umsetzung dieser, den Programmfluß steuernden Strukturen keine großen Schwierigkeiten ergeben, sind doch einige Fallstricke zu überspringen. Dies liegt im wesentlichen daran, daß die Programmstrukturen in C etwas flexibler sind als die Basic-Strukturen. Dies trifft sowohl für die FOR-Schleife (kann in C mehrere Abbruchbedingungen enthalten), als auch auf die Fallunterscheidung zu. Für letztere gibt es zwei Möglichkeiten.

Zum einen die sowohl in Basic als auch die in C vorhandene IF-ELSEIF-Kette. Zum anderen gibt es in C noch die »richtige« Fallunterscheidung mit SWITCH. Diese läßt sich in Basic mit Hilfe einiger Labels einfach mit ON n GOTO (bzw. ON n GOSUB) nachbilden. Dabei ist jedoch zu beachten, daß im Fall von Basic n nur eine lückenlose Folge von Werten 0,1,2,...N sein darf, während in C n einen als Fall definierten beliebigen konstanten Ausdruck annehmen darf.

Alles in allem läßt sich sagen, daß bei der Umsetzung von C-Listings in Basic-Listings (und umgekehrt) bei der Übertragung der Programmstruktur wohl die geringsten Schwierig-

keiten auftreten dürften – sofern sich ein Basic-Listing nicht durch wilde GOTO-Sprünge über die Regeln sinnvollen Programmierens hinwegsetzt.

Etwas subtiler sind die Schwierigkeiten, die bei Unterprogrammen und Funktionen auftreten. Dies liegt zum einen daran, daß C zwar Funktionen, aber keine Unterprogramme kennt, während in Basic der Fall genau umgekehrt liegt (so man die selten brauchbaren Einzeilen-Funktionen vernachlässigt). Zum anderen werden auch die Parameter in unterschiedlicher Art übergeben: C übergibt an seine Funktionen den Wert der jeweiligen Variable, während in Basic die Adresse übergeben wird, an der sich der Wert befindet (Übergabe durch Referenz). Diese unterschiedlichen Übergabemechanismen haben weitreichende Folgen. Weist man in einem Basic-Unterprogramm einer übergebenen Variablen einen

Parameter, der das Ergebnis übernehmen kann. Beim Aufruf des Unterprogramms sollte man durch Setzen von Klammern sicherstellen, daß die Parameter an eine temporäre Variable zugewiesen werden, da so die Werte der Parameter im aufrufenden Programm nicht vom Unterprogramm verändert werden können. Nicht so einfach ist die Situation bei GOSUB-Unterprogrammen in Basic, da

Basic-Programmen die gleichen Betriebssystem-Routinen zu nutzen, statt der eingebauten WINDOW- und MENU-Befehle. Umgekehrt wird man bei der Umwandlung von Basic nach C allerdings nicht umhin können, sich für die Behandlung von Windows und Menüs eigene Funktionen zu schreiben. Auch die in Basic vorhandene Unterbrechungsverarbeitung sollte man sich in C am besten mit Hilfe der Routinen der Amiga-Libraries erarbeiten.

Die größte Schwierigkeit bei der Umwandlung von C nach Basic haben wir uns bis zum Schluß aufgehoben. Der Datentyp »struct« ist in Basic leider vollkommen unbekannt, während er in für den C-Programmierer geradezu unverzichtbar ist. Doch mit Hilfe einiger kurzer Unterprogramme und globalen Variablen läßt sich auch diese Klippe umschieben. Dabei sollten Sie jedoch auf folgende Eigenheit achten:

In vielen Strukturen benutzt man Zeiger, die entweder auf Strings deuten oder Strukturen miteinander verknüpfen (verkettete Listen). Derartige Zeiger können in Basic nicht ohne eine gehörige Portion Vorsicht implementiert werden, da sich die Speicheradressen aller Variablen in Basic während der Laufzeit ändern können. Dies hat natürlich zur Folge, daß ein einmal gültiger Zeiger auf eine Zeichenkette nach einer derartigen Verschiebung auf alles mögliche zeigt, nur nicht mehr auf die ursprüngliche Zeichenkette. Als Lösung bietet sich an, die Strukturen nicht im normalen, von Basic benutzten Speicherbereich abzulegen, sondern hierfür den Systemspeicher zu benutzen. Bild 1 verdeutlicht den Aufbau von Strukturen.

Dabei sollten Sie natürlich bedenken, diesen Speicherplatz nach der Programmausführung wieder an das System zurückzugeben. Um uns die

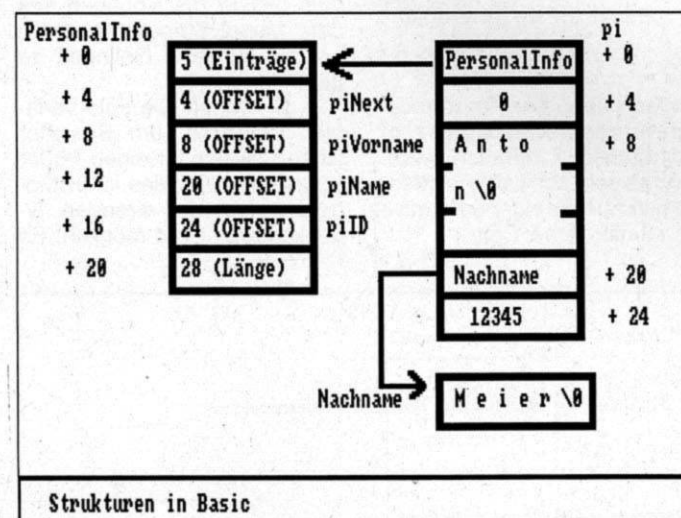


Bild 1. Der Aufbau von Strukturen in Basic

neuen Wert zu, so ändert sich dieser Wert auch im Hauptprogramm. In einer C-Funktion bleibt dagegen der Wert der Variablen im Hauptprogramm unverändert.

Daher ist es ratsam, bei der Übersetzung eines Basic-Unterprogramms in C die Parameter der C-Funktion als Zeiger auf die jeweiligen Variablen zu deklarieren und beim Funktionsaufruf dann die Adresse dieser Variablen zu übergeben. Setzt man umgekehrt eine C-Funktion in ein Basic-Unterprogramm um, so muß man dafür sorgen, daß der Funktionswert auch zurück an das aufrufende Programm gelangt. Daher benötigt das Basic-Unterprogramm einen zusätzlichen Pa-

rameter, der das Ergebnis übernehmen kann. Beim Aufruf des Unterprogramms sollte man durch Setzen von Klammern sicherstellen, daß die Parameter an eine temporäre Variable zugewiesen werden, da so die Werte der Parameter im aufrufenden Programm nicht vom Unterprogramm verändert werden können. Nicht so einfach ist die Situation bei GOSUB-Unterprogrammen in Basic, da

Windows und Menüs

Werden in C-Programmen eigene Windows oder Menüs benutzt, so ist es häufig einfacher, auch in den entsprechenden

Ungleich

Ein großer Teil der im AMIG Listings ist in den Sprachen Basic und ein Listing der einen Sprach

chen C und Basic

e Brüder

Magazin veröffentlichten
geschrieben. Wie geht man vor, um
n die andere umzuwandeln?

Verwaltungsarbeit zu vereinfachen, benutzen wir zur Beschaffung von Speicher am besten die Funktion »AllocRemember«, die nicht nur Speicher allokiert, sondern auch noch eine verkettete Liste des belegten Speichers unterhält. Damit kann der gesamte belegte Speicher am Programmende mit einem einzigen Aufruf der Systemroutine »FreeRemember« wieder freigegeben werden. Zu dieser Verwaltungsarbeit wird die erste globale Variable »REMEMBER.KEY« benötigt. Wir haben für Sie eine kleine Sammlung von Unterprogrammen vorbereitet, die einige Eigenheiten von C in Basic implementieren (Listing 1). Damit umgehen Sie viele Widerwärtigkeiten beim Konvertieren. Sie sollten aber die folgenden Hinweise genau beachten.

Definition

Bei den Strukturen unterscheiden wir zwei Fälle: Zum einen muß definiert werden, wie die Struktur aussieht, zum anderen müssen dann auch »Verkörperungen« dieser Struktur implementiert werden. Bei der Definition müssen wir uns merken, welche »Mitglieder« die Struktur besitzt und wo diese innerhalb der Struktur zu finden sind. Dazu dient das globale Feld »STRUCT.MEMBERS«, das den jeweiligen Abstand eines Mitglieds vom Beginn der Struktur aus enthält. Das Unterprogramm »DEFSTRUCT« setzt die jeweiligen globalen Variablen auf den Anfangszustand. Die Unterprogramme »CHAR«, »VOID.PTR« etc. sorgen dafür, daß die jeweiligen Abstände vom Strukturstart eingetragen werden. Das Unterprogramm »ENDSTRUCT« überträgt diese Liste in den Systemspeicher, damit diese Informationen den späteren Verkörperungen der Struktur zur Verfügung stehen. Die globale Variable »STRUCT.STATUS« steuert das Verhalten

der Unterprogramme CHAR, VOID.PTR etc. wie folgt:

Werden diese Unterprogramme innerhalb einer Strukturdefinition aufgerufen, so tragen sie die jeweilige Position in die entsprechende Strukturliste ein. Werden sie dagegen außerhalb einer Strukturdefinition aufgerufen, so reservieren sie den benötigten Speicherplatz für den jeweiligen Eintrag. Um von einer Strukturimplementierung aus die zugehörige Strukturdefinition zu finden, ist der erste Eintrag einer Implementierung immer ein Zeiger auf die passende Definition. Die mit diesen Unterprogrammen implementierten Strukturen kommen von der Syntax her denen in C schon sehr nahe. Weniger erquicklich ist die Situation jedoch bei den Wertzuweisungen. Auch diese können nur mit Hilfe von Unterprogrammen erfolgen, wobei auch noch unterschieden werden muß, ob Werte von einer Variablen an eine Struktur übergeben werden sollen (ASSIGN.TO) oder umgekehrt (ASSIGN.FROM). Bei diesen Zuweisungen ist, wie auch bei den anderen Struktur-Unterprogrammen, zu beachten, daß keinerlei Fehlerüberprüfungen unternommen werden, da diese den Programmumfang doch erheblich hätten ansteigen lassen. Die beiden Unterprogramme CSTRING und BSTRING dienen nur dazu, Basic-Zeichenketten in C-Zeichenketten (diese zeichnen sich durch eine abschließende »0« aus) - und umgekehrt - umzuwandeln, da alle Zeichenketten innerhalb von Strukturen den C-Konventionen folgen.

Mit diesem Rüstzeug versehen, dürfte es keine großen Schwierigkeiten mehr machen, die gängigen C-Listings in Basic-Programme umzuwandeln. Etwas schwieriger ist allerdings der umgekehrte Fall, da Unterbrechungsverarbeitung und andere abstrakte Konzepte in Basic eingebaut sind,

Amiga-Basic

C (ANSI)

1. Einfache Verzweigung

```
IF (Bed) THEN
(Fall 1)
ELSE
(Fall 2)
END IF
```

```
if (Bed) {
(Fall 1)
} else {
(Fall 2)
}
```

2. abweisende Schleife

```
FOR I = 1 TO M STEP S
....
NEXT I
```

```
for( i = 0; i < M; i += s ) {
....
}
```

```
WHILE (Bed)
....
WEND
```

```
while (Bed) {
....
}
```

3. nicht-abweisende Schleife

```
REPEAT:
....
IF (Bed) GOTO REPEAT
```

```
do {
....
} while (Bed);
```

4. Fallunterscheidung

```
IF (Bed1) THEN
(Fall 1)
ELSEIF (Bed2)
(Fall 2)
....
ELSEIF (BedN)
(Fall N)
ELSE
(sonst)
END IF
```

```
if (Bed1) {
(Fall 1)
} else if (Bed2) {
(Fall 2)
....
} else if (BedN) {
(Fall N)
} else {
(sonst)
}
```

```
ON N GOTO
CASE1,CASE2,...,CASEN
DEFAULT:
GOTO ENDCASE
CASE1: ....
GOTO ENDCASE
....
CASEN: ....
ENDCASE:
```

```
switch (n) {
case 1 : ....
break;
case 2 : ....
break;
....
default : ....
}
```

5. Implementation von Unterprogrammen und Funktionen

```
SUB UPROG(I%,L&,F!,D#)
STATIC
....
END SUB
```

```
void uprog(short *i, long *l,
float *f, double *d) {
....
}
```

```
SUB FCT(ERG#, D#, L&)
STATIC
....
ERG# = ....
END SUB
```

```
double erg;
double fct( double d, long l) {
....
return ....;
}
```

6. Aufruf von Unterprogrammen und Funktionen

```
CALL UPROG(I%,L&,F!,D#)
```

```
uprog(&i, &l, &f, &d);
```

```
CALL FCT(ERG#, (D#), (L&))
```

```
erg = fct( d, l);
```

Tabelle 1. Die wichtigsten Elemente von C und Basic gegenübergestellt

während sie in C durch Rückgriff auf die Systemroutinen »nachprogrammiert« werden. Da jedoch jeder Besitzer des Amiga über Amiga-Basic, nicht

unbedingt jedoch über einen C-Compiler verfügt, ist die Umwandlung von C nach Basic sicherlich von größerem Interesse als der umgekehrte Fall. so


```

ON BREAK GOSUB BreakHit
' ON ERROR GOTO ReleaseMem

' Externe Funktionen
LIBRARY "exec.library"
LIBRARY "intuition.library"
DECLARE FUNCTION AllocRemember&( RememberPtr&, size&, Flags&) LIBRARY

' Voreinstellungen
OPTION BASE 0
DEFBNG a-z

DIM SHARED SIZEOFLONG, MAX.MEMBERS%
' Konstanten
SIZEOFLONG = 4&
MAX.MEMBERS% = 20

' globale Variable
DIM SHARED STRUCT.MEMBERS(MAX.MEMBERS%)
DIM SHARED REMEMBER.KEY, STRUCT.LENGTH, STRUCT.STATUS, MEMBER.NO
STRUCT.LENGTH = 0&
STRUCT.STATUS = 0& ' <> 0, falls innerhalb von Struktur-Definition
MEMBER.NO = 0&
REMEMBER.KEY = 0&

' Unterprogramme zum Umgang mit Strukturen
SUB defstruct STATIC
    STRUCT.STATUS = 1&
    STRUCT.LENGTH = SIZEOFLONG ' Erster Eintrag ist Zeiger auf Struktur-Definition
    MEMBER.NO = 1&
    FOR i% = 0 TO MAX.MEMBERS%
        STRUCT.MEMBERS(i%) = 0&
    NEXT i%
END SUB

SUB endstruct( struct.name ) STATIC
    STRUCT.STATUS = 0&
    STRUCT.MEMBERS(0) = MEMBER.NO
    STRUCT.MEMBERS(MEMBER.NO) = STRUCT.LENGTH
    bytes = (MEMBER.NO+1) * SIZEOFLONG
    struct.name = AllocRemember&(VARPTR(REMEMBER.KEY), bytes, 0)
    CopyMem VARPTR(STRUCT.MEMBERS(0)), struct.name, bytes
END SUB

SUB allot( instance, size ) STATIC
    IF ( STRUCT.STATUS = 0 ) THEN
        instance = AllocRemember&(VARPTR(REMEMBER.KEY), size, 0)
    ELSE
        STRUCT.MEMBERS(MEMBER.NO) = STRUCT.LENGTH
        STRUCT.LENGTH = STRUCT.LENGTH + size
        instance = MEMBER.NO
        MEMBER.NO = MEMBER.NO + 1
    END IF
END SUB

SUB struct( struct.name, instance ) STATIC
    size = CLNG(PEEK(STRUCT.name + PEEK(STRUCT.name) * SIZEOFLONG))
    allot instance, size
    IF (STRUCT.STATUS = 0) THEN
        POKEL instance, struct.name
    END IF
END SUB

SUB void.ptr( instance ) STATIC
    allot instance, 4&
END SUB

SUB short.ptr( instance ) STATIC
    allot instance, 4&
END SUB

SUB long.ptr( instance ) STATIC
    allot instance, 4&
END SUB

SUB float.ptr( instance ) STATIC
    allot instance, 4&
END SUB

SUB double.ptr( instance ) STATIC

```

```

        allot instance, 4&
    END SUB

SUB char.ptr( instance ) STATIC
    allot instance, 4&
END SUB

SUB short( instance ) STATIC
    allot instance, 2&
END SUB

SUB long( instance ) STATIC
    allot instance, 4&
END SUB

SUB float( instance ) STATIC
    allot instance, 4&
END SUB

SUB double( instance ) STATIC
    allot instance, 8&
END SUB

SUB char( instance, bytes ) STATIC
    allot instance, bytes
END SUB

SUB assign.to( instance, member, value.ptr ) STATIC
    struct.descr = PEEKL(instance)
    last.member = PEEKL(struct.descr)
    member.offset = struct.descr + member * SIZEOFLONG
    IF member = 0 THEN
        length = PEEKL(struct.descr + last.member * SIZEOFLONG)
        CopyMem value.ptr, instance, length
    ELSEIF member < last.member THEN
        offset = PEEKL(member.offset)
        length = PEEKL(member.offset+SIZEOFLONG) - offset
        CopyMem value.ptr, instance+offset, length
    END IF
END SUB

SUB assign.from( value.ptr, instance, member ) STATIC
    struct.descr = PEEKL(instance)
    last.member = PEEKL(struct.descr)
    member.offset = struct.descr + member * SIZEOFLONG
    IF member < last.member THEN
        offset = PEEKL(member.offset)
        length = PEEKL(member.offset+SIZEOFLONG) - offset
        CopyMem instance+offset, value.ptr, length
    ELSEIF member = 0 THEN
        length = PEEKL(struct.descr + last.member * SIZEOFLONG)
        CopyMem instance, value.ptr, length
    END IF
END SUB

SUB strepy( destptr, sourceptr ) STATIC
    dp = destptr
    sp = sourceptr
    a = PEEK(sp)
    WHILE (a <> 0)
        POKE dp, a
        dp = dp + 1
        sp = sp + 1
        a = PEEK(sp)
    WEND
    POKE dp, a
END SUB

SUB cstring(strptr, a$) STATIC
    IF STRUCT.STATUS = 0 THEN
        size = LEN(a$) + 1
        allot strptr, size
        CopyMem SADD(a$), strptr, size
        POKE strptr+size-1, 0
    END IF
END SUB

SUB bstring(a$, strptr) STATIC
    IF (STRUCT.STATUS = 0) THEN
        i = 0
        WHILE PEEK(strptr+i) <> 0
            i = i + 1

```



```

WEND
b$ = SPACE$(1)
CopyMem strptr, SADD(b$), 1
a$ = b$
END IF
END SUB

SUB freestructs STATIC
FreeRemember VARPTR(REMEMBER.KEY),-1&
END SUB

' Beispielprogramm zum Umgang mit Strukturen in Basic
' Äquivalente C-Formulierung
defstruct
void.ptr piNext
Next ;
char piVorname, 12&
char.ptr piName
long piID
endstruct PersonalInfo

defstruct
struct PersonalInfo, glArbeiter
beiter ;
float glGehalt
endstruct GehaltsListe

struct PersonalInfo, pi
struct GehaltsListe, gl

Nach = 0&
ID = 12345&
Gehalt! = 2937.92
VN$ = "Anton"+CHR$(0)
cstring Nachname, "Meier"

```

```

assign.to pi, piNext, VARPTR(Nach) ' gl.glArbeiter.piNext = NULL ;
assign.to pi, piVorname, SADD(VN$) ' strcpy(gl.glArbeiter.piVorname-
,"Anton") ;
assign.to pi, piName, Nachname ' gl.glArbeiter.piName = "Meier"
;
assign.to pi, piID, VARPTR(ID) ' gl.glArbeiter.piID = 12345 ;
assign.to gl, glGehalt, VARPTR(Gehalt!) ' gl.glGehalt = 2937.92 ;
assign.to gl, glArbeiter, pi
' ....

assign.from pi, gl, glArbeiter
assign.from VARPTR(Nach), pi, piNext
assign.from SADD(VN$), pi, piVorname
assign.from Nachname, pi, piName
assign.from VARPTR(ID), pi, piID
assign.from VARPTR(Gehalt!), gl, glGehalt
bstring NN$, Nachname
bstring VN$, SADD(VN$)
PRINT Nach,VN$,NN$,ID,Gehalt!
FINISHED:
freestructs

END

ReleaseMem:
PRINT "Fehler Nummer:", ERR
RESUME FINISHED

BreakHit:
RETURN FINISHED

```

Listing 1. Mit dieser Sammlung von Unterprogrammen erleichtern Sie sich das Konvertieren von C nach Basic

Quelltextportierung von Aztec nach Lattice

cc2lc - ganz einfach?

Wenn gleich C zur Vereinfachung der Übertragung von Quelltexten entwickelt wurde, können kleinere und größere Unterschiede in den jeweiligen Compilerimplementationen doch Probleme aufwerfen, so daß gerade bei systemnaher Programmierung unter Ausnutzung spezifischer Compilerigenschaften die genaue Kenntnis beider beteiligter Compiler nötig ist.

Dieser Artikel gibt einen Überblick über die bei der Quelltextportierung von Aztec-C auf Lattice-C-5.0 zu erwartenden Probleme. Er kann und soll nicht die Lattice-Compilerdokumentation ersetzen, deren Kenntnis in jedem Falle vorausgesetzt wird.

Die Portierungsrichtung ist übrigens nicht ganz willkürlich gewählt: Da von Manx schon seit einiger Zeit nichts mehr zu hören war, steigen viele Programmierer auf das wesentlich

Die Unterschiede zwischen den beiden Amiga-C-Compilern erschweren die Umsetzung von Quelltexten auf den jeweils anderen Compiler.

Von Ralph Babel

leistungsfähigere Entwicklungspaket von Lattice um, wollen jedoch bereits entwickelte Software nicht zurücklassen. Auch sind viele Aztec-C-Beispielprogramme – etwa dieses Heftes – durchaus für Lattice-Programmierer interessant, bedürfen jedoch vor der fehlerfreien Ausführung einiger kleiner Modifikationen.

Es geht daher nur um die Umsetzung lauffähiger Programme, wobei auch hier Einschränkungen bezüglich des Schwierigkeitsgrades zu machen sind: Ein Arno-Gölzer-Programm ist sicherlich leichter

zu übersetzen als Matt-Dillon-Quelltexte oder mit #asm gespickte Leo-Schwab-Hacks. Die Portierung von Overlay-Programmen (oder die Anwendung der Aztec-Funktionen segload() und freeseg()), Systembibliotheken und Gerätetreibern bleibt wohl erfahrenen Profis vorbehalten.

Die Unterschiede zwischen den beiden Amiga-C-Compilern selbst (im Unterschied zu den im folgenden beispielsweise noch zu besprechenden Bibliotheken) sind weit weniger bedeutend als üblicherweise angenommen, da Aztec eine

Untermenge von Lattice darstellt – die umgekehrte Portierungsrichtung ist sicherlich die schwierigere. Ein typisches Beispiel für Inkompatibilität stellt der Ausdruck

```
a[i] = ++i;
```

dar. Soll hier vor oder nach der Erhöhung von i die Zieladresse berechnet werden? Die Implementatoren von Aztec und Lattice waren sich zumindest in diesem Fall nicht einig, und trotzdem haben beide recht.

Glücklicherweise verlassen sich Programmierer üblicherweise nicht auf die Reihenfolge der Auswertung von Teilausdrücken oder andere Nebeneffekte, da dies allgemein als schlechter Programmierstil angesehen wird.

Andere Unterschiede betreffend Eigenschaften, deren Implementationen normalerweise in den spezifischen Include-Dateien versteckt sind und vom

Programmierer nicht beachtet werden müssen, etwa das offset()-Makro und seine Spezialform **ECOFFSET** aus `<libraries/configregs.h>`.

Lediglich der neue ANSI-Präprozessor von Lattice-C kann zu Problemen führen - etwa bei der Ersetzung von Parametern in Strings - und sollte durch

```
LC -co programmname
```

ausgeschaltet werden.

Include-Dateien

Sofern das Programm nicht zu der seltenen Sorte gehört, die nach der Eingabe von

```
LC -L programmname
```

sofort lauffähig ist, betrifft der erste Schritt bei der Umsetzung meist die Anpassung der Liste verwendeter Include-Dateien. So machen unterschiedliche Verkettungen - viele Include-Dateien ziehen weitere nach sich - unter Lattice-C möglicherweise die explizite Angabe weiterer Include-Dateien notwendig, die von Aztec-C automatisch nachgeladen werden.

So sollte - alphabetische Reihenfolge hin oder her - `<exec/types.h>` immer als erste Datei spezifiziert werden, beispielsweise bei Arno Gölzers Programmen in diesem Heft. Auch die Dateien `<hardware/custom.h>` und `<hardware/dma-bits.h>` fehlen oft und machen sich durch Fehlermeldungen beim Bezeichner custom und solchen Konstanten, die mit **DMA** beginnen, bemerkbar. Im ersten Fall muß neben der Include-Datei zusätzlich noch die Anweisung

```
extern struct Custom _  
far custom;
```

in den Quelltext mitaufgenommen werden, da dies unter Aztec-C automatisch geschieht. Gleiches gilt für die beiden CIA-Symbole »iaa« und »ciab«.

Treten beim anschließenden Binden durch **BLink** nichtauflösende Referenzen mit Namen wie `_OFF_DISPLAY`, `_SetDrPt`, `_SetAfPt` oder `_C_MOVE` auf, so wird außerdem `<graphics/gfxmacors.h>` benötigt.

Da Include-Dateien üblicherweise keinen Code erzeugen, können alle genannten Dateien immer ohne negative Nebeneffekte (von geringfügig längerer Übersetzungszeit einmal abgesehen) eingebunden werden.

Die Aztec-Datei `<functions.h>` enthält die Funktionsdeklarationen für die Routinen der

Amiga-Systembibliotheken und muß durch das Lattice-Äquivalent `<proto/all.h>` ersetzt werden. Dies kann allerdings auch Probleme nach sich ziehen, etwa wenn im Programmcode weitere Funktionsdeklarationen vorkommen oder Basis-symbole für Bibliotheken definiert werden. Programme von Leo Schwab sind ein bekanntes Beispiel hierfür. Hier finden sich Zeilen wie

```
void *IntuitionBase;  
void *GetMsg();
```

Da durch `<proto/all.h>` jedoch

```
extern struct Intuition-  
Base *IntuitionBase;  
struct Message *GetMsg  
(struct MsgPort *);
```

vereinbart wurde, erkennt der Compiler Widersprüche und meldet Fehler. Diese lassen sich dadurch lösen, daß die Vereinbarung für Basissymbole den Informationen in den Proto-Dateien angepaßt und Funktionsdeklarationen ersatzlos entfernt werden.

Bleibt noch ein besonderer Fall, der unter Aztec-C sehr häufig vorkommt: Das Programm scheint gar keine `#include`-Anweisungen zu enthalten. Dies ist auf eine populäre Fähigkeit des Aztec-Compilers zurückzuführen, die das »Vorübersetzen« von Include-Dateien erlaubt. Hier werden zur Verkürzung der Übersetzungszeiten einmalig alle benötigten Include-Dateien übersetzt und später dann nur noch die bei diesem ersten Durchlauf entstandene Symboldatei angegeben. Findet sich im Verzeichnis der zum Programm gehörigen Quelltexte keine Datei, die ausschließlich aus `#include`-Anweisungen besteht, so müssen die notwendigen Include-Dateien manuell ausfindig gemacht werden.

Das Aztec-Präprozessorsymbol `_FUNC` kennt übrigens in Lattice-C keine Entsprechung, wird aber meist ohnehin nur zur Fehlersuche verwendet.

»Sie wurden gewarnt!«

Sollte das Programm nach den genannten Änderungen problemlos durch

```
LC programmname
```

übersetzt werden können (wenn nicht - keine Panik, es kommen noch wesentlich mehr Hinweise), so mag es den ehemaligen Aztec-Programmierer irritieren, daß eine solche Mas-

se an Warnungen auf ihn einprasselt. Lattice-C ist, was die korrekte Verwendung von Datentypen angeht, wesentlich aufmerksamer als die meisten anderen C-Compiler. Viele dieser Warnungen können jedoch mit großer Wahrscheinlichkeit ohne nachteilige Folgen einfach ignoriert werden, da sie aus Zuweisungen (Warnung 30) oder Übergabe (Warnung 88) inkompatibler Zeigertypen resultieren. Unter einem M68000-Prozessor sind diese aufgrund des linearen 32-Bit-Adreßraums jedoch nicht weiter bedeutend und können unter Lattice-C-5.02 zu einem Großteil durch die Option `-cq` unterdrückt werden.

Neben diesen beiden Warnungen (Warning 30: pointers do not point to the same object und Warning 88: argument type incorrect) sind auch

- warning 84: redefinition of pre-processor symbol

- warning 89: constant converted to required type

- warning 93: no reference to identifier

ohne tiefere Bedeutung für die Lauffähigkeit eines Programms.

Da Aztec-C eine fehlende **return**-Anweisung in **int**-Funktionen nicht bemängelt (viele **void**-Funktionen werden - entsprechend dem älteren Kernighan&Ritchie-Standard ([2]) - ohne explizites **void** vereinbart), können natürlich auch hier weitere Warnungen auftreten. Warnungen dieses Typs (Warning 85: function return value mismatch) lassen sich durch die Lattice-Option `-cw` unterdrücken. Durch Übersetzung mit

```
LC -cq -j841891931  
programmname
```

kann man sich daher leichter auf die möglicherweise bedeutenden Meldungen konzentrieren.

Meldung 30 kann übrigens auch als Fehlermeldung (anstelle einer Warnung) auftreten, etwa durch Zuweisung von ganzzahligen Werten ungleich null an Zeigertypen, was unter Aztec-C nicht bemängelt wird. Anweisungen wie

```
struct Custom *p =  
Oxdff000;
```

ändert man daher durch einen expliziten »cast« wie folgt um:

```
struct Custom *p =  
(void *)=xdff000;
```

Auch weitere unbedeutende Warnungen sind möglich.

Durch Zeilen wie

```
LC -cq -j841891931  
programme
```

wurde bereits angedeutet, daß viele Compileigenschaften durch Kommandozeilenoptionen gesteuert werden können. Bekanntestes Beispiel sind wohl die Speichermodelle, die die Adressierung von Code und Daten regeln. Wichtiger ist aber die Datenadressierung, die bei beiden relativ zum Basisregister A4 erfolgt, was alle Daten in einem Segment vorausgesetzt. Mit

```
LC -Lcdn programmname
```

wird aller Code und werden alle Daten in jeweils ein Segment gepackt und sämtliche Debugging-Informationen (auch hierzu später noch mehr) entfernt, so daß ein relativ kurzes ausführbares Programm entsteht. Sollte sich **BLink** trotz der Verwendung dieser Optionen noch mit der Fehlermeldung Error 510: Reference to unmerged data item beschweren, so muß das Programm mit

```
LC -b0 programmname
```

übersetzt werden. Dies bewirkt die absolute Adressierung aller statischer Daten.

Die Datenablage wird durch eine weitere wichtige Option beeinflusst: `-a`. Bekanntlich müssen auf einem Amiga alle Daten, die durch DMA von den Custom-Chips gelesen werden - Grafik- und Audiodaten z.B. - im sog. »Chip-Memory« abgelegt werden. Wird der Speicher durch das Programm angefordert, so stellt dies kein Problem dar, im Unterschied zu statischen Daten, die im resultierenden Lademodul einer speziellen Markierung bedürfen (vergleiche Literaturhinweise [1, Abschnitt 22.3.2]). Was unter Aztec-C durch die Linker-Option `+c` (auch in den Formen `+cd` oder `+cdb`) erreicht wird, erreichen Sie unter Lattice-C durch

```
LC -adb programmname
```

Ist es also zunächst bedeutend, das Programm einfach nur lauffähig zu bekommen (ohne Rücksicht auf die Effizienz), so können Sie ganz plump durch

```
LC -adb -b0 -coqvw -  
j841891931 -r0 -Lcdn  
programmname
```

Ihr Glück versuchen.

Weitere Optionen werden im jeweiligen Zusammenhang erklärt.

Die ersten größeren Unterschiede ergeben sich, wenn nicht nur ganzzahlige, sondern auch Fließkommaabrechnungen durchgeführt werden sollen. Aztec-C verwendet im Default das sog. Fast-Floating-Point-Format (kurz FFP), während Lattice die doppelgenaue IEEE-Arithmetik einsetzt.

Wird die Datei `<math.h>` eingebunden, kommen im Quelltext die Datentypen `float` oder `double` vor oder werden vom Linker diverse `__CX`-Symbole als nicht definiert reklamiert, so sind dies untrügliche Zeichen für Fließkommaoperationen im Quelltext. In diesen Fällen müssen die Include-Dateien `<libraries/mathfp.h>` und `<proto/mathfp.h>` durch `#include` eingelesen und die `mathfp.library` geöffnet werden. Dies erreichen Sie durch

```
struct Library *MathBase; /* global! */
/*...*/
CloseLibrary(MathBase);
```

- natürlich mit den zugehörigen Fehlerabfragen. Das Öffnen sollte während der Initialisierung und das Schließen in der üblichen »Aufräumaroutine« erfolgen. Werden mehr als nur die Grundrechenarten benötigt (etwa `sqrt()`, `sin()` oder `atan()`), so muß auf die gleiche Weise zusätzlich die `mathtrans.library` geöffnet werden. Das zugehörige Basissymbol heißt hier `MathTransBase`.

Die Übersetzung erfolgt dann in beiden Fällen durch

```
LC -ff -lf programmname
```

Durch den Inline-Code (`<proto/mathfp.h>`) werden solche Programme relativ schnell.

Bibliotheken

Wichtiger Teil eines C-Compilers sind die Bibliotheken, die den Code der externen Standardfunktionen enthalten, etwa für I/O oder Speicherverwaltung. Trotz großer Gemeinsamkeiten gibt es doch einige Aztec-Funktionen, die unter Lattice-C nicht oder nur in anderer Form vorhanden sind. Nicht vorhanden sind beispielsweise die Screen-Funktionen, die jedoch durch folgende Präprozessordefinitionen ersetzt werden können:

```
#define scr_beep()
printf("\a")
#define scr_bs()
printf("\b")
#define scr_cdelete()
printf("\233P")
```

```
#define scr_clear()
printf("\f")
#define scr_cr()
printf("\r")
#define scr_curs(row, col)
printf("\233%d;%dH", (short)(row+1), (short)(col+1))
#define scr_cursrt()
printf("\233C")
#define scr_cursup()
printf("\v")
#define scr_eol()
printf("\233;")
#define scr_home()
printf("\233H")
#define scr_ldelete()
printf("\233M")
#define scr_lf()
printf("\204")
#define scr_linsert()
printf("\233L")
#define scr_tab()
printf("\t")
```

Die Debugging-Funktionen zur Ausgabe auf ein serielles Terminal sind für Lattice-C in einer separaten Bibliothek, `<debug.lib>`, enthalten, die explizit beim Linken spezifiziert werden muß:

```
lc -L+LIB:debug.lib
programmname
```

Aztecs `dos-packet()`-Funktion schickt ein Amiga-DOS-Paket an einen Handler-Prozeß und muß unter Lattice-C selbst geschrieben werden, was mit Hilfe beispielsweise von Kapitel 21 aus dem Amiga-Guru-Buch (siehe Literaturhinweise) nicht weiter schwerfallen dürfte.

Nur mit solider Kenntnis des Programms lassen sich die meist sehr system- und compilerabhängigen Anwendungen der Funktionsfamilie »exec()« und »fexec()« ersetzen, die unter Lattice-C nicht vorhanden sind. »fork()« etc. haben entfernte Ähnlichkeiten.

Leichter adaptierbar ist das Durchsuchen eines Verzeichnisses mit der Funktion `scdir()`, die in der Lattice-Bibliothek durch das Funktionspaar `dfind()` und `dnext()` relativ ähnlich implementiert sind, hier jedoch auch Amiga-DOS-Muster erkennt und nicht auf die MS-DOS-Jokerzeichen »*« und »?« beschränkt ist.

Wichtiger Teil der Bibliotheken ist die Art, in der die Programmausführung abgebrochen werden kann. Üblicherweise wird während der I/O-Operationen eine bestimmte Funktion aufgerufen, die sich um die Möglichkeit des Abbruchs kümmert. Unter Aztec ist dies `Chk_Abort()`. Sie liefert

den Wert null zurück, wenn kein Abbruch verlangt wurde. Ist CTRL-C gedrückt worden, wird der Inhalt der Variablen `Enable_Abort` geprüft: Ist er null, so wird die Möglichkeit des Abbruchs unterbunden, und `Chk_Abort()` liefert einen Wert ungleich null zurück. Der Default für `Enable_Abort` ist jedoch ein Wert ungleich null, der im Falle von CTRL-C einen sofortigen Ausstieg durch die Systemfunktion `_abort()` bewirkt, es erfolgt also keine Rückkehr aus `Chk_Abort()`. Will ein Aztec-C-Programmierer die Abbruchmöglichkeit (temporär) unterbinden, so setzt er entweder `Enable_Abort` auf null oder ersetzt die Funktion `Chk_Abort()` durch eine eigene.

Lattice-C verhält sich ähnlich, versteckt jedoch implementationsabhängige Details: Anstelle der Variablen `Enable_Abort` kann der Programmierer durch die Funktion `onbreak()` eine Funktion festlegen, die im Falle eines Abbruchs den Wert null zurückgibt. Der Default für diese »Break-Trap« ist `CXBRK()`, deren Quelltext sich im Verzeichnis »source« auf den Lattice-Disketten befindet. Ist also kein Abbruch erwünscht, so wird dies unter Lattice durch

```
int nobreak(void) {
return 0; }
/* ... */
onbreak (nobreak);
angezeigt.
```

Soll an vom Programmierer bestimmten Stellen eines Programms zusätzlich auf Abbruch geprüft werden (etwa in I/O-losen Phasen), so kann die Funktion `chkabort()` (oder auch `Chk_Abort()`) explizit aufgerufen werden.

Entgegen der Erwartung ist die vom Compiler verwendete int-Größe selten Ursache von Portierungsproblemen, auch wenn in Lattices Default-Modus ints 32 Bits breit sind. Ausgenommen sind hierbei 16-Bit-Aztec-Programme, deren Funktionen von der geringeren Int-Breite abhängt. Beispiele sind hier hartcodierte Objektgrößen (ohne die Verwendung der »sizeof()«-Funktion) oder wenn der »Überlauf« einer int-Variablen (oder der Verlust der oberen Stellen bei der Zuweisung) vorausgesetzt wird. Diese Fälle sind unüblich und mehr von theoretischer Natur, nicht zuletzt, weil viele Aztec-Programmierer aus Sicherheitsgründen mit der Option `+l` (für 32-Bit-Integers) übersetzen und sich so schwer zu entdeckende Bugs ersparen, etwa bei der Pa-

rameterübergabe oder bei der Differenz zweier Zeiger, die bei Aztec-C als int - auch bei 16-Bit-Ints - angesehen wird.

Lediglich wenige Bibliotheksfunktionen stellen den Portierer vor Probleme: So wird unter Aztec-C gelegentlich eine Funktion namens `lmalloc()` verwendet, die bei Verwendung von 32-Bit-Integers natürlich durch `malloc()` ersetzt werden kann.

Besonders unangenehm tritt hier jedoch die Funktion `rand()` hervor, da sie unter Aztec-C unabhängig von der Integer-Größe (also auch bei `cc +l`) Werte im Bereich von 0 bis 32767 zurückgibt - im Gegensatz zu Lattice-C, dessen Wertebereich sich - gemäß ANSI-Norm - von 0 bis `INT_MAX` (siehe `<limits.h>`) erstreckt.

Beim Skalieren von Zufallszahlen gehen Aztec-Programmierer aber meist davon aus, daß 32767 der größtmögliche Ergebniswert ist und erhalten so unter Lattice viel zu große Ergebnisse, die möglicherweise das Überschreiben unschuldigen Speichers verursachen könnten, etwa bei Verwendung als Feldindizes. Folgende Korrektur löst das Problem:

```
#include <math.h>
#define rand() ((rand()
) & 0x7fff)
```

Kaum portierbar sind allerdings die Funktionen `signal()` und `raise()`, die aber ohnehin auf beiden Compilern die ANSI-Konformität vermissen lassen.

Assembler

Durch die von Aztec-C gebotene Möglichkeit zum sog. »Inline-Code« (siehe folgenden Abschnitt) ist es wichtig, die sich auf der Assemblerebene ergebenden Unterschiede zu kennen. Dazu gehören in erster Linie Assemblersyntax und Verwendung der Register durch den Compiler. Ganz ohne Kenntnisse in M68000-Assembler geht es übrigens nicht.

Zur genauen Information über die Schnittstelle für die Kombination von C und Assembler verweisen wir auf die entsprechende Literatur (siehe Hinweise am Ende dieses Artikels). In Kürze: Wird von einem C-Programm eine Funktion (C oder Assembler) aufgerufen, so werden zunächst die Parameter in umgekehrter Reihenfolge auf dem Stapel abgelegt und anschließend ein Unterprogramm zur gewünschten Funktion durchgeführt. Globale Symbole - wie sie Funk-

tionsnamen darstellen - müssen bei der Verwendung in Assembler mit einem führenden Unterstrich versehen werden. Hierzu ein Beispiel:

```
_sqr    move.l    4(sp),d0
        mulu     d0,d0
        rts
```

Zuerst wird der Parameter dieser Funktion (die von C aus mit dem Namen `sqr()` zur Verfügung steht) vom Stapel gelesen. Er befindet sich 4 Byte über der aktuellen Position des Stapelzeigers, da sich darunter noch die Rücksprungadresse (durch den Unterprogrammaufruf dort abgelegt) befindet. Da Lattice-C im Default mit 32-Bit-Integers arbeitet, muß auch ein Langwort (.L) geladen werden. Dieser Wert wird anschließend quadriert, und es erfolgt der Rücksprung zur aufrufenden Routine. Das Ergebnis muß sich bei Rückkehr in D0 befinden, was hier auch ohne explizite MOVE-Anweisung bereits der Fall ist. Wird ein 16-Bit-Integer-Aztec-C-Programm unter Lattice im Default-Modus übersetzt, so muß außerdem unbedingt darauf geachtet werden, daß das höherwertige Wort des Resultats einer Assemblerfunktion auch wirklich gelöscht ist oder daß die Funktion explizit als `short` deklariert wird.

Unterschiede ergeben sich auch bei den freien Prozessorregistern: So müssen unter Aztec-C die Register D4 bis D7 sowie A2 bis A5 gerettet werden, unter Lattice-C jedoch - entsprechend dem Amiga-Standard - zusätzlich die Inhalte von D2, D3 und A6, da diese vom Compiler für die Speicherung von Registervariablen und temporären Ergebnissen verwendet werden.

Die größten Unterschiede bestehen jedoch in den Steueranweisungen der beiden Assembler, insbesondere beim Symbolim- und -export.

So ist unter Lattice-C der OR-Operator nur in Form des Ausdruckszeichens (nicht durch den vertikalen Strich à la C), EXOR- und Modulo-Operator jedoch gar nicht implementiert.

Ebenfalls nicht vorhanden sind die Anweisungen `EQU`, `REG` und `FREG`, die der Definition von Registernamen dienen. Im Falle der Verwendung in einem Aztec-Assemblerprogramm müssen diese Synonyme direkt durch ihre Register ersetzt werden.

Eine `BLANKS`-Anweisung wird von Lattice nicht unter-

stützt, das Operatorfeld darf keine Leerzeichen enthalten. Die Familie der `LIST`-Anweisungen sind unter Lattices ASM durch die Option `-l` implementiert, beeinflussen die Codeerzeugung aber ohnehin nicht und können daher entfallen. Ebenfalls als Kommandozeilenoption (`-m`) steht Azteks `MACHINE`-Anweisung zur Verfügung, Zeilen mit dem Pseudo-Opcode `MC68881` können ebenfalls entfallen. Für den Suchpfad der `INCLUDE`-Anweisung existiert keine Environment-Variablen, er muß explizit als Kommandozeilenoption angegeben werden, etwa durch

```
ASM -iLC:/Assembler_
Headers/ programmname
```

Unter früheren Versionen des Aztec-Assemblers wurden Makroparameter durch `«%1»` bis `«%9»` spezifiziert. Dies muß durch `/1` bis `/9` ersetzt werden.

Die Anweisung `DCB` dient dazu, einen Speicherblock zu reservieren und mit einem konstanten Wert zu füllen, etwa in der Form

```
dc.b    BUFFERSIZE,-1
```

Dies reserviert einen aus `BUFFERSIZE` Langwörtern bestehenden Speicherbereich und füllt jedes Langwort mit dem Wert `-1`. Unter Lattice-ASM können nur mit null gefüllte Speicherbereiche durch `DS` (`define Storage`) reserviert werden. Andere Fälle der `DCB`-Anweisung müssen entweder durch Assemblercode initialisiert oder durch `DC` explizit aufgeführt werden. Der Befehl `EVEN` (zur Ausrichtung auf eine Wortadresse) entspricht unter Lattice `«CNOP 0,2»` oder `«DS.W 0»`.

Die Anweisung `PUBLIC` dient unter Aztecs AS sowohl dem Import als auch dem Export von Symbolen. So muß

```
public _function
durch
```

```
xdef _function
```

ersetzt werden, wenn im gleichen Assemblermodul eine Definition dieses Symbols erfolgt, etwa durch eine Zuweisung oder Verwendung als Sprungmarke. Andernfalls entspricht sie

```
xref _function
```

Auch kann unter AS durch eine pure Referenz - unabhängig davon, ob sie benötigt wird oder nicht - bewirkt werden, daß ein Teil einer Bibliothek eingebunden wird. Dies wird beispielsweise durch die Anweisung `«PUBLIC .begin»` erledigt. Latti-

ces ASM behält nur solche Referenzen im Objektmodul, die auch wirklich verwendet werden.

Die `GLOBAL`-Anweisung dient dazu, globalen Speicher zu reservieren und ähnelt `DS`.

```
_global _var,8
```

wird daher durch

```
_var    ds.b      8
        xdef      _var
```

ersetzt und kann, wie viele andere hier genannte Anpassungen, auch durch ein Makro implementiert werden. Probleme entstehen nur dann, wenn ein bestimmtes Symbol in mehreren Modulen auf diese Weise definiert wurde. Der Aztec-Linker verwendet die maximale Größenangabe und betrachtet alle anderen als Deklarationen. Ein solches Feature existiert bei Lattice-C nicht, so daß, wenn `BLink` sich über mehrfach definierte Symbole beschwert, manuell die größte Definition ausfindig gemacht, auf die oben genannte Weise ersetzt und alle anderen Stellen zu

```
xref _var
```

geändert werden müssen. Die `BSS`-Anweisung reserviert nur lokalen Speicher und kann durch `DS` ohne `XDEF` ersetzt werden.

Beide Assembler unterstützen die Speichermodelle und Segmentierung der C-Compiler, jedoch auf unterschiedliche Weise: Aztec hält hierfür die Anweisungen

```
near code
near data
far code
far data
```

bereit und legt Code- und Datenbereich durch die Anweisungen `CSEG` und `DSEG` fest. Unter Lattice wird dies kombiniert durch die `CSECT`-Direktive erledigt. Da für die Funktionsfähigkeit eines Programms die absolute Adressierung ausreicht, können `NEAR` und `FAR` entfallen sowie `CSEG` durch `SECTION CODE` und `DSEG` durch `SECTION DATA` ersetzt werden. Die `SECTION`-Anweisung sollte zu Beginn des Quelltextes - im Unterschied zu Metacomcos Makroassembler - stehen, da ansonsten Fehlermeldungen auftreten können.

Eine selten verwendete Fähigkeit des Aztec-Linkers ist aufwendiger umzusetzen: die Anweisung `ENTRY`. Diesem Schlüsselwort folgt der Name eines globalen Symbols, das vom Linker als Eintrittspunkt für

das entstehende Lademodul verwendet werden soll. Für die Anpassung an Lattice wird das diesem Symbol folgende Code-segment an den Beginn des Quelltextes verschoben und das entstandene Modul beim Binden als erstes Objektmodul spezifiziert. Fortgeschrittene können `ENTRY` auch durch den `SECTION`-Namen `«NTRY-HUNK»` implementieren.

Inline-Code

Der Aztec-C-Compiler kennt eine besondere Anweisung, die es erlaubt, Assembleranweisungen direkt in den Quelltext eines C-Programms einzustreuen, eine sehr praktische, aber möglicherweise auch sehr gefährliche oder gar unnötige Sache.

Hier müssen wir zwei bis drei Fälle unterscheiden:

- 1.) »echter« Inline-Code
- 2.a) separate Assemblerfunktionen
- 2.c) Assemblerfunktionen mit C-Kopf.

Der erste Fall kann uns vor Probleme stellen, die als »schwierig« bis »nicht portierbar« eingestuft werden können. Lassen Sie uns daher zur Eingewöhnung zunächst einen Blick auf die beiden letzteren werfen: 2a ist daran erkennbar, daß die `«asm»`-Anweisung außerhalb einer C-Funktion verwendet wird. Der Programmierer hat sich also lediglich die Aufteilung seines Quelltextes in ein C- und ein Assembler-Modul gespart. In diesem Fall muß der Bereich zwischen `«asm»` und `«endasm»` in eine separate Datei kopiert und (inklusive der beiden begrenzenden Anweisungen) aus dem C-Quelltext entfernt werden. Der C-Quelltext muß nun um Importanweisungen für die im Assemblermodul definierten Symbole ergänzt werden, entsprechend etwa im Assemblermodul definierte globale C-Symbole. Des weiteren müssen die globalen Symbole des Assemblermoduls exportiert werden, hierzu ein Beispiel:

```
int    globvar:
# asm
_bzero movea.l 4(sp),a0
        move    8(sp),d0
        bra.s   2$
1$     clr.b    (a0)+
2$     dbf      d0,1$
        rts
# endasm

void cfunc()
```



```
{
/* ... */
bzero (a, BUFSIZ);
/* ... */
}
```

Dies wird nun zu einem Assemblermodul:

```
section code
xdef _bzero
_bzero movea.l 4(sp),a0
move.l 8(sp),d0
;L hinzugefügt!
bra.s 2$
1$ clr.b (a0)+
2$ dbf d0,1$
rts
end
```

und zu einem C-Modul:

```
int globar;

extern void bzero
(char *, int);
void cfunc()
{
/* ... */
bzero(a, BUFSIZ);
/* ... */
}
```

Griffe die Assemblerfunktion auf die Variable **globvar** zu, so müßte dies durch die Anweisung

```
xref _globvar
```

angezeigt werden.

Beachten Sie übrigens auch die (bereits besprochene) Änderung von **MOVE** in **MOVE.L**. Im Fall, daß einem 16-Bit-Parameter weitere folgen, ändern sich natürlich die Parameter-Offsets (relativ zu Stapelzeiger).

Eine ebenfalls noch leicht pointbare Variable des vorherigen Falls liegt vor, wenn nur der Funktionskopf in C definiert wurde, die eigentliche Funktion jedoch in Assembler. Auch hier ein Beispiel:

```
viod bzero(dest, bytes)
char *dest
int bytes
{
#asm
movea.l 8(a5),a0
move 12(a5),d0
bra.s 2$
1$ clr.b (a0)+
2$ dbf d0,1$
#endasm
}
```

Hier wird vom C-Compiler der Funktionsein- und -ausgang erzeugt, der bei der Portierung nach Lattice explizit angegeben werden muß:

```
section code
xdef _bzero
```

```
_bzero link a6,#0
;hinzugefügt
movea.l 8(a5),
a0
move.l 12(a5)
,d0
;L, wie gehabt
bra.s 2$
1$ clr.b (a0)+
2$ dbf d0,1$
unlk a5
;hinzugefügt
rts
;hinzugefügt
end
```

Durch diese vom Aztec-Compiler erzeugten Anweisungen wird erreicht, daß die Parameter ab 8 (A5) adressiert werden können. A5 wird daher auch als »Frame-Pointer« bezeichnet. Auch hier müssen 16-Bit-Parameter durch Hinzufügen von .L erweitert werden, entsprechend gilt für die Anpassung der Offsets.

Wirklich schwierig wird es mit der #asm-Anweisung erst, wenn Assemblercode (scheinbar) wahllos in C-Funktionen eingestreut wurde. Zwar gibt es unter Lattice-C die funktionell identische Prozedur **_emit()**, mit deren Hilfe Assembleranweisungen eingefügt werden können, hierzu müssen jedoch vorher die den Assembleranweisungen entsprechenden Zahlenwerte bestimmt werden, etwa mit Hilfe des Assemblers. Glücklicherweise wird solcher »echter« Inline-Code nur selten verwendet und kann in den meisten Fällen bei Verwendung der absoluten Datenadressierung entfallen, da er lediglich dem Laden des Basisregisters A4 dient. Das Retten von Registern durch #asm, etwa beim Eintritt in eine Interrupt-Funktion, kann bei Lattice-C unterbleiben, da hier die Register entsprechend dem Amiga-Standard verwendet werden.

Ähnlich dem Inline-Code zum Setzen und Retten von Registern in Interrupt-Funktionen können auch die unter Lattice-C nicht vorhandenen - weil nicht benötigten - Funktionen **int_start()** und **int_end()** ersatzlos gestrichen werden. Bei der Erzeugung neuer Prozesse und Tasks, etwa durch **CreateTask()** oder **CreateProc()** muß unter Aztec-C die als Parameter übergebene Eintrittsfunktion zu Beginn mit der Anweisung **geta4()** versehen werden. Diese Anweisung wird von Lattice-C verstanden, kann aber bei Verwendung der Compileroption **-b0** wegfallen. Wird das Small-Data-Model verwendet, so be-

wirkt die Option **-y**, daß beim Eintritt in jede Funktion automatisch das Basisregister A4 geladen wird.

Eine weitere Spezialität stellt das Binden ohne Startup-Code dar. Hier wird bei Aztec mit der Option **+b** compiliert, unter Lattice muß der Linker separat vom Compiler, ohne Angabe des sonst üblichen Startup-Codes **LIB:c.o** aufgerufen werden.

Die Überprüfung auf Stapelüberlauf wird von Lattice per Default durchgeführt und muß explizit durch Option **-v** abgeschaltet werden, während **+m** sie unter Aztec-C erst einschaltet. Sollte ein Aztec-Programm die Symbole **__stkchk()** und **__stkover()** verwenden, so weist dies darauf hin, daß der Programmierer eine eigene Überprüfung des Stapelüberlaufs vornimmt. Diese ist nicht direkt portierbar, sollte jedoch auch keinen direkten Einfluß auf die Programmausführung haben.

Sogenannte »Load-and-Stay-Resident«-Programme (oder, in Anlehnung an MS-DOS, auch **TSRs**) sind kompatibel implementiert, es muß bei Lattice lediglich mit dem Startup-Code **LIB:cback.o** gelinkt werden. Bei Aztec-C ist **detach.o** für die »Abnabelung« verantwortlich.

Zwei Tricks, die von Aztec-Programmierern gerne zur Optimierung der Programmlänge angewandt werden, bestehen in der Reduzierung des aus der Kommandozeilenanalyse resultierenden Overheads. Hierzu wird die Eintrittsfunktion nicht **main()**, sondern **__main()** genannt. Diese erhält zwei Parameter: als ersten Parameter die Anzahl der gültigen Zeichen der Kommandozeile und als zweiten einen Zeiger auf die Kommandozeile selbst. Auch unter Lattice-C existiert eine solche Funktion **__main()**, sie erhält jedoch lediglich einen Zeiger auf einen mit einem Nullbyte beendeten String. Dieser Unterschied sollte auch C-Einsteiger vor kein Problem stellen. Stub-Definitionen der internen Funktionen **__cli_parse()** und **__wb_parse()**, die für die Auswertung von Parametern zuständig sind, können entfallen, haben jedoch keinen Seiteneffekt auf die Funktionsfähigkeit eines Programms.

Wenn's dann nicht läuft...

Verabschiedet sich das auf die beschriebene Weise adaptierte Programm bei Ausführ-

ung noch immer durch einen Guru oder ähnliches, so bleibt wohl nur noch der Source-Level-Debugger. Dort verfolgen Sie die Programmausführung bis zur Crash-Position (mit Trace geht es Funktionen hinein) und untersuchen den Objektcode des zugehörigen Programmsegments durch CPR im Mixed-Modus oder durch OMD.

Auch das Startup-Modul **catch.o** leistet bei solch schwierigen Fällen hilfreiche Dienste, da es Abstürze abfangen kann und dabei die Stelle und den Umstand des Absturzes näher eingrenzen hilft.

Auf diese »allgemeine« Weise übertragene Programme sind natürlich noch nicht optimiert und benötigen gewisse Verfeinerungen, um alle Eigenschaften des Lattice-Compilers nutzen zu können. Dazu gehören dann beispielsweise die Parameterübergabe in Registern und der geschickte Einsatz der Speichermodule. Auch der »Globale Optimizer« kann in einigen Fällen, speziell beim Umgang mit I/O-Registern, erst eingesetzt werden, wenn das Schlüsselwort **volatile** richtig angewandt wird, da ansonsten - im Unterschied zu Aztec-C - Optimierungen vorgenommen werden, die über den Rahmen des Erwünschten hinausgehen. Die Umsetzung auf 16-Bit-Integer (Lattice-Option **-w**) ist unter Lattice durch die Unterstützung von ANSI-Prototypen zwar wesentlich einfacher, bringt aber nur selten mehr als rund 1 Prozent der Programmlänge.

Viele der hier genannten Tips können übrigens auch von Aztec-Programmierern berücksichtigt werden, ohne daß diese deshalb irgendwelche Leistungseinbußen hinnehmen müssen. Die Anwendung der Präprozessorsymbole **AZTEC_C** und **LATTICE_C** tun ein übriges zur leichten Portierung.

so

»Using C will definitely cut your life expectancy by ten years or more« - Carl Sassenfath, Gurus Guide #1, Seite 45.

Literaturhinweise

- [1] Babel, Ralph: Das Amiga-Guru-Buch, Traunstein, 1989, Selbstverlag
- [2] Kernighan, Brian W. & Dennis M. Ritchie, The C Programming Language, 1978, ISBN 0-13-110163-3, Prentice-Hall, Inc.
- [3] M68000, 8-/16-/32-Bit Microprocessors Programmers' References Manual, ISBN 1-870760-02-6, Motorola, Inc.

Ralph Babel ist der Programmierer des Diskettenmonitors »Discovery«. Er schreibt Artikel für verschiedene Computermagazine. Sie können ihn über unsere Verlagsanschrift erreichen (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Herrn Ralph Babel, Hans-Pinsel-Straße 2, 8013 Haar bei München).

Mathematische Spieltheorie

Spiel mit Zahlen

Warum habe ich gegen meinen Schachcomputer keine Chance? Hier erfahren Sie mehr über die mathematischen Hintergründe.

Von Jürgen Singer

Computer gelten als fleißig, aber dumm. Wie sollen diese »geistlosen« Maschinen je die Spielstärke eines kreativ denkenden menschlichen Gegners erreichen? Natürlich werden Sie einwenden, auch Fleiß hat seine guten Seiten. Die Maschine könnte ja alle denkbaren Möglichkeiten durchkalkulieren und sich dann für die beste ent-

scheiden – eine Möglichkeit, die der Mensch bei aller Kreativität nicht hat.

Doch wie groß wäre der Rechenaufwand für diese Sisyphusleistung? Je nach Situation haben Sie für den nächsten Zug die Wahl unter etwa sechzig Möglichkeiten. Zu bedenken gilt es jedoch nicht nur den nächsten Zug, sondern auch die mögliche Reaktion des Gegners (nochmals etwa sechzig Züge pro eigener Zugmöglichkeit) und alle weiteren Züge

(einschließlich Reaktionen des Gegners) bis zum zwangsläufigen Matt. Gehen wir davon aus, daß eine Partie im Schnitt etwa 60 Züge dauert, so ist dies ein Rechenaufwand von 60 hoch 120 Varianten, die kalkuliert sein wollen, bevor die nächste Figur gesetzt wird – alles in allem eine Zahl mit enorm vielen Nullen. Zur Berechnung des ersten Zuges veranschlagen wir am besten einen Aufwand von einigen Jahren – immer vorausgesetzt, wir haben einen Super-



computer vom Format einer Cray zur Verfügung, den wir mit unseren Daten füttern können.

Sie haben keine Cray? Und auch keine Zeit, um solange zu warten? Dann sollten Sie sich einen effektiveren Algorithmus einfallen lassen. Denn es ist sicher nicht sinnvoll, jedesmal alle Züge zu kalkulieren, bevor Sie einen Spielstein setzen. Bevor wir uns mit unseren Überlegungen an das – relativ komplizierte – Schachspiel wagen, lassen Sie uns mit einem einfa-

chen Beispiel beginnen: Es gibt zwei Spieler, die wir »Links« und »Rechts« nennen wollen. Zudem gibt es »Bauklötzchen«, die die Aufdrucke »L« und »R« tragen. Ein Spielzug von »Links« bedeutet, einen Klotz mit dem Aufdruck »L« zu entfernen, analog darf »Rechts« bei jedem Zug einen »R«-Klotz entfernen. Die Klötze sind übereinander gestapelt, so daß beim Entfernen eines Klotzes auch alle weiteren Klötze, die dann nicht mehr mit der Erde ver-

bunden sind, entfernt werden. Als Verlierer gilt der Spieler, der zuerst nicht mehr ziehen kann. Selbstverständlich würden Sie es vorziehen, der Gewinner zu sein, doch wie stellt man das an?

Zum Aufwärmen betrachten wir die Situation in Bild 1: Hier liegen zwei getrennte Haufen vor. Es entscheidet allein die Anzahl der vorhandenen Steine, sofern niemand auf die Idee verfällt, einen Stein zu entfernen, der noch weitere Steine ►

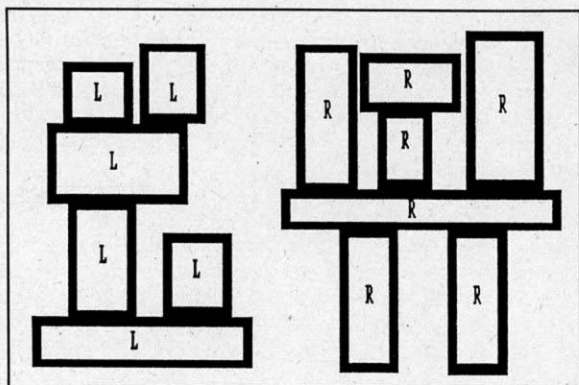


Bild 1.
Die
Anzahl
Steine
ent-
scheidet
über
den
Gewin-
ner

trägt. Da »Links« über 6 Steine verfügt, während »Rechts« deren 7 hat, gewinnt »Rechts« auf jeden Fall (sofern er keine unvernünftigen Züge macht), unabhängig davon, ob »Links« oder »Rechts« das Spiel beginnt.

Wie sieht die Situation nun in Bild 2 aus? Jeder Spieler ver-

4a dargestellt: Beginnt »Rechts«, so kann er nur den oberen Klotz nehmen; im nächsten Zug nimmt »Links« den unteren Klotz und gewinnt damit. Beginnt dagegen »Links«, so nimmt er sofort den unteren Klotz und entfernt damit auch den oberen Klotz. Damit kann »Rechts« nicht mehr ziehen

– ein Nullspiel, bei dem »Links« beginnt. Ist nun andererseits »Rechts« als erster am Zug, so kann er wieder einen Stein von einem zu »Links« gehörenden Stapel herunternehmen. Die beste Chance für »Links« besteht darin, »Rechts« einen Zug zu stehlen, daher wird »Links« den verbleibenden Turm neh-

von 7,5. Stellen wir uns die Frage, welche Züge hier möglich sind, so ergibt sich für »Rechts« nur ein möglicher Zug, der zur Position $7+1 = 8$ für »Links« führt. »Links« kann dagegen entweder einen Klotz aus seinem privaten Stapel nehmen, was 6,5 ergeben würde, oder besser er nimmt den 1/2-Stapel,

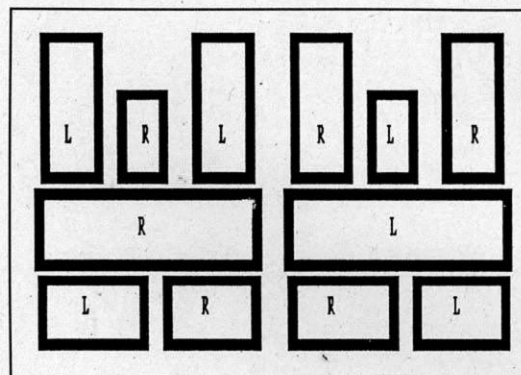


Bild 3.
Änderungen
durch
Mischen?

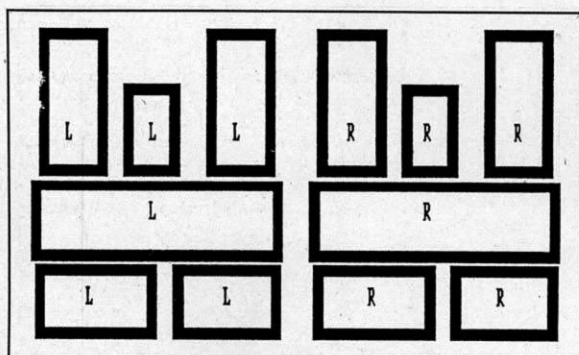


Bild 2.
Bei
gleicher
Anzahl
gewinnt,
wer als
zweiter
zieht

fügt über die gleiche Anzahl von Klötzen. Bei optimalen Zügen (das heißt jeder Spieler nimmt immer nur jeweils einen Stein) kann derjenige zuerst nicht mehr ziehen, der das Spiel begonnen hat. Ein Spiel bei dem der, der anfängt, zwangsläufig verliert, soll »Nullspiel« heißen.

Bild 3 unterscheidet sich von Bild 2 dadurch, daß einige Klötze zwischen den beiden Stapeln ausgetauscht wurden. Wie ist die Situation jetzt? Aus Symmetriegründen kann man natürlich vermuten, daß auch hier keiner der Spieler einen echten Vorteil haben wird, doch stimmt das? Sicherlich, denn was auch immer ein Spieler unternimmt, sein Gegner kann mit genau demselben Zug am anderen Stapel kontern. Somit verliert auch hier derjenige, der das Spiel beginnt und wir haben wieder ein klassisches Nullspiel. Die Hauptschwierigkeit bei diesem Spiel ist, daß es unser Gegner einrichten kann, uns einige Züge zu stehlen, indem er Klötze entfernt, die Klötze von uns tragen. Die einfachste derartige Situation ist in Bild

und »Links« gewinnt wieder. Beträgt der Vorteil von »Links« genau einen Zug? Wir können diese Hypothese testen, indem wir »Rechts« einen Zug mehr verschaffen (Bild 4b). Dann aber kann »Rechts« gewinnen, unabhängig davon, wer beginnt. Also war das Hinzufügen eines ganzen Zuges zuviel des Gu-

Was sind halbe Züge?

ten – der Vorteil von »Links« wurde überkompensiert. Daher ist der Vorteil von »Links« kleiner als die Vorgabe eines freien, ganzen Zuges. Stellen wir also die Hypothese auf, daß der Vorteil von »Links« einen halben Zug beträgt. Wenn wir mit unserer Vermutung recht haben, dann muß Bild 4c ein Nullspiel darstellen, denn der Vorteil von zwei halben Zügen für »Links« sollte den freien Zug von »Rechts« gerade aufheben. Lassen wir also »Links« beginnen. Es kann nur einer der beiden Zweierstapel entfernt werden. »Rechts« kann nun seinen Klotz vom zu »Links« gehörenden Klotz entfernen und wir erhalten die Situation von Bild 4d

men und wir enden wieder bei der Situation 4d. Da nun aber »Rechts« am Zug ist, gewinnt »Links«. Also handelt es sich bei Bild 4c tatsächlich um ein Nullspiel und unsere Hypothese, daß Bild 4a einen Vorsprung von einem halben Zug für »Links« bedeutet, hat sich bestätigt. Werfen wir einen Blick auf Bild 5. Wie sofort ersichtlich ist, hat diese Figur einen Wert

was zu 7 führt. Diese Optionen wollen wir in Zukunft durch die Gleichung $\{7|8\} = 7,5$ ausdrücken. Allgemeiner gilt also $\{n|n+1\} = n + 1/2$. Es gibt auch noch die einfachere Gleichung $\{n|1\} = n+1$; denn hat »Links« $n+1$ freie Züge, so hat er nach seinem Zug natürlich noch n freie Züge, »Rechts« kann dagegen überhaupt nicht ziehen.

Was ist ein Spiel?

Um den Begriff »Spiel« exakt fassen zu können, muß dieser zunächst definiert sein. Wir einigen uns für diesen Artikel auf folgende Definition:

1. Es gibt genau zwei Spieler, hier als »Links« und »Rechts« bezeichnet.
2. Es gibt verschiedene (endlich viele) Positionen, häufig gibt es eine definierte Startposition.
3. Es gibt feste Spielregeln, die von einer Position aus bestimmte Züge erlauben, welche zu neuen Positionen führen. Diese neuen Positionen sind die »Optionen« der aktuellen Position.
4. »Links« und »Rechts« ziehen während des ganzen Spiels abwechselnd.
5. Derjenige Spieler, der nicht mehr ziehen kann, hat verloren.
6. Das Spiel ist nach einer endlichen Zahl von Zügen beendet. Es gibt keine Schleifen, bei denen eine bestimmte Position immer wieder auftritt.
7. Beide Spieler verfügen über die vollständige Information über ihre Optionen und die ihres Gegners.
8. Es gibt keine Zufallszüge (Würfeln oder Ziehen von Karten).

Beispiele:

- Schach und Dame: verstoßen gegen Punkte 5 und 6.
- Backgammon: verstößt gegen Punkt 8.
- Schiffeversenken: verstößt gegen Punkt 7, denn die Position des Gegners ist unbekannt.
- Pflock-Solitaire: ist ein Ein-Personen-Spiel, verstößt damit gegen Punkt 1.
- Nim: erfüllt alle Bedingungen und zusätzlich eine weitere: Jeder Spieler kann von jeder Position aus die gleichen Züge machen. Ein derartiges Spiel heißt neutral.

Daß einige Spiele gegen die hier genannten Bedingungen verstoßen, bedeutet nicht, daß Sie nicht mit denselben Algorithmen erfaßt werden können. Sie werden nur der Einfachheit halber aus der Betrachtung ausgeschlossen.

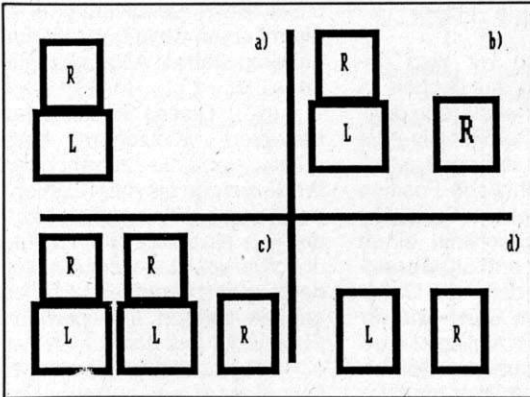


Bild 4.
Vorteil
durch
halbe
Züge?

Die einfachste Gleichung ist zweifellos $\{1\} = 0$, denn da niemand ziehen kann, verliert automatisch derjenige, der das Spiel beginnt und wir haben ein Nullspiel vorliegen.

Betrachten wir nun ein neues Spiel: Es wird auf einem $n \times m$ -Schachbrett gespielt. Jeder Spieler besitzt Steine, die sein Symbol (L oder R) tragen. Der Spieler »Links« darf mit seinen Figuren nur nach Osten ziehen, während »Rechts« nur nach Westen ziehen darf. Die Zugweite bleibt jedem Spieler freigestellt, er muß jedoch mindestens ein Feld weit ziehen. Einzige Ausnahme ist der Fall, wenn er genau über einer gegnerischen Figur steht und unter dieser ein freies Feld ist: Hier

ser für »Links« ist deshalb ein Sprung, der den Vorsprung von »Rechts« auf 1 schrumpfen läßt. Also ist die Bewertung der Situation in Bild 7 $\{-1 | 0\} = -1/2$. Ganz analog läßt sich bei Springer jede Situation dann einfach durch Auszählen der Felder östlich der »Linken« Figuren und abziehen der Felder westlich der »Rechten« Figuren bewerten, wenn keine Sprünge mehr möglich sind.

Spielbäume

Um eine beliebige Ausgangssituation zu bewerten, müssen wir erst alle Zugmöglichkeiten durchspielen, bis wir jeweils eine Bild 7 entsprechende Stellung erreichen. Da wir

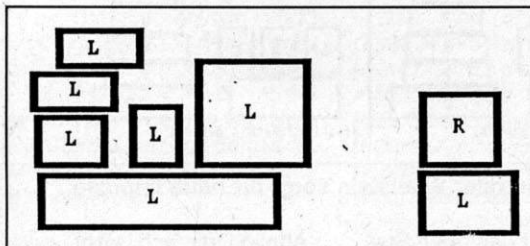


Bild 5. Der
entscheidende
Fortschritt:
die Bewertung
von
Spielsituationen

darf er auswählen, ob er nicht, statt einen normalen Zug zu machen, die gegnerische Figur überspringen möchte, wobei eine schon einmal übersprungene Figur selbst keine anderen Figuren mehr überspringen darf. Bild 6 zeigt als Beispiel ein 4×6 -Springer-Spiel. Solange sich die Figuren nicht überspringen können, ist der Fall klar: Es handelt sich dann um unser Bauklötzchen-Spiel in verkappter Form.

Doch welche Vorteile bringt ein Sprung? Betrachten wir dazu Bild 7: Ist »Rechts« am Zug, so bietet sich keine Sprungmöglichkeit, und »Rechts« zieht ein Feld nach Westen, wobei ein Nullspiel entsteht. Ist dagegen »Links« am Zug, so kann er entweder nach Osten ziehen, was einen Vorsprung von zwei Zügen für »Rechts« ergibt; bes-

deren Bewertung kennen, können die Ergebnisse dann rückwärts wieder in die vorhergehenden Züge eingesetzt werden, bis schließlich auch die Ausgangssituation bewertet ist. Ein entsprechender Spielbaum kann in Bild 8 bewundert werden. Dabei gilt es aber noch eine Schwierigkeit zu lösen: Wie berechnet man $\{0.5 | 3.5\} = 1$ oder $\{-2.5 | -0.5\} = -1$? Zum besseren Verständnis des Berechnungsprinzips dient Bild 9: Hier ist ein Zahlenbaum angegeben, der die Zahlengerade nach Brüchen von Zweierpotenzen unterteilt. Die Bewertung unserer dubiosen Stellung erhalten wir als die »einfachste« Zahl, die tatsächlich größer als die »Links«-Option und kleiner als die »Rechts«-Option ist. Dabei heißt einfache Zahl nichts anderes, als mög-

lichst nahe an der Wurzel (der 0) des Zahlenbaumes. Einige interessante Neuerungen ergeben sich, wenn die »Links«-Optionen größer sind als die »Rechts«-Optionen, denn dann ergeben sich als Bewertung keine Zahlen mehr. Aus Platzgründen können wir hier leider nicht auf diese Eigenschaften eingehen, sie werden jedoch in der angegebenen mathematischen Literatur ausführlich behandelt, so daß Sie Ihrer Neu-

programm benötigt werden: Ein Teil des Programms muß bestimmen, welche Züge möglich sind, ein anderer Teil muß die nach einer bestimmten Anzahl von Zügen entstandene Situa-

Unscharfe Spiele

tion bewerten. Was in unserem Beispiel fehlt, ist die Möglichkeit für einen Spieler, den besten unter mehreren Zügen auszuwählen. Auf diese Pro-

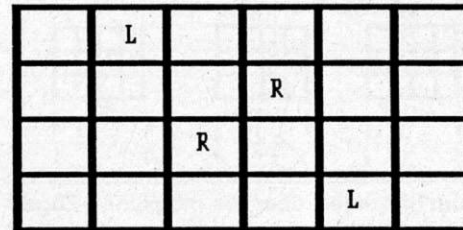


Bild 6. Neue
Möglichkeiten
mit
»Springer«

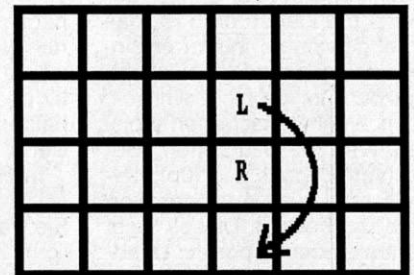


Bild 7.
Auch hier ist von
Bedeutung,
welcher
Spieler am Zug
ist

gier keinen Zwang antun müssen.

Listing 1 zeigt ein C-Programm, das den Spielbaum aus Bild 8 (siehe auch Bild 9) berechnet. Dabei existiert im Speicher kein direktes Abbild des Spielbaums, sondern er wird durch wiederholte Aufrufe der Bewertungsfunktion (Rekursion) implizit erzeugt. Die dargestellte Situation ist sehr einfach, da jeder Spieler nur eine Figur besitzt, und auch die Zugmöglichkeiten sind sehr gering – in »richtigen« Spielen, wie etwa Dame muß hier schon etwas mehr Aufwand getrieben werden. Trotz dieser Einschränkungen enthält das Listing fast alle Teile, die von einem Spiel-

grammteil werden wir später noch ausführlich zu sprechen kommen. Vorerst geben wir uns damit zufrieden, Werte wie $\{-3.5 | -0.25\}$ berechnen zu können.

Lassen Sie uns einen Blick auf Tabelle 1 werfen: Bis auf die mit »unscharf« bezeichnete Situation sind uns alle Möglichkeiten schon untergekommen. Um auch Situationen zu erfassen, in denen der Spieler gewinnt, der das Spiel beginnt, benötigen wir in unserem Klötzchenspiel eine neue Art von Spielstein. Dieser ist einfach ein Klotz ohne Aufdruck, daher kann er von jedem der Spieler genommen werden. Betrachten wir das Spiel in Bild 10a). Da der unterste Klotz keinen Auf-

Unscharfe Spiele

	Links beginnt		
		Links gewinnt	Rechts gewinnt
	Links gewinnt	positiv (L gewinnt)	Null (2 gewinnt)
	Rechts gewinnt	unscharf (1 gewinnt)	negativ (R gewinnt)

Tabelle 1. So sieht das bis zum Ende berechnete Spiel aus

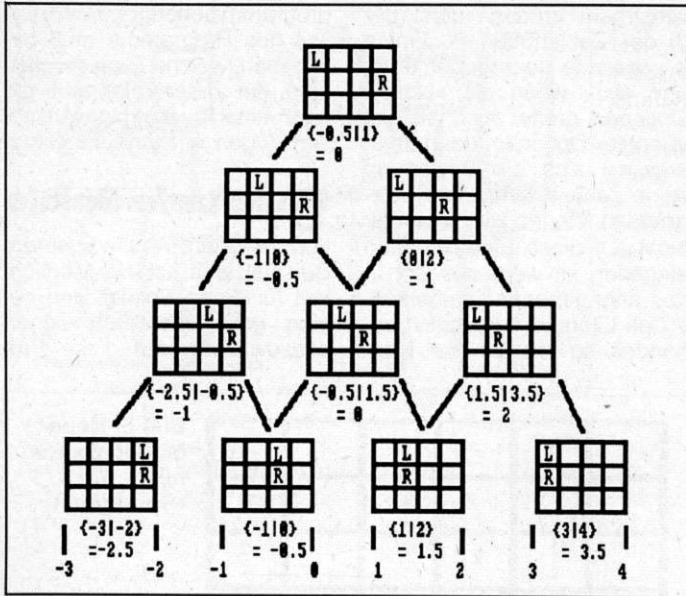


Bild 8. Der Spielbaum informiert über die möglichen Züge

druck hat, kann der Spieler, der zuerst mit Ziehen dran ist, das Spiel gewinnen, indem er einfach den ganzen Stapel nimmt. Trotzdem ist es nicht sinnvoll, auch einem unscharfen Spiel den Wert 0 zuzuordnen. Betrachten wir dazu Bild 10b), welches aus zwei Versionen von Bild 10a) besteht. Dieses Spiel ist nun eindeutig positiv: Unabhängig davon wer beginnt, hat »Links« genügend freie Züge, um dafür zu sorgen, daß »Rechts« als erster einen neutralen Stein nehmen muß – »Links« nimmt dann den anderen neutralen Stein und gewinnt. Wäre der Wert von Bild 10a) 0, so müßte auch der Wert von Bild 10b) 0 sein, denn die Summe zweier Nullspiele ist wieder ein Nullspiel.

Wie viele Züge für »Links« müssen wir hinzufügen, um ein positives Spiel zu erhalten, wie

viele Züge für »Rechts« für ein negatives Spiel? Fügen wir einen ganzen Zug für »Links« hinzu, so ist das Spiel sicher positiv, doch wie steht es mit einem halben Zug, oder gar einem viertel Zug.

In Bild 10c) fügen wir einen $\frac{1}{64}$ Zug hinzu (bitte rechnen Sie nach – es stimmt tatsächlich), und schon ist das Spiel positiv, denn falls »Links« beginnt, nimmt er den neutralen Stein und entfernt damit den ganzen Stapel.

Im anderen Stapel gehört »Links« jedoch der unterste Stein und er hat auch hier den letzten Zug. Dabei ist es vollkommen belanglos, wie viele Steine »Rechts« auf dem Stein von »Links« liegen hat, solange der Wert des Stapels nur größer als 0 ist. Wir können die Situation aber auch umkehren und einen negativen Stapel (Bild 10d) hinzufügen.

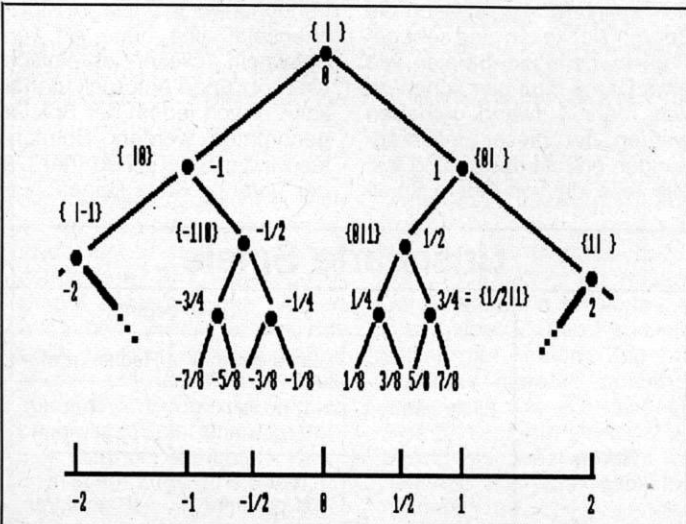


Bild 9. Eine neue Variante des Zahlenbaums aus Bild 8

Jetzt gewinnt in jedem Fall »Rechts«.

Damit haben wir also gezeigt: Der Wert des Stapels in Bild 10a) ist kleiner als jede positive Zahl, aber größer als jede negative Zahl, dennoch ist er nicht 0. Eine ähnliche Position erhält man bei einem Spiel, das als einzigen Spielstein einen neutralen Klotz enthält: Wer anfängt gewinnt, denn die Optionen bei diesem Spiel sind für beide Spieler ein Nullspiel, also ist der Wert unseres Spieles $\{0|0\}$. Da dieser Wert häufiger auftritt, wollen wir ihn mit dem Symbol $\ast$ abkürzen. Allgemeiner können wir dann auch schreiben $\{x|x\} = x + \ast = x\ast$, ähnlich wie wir ja auch $3\frac{1}{2}$ für $3 + \frac{1}{2}$ schreiben. Setzen wir zwei neutrale Klötze nebeneinander, so erhalten wir natürlich wieder ein Nullspiel, also gilt die Gleichung $\ast + \ast = 0$.

0 Steinen zurückbleibt, oder er nimmt einen Stein und läßt den anderen stehen. Allgemein gilt: $\ast n = \{\ast 0, \ast 1, \dots, \ast (n-1) | \ast 0, \ast 1, \dots, \ast (n-1)\}$. Dieses Spiel mit den neutralen Klötchen heißt »Nim« und die zugehörigen Werte werden als »Nim-Zahlen« bezeichnet. Wir sehen sofort, daß ein einzelner Nim-Haufen unscharf ist, denn der Spieler, der beginnt, nimmt einfach den ganzen Haufen und gewinnt. Ebenso ist uns sofort klar, daß ein Spiel aus zwei gleich großen Haufen ein Nullspiel ist, denn der zweite Spieler kann gewinnen, indem er jeden Zug des ersten Spielers nachmacht. Als Gleichung ausgedrückt heißt dies: $\ast n + \ast n = 0$. Da sich zwei gleich große Haufen zu 0 addieren, kann man sie bei der Betrachtung von Spielen mit mehreren Haufen auch einfach weglassen.

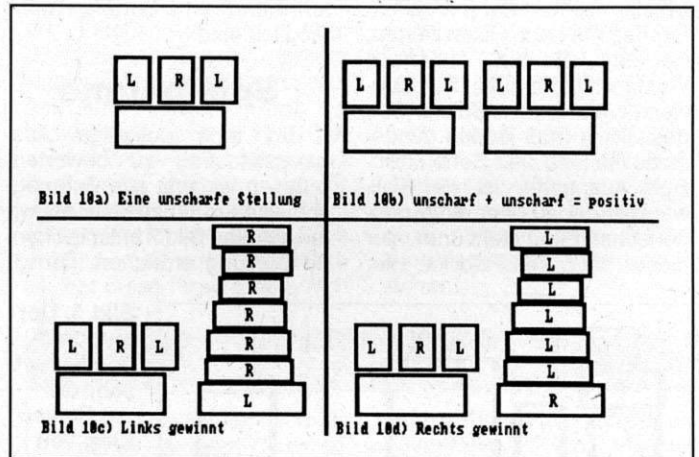


Bild 10. Ein neutraler Spielstein sorgt für neue Impulse

Da Spiele mit neutralen Steinen offensichtlich interessant werden können, betrachten wir doch einfach Varianten, die nur aus neutralen Steinen aufgebaut werden. Wie üblich, spielen wir mit mehreren Haufen von Klötzen (Bild 11). Führen wir

Ähnlich einfach sieht man, daß zwei ungleiche Haufen wieder ein unscharfes Spiel ergeben: der erste Spieler macht beide Haufen gleich hoch. Damit ist in einem Spiel mit drei Haufen der Spieler der Verlierer, der entweder zuerst einen Haufen ganz entfernt, oder aber einen Haufen gleich hoch macht wie einen anderen. Aus diesen Gründen ist es einem Spieler dringend angeraten, im Spiel $\ast 1 + \ast 2 + \ast 3$ nicht anzufangen – denn wer anfängt, verliert. Damit ist aber gezeigt: $\ast 1 + \ast 2 + \ast 3 = 0$. Addieren wir zu dieser Gleichung auf beiden Seiten $\ast 1, \ast 2$ oder $\ast 3$, so erhalten wir wegen $\ast n + \ast n = 0$: $\ast 2 + \ast 3 = \ast 1$, $\ast 1 + \ast 3 = \ast 2$, $\ast 1 + \ast 2 = \ast 3$. Aus diesen etwas merkwürdigen Gleichungen ist ersichtlich, daß sich Nim-Zahlen anders addieren als reale Zahlen. Da wir hier nicht zu weit in die Theorie abschweifen

Nim und Nim-Zahlen

als Abkürzung für den Wert eines Haufens aus n neutralen Klötzen das Symbol $\ast n$ ein. Damit gelten die folgenden Gleichungen: $\ast 0 = \{1\} = 0$, $\ast 1 = \{\ast 0 | \ast 0\} = \ast$. Für Haufen, die aus mehreren Klötzen bestehen, haben wir mehrere mögliche Spielzüge.

Damit ergibt sich $\ast 2 = \{\ast 0, \ast 1 | \ast 0, \ast 1\} = \{0, \ast 1 | 0, \ast\}$, den der Spieler, der am Zug ist, kann entweder beide Steine nehmen, wobei ein Haufen aus

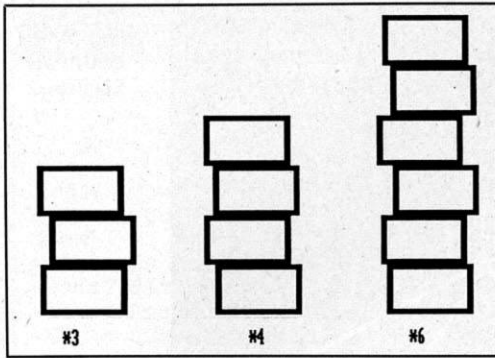


Bild 11.
Nim-Spiele
und
Nim-Zahlen

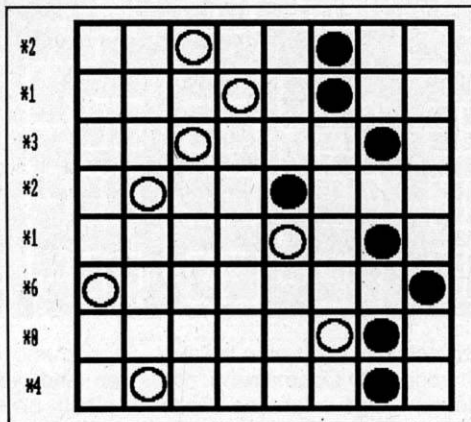


Bild 12.
Das Northcot-
Spiel

wollen, sei nur gesagt, daß die Addition von Nim-Zahlen dem Bilden der »exklusiven oder«-Verknüpfung der zugehörigen normalen Zahlen entspricht, also $*6 + *4 = *(6 \text{ XOR } 4) = *2$. Mit Hilfe dieser Rechenregeln läßt sich bestimmen, wie in Bild 11 gezogen werden muß, um zu gewinnen: der Zug von $*3$ nach $*2$ liefert $*2 + *4 + *6 = 0$, daher wird dem Gegner ein Nullspiel überlassen und derjenige, der jetzt am Zug ist, kann das Spiel gewinnen.

Erweitertes Nim

Wenden wir uns einem Spiel zu, das nach seinem Erfinder Northcot benannt ist (Bild 12). Es wird auf einem Schachbrett gespielt, wobei jeder Spieler acht Damesteine besitzt. Jeder Spieler darf nur in seiner Reihe ziehen, wobei jedoch ein gegnerischer Stein nicht übersprungen werden darf. Auf den ersten Blick hat dieses Spiel nichts mit Nim zu tun, doch interpretiert man den Abstand zwischen den jeweiligen Steinen einer Reihe als einen Stapel von Klötzen, so entspricht das aufeinander zu ziehen dem Verringern des Stapels. Zum Unterschied zum normalen Nim kann hier jedoch ein Spieler auch Klötze zum Stapel hinzufügen, indem er vom Gegner weg zieht. Dies braucht uns jedoch nicht weiter zu beunruhigen, denn jeden Zug, bei dem

ein Spieler einen Haufen vergrößert, kann sein Gegenspieler im nächsten Zug wieder rückgängig machen. Besitzt man also für gewöhnliches Nim eine Gewinnstrategie, so auch für diese erweiterte Version.

Daß sich Northcots Spiel durch Nim beschreiben läßt, ist kein Zufall: Es gibt einen mathematischen Satz, daß jedes neutrale Spiel äquivalent zu erweitertem Nim ist. Es stellt sich folgende Frage: Haben wir ein neutrales Spiel, bei dem beide Spieler die Optionen $\{*a, *b, *c, \dots | *a, *b, *c, \dots\}$ besitzen, welcher Nim-Haufen entspricht dann diesem Spiel! Um noch etwas konkreter zu werden: Welche Nim-Zahl gehört zu $*x = \{*0, *1, *4, *9 | *0, *1, *4, *9\}$? Die Nim-Zahl $*x$ läßt sich sehr einfach mit Hilfe der sogenannten MEX-Regel bestimmen (MEX = minimal excluded number, d. h. die kleinste ausgeschlossene Zahl). Die kleinste Zahl, die in unserem Beispiel nicht vorkommt, ist $*2$. Wie wir schon wissen, hat $*2$ die Zahlen $*0$ und $*1$ als Optionen, die Zahlen $*4$ und $*9$ können wir uns als solche Optionen vorstellen, bei denen der Haufen von 2 auf 4 oder 9 Steine vergrößert werden kann.

Das Spiel Nim (und damit alle neutralen Spiele) haben den entscheidenden Vorteil, daß sie noch einigermaßen leicht berechenbar sind, für unser erstes Spiel, bei dem die Klötzchen

durch Aufdrucke den jeweiligen Spielern zugeordnet waren, gilt dies nicht. Wollen wir hierfür ein Programm für unseren Computer schreiben, so müssen wir damit rechnen, daß die Zeit, die der Computer zur Berechnung der Spielpositionen benötigt, exponentiell mit der »Länge« der eingegebenen Anfangssituation zunimmt. Derartige Spiele nennt man »hart«. Weitere harte Spiele sind etwa Dame und Schach. Führt man eine (sehr konservative) Schätzung der benötigten Rechenzeit für ein Dame-Spiel durch, so erhält man für die schnellsten heute verfügbaren Computer Laufzeiten von einigen 10000 Jahren. Da es aber Schach- und Dameprogramme überall zu kaufen gibt, heißt dies, daß diese Programme nach anderen Prinzipien arbeiten müssen. Doch auch diese Spielprogramme können auf den Spielbaum nicht verzichten. Aber es bietet sich doch eine erste Verbesserung an: Im Spielbaum von Bild 8 ist bei jedem Zug offen, welcher der beiden Spieler nun tatsächlich zieht – denn von Interesse war ja die Bewertung der Position, unabhängig davon, wer gerade am Zug ist. Bei einem konkreten Spiel, das auf einem Computer implementiert werden soll, ist die Situation etwas anders: Es ist bekannt, welcher Spieler am Zug ist – daher kann man den Spielbaum um etwa die Hälfte reduzieren, doch dies alleine hilft nicht sehr viel, denn die Hälfte von einigen zehntausend Jahren Rechenzeit sind immer noch einige fünftausend Jahre. Es führt also kein Weg daran vorbei: Es ist einfach nicht möglich, den kompletten Spielbaum zu berechnen. Also bleibt nichts anderes übrig, als die Berechnun-

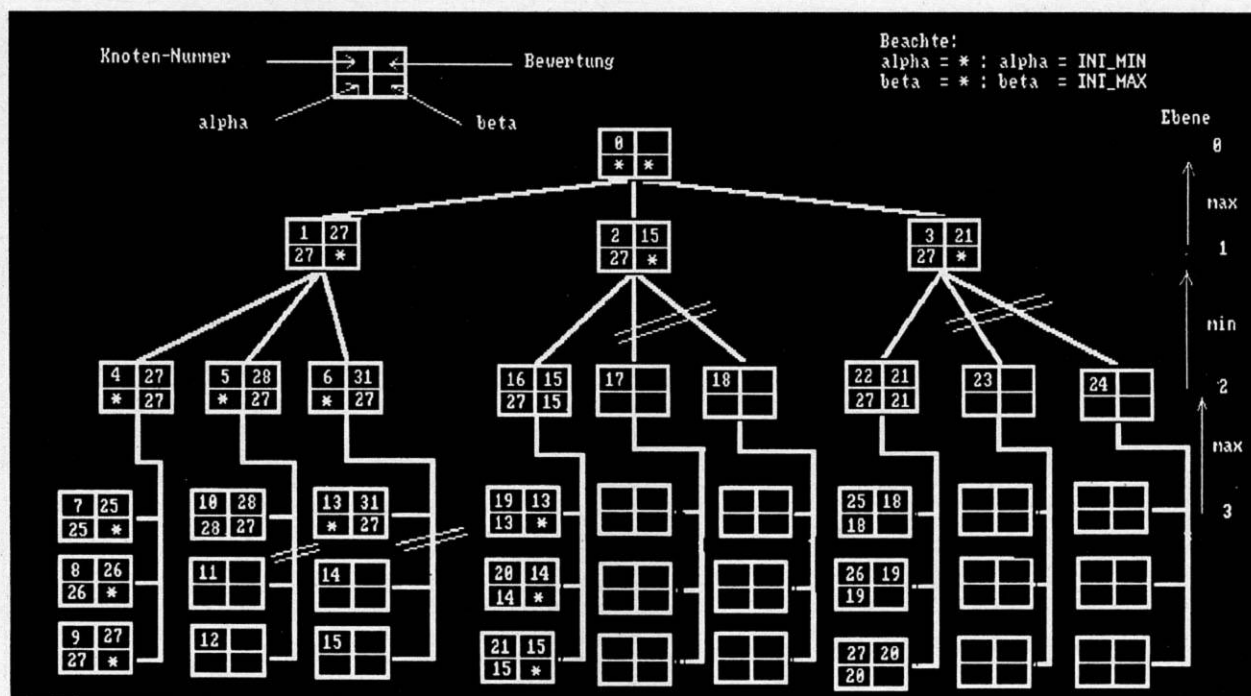
gen nach einigen Ästen des Baumes abubrechen und zu versuchen, die dann entstandene Situation zu bewerten. Da diese Bewertung nur eine Abschätzung sein kann, müssen wir uns nicht um unsere seltsamen Nim-Zahlen kümmern, sondern es genügt vollkommen, sich auf ganze Zahlen zu beschränken. Auch hier gilt wieder, je größer die Zahl, um so besser für den Inhaber der bewerteten Stellung. Logischerweise heißt dies, daß der Gegner versucht möglichst kleine Zahlen (aus unserer Sicht) als Bewertung zu erreichen. Die Bewertung selbst ist dabei die größte Schwierigkeit, denn um die Bewertung anhand einer Spielposition durchzuführen (die auch noch relativ kompliziert sein kann), müssen wir das in Frage stehende Spiel gut beherrschen, wir müssen also Experten für dieses Spiel sein.

Nicht umsonst wird die Programmierung von Dame- und Schachspielen den Bereichen der Künstlichen Intelligenz und den Expertensystemen zugeordnet. Die Erfahrung in der Bewertung der Situation läßt sich auch durch keinen noch so gut programmierten Spielbaum oder andere Tricks ausgleichen, denn ist erst einmal eine Situation falsch bewertet worden, so kann nichts unseren Spielbaum davon abhalten, diese falsche Bewertung bis nach oben an die Wurzel durchzureichen, was natürlich zu vollkommen unsinnigen Zügen führen kann. Da zur Bewertung der jeweiligen Situation Expertenwissen vonnöten ist, soll hier auch nicht näher darauf eingegangen werden (hier geht es ausschließlich um die mathematischen Grundlagen). Als Kriterien können etwa die An-

Bewertungen von Spielpositionen

- $\{a, b, c, \dots | d, e, f, \dots\}$ ist eine Spielposition, bei der die Optionen von »Links« zu den Werten $a, b, c, \dots$ führen, während die Optionen von »Rechts« zu den Werten $d, e, f, \dots$ führen.
- Die ganzen Zahlen entsprechen in dieser Schreibweise folgenden Ausdrücken:
 - $\{1\} = 0, \{0\} = 1, \{1\} = 2, \dots \{n\} = n+1$
 - $\{1\} = -1, \{1-1\} = -2, \dots \{1-n\} = -(n+1)$
- Es treten auch halbzahlige Züge auf:
 - $\{0\} = 1/2, \{1\} = 3/2, \dots \{n\} = n + 1/2$
 - $\{-1\} = -1/2, \dots \{-(n+1)\} = -(n + 1/2)$
- Zweierpotenzen:
 - $\{0\} = 1/2, \{0\} = 1/4, \{0\} = 1/8, \dots \{0\} = 1/(2^n) = 1/(2(n+1))$
- verallgemeinerte Zweierpotenzen:
 - $\{p/2\} n | \{p+1/2\} n = (2p+1) / (2(n+1))$, Beispiel: $\{1/2\} | \{3/4\} = 5/8$
- Einfachheitsregel:
 - $\{x\} | y = z$ Falls es die Zahl z gibt, so ist z die **einfachste** Zahl, die strikt zwischen x und y liegt. Je einfacher eine Zahl, desto näher liegt sie an der Wurzel des Zahlenbaums aus Bild 5.

Bild 13.
Der Spielbaum zu Listing 2



zahl der Figuren, deren Konzentration um bestimmte Stellen, deren Bewegungsfreiheit und ähnliches gelten. Bei professionellen Dameprogrammen werden teilweise bis zu 50 Kriterien zur Bewertung einer Position zu Rate gezogen. Somit haben wir also klar erfaßt: Ein Spielprogramm (für ein strategisches Spiel) muß die folgenden Programmteile enthalten: eine Bewertungsfunktion, die es erlaubt, der an die Funktion übergebene Position eine (ganze) Zahl zuzuordnen. Diese Zahl ist um so größer, je besser die Position für den Spieler ist und um so kleiner, je besser die Position für den Gegner ist. In unserem Beispiel (Listing 2 und Bild 13) wird die Bewertung einfach aus einem Feld übernommen. Als weiteres Element wird ein Zuggenerator benötigt, der alle - von der aktuellen Position aus - möglichen Züge erkennt, die wir vornehmen können. Auch hier beschränkt sich unser Beispiel: In jeder Situation sind hier drei Züge möglich.

Wir verfügen für unser Pseudo-Spiel also über einen Zuggenerator und über eine Bewertungsfunktion. Nun müssen wir den Spielbaum durchlaufen, um den für uns günstigsten Zug zu finden. Dazu wird die Funktion Spielbaum aufgerufen. Bevor diese sich rekursiv aufruft, erzeugt der Zuggenerator die sich aus den möglichen Zügen ergebenden neuen Positionen, die als Ausgangspositionen dem Zuggenerator beim rekursiven Aufruf übergeben werden. Nach einer bestimmten,

vorher festgelegten Anzahl von rekursiven Aufrufen wird die Bewertungsfunktion aufgerufen, die der entstandenen Position eine (ganze) Zahl zuordnet. Abhängig von der Anzahl der Rekursionen wären auf dieser untersten Ebene entweder wir oder unser Gegner am Zug gewesen. Entsprechend wird einer Position eine Ebene höher im Baum das Maximum oder das Minimum der Bewertungen der Züge übergeben, die als Optionen dieser Position möglich sind. Eine Ebene höher wiederholt sich dieses Spiel, mit dem Unterschied, daß der andere Spieler am Zug ist; entsprechend wird das Maximum oder das Minimum übergeben.

So wechseln sich auf dem Weg nach oben zur Wurzel des Baumes hin immer die Bildung von Maximum und Minimum auf den jeweiligen Ebenen ab. Um am Ende zu entscheiden, welcher Zug ausgeführt werden soll, muß neben der Bewertung auch die Option, der diese Bewertung zugeordnet wird, mit nach oben übergeben werden.

Die Tatsache, daß sich Bildung von Maximum und Minimum

abwechseln, birgt eine weitere Chance zur Optimierung des Berechnungsverfahrens. Dazu dienen zwei weitere Parameter, die der Funktion Spielbaum übergeben werden. Diese Parameter haben traditionell die Namen »alpha« und »beta«. Hatten wir in der einen Ebene etwa das Maximum zu bilden, und nehmen wir an, daß dieses Maximum, bei der im Spielbaum zuerst durchlaufenen Option dieser Ebene 5 betrug.

In der Ebene darüber soll das Minimum gebildet werden. Ist bei der nächsten Position, von deren Optionen das Maximum gebildet werden muß, der erste Wert größer als 5, so wissen wir, daß das Maximum ebenfalls größer als 5 ist. Da in der Ebene darüber das Minimum gesucht wird und der Wert 5 vorhanden ist, kann dieses Minimum nicht größer als 5 sein - daher können wir uns die Bewertung der restlichen Optionen dieser Position sparen.

Ein analoges Vorgehen gilt natürlich auch, wenn die untere Ebene eine Bildung des Minimums erfordert und die Ebene darüber Maxima bildet. Ent-

sprechend diesen beiden Fällen sind die beiden Parameter »alpha« und »beta« wie in Tabelle 2 angegeben zu setzen. Mit Hilfe dieser beiden Parameter ist es in günstigen Fällen möglich, bis zu 50 Prozent der Optionen aus logischen Gründen auszuschließen und damit Rechenzeit zu sparen.

Die beschriebenen Algorithmen können selbstverständlich auf ein beliebiges Spiel angewendet werden, solange dieses unter logischen Gesichtspunkten untersucht werden kann (Baller- und Action-Spiele lassen sich nur schwer fassen). Sie brauchen dazu nur die genannten Optimierungen in den Zuggenerator Ihres Spiels einbauen.

Auf diese Art versehen Sie Ihr Spiel mit »logischen Überlegungen«, auch ohne Jahrzehnte auf den nächsten Gedanken-gang warten zu müssen. so

Jürgen Singer arbeitet im Institut für Strömungsmechanik an der Universität Erlangen-Nürnberg. Sie können ihn über unsere Verlagsanschrift erreichen (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Herrn Jürgen Singer, Hans-Pinsel-Straße 2, 8013 Haar bei München).

Der Alpha-Beta-Algorithmus

Bedingung	Max-Ebene	Min-Ebene
Aktuelle Bewertung > alpha	alpha = aktuelle Bewertung	--
Aktuelle Bewertung < alpha	--	Abbruch
Aktuelle Bewertung > beta	Abbruch	--
Aktuelle Bewertung < beta	--	beta = aktuelle Bewertung

Tabelle 2. Mit dem Alpha-Beta-Algorithmus optimieren Sie die Suche nach dem optimalen Zug

Literatur zum Thema

Zur Spieltheorie gibt es inzwischen relativ umfassende Fachliteratur, wir haben einige für Sie ausgewählt:

1. Mathematik	2. Computer
<p>John H. Conway; Über Zahlen und Spiele; Vieweg Verlag 1983 Berlecamp, Conway, Guy; Gewinnen – Strategien für mathematische Spiele, – Band 1: Von der Pike auf, Vieweg Verlag 1985 – Band 2: Bäumchen-wechsle-dich, Vieweg Verlag 1986 – Band 3: Fallstudien, Vieweg Verlag 1986 – Band 4: Solitairespiele, Vieweg Verlag 1985</p>	<p>Stefan Berchtold; Dame mit Strategie; in C'T-Magazin für Computertechnik, 8/89 Seite 114ff, Heise Verlag. A. K. Dewdney; Meisterliche Dameprogramme; in Verständliche Forschung: Computer-Kurzweil; Spektrum der Wissenschaft Verlagsgesellschaft 1988. D. N. L. Levy; Computergames I; Springer Verlag. D. N. L. Levy; Computergames II; Springer Verlag.</p>

```

/*-----
file: spielbaum.c           language: Lattice C 5.02
status: released, in use    system: AmigaDos, Kick 1.3
compilation: lc -Lm spielbaum.c version: 1.0 (25-Aug-89)

(c) 1989 Jürgen K. Singer, Nürnberg
-----

Berechnung des Spielbaumes für das Spiel >>3 * MAXX Springer<<

-----*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAXX 4

typedef struct
{
    int lx ;
    int ly ;
    int rx ;
    int ry ;
} Position ; /* Position ist gegeben durch Spalte und Reihe des Stei-
nes */

/* von >>Links<< und >>Rechts<< */

/*----- Prototypen -----*/
double simplest( double l, double r ) ;
double Bewertung( Position p ) ;
/*-----*/

int err = 0 ;

double simplest( double l, double r )
/*-----
Berechnet die >>einfachste<< Zahl x, für die gilt l < x < r.
Dabei ist >>einfach<< definiert als >>möglichst nahe der
Wurzel im Zahlenbaum<<.
Falls linker Rand rechts vom rechten Rand, gibt es keine derartige
Zahl. Die zugehörigen theoretischen Betrachtungen finden sich in:
Berlekamp, Conway, Guy; Gewinnen – Strategien für mathematische
Spiele; Band 1: Von der Pike auf, Vieweg Verlag 1985.

Argumente:                linker und rechter Intervallrand
Funktionswert: a) falls err = 0: einfachste Zahl im Intervall
               b) falls err = -1: 0, da ungültige Argumente
-----*/
{
    double x ;
    double a ;

    if( l >= r ) /* Es gibt keine derartige Zahl x, setze Fehlercode */
    {
        err = -1 ;
        return 0.0 ;
    }
    a = 0.5 ;
    if( 0 <= l ) /* 0 <= l < r */
        for( x = floor(l+1) ; x >= r ; a /= 2.0 )
            x -= a ;

```

```

else if( r <= 0 ) /* l < r <= 0 */
    for( x = ceil(r-1) ; x <= l ; a /= 2.0 )
        x += a ;
else /* r < 0 < l */
    x = 0.0 ;
return x ;
} /* simplest */

double Bewertung( Position p )
/*-----
Bewerte eine gegebene Position. Diese Funktion ruft sich selbst
solange auf, bis eine einfach zu bewertende Position erreicht ist;
dies ist hier der Fall, wenn kein Sprung mehr möglich ist.
-----*/
{
    double b ;
    Position pl ; /* Option für >>Links<< */
    Position pr ; /* Option für >>Rechts<< */

    if( p.lx > p.rx ) /* kein Sprung mehr möglich: Bewertung ist nun */
        b = MAXX + 1 - p.lx - p.rx ; /* einfach Differenz der Züge */
    else /* Noch keine einfache Situation erreicht */
    {
        pl = pr = p ;
        if( (p.lx == p.rx) && (p.ly == (p.ry+1)) && (p.ry > 1) )
            p.ly -= 2 ; /* Sprung möglich */
        else
        {
            pl.lx += 1 ; /* >>Links<< zieht ein Feld nach Ost */
            pr.rx -= 1 ; /* >>Rechts<< zieht ein Feld nach West */
            b = simplest( Bewertung(pl), Bewertung(pr) ) ;
        }
    }
    return b ;
} /* Bewertung */

#define FMTSTR "Position Links: %d, Rechts: %d, Bewertung %lf\n"

void main( int argc, char *argv[])
/*-----
Berechne den Spielbaum für 3 * MAXX Springer. Argument 1 = Position
von >>Links<< in Reihe 3 (1 <= lx <= MAXX), Argument 2 =
Position von >>Rechts<< in Reihe 2 (1 <= rx <= MAXX).
-----*/
{
    Position p ;

    if( argc != 3 ) exit(-1) ;
    p.ly = 3 ; /* >>Links<< beginnt in Reihe 3 */
    p.ry = 2 ; /* >>Rechts<< beginnt in Reihe 2 */
    p.lx = atoi(argv[1]) ; /* Spalte von >>Links<< */
    p.rx = atoi(argv[2]) ; /* Spalte von >>Rechts<< */
    printf( FMTSTR, p.lx, p.rx, Bewertung(p) ) ;
    if( err ) printf( "Bewertung ungültig!\n" ) ;
} /* main */

/* Listing 1: Berechnung des Spielbaumes zu Bild 9 */

```

Listing 1.
Die Berechnung eines
Spielbaums


```

/*-----
file:  alphabeta.c          language: Lattice C 5.02
status: released           system: AmigaDos, Kick 1.3
compile: lc -Lc alphabeta.c version: 1.0 (03-Sep-89)

(c) 1989 Jürgen K. Singer, Nürnberg
-----

Berechnung des Spielbaumes für ein Pseudo-Spiel mit Hilfe des
alpha - beta - Algorithmus nach dem Mini-Max-Prinzip
-----*/

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

+#if !defined(NIL)
#define NIL 0L
#endif

/* maximale Rekursionstiefe = Spielstufe */
#define MAXREKURS 3

/* soll Minimum oder Maximum gebildet werden? */
#define MINIMAX(x) ( (x) & 1 )

typedef int Stellung ; /* Figurenkonstellation */

struct option {
    Stellung s ; /* Figurenkonstellation */
    int      b ; /* Bewertung */
} ;

typedef struct option Option ; /* mögliche Züge */

/* Prototypen */
Option Spielbaum( Option alt, int RekTiefe, int alpha, int beta ) ;
int Zuggenerator( Option alt, Option **neu ) ;
void Bewertung( Option *alt ) ;

void main(int argc, char *argv[])
{
    Option Start ;
    Option Zug ;
    int alpha = INT_MIN ;
    int beta  = INT_MAX ;

    Zug = Spielbaum( Start, 0, alpha, beta ) ;
    printf( "Optimaler Zug: Stellung: %d Bewertung: %d\n", Zug.s,
    Zug.b ) ;
}

/*----- Dummy - Funktionen -----*/
int Zuggenerator( Option Start, Option **neu)
{
    static Stellung nr = 0 ;
    int Anzahl ;
    int i ;

    Anzahl = 3 ;
    *neu = (Option *) malloc( Anzahl * sizeof(Option) ) ;
    for( i = 0 ; i < Anzahl ; i++ )
    {
        (*neu + i)->s = ++nr ;
        (*neu + i)->b = 0 ;
    }
    return Anzahl ;
}

void Bewertung( Option *o )
{
    /* Dimension = Gesamtzahl der möglichen Züge */
    static int b[/* (summe i=0 ... MAXREKURS (j1)) - 1 */] =

```

```

{
    -6,
    -5, -4, -3,
    -2, -1, 0,
    25, 26, 27, 28, 29, 30, 31, 32, 33,
    10, 11, 12,
    13, 14, 15, 16, 17, 18, 19, 20, 21,
    22, 23, 24,
    1, 2, 3, 4, 5, 6, 7, 8, 9
} ;

o->b = b[o->s] ;
}
/*-----*/

Option Spielbaum( Option alt, int RekTiefe, int alpha, int beta)
{
    Option *neu = NIL ;
    Option Zug ;
    Option Bester ;
    int Zugzahl ;
    int i ;

    if( RekTiefe < MAXREKURS ) /* Noch nicht auf unterster Ebene */
    {
        if( MINIMAX(RekTiefe) ) /* min-Ebene */
            Bester.b = INT_MAX ;
        else /* max-Ebene */
            Bester.b = INT_MIN ;
        Zugzahl = Zuggenerator( alt, &neu ) ; /* Finde alle mögl. Züge */
        for( i = 0 ; i < Zugzahl ; i++ ) /* Finde zu jedem möglichen
        Zug */
        {
            /* dessen Optionen */
            Zug = Spielbaum( neu[i], RekTiefe + 1, alpha, beta ) ;
            if( MINIMAX(RekTiefe) ) /* min - Ebene */
            {
                if( Bester.b > Zug.b ) /* neuer bester Zug gefunden */
                {
                    Bester.b = Zug.b ;
                    Bester.s = (neu + i)->s ;
                    beta = Bester.b ;
                }
                if( alpha > Zug.b ) break ; /*Züge können nur schlechter*/
                /* sein: Abbruch */
            }
            else /* max - Ebene */
            {
                if( Bester.b < Zug.b ) /* neuer bester Zug gefunden */
                {
                    Bester.b = Zug.b ;
                    Bester.s = (neu + i)->s ;
                    alpha = Bester.b ;
                }
                if( beta < Zug.b ) break ; /* nur schlechtere Züge:
                Abbruch */
            }
        }
        free( neu ) ; /* vergessen der nicht benutzten Optionen */
        return( Bester ) ;
    }
    else /* unterste Ebene erreicht: Bewerte die entstandene Position */
    {
        Bewertung( &alt ) ;
        return( alt ) ;
    }
}

```

Listing 2.
Die Umsetzung des
Alpha-Beta-Algorithmus in C

Sprites unter der Lupe

COLA, FANTA, SPRITE

Von Arno Gölzer

Was wäre ein schnelles Actionspiel ohne Sprites? Richtig, ein fußkranken Actionspiel. Diese kleinen Objekte sind das Salz in der Suppe des Spieleprogrammierers.

Sprites sind kleine, frei bewegliche Grafikobjekte, die bevorzugt in Spielen eingesetzt werden. Der Amiga hält einige interessante Routinen bereit, die das Arbeiten mit diesen »guten Geistern« des Spielers leichter machen. Auf eine Programmierung der Hardware können Sie daher in den meisten Fällen verzichten – im Gegensatz zu gewissen Steinzeitcomputern, denen der Programmierer nur unter verschärfter POKerei eine dieser Miniaturgrafiken entlocken konnte.

Spiele sind jedoch nicht mehr die einzigen Einsatzmöglichkeiten für Sprites. Längst nutzt man die Vorteile auch in Text- oder Grafikprogrammen.

Auch die Arbeit auf dem Amiga, speziell mit der Workbench, ist ohne Sprites nur schwer vorstellbar: Versuchen Sie einmal, kurze Zeit ohne Mauszeiger auszukommen. Schnell stellen sich Entzugserscheinungen ein. Moment mal... Sprites... Mauszeiger??? Sie vermuten richtig. Der Mauszeiger ist nichts anderes als einer der acht zur Verfügung stehenden Hardwaresprites des Amiga-Grafiksystems. Sprites sind von 0 bis 7 durchnummeriert, dabei ist Nummer 0 für den Pointer reserviert.

Natürlich wissen Sie, daß Sie ihn beispielsweise mit dem Preferences-Programm beliebig verändern können. Aber auch vom eigenen Programm aus ist das sehr einfach möglich und – gerade für Spiele – durchaus interessant. Unser Beispielprogramm »HotBottom« (Listing 2) macht dafür von der Betriebssystem-Funktion `SetPointer()` Gebrauch:

```
SetPointer(&Window,
&data[0],h,b,x,y);
```

Die Funktion erlaubt die Definition eines eigenen Mauszeigers für jedes Fenster. Dieser

Spiele ohne Sprites schmecken wie Oktoberfest ohne Bier, wie Sommer ohne Sonne, wie Kaugummi ohne Zucker, wie...

Zeiger ist immer dann sichtbar, wenn sich das jeweilige Fenster im aktivierten Zustand befindet. Die Parameter dieser Funktion von links nach rechts: a) die Adresse des Fensters, in dem der neue Pointer sichtbar sein soll (`&Window`), b) die Adresse des Datenblocks, der die Pointerdaten enthält (`&data`), c) die Höhe des Pointers (`h`), d) die auf maximal 16 Pixel beschränkte Breite (`b`) und e/f) die x/y Koordinaten (`x,y`) des sogenannten Hot-Spots, bezüglich der linken oberen Sprite-(Pointer-)Ecke.

Register für Farbe			
Sprites	1	2	3
0 und 1	17	18	19
2 und 3	21	22	23
4 und 5	25	26	27
6 und 7	29	30	31

Tabelle 1. Die Farbgisternnummern der Sprites

Als Hot-Spot bezeichnet man die »empfindliche« Stelle im Mauszeiger, mit der Sie beispielsweise ein Workbench-Objekt auswählen können. Unklar hierbei ist nur noch das Aussehen des Datenblocks, dessen Adresse die Funktion als zweiten Parameter erwartet. Es handelt sich um ein UWORD-Array im Chip-Memory, dessen beiden ersten und beiden letzten Einträge gleich 0 sein müssen:

```
UWORD Data[] =
{ 0, 0, ... 0, 0 };
```

Zwischen diesen für das System reservierten Array-Elementen befindet sich die »Beschreibung« des Pointersprites. Wie sieht diese Beschreibung aus? Ein Hardware-sprite besteht aus maximal drei Farben. Aus Tabelle 1 geht hervor, daß immer zwei Sprites (ein

Spritepaar) auf die gleichen Farbgistern zugreifen. Die Farben des Pointers, es handelt sich ja um Sprite Nummer 0, finden Sie in den Farbgistern 17 bis 19.

Das oben angesprochene Daten-Array enthält als Beschreibung für jede Zeile des Pointers zwei hintereinander liegende, 16 Bit breite Werte. Stellen Sie sich die beiden Werte in binärer Form untereinander vor, etwa so:

```
%1100101001110011
%1001111001100110
```

Lesen Sie die senkrecht übereinanderliegenden Bits von unten nach oben. Liegen zwei Nullen (00) übereinander, so ist das Sprite an dieser Stelle der Zeile transparent. Lesen Sie (von unten nach oben) die Bits 01, so ist dieser Punkt in der Farbe des ersten (%01=1) zuständigen Registers gefärbt (siehe Tabelle 1). Die Bitkombi-

des Farbgisterns Nummer 19 (%11) gefärbt.

Der drei Zeilen hohe Pointer enthält keine transparente (%00) Zeile. Vor und hinter den für die drei Zeilen notwendigen sechs Elementen sind jeweils die beiden 0-Einträge aufgeführt. Beachten Sie bitte, daß alle Spritedaten, also auch die des Pointers, im Chip-Memory, also den unteren 512 KByte des Arbeitsspeichers, liegen müssen. In Listing 2 kopieren wir die Daten mit der `CopyMem()`-Funktion »nach unten«. Ein UWORD-Datenblock, wie soeben beschrieben, muß auch bei der Definition eines eigenen Sprites vereinbart werden. Wichtig ist dabei, daß Sie für jedes Sprite einen eigenen Datenblock vereinbaren, auch wenn Sie mehrere gleich aussehende Sprites darstellen möchten.

Zu dem Daten-Array kommt bei der Sprite-Definition die Festlegung der sogenannten SimpleSprite-Struktur (Listing 1) hinzu, deren Elemente die Form und die Position des Sprites beschreiben.

Als erste Komponente finden wir dort die Adresse des Spritedatenblocks: »`poscldata`«. Die Höhe des Sprites tragen wir in »`height`« ein – wie erwähnt dürfen Sprites beliebig hoch sein. Eine Variable für die Breite ist in der Struktur nicht enthalten – Hardware-Sprites haben ja eine fest definierte Breite von 16 Pixel. Für die Arbeit mit breiteren Objekten muß man entweder mehrere Sprites zusammenfügen oder aber auf gänzlich andere Objekte, z.B. BOBs (»Blitter-Objects«) zurückgreifen. Die Sprite-Dimensionen sind von der gewählten Display-Auflösung unabhängig. Die Größe eines Sprite-Pixels entspricht immer der Größe eines Bildschirm-Pixels im Lores-Modus.

Die SimpleSprite-Struktur-Variablen `x` und `y` enthalten die Position des Sprites. Sie werden ständig aktualisiert, so daß Sie hier jederzeit die Position eines Sprites erfahren können. Die letzte Komponente namens »`num`« steht für die Sprite-Nummer (siehe unten).

Die Beschreibung des Sprites ist somit abgeschlossen. Vor dessen Darstellung muß aber noch der Screen, oder all-

nation 10 verweist auf das zweite (%10=2) und schließlich die Kombination 11 auf das dritte (%11=3) Farbgistern.

Schauen wir uns die Daten des Pointers in Listing 2 an. Sie finden sie gleich am Anfang des Listings im UWORD-Array `PointerDat[]`. Die erste Zeile, bestehend aus 0xFFFF und 0, ist komplett in der Farbe des Registers 17 gefärbt:

```
%1111111111111111 =
0xFFFF
%0000000000000000 =
0x0000
```

Denn in jeder Spalte finden wir, von unten nach oben gelesen, die Bitkombination 01. Binär 01 entspricht dezimal 1, das erste Farbgistern des Sprites 0 ist 17. Die zweite Zeile, 0 und 0xFFFF, ist in der Farbe des Registers 18 (%10) und die dritte Zeile, zweimal 0xFFFF, in der

gemein das Display, für die Verwendung von Sprites vorbereitet werden. Dies geschieht durch das Setzen des View-Modus Flags »SPRITES«. Wie aus Listing 2 hervorgeht, ist der Workbenchscreen bereits auf Sprites vorbereitet, das Flag muß nicht explizit gesetzt werden.

Nachdem alle Vorbereitungen abgeschlossen sind, meldet GetSprite() die Kleingrafik an:

```
SpriteNr=GetSprite
(&Sprite,nr);
```

GetSprite() verlangt als Parameter einen Zeiger auf die SimpleSprite-Struktur und die gewünschte Spritenummer. Ist das Sprite mit der angegebenen Nummer noch frei, so entspricht der Returnwert der Funktion dieser Nummer. GetSprite() kehrt mit dem Wert -1 zurück, wenn das Sprite bereits reserviert war. Bei der Angabe von -1 als Spritenummer-Parameter, liefert die Funktion die Nummer des ersten freien Sprites.

Wie weiter oben bereits angedeutet, können wir auf dem Amiga bis zu acht Hardware-Sprites, mit den Nummern 0 bis 7, definieren. Bei der Wahl der Spritenummer sollten Sie folgendes beachten: Wenn die Sprites so auf den Bildschirm positioniert sind, daß sie sich teilweise verdecken, ist das Sprite mit einer niedrigeren

Stimmung der Spritenummer kennen. Das Gegenstück zu GetSprite() ist FreeSprite (SpriteNummer). Diese Funktion löscht das Sprite mit der angegebenen Nummer vom Display und meldet es beim System wieder ab. Somit ist es für die Neuverwendung – eventuell von einem anderen Programm – wieder frei. Der Parameter ist die von GetSprite() gelieferte Spritenummer.

Möchten Sie nicht nur ein, sondern alle Sprites auf einen Schlag verschwinden lassen, hilft Ihnen das Makro OFF_SPRITE, indem es einfach den DMA-Kanal abschaltet. Das Gegenstück ON_SPRITE stellt den Urzustand wieder her. Nach dem Aufruf der GetSprite()-Funktion ist das Sprite noch nicht sichtbar. Für die Darstellung und die Bewegung der Sprites ist

```
MoveSprite
(&VP,&SSprite,x,y);
```

zuständig. Der erste Parameter, die Adresse »VP«, zeigt auf die ViewPort-Struktur des Displays. Pointer »SSprite« verweist auf die SimpleSprite-Struktur, die das Sprite beschreibt. Die restlichen Parameter stehen für die X/Y-Position der linken oberen Sprite-Ecke, bezüglich der linken oberen Ecke des Displays. Mit dieser Funktion kommt auch endlich – im wahrsten Sinne des Wortes – Bewegung ins Spiel. Setzen Sie MoveSprite()

denn die Bewegungen der Sprites erfolgen in sehr hoher Geschwindigkeit.

Manchmal ist es notwendig – beispielsweise für eine Explosion – das Aussehen eines Sprites zu verändern. Glücklicherweise bleibt uns hierfür die umständliche Sequenz des Freigebens und der Neudefinition erspart:

```
ChangeSprite(&VP,
&SSprite,&data[0]);
```

Diese Funktion verändert das Aussehen des Sprites der von der SimpleSprite-Struktur, auf die der Pointer »SSprite« verweist, beschrieben wird. Der

Sprites. Die Farben erreichen Sie über die Farbregister 17 bis 31. Die beiden oberen der vier Bits werden von dem Sprite des Paares mit der höheren Nummer bestimmt.

Die ganze Farbenpracht wird jedoch erst nach der Definition eines Sprite-Paares als Attached-Sprite sichtbar. Setzen Sie hierzu in beiden Daten-Arrays, im zweiten UWORD, das Attached-Bit:

```
Data[1]=SPRITE_ATTACHED;
```

Nachdem Sie Aussehen und Bewegung der Sprites beliebig beeinflussen können, haben

Bit	Ereignis (Kollision von):
15	ungenutzt
14	Sprite 4 (oder 5) mit Sprite 6 (oder 7)
13	Sprite 2 (oder 3) mit Sprite 6 (oder 7)
12	Sprite 2 (oder 3) mit Sprite 4 (oder 5)
11	Sprite 0 (oder 1) mit Sprite 6 (oder 7)
10	Sprite 0 (oder 1) mit Sprite 4 (oder 5)
9	Sprite 0 (oder 1) mit Sprite 2 (oder 3)
8	BitPlane (gerade Nummer) mit Sprite 6 (oder 7)
7	BitPlane (gerade Nummer) mit Sprite 4 (oder 5)
6	BitPlane (gerade Nummer) mit Sprite 2 (oder 3)
5	BitPlane (gerade Nummer) mit Sprite 0 (oder 1)
4	BitPlane (ungerade Nummer) mit Sprite 6 (oder 7)
3	BitPlane (ungerade Nummer) mit Sprite 4 (oder 5)
2	BitPlane (ungerade Nummer) mit Sprite 2 (oder 3)
1	BitPlane (ungerade Nummer) mit Sprite 0 (oder 1)
0	BitPlane (gerade Nummer) mit BitPlane (ungerade Nummer)

Tabelle 3. Der Inhalt des Registers »CLXDAT« berichtet über das Kollisionsereignis

dritte Parameter zeigt auf das Array mit den neuen Spritedaten. Auch hier denken Sie bitte daran: Daten im Chip-Memory und je zwei UWORD-Variablen vor und nach den eigentlichen Datenblöcken einfügen. Wie aus Tabelle 1 ersichtlich ist, greifen immer zwei Sprites auf die gleichen Farbregister zu. Ein solches Spritepaar kann man zu einem einzigen Sprite verbinden, einem sogenannten Attached-Sprite. Der Vorteil: Attached-Sprites können in 15 Farben dargestellt werden. Wie bitte, zweimal drei Farben (die zudem noch gleich sind) ergeben plötzlich 15 verschiedene Farben? Genau. Beide Sprites müssen dazu an die gleiche Bildschirmposition gebracht werden. Da eine Zeile eines Sprites durch zwei UWORDS dargestellt wird, liegen bei zwei positionsgleichen Sprites vier UWORDS übereinander. Mit vier übereinanderliegenden Bit, die einem Pixel eines Attached-Sprites entsprechen, können $2^4 = 16$ Zustände dargestellt werden.

Das ergeben 15 Farben (plus eine transparente) für Attached-

Sie sicher schon die ersten Ideen für Ihr neuestes Amiga-Spiel notiert. Vielleicht handelt es sich um ein Geschicklichkeitsspiel wie dieses: Der Spieler hat die Aufgabe, durch einen geschickten Wurf mit einem Pfeil, möglichst mittig eine mehrfarbige Zielscheibe mit konzentrischen Kreisen zu treffen.

Mit den bisher erhaltenen Informationen läßt sich dieses Spiel relativ problemlos realisieren – bis zur Auswertung. Wie stellt man fest, in welchem Farbring sich der Pfeil befindet? Oder allgemeiner definiert: Wie stellt man die Kollision eines Hardware-Sprites mit dem Hintergrund oder einem anderen Sprite fest?

Die Kollisionsabfrage mit dem Hintergrund setzen wir mit der Grafik-Funktion ReadPixel() um. Für unser Beispiel bedeutet das, wir müssen mit ReadPixel() die Farbe unter dem Pfeil ermitteln und den Wurf entsprechend bewerten. Das Zusammenprallen zweier Sprites könnten wir über deren Positionen ermitteln. Ein Beispiel: Möchten Sie kontrollie-

Bit	Melde, wenn gesetzt, Kollision mit:	Melde, wenn gelöscht, Kollision mit:
15	Sprite (6 und) 7	Sprite 6
14	Sprite (4 und) 5	Sprite 4
13	Sprite (2 und) 3	Sprite 2
12	Sprite (0 und) 1	Sprite 0
11	Plane 6	-
10	Plane 5	-
9	Plane 4	-
8	Plane 3	-
7	Plane 2	-
6	Plane 1	-
5	gesetztem Bit der Plane 6	gelöschtem Bit der Plane 6
4	gesetztem Bit der Plane 5	gelöschtem Bit der Plane 5
3	gesetztem Bit der Plane 4	gelöschtem Bit der Plane 4
2	gesetztem Bit der Plane 3	gelöschtem Bit der Plane 3
1	gesetztem Bit der Plane 2	gelöschtem Bit der Plane 2
0	gesetztem Bit der Plane 1	gelöschtem Bit der Plane 1

Tabelle 2. Über das Register »CLXCON« bestimmen Sie, welche Sprite-Kollisionen Sie gemeldet wünschen

Nummer immer vor dem mit einer höheren Nummer zu sehen.

Diese Prioritäten sind in der Hardware festgelegt und somit nicht veränderbar. Weiter unten, bei der Besprechung der Kollisionserkennung, lernen Sie weitere Kriterien für die Be-

in den Rumpf einer Schleife und ändern Sie bei jedem Durchgang die Werte x und y. Positive x-Werte lassen das Sprite nach rechts wandern, positive y-Werte veranlassen es zu einer Bewegung nach unten. Fügen Sie jedoch beim ersten Testen eine Warteschleife ein,

ren, ob zwei gleichgroße Kugeln kollidiert sind, so müssen Sie ständig prüfen, ob die Mittelpunkt der Kugeln sich näher als der Kugeldurchmesser sind.

Beide Methoden haben den Vorteil, daß sie sehr einfach zu formulieren sind. Sie haben aber beide auch einen Nachteil: die Kollisionskontrolle kostet viel Zeit. Da sie bei jeder Sprite-Bewegung durchgeführt werden muß, kann sich dies, besonders beim Arbeiten mit vielen Sprites, durch eine unregelmäßige Bewegung bemerkbar machen.

Die optimale Lösung für beide Fälle machte den Zusatz »in den meisten Fällen« in der Einleitung nötig, denn Abhilfe schaffen hier zwei Hardware-Register:

»clxcon« und »clxdat«. Sie erreichen diese Register über die im Headerfile »hardware/custom.h« definierte Custom-Struktur. Sie enthält namentlich die Hardware-Register, die sich im Speicher ab der Adresse 0xDFF000 befinden:

```
custom.clxcon=0x41;
```

Register clxcon ist für die Kollisions-Kontrolle verantwortlich. Hier legen Sie durch das Setzen einzelner Bits fest, welche Kollisionen registriert werden sollen.

Tabelle 1 zeigt die Zusammengehörigkeit zweier Sprites. Auch bei der Kollisionsabfrage werden immer zwei Sprites zu einem Spritopaar zusammengefaßt. Die Sprites mit der geraden (und dadurch kleineren) Nummer werden automatisch überwacht. Möchten Sie aber auch Informationen über die anderen Sprites, es sind dies die Sprites mit den Nummern 1, 3, 5 und 7, so müssen Sie dies über die oberen 4 Bit des clxcon-Registers mitteilen (siehe auch Tabelle 2):

clxcon-Bit:	12	13	14
Sprite-Nr.:	1	3	5
clxcon-Bit:	15		
Sprite-Nr.:	7		

Durch das Setzen der jeweiligen Bits bestimmen Sie, welche Sprites Sie kontrollieren möchten. Mit den Bits 6 bis 11 legen Sie fest, welche BitPlanes Sie testen wollen:

clxcon-Bit:	6	7	8
Plane-Nr.:	1	2	3
clxcon-Bit:	9	10	11
Plane-Nr.:	4	5	6

Mittels der restlichen Bits teilen Sie mit, ob Sie eine Meldung beim Zusammentreffen mit einem gesetzten Plane-Bit oder einem gelöschten Plane-Bit wünschen.

clxcon-Bit:	0	1	2
Plane-Nr.:	1	2	3
clxcon-Bit:	3	4	5
Plane-Nr.:	4	5	6

Ein gesetztes Bit bedeutet: Meldung einer Sprite-Kollision mit einem gesetzten Bit in der entsprechenden BitPlane. Ein Beispiel macht den Sachverhalt verständlicher: Wir arbeiten in einem Screen mit zwei Planes und wünschen eine Meldung, wenn Sprite 6 mit der Farbe aus dem vierten Farbregeister zusammenstößt.

Sprite 6 ist der Sprite mit der kleineren Nummer aus dem Sprite-Paar Nummer 4, demnach interessieren uns die oberen 4 clxcon-Bit nicht. Da das erste Farbregeister als Nummer 0 geführt wird, hat das vierte Farbregeister die Farbregeister-Nummer 3 (Farbregeister-Nr.: 0, 1, 2, 3, 4 ... 31). Die entsprechende Farbe erscheint daher auf dem Bildschirm, wenn die Bits der ersten Plane und die Bits der zweiten Plane gesetzt sind, denn 3 entspricht binär %11. Daher muß clxcon folgendermaßen belegt werden:

clxcon-Bit:	15	14	13
Koll.-Wert:	0	0	0
clxcon-Bit:	12	11	10
Koll.-Wert:	0	0	0
clxcon-Bit:	9	8	7
Koll.-Wert:	0	0	1
clxcon-Bit:	6	5	4
Koll.-Wert:	1	0	0
clxcon-Bit:	3	2	1
Koll.-Wert:	0	0	1
clxcon-Bit:	0		
Koll.-Wert:	1		

Ein weiteres Beispiel: Sprite 3 soll in einem Screen mit vier Planes an allen Punkten, die in der Farbe des Farbregeisters Nummer 2 gefärbt sind, zusammenstoßen.

Die Überwachung des Sprites mit der ungeraden Nummer 3 müssen wir anmelden, bevor wir es abfragen können. Weiterhin haben wir vier Planes – eine Kollision soll aber nur gemeldet werden, wenn nur die Bits in der zweiten Plane gesetzt sind (Farbregeisternummer 2 entspricht binär %10). Um Falschinformationen zu vermeiden, verlangen wir ausschließlich Meldungen über die zweite Bitplane:

clxcon-Bit:	15	14	13
Koll.-Wert:	0	0	1
clxcon-Bit:	12	11	10
Koll.-Wert:	0	0	0
clxcon-Bit:	9	8	7
Koll.-Wert:	0	0	1
clxcon-Bit:	6	5	4
Koll.-Wert:	0	0	0
clxcon-Bit:	3	2	1
Koll.-Wert:	0	0	1
clxcon-Bit:	0		
Koll.-Wert:	0		

Das Ergebnis der Kollisionsmeldung entnehmen wir den unteren 14 Bit des Registers clxdat (Bit 15 ist ungenutzt). Die Bedeutung der Bits entnehmen Sie Tabelle 3.

Beachten Sie beim Auslesen des Registers, daß dieses nach dem Lesen sofort gelöscht und erst beim nächsten Start des Elektronenstrahls neu beschrieben wird.

Betrachten wir uns noch einmal unsere beiden Beispiele: Welche Werte sind bei einer Kollision dem clxdat-Register zu entnehmen? Beispiel 1: Sprite 6 soll mit den beiden untersten Planes zusammenstoßen:

clxcon-Bit:	15	14	13
BitMeldung:	0	0	0
clxcon-Bit:	12	11	10
BitMeldung:	0	0	0
clxcon-Bit:	9	8	7
BitMeldung:	0	1	0
clxcon-Bit:	6	5	4
BitMeldung:	0	0	1
clxcon-Bit:	3	2	1
BitMeldung:	0	0	0
clxcon-Bit:	0		
BitMeldung:	0		

Die Meldung besagt, daß Sprite 6 mit einer geraden BitPlane (Plane 2, 4, 6) und mit einer ungeraden BitPlane (Plane 1, 3, 5) zusammengetroffen ist. Da wir in clxcon nur Informationen über die beiden untersten BitPlanes angefordert haben, können wir sicher sein, daß eine Kollision stattgefunden hat. Wir erfragen sie folgendermaßen:

```
nr=custom.clxdat;
if((nr&0x110)==0x110){
...
Kollision();
...
}
```

Die Lösung für das zweite Beispiel – Sprite 3 soll mit der zweiten BitPlane kollidieren (überlegen Sie zunächst selbst, bevor Sie den Kasten betrachten):

clxcon-Bit:	15	14	13
BitMeldung:	0	0	0
clxcon-Bit:	12	11	10
BitMeldung:	0	0	0
clxcon-Bit:	9	8	7
BitMeldung:	0	0	0
clxcon-Bit:	6	5	4
BitMeldung:	1	0	0
clxcon-Bit:	3	2	1
BitMeldung:	0	0	0
clxcon-Bit:	0		
BitMeldung:	0		

Die Meldung lautet: Sprite 3 (oder 2) ist mit einer geraden BitPlane zusammengestoßen:

```
...
if((nr&0x40)==0x40){
...
}
```

Sie sehen, hier ist die Meldung nicht eindeutig (es sei denn, es handelt sich um ein Attached-Sprite)! Sowohl Sprite 2 als auch Sprite 3 bringen bei dem eingestellten Kontroll-Wert die gleiche Meldung. Um mögliche Fehler zu vermeiden, müssen Sie bereits beim Anmelden der Sprites mit GetSprite() die Numerierung beachten.

Nach so viel Theorie ist etwas Entspannung bei einem Spielchen nötig. Testen Sie doch gleich einmal das Demoprogramm für Pointer & Sprites, »HotBottom« (Listing 2).

Ein Sprite, in Form eines Balls, bewegt sich darin innerhalb eines Balkens über das Spielfeld. Er reflektiert an den Seiten des Feldes und an kreisförmigen Hindernissen. Die Hindernisse werden »angenagt«, wobei sich die Flugrichtung des Balls verändert.

Ihre Aufgabe ist es, dafür zu sorgen, daß der Ball den Spielfeldboden nicht berührt. Als Hilfsmittel steht Ihnen dafür der als Break-Out-Schläger modifizierte Mauszeiger zur Verfügung. Dieser ist gemeinweise nur sichtbar, wenn sich der Ball in Bodennähe befindet – und auch in der Titelleiste, falls Sie das Spiel vorzeitig beenden möchten.

Springt der Ball trotz aller Vorsicht doch einmal auf den Boden, so wächst die Game-Over-Säule am linken Spielfeldrand ein wenig höher. Säule und Balken färben sich rot, um Sie zu warnen, wenn das Spielende kurz bevorsteht.

Einige Punkte des Listings sollen nicht unerwähnt bleiben. Betrachten wir uns die Funktion OpenW(). Hier werden die Libraries und das Window geöffnet, die Sprite-Farben festgelegt, Speicher für Chip-Memory angefordert und belegt und die

SimpleSprite-Struktur mit den noch fehlenden Informationen über die Lage der Spritedaten ergänzt. Hier vereinbaren wir auch Speicher für den »Dummy-Pointer« – das ist der unsichtbare Mauszeiger, der aktiviert wird, sobald sich der Ball unter einer bestimmten Position befindet.

CloseW() gibt alle angeforderten Speicherbereiche wieder ans System zurück. In der Hauptfunktion main() teilen wir CLXCON mit, daß wir Informationen über die Kollision mit gesetzten Bits der ersten BitPlane wünschen.

Diese Kollision findet statt, wenn unser Sprite mit einer Farbe des zweiten oder des vierten Farbregisters zusammentrifft – bei beiden ist das Bit der ersten Plane gesetzt. Wir testen demnach CLXDAT auf den Wert 4 (=0001) – also auf Kollision des Sprites 2 mit einer ungeraden BitPlane.

Weiterhin erwarten wir den Wert 0x200. Er wird beim Zusammentreffen des Schlägers mit dem Ball gesetzt (Kollision Sprite 0 mit Sprite 2 = CLXDAT-Bit 9 = 0x200). Sie finden die Abfrage am Ende des Quellcodes.

MoveSprite() bewegt den Ball (siehe Anfang der While-Schleife in main()), indem zu der aktuellen X- und Y-Position des Sprites ein entsprechender Wert addiert wird. Die Kollision des Spielfeldrandes überprüfen wir über die Sprite-Position. Das x- bzw. y-Inkrement wird beim Zusammentreffen negiert, so scheint der Ball am Spielfeldrand zu reflektieren.

Stellen Sie doch einige Versuche mit dem Programm an – besonders interessant (und auch etwas schwieriger) wird das Spiel nach dem Entfernen des MakeCList()-Aufrufs in main(). Am besten gleich testen – viel Spaß dabei! so

```
REGISTER SHORT i,j,x,y,p=0;

Points=0;
XPos=20+VBeamPos();
YPos=20;
SetRast(RP,2);
RefreshWindowFrame(win);
for(i=0;i<10;i++){
    x=i*64;
    p=p*0:1;
    SetAfPt(RP,Pattern+p,3*p);
    for(j=1;j<6;j++){
        y=j*40;
        DrawC(1,30+x,y,5);
        if(i<9) DrawC(3,64+x,y+20,5);
    }
    Score();
}
VOID MakeCList(y)
    SHORT y;
{
    if(GAlert) NewCol=((NewCol+0x100)>0xFO0)?0:NewCol+0x10
0;
    else NewCol=0xFF0;
    CINIT(UCList,5);
    CWAIT(UCList,y-3,0);
    CMOVE(UCList,custom.color[2],NewCol);
    CWAIT(UCList,y+9,0);
    CMOVE(UCList,custom.color[2],OldCol);
    CWAIT(UCList,254,0);
    CMOVE(UCList,custom.color[1],0xF00);
    CEND(UCList);
    VPort->UCopIns=UCList;
    RemakeDisplay();
}
VOID main()
{
    SHORT x=4,y=2,collision=custom.clxdatt;
    USHORT code;
    ULONG class;
    OpenW();
    Init();
    custom.clxcon=0x41;
    while(1){
        class=GetMessage(&code);
        if(class==CLOSEWINDOW) break;
        MoveSprite(VPort,&Sprite,XPos+=x,YPos+=y);
        if((YPos<200)&&(Screen->MouseY>10))
            SetPointer(win,&Dummy[0],1,1,0,0);
    }
}
```

```
else SetPointer(win,&Pointer[0],3,16,0,0);
MakeCList(YPos);
if(XPos>=620) x=-4;
else if(XPos<=16) x=4;
if(YPos>=246){
    y=-2;
    Points++;
    if(Points>9){
        SetWindowTitles(win,GOVER,-1);
        do{
            WaitPort(win->UserPort);
            class=GetMessage(&code);
        }while(KEYDOWN);
        SetWindowTitles(win,GNAME,-1);
        Init();
    }
    else Score();
}
else if(YPos<=14) y=2;
collision=custom.clxdatt;
if((collision&0x200)==0x200) y=-2;
else if((collision&4)==4){
    DrawC(2,XPos+10+x,YPos+3+y,4);
    x=-x;
}
}
CloseW();
}
```

Listing 1. Hardware-Sprites werden mittels der SimpleSprite-Struktur beschrieben

```
#include <graphics/sprite.h>
#include <hardware/custom.h>
#include <exec/memory.h>
#include <exec/types.h>
#include <functions.h>
#define KEYDOWN ((code!=SELECTUP)&&(class!=CLOSEWINDOW))
#define TDIM 100
#define GNAME ((STRPTR)" *- HotBottom *-")
#define GOVER ((STRPTR)"--->> GAME-OVER <--- (Klick...)")
SHORT SpriteNr=-1;
SHORT GAlert=0;
SHORT Points;
SHORT NewCol,OldCol;
SHORT XPos,YPos;
WORD Buffer[10];
UWORD *Pattern,PatternDat[9]={
    0xFFFF,0x1111,0x2222,
    0x4444,0x8888,0x1111,
    0x2222,0x4444,0x8888
};
UWORD *Pointer,*Ball,*Dummy;
UWORD PointerDat[]={
    0,0, 0xFFFF,0, 0,0xFFFF, 0xFFFF,0xFFFF, 0,0
};
UWORD BallDat[]={
    0,0, 0x1800,0x2000,0x1C00,
    0x6000,0x8C00,0xF200,
    0x8000,0xFE00,0xC200,
    0xFE00,0x7C00,0x7C00,
    0x3800,0x3800, 0,0
};
struct SimpleSprite Sprite={
    NULL,7,10,10,2
};
struct NewWindow nwin={
    0,0,640,256,0,1,CLOSEWINDOW|MOUSEBUTTONS,
    WINDOWCLOSE|RMTRAP|ACTIVATE,NULL,NULL,
    GNAME,NULL,NULL,0,0,0,WBENCHSCREEN
};
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Screen *Screen;
struct Window *win;
struct RastPort *RP;
struct ViewPort *VPort;
struct UCopList *UCList=NULL;
struct TmpRas Raster;
struct AreaInfo AInfo;
struct PlanePTR Bitplane;
```



```

/*
\\ Funktionen
*/
VOID CloseW()
{
    if(SpriteNr>=0)        FreeSprite(SpriteNr);
    if(Pattern)            FreeMem(Pattern,32);
    if(Pointer)            FreeMem(Pointer,20);
    if(Ball)               FreeMem(Ball,36);
    if(win)               CloseWindow(win);
    if(Bitplane)          FreeRaster(Bitplane,TDIM,TDIM);
    if(UCList){            FreeVPortCopLists(VPort);
                          RemakeDisplay();
    }
    if(GfxBase)            CloseLibrary(GfxBase);
    if(IntuitionBase)      CloseLibrary(IntuitionBase);
    exit(0);
}

VOID OpenW()
{
    REGISTER SHORT i;
    if(!(IntuitionBase=(struct IntuitionBase *)
        OpenLibrary("intuition.library",0))) CloseW();
    if(!(GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library",0))) CloseW();
    if(!(UCList=(struct UCopList *)
        AllocMem(sizeof(struct UCopList),
        MEMF_PUBLIC|MEMF_CLEAR))) CloseW();

    if(!(Bitplane=AllocRaster(TDIM,TDIM))) CloseW();
    InitTmpRas(&Raster,Bitplane,RASSIZE(TDIM,TDIM));
    InitArea(&AInfo,Buffer,2);
    if(!(win=OpenWindow(&win))) CloseW();
    RP = win->RPort;
    RP->TmpRas = &Raster;
    RP->AreaInfo = &AInfo;
    Screen = IntuitionBase->ActiveScreen;
    VPort = &Screen->VPort;
    SetRGB4(VPort, 21,12,12,12);
    SetRGB4(VPort, 22,10,10,10);
    SetRGB4(VPort, 23, 8, 8, 8);
    OldCol=GetRGB4(VPort->ColorMap,2);
    if(!(Dummy=(UWORD *)
        AllocMem(10,MEMF_CHIP|MEMF_CLEAR))) CloseW();
    if(!(Pointer=(UWORD *)AllocMem(20,MEMF_CHIP))) CloseW();
    CopyMem(PointerDat,Pointer,20);
    if(!(Ball=(UWORD *)AllocMem(36,MEMF_CHIP))) CloseW();

```

```

CopyMem(BallDat,Ball,36);
if(!(Pattern=(UWORD *)AllocMem(32,MEMF_CHIP))) CloseW();
CopyMem(PatternDat,Pattern,32);
Sprite.posctldata=&Ball[0];
SpriteNr=GetSprite(&Sprite,2);
}

ULONG GetMessage(code)
{
    USHORT *code;

    struct IntuiMessage *msg;
    ULONG class=0;
    if(msg=(struct IntuiMessage *)GetMsg(win->UserPort)){
        class=msg->Class;
        *code=msg->Code;
        ReplyMsg(msg);
    }
    return(class);
}

VOID DrawC(col,x,y,r)
{
    SHORT col,x,y,r;

    SetAPen(RP,col);
    SetDrMd(RP,JAM1);
    AreaEllipse(RP,x,y,2*r,r);
    AreaEnd(RP);
}

VOID Block(col,x1,y1,x2,y2)
{
    SHORT col,x1,y1,x2,y2;

    SetAPen(RP,col);
    SetDrMd(RP,JAM1);
    RectFill(RP,x1,y1,x2,y2);
}

VOID Score()
{
    SHORT col=Points>8?3:2;
    top=254-24*Points;

    GAlert=-2+col;
    Block( 1,0, 10,10,256);
    Block(col,2,top, 8,254);
}

VOID Init()
{

```

Listing 2. Spiele sind die Hauptanwendungsgebiete für Sprites

Die Amiga-Hardware stellt dem Programmierer acht Sprites zur Verfügung. Für viele Anwendungen, besonders für Spiele, reicht das nicht aus, zumal das erste Sprite auch noch für die Darstellung des Mauszeigers reserviert ist. Abhilfe schafft die Verwendung der sogenannten virtuellen Sprites (»VSprites«), die sich auf dem Display gleich mehrfach darstellen lassen.

Im Grunde genommen sind virtuelle (scheinbare) Sprites, kurz VSprites genannt, einfach Hardware-Sprites, die im Display durch die Mehrfachverwendung der Spritkanäle »gemultiplext« werden können. Daher gelten die meisten der Regeln für die Programmierung der Hardware-Sprites (siehe Seite 53) auch für VSprites.

Der wichtigste Unterschied ist der Wegfall der Beschränkung auf die Gesamtzahl von acht. Eine maximale Anzahl ist nicht vorgeschrieben, sie ist

»Graphic Elements« – Teil 1: VSprites

Sprites virtuell

Was tun, wenn die acht Amiga-Sprites nicht ausreichen? Wir bauen unsere eigenen mit Hilfe eines kleinen Tricks: Wir verwenden virtuelle Sprites.

Von Arno Gölzer

vom Anwendungsfall abhängig. Vor dem ersten Einsatz der VSprites müssen wir dem System mitteilen, welche der acht Hardware-Sprites dafür abgezweigt werden sollen. Diese werden – soweit sie noch frei sind – als belegt gekennzeichnet und für die Darstellung der VSprites abgezweigt.

Mehrere VSprites, die auf dem gleichen Hardware-Sprite basieren, lassen sich nicht direkt nebeneinander darstellen. Die beiden Bilder müssen mindestens durch eine Rasterzeile getrennt sein. Bei Nichtbeachtung dieser Regel, ist nur der VSprite mit der kleineren X-Position sichtbar.

Alle VSprites können in Form und Farbe völlig frei gestaltet werden. Je nach Aufwand fallen daher eine Menge Daten an. Glücklicherweise übernimmt deren Verwaltung das Grafiksystem für uns, so daß uns nur noch die Arbeit der Initialisierung einiger Datenstrukturen bleibt.

Für Hardware-Sprites ist, neben der Festlegung der Image-Daten, lediglich die Vereinbarung der kurzen SimpleSprite-Struktur nötig. Um virtuelle Sprites verwalten zu können, erwartet das System die Belegung zweier Strukturen: die GELSInfo- und die VSprite-Struktur.

Die erste der beiden verwaltet eine Reihe von Grafik-Elementen (Graphic Elements = GELS), die, ähnlich den Area-Objekten, durch Aufruf eines einzigen Befehls auf den Bildschirm gebracht werden. Außer den VSprites kennt das System die Grafik-Elemente BOBs, AnimComps und AnimObjects.

Welche Werte müssen wir in GELsInfo (Listing 1) für VSprites belegen? Die erste Komponente, ein Byte namens »sprRsvd«, beinhaltet die Information, welche Hardware-Sprites vom System für die Verwendung der VSprites verlangt werden:

Sprite-Reserved-Bits												
sprRsvd Bit:	7	6	5	4	3	2	1	0				
Sprite-Nr.:	7	6	5	4	3	2	1	0				

Für jedes gewünschte Sprite müssen wir in sprRsvd das der Spritenummer korrespondierende Bit setzen. Möchten wir beispielsweise für die Darstellung unserer VSprites die Hardware-Sprites mit den Nummern 6 und 7 einsetzen, setzen wir die Bits 6 und 7 und setzen als sprRsvd-Wert $\%11000000 = 192$ ein.

Die nächste Variable »Flags« wird ausschließlich vom System genutzt. Wie oben erwähnt, verwaltet die GELsInfo-Struktur eine Kette von GELs. Das Struktur-Element »GEL-Head« erwartet die Adresse des Kopfes, also die des ersten Struktur und »GEL-Tail« die der letzten Struktur der GELs-Kette. Es handelt sich hierbei lediglich um Dummy-Strukturen, die keine Daten eines Grafik-Elements enthalten sollten.

Die beiden nächsten Parameter, nextLine und lastColor, sind wieder nur für das System bestimmt, jedoch müssen wir für den notwendigen Speicherplatz sorgen (siehe GetGELsInfo() in Listing 3). Es folgt die Adresse auf die in graphics/gels.h definierte collTable-Struktur. Es handelt sich hierbei um ein Array mit 16 Funktions-Adressen, die nach Kollisionen von Grafik-Elementen aufgerufen werden. Die Kollisionskontrolle stellt ein weiteres Bonbon der VSprite-Verwaltung dar. Das System nimmt uns die Überwachung der VSprites ab.

Welche Funktionen bei welchen Kollisionen gestartet werden, legen wir mittels zweier Masken innerhalb der VSprite-Struktur fest – hierzu lesen Sie später mehr.

Die folgenden vier WORD-Variablen stellen die Grenzen für die Elemente der GELs-Kette dieser GELsInfo-Struktur dar. Überschreitet ein VSprite diese Grenze, wird, wenn gewünscht, die Kollisionsroutine[0] aus dem collTable-Array aufgerufen. Die beiden letzten GELsInfo-Werte sind wiederum System-Variablen.

Die immer wieder nach dem gleichen Modus ablaufende Initialisierung der GELsInfo-Struktur schreitet förmlich nach einer Automatisierung. Daher stellen wir Ihnen in unserem GELTool.h (Listing 3) Routinen zur Verfügung, die Ihnen unter anderem die Initialisierungsarbeit und auch das Freigeben der GELsInfo-Struktur abnehmen.

Die Funktion GetGELsInfo(& RastPort, sprRsvd, lm, tm, rm, bm) verlangt als Parameter (von links nach rechts) die Adresse des Display-RastPorts, das sprRsvd-Byte, wie oben besprochen, und dann die Grenzen der GELs-Kette in der Reihenfolge linker Rand, oberer Rand, rechter Rand und unterer Rand.

Die Funktion liefert die Adresse der fertig initialisierten GELsInfo-Struktur oder, im Fehlerfall, den Wert NULL.

```
...
struct GGELsInfo *GI;
...
GI=GetGELsInfo(R,s,l,t,
r,b);
...
```

Das Gegenstück zu GetGELsInfo() stellt die Funktion FreeGELsInfo(GELsInfo) dar. Sie erwartet als Parameter den Rückgabewert der GetGELsInfo()-Funktion. Ihre Aufgabe ist die Freigabe des durch die GELsInfo-Struktur belegten Speichers. Die Voraussetzungen für die Verwendung von VSprites sind jedoch erst mit der Vereinbarung der VSprite-Struktur (Listing 2) abgeschlossen. Diese Struktur ist, von der Bedeutung her, etwa mit der SimpleSprite-Struktur der Hardware-Sprites vergleichbar. Sie ist jedoch sehr viel umfangreicher. Bei der Beschreibung dieser Struktur werden die Besonderheiten der VSprites besonders deutlich.

Werkzeuge zur Initialisierung

Beachten Sie, daß diese Struktur auch bei der Verwendung von BOBs (Blitter Objects) von großem Interesse ist. Wir besprechen sie an dieser Stelle jedoch gezielt für VSprites.

NextVSprite ist das nächste VSprite (oder ein mittels der VSprite-Struktur spezifiziertes Grafikelement) in einer Liste von Grafikelementen. **PrevVSprite** ist das vorhergehende VSprite in dieser Liste. Die Variablen NextSprite und Prev-

Sprite verwaltet das Grafiksystem selbst.

DrawPath, ClearPath, OldY, OldX interessieren uns im Moment weniger, sie sind nur beim Programmieren von BOBs wichtig. **Flags:** Das Setzen des Flags VSPRITES kündigt die Arbeit mit VSprites an.

Setzt das System das Flag VSOVERFLOW, so konnte das Display, wegen der Verwendung zu vieler VSprites (an einer Position), nicht korrekt aufgebaut werden. Bei unserem Beispielprogramm »VS« (Listing 4) läßt sich das gut beobachten. Befinden sich viele VSprites in einer Zeile, so wird ein Teil davon einfach nicht dargestellt. Mindestens zwei VSprites in einer Zeile liegen dann dem gleichen Hardware-Sprite zugrunde. Zur korrekten Darstellung sollten sie, wie schon angedeutet, mindestens durch eine Rasterzeile getrennt sein. Das Flag GELGONE deutet darauf hin, daß sich wenigstens ein Grafik-Element außerhalb der in der GELsInfo-Struktur angegebenen Grenzen befindet.

Y, X ist die Position des VSprites. **Height** ist die Höhe des VSprites in Pixeln gemessen. **Width** ist die Breite des VSprites in WORDs. Da alle Sprites immer 16 Pixel, also ein WORD, breit sind, muß hier der Wert 1 eingetragen werden. Diese und auch die nächste Variable ist bei der Verwaltung der BOBs interessanter. **Depth** ist die Anzahl der BitPlanes, für VSprites kommt nur der Wert 2 in Frage, bei BOBs sieht das etwas anders aus (siehe Seite 62).

MeMask, HitMask: Diese beiden Variablen sind für die Kollisionserkennung von größter Bedeutung. Weiter oben haben wir bereits angesprochen, daß das System die Kollisionsabfrage zwischen Grafik-Elementen für uns übernimmt. Alle GELs einer Kette können in die Abfrage einbezogen werden. Welche GELs womit kollidieren, legen wir mit den »Hit-Me«-Masken fest.

Möglich ist die Überwachung der Kollisionen GEL/GEL und GEL/Border. Der einfachste Fall ist sicher die Festlegung der Kontrolle über die Randkollision. Setzen wir Bit 0 der Hit-Maske des zu kontrollierenden GELs mit dem Wert

```
%000000000000000001
```

so wird die Funktion mit der Adresse an Position 0 des Funktions-Arrays »collTable« der GELsInfo-Struktur aufgerufen,

wenn sich das Grafik-Element an den, in der gleichen Struktur festgelegten, Grenzen befindet. Welche Funktion das ist, legen wir vor der Vereinbarung der VSprite-Struktur mittels der Funktion

```
SetCollision(0, CheckBorder, &GELsInfo);
```

fest. SetCollision() schreibt die Adresse der selbstdefinierten Funktion CheckBorder() an die Position 0 der Funktionstabelle »collTable« der GELsInfo-Struktur, deren Adresse als dritter Parameter angegeben ist.

GELs auf Kollisionskurs

Rufen wir jetzt die Funktion DoCollision(&rp) auf, als Parameter dient die Adresse des RastPorts, wird, falls eine Kollision festgestellt wurde, nach CheckBorder() verzweigt. Als Parameter werden dabei die Adresse der VSprite-Struktur des GELs und ein Flag-Byte übergeben:

```
VOID CheckBorder(vs,
flag)
struct VSprite *vs;
BYTE flag;
{
...
}
```

Der Byte-Wert übermittelt uns, an welcher Grenze das GEL kollidiert ist. Es bedeuten die Symbole LEFTHIT – linker Rand; TOPHIT – oberer Rand; RIGHTHIT – rechter Rand; BOTTOMHIT – unterer Rand. Da bei einem Aufruf, z.B. bei der Kollision mit der linken oberen Ecke, mehrere Werte, für das Beispiel LEFTHIT und TOPHIT, gesetzt sein könnten, ist das betreffende Flag durch eine AND-Verknüpfung zu erfragen. Zum Beispiel für den oberen Rand:

```
...
if((flag & TOPHIT) == TOPHIT){
...
}
```

Bei Kollisionen zwischen GELs (GEL/GEL) können die Funktionen der zweiten bis fünfzehnten Adresse der Funktionstabelle in GELsInfo aufgerufen werden. Welche, entscheidet die Bitbelegung der AND-Verknüpfung der Hit-Maske des einen GELs mit der Hit-Maske eines anderen GELs. Ein Beispiel verdeutlicht den Sachverhalt: Angenommen,

wir arbeiten mit zwei bewegten VSprites. Bei einer Kollision soll die selbstdefinierte Funktion `CheckColl()` aufgerufen werden. Erster Schritt – die `SetCollision()`-Funktion aufrufen:

```
SetCollision(1, CheckColl, GELsInfo);
```

Die Adresse der `CheckColl()`-Funktion wird dadurch in `collTable` an der zweiten Stelle eingetragen – die erste Stelle belegen wir mit `SetCollision(0,...)`.

Zweiter Schritt – VSprite-Struktur initialisieren. Dabei setzen wir beide Me-Masken und beide Hit-Masken der GELs gleich

```
%00000000000000010
```

Das zweite Bit ist gesetzt, daher wird auch bei einer Kollision die zweite Funktion der Kollisionstabelle aufgerufen, und das ist in unserem Beispiel `CheckColl()`.

Ob eine Kollision stattfand, ermittelt – wie im ersten Beispiel – `DoCollision()`. In diesem Fall werden unserer Funktion jedoch, anstelle einer VSprite-Strukturadresse und einem Flag, zwei VSprite-Strukturadressen, nämlich die der beiden zusammengestoßenen VSprites, übergeben:

```
VOID CheckColl(vs1, vs2)
struct VSprite*vs1, *vs2;
{
    ...
}
```

Die erste Adresse stammt von dem VSprite, welches sich bei der Kollision links und/oder oberhalb des anderen befand. Die BitNummer des Ergebnisses der AND-Verknüpfung der Me-Maske des einen, mit der Hit-Maske des anderen GELs bestimmt also die Funktion, die bei einer Kollision aufgerufen werden soll. Sind im Ergebnis mehrere Bits gesetzt, entscheidet das niederwertigste Bit über den Aufruf:

```
%0000101011000010
```

In diesem Beispiel würde `DoCollision()` bei einer »passenden« Kollision die zweite Funktion der Tabelle aufrufen. Voraussetzung für die erfolgreiche Überwachung ist die Berechnung der sogenannten Borderline und der `CollMask` (siehe unten).

Weiterhin muß erwähnt werden, daß `DoCollision()` manchmal Kollisionen »übersieht«, besonders wenn Sie viele VSprites verwenden und nur wenige Hardware-Sprites einsetzen. In diesen Fällen sollte man auch

die Kollisionsroutine besonders kurz halten (siehe Demo Listing 4). **ImageData** ist die Adresse des UWORD-Arrays, welches die Daten für den Sprite enthält. Die Daten werden wie »normale« Sprite-Daten ermittelt, jedoch fallen bei VSprites die beiden 0-UWORDS vor und nach der Sprite-Beschreibung weg. Das Array muß im Chip-Memory liegen, VSprites mit gleichem Aussehen dürfen hier, anders als bei Hardware-Sprites, auf das gleiche Array zurückgreifen.

BorderLine: der Wert dieser VSprite-Strukturkomponente ist für die Kollisionserkennung nötig. Ein VSprite darf mehrere Zeilen hoch sein. Eine Spritezeile wird mit 2 UWORDS beschrieben. Der Wert **Borderline** ist das Ergebnis der ODER-Verknüpfung aller WORDs und ist daher genau ein WORD breit (siehe auch unter `CollMask`).

Ganz ohne Bitfriemelei

CollMask (siehe auch »BorderLine«). `CollMask` enthält die ODER-Verknüpfung der beiden VSprite-Planes. Es handelt sich demnach um ein WORD-Array, dessen Elementzahl von der Höhe des VSprite abhängt. Bevor Sie sich Sorgen um die Berechnung der Werte **BorderLine** und **CollMask** machen, sei erwähnt, daß bereits eine Funktion eingerichtet wurde, die uns dieses Übel abnimmt:

```
InitMasks(&VSprite);
```

Die `InitMasks()`-Funktion, mit der Adresse der entsprechenden VSprite-Struktur als Parameter, übernimmt die Berechnung für uns.

SprColors ist die Adresse eines UWORD-Arrays, welches aus drei Werten besteht. Hier befinden sich die VSprite-Farben, die wie gewohnt als 16-Bit Wert (wovon allerdings bisher nur die unteren 12 genutzt werden), angegeben wird. Zum Beispiel für weiß: `0xFFFF`. Jedes VSprite kann über eine eigene Farbenpalette verfügen oder auch die eines anderen mitbenutzen.

VSBob enthält die Adresse einer Bob-Struktur, die jedoch, wenn das `VSPRITE`-Flag gesetzt ist, ignoriert wird. **PlanePick**, **PlaneOnOff**: Diese Werte sind nur bei der Verwendung von BOBs nötig. **VUserExt** für das Einbinden eigener Werte oder Strukturen in die VSprite-Struktur. Die Vereinbarung des `VUserStuff`-Symbols muß vor

das Einbinden des Headerfiles »graphics/gels.h« gelegt werden, sonst wird das Symbol als Amigadatentyp `SHORT` festgelegt. Möchte man keine Struktur, die z.B. Daten wie Geschwindigkeit oder weitere Image-Daten enthalten könnte, definieren, läßt sich `VUserStuff` als Kennnummer (ID) für VSprites verwenden (siehe Beispiel-Listing 4).

Auch für die Initialisierung der VSprite-Struktur haben wir eine Hilfsfunktion erstellt:

```
...
struct VSprite *VS;
...
VS=GetVSprite(&rp, x, y, h,
me, hi, d, c, u);
...
```

`GetVSprite()` füllt die VSprite-Struktur mit Werten auf. Die Parameter (von links nach rechts) sind: die Adresse des Display-RastPorts; die X/Y-Position des VSprites (bezogen auf die linke obere Ecke des ViewPorts oder Screens), die Höhe des VSprites; `MeMask` und `HiMask`; die Adresse der Image-Daten; die Adresse des Color-Arrays und die `VUserExt`-Variable. Die Funktion liefert die Adresse der fertig initialisierten VSprite-Struktur oder, falls ein Fehler auftrat, den Wert `NULL`.

`GetVSprite` ruft die »Universal-Funktion« `GetGel()` auf, welche neben VSprites auch BOBs initialisieren kann. Wir werden uns im Artikel über BOBs auf Seite 62 näher mit ihr beschäftigen.

Die Freigabe der VSprite-Struktur übernimmt das Makro `FreeVSprite(vsp, rp, vp, kill)`;

Die Parameter von links nach rechts sind: die Adresse der mittels `GetVSprite()` ermittelten VSprite-Struktur; die Adresse des RastPorts; die Adresse des ViewPorts und das Kill-Flag. Ist »Kill« ungleich null, baut die Funktion das Display, ohne den genannten VSprite, neu auf.

Möchte man in einer Schleife mehrere VSprites freigeben, ist das Setzen des Kill-Flags erst beim letzten Aufruf nötig.

```
...
for(i=0; i<n; i++){
FreeVSprite(vp[i], rp, vp,
i==(n-1));
}
...
```

Nachdem wir nun die Schritte a) Initialisierung der `GelsInfo`-Struktur; b) Festlegung der Kollisionsroutinen mit `SetCollision()`;

und c) Initialisierung der VSprite-Struktur erledigt haben, ist es an der Zeit den VSprite darstellen zu lassen. Zum ersten muß die GELs-Liste nach den Y-Positionen sortiert werden, `SortGList(rp)` erledigt das für uns, wenn wir die Adresse des Display-RastPorts angeben. Als nächstes erzeugt `DrawGList(rp, vp)` die für die Darstellung notwendigen Copperlisten. Die Parameter sind die Adresse des RastPorts und die des ViewPorts. Als letztes sorgt beim einfachen Grafikdisplay die Sequenz `MakeVPort()`, `MrgCop()` und `LoadView()` (siehe hierzu Artikel »Display« in diesem Heft) dafür, daß die GELs-Kette endlich sichtbar wird. Intuition-Anwender haben es, wie so oft, etwas einfacher: `RemakeDisplay()` läßt die Grafik-Elemente erscheinen.

Das Makro `DrawGels(rp, vp)`, welches in unserem Headerfile definiert ist, vereinfacht die Vorgehensweise. Der Aufruf, er verlangt die Adresse des Rast-

Alle auf einen Schlag

Ports und die des ViewPorts, stellt die GELs-Kette »auf einen Schlag« dar. Die letzte VSprite-Funktion (eigentlich auch ein Makro) unseres Tools ist `MoveVSprite(vsp, rp, vp, x, y)`. Mit diesem Hilfsmittel kann ein einzelner VSprite bewegt werden. Die Parameter sind: die Adresse des betreffenden VSprites, die des RastPorts und des ViewPorts und die Inkremente x und y. Diese werden beim Aufruf zu der aktuellen VSprite-Position addiert.

Mit allen Hilfsmitteln ausgerüstet kann jetzt endlich die Tat folgen. In unserem Demo-Listing »VS« (Listing 4) verlangen wir die Darstellung von 20 (!) VSprites in einem Display. Alle Sprites greifen auf die gleichen Image-Daten zurück, jedoch werden sie mittels fünf verschiedener Farbenpaletten (5 x 3 = 15 Farben) auf dem Workbenchscreen dargestellt. Jeweils vier VSprites erscheinen also in der gleichen Farbe. Da wir als `sprRsvd`-Wert `0xFE` angegeben haben, wird auch Sprite 1, das mit Sprite 0 (dem Pointer) ein Sprite-Paar bildet, in die Darstellung mit einbezogen. Da jeweils ein Sprite-Paar über ein gemeinsames Farbregerster verfügt, nimmt der Mauszeiger ständig die Farbe seines »Partners« an. In `OpenW()` legen wir fest, daß im Falle einer

Randkollision die Funktion BorCon() starten soll. Dort negieren wir den Geschwindigkeits-Vektor, der für die Kollision verantwortlich war. Weiterhin sollen bei GEL/GEL Kollisionen, die als Ergebnis der AND-Verknüpfung der Hit/Me-Masken die Bits 1-5 setzen, die Funktion MarCon() aufgerufen werden. In der darauffolgenden Schleife legen wir die Hit/Me-Masken in Abhängigkeit des Farbwertes fest. Kollisionen unter gleichfarbigen VSprites rufen über DoCollision() die Funktion MarCon() auf. Diese haben wir, wegen **Unterscheidung über Kennnummer** der hohen VSprite-Anzahl extrem kurz gehalten – wir ermitteln nur die VSprites-IDs. Diese Kennnummern legen wir ebenfalls in der zweiten FOR-Schleife der OpenW()-Funktion fest.

Der Aufruf des Makros DrawGels() macht die Grafik-Elemente sichtbar. In einer Endlosschleife innerhalb der main()-Funktion bewegen wir alle VSprites, indem wir den X/Y-Positionen die entsprechenden Werte aus den Arrays vX[] und vY[] addieren und danach das Display mittels DrawGels() neu darstellen lassen. Natürlich könnten wir hier auch MoveVSprite() einsetzen. Jedoch würde die Funktion nach der Verän-

derung jedes Sprites das Display neu berechnen. Diese Funktion eignet sich besser für die Bewegung eines einzelnen VSprites. Der Aufruf von DoCollision() nach jeder Neudarstellung des Displays kontrolliert die VSprites. Stellte die Funktion eine Kollision fest, ruft sie MarCon() auf, welche die Kennnummern der beteiligten VSprites ermittelt.

Nach einem Bildschirmblitz entfernen wir einen davon und die Schleife beginnt mit einem neuen Durchlauf. Die Funktion CloseW() gibt vor Programmen, die alle angeforderten Speicherplätze wieder an das System zurück. Mit diesem Demo-Listing lassen sich sehr viele Experimente anstellen. Wie verhält sich z.B. das Programm, wenn 40 VSprites verlangt werden.

Lassen sich alle VSprites in unterschiedlichen Farben darstellen? Oder: was passiert, wenn Sie als sprRsvd einen kleineren Wert angeben und was geschieht nach der Angabe von 0xFF?

Testen Sie das Programm – verändern Sie einfach alle Parameter, die Ihnen nicht 100prozentig klar sind und Sie werden sich offenstehende Fragen bald selbst beantworten.

SO

```
struct GelsInfo
{
    BYTE sprRsvd;
    UBYTE Flags;
    struct VSprite *gelHead, *gelTail;
    WORD *nextLine;
    WORD **lastColor;
    struct collTable *collHandler;
    short leftmost, rightmost, topmost, bottommost;
    APTR firstBlissObj, lastBlissObj;
};
```

Listing 1. Die Struktur GelsInfo informiert über Grafik-Elemente

```
struct VSprite
{
    struct VSprite *NextVSprite;
    struct VSprite *PrevVSprite;
    struct VSprite *DrawPath;
    struct VSprite *ClearPath;
    WORD OldY, OldX;
    WORD Flags;
    WORD Y, X;
    WORD Height;
    WORD Width;
    WORD Depth;
    WORD MeMask;
    WORD HitMask;
    WORD *ImageData;
    WORD *BorderLine;
    WORD *CollMask;
    WORD *SprColors;
    struct Bob *VSBob;
    BYTE PlanePick;
    BYTE PlaneOnOff;
    VUserStuff VUserExt;
};
```

Listing 2. Die VSprite-Struktur beschreibt neben VSprites auch BOBs

```
/*
  Gölzy's > GELTool.h <
  einfache Handhabung
  der Grafikelemente
*/
#define DrawGels(rp, vp) MoveGel(NULL, rp, vp, 0, 0)
#define MoveVSprite(vsp, rp, vp, x, y) MoveGel(vsp, rp, vp, x, y)
#define MoveBob(bob, rp, vp, x, y) MoveGel(bob->BobVSprite, rp, vp, x, y)
#define GetVSprite(rp, x, y, h, me, hi, d, c, u) GetGel(rp, VSPRITE, 0, x, y, 1, h, 2, me, hi, d, c, 0, 0, FALSE, u, 0)
#define FreeVSprite(vsp, rp, vp, kill) FreeGel(vsp, rp, vp, kill)
#define FreeBob(bob, rp, vp, kill) FreeGel(bob->BobVSprite, rp, vp, kill)
VOID MoveGel(vsp, rp, vp, x, y)
{
    struct VSprite *vsp;
    struct RastPort *rp;
    struct ViewPort *vp;
    WORD x, y;

    if(vsp){
        vsp->X+=x;
        vsp->Y+=y;
    }
    SortGList(rp);
    DrawGList(rp, vp);
    RemakeDisplay();
}
VOID FreeGel(vsp, rp, vp, kill)
{
    struct VSprite *vsp;
    struct RastPort *rp;
    struct ViewPort *vp;
    SHORT kill;

    struct DBufPacket *Dbp;
    struct Bob *b;
    WORD *bl, *cm, *sb, *db, f, w, h, d, n=0;
    BYTE i, m, pp, po;
    if(vsp){
        f = vsp->Flags;
        w = vsp->Width;
        h = vsp->Height;
        d = vsp->Depth;
        bl = vsp->BorderLine;
        cm = vsp->CollMask;
        b = vsp->VSBob;
        if((f & VSPRITE) == VSPRITE) RemVSprite(vsp);
        else if(b) RemBob(b);
        if(kill) DrawGels(rp, vp);
        if(bl) FreeMem(bl, w * sizeof(WORD));
        if(cm) FreeRaster(cm, (SHORT)(16 * w), (SHORT)h);
        if(b){
            pp = vsp->PlanePick;
            po = vsp->PlaneOnOff;
            m = po & !pp;
            for(i=0; i<8; i++) if(m & (1<<i)) n++;
            sb = b->SaveBuffer;
            Dbp = b->DBuffer;
            if(sb) FreeRaster(sb, (SHORT)(16 * w), (SHORT)(d+n)*h);
            if(Dbp){
                db = Dbp->BufBuffer;
                if(db) FreeRaster(db, (SHORT)(16 * w), (SHORT)((d+n)*h));
                FreeMem(Dbp, sizeof(struct DBufPacket));
            }
            Dbp = NULL;
            FreeMem(b, sizeof(struct Bob));
        }
        FreeMem(vsp, sizeof(struct VSprite));
    }
}
struct VSprite *GetGel(rp, f, bf, x, y, w, h, d, me, hi, da, c, pp, po, dbuf, vu, bu)
{
    struct RastPort *rp;
    BOOL dbuf;
    WORD f, bf, x, y, w, h, d, me, hi, *da, *c;
    BYTE pp, po;
    VUserStuff vu;
    BUserStuff bu;
}
```



```

struct VSprite *VSprite;
struct Bob *Bob;
struct DBufPacket *Dp=NULL;
WORD *bl=NULL,*cm=NULL,*db=NULL,*sb=NULL,n=0;
BYTE i,m=po&!pp;
if(!(VSprite=(struct VSprite *)
    AllocMem(sizeof(struct VSprite),MEMF_CLEAR|MEMF_PUBL-
IC)))){
    return(NULL);
}
VSprite->Flags          = f;
VSprite->X              = x;
VSprite->Y              = y;
VSprite->Width          = w;
VSprite->Height         = h;
VSprite->Depth         = d;
VSprite->MeMask         = me;
VSprite->HitMask        = hi;
VSprite->ImageData      = da;
if(!(bl=(WORD *)AllocMem(w*sizeof(WORD),MEMF_CLEAR|MEMF_PUBL-
IC)))){
    return(NULL);
}
VSprite->BorderLine     = bl;
if(!(cm=(WORD *)AllocRaster((SHORT)(16*w),(SHORT)h)))
    return(NULL);
VSprite->CollMask       = cm;
VSprite->SprColors      = (f&VSPRITE)==VSPRITE?c:NULL;
VSprite->PlanePick      = pp;
VSprite->PlaneOnOff     = po;
VSprite->VUserExt       = vu;
InitMasks(VSprite);
if((f&VSPRITE)!=VSPRITE){
    for(i=0;i<8;i++) if(m&(1<i)) n++;
    if(!(Bob=(struct Bob *)
        AllocMem(sizeof(struct Bob),MEMF_CLEAR|MEMF_PUBL-
PUBLIC)))){
        return(NULL);
    }
    VSprite->VSBob       = Bob;
    Bob->Flags          = bf;
    if((f&SAVEBACK)==SAVEBACK){
        if(!(sb=(WORD *)AllocRaster((SHORT)(16*w),(S-
HORT)((d+n)*h)))){
            return(NULL);
        }
        Bob->SaveBuffer  = sb;
        Bob->ImageShadow= VSprite->CollMask;
        Bob->BobVSprite  = VSprite;
        if(dbuf){
            if(!(Dp=(struct DBufPacket *)
                AllocMem(sizeof(struct DBufPacket),M
EMF_CLEAR|MEMF_PUBLIC)))){
                return(NULL);
            }
            if(!(db=(WORD *)AllocRaster((SHORT)(16*w),(S-
HORT)((d+n)*h)))){
                return(NULL);
            }
            Dp->BufBuffer = db;
            Bob->DBuffer  = Dp;
            Bob->BUserExt = bu;
        }
        if((f&VSPRITE)==VSPRITE)AddVSprite(VSprite,rp);
        else
            AddBob(Bob,rp);
        return(VSprite);
    }
}
struct Bob *GetBob(rp,f,bf,x,y,w,h,d,me,hi,da,pp,po,dbuf,vu,bu)
struct RastPort *rp;
BOOL dbuf;
WORD f,bf,x,y,w,h,d,me,hi,*da;
BYTE pp,po;
VUserStuff vu;
BUserStuff bu;
{
    struct VSprite *VSprite;
    f&= VSPRITE;
    if(VSprite=GetGel(rp,f,bf,x,y,w,h,d,me,hi,da,NULL,pp,po,dbuf-
,vu,bu))

```

```

        return(VSprite->VSBob);
    else
        return(NULL);
}
VOID FreeGelsInfo(GelsInfo)
    struct GelsInfo *GelsInfo;
{
    struct collTable *ct;
    struct VSprite *he,*ta;
    WORD *nl,**lc;
    if(GelsInfo){
        he=GelsInfo->gelHead;
        ta=GelsInfo->gelTail;
        nl=GelsInfo->nextLine;
        lc=GelsInfo->lastColor;
        ct=GelsInfo->collHandler;
        if(he) FreeMem(he,sizeof(struct VSprite));
        if(ta) FreeMem(ta,sizeof(struct VSprite));
        if(nl) FreeMem(nl,8*sizeof(WORD));
        if(lc) FreeMem(lc,8*sizeof(LONG));
        if(ct) FreeMem(ct,sizeof(struct collTable));
        FreeMem(GelsInfo,sizeof(struct GelsInfo));
    }
}
struct GelsInfo *GetGelsInfo(rp,sr,lm,tm,rm,bm)
    struct RastPort *rp;
    BYTE sr;
    SHORT lm,tm,rm,bm;
{
    struct GelsInfo *GelsInfo;
    struct VSprite *Head,*Tail;
    struct collTable *ct;
    WORD *nl,**lc;
    if(!(GelsInfo=(struct GelsInfo *)
        AllocMem(sizeof(struct GelsInfo),MEMF_CLEAR|MEMF_PUB-
LIC)))){
        return(NULL);
    }
    if(!(Head=(struct VSprite *)
        AllocMem(sizeof(struct VSprite),MEMF_CLEAR|MEMF_PUBL-
IC)))){
        FreeGelsInfo(GelsInfo);
        return(NULL);
    }
    if(!(Tail=(struct VSprite *)
        AllocMem(sizeof(struct VSprite),MEMF_CLEAR|MEMF_PUBL-
IC)))){
        FreeGelsInfo(GelsInfo);
        return(NULL);
    }
    if(!(nl=(WORD *)AllocMem(8*sizeof(WORD),MEMF_CLEAR|MEMF_PUBL-
IC)))){
        FreeGelsInfo(GelsInfo);
        return(NULL);
    }
    if(!(lc=(WORD **)AllocMem(8*sizeof(LONG),MEMF_CLEAR|MEMF_PUB-
LIC)))){
        FreeGelsInfo(GelsInfo);
        return(NULL);
    }
    if(!(ct=(struct collTable *)
        AllocMem(sizeof(struct collTable),MEMF_CLEAR|MEMF_PU-
BLIC)))){
        FreeGelsInfo(GelsInfo);
        return(NULL);
    }
    GelsInfo->sprRsrvd  =sr;
    GelsInfo->nextLine  =nl;
    GelsInfo->lastColor =lc;
    GelsInfo->collHandler=ct;
    GelsInfo->leftmost  =lm;
    GelsInfo->rightmost =rm;
    GelsInfo->topmost   =tm;
    GelsInfo->bottommost=bm;
    InitGels(Head,Tail,GelsInfo);
    rp->GelsInfo       =GelsInfo;
    return(GelsInfo);
}

```

Listing 3. Unser Headerfile GELTool.h unterstützt Sie beim Arbeiten mit beweglichen Grafik-Elementen


```

/*
cc VS.c +L
ln VS.o -lc32
*/
/*
\\ Götz's VSprite Demo
*/
#include <intuition/intuitionbase.h>
#include <graphics/collide.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <functions.h>
#include <graphics/gels.h>
#include "GELTool.h"
#define VS 20

SHORT Id1=0, Id2=0;
WORD vX[VS]={
    5,-5,5,-5,4,-4,4,-4,3,-3,3,-3,2,-2,2,-2,1,-1,1,-1
};
WORD vY[VS]={
    1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5
};
WORD *Ball,BallDat[]={
    0x1800,0x2000,0x1C00,0x6000,0x8C00,0xF200,0x8000,
    0xFE00,0xC200,0xFE00,0x7C00,0x7C00,0x3800,0x3800
};
WORD Col[5][3]={
    0xAAA,0x888,0x555,    0xA00,0x800,0x500,
    0xF00,0xB00,0x800,    0xAAA,0x880,0x550,
    0x0A0,0x080,0x050
};
TEXT DName[]=" Götz's VSprite-Demo";
struct NewWindow nwin=

```

```

200,40,250,150,0,1,CLOSEWINDOW MOUSEBUTTONS,
WINDOWCLOSE RMBTRAP1 ACTIVATE,NULL,NULL,
DName,NULL,NULL,0,0,0,0,WBENCHSCREEN
};
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Window *win;
struct RastPort *RP;
struct ViewPort *VP;
struct VSprite *VSp[VS];
struct GelsInfo *GelsInfo;
/*
\\ Funktionen
*/
VOID BorCon(vsp,witch)
    struct VSprite *vsp;
    SHORT witch;
{
    SHORT id=vsp->VUserExt;

    if(((witch&LEFTHIT)==LEFTHIT)||((witch&RIGHTHIT)==RIGHTHIT))
        vX[id]*=-1;
    if(((witch&TOPHIT)==TOPHIT)||((witch&BOTTOMHIT)==BOTTOMHIT))
        vY[id]*=-1;
}
VOID MarCon(vsp1,vsp2)
    struct VSprite *vsp1,*vsp2;
{
    Id1=vsp1->VUserExt;
    Id2=vsp2->VUserExt;
}
VOID CloseW()

```

Fortsetzung auf Seite 146

»Graphic Elements« - Teil 2: BOBs

Blitters Objekte

Trotz ihrer Vorzüge gegenüber den Hardware-Sprites lassen unsere VSprites noch einige Wünsche offen. Die Blitter-Objekte werden den Wünschen des verwöhnten Programmierers auf jeden Fall gerecht.

Von Arno Gölzer

Die Verwandtschaft der VSprites mit den Hardware-Sprites läßt sich nicht leugnen. Die »Blitter-Objekte«, kurz BOBs genannt, sind dagegen völlig andere Grafik-Elemente. Ihr Name verrät es: nicht der Copper, sondern sein Kollege, der Blitter, ist Ursprung ihres Wesens. Die BOBs werden aber, ebenso wie die VSprites, nicht rein hardwaremäßig, sondern mittels ausgeklügelter System-Software generiert.

Die Arbeit des Blitters konzentriert sich auf das Kopieren der BOB-Grafik in die Zeichenfläche sowie das Retten und Rückkopieren des alten Inhalts der Zeichenfläche. BOBs dür-

fen, solange damit die Anzahl der Bitebenen (Planes) des Ziel-RastPorts nicht überschritten wird, aus bis zu fünf Planes bestehen. Anzahl und die Registernummern der Farben sind von der Anzahl der BOB-Planes und deren Platzierung in den Ziel-Planes abhängig.

Die Breite der BOBs ist, anders als die der Sprites, nicht auf 16 Pixel beschränkt. Die maximale Breite sind 64 Words, was 16 x 64=1024 Pixel (Blitter-Limit) entspricht. Die maximale Höhe beträgt ebenfalls 1024 Pixel. Die einzelnen Punkte eines Sprites haben, unabhängig von der Auflösung des aktuellen Displays, immer die Größe eines Bildschirmpixels im LoRes-Modus. Die Bildpunkte einer BOB-Grafik entsprechen dage-

gen in ihrer Größe dem Pixelformat des Displays, in das sie kopiert wird. BOBs sind also sehr flexibel und gut für die Spieleprogrammierung geeignet. Doch diese Flexibilität hat auch ihren Preis - der Aufwand für die Initialisierung der notwendigen Strukturen ist hoch. Neben der »GelsInfo« und der »VSprite«-Struktur, muß noch eine spezielle BOB-Struktur mit Werten versorgt werden.

Blitter macht's möglich

Glücklicherweise nimmt uns das Headerfile »GELTool.h« (Listing 3 des ersten Teils) ein großes Stück der anfallenden Arbeit ab. Zunächst aber - als Grundlage - die Besprechung der Strukturen und deren Besonderheiten in bezug auf BOBs. Mit der »GelsInfo«-Struktur, die eine Kette von Grafik-Elementen verwaltet, haben wir uns bereits im Teil 1 des »Graphic Elements«-Artikels eingehend befaßt, so daß wir hier auf

eine ausgedehnte Besprechung verzichten können.

Allein zur Vereinbarung der ersten Komponente - Sie erinnern sich an das Byte »sprRsrvd« - ist noch ein Wort zu sagen. Diese Variable gibt an, welche Hardware-Sprites für den Einsatz der VSprites abgezweigt werden sollen. Besteht Ihre »GEL«-Kette nur aus BOBs, können Sie »sprRsrvd« auf 0 setzen.

Die Vereinbarung der Struktur übernimmt, wie auch bei den VSprites, die »GELTool.h«-Funktion:

```

...
struct GelsInfo *GI;
...
GI=GetGelsInfo(R,s,l,t,
r,b);
...

```

Als Gegenstück dazu gibt »FreeGelsInfo(GelsInfo)« den für die Struktur notwendigen Speicher wieder frei.

Weit interessanter als die »GelsInfo«-Struktur wird für uns jetzt die VSprite-Struktur. Tatsächlich - wir müssen uns auch


```

struct VSprite
{
    struct VSprite *NextVSprite; /* nächstes GEL in der Kette */
    struct VSprite *PrevVSprite; /* letztes GEL-Kettenglied */
    struct VSprite *DrawPath; /* Zeichen-Reihenfolge der GELs */
    struct VSprite *ClearPath; /* Reihenfolge - Löschen und
                                Restaurieren */

    WORD OldY, OldX; /* Position des geretteten
                    Zeichenflächen-Rechtecks */

    WORD Flags; /* GEL-Flags - unterscheidet: User- und Systemflags */
    WORD Y, X; /* die Position der linken oberen Ecke des GELs */
    WORD Height; /* die Höhe des GEL in Pixel */
    WORD Width; /* die GEL-Breite in WORDs (16 Bit) */
    WORD Depth; /* Anzahl der GEL-Planes (max. 5 -
                bei VSprites immer 2) */

    WORD MeMask; /* mit HitMask zusammen zur
                Kollisionserkennung nötig */

    WORD HitMask; /* Kollisionserkennung - Bit 0 gesetzt:
                Border-Kollision */

    WORD *ImageData; /* die im Chip-Memory gelagerten GEL-Bilddaten */
    WORD *BorderLine; /* ODER-Verknüpfung aller GEL-Zeilen */
    WORD *CollMask; /* ODER-Verknüpfung aller GEL-Planes (Schatten) */
    WORD *SprColors; /* das Farb-Array für VSprites */
    struct BOB *VSBOb; /* die BOB-Struktur beschreibt das Blitter-
                        Objekt */

    BYTE PlanePick; /* in welche Plane(s) soll der BOB kopiert
                    werden? */

    BYTE PlaneOnOff; /* in welche Planes gehört der >>Bild-
                    Schatten <<? */

    VUserStuff VUserExt; /* Variable für den eigenen Gebrauch */
};

```

Listing 1. Auch BOBs werden unter anderem über die VSprite-Struktur verwaltet

bei der Programmierung von BOBs mit ihr beschäftigen. Listing 1 zeigt sie noch einmal, mit einer Kurzbeschreibung aller Variablen. Einige davon haben wir ja bereits näher beleuchtet.

Die ersten beiden Adressen verweisen auf das nächste bzw. vorhergehende Glied der »GEL«-Kette. »DrawPath« enthält die Reihenfolge, in der die Objekte auf den Bildschirm gezeichnet werden. »ClearPath« beschreibt, in welcher Folge der Blitter die überschriebenen Bereiche der Zeichenfläche restaurieren muß. Der Blitter kopiert vor der Darstellung eines »SAVEBACK«-BOBs (siehe auch unter Flags) den rechteckigen Bereich der Zeichenfläche, der mit der BOB-Grafik überschrieben wird, in einen Puffer. Nach dem Bewegen oder Löschen des BOBs gibt der Blitter den Inhalt des Puffers zurück. Die beiden »Word«-Variablen »OldX« und »OldY« enthalten die Position der linken oberen Ecke des Zielbereichs.

Einige der in »graphics/gels.h« definierten VSprite-Flags haben wir schon kennengelernt. Hier eine Liste, die auch die BOB-Flags berücksichtigt:

VSPRITE: stellt die VSprite-Struktur auf die Arbeit mit VSprites ein

SAVEBACK: der für die Darstellung eines BOBs nötige

```

struct BOB
{
    WORD Flags; /* die BOB-Flags - unterscheidet: User- und
                Systemflags */
    WORD *SaveBuffer; /* der Puffer für den überschriebenen
                    Zeichenbereich */
    WORD *ImageShadow; /* der Bild-Schatten (ODER-Verknüpfung
                    der Planes) */
    struct BOB *Before; /* wird vor dem hier beschriebenen BOB
                        gezeichnet */
    struct BOB *After; /* wird nach dem hier beschriebenen BOB
                        gezeichnet */
    struct VSprite *BOBVSprite; /* die VSprite-Struktur (Listing 1) */
    struct AnimComp *BOBComp; /* die Adresse einer AnimComp-
                    Struktur */
    struct DBufPacket *DBuffer; /* Buffer für Double-Buffering-
                    Betrieb */
    BUserStuff BUserExt; /* Variable für den eigenen Gebrauch */
};

```

Listing 2. Für ein BOB ist die Vereinbarung der BOB-Struktur nötig

Speicherbereich wird an die Adresse »SaveBuffer« der BOB-Struktur gerettet. Nach der Darstellung wird die Zeichenfläche restauriert. Ist »SAVEBACK« nicht gesetzt, zeichnet der BOB bei der Bewegung wie ein Pinsel in den »RastPort«.

Der SaveBuffer muß groß genug sein, um alle überschriebenen Planes wiederherstellen zu können. Die Speichergröße errechnet sich aus:

$16 * \text{Width} * \text{Height} * P$

Der Faktor 16 rechnet den »Word-Width«-Wert in Pixel um. Das »P« steht für die Anzahl der überschriebenen Planes. Es

handelt sich dabei **nicht** um den »Depth«-Wert, sondern um die Summe Depth + Anzahl der Planes, die mit dem Bild-Schatten überschrieben werden (siehe unter »PlanePick«/»PlaneOnOff«). Das System fordert den notwendigen Speicher nicht an, sondern der Programmierer muß – solange er nicht über »GELTool.h« verfügt – selbst für einen entsprechend großen Puffer sorgen.

OVERLAY: die gelöschten (nicht gesetzten) Punkte der BOB-Grafik erscheinen nach dem Setzen des »OVERLAY«-Flags durchsichtig. Unser Beispielsprogramm (Listing 3) verdeutlicht den »OVERLAY«-Modus. Nur während dem Festhalten der linken Maustaste bewegt sich der BOB mit gesetztem »OVERLAY«-Bit.

GELGONE: wird vom System gesetzt, wenn sich ein »GEL«, gleichgültig, ob VSprite oder BOB, außerhalb des mittels der »GelsInfo«-Struktur beschriebenen Rechtecks befindet.

Die folgenden Elemente können wie unter VSprites be-

»BorderLine« beinhaltet – nach dem Aufruf der »InitMasks()«-Funktion – das Ergebnis der ODER-Verknüpfung aller »GEL«-Zeilen. Der erforderliche Speicher errechnet sich daher aus $16 * \text{Width}$. »CollMask« muß das Ergebnis der ODER-Verknüpfung (Bild-Schat-

BOBs in allen Farben

ten) aller »GEL«-Planes aufnehmen können. Der Speicherbedarf ist daher der gleiche wie für eine Plane: $16 * \text{Width} * \text{Height}$.

Die Adresse »SprColors« ist nur bei der Verwendung der VSprites von Interesse. BOBs kommen ohne Farbpalette aus, da sich ihre Farbe aus Anzahl und Belegung der BOB-Bit-Planes ergibt.

Der Pointer »VSBOb« verweist auf die BOB-Struktur, die das Blitter-Objekt näher beschreibt. Bevor wir diese Struktur unter die Lupe nehmen, wollen wir noch die Funktion der restlichen VSprite-Strukturvariablen besprechen.

Die beiden folgenden Werte sind für die Farbgebung des BOBs von größter Bedeutung. Das Byte »PlanePick« teilt dem System mit, welche RastPort-Planes mit den BOB-Planes überschrieben werden sollen. Möchten Sie z.B. in einem Display mit fünf Planes die erste und die vierte RastPort-Plane mit den beiden Planes (Depth = 2) eines BOBs überschreiben, so erhält PlanePick den Wert »%01001 = 9«:

Plane: 5 4 3 2 1
PPick: 0 1 0 0 1

»PlaneOnOff« gibt an, in welche der restlichen Planes, im Beispiel die Planes 2, 3 und 5, der Bildschatten (ODER-Verknüpfung aller Planes) kopiert werden soll (siehe auch unter »ImageShadow«).

Nach der Belegung der VSprite-Struktur errechnen Sie mit dem Aufruf der Funktion »InitMasks(VSprite)« die Werte für »BorderLine« und »CollMask«. Bevor wir das Blitter-Objekt auf dem Bildschirm verwenden können, müssen wir es noch etwas näher beschreiben. Dies erfolgt über die BOB-Struktur (Listing 2). Die Adresse dieser Struktur tragen wir unter »VSBOb« in der VSprite-Struktur ein.

Als erste Variable begegnen wir einem Word zur Aufnahme der speziellen BOB-Flags. Wir unterscheiden sieben Flags:

BOBISCOMP: das BOB gehört zu einer Animationskomponente. Die Adresse »BOB-Comp« der BOB-Struktur zeigt auf die entsprechende Anim-Comp-Struktur (siehe dort).

BOBSAWAY: ist dieses Flag gesetzt, wird das BOB beim nächsten »DrawGList()«-Aufruf gelöscht. Das in »graphics/gels.h« definierte Makro »RemBOB(&BOB)« macht von dieser Möglichkeit Gebrauch. Möchten Sie ein BOB sofort löschen, rufen Sie die Funktion »RemlBOB(&BOB,&RP,&VP)« auf. Beide Funktionen erwarten als Parameter die Adresse der BOB-Struktur. »RemlBOB()« benötigt zusätzlich noch die Adressen der zuständigen Strukturen »RastPort« und »ViewPort«. Beachten Sie bitte, daß im »Double Buffering«-Modus das BOB erst nach dem zweiten Aufruf der »DrawGList()«-Funktion (für die zweite BitMap) gelöscht wird.

BOBNIX: wird vom System nach der Beendigung der »DrawGList()«-Funktion gesetzt, wenn das BOB vollständig gelöscht wurde (wichtig im Double-Buffering-Modus)

BDRAWN: wird vom System gesetzt, nachdem das Blitter-Objekt gezeichnet wurde

BWAITING: wird vom System gesetzt, wenn noch Grafik-Elemente einer »GEL«-Kette vor dem mit dieser Struktur beschriebenen BOB zu zeichnen sind.

SAVEBOB: siehe VSprite-Flag »SAVEBACK«

SAVEPRESERVE: wird im Double-Buffering-Modus vom System benutzt.

Es folgt die Adresse des SaveBuffers. Weiter oben, bei der Besprechung des »SAVEBACK«-Flags, haben wir bereits vom »SaveBuffer« gehört. Hier legt das System den Bereich der Zeichenfläche ab, der vom BOB überschrieben wird. So kann die Fläche nach dem Löschen des BOBs wiederhergestellt werden.

»Image-Shadow« und »Coll-Mask« der VSprite-Struktur weisen beide auf die gleiche Adresse: den Bild-Schatten (ODER-Verknüpfung aller Planes) des BOB-Images. Dieser Bildschatten wird in die mittels »PlaneOnOff« bestimmten Planes kopiert. Auf diese Weise wird die Farbe der einzelnen BOB-Pixel festgelegt, sofern man weniger BOB-Planes als RastPort-Planes definiert hat.

Die Adressen »Before« und »After« zeigen auf die BOB-Strukturen, die der Blitter vor, beziehungsweise nach dem

gerade beschriebenen BOB zeichnen soll. »BOBVSprite« zeigt auf die VSprite-Struktur, deren Adresse »VSBOb« auf die eigene BOB-Struktur verweist. So sind beide Strukturen miteinander verkettet.

Der folgende Pointer zeigt auf eine »AnimComp«-Struktur. Blitter-Objekte können nicht nur auf dem Bildschirm verschoben, sondern auch in der Art eines Daumenkinos animiert werden.

Ein solcher »Film«, bestehend aus mehreren einzelnen Bildern, nennt man eine Animations-Komponente. Mehrere solcher Elemente lassen sich – mittels der »AnimOb«-Struktur – zu einem Animations-Objekt zusammenfassen.

Ein fahrendes Auto mit sich drehenden Rädern, einer flatternden Fahne und dem grinenden Fahrer könnte z.B. als Animations-Objekt definiert sein.

Silicon Motion

Räder, Fahne, Fahrer und Auto wären in diesem Fall eigenständig animierte Grafik-Elemente: Animations-Komponenten.

Das ganze Objekt kann, während die »Komponenten-Filme« laufen, kontinuierlich verschoben und beschleunigt oder verzögert werden. Jedoch ist der dafür notwendige Aufwand entsprechend hoch und eine genaue Beschreibung würde den Rahmen dieses Artikels sprengen.

Es geht weiter mit der Adresse »DBuffer« der BOB-Struktur. Sie zeigt im »Double Buffering«-Modus auf eine »DBufPacket«-Struktur, ansonsten tragen Sie hier 0 ein. Beachten Sie bitte, daß ein Wert ungleich 0 den Double-Buffering-Modus aktiviert. In diesem Modus arbeitet man mit zwei BitMaps, zwischen welchen ständig umgeschaltet werden. Diese Technik, man verwendet sie bei besonders großen BOBs, ist im Beitrag zu »Double Buffering« in diesem Sonderheft auf Seite 68 ausführlich beschrieben. Nebenbei: Manchmal ist es gar nicht notwendig, auf diese komplizierte Technik zurückzugreifen: ein synchronisierendes »WaitTOF()« vor der BOB-Bewegung reicht oft schon aus.

Das System braucht im »Double Buffering«-Modus für die zweite BitMap einen weiteren »SaveBuffer«. Diesen Speicherbereich, er hat die gleiche Größe wie der weiter oben beschriebene »SaveBuffer«, müssen Sie wieder selbst reservieren.

Die Adresse des Speicherbereichs erwartet das System in der Komponente »BufBuffer« der »DBufPacket«-Struktur:

DBufPacket->BufBuffer= Adresse;

Die anderen Werte dieser Struktur, wie etwa die x/y-Position des wiederherzustellenden Display-Bereichs, sind nur für das System wichtig.

Die letzte BOB-Strukturvariable, »BUserExt«, ist vom Typ »BUserStuff«. Das in »graphics/gels.h« definierte Symbol kann wie »VUserStuff« zum Einbinden eigener Variablen in die Struktur eingesetzt werden. Nach der Initialisierung aller Strukturen fügt die Funktion »AddBOB(&BOB,&Rp)« das neue Grafikelement (BOB) in die »GEL«-Liste des »RastPorts« (RP) ein.

Sie sehen, der Initialisierungsaufwand beim Arbeiten mit BOBs ist tatsächlich beachtlich. Drei große Strukturen verlangen die Versorgung ihrer Komponenten mit passenden Werten. Dabei müssen viele Speicherbereiche unterschiedlicher Größe allokiert werden. Die Speichergröße ist von anderen Strukturvariablen abhängig und zu berechnen etc.

Aber kein Grund zur Resignation! Die Vorzüge der Blitter-Objekte sind zu verlockend – und nicht zuletzt: Wir verfügen über ein Werkzeug, mit deren Hilfe die Programmierung der Amiga-Grafikobjekte zum Kinderspiel wird: »GEL-Tool.h« (Listing 3 von Teil 1).

Die Funktionen (bzw. Makros) »GetVSprite()«, »FreeVSprite()« und »MoveVSprite()« haben wir schon im ersten Teil dieses Artikels getestet. Die gleichen Funktionen (Makros) existieren auch für BOBs. Nach dem Aufruf der »GetGels()«-Funktion richtet die Funktion »GetBOB()« die Strukturen VSprite und BOB vollständig ein. Dabei wird der notwendige Speicher automatisch errechnet und allokiert, sowie das BOB in die »GEL«-Liste eingefügt:

```
...
struct BOB *b;
...
b=GetBOB(rp,f,bf,x,y,w,
h,d,me,h1,da,pp,po,dbuf,
vu,bu);
...
```

Wie unschwer zu erkennen ist, liefert »GetBOB()« die Adresse einer BOB-Struktur. Die Parameter bedeuten, von links nach rechts:

rp: die Adresse des RastPorts, in dem der BOB eingesetzt werden soll

f: die Flags der VSprite-Struktur

bf: die Flags der BOB-Struktur

x,y: die Position der linken oberen Ecke des BOBs, in bezug auf die linke obere Display-Ecke

w: die Breite in Words

h: die Höhe in Pixel

d: die Anzahl der BOB-Planes (Depth)

me: das »MeMask«-Byte der VSprite-Struktur für die Kollisionsabfrage

hi: das »HitMask«-Byte der gleichen Struktur

da: die Adresse des Image-Daten-Arrays

pp: der »PlanePick«-Wert

po: der »PlaneOnOff«-Wert

dbuf: eine Boolesche Variable; ist sie gleich »TRUE«, richtet die Funktion den für den Double-Buffering-Modus notwendigen Speicher ein

vu: »VUserStuff«

bu: »BUserStuff«

Alle weiteren Werte errechnet »GetBOB()« selbst.

Das Gegenstück hierzu – FreeBOB(bob,rp,vp,kill) – gibt den für die BOB-Struktur und deren Buffer reservierten Speicher an das System zurück. Der erste Parameter zeigt auf die betreffende BOB-Struktur. Die Bedeutung der anderen Parameter ist mit jenen der »FreeVSprite()«-Funktion identisch.

Das Synonym für »MoveVSprite()« heißt erwartungsgemäß »MoveBOB(&BOB,&RP,&VP,x,y)«. Die Funktion bewegt einen einzelnen BOB um x Pixel in der Horizontalen und um y Pixel in der Vertikalen. Die beiden anderen Parameter sind die Adressen der Strukturen »RastPort« und »ViewPort«.

Unser Beispiel (Listing 3) demonstriert die genannten »GEL-Tool.h«-Funktionen. Ein 192 x 40 Pixel großes BOB mit zwei Planen bewegt sich über den Workbench-Screen und reflektiert an dessen Grenzen.

Da wir Ihnen nicht zumuten wollen, die Riesenmenge an Image-Daten abtippen zu müssen, haben wir auf bereits bestehende Daten zurückgegriffen und so einen recht interessanten BOB geschaffen. Lassen Sie uns einen Blick in den Quellcode des, dank »GEL-Tool.h« recht kurzen, Demos werfen.

Alle Vorarbeiten, wie das Öffnen von Libraries und Windows, übernimmt »OpenW()«. »InitBitMap()« initialisiert darin

eine Dummy-BitMap, die als Plane-Pointer zwei Zeiger auf je eine Hälfte des für das Image reservierten Speicherbereichs erhält.

Mittels »BitBitMap()« kopieren wir die linke obere Screen-Ecke in diesen Bereich, denn dort befindet sich unser Window. So haben wir schnell die Daten für das BOB-Image besorgt: ein BOB mit dem Aussehen eines Windows. Den Aufruf der »GetGelsInfo()«-Funktion richtet die »GelsInfo«-Struktur ein. Vor der Festlegung der beiden anderen Strukturen, legen wir die Kollisionsroutine für den

»GEL«-Rand fest: »BorCon()« negiert den Geschwindigkeitsvektor, der für die Kollision verantwortlich war (siehe Teil 1: Kollision zwischen Graphic-Elementen und Border).

Die Funktion »GetBOB()« macht ihrem Namen alle Ehre und beschafft uns ein »SAVEBACK«-BOB mit den gewünschten Spezifikationen. »DrawGels()« zeichnet die »GEL«-Kette in den RastPort.

In der Hauptfunktion »main()« bewegt »MoveBOB()« unser neu erschaffenes Blitter-Objekt über den Bildschirm. Stellt »GetMessage()« dabei die Betäti-

gung der linken Maustaste fest, so setzen wir in der VSprite-Struktur das »OVERLAY«-Flag. Das BOB wird dann augenblicklich an allen Stellen, an denen die Hintergrundfarbe sichtbar ist – verblüffend aber wahr – durchsichtig.

Nach der Anwahl des Close-Gadgets gibt »CloseW()« den reservierten Speicher wieder an das System zurück und beendet das Programm. Vielleicht ersetzen Sie einmal – nur um zu sehen, was passiert – die beiden Zeilen im »else«-Zweig der »main()«-Funktion durch die folgenden Zeilen:

```
if (code==SELECTDOWN)
BOB->BOBVSprite-
> Flags&= ~SAVEBACK;
else BOB->BOBVSprite-
> Flags | SAVEBACK;
```

Erstaunt? In dieser Art finden Sie sicher noch eine Menge anderer Möglichkeiten zur Manipulation – viel Spaß dabei!

Martin Jobst/so

Arno Gözler ist ein Mann der ersten Stunde. Er schreibt seit über zwei Jahren für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Arno Gözler, Hans-Pinsel-Straße 2, 8013 Haar bei München).

```
/*
cc BOB.c +L
ln BOB.o -lc32
*/
/*
\\ Gözly's BOB Demo
*/
#include <intuition/intuitionbase.h>
#include <graphics/collide.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <functions.h>
#include <graphics/gels.h>
#include "GELTool.h"

#define HEIGHT 40
#define WIDTH 192

WORD vX=4, vY=2;
PLANEPTR Image;
TEXT DName[]=" Gözly's BOB-Demo";
struct NewWindow nwin={
0,0,WIDTH,HEIGHT,0,1,CLOSEWINDOW|MOUSEBUTTONS|WINDOWCLOSE|ACTIVATE,
NULL,NULL,DName,NULL,NULL,0,0,0,0,WBENCHSCREEN
};
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Window *win;
struct Screen *Screen;
struct RastPort *RP;
struct BitMap *BM,DBM;
struct ViewPort *VP;
struct GelsInfo *GelsInfo;
struct BOB *BOB;
/*
\\ Funktionen
*/
VOID BorCon(vsp,witch)
struct VSprite *vsp;
SHORT witch;
{
if(((witch&LEFTHIT)==LEFTHIT)||((witch&RIGHTHIT)==RIGHTHIT))
vX*=-1;
if(((witch&TOPHIT)==TOPHIT)||((witch&BOTTOMHIT)==BOTTOMHIT))
vY*=-1;
}
VOID CloseW()
{
if(BOB) FreeBOB(BOB,RP,VP,1);
if(Image) FreeRaster(Image,WIDTH,2*HEIGHT);
if(GelsInfo) FreeGelsInfo(GelsInfo);
if(win) CloseWindow(win);
if(GfxBase) CloseLibrary(GfxBase);
if(IntuitionBase) CloseLibrary(IntuitionBase);
exit(0);
}
VOID OpenW()
{
if(!(IntuitionBase=(struct IntuitionBase *)
OpenLibrary("intuition.library",0))) CloseW();
```

```
Screen=IntuitionBase->ActiveScreen;
if(!(GfxBase=(struct GfxBase *)
OpenLibrary("graphics.library",0))) CloseW();
if(!(win=OpenWindow(&nwin))) CloseW();
VP=&Screen->ViewPort;
RP=&Screen->RastPort;
BM=RP->BitMap;
InitBitMap(&DBM,2,192,20);
if(!(Image=(PLANEPTR)AllocRaster(WIDTH,2*HEIGHT))) CloseW();
DBM.Planes[0]=Image;
DBM.Planes[1]=Image+RASSIZE(WIDTH,HEIGHT);
BlitBitMap(BM,0,0,&DBM,0,0,WIDTH,HEIGHT,0x00,0xFF,NULL);
Move(RP,8,26);
Text(RP,"Klick hier für OVERLAY",22);
if(!(GelsInfo=GetGelsInfo(RP,0,0,0,640,256))) CloseW();
SetCollision(0,BorCon,GelsInfo);
if(!(BOB=GetBOB(RP,SAVEBACK,0,0,0,WIDTH/16,HEIGHT,2,0,1,
Image,3,0,FALSE,0,0)))
CloseW();
DrawGels(RP,VP);
}
ULONG GetMessage(code)
USHORT *code;
{
struct IntuiMessage *msg;
ULONG class=0;

if(msg=(struct IntuiMessage *)GetMsg(win->UserPort)){
class=msg->Class;
*code=msg->Code;
ReplyMsg(msg);
}
return(class);
}
VOID main()
{
USHORT code;
ULONG class;

OpenW();
while(1){
class=GetMessage(&code);
if(class==CLOSEWINDOW) break;
else if(class==MOUSEBUTTONS){
if(code==SELECTDOWN) BOB->BOBVSprite->Flags|=
OVERLAY;
else BOB->BOBVSprite->Flags&= ~OVERLAY;
}
WaitTOF();
MoveBOB(BOB,RP,VP,vX,vY);
DoCollision(RP);
}
CloseW();
}
```

Listing 3. Das BOB-Demoprogramm verwandelt ein Window in ein Blitter-Objekt

Jonglieren mit zwei Displays ist für die meisten Spieleprogrammierer unerlässlich, da nur so ohne großen Aufwand eine stehende Grafik in einer vorbeiziehenden Landschaft dargestellt werden kann. Um den Aufbau des DualPlayfield-Modus leichter zu verstehen, sehen wir uns zunächst die BitMap-Struktur (Listing 1) an. Sie enthält wichtige Informationen über den Aufbau des Displays. So findet man darin auch die Adressen der einzelnen Bitplanes. Das Array »Planes[8]« bietet Platz für acht Adressen, die auf die Planes verweisen. Man nennt diese Adressen »Plane-Pointer«. Im Headerfile graphics/gfx.h ist sogar ein Datentyp dafür definiert:

```
typedef UBYTE *PLANEPTR;
```

In der gegenwärtigen Betriebssystemversion können jedoch nur sechs Planes zum Aufbau eines Displays genutzt werden. Dabei ist Plane 6 schon eine »Zusatzplane«, das nur bei den Spezial-ViewModi Verwendung findet. Einen dieser Modi durchleuchten wir wegen seiner Eignung zur Spieleprogrammierung genauer: den DualPlayfield-Modus.

Was verbirgt sich hinter diesem Begriff? Wie bereits eingangs erwähnt, setzt sich bei der Verwendung von DualPlayfield das Bild auf dem Monitor aus zwei übereinanderliegenden Displays (Playfields) zusammen. Sie sind beide, ob-

Zwei Bilder in einem

wohl sie einem gemeinsamen ViewPort angehören, voneinander völlig unabhängig. Jedes Playfield verfügt über eine eigene BitMap-Struktur mit bis zu drei Planes, über eine eigene RastPort-Struktur und schließlich noch über eine eigene RasInfo-Struktur. Die erste RasInfo-Struktur ist mit der zweiten über die Adresse »Next« verkettenet. Die ViewPort-Strukturkomponente »RasInfo« verweist auf das erste Glied dieser Strukturkette:

```
ViewPort.RasInfo=
  RasInfo_A;
RasInfo_A.Next =
  RasInfo_B;
RasInfo_B.Next =NULL;
```

In diesem Beispiel enthält die Struktur namens RasInfo_A die Informationen über Play-

Der DualPlayfield-Modus

Nimm

Der Amiga erlaubt die gleichzeitige Darstellung von zwei übereinanderliegenden Displays. Diese auf den ersten Blick sinnlose Einrichtung entpuppt sich als wahres Juwel in der Spieleprogrammierung.

field A, welches bildlich gesprochen über Playfield B liegt und dieses teilweise verdeckt. Wieso teilweise? Bei der Verwendung von drei Bitplanes pro Playfield stehen für jedes Playfield $2^3 = 8$ Farben zur Verfügung. Die ViewPort-ColorMap ist aber für 16 Farben ausgelegt:

```
ViewPort.ColorMap=
  GetColorMap(16);
```

Die ersten acht Einträge der ColorMap sind die Farben 0 bis 7 des ersten Playfields, dagegen entsprechen die zweiten acht Einträge den Farben 0 bis 7 des zweiten Playfields. Da man mit drei Planes nur acht Farben darstellen kann, werden die ersten acht Einträge auf die folgenden acht kopiert. Somit ist die Farbe 0 gleich Farbe 8 und 1 gleich 9 usw. Hier zwei Beispiele, die das gleiche bewirken.

```
SetAPen(&RP_B, 5);
SetAPen(&RP_B, 13);
```

Der erste Eintrag der ColorMap bestimmt wie gewohnt die Hintergrundfarbe. Wenn Sie nun in einem Playfield die Hintergrundfarbe (Eintrag 0 oder 8 der ColorMap, je nach Playfield) setzen, zeichnen Sie nicht etwa in der erwarteten Farbe, sondern transparent! Das »dar-

unterliegende« Playfield wird daher an diesen Stellen sichtbar. Es stellt sich die Frage nach dem Sinn der zweiten transparenten »Farbe« (Eintrag Nummer 8 der ColorMap), da sich doch unter Playfield B nicht noch ein drittes Display befindet. Ganz einfach: Die Anordnung der Displays kann durch das Setzen bzw. Löschen des Mode-Flags

PFBA

beliebig verändert werden. Ist PFBA gesetzt (Mode = DUALPF II PFBA), liegen die Playfields gemäß den beiden letzten Buchstaben des Symbols, also B vor A. Löscht man PFBA (Mode = DUALPF), so herrscht wieder, wie zu Beginn, die Anordnung AB. Mit dieser Methode läßt sich übrigens sehr be-

können auf Playfield A dargestellt werden, während auf Playfield B das eigentliche Spiel zu sehen ist. Dabei müssen Sie zu keiner Zeit abfragen, ob beispielsweise ein Laserstrahl die Cockpit-Grafik Ihres UFOs zu überzeichnen droht, da beide Playfields völlig unabhängig voneinander sind. Die Displays können beliebig gegeneinander verschoben werden. Ein Playfield kann sogar nach allen Richtungen gescrollt werden, ohne daß dies einen Einfluß auf das andere hat. Die zuletzt genannte Fähigkeit haben wir uns in unserem Beispielprogramm zunutze gemacht, um den Speicher des Amiga einmal genau unter die Lupe zu nehmen. In diesem Demo (Listing 2) sind zwei Playfields mit je drei Planes definiert. Während die Grafik in Playfield A ständig sichtbar ist, scrollt B, den Bewegungen des an Port II angeschlossenen Joysticks folgend, durch den Arbeitsspeicher Ihres Amigas! Ein Druck auf den Feuerknopf beendet das Programm.

Das Listing hilft

Bevor Sie in dem Arbeitsspeicher wühlen, noch einen Blick ins Listing? Die Funktion OpenW() öffnet ein einfaches Grafik-Display im DUALPF-Modus. Die Strukturen RastPort, BitMap und RasInfo sind in einem Array angelegt, so daß die doppelte Initialisierung dieser Strukturen in einer Schleife erfolgen kann. Der bedingte Ausdruck im Schleifenrumpf weist RasInfo[0].Next

```
struct BitMap
{
  UWORD BytesPerRow; /* Bytes pro BitMap-Zeile (Breite/8) */
  UWORD Rows; /* Anzahl der Zeilen (Höhe) */
  UBYTE Flags; /* ??? ungenutzt */
  UBYTE Depth; /* Anzahl der BitPlanes (BitEbenen) */
  UWORD pad; /* Platzhalter - PLANEPTR beginnen auf LONGWORD */
  PLANEPTR Planes[8]; /* die Adressen der BitEbenen im Speicher */
};
```

Listing 1. Die BitMap-Struktur verwaltet elementare Daten über das Grafikdisplay

quem das sog. »Double-Buffering« realisieren. Mit dieser Technik befaßt sich jedoch ein anderer Artikel auf Seite 68 in diesem Heft. Uns interessiert was man denn mit dem DualPlayfield-Modus in Spielen überhaupt anfangen kann. Nun, Elemente wie etwa das Armaturenbrett eines Autos, das Cockpit eines Raumfahrzeugs oder die Spielstandsanzeige

die Adresse von RasInfo[1] zu, während RasInfo[1].Next den Wert Null erhält. Somit ist festgelegt: RasInfo[0] gehört zu Playfield A und RasInfo[1] zu Playfield B. A liegt über B, da in ViewPort.Modes das Flag PFBA nicht gesetzt ist. Die Funktion CloseW() schließt alle Ressourcen vor Programmende. Nachdem InitStick() den Joystick an Port II angemeldet hat, baut


```

/*
Aztec C 3.4a
cc DualPF.c +L
ln DualPF.o -lc32
*/
/*
\\ Gölzy's DualPlayfield-Demo
*/
#include <exec/memory.h>
#include <exec/types.h>
#include <graphics/gfxbase.h>
#include <graphics/view.h>
#include <devices/inputevent.h>
#include <devices/gameport.h>
#include <functions.h>

#define TMPDIM 200
#define WIDTH 320
#define HEIGHT 256

WORD Buffer[10], ColTab[16]={
    0x888,0x999,0xAAA,0xBBB,0xCCC,0xDDD,0xEEE,0xFFF,
    0x000,0xF00,0x0F0,0x00F,0xFC0,0xFF,0xC55,0x5F5
};
ULONG GPDev;
struct GamePortTrigger GPTrigger={
    GPTF_UPKEYS|GPTF_DOWNKEYS,0,1, 1
};
struct GfxBase *GfxBase;
struct View *OldView, View;
struct MsgPort *GPort;
struct IOSReq *GPRRequest;
struct InputEvent ReadStick;
struct ViewPort VPort;
struct RasInfo RasInfo[2];
struct BitMap BMap[2];
struct RastPort RP[2];
struct TmpRas Raster;
struct AreaInfo AInfo;
struct PlanePTR Bitplane;

VOID CloseW()
{
    REGISTER SHORT i,j;

    if(OldView){
        LoadView(OldView);
        WaitTOF();
    }
    FreeVPortCopLists(&VPort);
    FreeCprList(View.LOFCprList);
    if(VPort.ColorMap) FreeColorMap(VPort.ColorMap);
    for(i=0;i<2;i++)
        for(j=0;j<3;j++){
            if(BMap[i].Planes[j])
                FreeRaster(BMap[i].Planes[j],WIDTH,HEIGHT);
        }
    if(Bitplane) FreeRaster(Bitplane,TMPDIM,TMPDIM);
    if(GPDev) CloseDevice(GPRRequest);
    if(GPRRequest) DeleteStdIO(GPRRequest);
    if(GPort) DeletePort(GPort);
    if(GfxBase) CloseLibrary(GfxBase);
    exit(0);
}

VOID OpenW()
{
    REGISTER SHORT i,j;

    if(!(GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library",0))) CloseW();
    if(!(GPort=CreatePort("GPort",0))) CloseW();
    if(!(GPRRequest=CreateStdIO(GPort))) CloseW();
    if(GPDev=(ULONG)
        OpenDevice("gameport.device",1,GPRRequest,0)) CloseW();
    if(!(Bitplane=AllocRaster(TMPDIM,TMPDIM))) CloseW();
    InitTmpRas(&Raster,Bitplane,RASSIZE(TMPDIM,TMPDIM));
    InitArea(&AInfo,Buffer,2);
    OldView=GfxBase->ActiView;
    InitView(&View);
    View.ViewPort=&VPort;
    View.Modes =DUALPF;

```

```

InitVPort(&VPort);
VPort.DWidth =WIDTH;
VPort.DHeight =HEIGHT;
VPort.RasInfo =&RasInfo[0];
VPort.ColorMap =GetColorMap(16);
VPort.Modes =DUALPF;
for(i=0;i<2;i++){
    InitRastPort(&RP[i]);
    InitBitMap(&BMap[i],3,WIDTH,HEIGHT);
    for(j=0;j<3;j++){
        if(!BMap[i].Planes[j]=(PLANEPTR)
            AllocRaster(WIDTH,HEIGHT))) CloseW();
        BitClear(BMap[i].Planes[j],RASSIZE(WIDTH,HEIGHT),0);
    }
    RasInfo[i].BitMap =&BMap[i];
    RasInfo[i].RxOffset=0;
    RasInfo[i].RyOffset=0;
    RasInfo[i].Next =i?NULL:&RasInfo[1];
    RP[i].BitMap =&BMap[i];
    RP[i].TmpRas =&Raster;
    RP[i].AreaInfo =&AInfo;
}
LoadRGB4(&VPort,ColTab,16);
MakeVPort(&View,&VPort);
MrgCop(&View);
}

VOID SetGPRRequest(doit,c,l,d)
    BOOL doit;
    UWORD c;
    ULONG l;
    APTR d;
{
    GPRRequest->io_Command =c;
    GPRRequest->io_Length =l;
    GPRRequest->io_Data =d;
    if(doit) DoIO(GPRRequest);
}

VOID InitStick()
{
    UBYTE Data=GPCT_RELJOYSTICK;
    ULONG lg=sizeof(struct GamePortTrigger),
        ll=sizeof(struct InputEvent);

    SetGPRRequest(1,GPD_SETCTYPE,1,&Data);
    SetGPRRequest(1,GPD_SETTRIGGER,lg,&GPTrigger);
    SetGPRRequest(0,GPD_READEVENT,ll,&ReadStick);
}

VOID DrawC(x,y,r)
    SHORT x,y,r;
{
    AreaEllipse(&RP[0],x,y,r,r);
    AreaEnd(&RP[0]);
}

VOID Print(RP,text,col,x,y)
    struct RastPort *RP;
    STRPTR text;
    SHORT col,x,y;
{
    SHORT i,len=strlen(text);

    SetDrMd(RP,JAM1);
    for(i=1;i>=0;i--){
        SetAPen(RP,i?2:col);
        Move(RP,x+i,y+i);
        Text(RP,text,len);
    }
}

VOID DrawVP()
{
    REGISTER SHORT i,x,y;

    for(i=1;i<9;i++){

```

Listing 2. Das DualPlayfield-Demo gestattet einen ungetrübten Blick in den Arbeitsspeicher des Amiga


```

SetAPen(&RP[0],i==8?0:1);
DrawC( 90,128,95-5*i);
DrawC(229,128,95-5*i);
}
Print(&RP[0],"*- Speicher-Brille *-","7,68,25);
Print(&RP[0],"> > Joystick in Port II <<","5,68,230);
Print(&RP[1],"Viel Spaß","4,55,125);
Print(&RP[1],"mit dem","4,60,140);
Print(&RP[1],"Dual ","1,200,110);
Print(&RP[1],"Playfield","6,193,130);
Print(&RP[1],"View-Modus!","6,185,150);
LoadView(&View);
}

VOID main ()
{
    UBYTE Data=GPCT_NOCONTROLLER;
    OpenW();
    InitStick();
}

```

DrawVP() die Grafik und den Text beider Playfields auf. Die Variable RP[0] steht für den RastPort des Playfields A und RP[1] für den von B. Der letzte Funktionsaufruf in DrawVP() öffnet das fertig erstellte Display im DualPlayfield-Modus. In der Hauptschleife main() überwachen wir in einer Endlosschleife den GamePort. Je nach Richtung des Steuerknüppels verschieben wir Playfield B unter A um 8 Pixel, was

recht rasant ist. Dabei beachten wir die Grenzen des RastPorts B nicht. Somit wird, sobald wir ihn verlassen, der Inhalt des Arbeitsspeichers sichtbar. Besonders interessant sieht der Speicher nach einem Grafikspiel aus! Registriert der GamePort den Feuerknopf, unterbrechen wir die Schleife, um mit CloseW() das Programm zu beenden. Es sollte noch erwähnt werden, daß der ganze »Zauber« durch die Betätigung der

```

DrawVP();
while(1){
    SendIO(GPRequest);
    WaitIO(GPRequest);
    if(ReadStick.ie_Code==IECODE_LBUTTON) break;
    else{
        if(ReadStick.ie_X)
            RasInfo[1].RxOffset+=ReadStick.ie_X>0?8:-8;
        if(ReadStick.ie_Y)
            RasInfo[1].RyOffset+=ReadStick.ie_Y>0?8:-8;
        ScrollVPort(&VPort);
    }
}
SetGPRequest(1,GPD_SETCTYPE,1,&Data);
CloseW();
}

```

Listing 2. (Schluß)

Tastenkombination »Linke Amiga n« unterbrochen wird. Der Workbenchscreen kommt nach vorne, kann aber, weil wir nicht mit einem Screen arbeiten, nicht mehr hinter unser Display geklappt werden. Abhilfe schafft z.B. ein Aufruf der LoadView()-Funktion, worauf wir aber in unserem Demo verzichtet haben. Vielleicht konnten wir Sie mit diesem kurzen Artikel etwas neugierig machen, oder haben Sie sogar schon eine

Idee, wo Sie den Modus einsetzen können? Wenn nicht, experimentieren Sie doch einfach etwas mit dem Demo-Listing, um den DualPlayfield-Modus besser kennenzulernen. *pe*

Arno Gölzer ist ein Mann der ersten Stunde und schreibt schon einige Jahre für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Herrn Gölzer, Hans-Pinsel-Straße 2, 8013 Haar bei München).

Fließende Bewegungen durch Double-Buffering

Flacker-Fixer

Von Arno Gölzer

Schweißgenäht, den Steuerknüppel fest mit beiden Händen umschlungen, kämpft der Spieler in seiner Lieblingsflugsimulation gegen den Absturz – alles vergebens! Macht nichts – fünf Sekunden nach dem Crash, so lange braucht er, um Chips und Cola nachzufüllen, wagt er einen neuen Start. In atemberaubender Geschwindigkeit fegt er mit seinem Flugzeug über die Startbahn. Die Landschaft fliegt nur so an ihm vorbei – diesmal klappt's bestimmt...

Erst eine schnelle, sauber animierte Grafik läßt ein Spiel so realistisch erscheinen, daß eine derartige Begeisterung bei dem Spieler hervorgerufen wird. »Ruckelgrafiken« sind nicht gefragt – der Amiga kann es besser. Doch wie geht man als Programmierer vor? Zunächst können Sie festhalten: bei einer Animation muß, um die Illusion einer Bewegung zu erwecken, die Grafik ständig verändert werden.

Für weiche und fließende Bewegungen verwendet man eine besondere Technik: das Double-Buffering. Wir stellen Ihnen mehrere Methoden dazu vor.

Zerlegen wir einen Bewegungsabschnitt, so ergeben sich die Schritte a) Grafik Neuberechnen und b) neue Grafik darstellen. Beide Schritte wechseln sich ständig ab. Dabei ist es wichtig zu unterscheiden, ob vor der Darstellung die alte Grafik gelöscht werden muß oder diese mit dem neuen Bild überschrieben wird. Die Animationssysteme einiger Malprogramme arbeiten mit der zuletzt genannten Methode. Geht es aber um Vektorgrafiken, wie bei unserem Beispiel »Flugsimulation«, muß das Zeichen-

display vor jedem Animations-schritt gelöscht werden. Arbeitet man mit einem Display, macht sich das Löschen zwischen zwei Darstellungen als unangenehmes Flimmern bemerkbar. Darüber hinaus benötigen aufwendigere Grafiken relativ lange zum Aufbau, was sich störend bemerkbar macht.

Die Lösung heißt »Double-Buffering«. Bei dieser Technik wird, der Name läßt es vermuten, mit zwei hintereinanderliegenden Zeichenpuffern oder mit zwei Displays gearbeitet. Während die Grafik auf dem

vorderen Bildschirm zu sehen ist, wird der hintere gelöscht und nach einer Neuberechnung die abgeänderte Grafik dargestellt. Dann vertauscht man die Bildschirme und das Spiel beginnt von neuem: hintere Grafik löschen, Grafik verdeckt zeichnen, Bildschirme vertauschen etc. Alle bislang störende Effekte wie Löschen oder Neuzeichnen des Bildes erfolgen versteckt im Hintergrund. Beim Arbeiten mit Intuition können Windows und Screens als Zeichenfläche genutzt werden:

```

for(index=0,..;...){
    ZeichneGrafik(&Window
[index]);
    WindowToFront(&Window
[index]);
    index=index?0:1;
}
...

```

In diesem Listingausschnitt benutzen wir zwei gleich große Fenster, die sich an der gleichen Screen-Position befinden.

Im Schleifenrumpf des Programmfragments baut die Funktion »ZeichneGrafik()« auf dem im Hintergrund liegenden Window[0] eine Grafik auf. Dann klappt WindowToFront() das Fenster in den Vordergrund. Der bedingte Ausdruck schaltet die Indexvariable wie ein FlipFlop je nach Ausgangswert auf 0 oder 1 um. Das bedeutet: Im nächsten Schleifendurchlauf wird die Grafik in Window[1] aufgebaut, das dann nach vorne gebracht wird. Beide Windows wechseln sich ab, bis das Schleifenkriterium erfüllt ist.

Bei Screens oder einem selbst erstellten Grafikdisplay, kann das Double-Buffering Flag »PFBA« genutzt werden. Das Display wird im DualPlayfield-Modus erstellt (Sie finden in diesem Heft auf Seite 66 einen Artikel, der sich mit diesem View-Modus befaßt). Das zu Beginn hintenliegende Playfield B dient als Extra-Buffer und wird mit der Grafik versehen. Ist diese aufgebaut, wird in der View-Struktur das Flag »PFBA« gesetzt und die Copperlisten neu berechnet, woraufhin die Playfields die Plätze tauschen. Playfield B liegt nun vor A und ist demnach sichtbar. Der Vorgang wiederholt sich mit gedrehten Vorzeichen: das unsichtbare Playfield A löschen und die neu berechnete Grafik darstellen, PFBA-Flag löschen und Copperlisten Neuberechnen lassen. Dann befinden sich die Displays in ihrer ursprünglichen Lage. Anschaulich wird diese Methode, wenn Sie in unserem DualPlayfield-Demolisting (Seite 66) die zweite if-Abfrage innerhalb der while-Schleife durch folgende Zeilen ersetzen:

```
if(ReadStick.ie_X){
VPort.Modes=ReadStick.
ie_X>0
?DUALPF|PFBA:DUALPF;
MakeVPort(&View,&VPort);
MrgCop(&View);
}
```

Nach erfolgreichem Compilieren können Sie durch Bewegen des Joysticks in Y-Richtung durch den Speicher »wandern«. Bewegen Sie ihn jedoch in X-Richtung, werden die Playfields vertauscht. Damit ist der Speicherinhalt im Vordergrund und die »Brille« im Hintergrund zu sehen.

Die beschriebenen Double-Buffering-Arten sind einfach zu realisieren, sie haben aber gravierende Nachteile. So ist das Umschalten der Windows lang-

sam und Sie können im DualPlayfield-Modus nur drei Planes pro Display aktivieren. Der Blitter bietet uns eine Möglichkeit, diese Nachteile zu umgehen: Sie initialisieren zwei Buffer (Windows oder auch ein Window und ein RastPort), zeichnen auf die unsichtbare Zeichenfläche und »clippen« das Bild mit der Blitterfunktion ClipBlit() in den Vordergrund. Die Blitter-Funktion finden Sie in unserem Artikel auf Seite 72 näher erläutert. Diese Technik ist für kleinere animierte Grafi-

Der Blitter macht's

ken geeignet und nicht komplizierter zu handhaben als das Jonglieren mit Windows. BLTArrow.c (Listing 1) demonstriert diese Technik. Das Demoprogramm öffnet zwei »übereinander« liegende Fenster und bewegt mit der eben beschriebenen Methode einen dreidimensionalen »Drahtgitter-Pfeil«. Der Clou: Sie können das Geschehen im Extra-Buffer beobachten, indem Sie das vordere Fenster etwas zur Seite schieben! Obwohl die Grafik nicht sehr aufwendig ist, werden sofort alle Vorteile des Double-Bufferings klar – die Animation auf dem vorderen Window ist ruhiger und flüssiger als die andere.

Wie funktioniert das Programm? Betrachten wir die Funktion main(). Der Aufruf der OpenW()-Funktion öffnet alle notwendigen Libraries und die Fenster. NewDraw() löscht den Zeichenbereich des hinteren Windows (Window[0]), zeichnet den Pfeil und clippt die Zeichnung in das vordere Window. Im Rumpf der while-Schleife wird, solange das CloseGadget nicht betätigt wurde, die Zeichnung mittels Turn() gedreht. Turn() berechnet die Positionen der Eckpunkte neu und ruft die Funktion NewDraw() auf – der Kreis ist geschlossen. CloseW() schließt vor Programmende alle Ressourcen.

Mit dieser Methode verlieren Sie keine Zeit durch das Wechseln der Windows und die Anzahl der Planes wird nicht beeinträchtigt.

Diese Vorteile bietet auch eine kompliziertere Technik, welche wir als letztes besprechen: Double-Buffering mit dem Copper. Wir benötigen zum Double-Buffering zwei getrennte RastPorts zum Zeichnen und daher auch zwei getrennte BitMap-Strukturen. Beide BitMaps

müssen abwechselnd im Vordergrund sein. Wie wird eine BitMap sichtbar? Die View-Struktur enthält die Adresse der ViewPort-Struktur, diese wiederum verweist auf die RasInfo-Struktur, welche endlich die Adresse der BitMap-Struktur enthält. Die Funktionen MakeVPort() und MrgCop() erstellen die Copperlisten des Displays. LoadView() macht dieses sichtbar. Um zwischen beiden BitMaps umschalten zu können, benötigen wir zwei Copperlisten. Beide Listen müssen eine andere BitMap berücksichtigen. Ein Umschalten der Copperlisten bringt uns den gewünschten Effekt. Die Adressen der Copperlisten finden wir in der View-Struktur. Wenn Sie nicht im Interlace-Modus arbeiten, ist nur View.LOFCprList von Interesse. Also los: Wir lassen für die erste BitMap-Struktur eine Copperliste berechnen. Die Adresse der Liste sichern wir in einer Variablen. Dann geben wir der RasInfo-Struktur die Adresse der anderen BitMap-Struktur und lassen eine weitere Copperliste erstellen, ohne das Display zu laden. Zuvor müssen wir die View-Variable LOFCprList (und gegebenenfalls auch SHFCprList) auf Null setzen, da sich sonst der Amiga hartnäckig weigert, eine weitere Liste zu berechnen. Auch die Adresse dieser Liste wird gesichert:

```
CList[i]=View.LOFCprList;
List;
```

Am besten übernehmen Sie beide Adressen in ein Array, so können Sie später sehr einfach zwischen den Copperlisten umschalten. Jetzt können Sie das Display mittels LoadView() sichtbar machen. Unsere Zeichnung geben wir auf dem gerade nicht sichtbaren RastPort aus. Danach geben wir der View-Struktur die Adresse der Copperliste bekannt, bei der die RasInfo-Struktur die gleiche BitMap-Adresse enthält, des gerade benutzten RasPorts:

```
View.LOFCprList=
CList[i];
```

Jetzt laden wir das Display mit LoadView(&View). Der Vorgang wiederholt sich: Zeichnen auf den verdeckten RastPort, die Adressen der Copperlisten vertauschen und Aufruf der LoadView()-Funktion...

Das Beispiel »CPRarrow« (Listing 2) verdeutlicht diesen Prozeß. Dieses Programm stellt einen Pfeil als 3D-Drahtgitter dar, allerdings ist er diesmal etwas

größer. Er läßt sich mit dem Joystick in jede beliebige Lage drehen. Das Programm wird mit dem Feuerknopf beendet.

OpenW() initialisiert in der ersten for-Schleife die beiden BitMap- und RastPort-Strukturen. In der anderen Schleife, weiter unten in der Funktion, löschen wir bei jedem Schleifendurchgang die View-Komponente LOFCprList. Eigentlich ist dies nur bei jedem zweiten Schleifendurchgang nötig. Die RasInfo-Struktur erhält die Adresse der ersten BitMap-Struktur. Es folgt die Berechnung der ersten Copperliste, deren Adresse wir in CList[0] sichern. Beim zweiten Durchlauf der Schleife erhält RasInfo die Adresse der zweiten BitMap-Struktur. Die Adresse der zweiten Copperliste wandert nach CList[1]. Wir benötigen zum

CPRarrow zieht alle Register

Umschalten der Listen eine Variable, die als Index dient: CListNr. Die View-Struktur verweist nach Ende der OpenW()-Funktion auf die Liste der in CList[1] gesicherten Adresse. Da wir auf den verdeckten RastPort zeichnen möchten, muß unsere Index-Variable zu Beginn den Wert 0 erhalten (RastPort[0] nicht sichtbar). Nach der Initialisierung des Joysticks ruft main() die Funktion NewDraw() auf. Dort wird mit dem Index der Zeichenrastport festgelegt und die Zeichnung aufgebaut. Danach geben wir die Adresse der entsprechenden Copperliste bekannt (gleicher Index) und laden das Display – die Zeichnung wird sichtbar. Jetzt muß für den nächsten Aufruf der NewDraw()-Funktion der Index umgeschaltet werden. Der bedingte Ausdruck in der letzten Zeile von NewDraw() übernimmt diese Aufgabe, er fungiert als Flip-Flop. Die Funktion NewDraw() zeichnet beim nächsten Aufruf in den anderen, unsichtbaren RastPort, welcher wieder geladen wird, Index umschalten etc. ...

Mit diesen vier Double-Buffering-Verfahren sollten die Zeiten der »Ruckel- und Flimmergrafiken« endgültig vorbei sein. Wann verbannen Sie das Flackern vom Bildschirm? pe

Arno Götzler ist ein Mann der ersten Stunde und schreibt schon einige Jahre für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Herrn Arno Götzler, Hans-Pinsel-Straße 2, 8013 Haar bei München).


```

/* Aztec C 3.4a
cc BLTArrow.c +L
ln BLTArrow.o -lm32 -lc32 */
/* \\ Gölzy's >>BLITTER<< Double-Buffering Demo */

#include <intuition/intuitionbase.h>
#include <graphics/gfxmacros.h>
#include <exec/types.h>
#include <functions.h>

#define WIDTH 140
#define HEIGHT 90

SHORT Kopf[7]={
    4,0,1,2,0,3,4
};
SHORT Rumpf[9]={
    6,7,8,5,9,10,11,12,9
};
FLOAT px[13]=/* X-Positionen der Eckpunkte des Pfeils */
    -30.0,-15.0,-15.0,-15.0,-15.0,-15.0,
    -15.0,-15.0,-15.0,30.0,30.0,30.0,30.0
};
FLOAT py[13]=/* Y-Positionen der Eckpunkte des Pfeils */
    0.0,10.0,10.0,-10.0,-10.0,5.0,
    5.0,-5.0,-5.0,5.0,5.0,-5.0,-5.0
};
FLOAT pz[13]=/* Z-Positionen der Eckpunkte des Pfeils */
    0.0,10.0,-10.0,-10.0,10.0,5.0,
    -5.0,-5.0,5.0,5.0,-5.0,-5.0,5.0
};
struct NewWindow nwin={
    60,30,WIDTH,HEIGHT,-1,-1,CLOSEWINDOW,0,NULL,NULL,
    (STRPTR)"Gölzy's Arrow II",NULL,NULL,0,0,0,0,WBENCHSCREEN
};
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct MathTransBase *MathTransBase;
struct Window *win[2];
struct RastPort *RP[2];

/*
\\ Funktionen
*/
VOID CloseW()
{
    REGISTER SHORT i;
    for(i=0;i<2;i++){
        if(win[i]) CloseWindow(win[i]);
        if(MathTransBase) CloseLibrary(MathTransBase);
        if(GfxBase) CloseLibrary(GfxBase);
        if(IntuitionBase) CloseLibrary(IntuitionBase);
        exit(0);
    }
}
VOID OpenW()
{
    REGISTER SHORT i;
    if(!(IntuitionBase=(struct IntuitionBase *)
        OpenLibrary("intuition.library",0))) CloseW();
    if(!(GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library",0))) CloseW();
    if(!(MathTransBase=(struct MathTransBase *)
        OpenLibrary("mathtrans.library",0))) CloseW();
    for(i=0;i<2;i++){
        nwin.Flags|=1?ACTIVATE|WINDOWCLOSE|WINDOWDRAG:0;
        if(!(win[i]=OpenWindow(&nwin))) CloseW();
        RP[i]=win[i]->RPort;
    }
}
VOID NewDraw()
{
    REGISTER SHORT i,x,y,index;
    SetAPen(RP[0],2);
    RectFill(RP[0],2,10,WIDTH-4,HEIGHT-2);
    SetAPen(RP[0],3);
    Move(RP[0],2*(SHORT)px[1]+WIDTH/2,(SHORT)py[1]+HEIGHT/2);
    for(i=0;i<7;i++){
        index=Kopf[i];
        x=2*(SHORT)px[index]+WIDTH/2;
        y=(SHORT)py[index]+HEIGHT/2;
        Draw(RP[0],x,y);
    }
    Move(RP[0],2*(SHORT)px[2]+WIDTH/2,(SHORT)py[2]+HEIGHT/2);
    Draw(RP[0],2*(SHORT)px[3]+WIDTH/2,(SHORT)py[3]+HEIGHT/2);

```

```

Move(RP[0],2*(SHORT)px[5]+WIDTH/2,(SHORT)py[5]+HEIGHT/2);
for(i=0;i<9;i++){
    index=Rumpf[i];
    x=2*(SHORT)px[index]+WIDTH/2;
    y=(SHORT)py[index]+HEIGHT/2;
    Draw(RP[0],x,y);
}
for(i=0;i<3;i++){
    Move(RP[0],2*(SHORT)px[6+i]+WIDTH/2,(SHORT)py[6+i]+HEIGHT/2);
    Draw(RP[0],2*(SHORT)px[10+i]+WIDTH/2,(SHORT)py[10+i]+HEIGHT/2);
}
ClipBlit(RP[0],2,10,RP[1],2,10,WIDTH-4,HEIGHT-11,0xc0);
}
VOID Turn(wx,wy)
    FLOAT wx,wy;
{
    REGISTER SHORT i;
    FLOAT xp,yp,zp,swx,cwx,swy,cwy;
    swx=sin(wx);
    cwx=cos(wx);
    swy=sin(wy);
    cwy=cos(wy);
    for(i=0;i<13;i++){
        yp=py[i];
        zp=pz[i];
        py[i]=yp*cwx+zp*swx;
        pz[i]=-yp*swx+zp*cwx;
        xp=px[i];
        zp=pz[i];
        px[i]=xp*cwy+zp*swy;
        pz[i]=-xp*swy+zp*cwy;
    }
    NewDraw();
}
ULONG GetMessage()
{
    struct IntuiMessage *msg;
    ULONG class=0;
    if(msg=(struct IntuiMessage *)GetMsg(win[1]->UserPort)){
        class=msg->Class;
        ReplyMsg(msg);
    }
    return(class);
}
VOID main ()
{
    OpenW();
    NewDraw();
    while(GetMessage()!=CLOSEWINDOW) Turn(.03,.03);
    CloseW();
}

```

Listing 1. Double-Buffering mit dem Blitter. Verschiebt man das Window, wird der Extra-Buffer sichtbar.

```

/* Aztec C 3.4a
cc CPRArrow.c +L
ln CPRArrow.o -lm32 -lc32 */
/* \\ Gölzy's >>COPPER<< Double-Buffering Demo */

#include <exec/memory.h>
#include <exec/types.h>
#include <graphics/gfxbase.h>
#include <graphics/view.h>
#include <graphics/copper.h>
#include <devices/inputevent.h>
#include <devices/gameport.h>
#include <functions.h>

#define WIDTH 320
#define HEIGHT 256

SHORT CListNr=0;
SHORT Kopf[7]={
    4,0,1,2,0,3,4
};

```

Listing 2. Weiche Animation mit Double-Buffering. Dieses Demo arbeitet mit zwei Copperlisten.


```

SHORT Rumpf[9]={
    6,7,8,5,9,10,11,12,9
};
ULONG GPDDev;
FLOAT px[13]={ /* X-Positionen der Eckpunkte des Pfeils */
    -120.0,-60.0,-60.0,-60.0,-60.0,-60.0,
    -60.0,-60.0,-60.0,120.0,120.0,120.0,120.0
};
FLOAT py[13]={ /* Y-Positionen der Eckpunkte des Pfeils */
    0.0,40.0,40.0,-40.0,-40.0,20.0,
    20.0,-20.0,-20.0,20.0,20.0,-20.0,-20.0
};
FLOAT pz[13]={ /* Z-Positionen der Eckpunkte des Pfeils */
    0.0,40.0,-40.0,-40.0,40.0,20.0,
    -20.0,-20.0,20.0,20.0,-20.0,-20.0,20.0
};
struct GamePortTrigger GPTrigger={
    GPTF_UPKEYS|GPTF_DOWNKEYS,0,1, 1
};
struct GfxBase *GfxBase;
struct View *OldView,View;
struct cprlist *CList[2];
struct MsgPort *GPort;
struct IOStdReq *GRequest;
struct MathTransBase *MathTransBase;
struct InputEvent ReadStick;
struct ViewPort VPort;
struct RasInfo RasInfo;
struct BitMap BMap[2];
struct RastPort RP[2],*RPort;

/*
\\ Funktionen
*/

VOID CloseW()
{
    REGISTER SHORT i;

    if(OldView){
        LoadView(OldView);
        WaitTOP();
    }
    FreeVPortCopLists(&VPort);
    for(i=0;i<2;i++) FreeCprList(CList[i]);
    if(VPort.ColorMap) FreeColorMap(VPort.ColorMap);
    for(i=0;i<2;i++){
        if(BMap[i].Planes[0])
            FreeRaster(BMap[i].Planes[0],WIDTH,HEIGHT);
    }
    if(GPDDev) CloseDevice(GRequest);
    if(GRequest) DeleteStdIO(GRequest);
    if(GPort) DeletePort(GPort);
    if(MathTransBase) CloseLibrary(MathTransBase);
    if(GfxBase) CloseLibrary(GfxBase);
    exit(0);
}

VOID OpenW()
{
    REGISTER SHORT i;

    if(!(GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library",0))) CloseW();
    if(!(MathTransBase=(struct MathTransBase *)
        OpenLibrary("mathtrans.library",0))) CloseW();
    if(!(GPort=CreatePort("GPort",0))) CloseW();
    if(!(GRequest=CreateStdIO(GPort))) CloseW();
    if(GPDDev=(ULONG)
        OpenDevice("gameport.device",1,GRequest,0)) CloseW();
    OldView=GfxBase->ActiView;
    InitView(&View);
    InitVPort(&VPort);
    for(i=0;i<2;i++){
        InitRastPort(&RP[i]);
        InitBitMap(&BMap[i],1,WIDTH,HEIGHT);
        if(!(BMap[i].Planes[0]=(PLANEPTR)
            AllocRaster(WIDTH,HEIGHT))) CloseW();
        RP[i].BitMap=&BMap[i];
    }
    View.ViewPort =&VPort;
    VPort.DWidth =WIDTH;
    VPort.DHeight =HEIGHT;
    VPort.RasInfo =&RasInfo;
    VPort.ColorMap =GetColorMap(2);
}

```

```

RasInfo.RxOffset=0;
RasInfo.RyOffset=0;
RasInfo.Next =NULL;
for(i=0;i<2;i++){ /* zwei LOF-Copperlisten berechnen lassen */
    View.LOFCprList=NULL;
    RasInfo.BitMap=&BMap[i];
    MakeVPort(&View,&VPort);
    MrgCop(&View);
    CList[i]=View.LOFCprList;
}
}

VOID SetGRequest(doit,c,l,d)
    BOOL doit;
    UWORD c;
    ULONG l;
    APTR d;
{
    GRequest->io_Command =c;
    GRequest->io_Length =l;
    GRequest->io_Data =d;
    if(doit) DoIO(GRequest);
}

VOID InitStick()
{
    UBYTE Data=GPCT_RELJOYSTICK;
    ULONG lg=sizeof(struct GamePortTrigger),
        li=sizeof(struct InputEvent);

    SetGRequest(1,GPD_SETCTYPE,1,&Data);
    SetGRequest(1,GPD_SETTRIGGER,lg,&GPTrigger);
    SetGRequest(0,GPD_READEVENT,li,&ReadStick);
}

VOID NewDraw()
{
    REGISTER SHORT i,x,y,index;

    RPort=&RP[CListNr]; /* Zeichenrastport festlegen */
    SetRast(RPort,0);
    Move(RPort,(SHORT)px[1]+160,(SHORT)py[1]+128);
    for(i=0;i<7;i++){
        index=Kopf[i];
        x=(SHORT)px[index]+160;
        y=(SHORT)py[index]+128;
        Draw(RPort,x,y);
    }
    Move(RPort,(SHORT)px[2]+160,(SHORT)py[2]+128);
    Draw(RPort,(SHORT)px[3]+160,(SHORT)py[3]+128);
    Move(RPort,(SHORT)px[5]+160,(SHORT)py[5]+128);
    for(i=0;i<9;i++){
        index=Rumpf[i];
        x=(SHORT)px[index]+160;
        y=(SHORT)py[index]+128;
        Draw(RPort,x,y);
    }
    for(i=0;i<3;i++){
        Move(RPort,(SHORT)px[6+i]+160,(SHORT)py[6+i]+128);
        Draw(RPort,(SHORT)px[10+i]+160,(SHORT)py[10+i]+128);
    }
    View.LOFCprList=CList[CListNr]; /* Copperlisten tauschen */
    LoadView(&View);
    CListNr=CListNr?0:1;
}

VOID Turn(wx,wy)
    FLOAT wx,wy;
{
    REGISTER SHORT i;
    FLOAT xp,yp,zp,swx,cwx,swy,cwy;

    swx=sin(wx);
    cwx=cos(wx);
    swy=sin(wy);
    cwy=cos(wy);
    for(i=0;i<13;i++){
        yp=py[i];
        zp=pz[i];
        py[i] = yp*cwx+zp*swx;
        pz[i] = -yp*swx+zp*cwx;
    }
}

```

Listing 2. (Fortsetzung)


```

xp=px[i];
zp=pz[i];
px[i] = xp*cpy+zp*swy;
pz[i] = -xp*swy+zp*cpy;
}
NewDraw();
}

VOID main ()
{
  UBYTE Data=GPCT_NOCONTROLLER;
  FLOAT w1,w2;

  OpenW();
  InitStick();
  NewDraw();
}

```

```

while(1){
  SendIO(GPRequest);
  WaitIO(GPRequest);
  if(ReadStick.ie_Code==IECODE_LBUTTON) break;
  else{
    if(ReadStick.ie_X) w2=ReadStick.ie_X>0?-0.03:0.03;
    else w2=0;
    if(ReadStick.ie_Y) w1=ReadStick.ie_Y>0?-0.03:0.03;
    else w1=0;
    Turn(w1,w2);
  }
}
SetGPRequest(1,GPD_SETCTYPE,1,&Data);
CloseW();
}

```

Listing 2. (Schluß)

Blitt-schnelle Grafik

Schneller kopieren mit dem Blitter

Von Arno Gölzer

Mit dem »BLock Image Transfer«, besser bekannt unter dem Namen »Blitter«, stehen dem Spieleprogrammierer alle Türen offen. Dabei ist es nicht einmal notwendig, sich mit der Hardware auseinanderzusetzen. Das Betriebssystem spricht die Blitterfunktionen effektiv genug an.

Der Chip, mit dem wohlklingenden Namen Agnus, vereint unter anderem die beiden Coprozessoren Copper und Blitter. Die Haupt- und Lieblingsaufgabe des Blitters ist das Kopieren von Daten, wobei er die Kopie gleichzeitig durch die Verknüpfung mit anderen Speicherbereichen manipulieren kann. Darüber hinaus füllt er mit atemberaubender Geschwindigkeit Flächen und zeichnet ebenso schnell Linien auf die Zeichenfläche vor den Augen des staunenden Amiga-Users.

Wegen dieser Eigenschaften und ob seiner rasanten Arbeitsgeschwindigkeit, findet der Blitter immer dort, wo es um schnelle Grafik geht, seinen Einsatz. Der Coprozessor kann direkt über die Hardware-Register (siehe im Artikel über den »Copper« auf Seite 84 - Listing 1) manipuliert und gestartet werden. Wir möchten jedoch hier die speziellen Systemroutinen der Graphics-Library näher beleuchten, die den Blitter veranlassen, Speicherbereiche zu kopieren. Beachten Sie hierbei, daß die Höhe und die Breite des Kopierbereichs 1024 Pixel nicht überschreiten darf.

Zu jedem Spiel gehört einfach eine gute Grafik - und schnell muß sie natürlich auch sein. Der Amiga hat interessante Fähigkeiten mit überraschendem Tempo auf Lager. Der Blitter macht's möglich.

Bevorzugte Aufgabe des Blitters ist es, Daten durch den Speicher zu jagen. Nach dem Test der ersten Funktion, die wir Ihnen vorstellen möchten, werden Sie diese Neigung verstehen und vor allem nutzen: Clip-Blit(&von,x1,y1,&nach,x2,y2,b,h,m) ermöglicht das Kopieren eines rechteckigen Bereiches von RastPort »von« nach RastPort »nach«, in einer wahrhaft atemberaubenden Geschwindigkeit. Die linke obere Ecke des Quellrechtecks befindet sich an der Position x1/y1 im RastPort »von«.

Das Rechteck ist »b« Pixel breit und »h« Pixel hoch. Clip-Blit() kopiert es so, daß sich dessen linke Ecke nach dem Clip an der Position x2/y2, im RastPort »nach« befindet. Natürlich können Sie für beide RastPorts auch die gleiche Adresse angeben.

Für diese Funktion haben wir auch gleich ein Demo-Listing vorbereitet: »BBrush.c« (Listing 1). Nach dem Start des Programms können Sie, in einer

speziell angelegten Zeichenfläche, mit einem Ausschnitt des beim Aufruf gerade aktiven Windows zeichnen. Bei der Betätigung der linken Maustaste wird bei jeder Mausbewegung der Quellbereich unverändert in den Zielbereich kopiert. Mit Druck auf die rechte Taste erscheint jedoch das kopierte Rechteck invertiert.

Wir haben die Möglichkeit der Veränderung des Kopierbereichs durch eine logische Verknüpfung mit dem Zielbereich genutzt. Die Art der Verknüpfung teilen wir dem Blitter über den letzten Parameter der Clip-Blit()-Funktion »m« mit.

Die UBYTE-Variable »m« steht für Minterm. Tabelle 1 zeigt: mit den oberen 4 Bit ist die Verknüpfungsart codiert. Nehmen wir das einfachste Beispiel: der Wert 0x80 als Minterm ergibt als Ergebnis RastPort V (von) AND RastPort N (nach). Das heißt, nur wo in beiden RastPorts Punkte gesetzt sind, ist nach der Verknüpfung auch wirklich ein Punkt zu se-

hen. Ein weiteres Beispiel, diesmal mit einem Minterm-Wert von 0x40. Der Blitter setzt einen Punkt, wenn in V ebenfalls ein Punkt, aber in N kein Punkt gesetzt war.

Die Verknüpfung erfolgt Bit-Plane für Bit-Plane. Die Farbe richtet sich nach dem Inhalt des Farbregisters mit der Nummer des neuen Wertes (siehe Bild 1). Alle in unserer Tabelle 1 aufgeführten Minterms können des weiteren noch untereinander mit OR verknüpft werden. Bei den vier möglichen Bits gibt es $2^4=16$ Verknüpfungsmöglichkeiten. Wieder einige Beispiele:

$(1V \& 1N) \mid (1V \& N)$

Die Werte 0x10 und 0x20 wurden mit ODER zu 0x30 verknüpft. Der obige Ausdruck läßt sich noch vereinfachen, indem man »1V« ausklammert:

$1V \& (1N \mid N) = 1V$

Der Klammerausdruck »1N | N« wird zu 1 und wir erhalten demnach als Ergebnis »1V & 1«, was »1V« entspricht. Der Wert 0x30 invertiert demnach den Bereich in RastPort V und legt ihn in RastPort N ab. So wurde auch die Invertierung in BBrush.c (Listing 1) erreicht. Welche Auswirkung hat der Wert 0xC0? Um diesen Wert zu erreichen, müssen wir 0x80 und 0x40 mit ODER verknüpfen:

$(V \& N) \mid (V \& 1N) = V$

Hier wird der Quellbereich ohne Veränderung übernommen. Nehmen Sie sich doch das Programm Brush vor und testen Sie einmal alle 16 Minterms durch! Wenn Sie die Ein-

Verknüpfung	Minterm
IV & IN	00010000 = 0x10
IV & N	00100000 = 0x20
V & IN	01000000 = 0x40
V & N	00010000 = 0x80

Tabelle 1. Mit Minterms werden Daten verknüpft

gabe von Kommandoparametern erlauben, wird die Aufgabe besonders einfach. Testen Sie dann gleich einmal Minterm 0x60. Er veranlaßt den Blitter, beide RastPorts mit EOR zu verknüpfen, das heißt nur an den Stellen werden Punkte gesetzt, wo entweder in V oder in N ein Punkt gesetzt war. Zweimaliges Blitten mit gleichen Parametern stellt den ursprünglichen Zustand wieder her. In Spielen wird diese Möglichkeit oft für die unterschiedlichsten Effekte genutzt, z.B. wird mit dieser Technik im Spiel Reflex (in diesem Heft auf Seite 6) eine »Explosion« dargestellt.

ClipBlit() blittet von RastPort zu RastPort. Wagen wir uns noch eine Ebene tiefer und arbeiten mit BitMaps, denn oft ist es nicht notwendig, einen komplett initialisierten RastPort im Arbeitsspeicher mitzuschleppen. In vielen Fällen, z.B. für den »UNDO«-Buffer in Zeichenprogrammen, reichen Informationen über die Lage des Speicherplatzes für die einzelnen BitPlanes und deren Größe aus. Diese Informationen sind in der BitMap-Struktur – Listing 2 zeigt sie noch einmal – zusammengefaßt. Hier die Initialisierung und die Freigabe einer BitMap-Struktur namens BM:

```
...
struct BitMap BM;

...
InitBitMap(&BM,p,b,h);
for(i=0;i<p;i++){
  BM.Planes[i]=
  (PLANPTR)AllocRaster
  (b,h));
  if(BM.Planes[i])
  BltClear
  (BM.Planes[i],RASSIZE
  (b,h),1);
  else quit();
}

...
quit()
{
  ...
  for(i=0;i<p;i++){
    if(BM.Planes[i]) Free-
    Raster(BM.Planes
    [i],b,h);
  }
  ...
}
```

Mit der InitBitMap()-Funktion legen Sie fest, wie viele Planen (p) von der Größe b*h (Breite*Höhe) Pixel Sie benötigen. Die Funktion AllocRaster reserviert den notwendigen Speicherplatz im Chipmemory. Als Rückgabewert erhalten Sie die Adresse der Plane oder – falls ein Fehler auftrat – »NULL«.

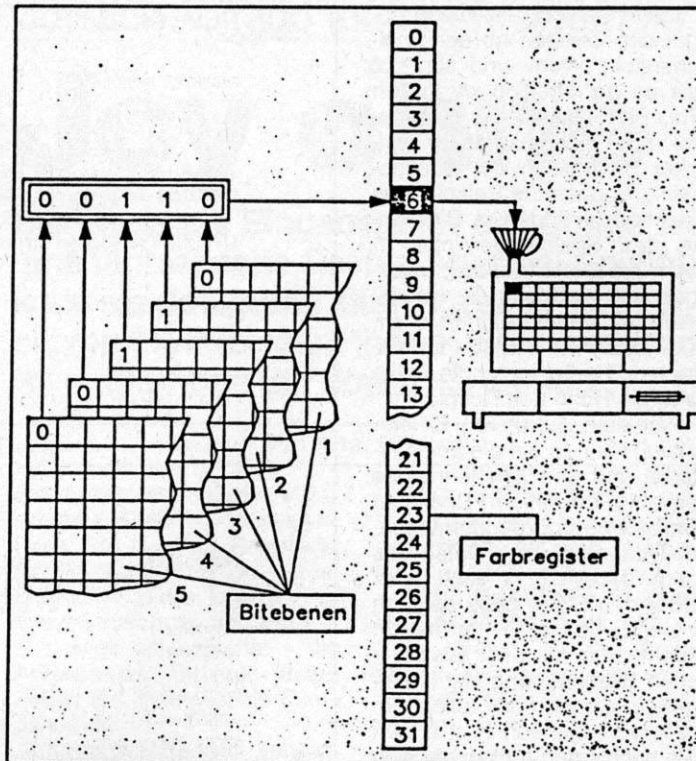


Bild 1. Die Bitebenen legen die Farben eines Punktes auf dem Bildschirm fest

Die Blitter-Funktion BltClear() löscht Plane, auf die mittels des ersten Parameters verwiesen wurde. Die notwendige Größe errechnet das im Headerfile »graphics/gfx.h« definierte Makro »RASSIZE()«. Der dritte Parameter ist ein Flag mit folgender Bedeutung: ist Bit 1 gesetzt, deutet die Funktion die oberen 16 Bits des zweiten Parameters (hier RASSIZE(w,h) als die Anzahl der zu löschenden Zeilen und die unter 16 Bits als die Anzahl der zu löschenden Bytes pro Zeile – ein Rechteck wird gelöscht. Ist jedoch Bit 1 gelöscht, so interpretiert BltClear() den zweiten Parameter als die tatsächliche Anzahl der zu löschenden Bytes. Setzt man Bit 0 auf eins, wartet das Programm, bis der Blitter mit Löschen fertig ist.

In unserem Beispiel gibt demnach der zweite Parameter die Anzahl der Bytes an, die in der BitPlane gelöscht werden sollen. Dabei wartet das Programm so lange, bis der Blitter seine Arbeit verrichtet hat.

Es versteht sich von selbst, daß der disziplinierte Programmierer die reservierten BitPlanes vor Programmende wieder freigibt. Hier geschieht das mit FreeRaster(). Parameter sind neben der Adresse der Plane, deren Breite und Höhe. Das waren die Vorbereitungen, nun zur Anwendung:

```
struct BitMap von,nach;
SHORT x1,y1,x2,y2,b,h;
UBYTE m,M;
PLANPTR t;

...
BltBitMap(&von,x1,y1,&
nach,x2,y2,b,h,m,M,t);
```

Diese riesige Anzahl von aktuellen Parametern sind notwendig, um ein Rechteck von einer BitMap in die andere zu blitten. Aber keine Angst, die Funktion ist der vorher besprochenen sehr ähnlich. Die Parameter »von« und »nach« sind diesmal die Adressen der beiden BitMaps. Auch hier können beide gleich sein, nur müssen wir in diesem Fall Platz für einen Zwischenspeicher schaffen, falls sich die Bereiche überschneiden. Dieser Zwischenspeicher könnte z.B. ein temporäres Raster sein, welches man für die Area- oder die Füll-Funktionen angelegt hat. Er muß nur groß genug sein, um den überlappenden Bereich aufnehmen zu können.

Der letzte Parameter, im Beispiel »t«, verweist auf den Zwischenspeicher. Das große M im Aufruf steht für Maske. Hiermit geben Sie an, welche BitPlane(s) die Funktion blitten soll. Für jede Plane müssen Sie ein Bit setzen, wobei die Nummer der Plane in der BitMap-Struktur gleich dem Bit in Maske ist.

Setzen Sie also Bit 0 und 1, wenn Sie möchten, daß die Planen 0 und 1 geblitten werden sollen. Ansonsten entsprechen die Parameter denen der bereits besprochenen ClipBlit()-Funktion.

Eine Funktion, die von einem RastPort in eine BitMap kopiert, existiert nicht. Vielleicht fällt Ihnen eine ein? Denken Sie dabei an die einzelnen Komponenten der RastPort-Struktur – findet man da nicht die Adresse der BitMap-Struktur?

Das Kopieren aus einer BitMap in einen RastPort ist dagegen bereits möglich: BltBitMapRastPort(&von,x1,y1,&nach,x2,y2,b,h,m). Die Adresse »von« ist die Quell-BitMap und »nach« die des Ziel-RastPorts. Alle anderen Parameter können Sie von den vorangegangenen Funktionen übernehmen. Die folgende Funktion kopiert ebenfalls ein Rechteck aus einer Quell-BitMap in einen Ziel-RastPort. Dabei kann der Bereich nicht nur durch die Minterms manipuliert werden, sondern noch durch eine zusätzliche Bitplane, die als Maske oder »Sieb« fungiert:

```
BltMaskBitMapRastPort(&
von,x1,y1,&nach,x2,y2,b,
h,m,&p);
```

Sie sehen, der Aufruf unterscheidet sich von dem der BltBitMapRastPort()-Funktion nur durch den zusätzlichen Parameter »p«. Er steht für die Adresse der Sieb-Plane. An den Stellen, wo die Punkte des Siebes gesetzt sind, ist er für den Quellbereich durchlässig. An gelöschten Stellen ist dagegen kein Durchkommen.

Ganz ähnlich funktioniert BltPattern(&rp,&p,xa,ya,xey,n). Die Funktion zeichnet ein Rechteck »durch eine Maske« auf RastPort »rp«. Gesetzte Farben, der Zeichenmodus und Pattern werden dabei aus der RastPort-Struktur übernommen.

Die Maske ist wieder, wie bei BltMaskBitMapRastPort() beschrieben, eine einzelne BitPlane. Wir nennen Sie im Beispiel »p«. Diese Plane ist für das Rechteck an den gelöschten Plane-Punkten durchlässig.

Die linke obere Ecke des Rechtecks liegt an der Position xa/ya, die rechte untere Ecke bei xey/ye. Parameter »n« gibt den BytesPerRow-Wert der Masken-Plane an. Sie kennen diese Bezeichnung aus der BitMap-Struktur. Ist Ihre Sieb-Plane z.B. 640 Pixel breit, so errechnet sich der entsprechende

BytesPerRow-Wert aus $640/8 = 80$ Byte.

Ersetzen Sie, zur Verdeutlichung, die ClipBlit()-Funktion aus unserem Demo-Listing 1 durch die folgende Zeile:

```
BltPattern(RPW,RPA-
>BitMap->Planes[0],
XPos,y,XPos+64,y+32,80);
```

Wir benutzen die erste Plane des Workbench-Screens als Sieb. Da sie im Hires-Modus eine Breite von 640 Pixeln mißt, geben wir als BytesPerRow-Wert 80 an. Das Rechteck soll 64 Punkte breit und 32 Punkte hoch sein. Beim Zeichnen scheint das Rechteck nur an den Stellen durch, wo in Plane 0 Punkte gesetzt sind. Sie können das Rechteck durch das Setzen von Zeichenfarbe, -modus und -muster beliebig verändern.

Die letzte Funktion unseres Blitter-Exkurses kopiert ebenfalls Daten. Es handelt sich aber diesmal um gepackte Daten, wie etwa die Zeichen aus den Amiga-Fonts. Die Zeichen werden im Speicher so dicht aneinandergereiht, daß keine Bitspalte Platz zwischen ihnen findet.

```
APTR von;
struct RastPort *nach;
SHORT pos,mod,x,y,b,h;
...
BltTemplate(von,pos,mod,
nach,x,y,b,h);
...
```

BltTemplate() liest diese gepackten Zeichen ab der Speicherstelle »von« und schreibt sie an der Position x/y in den RastPort »nach«. Der Parameter »pos« gibt dabei an, an welcher Bit-Position der Quellbereich, in unserem Beispiel das Zeichen, zu finden ist. Die Breite des Speicherbereichs, in dem die Daten gepackt untergebracht sind, gibt der »mod«-Wert (für »Modulo«) in Byte an. Die restlichen beiden Parameter b und h stehen für die Breite und die Höhe des Rechtecks.

Vor dem Aufruf von BltTemplate() können Sie die Zeichenfarbe und den Zeichenmodus beliebig einstellen. Beachten Sie bitte beim Experimentieren, daß der Blitter nur Daten aus dem Chip-Memory lesen kann. Über den Blitter gäbe es noch vieles zu berichten – vermutlich könnte man mit einer ausführlichen Beschreibung fast ein eigenes Sonderheft füllen. In dem vorliegenden Heft finden Sie noch einen Artikel, der sich mit dieser Materie befaßt, es geht dort um die für die Spieleprogrammierung wichtigen BOBs – die »Blitter Objects«.

SO

Arno Götzler ist ein Mann der ersten Stunde. Seit über zwei Jahren arbeitet er als Autor für AMIGA-Magazin und AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Herrn Arno Götzler, Hans-Pinsel-Straße 2, 8013 Haar bei München).

```
VOID OpenW()
{
    REGISTER SHORT i;

    if(!(IntuitionBase=(struct IntuitionBase *)
        OpenLibrary("intuition.library",0))) CloseW();
    awin=(struct Window *)IntuitionBase->ActiveWindow;
    if(!(GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library",0))) CloseW();
    if(!(win=OpenWindow(&nwin))) CloseW();
}

ULONG GetMessage(code)
    USHORT *code;
{
    struct IntuiMessage *msg;
    ULONG class=0;

    if(msg=(struct IntuiMessage *)GetMsg(win->UserPort)){
        class=msg->Class;
        *code=msg->Code;
        ReplyMsg(msg);
    }
    return(class);
}

VOID DrawBrush(mt)
    UBYTE mt;
{
    ULONG class;
    USHORT code;
    SHORT y,keyup=0;

    while(!keyup){
        WaitPort(win->UserPort);
        while(class=GetMessage(&code)){
            if((code==SELECTUP)|| (code==MENUUP)) keyup=1;
            if(class==MOUSEMOVE){
                y=YPos;
                if(y>10) ClipBlit(RPA,0,0,RPW,XPos,
                    y,80,40,mt);
            }
        }
    }
}

VOID main()
{
    ULONG class=0;
    USHORT code,quit=0;

    OpenW();
    while(!quit){
        WaitPort(win->UserPort);
        while(class=GetMessage(&code)){
            if((class==MOUSEBUTTONS)&&KEYDOWN)
                DrawBrush((code==SELECTDOWN)?0xC0:0x-
30);
            else if(class==CLOSEWINDOW) quit=1;
        }
    }
    CloseW();
}
```

Listing 1. Blitter macht's möglich: Zeichnen mit dem Teil eines RastPorts als Brush!

```
/*
cc BBrush.c +L
ln BBrush.o -lc32
*/
/*
\\ Gözly's Blitter-Brush.c
*/
#include <intuition/intuitionbase.h>
#include <exec/memory.h>
#include <exec/types.h>
#include <functions.h>

#define KEYDOWN ((code==SELECTDOWN)|| (code==MENUDOWN))
#define RPA (awin->RPort)
#define RPW (win->RPort)
#define XPos (win->MouseX)
#define YPos (win->MouseY)

struct NewWindow nwin={
    0,0,640,216,-1,-1,CLOSEWINDOW|MOUSEMOVE|MOUSEBUTTONS,
    REPORTMOUSE|RMETRAP|WINDOWCLOSE|ACTIVATE,
    NULL,NULL,(STRPTR)"Gözly's Blitter-Brush",
    NULL,NULL,0,0,0,0,WBENCHSCREEN
};

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Window *win,*awin;

VOID CloseW()
{
    if(win) CloseWindow(win);
    if(GfxBase) CloseLibrary(GfxBase);
    if(IntuitionBase) CloseLibrary(IntuitionBase);
}
```

```
struct BitMap
{
    UWORD BytesPerRow; /* Bytes pro BitMap-Zeile (Breite/8) */
    UWORD Rows; /* Anzahl der Zeilen (Höhe) */
    UBYTE Flags; /* ??? -- vermutlich ungenutzt */
    UBYTE Depth; /* Anzahl der BitPlanes (BitEbenen) */
    UWORD pad; /* Platzhalter -- PLANEPTR beginnen auf LONGWORD */
    PLANEPTR Planes[8]; /* Adressen der BitEbenen im Speicher */
};
```

Listing 2. Die BitMap-Struktur verwaltet elementare Daten über den Bildschirminhalt

Joystick-Abfrage für Ihre Programme

Kein Joy ohne Stick

Wollen Sie auf dem Bildschirm zeichnen, ist die Maus ein ideales Eingabegerät. Wenn es aber darum geht, ein Auto geschickt über einen Parcours zu steuern oder ein Männchen durch ein Labyrinth zu führen, ist der Joystick einfach nicht zu ersetzen. Schließlich ist er dazu bestimmt, seine Mikroschalter in Action-Games zu verschleiben.

Sie entscheiden sich also, in Ihrem neuesten Amiga-Game dem Joystick den Vorzug zu geben. Aber sofort stellt sich ein Problem: wie fragt man diesen Knüppel ab?

Eine Kontrollmöglichkeit ist das ständige Auslesen der Hardware-Register »joy0dat« (0xDFF00A) für den linken Port und »joy1dat« (0xDFF00C) für den rechten Port. Diese Technik eignet sich jedoch für Programme in einer Multitasking-Umgebung nicht sehr gut, da sie das System nur unnötig belastet.

Wir gehen besser den sicheren Weg über das Betriebssystem und kontrollieren den Joystick über das GamePort-Device. Der erste Schritt dazu ist das Öffnen des Devices mit der Exec-Funktion

```
OpenDevice(name,unr,req, flags)
```

Die Parameter der Funktion sind:

name: der Gerätenamen als nullterminierte Zeichenkette, in unserem Fall »gameport.device«

unr: die Gerätenummer (0 für den linken und 1 für den rechten Game-Port)

ireq: die Adresse einer »IOStdReq«-Struktur

flags: bei der Verwendung des GamePort-Device werden keine Flags angegeben.

Die Funktion liefert den »ULONG«-Wert 0, wenn das Gerät ordnungsgemäß geöffnet werden konnte. Allerdings muß die »IOStdReq«-Struktur bereits mit einigen Werten versorgt sein. Diese Aufgabe übernehmen die beiden Support-Routinen »CreatePort(pname, pri)« und »CreateStdIO(port)«.

»CreatePort« verlangt als Parameter die Zeichenkette »pname« und die Priorität »pri«. Diese Funktion liefert bei fehlerfrei-

Die Maus als Eingabegerät am Amiga ist eine fantastische Sache. Aber bei Spielen kann sie den guten alten Joystick nicht ersetzen. Wir zeigen Ihnen eine einfache Methode der Joystick-Abfrage.

Von Arno Gölzer

em Lauf als Ergebnis die Adresse des generierten »Message«-Ports. Sie dient als Parameter für die zweite Funktion: »CreateStdIO()«. Diese legt die Request-Struktur an und liefert – falls alles geklappt hat – die Adresse der Struktur. Treten Fehler auf, liefern beide Funktionen 0.

Für unser Beispiel verwenden wir Game-Port 2, da dieser frei ist. Darüber hinaus ist Port 1 im Normalfall schon vom Input-Device zur Abfrage der Maus belegt. Somit gestaltet sich der Aufruf wie folgt:

```
...
TEXT Name[]="gameport.
device";
ULONG Error;
struct MsgPort*MPort;
struct IOStdReq*IOReq;
...
MPort=CreatePort
("Name",0);
if(MPort==NULL) ... /
* Error */
IOReq=CreateStdIO
(MPort);
if(IOReq==NULL) ... /
* Error */
Error=OpenDevice
(Name,1,IOReq,0);
if(Error) ... /* Error */
...
```

Liefert »OpenDevice« keinen Fehler, so ist der rechte Game-Port bereit, unsere Wünsche, im Form vordefinierter Kommandos, zu empfangen. Diese sind im Headerfile »devices/gameport.h« vereinbart. Wir unterscheiden fünf verschiedene Kommandos:

GPD_ASKCTYPE: CType (Typ des aktuellen Controllers; siehe unten) erfragen

GPD_SETCTYPE: CType festlegen

GPD_ASKTRIGGER: erfragen der Bedingungen in Form der »GamePortTrigger«-Struktur (Listing 1)

GPD_SETTRIGGER: die Mitteilung der Testbedingungen mit Hilfe der eben genannten Struktur

GPD_READEVENT: liest die GamePort-Meldungen.

Zu Beginn unserer Arbeit sollten Sie testen, ob Ihnen »GamePort-Device« tatsächlich zur Verfügung steht:

```
UBYTE d;
...
IOReq->io_Command=
GPD_ASKTRIGGER
IOReq->io_Length=1;
IOReq->io_Data=&d;
if(DoIO(IOReq)!=0) ... /
* Error */
if(d!=GPCT_NOCON-
TROLLER) ... /* Error */
```

Nur wenn die über das Data-Byte empfangene Meldung besagt, daß bislang kein »Controller« angemeldet wurde, kann das Device die Überwachungsarbeit des Joysticks oder der Maus übernehmen. Wir unterscheiden folgende Controller-Typen:

GPCT_ALLOCATED: Game-Port ist bereits von einem anderen Task belegt

GPCT_NOCONTROLLER: GamePort ist noch frei

GPCT_MOUSE: die Maus als Controller

GPCT_RELJOYSTICK: Joystick als Controller, GamePort-Device meldet die Joystick-Richtung ständig

GPCT_ABSJOYSTICK: Joystick als Controller, GamePort-Device meldet die Joystick-Richtung nur bei einer Richtungsänderung

Der nächste Schritt ist die Festlegung des eigenen Controllertyps mit dem Kommando »GPD_SETCTYPE«. Dazu kann wieder eine Sequenz, ähnlich der oben angeführten, dienen. Diesmal ist jedoch der »io_Command Type« gleich »GPD_SETCTYPE« und die

Data-Komponente erwartet die Adresse eines Byte mit dem Controllertyp als Wert. Dazu ein Beispiel:

```
UBYTE d=GPCT_ABS-
JOYSTICK;
```

Möchten Sie dem System Ihre eigenen Testbedingungen mitteilen, so wiederholen Sie die Sequenz ein weiteres Mal. Das Kommando lautet dann »GPD_SETTRIGGER«. Das Data-Feld verlangt nun die Adresse der »GamePortTrigger«-Struktur. Die Datenlänge in Bytes errechnen Sie am besten mit dem Operator

```
sizeof(struct
GamePort-Trigger)
```

Die »GamePortTrigger«-Struktur (Listing 1) muß bis dahin bereits beschrieben sein. Hier die einzelnen Komponenten dieser kurzen Struktur:

gpt_Keys: gibt an, ob das Betätigen oder das Loslassen des Feuerknopfes (oder der Maustasten beim Controller »GPCT_MOUSE«) gemeldet werden soll. Hierfür stehen zwei Flags zur Verfügung: »GPTF_UPKEYS« veranlaßt eine Meldung bei Betätigung und »GPTF_DOWNKEYS« läßt das Loslassen des Buttons signalisieren. Natürlich können Sie beide Flags mit »ODER« verknüpfen, wenn Sie alle zwei Meldungen empfangen möchten.

gpt_Timeout: die Zeit zwischen zwei »ReadEvents« in Tics (1/50 Sekunden).

gpt_XDelta und **gpt_YDelta:** »Weg« zwischen zwei »ReadEvents« in Impulsen, die durch die Lochscheibe in der Maus erzeugt werden (in X/Y-Richtung).

Zum Schluß bereiten wir die »IOStdReq«-Struktur noch mittels des Kommandos »GPD_READEVENT« vor. Als Datum übergeben wir die Adresse der InputEvent-Struktur. Nach jedem Start des Kommandos mit »DoIO()« oder auch »SendIO()« (gefolgt von »WaitIO()« oder »WaitPort()«), kann die »InputEvent«-Struktur ausgelesen und so die Joystick-Stellung ermittelt werden.

Allerdings liefert die Struktur keine absoluten Werte, sondern lediglich die Richtung des

Sticks. Beinhaltet die »Input-Event«-Strukturkomponente »ie.X« einen negativen Wert, so befindet sich der Joystick an seinem linken Anschlag. Ist der Wert positiv, wird er gerade nach rechts gedrückt. Das Element »ie.Y« zeigt die vertikale Position an. Ein negativer Wert bedeutet, daß der Joystick in Richtung Monitor bewegt wurde.

Die Betätigung des Buttons ermitteln wir über die Komponente »ie.Code«. Es existieren folgende Flags (s. auch Headerfile »devices/inputevent.h«):

IECODE_LBUTTON: linker Mausknopf oder Feuerknopf des Joysticks

IECODE_RBUTTON: rechter Mausknopf

IECODE_MBUTTON: (noch) nicht existierender mittlerer Mausknopf

IECODE_NOBUTTON: weder Maustaste noch Feuerknopf wurden betätigt

IECODE_UP_PREFIX: Dieses Flag ist mit einem der drei zuerst beschriebenen mit »ODER« verknüpft und zeigt an, daß der Knopf losgelassen wurde.

Vor Programmende muß der Game-Port auf »GPCT_NO-CONTROLLER« eingestellt und der reservierte Speicher freigegeben werden. Die Freigabe erfolgt mit den Funktionen:

```
CloseDevice(IOReq);
DeleteStdIO(IOReq);
DeletePort(MPort);
```

Unser kleines Beispiellisting (Listing 2) soll Sie etwas zum Experimentieren anregen. Wir testen darin auf »ABSJOY-STICK«, geben Sie aber ruhig einmal andere Controller an. Die Funktion »OpenW()« öffnet uns neben den Libraries und dem Window auch das Game-Port-Device. Die Initialisierung des Joysticks erfolgt durch die

Funktionen »InitStick()« und »SetGPRequest()«. Diese übernehmen Prüfung und Festlegung des Controller-Typs und der Testbedingungen und treffen außerdem die Vorbereitungen zum Starten der »Read-Events«.

In der Hauptfunktion »main()« erfolgt, mittels einer Endlosschleife, der eigentliche Start mit »SendIO()« und »WaitIO()«. Entsprechend den Bewegungen des Joysticks – wir testen die ankommenden »InputEvents« ebenfalls in der while-Endlosschleife – zeichnet »DrawC()« eine Kreisfläche in unser Demo-Window. Nach der Betätigung des Feuerknopfs – wir erwarten gemäß der »GamePort-Trigger«-Struktur die Betätigung und das Loslassen des Knopfes – erfolgt das Programmende.

Dies erfolgt selbstverständlich nicht, ohne das Device auf »GPCT_NOCONTROLLER« zu schalten und den in »OpenW()« allokierten Speicher mittels deren Gegenstück »CloseW()« wieder freizugeben.

Wir hoffen, Ihnen mit diesem Beitrag den Einstieg in die Spieleprogrammierung mit Joystick erleichtert zu haben. Sie werden in den weiteren Artikeln der Rubrik »Know-how« immer wieder auf Teile des beschriebenen Beispiellprogramms stoßen. Allerdings haben wir dort auf den Test mit »ASKCTYPE« verzichtet, da wir davon ausgehen, daß Sie sich beim Austesten der Demos nicht gleichzeitig mit Ihrem Flugsimulator im siebenten Himmel befinden.

Martin Jobst/so

Arno Götzler ist ein Mann der ersten Stunde und schreibt schon einige Jahre für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Arno Götzler, Hans-Pinsel-Straße 2, 8013 Haar bei München).

```
struct GamePortTrigger {
    UWORD gpt_Keys;
    UWORD gpt_Timeout;
    UWORD gpt_XDelta;
    UWORD gpt_YDelta;
};
```

Listing 1. Die »GamePortTrigger«-Struktur legt fest, in welcher Weise ein Joystick oder die Maus getestet werden sollen

```
/*
cc Stick.c +L
ln Stick.o -lc32
*/
```

```
\\ Gözly's JoyStick-Demo
*/
#include <intuition/intuitionbase.h>
#include <graphics/gfxbase.h>
#include <exec/memory.h>
#include <exec/types.h>
#include <devices/inputevent.h>
#include <devices/gameport.h>
#include <functions.h>

#define TMPDIM 100

TEXT DName[] = "Gözly's JoyStick-Demo";
WORD Buffer[10];
ULONG GPDev;
struct GamePortTrigger GPTrigger = {
    GPTF_UPKEYS|GPTF_DOWNKEYS, 0, 1, 1
};
struct NewWindow hwin = {
    0, 0, 400, 210, 0, 1, 0, ACTIVATE, NULL, NULL,
    DName, NULL, NULL, 0, 0, 0, 0, WBENCHSCREEN
};
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Window *win;
struct RastPort *RP;
struct MsgPort *GPort;
struct IOStdReq *GPRequest;
struct InputEvent ReadStick;
struct TmpRas Raster;
struct AreaInfo AInfo;
PLANEPTR Bitplane;

/*
\\ Funktionen
*/

VOID CloseW()
{
    if(Bitplane) FreeRaster(Bitplane, TMPDIM, TMPDIM);
    if(GPDev) CloseDevice(GPRequest);
    if(GPRequest) DeleteStdIO(GPRequest);
    if(GPort) DeletePort(GPort);
    if(win) CloseWindow(win);
    if(GfxBase) CloseLibrary(GfxBase);
    if(IntuitionBase) CloseLibrary(IntuitionBase);
    exit(0);
}

VOID OpenW()
{
    if(!IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", 0)) CloseW();
    if(!GfxBase = (struct GfxBase *)
        OpenLibrary("graphics.library", 0)) CloseW();
    if(!win = OpenWindow(&hwin)) CloseW();
    RP = win->RPort;
    if(!GPort = CreatePort("GPort", 0)) CloseW();
    if(!GPRequest = CreateStdIO(GPort)) CloseW();
    if(GPDev = (ULONG)
        OpenDevice("gameport.device", 1, GPRequest, 0)) CloseW();
    if(!Bitplane = AllocRaster(TMPDIM, TMPDIM)) CloseW();
    InitTmpRas(&Raster, Bitplane, RASSIZE(TMPDIM, TMPDIM));
    InitArea(&AInfo, Buffer, 2);
    RP->TmpRas = &Raster;
    RP->AreaInfo = &AInfo;
}

LONG SetGPRequest(doit, command, length, data)
    BOOL doit;
    UWORD command;
    ULONG length;
    APTR data;
{
    GPRequest->io_Command = command;
    GPRequest->io_Length = length;
    GPRequest->io_Data = data;
    if(doit) return(DoIO(GPRequest));
    else return(0);
}
```



```

VOID InitStick()
{
    UBYTE Check,Data=GPCT_ABSJOYSTICK;
    ULONG lg=sizeof(struct GamePortTrigger),
           li=sizeof(struct InputEvent);

    if(SetGPRequest(1,GPD_ASKCTYPE,1,&Check)) CloseW();
    if(Check!=GPCT_NOCONTROLLER) CloseW();
    if(SetGPRequest(1,GPD_SETCTYPE,1,&Data)) CloseW();
    if(SetGPRequest(1,GPD_SETTRIGGER,lg,&GPTrigger))CloseW();
    SetGPRequest(0,GPD_READEVENT,li,&ReadStick);
}

VOID DrawC(x,y,r)
    SHORT x,y,r;
{
    AreaEllipse(RP,x,y,2*r,r);
    AreaEnd(RP);
}

VOID DrawPic()
{
    REGISTER SHORT i,j,l;

    SetDrMd(RP,JAM1);
    for(i=0;i<2;i++){
        SetAPen(RP,2-i);
        RectFill(RP,10+(4*i),15+(2*i),390-(4*i),205-(2*i));
    }
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            for(l=0;l<2;l++){
                SetAPen(RP,2-l*2);
                DrawC(100+i*100,60+j*50,25-2*i);
            }
        }
    }
}

```

```

SetDrMd(RP,JAM1|COMPLEMENT);
DrawC(200,110,20);
}

VOID main ()
{
    UBYTE Data=GPCT_NOCONTROLLER;
    SHORT x=200,y=110;

    OpenW();
    DrawPic();
    InitStick();
    while(1){
        SendIO(GPRequest);
        WaitIO(GPRequest);
        DrawC(x,y,20);
        if(ReadStick.ie_Code==IECODE_LBUTTON) break;
        if(ReadStick.ie_X){
            if(ReadStick.ie_X<0) x=100;
            else x=300;
        }
        else x=200;
        if(ReadStick.ie_Y){
            if(ReadStick.ie_Y<0) y= 60;
            else y=160;
        }
        else y=110;
        DrawC(x,y,20);
    }
    SetGPRequest(1,GPD_SETCTYPE,1,&Data);
    CloseW();
}

```

Listing 2. Unser Joystick-Demoprogramm überwacht über »GamePort«-Device einen »ABS«-Joystick

Animationen mit Superbitmaps

Grafik am laufenden Band

Eine Möglichkeit, Bewegungen auf den Bildschirm zu bringen, ist die Definition übergroßer BitMaps, den sogenannten SuperBitMaps (nicht zu verwechseln mit SuperBigMacs). Es handelt sich um BitPlanes, die größer sind als das Display in dem sie dargestellt werden. Intuition stellt uns mit dem SuperBitMap-Window ein vorgefertigtes Element zur Verfügung, welches die Arbeit mit großen BitMaps erlaubt. Da Intuition bei Spielen meist nicht benötigt wird, befassen wir uns in diesem Artikel mit einfachen Grafikdisplays, um zu sehen, welche Möglichkeiten uns hier offenstehen.

Wir initialisieren die View-, ViewPort- und BitMap-Struktur, lassen die Copperlisten berechnen und starten das Display. Die genaue Vorgehensweise für die Definition eines eigenen Views finden Sie in diesem Heft (Seite 81) im Artikel »Intuition ausgebremst«. Die

Ist es nicht faszinierend, wenn in einem Spiel aufwendige Grafiken wie ein endloses Band in einer atemberaubenden Geschwindigkeit über den Bildschirm scrollen? Fragen Sie sich auch jedesmal, wie das möglich ist? Dieser Artikel gibt die Antwort auf Ihre Fragen!

BitPlanes sollen diesmal jedoch größer sein, als der ViewPort. Die Ausmaße einer BitMap ist nur vom zur Verfügung stehenden Chip-Memory abhängig. Leider ist dieser Speicherbereich mit seinen 512 KByte nicht allzu groß bemessen, so daß die Vorausberechnung des benötigten Speichers sicherlich sinnvoll ist. Der Speicherbedarf einer BitMap oder allgemein eines Bildschirms (Screens) wächst proportional mit der Anzahl der gewünschten Punkte (Auflösung) und proportional mit der Anzahl der Bit-

ebenen (Farben). Berechnen wir als Beispiel einen Bildschirm im Hires-Interlace-Modus. Ein solcher Bildschirm arbeitet mit einer Auflösung von 640 x 512 Pixel. Angenommen wir benötigen die maximale Anzahl von Farben in diesem Modus und fordern entsprechend vier Planes an. Dann errechnet sich der benötigte Speicherplatz wie folgt:

Memory=(640*512)Punkte*
4Bitebenen/8=163840Byte

In Tabelle 1 finden Sie den Bedarf an Chipmemory, der mit

fünf Planes möglichen Displaymodis. Die BitMaps können jedoch noch wesentlich größer sein als die Displaygrößen in der Tabelle. Das Beispielprogramm »SuperBitMap.c« (Listing 1) fordert zwei Planes mit einer Größe von 200 x 2000 Pixel, was nach obenstehender Rechnung etwa 100 KByte des (Chip-)Arbeitsspeichers »kostet«!

Sind alle Vorbereitungen getroffen und das neue Display geladen, ist nur ein Ausschnitt der gesamten BitMap im ViewPort sichtbar (Bild 1). Das Scrolling erreichen Sie durch kontinuierliches Verschieben der Riesenplanes »unter dem ViewPort-Ausschnitt«. Die Komponenten RxOffset und RyOffset der RasInfo-Struktur beschreiben die X/Y-Koordinate des Punktes im Raster, welcher an der linken oberen Ecke des ViewPorts dargestellt werden soll. Ändern wir diese Variablen, so ändert sich, nach der Neuberechnung der Copperli-

Bit Ebenen	Farben	Farbreg.	320 x 256	320x 512	640 x 256	640 x 512	Punkte
1	2	0 und 1	10240	20480	20480	40960	Byte
2	4	0 bis 3	20480	40960	40960	81920	Byte
3	8	0 bis 7	30720	61440	61440	122880	Byte
4	16	0 bis 15	40960	81920	81920	163840	Byte
5	32	0 bis 31	51200	102400	—	—	—

Tabelle 1. SuperBitMaps sind sehr speicherintensiv

sten, die Position der BitMap im ViewPort-Fenster. Inkrementieren wir beispielsweise den X-Offset, so rollt das Raster nach links aus dem ViewPort, da die an der linken oberen Ecke des ViewPorts dargestellten Positionen weiter nach rechts im Raster wandern. Stellen Sie sich einfach ein Blatt Papier vor, das Sie unter einer Lupe verschieben. Skalieren Sie das Blatt am oberen Rand von links nach rechts aufsteigend. Bewegen Sie das Blatt unter der Lupe, daß sich an der linken oberen Ecke der Lupe immer höhere Werte der Skala befinden. Sie müssen dazu das Blatt nach links (oder die Lupe nach rechts) bewegen. Möchten Sie, daß das Raster nach oben wandert, müssen Sie, da sich der Ursprung des Bildschirmkoordinatensystems in der linken oberen Monitorecke befindet, die Y-Position in RasInfo-Struktur ebenfalls erhöhen. Die BitMap-Struktur enthält die wahren Ausmaße des Rasters, während in den ViewPort-Struktur-Variablen DWidth und DHeight die Maße des dargestellten Ausschnitts gespeichert sind.

Die Positionen, an denen ein Raster gerade noch dargestellt werden kann, sozusagen die Extrem-Werte der RasInfo-Variablen, errechnen sich wie folgt:

X-Position	min = 0
X-Position	max = Breite-DWidth
Y-Position	min = 0
Y-Position	max = Höhe-DHeight

Diese Werte müssen im Programm abgefragt werden. Läuft das Raster nach einer Richtung aus dem Display, erscheint dort der Inhalt des Arbeitsspeichers als Grafik, was besonders beim Verschieben in Y-Richtung eher merkwürdig aussieht. Was noch fehlt ist die Berechnung der CopperListen zur Darstellung der veränderten Position des Rasters im ViewPort. Hierzu dient die Funktion

```
ScrollVPort(&ViewPort);
```

Als Parameter verlangt sie die Adresse des ViewPorts.

Nach dem Einfügen der Funktion präsentiert sich das Raster an der neuen Position. Sowohl die Scroll-Richtung als auch die Scroll-Geschwindigkeit richten sich nach dem Inkrement der RastInfo-Variablen RxOffset und RyOffset. Sie legen damit fest, wie viele Pixel das Raster in der jeweiligen Richtung verschoben werden soll. Erhöhen Sie beispielsweise in einer Schleife die Offsets X und Y bei jedem Durchgang um 10, so bewegt sich das Raster bei jedem Aufruf der ScrollVPort()-Funktion um 10 Pixel nach links und

de beliebige Richtung gelenkt werden. Bewegungen des Steuerknüppels in Y-Richtung beschleunigen bzw. bremsen das Fahrzeug. Sollte man trotz aller Vorsicht dennoch mit einem Hindernis kollidieren, läßt sich das Fahrzeug am leichtesten in X-Richtung aus dem »Gestrüpp« abschleppen. Es fällt gar nicht auf, daß sich das Fahrzeug nicht bewegt, sondern die Landschaft unter dem Auto hinweg gezogen wird.

In der gegenwärtigen Version, des Betriebssystems kann der Workbenchscreen mit der Ta-

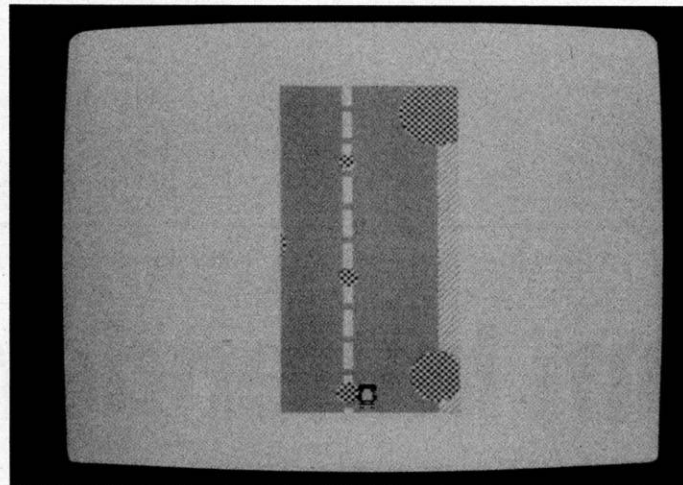


Bild 1. Nur ein Teil der BitMap wird im ViewPort sichtbar

oben. Wenn Sie in jedem Schleifendurchgang den Wert 1 addieren, erscheint die Bewegung wesentlich langsamer.

Diese Möglichkeit der Steuerung von Geschwindigkeit und Richtung des Rasters im Display haben wir auch in unserem SuperBitMap-Demo genutzt. Es handelt sich um ein kleines Spiel oder besser formuliert, um den Grundstock eines Spiels. Der Spieler soll einen sportlich rasanten Flitzer geschickt um die Bäume und Bepflanzungen einer der beliebten verkehrsberuhigten Straßen lenken. Sollten Sie mit Bäumen oder dem Bordstein zusammenstoßen, blitzt der Bildschirm, in Vertretung des Gesetzeshüters, mahnend auf.

Der Wagen wird mit dem Joystick in Game-Port 2 gesteuert. Der Feuerknopf beendet das Spiel. Das Fahrzeug kann in je-

stenkombination <Linke Amiga n> nach vorne gebracht werden. Da wir nicht mit einem Screen arbeiten, kann das Display nicht mehr vor die Workbench geklappt werden. Abhilfe schafft ein Aufruf der LoadView()-Funktion, worauf im Demo verzichtet wurde.

Bevor Sie mit dem Demolisting experimentieren, besprechen wir den Quellcode des Programms. Die Funktion OpenW() übernimmt vorbereitende Maßnahmen für den Programmablauf. Zu Beginn besorgt sie die Basisadresse der Grafik-Library, rettet die Adresse der aktuellen View-Struktur und kopiert die Sprite-Daten für den Wagen ins Chip-Memory. Danach öffnet sie das GamePort-Device für den Port 2 (Parameter 2 == 1 für Port 2; == 0 für Port 1). AllocRaster() besorgt uns ein BitPlane für ein

Temporär-Raster, welches wir später bei der Verwendung der Area-Funktionen benötigen. Jetzt werden die Strukturen für das Grafikdisplay initialisiert. Wir fordern dabei eine Color-Map mit 24 Farben an. Die ersten vier Farben (da zwei BitPlanes) brauchen wir zur Darstellung des Spielfeldes, die letzten drei sind die Farben des Auto-Sprites.

Das Ende der OpenW()-Funktion bilden die Aufrufe der Betriebssystem-Funktionen zur Berechnung der Copperlisten. Die Funktion CloseW() schließt alle erhaltenen Ressourcen in umgekehrter Reihenfolge. Auch die Copperlisten müssen freigegeben werden, was vielleicht nicht so einfach einzusehen ist, da der Speicher dafür nicht explizit angefordert wurde.

InitStick() gibt dem GamePort-Device bekannt, daß die Abfrage eines Joysticks verlangt wird. Danach zeichnet DrawVP() das Spielfeld mit Sprites. Beachten Sie, daß sich die Sprite-Position nicht auf den ViewPort, sondern auf den View bezieht. In DrawVP() wird auch das neue Display mit LoadView() dargestellt. Doch zuvor teilen wir dem Hardware-Register »clxcon« (0x98) mit, daß wir bei Kollisionen mit Farben, die in Plane 0 ein Bit setzen, eine Meldung wünschen. Es handelt sich dabei um die Farben 1 und 3, mit denen der Straßenrand und die Bäume gezeichnet sind.

Im Headerfile hardware/custom.h sind die Register der Custom-Chips, die sich ab Adresse 0xDFF000 im Speicher befinden, als Custom-Struktur vereint und namentlich festgehalten. Die Kollisionsmeldung erscheint später im Register »clxdat« (0x0E). Ist das Spielfeld sichtbar, verlangt der Aufruf »CreateProcFunktion()« den Start der Funktion MSpeed() als eigenen Prozeß. Diese Funktion ist im Heft 11 des AMIGA-Magazins ausführlich beschrieben, so daß wir auf eine Erklärung an dieser Stelle verzichten können. Wichtig ist nur, zu wissen, daß durch diesen Aufruf die Funktion MSpeed() quasi parallel zum Hauptprogramm läuft (Aztec C-Anwender bitte an die geta4()-Funktion denken!). MSpeed() hat die Aufgabe, ständig den Bildschirm in Y-Richtung zu verschieben und das Register clxdat zu überwachen.

Als Inkrement der Y-Offsets der RasInfo-Struktur dient uns die Variable »Speed«, welche

sich bei der Betätigung des Joysticks in Y-Richtung ändert. Beachten Sie, daß wir die Bewegung an den Extremwerten für RyOffset (0 und MAXH) nicht stoppen, sondern auf das andere Extrem umschalten. Dadurch entsteht eine geschlossene Schleife - Grafik am laufenden Band! Stellt MSpeed() einen Crash fest, so wird die

Hintergrundfarbe kurz rot gefärbt. In der Hauptfunktion main() überwachen wir in einer while-Schleife den GamePort. Empfangen wir eine Meldung über eine Joystick-Bewegung in X-Richtung, scrollen wir das Raster pixelweise in die gewünschte Richtung, jedoch maximal bis zum Extremwert. Drückt der Spieler den Feuer-

button, so erfüllen wir das Schleifenkriterium der while-Schleife (Quit = 1) und beenden mit CloseW() das Programm.

SuperBitMap.c ist natürlich kein ausgereiftes Spiel, aber es zeigt doch mit seinen knapp 300 Zeilen, was ohne großen Aufwand zu erreichen ist. Vielleicht haben Sie ja auch schon

eine neue Idee, bei der die beschriebene Technik zum Einsatz kommt? Also los - wir wollen Sie nicht aufhalten! *pe*

Arno Götzler ist ein Mann der ersten Stunde und schreibt schon einige Jahre für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Herrn Götzler, Hans-Pinsel-Straße 2, 8013 Haar bei München).

```

/* Aztec C 3.4a
cc SBM.c +L
ln SBM.o -lc32 */
/* \ Gölzy's SuperBitMap.c */

#include <exec/memory.h>
#include <exec/types.h>
#include <graphics/gfxbase.h>
#include <graphics/sprite.h>
#include <graphics/view.h>
#include <devices/inputevent.h>
#include <devices/gameport.h>
#include <hardware/custom.h>
#include <functions.h>

#define SDATA 36
#define TMPDIM 200
#define DWIDTH 100
#define DHEIGHT 200
#define WIDTH 200
#define HEIGHT 2000
#define MAXW (WIDTH-DWIDTH)
#define MAXH (HEIGHT-DHEIGHT)

VOID MSpeed();
BOOL Quit=FALSE;
SHORT SNr=-1,Speed=0,Inc=0;
USHORT *SAuto,SDATA[SDATA]={
    0,0,
    0x07E0,0x0000,0x0000,0x00FF,0x1008,0x1FF8,0x1008,0x1FF8,
    0x13C8,0x1C38,0x03C0,0x0C30,0x07E0,0x0810,0x07E0,0x0810,
    0x17E8,0x1818,0x17E8,0x1818,0x17E8,0x1818,0x0000,0x00FF,
    0x0000,0x0420,0x0000,0x00FF,0x00FF,0x0000,0x0000,0x0C30,
    0,0
};
WORD Buffer[10],ColTab[24]={
    0xFFF,0xAC6,0xDDD,0xFC8,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0xFEC,0xF00,0
};
UWORD Pattern[2][8]={
    0x1111,0x2222,0x4444,0x8888,0x1111,0x2222,0x4444,0x8888,
    0xCCCC,0xCCCC,0x3333,0x3333,0xCCCC,0xCCCC,0x3333,0x3333
};
ULONG GPDev;
struct SimpleSprite Auto={
    NULL,16,0,0,2
};
struct GamePortTrigger GPTrigger={
    GPTF_UPKEYS|GPTF_DOWNKEYS,0,1, 1
};
struct GfxBase *GfxBase;
struct Process *Process;
struct View *OldView,View;
struct MsgPort *GPort;
struct IOStdReq *GPRRequest;
struct InputEvent ReadStick;
struct ViewPort VPort;
struct RasInfo RasInfo;
struct BitMap BMap;
struct RastPort RP;
struct TmpRas Raster;
struct AreaInfo AInfo;
PLANEPTR Bitplane;

/*
\ Funktionen

```

```

*/

struct Process *CreateFunctionProc(name,pri,function,stacksize)
    STRPTR name;
    LONG pri,stacksize;
    VOID (*function)();
{
    struct MsgPort *mport;
    struct Process *proc;
    struct HelpSeg{
        LONG Len,Next;
        WORD OCode;
        VOID (*FAddr)();
    } *HSeg;

    if(!(HSeg=(struct HelpSeg *)
        AllocMem(sizeof(struct HelpSeg),MEMF_PUBLIC))) return(NULL);
    HSeg->Len =sizeof(struct HelpSeg);
    HSeg->Next =0;
    HSeg->OCode =0x4EF9;
    HSeg->FAddr =function;
    mport=(struct MsgPort *)
        CreateProc(name,pri,(LONG)(HSeg->Next))>>2,stacksize);
    proc=(struct Process*)((ULONG)mport-(ULONG)sizeof(struct Task));
    while(!FindTask(name));
    FreeMem(HSeg,sizeof(struct HelpSeg));
    return(proc);
}

VOID CloseW()
{
    REGISTER SHORT i;

    if(SNr>0) FreeSprite(SNr);
    if(OldView){
        LoadView(OldView);
        WaitTOF();
    }
    FreeVPortCopLists(&VPort);
    FreeCprList(View.LOFCprList);
    if(VPort.ColorMap) FreeColorMap(VPort.ColorMap);
    for(i=0;i<2;i++){
        if(BMap.Planes[i])
            FreeRaster(BMap.Planes[i],WIDTH,HEIGHT);
    }
    if(Bitplane) FreeRaster(Bitplane,TMPDIM,TMPDIM);
    if(GPDev) CloseDevice(GPRRequest);
    if(GPRRequest) DeleteStdIO(GPRRequest);
    if(GPort) DeletePort(GPort);
    if(SAuto) FreeMem(SAuto,SDATA*2);
    if(GfxBase) CloseLibrary(GfxBase);
    exit(0);
}

VOID OpenW()
{
    REGISTER SHORT i;

    if(!(GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library",0))) CloseW();
    OldView=GfxBase->ActiView;
    if(!(SAuto=(USHORT *)AllocMem(SDATA*2,MEMF_CHIP))) CloseW();
    CopyMem(SData,SAuto,SDATA*2);
    Auto.posctldata=SAuto;
    if(!(GPort=CreatePort("GPort",0))) CloseW();
    if(!(GPRRequest=CreateStdIO(GPort))) CloseW();
    if(GPDev=(ULONG)
        OpenDevice("gameport.device",1,GPRRequest,0)) CloseW();
}

```



```

if(! (Bitplane=AllocRaster(TMPDIM,TMPDIM))) CloseW();
InitView(&View);
InitVPort(&VPort);
InitRastPort(&RP);
InitTmpRas(&Raster, Bitplane, RASSIZE(TMPDIM, TMPDIM));
InitArea(&AInfo, Buffer, 2);
InitBitMap(&BMap, 2, WIDTH, HEIGHT);
for(i=0; i<2; i++){
    if(! (BMap.Planes[i]=(PLANEPTR)AllocRaster(WIDTH, HEIGHT)))
        CloseW();
    BltClear(BMap.Planes[i], RASSIZE(WIDTH, HEIGHT), 0);
}
View.ViewPort = &VPort;
View.Modes = SPRITES;
VPort.Modes = SPRITES;
VPort.DWidth = DWIDTH;
VPort.DHeight = DHEIGHT;
VPort.DxOffset = 160-DWIDTH/2;
VPort.DyOffset = 128-DHEIGHT/2;
VPort.RasInfo = &RasInfo;
VPort.ColorMap = GetColorMap(24);
RasInfo.BitMap = &BMap;
RasInfo.RxOffset=50;
RasInfo.RyOffset=MAXH;
RasInfo.Next = NULL;
RP.BitMap = &BMap;
RP.TmpRas = &Raster;
RP.AreaInfo = &AInfo;
LoadRGB4(&VPort, ColTab, 24);
MakeVPort(&View, &VPort);
MrgCop(&View);
}

VOID DrawC(x,y,r)
SHORT x,y,r;
{
    AreaEllipse(&RP, x,y,r,r);
    AreaEnd(&RP);
}

VOID DrawVP()
{
    REGISTER SHORT i,y,r;

    SetAPen(&RP, 2);
    for(i=0; i<10; i++){
        RectFill(&RP, 50, i*200, 150, (i+1)*200-1);
        SetAPen(&RP, 0);
        for(i=0; i<2000; i+=20){
            RectFill(&RP, 98, i, 102, i+15);
        }
        SetAPen(&RP, 3);
        SetAfPt(&RP, Pattern[0], 3);
        for(i=0; i<10; i++){
            RectFill(&RP, 0, i*200, 49, (i+1)*200-1);
            RectFill(&RP, 151, i*200, 200, (i+1)*200-1);
        }
        SetAPen(&RP, 1);
        SetAfPt(&RP, Pattern[1], 3);
        for(i=0; i<10; i++){
            y=240+i*160;
            r=(10-i)*5;
            DrawC(50,y,r);
            DrawC(150,y+80,r);
            DrawC(100,1100+i*70,i+1);
        }
        SNr=GetSprite(&Auto, 2);
        MoveSprite(&VPort, &Auto, 152, 185);
        custom.clxcon=(UWORD)0x41;
        LoadView(&View);
    }

LONG SetGPRequest(doit,c,l,d)
BOOL doit;
UWORD c;
ULONG l;
APTR d;
{
    LONG error=0;

    GPRequest->io_Command =c;

```

```

GPRequest->io_Length =1;
GPRequest->io_Data =d;
if(doit) error=DoIO(GPRequest);
return(error);
}

VOID InitStick()
{
    UBYTE Data=GPCT_RELJOYSTICK;
    ULONG lg=sizeof(struct GamePortTrigger),
        li=sizeof(struct InputEvent);
    if(SetGPRequest(1, GPD_SETCTYPE, 1, &Data)) CloseW();
    if(SetGPRequest(1, GPD_SETTRIGGER, lg, &GPTrigger)) CloseW();
    SetGPRequest(0, GPD_READEVENT, li, &ReadStick);
}

VOID main ()
{
    UBYTE Data=GPCT_NOCONTROLLER;
    SHORT x=50;

    OpenW();
    InitStick();
    DrawVP();
    if(! (Process=CreateFunctionProc("MSpeed", 0, MSpeed, 4000)))
        CloseW();
    while(!Quit){
        SendIO(GPRequest);
        WaitIO(GPRequest);
        if(ReadStick.ie_Code==IECODE_LBUTTON) Quit=TRUE;
        else{
            if(ReadStick.ie_X){
                x+=ReadStick.ie_X<0?-1:1;
                if(x>MAXW) x=MAXW;
                if(x<0) x=0;
                RasInfo.RxOffset=x;
                WaitTOF();
                ScrollVPort(&VPort);
            }
            if(ReadStick.ie_Y){
                Inc+=ReadStick.ie_Y<0?-1:1;
                Speed=Inc/10;
            }
        }
        while(FindTask("TFunc"));
        SetGPRequest(1, GPD_SETCTYPE, 1, &Data);
        CloseW();
    }

    VOID MSpeed()
    {
        BOOL kollision=FALSE;
        SHORT y=MAXH;
        UWORD clxdat=custom.clxdat;

        geta4();
        while(!Quit){
            y+=Speed;
            if(RasInfo.RyOffset!=y){
                if(y>MAXH) y=0;
                if(y<0) y=MAXH;
                RasInfo.RyOffset=y;
                WaitTOF();
                ScrollVPort(&VPort);
                clxdat=custom.clxdat;
                if((kollision==FALSE)&&((clxdat&4)==4)){
                    kollision=TRUE;
                    Speed=Inc=0;
                    SetRGB4(&VPort, 0, 15, 0, 0);
                    Delay(5);
                    SetRGB4(&VPort, 0, 15, 15, 15);
                }
            }
            else kollision=FALSE;
        }
    }
}

```

Listing 1. Das Beispielprogramm scrollt eine 2000 x 200 Pixel große BitMap

Alternative zu Screens: die View-Struktur

Intuition ausgebremst

Von Arno Gölzer

Intuition stellt uns Routinen zur Gestaltung des Bildschirms zur Verfügung, die fast keine Wünsche offenlassen. Diese Vielseitigkeit fordert aber auch einen immensen Verwaltungsaufwand. Einen Aufwand, der bei der Programmierung von Spielen nur selten gefragt ist.

Wir steigen bei der Realisierung unseres Ziels – eines einfachen Grafikbildschirms – tiefer ins Betriebssystem hinab und suchen die unterste Stufe des Displays. Eine kleine, unscheinbare Struktur ist verantwortlich für alles, was sich vor unseren Augen auf dem Monitor abspielt: die View-Struktur (Listing 1).

Welche Bedeutung sie hat, wird bei der Besprechung der einzelnen Komponenten dieser wichtigen Struktur klarer:

struct ViewPort • ViewPort

verweist auf den ersten ViewPort. Dieser ist ebenfalls eine Datenstruktur des Displays. In einem Task können mehrere ViewPorts initialisiert werden, aber nur einen View, der über sie wacht und ihre Fähigkeiten in seiner Struktur festlegt. Wir werden ViewPorts gleich eingehend beleuchten. Soviel sei vorweggenommen: Alle ViewPorts eines Views sind mit dem ersten Element ihrer Struktur verkettet. Der letzte ViewPort enthält als Adresse des Folge-ViewPorts den Wert NULL.

struct cprlist • LOFCprList

die Adresse der LongFrame-Copper-Liste. Der Copper ist einer der Coprozessoren des 6800er-Chips. Er überwacht den Elektronenstrahl, der das Bild auf den Monitor zeichnet und belegt in Abhängigkeit von dessen Position die Register der Customchips mit Bildschirmwerten. An welcher Position er diese Werte in die verschiedenen Register schreibt, wird in sogenannten Instruktionslisten festgelegt. LOFCprList ist die Adresse der Liste für ein Display ohne Zeilensprungverfahren.

struct cprlist • SHFCprList

diese Copper-Instruktionsliste ist für den Auflösungs-

Intuition ist leistungsfähig und bedienerfreundlich, aber leider speicherintensiv und langsam.

Hierbei hilft nur ein handgestricktes Display.

modus Interlace notwendig. Bei diesem Modus wird eine besondere Technik, das Zeilensprungverfahren, angewendet. Dabei teilt sich das darzustellende Bild in zwei Halbbilder, LongFrame und ShortFrame genannt, wobei SHFCprList die Adresse der Copper-Liste für ShortFrame ist.

SHORT DyOffset, DxOffset

bestimmen die X/Y-Position der linken oberen Ecke des Views auf dem Bildschirm.

UWORD Modes

die Darstellungsmodi des Amiga. Sie kennen sie aus der NewScreen-Strukturkomponente ViewModes. Im Headerfile graphics/view.h sind, neben der View-Struktur selbst, folgende Modi definiert:

```
#define PFBA 0x40L
#define DUALPF 0x400L
#define HIRES 0x8000L
#define LACE 4L
#define HAM 0x800L
#define SPRITES 0x4000L
#define VP_HIDE 0x2000L
#define GENLOCK_AUDIO 0x100L
#define GENLOCK_VIDEO 2L
#define EXTRA_HALFBRITE 0x80L
```

Sie werden bei der Besprechung der Strukturen in diesem Artikel immer wieder Parallelen zu Intuition-Elementen feststellen, da diese auf den View-Strukturen basieren. Bei der Erstellung eines eigenen Displays werden wir vom Betriebssystem mit mehreren Funktionen unterstützt. So belegt z.B. InitView(&View) die View-Struktur, auf die der als Parameter gelieferte Pointer verweist, mit den Standardwerten. Wir müssen später lediglich die Adresse des ersten ViewPorts und den View-Modus ergänzen.

Der Displaymodus richtet sich nach den Anforderungen des Programmierers. Dieser muß dabei bedenken, daß ein dem View zugehöriger ViewPort nur die hier vereinbarten Auflösungsmodi verwenden kann. Hat man HIRES INTERLACE angegeben, so können die ViewPorts alle Bildschirmauflösungen darstellen. Leider macht sich bei dieser Kombination das lästige Interlace-Flimmern bemerkbar – auch bei Viewports, die nicht in diesem Modus arbeiten (siehe Beispielpogramm Resolutions.c – Listing 5). Zur Komplettierung unseres Views fehlt somit nur noch die Adresse der ersten ViewPort-Struktur. Bei dieser Struktur (Listing 2), welche ebenfalls in graphics/view.h vereinbart ist, läßt sich die enge Verwandtschaft mit dem Intuition-Screen nicht verleugnen. Hier die Bedeutung der einzelnen Variablen:

struct ViewPort • Next

wie weiter oben erwähnt, sind alle ViewPorts eines Views miteinander verkettet. Der Anfang der Kette ist im ersten Element der View-Struktur vermerkt.

struct ColorMap • ColorMap

jeder ViewPort verfügt über eine eigene ColorMap-Struktur, die unter anderem die Farben für den ViewPort enthält. Das bedeutet, daß Sie für jeden ViewPort eine komplette Farbenpalette benutzen können. Die Funktion GetColorMap legt uns die Struktur an:

```
ViewPort.ColorMap=
GetColorMap(n);
```

Als Parameter erwartet die Struktur die Anzahl der Farben. Diese richtet sich nach dem gewählten ViewModus und der Anzahl der benutzten Bitplanes. Vor Ende des Programms gibt FreeColorMap() den beleg-

ten Speicher wieder frei:

```
FreeColorMap(&CMap);
```

Der Parameter ist die Adresse der mit GetColorMap() angelegten ColorMap-Struktur. Es folgen die vier ViewPort-Copperlisten:

struct CopList • Dsplns

Instruktionsliste für das Display des ViewPorts.

struct CopList • Sprlins

die Copperliste für die Darstellung von Sprites.

struct CopList • Crlins

dies ist die Color-Copperliste.

struct UCopList • UCoplins

dieses Feld kann auf eine vom Programmierer angelegte Copperliste verweisen.

SHORT DWidth, DHeight

hier werden die Ausmaße des ViewPorts in Pixel festgelegt.

SHORT DxOffset, DyOffset

die Position der linken oberen Ecke des ViewPort-Rechtecks, als Offset zum gesamten Display. Beachten Sie bitte, daß sich ViewPorts nicht überlappen dürfen. Mindestens eine Pixelreihe Abstand muß gehalten werden. Darüber hinaus ist es nicht möglich, ViewPorts nebeneinander darzustellen.

UWORD Modes

die Darstellungsart des ViewPorts. Es muß sich um einen der im Feld Modes der View-Struktur vereinbarten Modi handeln.

UBYTE SpritePriorities

Prioritäten der Sprites. Allerdings ist die Spritepriorität hardwaremäßig festgelegt und nicht veränderbar.

UBYTE reserved

reserviert für System.

struct RasInfo • RasInfo

Verbindung des ViewPorts zur BitMap. Auch hier steht uns wieder eine Initialisierungsroutine zur Verfügung:

InitVPort(&ViewPort) belegt die ViewPort-Struktur, auf die der Parameter verweist, mit Standardwerten. Nur vier Werte müssen wir ergänzen: die Breite und Höhe des ViewPorts, der

Verweis auf die ColorMap und den Pointer auf die RasInfo-Struktur.

Als einzige Unbekannte bleibt uns die RasInfo-Struktur (Listing 3). Sie kann, wie so viele Strukturen des Amiga-Betriebssystems, mittels des ersten Elements verkettet werden. Dies ist allerdings nur in den Spezialmodi, wie z.B. DUALPF notwendig. Es folgt die Adresse der BitMap-Struktur des Displays. Zum Schluß erwartet RasInfo noch die X/Y-Koordinate des Punktes im Raster, welcher an der linken oberen Ecke des ViewPorts dargestellt werden soll. Leider stellt uns das Betriebssystem keine Funktion zur Verfügung, welche die RasInfo-Struktur initialisiert, was in Anbetracht ihrer Kürze entschuldbar ist. Tatsächlich fehlt nur die Adresse der BitMap-Struktur. Diese Struktur (Listing 4) enthält grundlegende Informationen über das Display. Die Belegung der Komponenten besorgt:

```
InitBitMap(&BM,AP,W,H);
```

Als ersten Parameter erwartet die Funktion einen Zeiger auf die zu initialisierende BitMap-Struktur. Parameter zwei enthält die Anzahl der gewünschten Planes, während die beiden letzten Parameter die Breite und die Höhe einer Plane beschreiben. Dem Programmierer bleibt die Aufgabe, Speicherplatz für die Planes zu allokalieren und deren Adressen bekanntzugeben:

```
for (i=0;i<AP;i++){
if(!(BM.Planes[i]=
(PLANEPTR)AllocRaster
(W,H))) Error();
BltClear(BM.Planes[i],
RASSIZE(W,H),0);
}
```

Die Variablen entsprechen den Parametern des InitBitMap()-Aufrufs. Innerhalb des Schleifenrumpfes reserviert AllocRaster() den notwendigen Speicherplatz für jede Bildebene, der von BltClear sogleich gelöscht wird. RASSIZE() ist ein Makro, welches die Größe des zu löschenden Bereichs ermittelt. Es versteht sich von selbst, daß der für die Planes notwendige Speicher – allerdings nicht der für die verwaltende Struktur – im Chip-Memory liegen muß. Dafür sorgt AllocRaster(). BltClear() kann auch durch SetRast(&RP,col) ersetzt werden, wie auch aus dem Demo-programm Resolutions.c ersichtlich ist. Parameter sind

hierbei die Adresse des RastPorts und die gewünschte Farbe, mit welcher der Rastport gefüllt werden soll. In einer Intuition-Umgebung erhalten wir die Adresse eines RastPorts problemlos. Aber wie gelingt dies bei einem eigenen Display? Sie vermuten richtig, wir erstellen den RastPort selbst – oder besser wir lassen ihn erstellen:

```
InitRastPort(&RP);
RP.BitMap=&BM;
```

InitRastPort() übernimmt diese Aufgabe, wenn wir ihr die Adresse einer RastPort-Struktur übergeben. Wir müssen dann nur noch bekanntgeben, wo die BitMap-Struktur zu finden ist. Somit haben wir alle Daten beschafft, es trennen uns noch genau drei Funktionsaufrufe vom ersten eigenen Display. MakeViewport(&View, &Viewport) generiert aus den Daten der Strukturen View und ViewPort eine Copperliste. Falls die Komponente ColorMap der ViewPort-Struktur NULL enthält, nutzt MakeViewport() eine Default-Farbtabelle. Die Parameter sind die Adressen auf die fertig initialisierten Strukturen View und ViewPort. MrgCop(&View) fügt («merge») alle Copperlisten des Views mit der als Parameter gegebenen Adresse zu einer einzigen Instruktionsliste zusammen. Nun steht der Darstellung des Displays nichts mehr im Wege – oder? Stop! Wir müssen die Adresse des aktuellen Views noch retten, so daß wir unser Programm auch wieder korrekt beenden können:

```
struct View OldView;
OldView=GfxBase->
ActiView;
```

Ist die Adresse des alten Views nicht bekannt, gibt es kein Zurück zur Workbench bzw. zum CLI-Window. Der Aufruf der Funktion LoadView(&View) macht endlich das Display sichtbar. Durch den Aufruf dieser Funktion kann das Display auch wieder in den Vordergrund gebracht werden, wenn durch die Tastenkombination <Linke Amiga n> der Workbenchscreen nach vorne geklappt wurde. Zum besseren Verständnis sind alle besprochenen Prozeduren im Beispielpogramm Resolutions.c anschaulich demonstriert. Das Programm, welches Ihre Eingaben vom Joystick in Port 2 erwartet, erstellt ein Display im LoRes-Modus. Durch Betätigung des Sticks in die vier Richtungen schalten Sie auf einen

anderen Auflösungsmodus um. Programmende erfolgt nach der Betätigung des Feuerknopfs.

OpenW() ermittelt die Basisadresse der Graphics-Library, öffnet das GamePort-Device und erledigt die Initialisierung der Strukturen für das Display. InitStick() macht ihrem Namen alle Ehre, sie initialisiert über SetGPRequest() GamePort 2 (der rechte Port) für den Joystick.

Die Funktion DrawVP() zeichnet einige Ellipsen auf den neu initialisierten RastPort und macht die Zeichnung mittels LoadView() sichtbar. Danach warten wir in main() mittels WaitIO() auf die Betätigung des Sticks. Entsprechend der gewählten Richtung schalten wir den Modus um und berechnen die Copperlisten neu. Nach Betätigung des Feuerbuttons gibt CloseW() den reservierten Speicher wieder frei.

Wenn der alte View wieder eingeschaltet ist, wir warten zur Sicherheit mit WaitTOF() bis der

Elektronenstrahl die linke obere Ecke des Bildschirms erreicht hat, geben den für die Copperlisten notwendigen Speicher wieder ans System zurück. FreeVPorCopLists(&Viewport) gibt die Listen des ViewPorts frei. Die Funktionen FreeCprList(View.LOFCprList) und FreeCprList(View.SHFCprList) verrichten diese Arbeit für die Listen der View-Struktur. Die hier geschilderten Funktionen sind schnell in Programme, welche auf Intuition verzichten können, übernommen und dem jeweiligen Spiel angepaßt. Belohnt werden Sie für den Programmieraufwand mit einem rasanten Bildschirmaufbau und einem geringen Speicherbedarf. Sicher sind Sie mit uns der Meinung, daß dies der geringen Mühe wert war. pe

Der Autor Arno Gölzer ist ein Mann der ersten Stunde und schreibt schon einige Jahre für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Herrn Gölzer, Hans-Pinsel-Straße 2, 8013 Haar b. München).

```
struct View
{
    struct ViewPort *Viewport;
    struct cprlist *LOFCprList;
    struct cprlist *SHFCprList;
    SHORT DyOffset,DxOffset;
    UWORD Modes;
};
```

Listing 1. Die View-Struktur stellt die unterste Ebene des Displays dar

```
struct ViewPort
{
    struct ViewPort *Next;
    struct ColorMap *ColorMap;
    struct CopList *DspIns;
    struct CopList *SprIns;
    struct CopList *ClrIns;
    struct UCopList *UCopIns;
    SHORT DWidth,DHeight;
    SHORT DxOffset,DyOffset;
    UWORD Modes;
    UBYTE SpritePriorities;
    UBYTE reserved;
    struct RasInfo *RasInfo;
};
```

Listing 2. Der ViewPort ist eng mit dem Intuition-Screen verwandt

```
struct RasInfo
{
    struct RasInfo *Next;
    struct BitMap *BitMap;
    SHORT RxOffset,RyOffset;
};
```

Listing 3. Die RasInfo-Struktur koppelt ViewPort und BitMap


```
struct BitMap
{
    UWORD BytesPerRow; /* Bytes pro BitMap-Zeile (Breite/8) */
    UWORD Rows;        /* Anzahl der Zeilen (Höhe) */
    UBYTE Flags;        /* ??? ungenutzt */
    UBYTE Depth;        /* Anzahl der BitPlanen (BitEbenen) */
    UWORD pad; /* Platzhalter - PLANEPTR beginnen auf LONGWORD */
    PLANEPTR Planes[8]; /* die Adressen der BitEbenen
                        im Speicher */
};
```

Listing 4. Die BitMap
enthält elementare Daten über das Display

```
/*
Aztec C 3.4a
cc Resolutions.c +L
ln Resolutions.o -lc32
*/
/*
\\ Gölyz's Resolutions.c - die Bildschirmauflösungen des Amiga
*/
#include <exec/memory.h>
#include <exec/types.h>
#include <graphics/gfxbase.h>
#include <graphics/view.h>
#include <devices/inputevent.h>
#include <devices/gameport.h>
#include <functions.h>

#define WIDTH 320
#define HEIGHT 256
ULONG GPDev;
UWORD Mode[4]={
    0,LACE,HIRES,HIRES|LACE
};
STRPTR MText[4]={
    (STRPTR)" LoRes - Modus ",
    (STRPTR)" LowLace - Modus ",
    (STRPTR)" HiRes - Modus ",
    (STRPTR)" HiLace - Modus "
};
struct GamePortTrigger GPTrigger={
    GPTF_UPKEYS|GPTF_DOWNKEYS,0,1, 1
};
struct GfxBase *GfxBase;
struct View *OldView,View;
struct MsgPort *GPort;
struct IOSReq *GPrequest;
struct InputEvent ReadStick;
struct ViewPort VPort;
struct RasInfo RasInfo;
struct BitMap BMap;
struct RastPort RP;

VOID CloseW()
{
    if(OldView){
        LoadView(OldView);
        WaitTOF();
    }
    FreeVPortCopLists(&VPort);
    FreeCprList(View.LOFCprList);
    FreeCprList(View.SHFCprList);
    if(VPort.ColorMap) FreeColorMap(VPort.ColorMap);
    if(BMap.Planes[0]) FreeRaster(BMap.Planes[0],WIDTH,HEIGHT);
    if(GPDev) CloseDevice(GPrequest);
    if(GPrequest) DeleteStdIO(GPrequest);
    if(GPort) DeletePort(GPort);
    if(GfxBase) CloseLibrary(GfxBase);
    exit(0);
}

VOID OpenW()
{
    REGISTER SHORT i;
```

```
if(!(GfxBase=(struct GfxBase *)
    OpenLibrary("graphics.library",0))) CloseW();
if(!(GPort=CreatePort("GPort",0))) CloseW();
if(!(GPrequest=CreateStdIO(GPort))) CloseW();
if(GPDev=(ULONG)
    OpenDevice("gameport.device",1,GPrequest,0)) CloseW();
OldView=GfxBase->ActiView;
InitView(&View);
InitVPort(&VPort);
InitRastPort(&RP);
InitBitMap(&BMap,1,WIDTH,HEIGHT);
if(!(BMap.Planes[0]=(PLANEPTR)
    AllocRaster(WIDTH,HEIGHT))) CloseW();
View.ViewPort =&VPort;
View.Modes =HIRES|LACE;
VPort.DWidth =WIDTH;
VPort.DHeight =HEIGHT;
VPort.RasInfo =&RasInfo;
VPort.ColorMap =GetColorMap(2);
RasInfo.BitMap =&BMap;
RasInfo.RxOffset=0;
RasInfo.RyOffset=0;
RasInfo.Next =NULL;
RP.BitMap =&BMap;
SetRGB4(&VPort,0, 5, 5, 5);
SetRGB4(&VPort,1,10,10,10);
MakeVPort(&View,&VPort);
MrgCop(&View);
}

VOID Print(mnr)

    SHORT mnr;
{
    SHORT len=strlen(MText[mnr]);

    SetAPen(&RP,1);
    SetDrMd(&RP,JAM2);
    Move(&RP,WIDTH/2-len*4,50);
    Text(&RP,MText[mnr],len);
}

VOID SetGPrequest(doit,c,l,d)
    BOOL doit;
    UWORD c;
    ULONG l;
    APTR d;
{
    GPrequest->io_Command =c;
    GPrequest->io_Length =l;
    GPrequest->io_Data =d;
    if(doit) DoIO(GPrequest);
}

VOID InitStick()
{
    UBYTE Data=GPCT_RELJOYSTICK;
    ULONG lg=sizeof(struct GamePortTrigger),
        li=sizeof(struct InputEvent);

    SetGPrequest(1,GPD_SETCTYPE,1,&Data);
    SetGPrequest(1,GPD_SETTRIGGER,lg,&GPTrigger);
    SetGPrequest(0,GPD_READEVENT,li,&ReadStick);
}

VOID DrawWP()
{
    REGISTER SHORT i;

    SetRast(&RP,1);
    Print(0);
    SetAPen(&RP,0);
    for(i=32;i<WIDTH-1;i+=32)
        DrawEllipse(&RP,i-1,140,32,1/10);
    LoadView(&View);
}
```

Listing 5. Resolutions.c demonstriert,
wie man ein Display selbst erstellen kann


```

VOID main ()
{
    UBYTE Data=GPCT_NOCONTROLLER;
    SHORT nr=0;
    OpenW();
    InitStick();
    DrawVP();
    while(1){
        SendIO(GPRequest);
        WaitIO(GPRequest);
        if(ReadStick.1e_Code==IECODE_LBUTTON) break;
        else if(ReadStick.1e_X) nr=(ReadStick.1e_X>0)?0:1;
    }
}

```

```

else if(ReadStick.1e_Y) nr=(ReadStick.1e_Y>0)?2:3;
VPort.Modes=Mode[nr];
MakeVPort(&View,&VPort);
MrgCop(&View);
LoadView(&View);
Print(nr);
}
SetGPRequest(1,GPD_SETCTYPE,1,&Data);
CloseW();
}

```

Listing 5. (Schluß)

Die enormen Fähigkeiten auf dem Gebiet der Grafik verdankt der Amiga nicht zuletzt seinen Coprozessoren. Einer dieser Coprozessoren ist entscheidend am Bildaufbau beteiligt: der Copper.

Von Arno Gölzer

Bereits beim Aufbau eines einfachen Grafik-Displays (Sie finden in diesem Heft auf Seite 77 einen Artikel, der sich damit befaßt.) werden Sie, wenn auch eher am Rande, mit dem Copper konfrontiert. Es geht dabei um die Erstellung der Copper-Programme, den sogenannten Copper-Listen, durch die Betriebssystem-Funktionen MakeVPort() und MrgCop().

MakeVPort() ist für die ViewPort-(Zwischen-)Listen zuständig. MrgCop() fügt diese zu der Gesamt-(Hardware-)Instruktionsliste des Displays zusammen. Je nach Auflösungsmodus generieren die Funktionen eine oder - im Interlace-Modus - zwei Listen. Mit dem Aufruf der LoadView()-Funktion werden die Adressen dieser Listen in die Custom-Register »cop1lc« und »cop2lc« übertragen und

der Copper beginnt mit deren Ausführung.

Arbeiten Sie mit einem Intuition-Screen, so brauchen Sie sich um die Freigabe des für die Copperlisten notwendigen Speichers vor Programmende nicht zu kümmern. Beim Schließen des Screens gibt Intuition alle zugehörigen Copperlisten, auch die User-Copperlisten, wieder frei.

Haben Sie jedoch Ihr Display unabhängig von Intuition »gezimmert«, so sollten Sie als disziplinierter Programmierer die für die Listen reservierten Speicherplätze mittels der Funktionen FreeCopList() und FreeCprList() wieder freigeben. Die zuletzt genannte Funktion übernimmt die Freigabe der Hardware-Copperliste. Beide Funktionen erwarten die entsprechenden Strukturpointer. Ein Beispiel:

```

FreeCprList(View.LOF
CprList); /* struct

```

An der Coppercabana


```
cprlist */
FreeCopperList(ViewPort.
CopperList); /* struct
CopperList */
```

FreeVPortCopperLists(&ViewPort) kann die Funktion FreeCopperList() ersetzen, sie gibt alle Copperlisten des genannten ViewPorts frei, auch die vom User selbst installierten.

Möchte man nur die User-Copperliste entfernen, hilft der Aufruf der FreeVPortCopperLists()-Funktion, mit nachfolgender Neuberechnung des Displays – ohne User-Copperliste. Im Grafikdisplay geschieht das wie gewohnt mit den oben erwähnten Funktionen. Bei der Arbeit mit Intuition genügt der Aufruf der Funktion RemakeDisplay().

Die enge Verwandtschaft des Coppers mit dem Elektronenstrahl, der das Bild auf dem Monitor aufbaut, läßt sich nach der Besprechung der folgenden Copper-Funktionen nicht mehr leugnen: WaitTOF() wartet, bis der Elektronenstrahl mit dem Neuaufbau des Bildes beginnt. Diese Funktion setzt man vornehmlich dort ein, wo der Bildschirm aufbau einer ständigen Veränderung unterliegt. Sie vermeiden so das unschöne Flackern, welches z.B. während des Scrollings oder bei einer bewegten Grafik entstehen kann.

Die Funktion WaitBOVP(&ViewPort) stellt in gewisser Hinsicht das Gegenstück zu WaitTOF() dar. Sie hält das auszuführende Programm so lange an, bis der Elektronenstrahl die unterste Zeile des ViewPorts erreicht hat, dessen Adresse als Parameter übergeben wurde.

Das letzte Beispiel, VBeamPos(), ermittelt die aktuelle Y-Position des Elektronenstrahls: ULONG YPos; YPos=VBeamPos();

Bis zur Rückgabe und vor allem bis zur Auswertung des Rückgabewertes, stimmt dieser natürlich in aller Regel nicht mehr mit der tatsächlichen Strahlposition überein. Dennoch findet die Funktion, gerade bei der Spieleprogrammierung, einen sinnvollen Einsatz. Ihr Ergebnis kann als Zufallszahl oder zumindest als Verknüpfungs- oder Startwert für die in den Compiler-Libraries enthaltenen Zufallszahlengeneratoren dienen.

Sie sehen, die Arbeit des Coppers ist tatsächlich sehr eng an den Elektronenstrahl gebunden. Seine Aufgabe besteht darin, auf eine bestimmte

Rasterposition zu warten und beim Erreichen dieser Position den Inhalt eines Customchip-Registers zu ändern.

Die Register befinden sich ab der Adresse 0xDFF000 im Speicher. Jedoch sollten Sie nicht direkt, sondern über die im Headerfile »hardware/custom.h« definierte Custom-Struktur auf die Register zugreifen. Listing 1 stellt die Struktur vor und erläutert grob den Inhalt der einzelnen Komponenten.

Listing 1 zeigt, daß Copper nicht auf alle Register Zugriff hat. Auf die Register bis einschließlich »strlong« hat er keinen Einfluß und auf die folgenden, bis »dsksync«, nur dann, wenn Bit 0 des Copper-Control-Registers »copcon« gesetzt war.

Die Inhalte der übrigen Register können jederzeit über den Copper manipuliert werden. Zum Aufbau der hierzu notwendigen Copperliste stellt das Betriebssystem dem Programmierer drei Funktionen zur Verfügung:

```
CWait(&UCList,y,x);
CMove(&UCList,r,w);
CBump(&UCList);
```

Mit diesem nicht gerade gewaltigen Arsenal von Befehlen lassen sich in Verbindung mit den Hardware-Registern, die tollsten Effekte erzielen, z.B. vielfarbige Schriften. Aber auch die in vertikaler Richtung verschiebbaren Intuition-Screens und die VSprites erzeugen Sie auf diese Weise.

Eigentlich existiert neben Wait und Move noch der Skip-Befehl, der dafür sorgt, daß der nachfolgende Befehl der Copperliste übersprungen wird, falls sich der Rasterstrahl bereits unterhalb der angegebenen Position befindet. Dieser bleibt allerdings Assembler-Programmierern vorbehalten. In den Hochsprachen müssen wir auf Skip verzichten und uns mit den oben genannten Funktionen begnügen. Mit ihrer Hilfe bauen wir unsere Copperlisten auf.

CWait() implementiert in der Copperliste der UCopList-Struktur, auf die der erste Parameter verweist, den Befehl Wait. Dieser Befehl veranlaßt den Copper dazu, zu warten, bis der Elektronenstrahl die über die Parameter y und x angegebene Position auf dem Bildschirm erreicht. Beachten Sie bitte, daß die Funktion dabei den y-Wert vor dem x-Wert erwartet! Die Nulllinie der y-Position ist die oberste Zeile

des ViewPorts, in dem die Copperliste eingebunden wird. Die erste x-Position (Null) befindet sich links, ca. 55 bis 65 Positionen (je nach Einstellung des X-Offsets) außerhalb des eigentlichen Displays (siehe auch Listing 2 und 3). Der Start des Displays liegt also etwa bei Position 60, das Ende bei etwa 220. Daraus errechnet eine Wait-Auflösung von 4 Pixel im Hires-Modus für das Display (640/(220-60)). Nach jedem Copper-Kommando muß der Befehlszeiger der Liste mittels der CBump()-Funktion erhöht werden. Glücklicherweise sind jedoch im Headerfile »graphics/gfxmacros.h« für den vergesslichen Programmierer Makros definiert, die die Inkrementierung übernehmen: CWAIT() und CMOVE(). Beide werden mit den gleichen Parametern wie die entsprechenden Funktionen aufgerufen. Leider hat man auf das Vertauschen der Positionsparameter des CWAIT()-Makros verzichtet. Nachdem der Elektronenstrahl die in Wait vorgeschriebene Position erreicht hat, belegt Move das angegebene Register mit einem beliebigen Wert.

CMOVE() baut die notwendigen Befehle in unsere Copperliste ein. Der zweite Parameter des Makros ist das gewünschte Register, Parameter 3 stellt den Wert dar, der in dieses Register eingetragen werden soll. Die kleinste Auflösung in vertikaler Richtung beträgt ein Pixel. In

Vom Strahl ausgebremst

horizontaler Richtung macht sich jedoch die rasante Geschwindigkeit des Elektronenstrahls und die Ausführungszeit der Befehle Wait und Move bemerkbar:

```
CWAIT(&UCList,0,60);
CMOVE(&UCList,custom.
color[0],0xF00);
CMOVE(&UCList,custom.
color[0],0xFFFF);
```

Diese Zeilen sollten eigentlich ein Programm erzeugen, welches die linke obere Ecke des Displays kurz rot und danach gleich wieder weiß färben sollte. Wegen der oben genannten Gründe erscheint jedoch eine, im Hires-Modus ca. 16 Pixel lange rote Linie. Sollten Sie versuchen, Copper und Elektronenstrahl mit der folgenden Sequenz zu überlisten...

```
CWAIT(&UCList,0,60);
CMOVE(&UCList,custom.
```

```
color[0],0xF00);
CWAIT(&UCList,0,62);
CMOVE(&UCList,custom.
color[0],0xFFFF);
```

...und zu einer höheren Auflösung zu zwingen, ernten Sie vom Copper nur Hohngelächter und von Ihrem Display eine noch längere Linie, denn auch Wait hat natürlich, wie oben erwähnt, seine Ausführungszeit.

Die mit Hilfe der beiden Makros aufgebaute Copperliste wird bei jedem Strahldurchlauf abgearbeitet. Damit dies korrekt geschehen kann, darf sie eine bestimmte Länge nicht überschreiten. Stellen Sie sich vor, der Elektronenstrahl erreicht die rechte untere Ecke des Displays, also die Endposition, während der Copper noch mit seiner Instruktionsliste zu kämpfen hat.

Das Ende einer Copperliste kennzeichnet ein Befehl, den der Copper nicht ausführen kann: CWAIT(&UCList,10000,255). Demnach wartet er, bis ihn der nächste Start des Elektronenstrahls in der linken oberen Display-Ecke auffordert, seine Instruktionsliste erneut zu bearbeiten.

Für diese »Schlußmarkierung« existiert ein weiteres Makro, welches nur noch die Adresse der UCopList-Struktur verlangt:

```
CEND(&UCList);
```

Jetzt steht dem Start unserer eigenen Copperliste nichts mehr im Wege. Nachdem wir die Adresse unserer UCopList-Struktur im gewünschten ViewPort bekanntgegeben haben, fehlt nur noch die Einbindung der Liste mittels MakeVPort() und MrgCop() bzw. unter Intuition, RemakeDisplay(). Danach arbeitet der Copper immer wieder das gleiche Programm ab – 50mal in einer Sekunde.

Der Aufbau einer veränderbaren Copperliste – z.B. könnten Sie immer an der Mausposition die Hintergrundfarbe für mehrere Zeilen verändern – ist leider nicht ohne weiteres möglich. Die Copperliste muß vor jedem Neuaufbau »gesäubert« werden. Hierzu dient die Funktion UCopListInit(&UCList,n) oder kürzer, als Makro:

```
CINIT(&UCList,n);
```

Die UCopList-Struktur enthält die Adresse der vom User definierten Copperliste (vergleiche Headerfile graphics/copper.h). Ist diese Adresse gleich NULL, legt die Funktion einen Buffer an, der »n« Einträge aufnehmen kann und liefert einen

Zeiger darauf. **Achtung:** Diese Funktion legt allen Behauptungen in der gängigen Literatur zum Trotz kein UCopList-Struktur an! Der Speicher für diese Struktur muß, wie im Beispielprogramm gezeigt, mit AllocMem() angefordert werden. Gibt man aber als Parameter des CINIT()-Makros die Adresse einer UCopList-Struktur an, die bereits die Adresse einer User-Copperliste enthält, so bereitet das Makro die Struktur auf die Aufnahme einer neuen Copperliste vor. Nochmals Achtung: Es genügt nicht, wenn Sie die Struktur einfach löschen! Sie könnten dann zwar auch eine neue Copperliste aufbauen lassen, aber Sie verlieren den Verweis auf die alte Liste – somit können Sie den dafür reservierten Speicher nicht mehr freigeben!

Listing 2 zeigt den wichtigsten Teil eines Programms, mit der Lösung für eine veränderbare (oder besser: neu aufbaubare) User-Copperliste (eine fertige Lösung bietet das Programm »HotBottom« – siehe den Artikel »Cola, Fanta, Sprite« auf Seite 53). In MakeCList() reinitialisiert der Aufruf des Makros CINIT() bei jedem Aufruf der Funktion die UCopList-Struktur, so daß sofort eine neue Copperliste – speziell für

die neue Y-Position des Mauszeigers – aufgebaut werden kann. Die Liste veranlaßt den Copper, in einem Bereich von 10 Pixel um die Pointerposition den Wert des Farbregisters 0 zu ändern. Wir erreichen das Register sehr einfach über die Custom-Struktur:

```
custom.color[0]
```

Auch in unserem Demo-Listing (Listing 3) arbeiten wir mit Farben. CLIColor demonstriert die Handhabung des Coppers an einem einfachen Beispiel. Das Programm öffnet ein kleines Fenster in der Titelleiste des Workbenchscreens und

Vielfalt mit zwei Planes

stellt den Hintergrund dieses (immer noch nur 2 Bit-Planes großen) Screens in 16 verschiedenen Farbtönen dar – bis das Window geschlossen wird.

Rufen Sie das Programm aus dem CLI mit »RUN« auf, so können Sie problemlos weiterarbeiten. Die Hauptfunktion main() ruft gleich zu Beginn die Funktion OpenW() auf. Diese öffnet die benötigten Libraries und das Window. Weiterhin verschafft sie uns den Speicher für die UCopList-Struktur für den Strukturpointer UCList.

Warum vereinbaren wir nicht anstelle des Zeigers gleich eine komplette UCopList-Struktur? Bedenken Sie bitte, daß die Funktion FreeVPortCopLists() nicht nur den Speicher für die Copperliste, sondern auch den für die Struktur freigibt. Nach Programmende werden aber neben dem Speicher für den Programmcode auch die Speicheradressen aller Variablen freigegeben – darunter auch unsere UCopList-Struktur. Das Resultat ist ein besonders hässisch blinkender 0009er Guru!

Noch eine Frage: Müßte die Struktur nicht im Chip-Memory liegen, da doch der Copper nur Daten, die in den unteren 512 KByte abgelegt sind, verarbeiten kann? Nein!!! Den Copper interessiert nur seine Instruktionsliste und nicht die Struktur, die diese Liste verwaltet. Die Liste liegt selbstverständlich im Chip-Memory.

Die Funktion MakeCList() baut im Demo eine Copperliste von 33 Instruktionen zusammen. CWAIT() und CMOVE() wechseln sich ständig ab. CMOVE() ändert dabei den Inhalt den Farbregisters 0, in Abhängigkeit der Schleifenvariable. CEND() schließt die Liste ab. Nach der Belegung der ViewPort-Variable UCopIns, wird sie mit der Intuition-Funk-

tion RemakeDisplay() gestartet. Danach wartet CLICol nur noch auf die Betätigung des CloseGadgets. CloseW() sorgt für die Freigabe aller Ressourcen. Sie sehen, welche Effekte Sie, dank des Coppers, mit einem kleinen Programm von nicht einmal einer Seite erreichen können. Dabei haben wir uns nur um Farbregister 0 gekümmert und alle anderen Register außer acht gelassen – Listing 1 läßt ahnen, welche Möglichkeiten offenstehen.

Vielleicht versuchen Sie sich selbst einmal mit der Copper-Programmierung? Bedenken Sie, was Sie alleine mit der Manipulation des Farbregisters der Schriftfarbe erreichen können! Also los – sofort testen ist angesagt. Sie müssen nur in MakeCList() die Nullen in GetRGB4() und in CMOVE() gegen eine Eins tauschen! Oder versuchen Sie sich doch gleich mit der veränderbaren Copperliste aus Listing 2. Ganz gleich wie Sie sich entscheiden, wir sind sicher: Sie werden dabei eine Menge Spaß haben! so

Arno Götzler ist ein Mann der ersten Stunde und schreibt schon seit über zwei Jahren für das AMIGA-Magazin und die AMIGA-Sonderhefte. Sie können ihn über unsere Verlagsadresse erreichen (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Herrn Arno Götzler, Hans-Pinsel-Straße 2, 8013 Haar bei München).

```
struct Custom[
/* Copper hat keinen Zugriff auf die folgenden Register: */
WORD bltddat; /* Blitter Ausgabedaten */
WORD dmaconr; /* DMA-Kontrollregister lesen */
WORD vposr; /* Höherwert. Bit der vertikalen E.Strahl-
Position lesen */
WORD vhpshr; /* Vert. und horiz. E.Strahl-Position lesen */
WORD dskdatr; /* Disk-Daten lesen */
WORD joy0dat; /* Daten f Maus od Joystick an Port 0 (links) */
WORD joy1dat; /* Daten für Maus oder Joystick an Port 1
(rechts) */
WORD clxdat; /* Kollisions-Datenregister lesen */
WORD adkconr; /* Audio- und Diskkontrolle lesen */
WORD pot0dat; /* Potentionmeterstand Port 0 (links) lesen */
WORD pot1dat; /* Potentionmeterstand Port 1 (rechts) lesen */
WORD potinp; /* Potentiometer-Pin Daten lesen */
WORD serdatr; /* Serieller Port Daten und Status lesen */
WORD dskbytr; /* Disk-Datenbyte lesen */
WORD intenar; /* Interrupt-Enable lesen */
WORD intreqr; /* Interrupt-Request lesen */
APTR dskpt; /* Adresse der Disk-Daten */
WORD dsklen; /* Länge der Disk-Daten */
WORD dskdat; /* Disk-Daten schreiben */
WORD refptr; /* Memory Refresh Pointer */
WORD vposw; /* Höherwert. Bit der vertikalen E.Strahl-Pos.
schreiben */
WORD vhpshw; /* Vertikale und horizontale E.Strahl-Position
schreiben */
WORD copcon; /* Copper Kontrollregister */
WORD serdat; /* Serieller Port Daten und Stopbits schreiben */
WORD serper; /* Serieller Port Kontrollregister und Baudrate */
WORD potgo; /* Potentiometer-Port Daten schreiben; Start bei
Zugriff */
WORD joytest; /* Initialisiert joy0dat und joy1dat */
WORD strequ; /* für horiz. Sync. mit Vert.Blank und EquFrame */
WORD strvbl; /* Strobe für horiz. Sync. mit Vert. Blank */
WORD strhor; /* Strobe für horizontale Synchroisation */
```

```
WORD strlong; /* Kennzeichnung für lange horizontale Zeile */
/* Copper hat Zugriff auf die folgenden Register, wenn copcon gesetzt
ist: */
WORD bltcon0; /* Blitter Kontroll-Register 0 */
WORD bltcon1; /* Blitter Kontroll-Register 1 */
WORD bltawm; /* Maske für erstes Datenwort */
WORD bltalwm; /* Maske für letztes Datenwort */
APTR bltqpt; /* Adresse der Blitter-Quelldaten C */
APTR bltbpt; /* Adresse der Blitter-Quelldaten B */
APTR bltap; /* Adresse der Blitter-Quelldaten A */
APTR bltdpt; /* Adresse der Blitter-Zieldaten */
WORD bltsize; /* Dimension des Blitter-Rechtecks;Start bei
Zugriff */
WORD pad2d[3]; /* - */
WORD bltmod; /* Blitter-Modulo für Quelldaten C */
WORD bltbmod; /* Blitter-Modulo für Quelldaten B */
WORD bltamod; /* Blitter-Modulo für Quelldaten A */
WORD bltdmod; /* Blitter-Modulo für Zieldaten */
WORD pad3d[4]; /* - */
WORD bltodat; /* Blitter Quell-Datenregister C */
WORD bltbdat; /* Blitter Quell-Datenregister B */
WORD bltadat; /* Blitter Quell-Datenregister A */
WORD pad3b[4]; /* - */
WORD dsksync; /* Disk-Synchronisationscode */
/* Copper hat immer Zugriff auf die folgenden Register: */
ULONG cop1lc; /* Adresse der Long-Frame Copperliste */
ULONG cop2lc; /* Adresse der Short-Frame Copperliste */
WORD copjmp1; /* bei Zugriff Start der Copperliste cop1lc */
WORD copjmp2; /* bei Zugriff Start der Copperliste cop2lc */
WORD copins; /* Copper Befehlsregister */
WORD diwstrt; /* linke obere Ecke des Displays */
WORD diwstop; /* rechte untere Ecke des Displays */
WORD ddfstrt; /* Beginn Display-Daten */
WORD ddfstop; /* Ende Display-Daten */
WORD dmacon; /* DMA-Kontrollregister schreiben */
WORD clxcon; /* Kollisionskontrolle schreiben */
WORD intena; /* Interrupt-Enable schreiben */
```



```

UWORD intreq; /* Interrupt-Request schreiben */
UWORD adkcon; /* Audio- und Diskkontrolle schreiben */
struct AudChannel{
    UWORD *ac_ptr; /* Adresse der Audio-Daten */
    UWORD ac_len; /* Länge der Daten ab Adresse ac_ptr */
    UWORD ac_per; /* Periodendauer */
    UWORD ac_vol; /* Lautstärke */
    UWORD ac_dat; /* DMA-Datenbuffer */
    UWORD ac_pad[2]; /* - */
} aud[4]; /* die vier Audio-Kanäle */
APTR bplpt[6]; /* Adressen der Bitplanen 1-6 */
UWORD pad7c[4]; /* - */
UWORD bplcon0; /* Blitter Kontrollregister 0 - ViewModi */
UWORD bplcon1; /* Blitter Kontrreg 1 - Scrolling pixelweise */
UWORD bplcon2; /* Blitter Kontrreg 2 - Videoprioritäten */
UWORD pad83; /* - */
UWORD bpl1mod; /* Modulo für ungerade Bitplanen */
UWORD bpl2mod; /* Modulo für gerade Bitplanen */
UWORD pad86[2]; /* - */
UWORD bpldat[6]; /* BitPlane (1-6) Daten */
UWORD pad8e[2]; /* - */
APTR sprpt[8]; /* Adresse der Spritedaten 1-8 */
struct SpriteDef{
    UWORD pos; /* X/Y - Anfangsposition */
    UWORD ctl; /* Y - Endposition */
    UWORD dataa; /* WORD 1 der darzustellenden Sprite-Zelle */
    UWORD datab; /* WORD 2 der darzustellenden Sprite-Zelle */
} spr[8]; /* die 8 Hardware-Sprites */
UWORD color[32]; /* Farbregister 1-32 */
}

```

Listing 1. Die Custom-Struktur vereinfacht den Zugriff auf die Custom-Register

```

...
VOID MakeCList(y)
    SHORT y;
{
    if(y==OldY) return;
    CINIT(UCList,5);
    CWAIT(UCList,y-5,0);
    CMOVE(UCList,custom.color[0],NewCol);
    CWAIT(UCList,y+5,0);
    CMOVE(UCList,custom.color[0],OldCol);
    CEND(UCList);
    VPort->UCopIns=UCList;
    RemakeDisplay();
    OldY=y;
}
...
VOID main()
{
    ...
    if(class==MOUSEMOVE){
        MakeCList(Screen->MouseY) CloseW();
    }
    ...
}
...

```

Listing 2. Eine veränderbare Copperliste muß vor jedem Neuaufbau reinitialisiert werden

```

/*
cc CLIColor.o +L
ln CLIColor.o -lc32
*/
/*
\\ Gölyz's Copper Demo
*/
#include <intuition/intuitionbase.h>
#include <graphics/gfxmacros.h>
#include <hardware/custom.h>
#include <exec/memory.h>
#include <exec/types.h>
#include <functions.h>

```

```

struct NewWindow nwin={
    0,0,225,10,-1,-1,CLOSEWINDOW,
    ACTIVATE|WINDOWCLOSE|WINDOWDEPTH|WINDOWDRAG,
    NULL,NULL,(STRPTR)"Gölyz's CLI-Color",
    NULL,NULL,0,0,0,0,WBENCHSCREEN
};
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct UCList *UCList=NULL;
struct Screen *Screen;
struct Window *win;
struct ViewPort *VPort;

VOID CloseW()
{
    if(UCList){
        FreeVPortCopLists(VPort);
        RemakeDisplay();
    }
    if(win) CloseWindow(win);
    if(GfxBase) CloseLibrary(GfxBase);
    if(IntuitionBase) CloseLibrary(IntuitionBase);
}

VOID OpenW()
{
    REGISTER SHORT i;

    if(!(IntuitionBase=(struct IntuitionBase *)
        OpenLibrary("intuition.library",0))) CloseW();
    if(!(GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library",0))) CloseW();
    if(!(win=OpenWindow(&nwin))) CloseW();
    if(!(UCList=(struct UCList *)
        AllocMem(sizeof(struct UCList),MEMF_PUBLIC|MEM
        F_CLEAR))) CloseW();
    Screen=(struct Screen *)IntuitionBase->ActiveScreen;
    VPort =&Screen->ViewPort;
}

ULONG GetMessage()
{
    struct IntuiMessage *msg;
    ULONG class=0;

    if(msg=(struct IntuiMessage *)GetMsg(win->UserPort)){
        class=msg->Class;
        ReplyMsg(msg);
    }
    return(class);
}

VOID MakeCList()
{
    REGISTER SHORT i,col=GetRGB4(VPort->ColorMap,0);

    for(i=0;i<16;i++){
        CWAIT(UCList,i*16,0);
        CMOVE(UCList,custom.color[0],col&0xF1(15-i));
    }
    CEND(UCList);
    VPort->UCopIns=UCList;
    RemakeDisplay();
}

VOID main()
{
    SHORT Quit=0;
    ULONG class;

    OpenW();
    MakeCList();
    while(!Quit){
        Wait(1<<win->UserPort->mp_SigBit);
        class=GetMessage();
        if(class==CLOSEWINDOW) Quit=1;
    }
    CloseW();
}

```

Listing 3. Workbenchscreen muß Farbe bekennen: Copper-Demo »CLIColor«

Der Sturz ins Bodenlose

Ein antikes Sprichwort sagt:
»Ob Holzleiter, Strickleiter,
Abteilungsleiter oder
Räuberleiter – wer raufsteigt,
kann auch runterfallen.«

Von Hans Waldhäusel

Ganz unten ist man schnell bei »Fast Freddie«, einem neuen Vertreter der bekannten und beliebten – wenn auch schon etwas antiquierten – Leiter-, Sammel-, Hüpfspiele. In Assembler geschrieben, blitzschnell und manchmal hunds-gemein. Mit gespeichertem Highscore, fetziger Musik und jeder Menge Gegner, die sich einen Spaß daraus machen, den armen Fred vom Gerüst zu schubsen. Fred ist Pazifist – er kann sich nicht wehren; schießen ist nicht vorgesehen. Nur Schnelligkeit, Taktik und Reaktionsvermögen führen zum Ziel. Bloß – dieses Ziel ist 50 Level entfernt.

Wir haben für Sie eine Bildschirm-Hardcopy abgedruckt (Bild 1), damit Sie sich beim Weiterlesen eine bessere Vorstellung von »Fast Freddie« machen können. Vorbild und Anregung für dieses Spiel holte sich der Michael Baumgaertner, der Programm-Autor, aus seligen 64er-Tagen – »Fast Eddie« ist wohl jedem Umsteiger ein Begriff und die Namensverwandtschaft kommt nicht von ungefähr. Hier, auf dem Amiga, muß der schnelle Fred in jedem Level mindestens neun von zehn möglichen Gegenständen einsammeln, erst dann darf er in den nächsten Level wechseln.

Und auch dafür muß er sich noch plagen, da der Ausgang (ein Sprite namens »EXIT«) nicht stationär ist, sondern sich ebenfalls über den Bildschirm bewegt. Bei der Jagd nach den Gegenständen, die es einzu-

sammeln gilt, muß sich Freddie vor seinen Gegnern höllisch in Acht nehmen; die leiseste Berührung kostet ihn eines seiner fünf Leben. Für jeden gesammelten Gegenstand gibt es Punkte, und bei Verlassen des Levels mittels »EXIT« (anstatt der etwas schmerzhafteren Schwerkraft-Variante – »Da ist kein Geländeeeeeeeeer...!«) gibt es zusätzlich 100 Bonuspunkte für jedes noch in Reserve stehende Leben. Emsiges Punktehamstern wird belohnt: bei 5000, 10000, 15000 usw. Punkten gibt es ein Extraleben. Aber Fred ist keiner von den Starken; mehr als sieben Reserve-Leben kann er nicht mit-schleppen – schade! Aber schließlich kann man einem armen Wesen, das sich seine Punkte durch Laufen (Joystick links/rechts), Klettern (Joystick oben/unten) und Hüpfen (Feuerknopf) erschuften muß, nicht auch noch einen Koffer auf den Rücken binden – oder?

Um »Fast Freddie« zu starten, öffnen Sie bitte ein CLI-Window – ein Start von der Workbench ist nicht möglich! Wechseln Sie in jenes Directory, in dem sich das Highscore-File »HI« befindet, also beispielsweise

```
cd Freddie
```

wenn Sie »Fast Freddie« und das Highscore-File in einem eigenen Directory namens »Freddie« abgelegt haben. Dann tippen Sie nach dem CLI-Prompt einfach

```
FastFreddie
```

und Ihrem Spielspaß steht nichts mehr im Wege. Nach kurzer Ladezeit befinden Sie sich im Start-Bild und bekommen neben Stereo-Sound – den

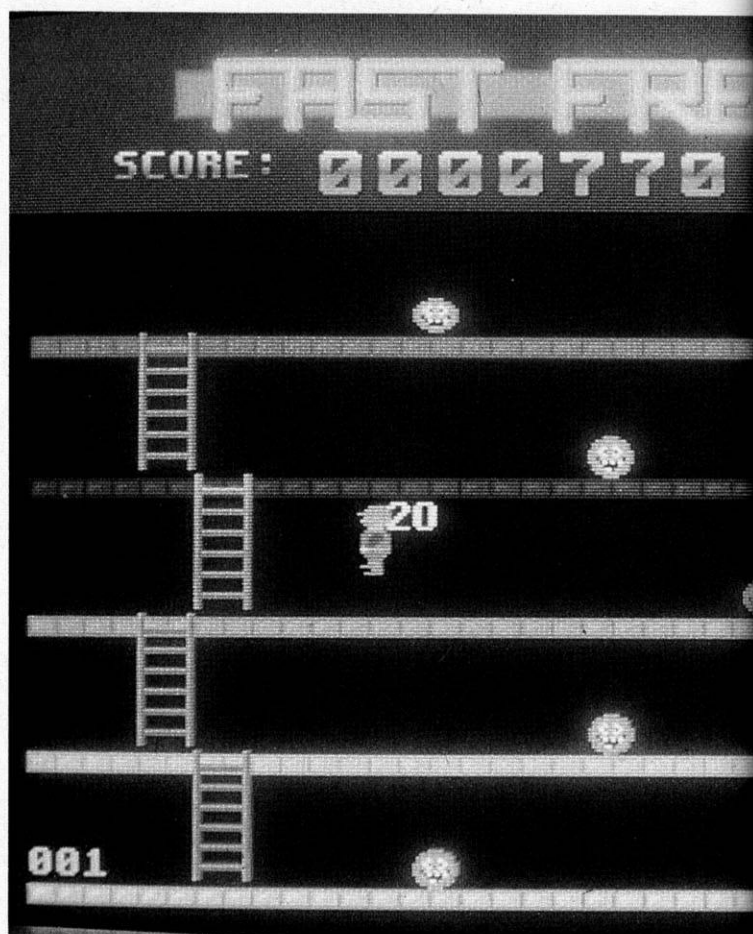


Bild 1. Auch nach einem steilen Aufstieg: Man ist schneller

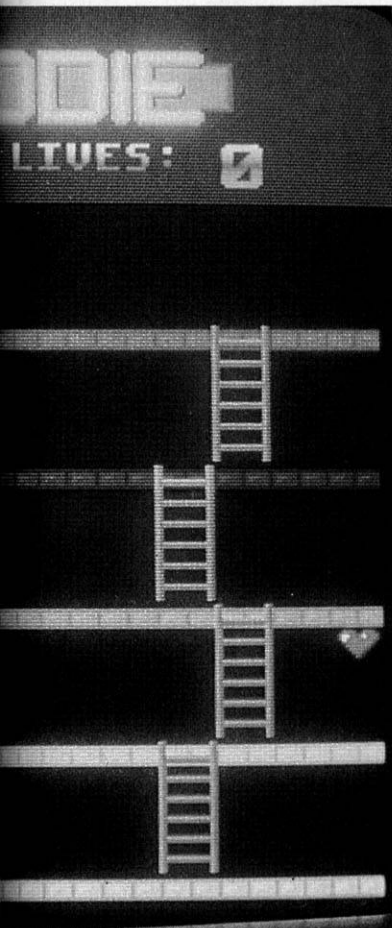
Sie übrigens durch Betätigen der Taste <m> ein- und ausschalten können – auch die Highscore-Tabelle präsentiert (da können Sie sich gleich ein Ziel für den nächsten Durchgang setzen). Nun liegt es an Ihnen, zu entscheiden, ob Sie durch Drücken des Joystick-Feuerknopfes in Level 0 beginnen wollen oder ob Sie durch Eingabe eines Codeworts in einen höheren Level einsteigen wollen bzw. dürfen. Diese Codeworte erhalten Sie auf zwei Arten: Entweder Sie schummeln und sehen im Source-Code nach (verwerflich, aber bequem) oder Sie verdienen sich die Kenntnis um diese Codes, indem Sie sich durch die Level durcharbeiten (mühevoll, aber unterhaltsam). Jeweils nach Beendigung der Level 10, 20, 30, 35, 40, 45 und 50 wird in der rechten unteren Bildecke eines der besagten Code-Worte eingeblendet. Wenn Sie sich

diese Codes merken, können Sie bei Programmstart durch die erwähnte Eingabe des jeweiligen Codes die einzelnen Level direkt ansteuern.

Eines dieser Code-Worte wollen wir Ihnen hier verraten: »QUIT«! Damit beenden Sie das Programm (gemein, was?). Dieses »QUIT« ist dann besonders wichtig, wenn Sie es geschafft haben, sich in der Bestenliste eintragen zu dürfen und diesen Erfolg der Nachwelt erhalten wollen – die Highscore-Abspeicherung auf Diskette erfolgt nur, wenn Sie das Programm mit diesem Codewort verlassen.

Irgendwo in der Mitte zwischen »spielen« und »Programm verlassen« liegt »Pause«; dankenswerterweise hat Michael Baumgaertner, der Programmator, eine Pause-Funktion eingebaut. Sollte sich bei Ihnen Nanosekunden vor Erreichen des fünfundvierzigsten Levels ein dringendes Be-

dürfnis einstellen, das Sie schon tapfer aber verzweifelt bis hierhin verschoben haben, so können wir Sie beruhigen: Drücken Sie die Taste <p> und der schnelle Fred und seine Kontrahenten warten erstarrt



unten als man denkt.

auf Ihre Rückkehr. Sind Sie wieder bereit und erleichtert, genügt ein Druck auf <SPACE> und das Gemetzel geht weiter.

Bevor wir zur Beschreibung des Quellcodes kommen, sollte eines zur allgemeinen Klarstellung gesagt sein: abzutippen gibt es nichts – oder kennen Sie jemanden, der bereit wäre, 30 (in Worten: dreißig) Seiten Source-Code abzutippen? Gemeint sind hier natürlich AMIGA-Sonderheft-Listing-Seiten (zweispaltig, klein gedruckt, usw.). Und das wäre nur das Hauptprogramm! Die Daten für die einzelnen Level und die Sound-Files kämen noch dazu. Also wir, die AMIGA-Redaktion, haben beschlossen: NEIN! Spielspaß mit »Fast Freddie« gibt's nur für Besitzer der Leserservice-Diskette – wir glauben, damit im Sinne unserer Leser eine vernünftige Entscheidung getroffen zu haben. Dafür bieten wir Ihnen auch je-

de Menge Files in der Schublade »FastFreddie« und in deren Sub-Directories.

Um spielen zu können, genügt es, die Files »Fast Freddie« und »HI« von der Leserservice-Diskette auf Ihre Spiele-Diskette umzukopieren; der Rest ist nur für Programmierer und Source-Code-Schnüffler wichtig. Bei dieser Gelegenheit noch eine Bemerkung zum Highscore-File »HI«: »Fast Freddie« prüft zwar, ob das File »HI« erreichbar – sprich im aktuellen Directory zu finden – ist, nimmt es Ihnen aber auch nicht übel, wenn es das File nicht entdecken kann und legt es in diesem Fall auch nicht neu an. Eine »pro forma«-Highscore-Tabelle ist nämlich im Programm integriert, und die wird dann auch angezeigt. Sie können sich in diese Liste sogar eintragen – aber der nächste Reset läßt Sie dann erleben, wie ver-gänglichlich Ruhm sein kann.

Listing - nein danke?

Sollten Sie, ganz gleich, aus welchem Grund, die Source-Files neu assemblieren wollen, beachten Sie bitte folgendes: geschrieben wurde das Programm für und mit dem »PRO-FORMAT« von Data-Becker. Sollten Sie noch eine Version dieses Assembler-Pakets besitzen, die den Befehl »even« nicht implementiert hat, so ersetzen Sie bitte sowohl im File »FastFreddie.src« als auch in »HI.src« die dort vorkommenden »even« durch »align.w«. Mit dem eingebauten Editor ist das ja kein großer Aufwand. Sie müssen dazu aber den für den Editor reservierten Speicherbereich vergrößern (200 000 Byte ist ein schöner, runder Wert). Beim Assemblieren stellen Sie bitte alle drei Auswahlfelder (SOURCE, DATA und BBS) auf »CHIP«. Ganz wichtig: das Hauptprogramm »FastFreddie.src« assemblieren Sie mit der Einstellung »normaler Code«; für die Generierung des Highscore-Files »HI« und der Hilfsdateien »Levels1« und »Levels2« müssen Sie bei der Assemblierung von »HI.src«, »Levels1.src« und »Levels2.src« unbedingt auf »absoluter Code« stellen! Für die Assemblierung des Hauptprogramms ist auch wichtig, daß Sie die externen Files in ihren Directories (»gfx«, »sprites« und »music«) belassen – die jeweiligen Pfadangaben sind im Hauptprogramm festgelegt.

»FastFreddie ist nicht multitaskingfähig – aber welches schnelle Action-Spiel ist das schon? Die Eingaberoutine für die Highscore-Eintragungen ist auch gerade keine Textverarbeitung (vertippen Sie sich bloß nicht!).

Ein Tip von Fred

Aber einen kleinen Trick, der in diesem Programm zur Anwendung kommt, wollen wir Ihnen doch etwas näherbringen. Wenn Sie erst ganz am Anfang Ihrer Assembler-Laufbahn auf dem AMIGA stehen, liegt der Verdacht nahe, daß Sie für das Löschen bestimmter Speicherbereiche den 68000er werken und in einer Schleife die Null-Bytes in den Speicher quetschen lassen – da bleibt für andere Dinge natürlich entsprechend weniger Zeit, und allzu schnell geht das auch nicht. Dabei drängt sich für solche Aktivitäten, also wenn es um großflächige Byte-Füllereien geht, einer der AMIGA-Co-Prozessoren geradezu auf: der sogenannte Blitter.

Der »Blitter« sitzt wie ein Untermieter gemeinsam mit dem »Copper« auf jenem Siliziumchip, dem sein Entwickler Jay Miner den Namen »AGNUS« gab (»Fat Agnus« und der neue, 1 MByte Chip-Memory verwaltende »Super Agnus« sind Weiterentwicklungen). Blitter steht für »BLOCK Image TransFER« (das zweite »T« ist während der Jahre, die dieses Miniatur-Maschinchen jetzt schon auf dem Buckel hat, sprachlich »dazugewachsen«). Aus dieser Bezeichnung läßt sich bereits erkennen, wofür dieser Chip-Teil in erster Linie entworfen wurde und wo daher naturgemäß seine Stärke liegt: den Inhalt eines Speicherblocks manipulieren und in einen anderen übertragen. Und das, weil hardwarecodiert, in blitzartiger Geschwindigkeit – ein gern zitiertes Beispiel für die Geschwindigkeit des Blitters ist das Zeichnen von Linien mit einer Million Punkten pro Sekunde.

Nun ist das Löschen eines Speicherbereichs eigentlich auch nur das Kopieren eines oder mehrerer Null-Bytes in den zu löschenden Bereich, und dazu ein Beispiel, wie man es »auch« machen kann:

a) als erste Aktion steht »keine« Aktion auf dem Programm. Es heißt warten, bis der Blitter mit der soeben laufenden Aktivität fertig ist.

- b) Blitter in den Kopier-Modus schalten und DMA-Kanal für Ziel »D« einschalten
- c) Anfangsadresse des zu löschenden Bereichs als Zieladresse deklarieren
- d) als »Quelle« wird »A« verwendet und mit dem gewünschten Wert, hier also »NULL«, beschrieben
- e) Nun den Modulo-Wert für Ziel D = 0
- f) Maskierung des Bereichs ausschalten
- g) Blitter-Sonder-Modi ausschalten
- h) Größe des gewünschten – zu löschenden – Bereichs angeben. Die Bereichsgröße ist als rechteckiges Feld anzugeben und ergibt sich aus »Größe = Höhe x 64 + Breite in Worten«.

Als Assembler-Quelltext liest sich das dann so:

```
wait:
BTST    #14,$DFF002 ;a)
BNE     wait

MOVE.w  #00000001
11110000,$DFF040 ;b)
MOVE.l  #Adresse,
$DFF054 ;c)
CLR.w   $DFF074 ;d)

CLR.w   $DFF066 ;e)
MOVE.w  #-1,
$DFF044 ;f)
MOVE.w  #-1,
$DFF046 ;f)
CLR.w   $DFF042 ;g)

MOVE.w  #Größe,
$DFF058 ;h)
```

RTS

Diese Routine ist besonders bei großen zu bearbeitenden Speicherfeldern wesentlich schneller als die oben genannte Schleife des Hauptprozessors. Überdies kann der 68000er weiterarbeiten, während der Blitter das Datenfeld löscht. Stark bemerkbar macht sich dieser Vorteil dann, wenn der Prozessor im FAST-Memory weiterarbeiten kann, weil er sich dann den Daten-Bus nicht mit dem Blitter zu teilen braucht.

Soweit zum Spezial-Tip von Michael Baumgaertner. Auch zu »Fast Freddie« wäre alles Wesentliche gesagt. Also dann: rauf auf die Leitern – und lassen Sie sich (und Freddie) nicht erwischen!

so

Hans Waldhäusel kam als Programmierer vom 128er zum Amiga. Sie erreichen ihn über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Herrn Waldhäusel, Hans-Pinsel-Straße 2, 8013 Haar).

Für diejenigen, die noch nicht wissen, was gemeint ist, haben wir das Bildschirmfoto eines der Spielfelder abgebildet (Bild 1). Jetzt hat's aber geklingelt! Im Gegensatz zum Vorbild bietet »SCHIEBESPIELCHEN« verschiedenste Arten von Spielfeldern. Arabische und römische Ziffern, Bildteile anstelle von Ziffern (fast schon ein Puzzle) und unterschiedliche Anzahlen von Schiebeplättchen stehen bereit, Ihnen beim Sortieren die Haare zu Berge stehen zu lassen.

Highscore-Speicherung und komfortable Mausbedienung

(warum hat der Programm-Autor sie wohl so genannt?), bei der es darauf ankommt, die durcheinandergebrachten Schiebeplättchen mit möglichst wenig Zügen in die richtige Reihenfolge zu bringen. Positionieren Sie den Mauszeiger über dem entsprechenden Feld und klicken Sie mit der linken Maustaste. Im darauf eingeblendeten Auswahl-Bild entscheiden Sie sich für eines der sechs vorgegebenen Spielfelder (Tabelle 1). »Elefant« und »Auto« sind puzzleähnlich aufgebaute Spielfelder, während die anderen vom Prinzip her dem »Original« gleichen, auch wenn der Redaktion kein Original mit 48 Feldern (Megabild) bekannt ist – kennen Sie eines?

Schie

**Sicher kennen Sie die Schiebereien
Seit den fünfziger Jahren hat
dieser Geduldsspiele nichts geändert
Hier ist endlich**

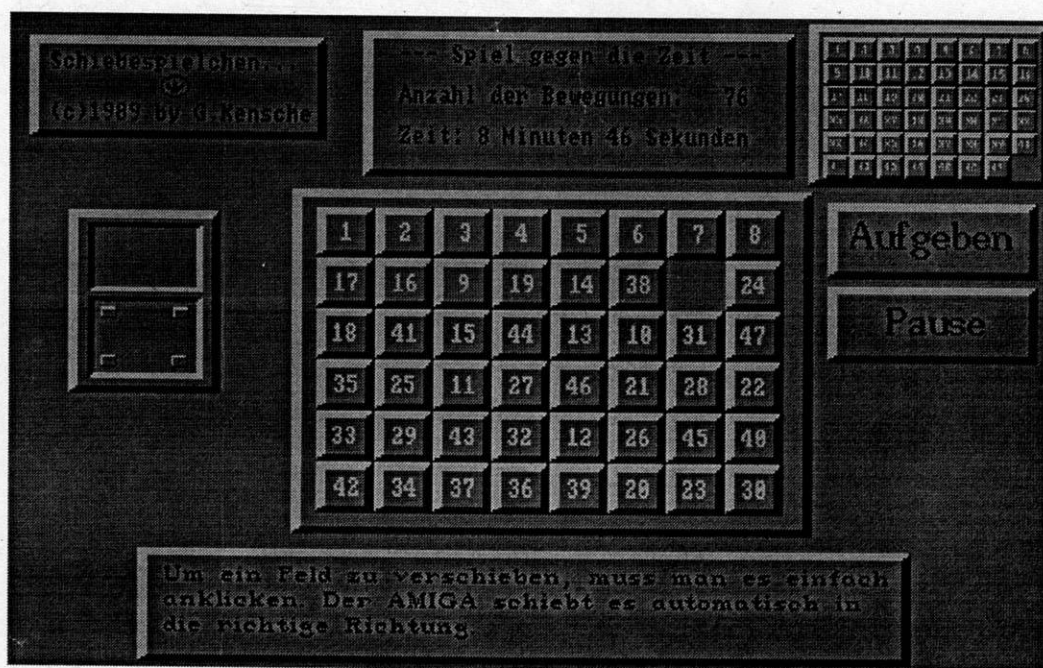


Bild 1. Das ist die Oberfläche von »SCHIEBESPIEL« für den AMIGA

sind selbstverständlich. Natürlich übernimmt der Computer die »Anfangs-Chaotisierung« des Spielfeldes. Und auch wenn es manchmal nicht so aussehen mag – jedes Spiel ist lösbar!

Starten Sie das Programm entweder durch Anklicken des Icons oder, wenn Sie Amiga-Basic schon geladen haben, durch Eingabe von

```
run "SCHIEBESPIELCHEN_
V1.4"
```

Das Spiel

Nach dem Start des Programmes wählen Sie einen der drei zur Verfügung stehenden Spielmodi.

Die vermutlich am meisten gespielte Variante ist »Normal«

Nach dem Anklicken des gewünschten Spielfeldes wird das entsprechende Bild von Diskette nachgeladen und vom Computer präpariert. Bei der Präparation können Sie zusehen – als mitfühlender Zeitgenosse hat Guido Kensch, der Programm-Autor, »nur« 400 bis 1000 Vertauschungen als Startvorgabe gewählt. Nach Abschluß der Vertauschungen werden Sie durch ein eingeblendetes Fenster aufgefordert, mittels Mausklick (links) den Wettstreit zu eröffnen. In der Abbildung sehen Sie im oberen Bildschirmbereich eingeblendet die verstrichene Zeit und die Anzahl der bisher getätigten Verschiebungen. In der rechten Bildhälfte sind zwei Gadgets angeordnet: »PAUSE« und »Aufgeben«. Im Falle, daß der Frust Sie über-

Amiga. Sobald Sie alle Felder in die richtige Reihenfolge gebracht haben, gelangen Sie automatisch in das Anfangsmenü. Sollten Sie es gar geschafft haben, einen der fünf Highscores zu knacken, dürfen Sie sich zuvor noch in der Bestenliste eintragen.

Highscore geknackt

Eine andere Art, an die Sache heranzugehen, ist Modus 2 – »Zeitspiel - 1 Spieler«. Die zu lösenden Bilder sind die gleichen, aber jetzt gilt es, eine Zeitvorgabe zu unterbieten. Den Zeitrahmen bestimmen Sie selbst, und zwar zwei, fünf oder zehn Minuten.

Als besonderen Anreiz können Sie auch gegen den Besten der jeweiligen Highscore-Liste antreten, dessen Zeit

Spielfelder

Standard 1	4 x 4 Felder
Standard 2	5 x 5 Felder
Megabild	6 x 8 Felder
Elefant	4 x 4 Felder
Auto	4 x 4 Felder
Klassik	6 x 5 Felder

Tabelle 1. Sechsfacher Spaß mit SCHIEBESPIEL

mannt und Sie »Aufgeben« anklicken, werden Sie ins Hauptmenü zurückgeschaltet. »Pause« ist nur zum Ausruhen gedacht, nicht aber zum Schummeln! »Pause« anklicken und dann bei angehaltener Stoppuhr in Ruhe nachdenken wollen gilt nicht – das Spielfeld wird abgedeckt (ätsch!). Verschieben Sie die Felder einfach durch Anklicken, den Rest macht Ihr

dann als Limit übernommen wird. Damit Sie sich nicht unabsichtlich übernehmen, wird die jeweilige Highscore-Liste vorher angezeigt. Während des Spieles läuft die eingeblendete Stoppuhr als Count-Down-Zähler.

Bleibt noch die dritte Spielvariante: »Zeitspiel - 2 Spieler«. Hier wird die Zeit, die der erste Spieler für die Lösung braucht,

die Zeit

bung

mit 15 Plättchen über 16 Felder.
sich an Prinzip und Aufbau
-Kunststoffplättchen und Ziffern.
die zeitgemäße Version.

gepatchte Version von Amiga-Basic in das Root-Directory (zum PAL-Patch lesen Sie weiter unten mehr);
f) laden und starten Sie das Programm »HIGHSCORE-ERSTELLEN«;
g) laden und starten Sie das Programm »SCHIEBESPIEL V1.4«;
h) spielen Sie und staunen Sie!

Die Arbeit

Die eigene Spiel-Diskette ist deshalb empfehlenswert, weil durch die Möglichkeit der Highscore-Speicherung der Schreibschutz der Diskette keinesfalls eingeschaltet sein darf. Das Programm prüft den Schreib-

dieses Bild die Auswahlmöglichkeiten. Welcher Bereich vom Programm beschrieben wird, entnehmen Sie bitte Bild 3 – der rot umrandete Bereich wird sowohl bei der Auswahl des Spielmodus als auch des Spielbildes für die Aus- und Eingabe verwendet. Die eingeblendeten Koordinaten sollen Ihnen die Orientierung erleichtern. Die Spielfelder selbst haben für alle Modi die gleiche Bildeinteilung. In Bild 2 haben wir die vom Programm bearbeiteten Bereiche eingerahmt und die Eckpunkte, soweit nötig, vermaßt. Alles, was Sie in den großen Mittelbereich zeichnen, wird von »SCHIEBESPIEL« automatisch zerschnippelt und verschoben. Die beiden kleinen Rechtecke rechts vom Spielfeld werden auf Mausklick abgefragt. Der Bildinhalt selbst ist für das Programm völlig irrelevant – Ihrer Fantasie und Ihrem Geschmack sind keine Grenzen gesetzt. Wichtig bei der ganzen Sache ist nur, daß Sie für jedes Bild genau den Namen verwenden, den das Programm und der »HIGHSCORE-MAKER« erwarten (siehe Tabelle 2), und daß Sie die Bilder vom IFF- in das ACBM-Format konvertieren.

Warum konvertieren? IFF ist zwar als Standard-Format für Bilder und Sounds allgemein anerkannt, läßt sich von Basic aus aber langsamer laden als ein im ACBM-Format vorliegendes File und braucht auch mehr Platz auf der Diskette. Die Laderoutine von »Schiebespielchen« verlangt nach ACBM-Dateien, und daher muß jedes von Ihnen gemalte Bild mit dem Basic-Programm »LoadILBM-SaveACBM« umgewandelt werden. Dieses Konvertierungsprogramm finden Sie auf der »EXTRAS«-Diskette im Verzeichnis »BASIC-DEMOS«.

Allerdings ist, ebenso wie bei Amiga-Basic, noch ein kleiner Patch notwendig – dazu später mehr. **Unbedingt** brauchen Sie das Bild »HAUPTMENU« und eines der Spielfelder.

Zur besseren Übersicht haben wir alle mit SCHIEBESPIELCHEN in Verbindung stehenden Files und Directories in Tabelle 2 zusammengefaßt. Es wird immer nur das im Hauptmenü angewählte Spielfeld geladen, so daß bei Nichtvorhandensein eines Bild-Files nicht mehr passiert, als daß Sie eine Fehlermeldung geschenkt bekommen.

Dies als kleiner Tip für alle, die nicht gleich »die volle Latte«

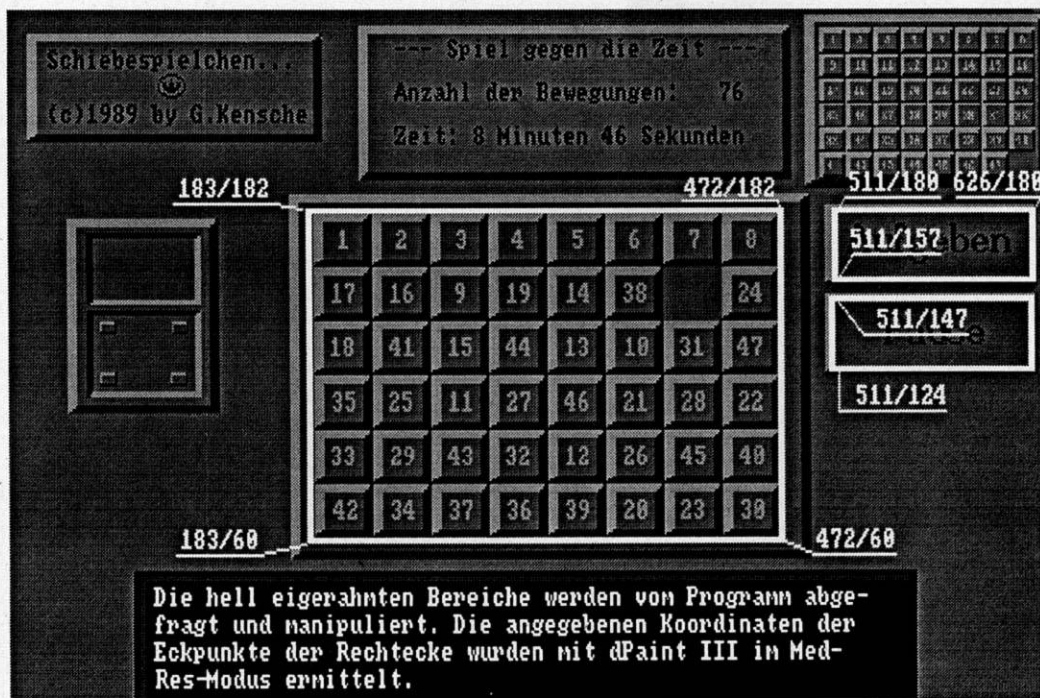


Bild 2. Die von SCHIEBESPIEL bearbeiteten Bild-Bereiche

zur Vorgabe für den zweiten, die dieser zu unterbieten hat.

Zwei Punkte im Hauptmenü wurden noch nicht erwähnt: »Highscore ansehen« und »QUIT« – Erläuterungen hierzu sind wohl überflüssig. Und wer tatsächlich eine Anleitung braucht, für den wird eine solche eingeblendet, wenn er im Hauptauswahl-Menü in das Titelfeld klickt.

Um in den Genuß des Spielvergnügens mit SCHIEBESPIELCHEN zu kommen, gibt es für Sie zwei Möglichkeiten: tippen oder kaufen. Falls Sie sich für »kaufen« entschieden haben und jetzt stolzer Besitzer der Leserservice-Diskette zu diesem Sonderheft sind, gehen Sie bitte folgendermaßen vor:

a) formatieren Sie eine leere Diskette;

b) kopieren Sie soweit als notwendig Directories und Files von Ihrer Workbench-Diskette (je nachdem, ob Sie von Ihrer SCHIEBESPIELCHEN-Diskette auch booten wollen oder nicht);

c) in das Directory »libs« kopieren Sie zusätzlich die »bmap«-Dateien »dos.bmap«, »exec.bmap«, »graphics.bmap« und »intuition.bmap«. Auch die »icon.library« muß vorhanden sein.

d) kopieren Sie den Inhalt (!) der Schublade »SCHIEBESPIELCHEN« von der Leserservice-Diskette auf Ihre Spieldiskette (auch die Sub-Directories!);

e) kopieren Sie eine auf PAL

schutz-Status und verweigert Ihnen den Spielspaß, wenn der Schreibschutz aktiviert ist!

Sie wollen (oder können) die Leserservice-Diskette nicht kaufen? Dann steht Ihnen jede Menge Arbeit bevor. Das Hauptprogramm (Listing 1) und den Highscore-Maker (Listing 2) müssen Sie auf jeden Fall abtippen. Die Spielfelder müssen Sie sich selbst zeichnen – oder wollten Sie eine DATA-Wüste vorgesetzt bekommen? – Na eben. Als Anhaltspunkt und kleine Hilfe für die Bildgestaltung haben wir zwei Bilder abgedruckt (Bild 2 und Bild 3), die bereits auf die richtigen Maße »zurechtgestutzt« sind. »Schiebespiel« lädt zu Beginn das Bild »HAUPTMENU« und schreibt in

zeichnen und malen wollen. Für das komplette Spiel sind folgende Dateien zusätzlich zu »HAUPTMENU« zu malen und mit **exakt dem angegebenen Namen** zu speichern: »AUTO«, »KLASSIK«, »ELEFANT« – das soll ein »Ottifant« sein – Otto möge uns verzeihen, »STANDARD«, »STANDARD II« und »MEGABILD«.

Die Filenamen werden sowohl vom Hauptprogramm als auch vom Highscore-Maker verwendet – bitte keine leichtfertigen Experimente!

Nach dem Abtippen der Listings 1 und 2 verfahren Sie ebenso wie die Besitzer der Programmservice-Diskette von a) bis e).

Anstelle des Kopiervorganges in Punkt d) legen Sie ein Di-

es der amerikanischen NTSC-Norm entspricht. Im AMIGA-Sonderheft 1 wurde ein kleines Patchprogramm veröffentlicht, mit dem dieses Manko ausgeglichen werden kann. SCHIEBESPIELCHEN braucht diese gepatchte Version. Für diejenigen, die weder das Sonderheft 1 noch eine bereits modifizierte Version von Amiga-Basic besitzen, hier noch einmal die Vorgangsweise (die Patches stammen übrigens von Peter Pörner und Alexander Wagner).

Patches

Das in Listing 3 abgedruckte Basic-Programm »HöhenPatch« nimmt Ihnen die Arbeit ab. In der vorliegenden Version ist es nur auf jene Basic-Versionen

Files und Directories

Directory	File	Listing #	Bemerkung
Root	SCHIEBESPIELCHEN_V1.4	Listing 1	
Root	HIGHSCORE-ERSTELLEN	Listing 2	**
libs	dos.bmap	*	
libs	exec.bmap	*	
libs	graphics.bmap	*	
libs	intuition.bmap	*	
libs	icon.library	*	
BILDER	HAUPTMENU		
BILDER	AUTO		
BILDER	KLASSIK		
BILDER	ELEFANT		
BILDER	STANDARD		
BILDER	STANDARD II		
BILDER	MEGABILD		
Root	AmigaBasic	*	
Root	HöhenPatch	Listing 3	**
IFF	MEGABILD.pic	**	

Tabelle 2. Dateien für SCHIEBESPIEL. Die mit »*« bezeichneten Dateien sind nicht auf der Leserservice-Diskette enthalten. Dateien mit der Kennung »« werden beim Spielen nicht benötigt.**

rectory »BILDER« an und speichern die konvertierten Bilder in diesem Sub-Directory. Die Files »Schiebespiel V1.4« aus Listing 1 und »HIGHSCORE-ERSTELLEN« aus Listing 2 legen Sie im Stammverzeichnis ab, ebenso die gepatchte Amiga-Basic-Version. Laden Sie Amiga-Basic und starten Sie das Programm »HIGHSCORE-ERSTELLEN«. Der Highscore-Maker legt im Sub-Directory »HIGHSCORE« die notwendigen Files an.

Das Ausgabefenster von Amiga-Basic ist in keiner der bis dato serienmäßig ausgelieferten Versionen in der Lage, den vollen PAL-Bildschirm auszunützen – anstelle der zur Verfügung stehenden 256 vertikalen Bildschirmpunkte können nur 200 angesprochen werden wie

anwendbar, die 103484 Byte oder 103500 Byte lang sind. Kontrollieren Sie bitte vorsichtshalber Ihre Version auf die richtige File-Länge mit dem LIST-Befehl und verwenden Sie für die Modifikation **keinesfalls das Original**, sondern eine Arbeitskopie:

```
list AmigaBasic
```

Wenn die oben genannten File-Längen mit Ihrer Amiga-Basic-Version übereinstimmen (und Sie Listing 3 abgetippt haben), können Sie loslegen. Laden und starten Sie Amiga-Basic und das Programm »HöhenPatch«. Nach Ende des Programmlaufes kontrollieren Sie wieder mit dem »list«-Befehl, ob die Filelänge noch die gleiche wie im Original ist (sicher ist si-

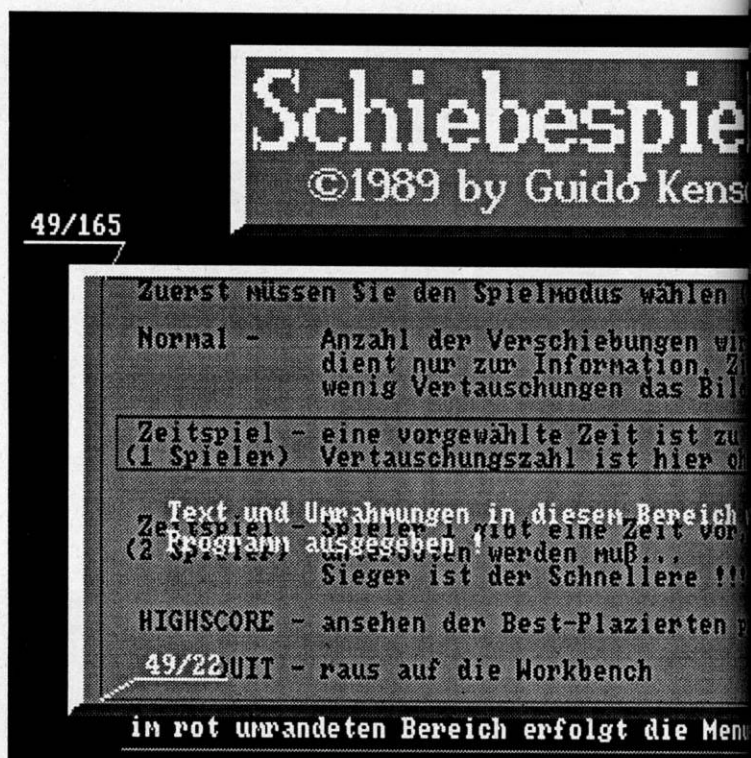


Bild 3. Für die Abfrage während der Spiel-Auswahl wird

cher). Wenn ja, haben Sie jetzt genau die BASIC-Version, die SCHIEBESPIELCHEN braucht. Wenn nicht, haben Sie irgendwo Blödsinn gemacht – fröhliches Suchen.

Es gibt aber noch eine veröffentlichte Amiga-Basic-Version (V1.0), deren File-Länge nicht mit den obengenannten Byte-Zahlen übereinstimmt. Sollte Ihr Amiga-Basic-File eine Länge von 93156 Byte haben, können Sie mit zwei kleinen Modifikationen das Patchprogramm aus Listing 3 verwenden: Die Zahl »&H57ec« muß durch »&H55e8« ersetzt werden, die Zahl »&H5c00« ist durch »&H2800« zu ersetzen.

Soweit zu Amiga-Basic. Das Konvertierungsprogramm »LoadILBM-SaveACBM« von der »EXTRAS«-Diskette muß allerdings auch noch an die volle PAL-Höhe angeglichen werden. Dazu laden Sie Amiga-Basic und das Konvertierungsprogramm und ersetzen in Zeile 279 (Leerzeilen mitgezählt!)

```
IF scrHeight% > 200
then kk = kk+2
```

die Zahl 200 durch die Zahl 256, so daß es jetzt heißen muß:

```
IF scrHeight% > 256
then kk = kk+2
```

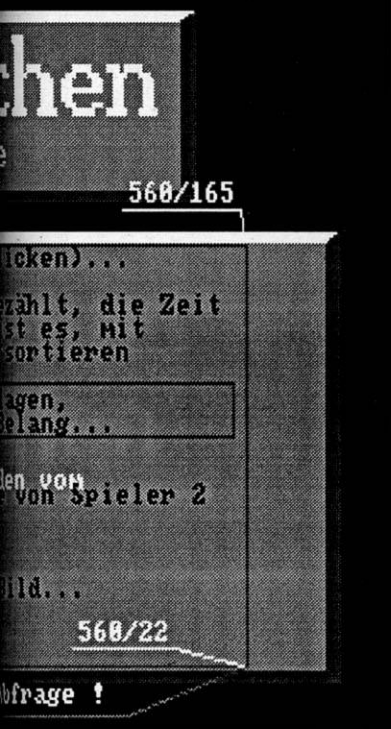
Jetzt ist auch der Konverter fit für unser Spiel.

SCHIEBESPIELCHEN ist eine überzeugende Demonstration dafür, was aus dem als

langsam verrufenen Amiga-Basic herauszuholen ist, wenn man die Betriebssystem-Routinen konsequent einsetzt. Die Verschiebungen, die der Computer zu Beginn jedes Spieles vornimmt, finden tatsächlich grafisch auf dem Bildschirm statt, auch wenn es wegen der enormen Geschwindigkeit wie Geflimmer aussieht! Aber es stecken noch andere erwähnenswerte Besonderheiten in diesem Programm – einige davon seien hier stellvertretend für alle kurz angerissen:

Feinheiten

Amiga-Basic lädt bekanntlich Bilder, die im ACBM-Format vorliegen, schneller als solche im IFF-ILBM-Format. Auf der mit Ihrem Rechner mitgelieferten Extras-Diskette finden Sie ein Basic-Programm zum Laden von ACBM-Files, genannt »LoadACBM«. Dieses Programm ist knapp 200 Zeilen lang und braucht für das Laden des Files »STANDARD« etwas mehr als sechs Sekunden. Die in diesem Programm verwendete Laderoutine ist auf die Anwendung zurechtgestutzt, mit weniger als 90 Programmzeilen deutlich kürzer und überdies etwas schneller (Ladezeit für »STANDARD« unter fünf Sekunden). Wenn Sie in Ihren Programmen nicht alle Fähigkeiten von »LoadACBM« brauchen – abkupfern erlaubt.



der rot umrandete Teil verwendet

Professionell gemacht ist auch der »Custom-Guru«, der immer dann eingeblendet wird, wenn ein Fehler auftritt, den das Programm nicht ausgleichen kann (z.B., wenn die Diskette schreibgeschützt ist). Sie

finden die Subroutine, die die Betriebssystem-Routine »DisplayAlert« einsetzt, zwar auch am Ende des Programmes unter dem Label »Alarm«, der besseren Übersicht halber haben wir den Block noch zusätzlich herausgelöst, um Ihnen das Einbinden in Ihre Programme zu erleichtern:

```
SUB Alarm(anz%,T$( ),
Height%) STATIC
anz%=anz%-1 : type%=
0 : A1$=""
FOR I%=0 TO anz%
Y%(I%)=I%*12+15
X%=320-4*LEN(T$(I%))
A1$=A1$+MKI$(X%)+CHR$(Y%
(I%))+T$(I%)+CHR$(0)
IF I%anz% THEN A1$=A1$+
CHR$(1) ELSE A1$=A1$+
CHR$(0)
NEXT I%
CALL DisplayAlert(type%,
SADD(A1$),Height%)
END SUB
```

Schreibschutzfehler:
'wenn Diskette nicht
Write-enabled ist
Mess\$(0)="Fehler
aufgetreten..."
Mess\$(1)="Die Diskette
muß Write-enabled
(Schreibschutz auf)
sein,"
Mess\$(2)="da der

Highscore gespeichert
wird !!!"

Alarm 3,Mess\$(),50

Es handelt sich hier um einen sogenannten »Recoverable Alert«, also um eine Alarm-Meldung, die nicht zwangsläufig zum Programmabbruch führt, sondern bei der nach Drücken der Maus-Taste eine Fortsetzung des aufrufenden Programms möglich ist. Der Einsprungpunkt ist das Label »Schreibschutzfehler«, das der Einfachheit halber aus dem »ON ERROR GOTO«-Kommando heraus angewählt wird. Da das Betriebssystem bei Aufruf einer Alert-Meldung den auszugebenden Text in einem Array erwartet, wird zuerst der gewünschte Text dem Array »Mess\$« zugewiesen. Unmittelbar danach erfolgt der Aufruf des Unterprogramms »Alarm«. Als Parameter werden übergeben: »3« als Anzahl der auszugebenden Text-Array-Felder (0 bis 2), »Mess\$(0)« als Information für das Betriebssystem, welches Array zu verwenden ist, und zu guter Letzt noch »50« als Höhe der Alert-Box in Pixel. Diese Routine ist übrigens eine »Untermenge« einer in AMIGA-Sonderheft 3 vorgestellten, dort aber allgemeiner gehaltenen Alert-Routine, die im Rahmen des Basic-Kurses »Amiga-Basic

im Höhenflug« veröffentlicht wurde (was, Sie haben dieses Sonderheft nicht? Ein schwerer Fehler, aber keine Angst, es kann noch nachbestellt werden).

Für alle, denen die vorliegenden Spielfelder nicht ausreichen, haben wir auf der Leserservice-Diskette im Directory »IFF« das File »MEGABILD.pic« abgelegt. So brauchen Sie sich nicht mit Konvertierungen abzuquälen, wenn Sie Ihr eigenes Spielfeld in SCHIEBESPIELCHEN integrieren wollen (geht übrigens ganz einfach - aber denken Sie an den HIGHSCORE-MAKER).

Erweiterungen

Eine nette Sache wäre, besonders für Anfänger, eine einstellbare Anzahl von Vertauschungen zu Beginn des Spiels. Anbieten würde sich das Feld links vom Verschiebereich, das noch nicht genutzt wird. Es würde genügen, die DATA-Werte für die Anzahl der Vertauschungen im Block »*** Bilderdaten *« zu manipulieren, bzw. den Befehl »READ Vertausch« durch das Einlesen einer Variablen zu ersetzen.

Das war's - wir wünschen Ihnen stundenlanges Vergnügen!

so

```
' Programm: Schiebepiel (c)1989 Guido Kenschke
' letzte Änderung: 6.7.89

WINDOW 1,"Schiebepielchen - 1989 Guido Kenschke...",(0,0)-(631,242),0

CLEAR ,40000&

ON BREAK GOSUB Keine.Unterbrechung
BREAK OFF ' <- hier muß später BREAK ON hin
ON ERROR GOTO Schreibschutzfehler

' *** Deklarationsteil, Eröffnung der Bibliotheken ***

DIM Bitplane%(5), cTabWork%(32), cTabSave%(32)

DECLARE FUNCTION xOpen& LIBRARY
DECLARE FUNCTION xRead& LIBRARY
DECLARE FUNCTION xWrite& LIBRARY
DECLARE FUNCTION AllocMem&() LIBRARY

LIBRARY ":libs/dos.library"
LIBRARY ":libs/exec.library"
LIBRARY ":libs/graphics.library"
LIBRARY ":libs/intuition.library"

DEFINT Min,Sek,X,X2,X3,Y,Y2,Y3,XPixel,Vertausch,Xmove,XMovanz,
Zufall,I
DIM Wert(8,6), ZFeld(3204)
RANDOMIZE TIMER

' *** testet, ob Schreibschutz offen ist, für Highscore ***
Anfang:
OPEN ":Test" FOR OUTPUT AS #1
CLOSE

' *** Hauptmenue, Auswahl des Spielmodus ***
```

```
Hauptmenue.eins: ' Ansprung, wenn Aufgegeben angeklickt wurde
Lade.Bild "Bilder/HAUPTMENÜ",Lesefehler$
IF Lesefehler$<>" " THEN Fehler.in.Bilddatei
```

```
Hauptmenue:
COLOR 1,3 : COLOR 0,2 : Hilf=MOUSE(0)
LOCATE 11,9 : Prnt "Zuerst müssen Sie den Spielmodus wählen
(anklicken)..."
LOCATE 13,9 : Prnt "Normal - Anzahl der Verschiebungen wird
gezählt, die Zeit"
LOCATE 14,21: Prnt "dient nur zur Information. Ziel ist es, mit"
LOCATE 15,21: Prnt "wenig Vertauschungen das Bild zu sortieren"
LOCATE 17,9 : Prnt "Zeitspiel -
eine vorgewählte Zeit ist zu schlagen,"
LOCATE 18,8 : Prnt "(1 Spieler) Vertauschungszahl ist hier ohne
Belang..."
LOCATE 21,9 : Prnt "Zeitspiel - Spieler 1 gibt eine Zeit vor, die
von Spieler 2"
LOCATE 22,8 : Prnt "(2 Spieler) unterboten werden muß..."
LOCATE 23,21: Prnt "Sieger ist der Schnellere !!!"
LOCATE 25,9 : Prnt "HIGHSCORE - ansehen der Best-Plazierten pro
Bild..."
LOCATE 27,14: Prnt "QUIT - raus auf die Workbench"
```

```
WHILE MOUSE(0)<1 OR (Spielmodus<0 AND Spielmodus<>-2) OR
Spielmodus>4
Y=PEEKW(WINDOW(7)+12)
Spielmodus=-INT((Y-104)/32)*(Y<213)-(Y>212)*4
IF Spielmodus<>X AND X>-1 AND X<5 THEN
LINE (52,X*32+94+(X=4)*16)-(550,X*32+120+(X=3)*16+(X=1)*8+
(X=4)*32),2,b
```

Listing 1. Das Hauptprogramm
SCHIEBESPIELCHEN_V1.4.


```

END IF
IF Spielmodus>-1 AND Spielmodus<5 THEN
  X2=(Spielmodus=3)*16+(Spielmodus=1)*8+(Spielmodus=4)*32
  LINE (52,Spielmodus*32+94+(Spielmodus=4)*16)-(550,Spielmodus*32+
  120+X2),3,b
END IF
X=Spielmodus
WEND
IF Spielmodus=-2 THEN Copyright
IF Spielmodus=3 THEN Highscores.ansehen
IF Spielmodus=4 THEN SYSTEM
GOSUB Text.loeschen

' *** Auswahl der Spieloberfläche ***

LOCATE 12,9 : Hilf=MOUSE(0)
Prnt "Bitte wählen Sie die Spieloberfläche, indem Sie sie anklicken"
Bildauswahl:
LOCATE 14,20 : Prnt "Name des Bildes" Größe"
LOCATE 16,20 : Prnt "- Standard 1" 4 x 4"
LOCATE 18,20 : Prnt "- Standard 2" 5 x 5"
LOCATE 20,20 : Prnt "- Megabild" 6 x 8"
LOCATE 22,20 : Prnt "- Elefant" 4 x 4"
LOCATE 24,20 : Prnt "- Auto (Auburn Speedster)" 4 x 4"
LOCATE 26,20 : Prnt "- Klassik (römische Zahlen)" 6 x 5"

WHILE MOUSE(0)<1 OR Bild<0 OR Bild>6
  Bild=INT((PEEKW(WINDOW(7)+12)-128)/16)
  IF Bild<>X AND X>-1 AND X<6 THEN
    LINE (148,X*16+118)-(422,X*16+128),2,b
  END IF
  IF Bild>-1 AND Bild<6 THEN
    LINE (148,Bild*16+118)-(422,Bild*16+128),3,b
  END IF
  X=Bild
WEND
GOSUB Text.loeschen

RESTORE
FOR I=0 TO Bild
  READ Bild$ 'Name des Bilds im Verzeichnis Bilder
  READ Hoehe 'Anzahl der Felder in Y-Richtung
  READ Breite 'Anzahl der Felder in X-Richtung
  READ XPixel 'Pixelbreite der Felder
  READ Start 'X-Startkoordinate
  READ Vertausch 'Anzahl der Vertauschungen
  READ XMovanz 'Anzahl der Schritte fuer Softscroll
  READ Xmove 'Pixelanzahl fuer Softscroll
NEXT I

IF Spielmodus=3 THEN RETURN
Spieler=1

' *** Bilderdaten ***
DATA Standard , 4, 4, 72, 110, 400, 9, 8
DATA Standard II, 5, 5, 56, 125, 500, 7, 8
DATA Megabild , 6, 8, 36, 146, 1000, 6, 6
DATA Elefant , 4, 4, 72, 110, 500, 9, 8
DATA Auto , 4, 4, 72, 110, 500, 9, 8
DATA Klassik , 4, 6, 48, 134, 500, 6, 8

' *** Ermittlung der Vorgabezeit in Zeitspielmodus ***

IF Spielmodus=1 THEN
  OPEN "R",1,":Highscore/"+Bild$+".Zeit",27
  FIELD #1,20 AS Spieler$,2 AS Versch$,5 AS Zeit$
  Hilf=MOUSE(0)
  LOCATE 12,20 : Prnt "Welche Vorgabezeit wollen sie schlagen ???"
  LOCATE 14,25 : Prnt "2 Minuten (nur für Profis)"
  LOCATE 16,25 : Prnt "5 Minuten (erfahrene Spieler)"
  LOCATE 18,25 : Prnt "10 Minuten (totaler Anfänger)"
  LOCATE 20,16 : Prnt "Highscore (Kampf gegen den Besten aus dem
  Highscore)"
  LOCATE 22,16 : Prnt STRING$(17,"-")>ZEIT-HIGHSCORES<"+STRING$
  (18,"-")
  Y=22 : GOSUB Highscore.Liste

  WHILE MOUSE(0)<1 OR Vorg.Zeit<0 OR Vorg.Zeit>4
    Vorg.Zeit=INT((PEEKW(WINDOW(7)+12)-112)/16)
    IF Vorg.Zeit<>X AND X>-1 AND X<4 THEN
      LINE (118-(X<3)*72,X*16+102)-(542+112*(X<3),X*16+112),2,b

```

```

END IF
IF Vorg.Zeit>-1 AND Vorg.Zeit<4 THEN
  LINE (118-(Vorg.Zeit<3)*72,Vorg.Zeit*16+102)-(542+112*(Vorg.Zeit
  <3),Vorg.Zeit*16+112),3,b
END IF
X=Vorg.Zeit
WEND
GOSUB Text.loeschen

Vorg.Zeit=((Vorg.Zeit=0)*120+(Vorg.Zeit=1)*300+(Vorg.Zeit=2)*600)
IF Vorg.Zeit=0 THEN Vorg.Zeit=VAL(LEFT$(Zeit$,2))*60+VAL(RIGHT$
(Zeit$,2))
CLOSE
END IF

' *** laden des Bilds und durchfuehren der Vertauschungen ***
FOR I=255 TO 1 STEP -1
  LINE (1,I)-(640,256-I),0
NEXT I
FOR I=0 TO 640
  LINE (I,0)-(640-I,256),0
NEXT I

Spielbeginn:
Lade.Bild "Bilder/"+Bild$,Lesefehler$
IF Lesefehler$ <> "" THEN Fehler.in.Bilddatei

LOCATE 1,30 : TIMER OFF
IF Spielmodus=0 THEN
  Prnt " --- Normales Spiel ---"
  Zeit=0 : ON TIMER(1) GOSUB Zeitzaeher.auf
ELSEIF Spielmodus=1 THEN
  Prnt "---- Spiel gegen die Zeit ----"
  Zeit=Vorg.Zeit : ON TIMER(1) GOSUB Zeitzaeher.ab
ELSEIF Spieler=1 THEN
  Prnt "Spieler 1: Zeit vorgeben..."
  Zeit=0 : ON TIMER(1) GOSUB Zeitzaeher.auf
ELSE
  Prnt "Spieler 2: Zeit unterbieten"
  ON TIMER(1) GOSUB Zeitzaeher.ab
  Vorg.Zeit=Zeit+1
END IF
Min=INT(Zeit/60) : Sek=Zeit MOD 60
LOCATE 3,30 : Prnt "Anzahl der Bewegungen: 0"
LOCATE 5,30 : Prnt "Zeit: "+STR$(Min)+" Minuten"+STR$(Sek)+"
Sekunden "

Zaeher=0
FOR I=1 TO Hoehe
  FOR T=1 TO Breite
    Zaeher=Zaeher+1
    Wert(T,I)=Zaeher
  NEXT T
NEXT I

X=Breite : Y=Hoehe ' Koordinaten des Leerfelds

' *** Routine zum Vertauschen der Felder ***

FOR I=1 TO Vertausch
  Zufall=INT(RND*4)+1

  IF Zufall=1 AND Y>1 THEN
    SWAP Wert(X,Y),Wert(X,Y-1) : Y=Y-1
    SCROLL (X*XPixel+Start,Y*20+44)-
    ((X+1)*XPixel+Start-1,Y*20+83),0,20

  ELSEIF Zufall=2 AND Y<Hoehe THEN
    SWAP Wert(X,Y),Wert(X,Y+1) : Y=Y+1
    SCROLL (X*XPixel+Start,Y*20+24)-
    ((X+1)*XPixel+Start-1,Y*20+63),0,-20

  ELSEIF Zufall=3 AND X>1 THEN
    SWAP Wert(X,Y),Wert(X-1,Y) : X=X-1
    SCROLL (X*XPixel+Start,Y*20+44)-
    ((X+2)*XPixel+Start-1,Y*20+63),XPixel,0

  ELSEIF Zufall=4 AND X<Breite THEN
    SWAP Wert(X,Y),Wert(X+1,Y) : X=X+1

```



```

    SCROLL ((X-1)*XPixel+Start,Y*20+44)-((X+1)*XPixel+Start-1,
    Y*20+63),-XPixel,0
END IF

NEXT I

' *** Spielroutine, bewegen der Felder durch Mausklick ***

V=0      ' Anzahl der Bewegungen
Zeitaus=0

GET (250,80)-(450,120),ZFeld
LINE (258,88)-(450,120),1,bf
LINE (250,80)-(442,112),0,bf
LINE (250,80)-(442,112),1,b
LOCATE 12,34 : Prnt "---> Spielbeginn <---"
LOCATE 13,34 : Prnt "Start mit Mausklick"
GOSUB Warte.auf.Maus
LINE (250,80)-(450,120),0,bf
PUT (250,80),ZFeld
TIMER ON

Bewege.Felder:

WHILE MOUSE(0)<1
    IF Zeitaus THEN Spiel.beendet
    LOCATE 5,30 : Prnt "Zeit:"+STR$(Min)+" Minuten"+STR$(Sek)+"
        Sekunden "
WEND

X2=X*XPixel+Start : Y2=Y*20+44
X3=MOUSE(1) : Y3=MOUSE(2)
Fertig=1 : Zaehler=0

IF X3>500 THEN
    IF Y3>65 AND Y3<90 THEN Hauptmenue.eins
    IF Y3>100 AND Y3<120 THEN
        TIMER STOP
        GET (250,80)-(450,120),ZFeld
        LINE (258,88)-(450,120),1,bf
        LINE (250,80)-(442,112),0,bf
        LINE (250,80)-(442,112),1,b
        LOCATE 12,34 : Prnt " ---> Pause <---"
        LOCATE 13,34 : Prnt "Weiter mit Mausklick"
        GOSUB Warte.auf.Maus
        LINE (250,80)-(450,120),0,bf
        PUT (250,80),ZFeld
        TIMER ON
        GOTO Bewege.Felder
    END IF
END IF

IF X3>X2 AND X3<X2+XPixel THEN Xrichtung=0 ELSE Xrichtung=
SGN(X3-X2)
IF Y3>Y2 AND Y3<Y2+20 THEN Yrichtung=0 ELSE Yrichtung=SGN(Y3-Y2)

IF Xrichtung=1 AND Yrichtung=1 AND X<Breite THEN
    SWAP Wert(X,Y),Wert(X+1,Y) : X=X+1 : V=V+1
    FOR I=1 TO XMovanz
        SCROLL ((X-1)*XPixel+Start,Y*20+44)-((X+1)*XPixel+Start-1,
        Y*20+63),-Xmove,0
    NEXT I

ELSEIF Xrichtung=0 AND Yrichtung=-1 AND Y>1 THEN
    SWAP Wert(X,Y),Wert(X,Y-1) : Y=Y-1 : V=V+1
    FOR I=1 TO 20 STEP 2
        SCROLL (X*XPixel+Start,Y*20+44)-((X+1)*XPixel+Start-1,Y*20+83),0,2
    NEXT I

ELSEIF Xrichtung=0 AND Yrichtung=1 AND Y<Hoehe THEN
    SWAP Wert(X,Y),Wert(X,Y+1) : Y=Y+1 : V=V+1
    FOR I=1 TO 20 STEP 2
        SCROLL (X*XPixel+Start,Y*20+24)-((X+1)*XPixel+Start-1,Y*20+63),0,-2
    NEXT I

ELSEIF Xrichtung=-1 AND Yrichtung=1 AND X>1 THEN
    SWAP Wert(X,Y),Wert(X-1,Y) : X=X-1 : V=V+1
    FOR I=1 TO XMovanz
        SCROLL (X*XPixel+Start,Y*20+44)-
        ((X+2)*XPixel+Start-1,Y*20+63),Xmove,0
    NEXT I

```

```

END IF

FOR I=1 TO Hoehe
    FOR XT=1 TO Breite
        Zaehler=Zaehler+1
        IF Wert(XT,I)<>Zaehler THEN I=Hoehe : XT=Breite : Fertig=0
    NEXT XT
NEXT I

LOCATE 3,54 : Prnt STR$(V)
IF Fertig=1 THEN TIMER STOP : GOTO Spiel.beendet

GOTO Bewege.Felder

' *** Alle Felder richtig... Statistik und Highscore ***

Spiel.beendet:

Lade.Bild "Bilder/HAUPTMENÜ",Lesefehler$
IF Lesefehler$ <> "" THEN Fehler.in.Bilddatei
Ladezahl=Ladezahl+1
Pufferinhalt=xRead$(FileHilf$,Ein.Puffer$,IcLaen&)
Tiefe%=PEEK(Ein.Puffer$+8)

Ben.Speicher=20480*(Tiefe%+1)+5000
IF FRE(-1)<Ben.Speicher THEN Lesefehler$="Zuwenig Speicher":
GOTO Raeum.auf

SCREEN 2,640,256,Tiefe%,2
WINDOW 2,"Schiebespielchen",,0,2
Fenster.Flag(67684&)

sSCREEN% =PEEK(WINDOW(7)+46)
VPort% =sSCREEN%+44
sBitMap% =PEEK(sSCREEN%+88)
ScrnTiefe%=PEEK(sBitMap%+5)
Farbzahl% =2^ScrnTiefe%

FOR Schleife=0 TO ScrnTiefe% - 1
    Bitplane$(Schleife)=PEEK(sBitMap%+8+(Schleife*4))
NEXT

CALL LoadRGB4$(VPort%,Tab.C&,Farbzahl%)

ELSEIF Ein.Text$="CMAP" THEN
    Ladezahl=Ladezahl+1
    Pufferinhalt=xRead$(FileHilf$,Puffer.C&,IcLaen&)

    FOR Schleife=0 TO Farbzahl% - 1
        rot%=PEEK(Puffer.C&+(Schleife*3))
        gru%=PEEK(Puffer.C&+(Schleife*3)+1)
        bla%=PEEK(Puffer.C&+(Schleife*3)+2)
        POKEW(Tab.C&+(2*Schleife)),(rot%*16)+(gru%)+(bla%/16)
    NEXT

ELSEIF Ein.Text$="CAMG" OR Ein.Text$="CCRT" THEN
    Ladezahl=Ladezahl+1
    Pufferinhalt=xRead$(FileHilf$,Ein.Puffer$,IcLaen&)

ELSEIF Ein.Text$="ABIT" THEN
    Ladezahl=Ladezahl+1
    FOR P=0 TO Tiefe%-1
        Pufferinhalt=xRead$(FileHilf$,Bitplane$(P),20480)
    NEXT

ELSE
    FOR Schleife=1 TO IcLaen&
        Pufferinhalt=xRead$(FileHilf$,Ein.Puffer$,1)
    NEXT
    IF (IcLaen& OR 1)=IcLaen& THEN
        Pufferinhalt=xRead$(FileHilf$,Ein.Puffer$,1)
    END IF

END IF

END IF

IF Ladezahl=5 THEN GOTO Laden.erfolgreich
IF Pufferinhalt&>0 THEN GOTO Lese.Schleife
IF Pufferinhalt&<0 THEN Lesefehler$="Lesefehler." : GOTO Raeum.auf
IF Ladezahl<5 THEN Lesefehler$="Datei kaputt..." : GOTO Raeum.auf

```

Listing 1. (Fortsetzung)


```
Laden.erfolgreich:
Lesefehler$=""
CALL LoadRGB4&(VPort&,Tab.C&,Farbzahl%)

Raum.auf:
IF FileHilf$ <> 0 THEN CALL xClose&(FileHilf&)
IF Eig.Puffer$ <> 0 THEN CALL FreeMem&(Eig.Puffer&,Mein.Puffer&)

END SUB

SUB Fenster.Flag(flag&) STATIC ' setzt Flags für das Fenster
wi&=WINDOW(7)
POKEL (wi&+24),PEEKL(wi&+24) XOR flag&
CALL RefreshWindowFrame(wi&)
END SUB

SUB Prnt(A$) STATIC ' schnellere Textausgabe als mit PRINT
CALL Text(WINDOW(8),SADD(A$),LEN(A$))
PRINT
END SUB

SUB Alarm(anz%,T$( ),Height%) STATIC ' erzeugt Guru-Screen
anz%=anz%-1 : type%=0 : A1$=""
FOR I%=0 TO anz%
Y$(I%)=I%*12+15
X%=320-4*LEN(T$(I%))
A1$=A1$+MKI$(X%)+CHR$(Y$(I%))+T$(I%)+CHR$(0)
IF I%<anz% THEN A1$=A1$+CHR$(1) ELSE A1$=A1$+CHR$(0)
NEXT I%
CALL DisplayAlert(type%,SADD(A1$),Height%)
END SUB

Schreibschutzfehler: 'wenn Diskette nicht Write-Enabled ist
Mess$(0)="Fehler aufgetreten..."
Mess$(1)="Die Diskette muß Write-enabled (Schreibschutz auf) sein,"
Mess$(2)="da der Highscore gespeichert wird !!!"
Alarm 3,Mess$( ),50
CLOSE
RESUME Anfang

Fehler.in.Bilddatei: 'wenn Fehler beim Lesen einer Bilddatei auftrat
Mess$(0)="Fehler aufgetreten..."
Mess$(1)="So eine verdammte Sch.... !!!"
Mess$(2)="Fehler bei: "+Bild$+" - Fehler: "+Lesefehler$
Alarm 3,Mess$( ),50
WINDOW CLOSE 2 : SCREEN CLOSE 2
GOTO Hauptmenue.eins

Copyright:
GOSUB Text.loeschen
LOCATE 12,13 : Prnt " -----> Schiebespielchen-
Anleitung <-----"
LOCATE 14,13 : Prnt "Schiebespielchen wurde im Zeitraum vom
10.4.-24.4.89"
LOCATE 15,13 : Prnt "geschrieben und danach ständig verbessert..."
LOCATE 17,13 : Prnt "Das Spiel selbst ist ganz einfach:"
LOCATE 19,15 : Prnt "Die Felder müssen ganz einfach in ihre richtige"
LOCATE 20,15 : Prnt "Reihenfolge gebracht werden."
LOCATE 21,15 : Prnt "Die Taktik ist auch ganz einfach: immer streng"
LOCATE 22,15 : Prnt "der Reihe nach die Felder sortieren."
LOCATE 23,15 : Prnt "
Der Highscore wird übrigens abgespeichert !!!"
LOCATE 24,15 : Prnt "...und raus gehts mit <Ctrl> - C..."
LOCATE 26,50 : Prnt "Thats all folks..."
GOSUB Warte.auf.Maus
GOSUB Text.loeschen
GOTO Hauptmenue

' Schiebespiel Ende
```

Listing 1. (Schluß)

```
' Schiebespiel Highscore-Creator 1989 G.Kensche
'
letzte Änderung: 27.5.89

PRINT "Highscore-Creator 1989 by Guido Kensche"+CHR$(13)
PRINT "Richtet den Highscore fuer das Schiebespiel ein"
```

```
PRINT "oder loescht ihn. Auf der Diskette muß sich das"
PRINT "Verzeichnis Highscore befinden !!!"
PRINT "Die .info-Dateien koennen gelöscht werden"
PRINT "(im CLI: ed Highscore , delete #?.info"
PRINT CHR$(13); "Starten mit beliebiger Taste"
WHILE INKEY$="" : WEND

FOR X=1 TO 6
READ Hilf$
FOR T=1 TO 2
OPEN "R",1,":Highscore/"+Hilf$,27
FIELD #1,20 AS Nam$,2 AS Versch$,5 AS Zeit$

FOR I=1 TO 5
LSET Nam$="The Famous Guy"
LSET Versch$=MKI$(3000)
LSET Zeit$="59:59"
PUT #1,I
NEXT I

Hilf$=Hilf$+".Zeit"
CLOSE
NEXT T
NEXT X
END

DATA Standard,Standard II,Megabild,Elefant,Auto,Klassik
```

Listing 2. »HIGHSCORE-ERSTELLEN« generiert die notwendigen Files im Directory »HIGHSCORE«. Das Directory müssen Sie selbst anlegen.

```
REM Höhen-Patch fr AmigaBasic
REM passt die Höhe des Output-Windows
REM an die verfügbare PAL-Höhe an
REM
REM erstmals veröffentlicht in
REM AMIGA-Sonderheft 1 / Seite 137 und 138
REM
REM Patches von Peter Pörner und Alexander Wagner
REM Programm von Peter Pörner
REM aufgewärmt fr SCHIEBESPIELCHEN von Hans Waldhäusel
REM
REM wenn möglich, die Konvertierung im RAM ablaufen lassen
REM (das Programm ist darauf ausgelegt !)
REM zuerst das Original unter >>OldAmigaBasic<< in die
REM RAM-Disk kopieren, dann Programm starten !

CLEAR, 50000&
NeueHoehe$=CHR$(1)+CHR$(0) 'erste 1=256

OPEN"ram:OldAmigaBasic" FOR INPUT AS #1 LEN=4096
OPEN"ram:NewAmigaBasic" FOR OUTPUT AS #2 LEN=4096

PRINT #2, INPUT$(&H57EC, #1); 'Anfang kopieren
PRINT #2, NeueHoehe$; 'neue Höhe
a$=INPUT$(2, #1) 'alte Höhe ignorieren

PRINT #2, INPUT$(&H7000, #1); 'Rest kopieren
PRINT #2, INPUT$(&H7000, #1);
PRINT #2, INPUT$(&H5C00, #1);

WHILE NOT EOF(1)
PRINT #2, INPUT$(1, #1);
WEND

CLOSE #1, #2

END
```

Listing 3. Mit »HöhenPatch« steht Ihnen die volle PAL-Auflösung in Amiga-Basic zur Verfügung

Grafische Meisterleistung: Memory

Zwilling gesucht

Sie meinen,
Sie haben ein gutes
Gedächtnis?
Na mal sehen...

Von Hans Waldhäusel

Kartenspiele auf dem Amiga haben Tradition. Poker (meist mit dem Zusatz »Strip«), Patiences, Baccara, Black Jack und so manches andere Spiel gibt es in mehreren Varianten. Wir präsentieren einen weiteren Vertreter dieses Genres: Memory.

Memory in der Version 2.0 ist in Assembler geschrieben, wird mit der Maus gesteuert und bietet in Anbetracht der Kürze des ausführbaren Programms (nur etwas mehr als 8 KByte) erstaunlich viel Spielspaß. Die Spielregeln von Memory zu erklären, hieße wahrscheinlich, Bier nach München zu tragen (oder die sprichwörtlichen Eulen nach Athen). Aber weil »wahrscheinlich« nicht »sicher« ist, finden Sie hier eine kurze Einführung.

Zuerst zum Vorbild: Ein Paket Karten, in dem jedes Bild zweimal vorkommt, wird gut durchgemischt und die Karten werden mit der Bildseite nach unten auf den Spieltisch gelegt. Es ist Aufgabe des Spielers, die Kartenpärchen herauszufinden und aus dem Spiel zu nehmen. Dazu deckt der Spieler eine der Karten auf und wählt eine zweite aus den noch verdeckten Karten, von der er weiß (oder glaubt, zu wissen), daß sie der gesuchte Zwilling ist. War die Wahl richtig, so ist das Pärchen aus dem Spiel zu nehmen und dem Spieler ein Punkt gutzuschreiben. Wenn nicht, werden beide Karten wieder umgedreht. Das Gehirntraining besteht nun darin, sich die bei jedem Fehlversuch gesehenen Karten zu merken und dieses Wissen bei nächster Gelegenheit nutzbringend anzuwenden. Nehmen mehrere Personen am Spiel teil, kommt nach jedem Fehlversuch der nächste an die Reihe. Ziel des Spieles ist, alle Pärchen mit möglichst wenig Fehlversuchen ein-

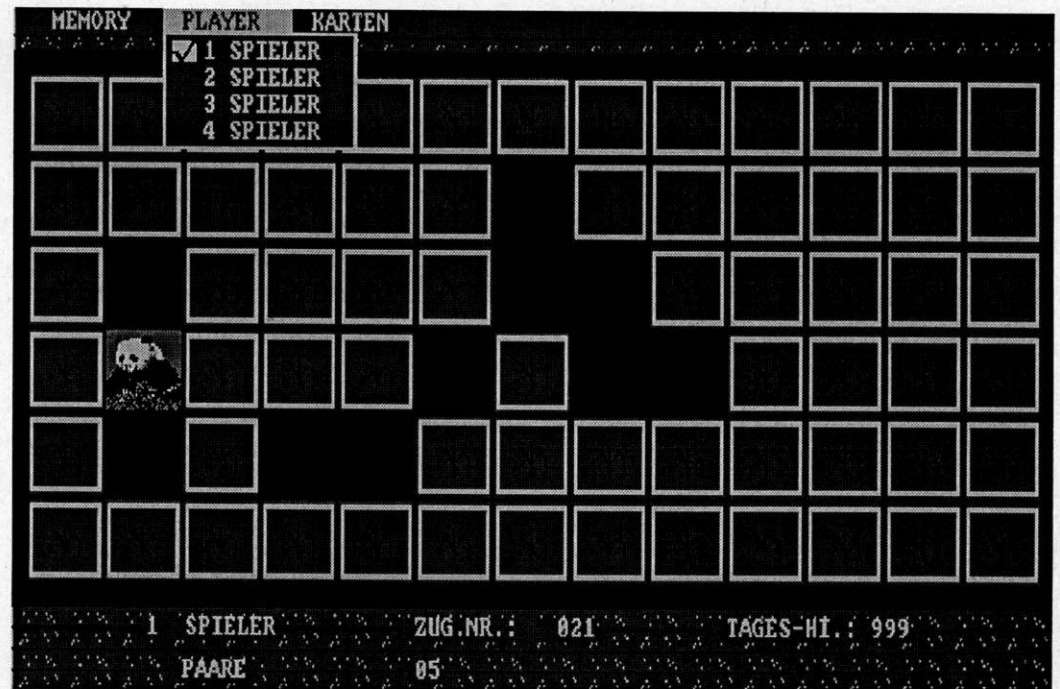


Bild 1. Auf der Suche nach dem Zwilling

ander zuzuordnen. Spielt man allein, so zählt die Anzahl der Versuche; bei mehreren Spielern zählt die Anzahl der »erbeuteten« Pärchen und damit die Anzahl der Punkte.

Bei der AMIGA-Implementation laden Sie zuerst die Workbench und dann Memory durch Anklicken des Icons. Ein Start

Modus und es wird immer die erste Karten-Datei geladen (Erläuterungen hierzu finden Sie bei der Erklärung der Menüpunkte). Nach bewährter AMIGA-Manier steht Ihnen jetzt durch Druck auf die rechte Maustaste eine Menüleiste mit folgenden Funktionen zur Verfügung:

MEMORY	ABOUT	hier hat sich der Programmator Andreas Herckner verewigt (Neu)start des Programms die Besten-Liste (Top-Ten) Programm beenden, zurück zur Workbench
PLAYER	- START - HI-SCORE - QUIT	Spielmodus für 1 bis 4 Mitspieler anwählen
KARTEN	- IFF-LADEN - IFF-SAVEN	neue Karten und Restart Leerbild-Raster auf beliebige Diskette speichern

aus dem CLI ist nicht vorgesehen. Nach dem Laden (die 8K sind schnell im Speicher) zeigt Ihnen Memory kurz die aktuellen Kartenbilder, was aber keine Rückschlüsse auf die Lage der Karten im Spielfeld zuläßt, da der Mischvorgang auf Zufallszahlen basiert.

Zu Spielbeginn befinden Sie sich immer im Ein-Spieler-

Einige der Menüpunkte bedürfen einer näheren Erläuterung. Beginnen wir mit der Anzahl der Mitspieler. Wie beim realen Kartenspiel ist im Mehr-Spieler-Modus nach jeder Fehlwahl der Nächste an der Reihe. Welcher Spieler am Zug ist, wird durch den Mauszeiger signalisiert, der immer die Kennzahl des aktiven Spielers zeigt.

Im unteren Bildschirm-Bereich werden sowohl die Anzahl der je Mitspieler gefundenen Pärchen als auch die Gesamtzahl der Versuche mitprotokolliert. Eine Änderung der Spielerzahl führt automatisch zu einem Neustart des Programmes.

Die Wahl von »START« löscht alle bis dahin aufgezeichneten Werte, also Zugzahl und Pärchen je Spieler. Auch die Karten werden neu gemischt, doch bleibt die zu Beginn geladene Karten-Datei aktiv.

Der Menüpunkt »IFF-LADEN« holt für Sie ein neues Karten-Paket aus der Schublade. Durch die Auslagerung der Daten in IFF-Files stehen Ihnen unbegrenzt viele verschiedene Spielflächen zur Verfügung (»unbegrenzt« ist natürlich abhängig von der Größe Ihres Massenspeichers, aber wenn Sie eine 80-MB-Hard-Disk mit MEMORY-Karten füllen wollen – bitte sehr). Der bei der Wahl von »IFF-LADEN« eingeblendete Requester erlaubt vollen Zugriff auf jedes Speichermedium. MEMORY gibt zwar beim ersten Aufruf dieser Funktion den Namen »MEMO_PIC« vor, doch sind Ihrer Fantasie bei der Namensgebung keine Grenzen gesetzt.

Mit »IFF-SAVEN« kommen wir zum künstlerischen Teil des Abends. Mit dieser Leerbild-Vorlage und einem Malprogramm zeichnen Sie Ihre eigenen MEMORY-Karten. Einzige Bedingung: Med-Res-Format, also 640 x 200 Punkte Auflösung. Die ersten vier Farben sind festgelegt, der Rest bleibt Ihrem Geschmack überlassen. Angenommen, Sie sind stolzer Besitzer eines Originals von »Deluxe Paint« (als Besitzer einer Raubkopie hätten Sie keinen Grund, stolz zu sein). Sie laden und starten »MEMORY« wie beschrieben und wählen den Menü-Punkt »IFF-SAVEN«. In den eingeblendeten Requester tragen Sie Pfad- und Filenamen für Ihr MEMORY-Leerbild ein, also z.B.

```
df1:Pictures/Med-Res/
memory.pic
```

Beenden Sie »MEMORY« und starten Sie »Deluxe Paint«. Am besten wählen Sie gleich von Beginn an die Med-Res-Auflösung. Laden Sie das File »memory.pic« und schützen Sie es vor versehentlichem Überschreiben, indem Sie aus dem »Effects«-Menü den Punkt »Background-Fix« wählen. Nun können Sie entweder wild drauflos malen oder beispielsweise Brushes laden und auf das Rastermaß von »MEMORY« verkleinern - verwenden Sie zum Verkleinern die Taste <h>; das ist viel bequemer als dauernd die Menü-Leiste auf- und abzufahren. Der Raster von »memory.pic« ist übrigens nur eine Größen-Indikation. Sie können diesen Raster unbe-

sorgt übermalen - »MEMORY« holt sich seine Spielkarten schon im richtigen Format. Aber hüten Sie sich, zwei gleiche Karten herzustellen - Sie machen sich nur selbst das »MEMORY«-Leben schwer. Und noch ein Tip: Besonders edel wirkt das Ganze, wenn Sie zum Schluß den Rahmen der Spielkarte mit der Hintergrundfarbe ausfüllen. Haben Sie alle Kästchen gefüllt, speichern Sie es unter einem beliebigen Namen; wie gesagt, »MEMORY« läßt Ihnen da völlig freie Hand. Wollen Sie Ihr Kunstwerk jedoch als Default-File deklarieren, damit es bei Programm-Start automatisch nachgeladen wird, müssen Sie es unter dem File-Namen »MEMO_PIC« im selben Directory wie »MEMORY« ablegen.

Genug des Vergnügens, kommen wir zur Arbeit. Als stolzer Besitzer der Leserservice-Diskette gehören Sie zu jenen Glücklichen, die Ihre gequälten Fingerspitzen nicht in Geschirrspülmittel zu baden brauchen. Sie kopieren einfach die in der Schublade »MEMORY« abgelegten Files »MEMORY-HI«, »MEMORY-VERS 2.0« und »MEMO_PIC« mit

```
copy dfx:memory/#?
to dfy: all
```

auf Ihre Spiele-Diskette (am besten in ein eigenes Directory oder noch besser auf eine MEMORY-Diskette wegen des deaktivierten Schreibschutzes). »x« und »y« stehen für die zur Verfügung stehenden Laufwerke, wobei Sie anstelle von »dfy:«

selbstverständlich auch »ram:« einsetzen können. Das wär's auch schon gewesen.

Etwas schwerer haben es diejenigen, die den Source-Code abtippen wollen (oder müssen - die Inflation nagt am Taschengeld). Listing 1 enthält das Hauptprogramm »MEMORY-VERS 2.0«. Nachdem Sie mit Sysphus im Geiste Bruderschaft getrunken haben, können Sie Ihren Assembler anwerfen. Wichtig dabei ist nur, daß Sie im Assemblierungs-Auswahlmenü »DATA« auf »CHIP« setzen. Auf einen Abdruck der Bilddaten haben wir - wie wir glauben in Ihrem Interesse - verzichtet. Das Leerbild-Raster wird vom Programm generiert (siehe »IFF-BILD SAVEN«), und malen ist sicher angenehmer als rund 23000 Bytes einzutippen. Auch das Highscore-File »MEMORY-HI« wird vom Programm automatisch angelegt, also wundern Sie sich bitte nicht, daß »nur« ein Listing einzugeben ist. Da Sie »MEMORY« aber nur von der Workbench aus aufrufen können, müssen Sie noch ein beliebiges TOOL-Icon von einer Ihrer Disketten für »MEMORY« ummodellern. Ein einfallsloses Beispiel: das DiskCopy-Icon von Ihrer Workbench. Rufen Sie das CLI auf und geben Sie ein:

```
copy SYSTEM/Diskcopy.
info ram:xyz.info
copy ram:xyz.info "dfy:
MEMORY-VERS 2.0.info"
```

Es ist, wie wir meinen, an der Zeit, ein wenig auf die Technik des Programms einzugehen.

Für die Assembler-Spezialisten unter Ihnen ist der sehr ausführlich kommentierte Source-Code wohl informativ genug. Für die weniger beschlagenen Leser (und Spieler) hier einige Anmerkungen:

MEMORY ist voll multitasking-fähig, was man so manchem berufsmäßigen Spiele-Programmierer ins Stammbuch schreiben sollte. Verwerflichkeiten wie Polling am Nachrichten-Port und damit Zeitdiebstahl an anderen Tasks oder absolute Speicherzugriffe werden Sie hier im Gegensatz zu vielen kommerziellen Programmen (Gott sei Dank) vergeblich suchen. Fehler werden korrekt abgefangen, Ressourcen sauber angefordert und nach Gebrauch ebenso sauber wieder freigegeben.

Tüftlerei

Während das Programm auf eine Eingabe des Spielers wartet, wird keine Rechenzeit verschwendet - der Programm-Task wird in den Status »Wait« versetzt und wartet auf ein klassifiziertes Ereignis am Nachrichten-Port (in diesem Fall eine der beiden Maustasten). Sehen Sie sich den Programmteil zwischen den Labels »loop« und »MsgOff« in aller Ruhe an. Wie uns Andreas Herckner mitteilte, ist MEMORY sein erstes Programm - freuen wir uns schon mal auf das nächste. so

Hans Waldhäusel kam als Programmierer vom 128er zum Amiga. Sie erreichen ihn über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z.Hd. Herrn Waldhäusel, Hans-Pinsel-Str. 2, 8013 Haarj).

```
*****
*** Programm - Memory-Amiga by A. Herckner ***
*** Vers. 14 4 89-02.23 ***
*** Copyright by A. Herckner ***
*** >>>* Data * muß im ChipRam stehen<<< ***
*** Assembler - Profimat von Data Becker ***
*****
```

```
_lvoWait equ -$13e
_lvoOpenLibrary equ -$228
_lvoCloseLibrary equ -$19e
_lvoOpenWindow equ -$cc
_lvoCloseWindow equ -$48
_lvoSetPointer equ -$10e
_lvoClearPointer equ -$3c
_lvoOpenScreen equ -$c6
_lvoCloseScreen equ -$42
_lvoDrawImage equ -$72
_lvoPrintIText equ -$d8
_lvoAutoRequest equ -$15c
_lvoDelay equ -$c6
_lvoFindTask equ -$126
_lvoWaitPort equ -$180
_lvoGetMsg equ -$174
_lvoReplyMsg equ -$17a
_lvoForbid equ -$84
```

```
_lvoPermit equ -$8a
_lvoLoadRGB4 equ -$c0
_lvoSetMenuStrip equ -$108
_lvoClearMenuStrip equ -$36
_lvoItemAddress equ -$90
_lvoOffMenu equ -$b4
_lvoOnMenu equ -$c0
_lvoOpen equ -$1e
_lvoClose equ -$24
_lvoWrite equ -$30
_lvoRead equ -$2a
_lvoCurrentDir equ -$7e
_lvoallocmem equ -$c6
_lvofreemem equ -$d2
_lvoIoErr equ -$84
_lvoActivateGadget equ -$1ce
_lvoAddGadget equ -$2a
_lvoReMoveGadget equ -$e4
_lvoOnGadget equ -$ba
_lvoRequest equ -$f0
_lvoEndRequest equ -$78
_lvoClearScreen equ -$30
_lvoMOVE equ -$f0
_lvoExit equ -$90

Pr_CLI equ 172
Pr_MsgPort equ 92
```



```

Callsys:Macro $1
JSR _lvo\1(A6)
Endm

Linksys:Macro $1,$2
MOVE.L A6,-(SP)
MOVE.L \2,A6
JSR _LVO\1(A6)
MOVE.L (SP)+,A6
Endm

MOVE.L SP,Initial_SP ;Stapel_Zeiger sichern
MOVE.L 4,A6 ;Zeiger_ExecBase laden
MOVE #0,A1 ;0 = eigener Task
Callsys FindTask ;Adresse suchen
MOVE.L DO,A4
TST.L PR_CLI(A4) ;laufen wir unter WB?
BNE Exit ;Nein dann **exit cli**
LEA PR_MsgPort(A4),A0
Callsys WaitPort ;Warte auf Start_Zeichen
LEA PR_MsgPort(A4),A0
Callsys GetMsg ;StartCode holen
MOVE.L DO,WBenchMsg ;Sichern zum AbMelden
LEA Intui,A1
MOVEQ #0,DO
Callsys OpenLibrary ;Librarys öffnen
MOVE.L DO,IntuitionBase ; " -Base sichern
BEQ Fehler
LEA Dos,A1
MOVEQ #0,DO
Callsys OpenLibrary
MOVE.L DO,DosBase
BEQ Fehler1
LEA Gfx,A1
MOVEQ #0,DO
Callsys OpenLibrary
MOVE.L DO,GfxBase
BEQ Fehler2
LEA Screen,A0 ;ScreenStruktur-Zeiger
Linksys OpenScreen,IntuitionBase ;Screen öffnen
MOVE.L DO,ScreenBase ;Screenadresse sichern
BEQ Fehler3
BSR Finit
LEA Window,A0 ;FensterStruktur-Zeiger
Linksys OpenWindow,IntuitionBase ;Fenster öffnen
MOVE.L DO,WindowBase ;FensterStruk.Adr.sichern
BEQ Fehler4
MOVE.L #30150,DO
MOVE.L #10002,D1 ;ChipSpeicher für Bilder
Callsys AllocMem
MOVE.L DO,I2
ADDI.W #192,DO
MOVE.L DO,ImagePic1
CMP.L #192,DO
BNE ProgStart ;alles I/O dann weiter
MOVE.L #FehlerText,FT ;sonst alles beenden
BSR Print1
BSR Wait
BRA Loop11

Loop1: MOVE.L OldDir,D1 ;EndeProg
Linksys CurrentDir,DosBase ;alles beenden
Loop11: MOVE.L I2,A1
MOVE.L #30150,DO
Callsys FreeMem
MOVE.L WindowBase,A0
Linksys ClearMenuStrip,IntuitionBase
MOVE.L WindowBase,A0
Linksys ClearPointer,IntuitionBase
MOVE.L WindowBase,A0
Linksys CloseWindow,IntuitionBase
Fehler4: MOVE.L ScreenBase,A0
Linksys CloseScreen,IntuitionBase
Fehler3: MOVE.L GfxBase,A1
Callsys CloseLibrary
Fehler2: MOVE.L Dosbase,A1
Callsys CloseLibrary
Fehler1: MOVE.L IntuitionBase,A1
Callsys CloseLibrary
Fehler: TST.L WBenchMsg ;gibs MSG

```

```

BEQ Exit ;Nein dann war's CLI
Callsys Forbid
MOVE.L WBenchMsg,A1
Callsys ReplyMsg ;TASK abmelden
Exit: MOVEQ #0,DO ;ReturnCode
MOVEQ #0,D1
MOVE.L Initial_SP,SP ;alter StapelZeiger
RTS ;AUFWIEDERSEHEN

;*****

ProgStart: BSR SetMenu ;Menue aufbauen
MOVE.L WindowBase,A0
MOVE.L 50(A0),RastBase ;Rastport - Zeiger
MOVE.L 86(A0),UserPort ;Userport - Zeiger
LEA HiSpeicher,A0
MOVE.L A0,HiZahlTab ;HiScore ZahlenTabelle
ADD.L #22,A0
MOVE.L A0,HiTextTab ; " TextZeigerTabelle
ADD.L #44,A0
MOVE.L A0,HiTextSpe ; " TexteSpeicher
MOVE.L WBenchMsg,A0 ;*
MOVE.L $24(A0),A0 ;Start Look
MOVE.L (A0),D1
MOVE.L D1,CdDir
Linksys CurrentDir,DosBase ;setzen
MOVE.L DO,OldDir ;*
MOVE.L #FileHI,D1 ;
MOVE.L #1005,D2
Linksys Open,DosBase ;HiScore laden
MOVE.L DO,FileHD
BEQ ErrH1 ;Fehler dann Hi Neu
MOVE.L DO,D1
MOVE.L #HiSpeicher,D2
MOVE.L #566,D3
LINKSYS Read,DosBase
MOVE.L DO,-(SP)
MOVE.L HiTextTab,A0
MOVE.L HiTextSpe,D0
MOVEQ #9,D1
Hiof1: ADD.L DO,(A0)+ ;Offsets für Hi-Texte
DBRA D1,Hiof1 ;zum Speicher addieren
MOVE.L FileHD,D1
Linksys Close,DosBase
MOVE.L (SP)+,DO ;
BPL EPA ;READfehler?
ErrH1: BSR HiNeu ;Ja Neu
EPA: BRA Iffread ;Ende Prog.Anfang
Neu: LEA ImageTabelle,A1 ;Neues Spiel
MOVEQ #1,D1
MOVEQ #38,DO
Karten: MOVE.B D1,39(A1)
MOVE.B D1,(A1)+ ;SpielKartenTab.anlegen
ADDQ #1,D1
DBRA DO,Karten
MOVE.B #0,Zeiger
MOVE.B #39,PaarZähler
LEA ImageTabelle,A0 ;**
MOVE.L #1000,D4
NZiehen: MOVE.B $dff006,DO
AND.B D4,DO
ANDI.L #%1111111,DO
CMPI.B #77,DO
BHI NZiehen
MOVE.B (A0,DO),D1
Ziehen2: DBF D3,Ziehen2 ;Karten mischen
MOVE.B $dff006,D3
ROL.B #2,D3
ANDI.L #%1111111,D3
CMPI.B #77,D3
BHI Ziehen2
MOVE.B (A0,D3),D2
MOVE.B D2,(A0,DO)
MOVE.B D1,(A0,D3)
DBF D4,NZiehen ;**

```

Listing 1. MEMORY – der Quell-Code wurde für den Data-Becker-Proformat geschrieben. Eine Konvertierung auf andere Assembler dürfte relativ einfach sein.


```

MOVE #0,ZugNr ;Zähler
MOVE.L #0,PaarZL ;Zähler
MOVE.B #0,SpielerNr
LEA Pointer,A1 ;1.Pointer
BSR SetSprite ;setzen
BSR SpielBild ;Bild aufbauen
Loop: BSR Wait
CMP #100,D6 ;Menue angewählt ?
BEQ Loop3 ;Menue auswertung
CMP #68,D7 ;Taste gedrückt ?
BEQ MousePos ;Karte aufdecken
BRA Loop
Wait: MOVE.L UserPort,A0 ;Userport-Zeiger
MOVE.B $f(A0),D1 ;SignalBit UNSERTASK
MOVEQ #0,D0
BSET D1,D0
Callsys Wait ;Warte auf Ereignis
GMsg: MOVE.L UserPort,A0
Callsys GetMsg ;Ereignis_PORT
TST.L D0 ;0 = Keine
BEQ MsgOff
MOVE.L D0,A1 ;Ereignis_ADRESSE
MOVE.L $14(A1),D6 ;hole Msg CLASS
MOVE $18(A1),D7 ; "" "" CODE
Callsys ReplyMsg ;EREIGNIS BESTÄTIGEN
CMP.L #40000,D6 ;war es ActiveWindow
BNE GMsg ;Nein
BSR SpielBild1 ;Refreh Window
MOVE.L #40000,D6
BRA GMsg
MsgOff: RTS ;Zurück meist Loop

```

```

MousePos: MOVEQ #0,D3 ;SpielProgAnf.
MOVEQ #0,D4
MOVE.L WindowBase,A0
MOVE $0e(A0),D4 ;Mouse Y Position
MOVE $0e(A0),D3 ;Mouse X Position
MOVE D3,MousePosX
MOVE D4,MousePosY
CMP # 8,D3
BLS Loop ;Ist die Maus
CMP # 632,D3
BGE Loop ;auf dem
CMP # 24,D4
BLS Loop ;Spielfeld ?
CMP # 216,D4
BGE Loop

```

```

ImageSuchen: MOVEQ #0,D0 ;KarteNummer?
MOVEQ #0,D1
MOVE MousePosX,D0 ;Karte auf der
MOVE MousePosY,D1 ;die Maus steht
SUBQ #8,D0
SUBI #24,D1
DIVU #48,D0 ;suchen
DIVU #32,D1
MOVE D0,MousePosX ;und
MOVE D1,MousePosY
MULU #13,D1 ;die Positionen
ADD D0,D1
MOVE.B D1,ImageNum ;festhalten
MOVE MousePosX,D0
MULU #48,D0
ADDQ # 8,D0
MOVE D0,MousePosX
MOVE MousePosY,D0
MULU #32,D0
ADDI #24,D0
MOVE D0,MousePosY
LEA ImageTabelle,A0 ;Kartentabelle
MOVEQ #0,D0
MOVE.B ImageNum,D0 ;KartenNummer
MOVE.B (A0,D0),D1 ;Karte holen
MOVE.B D1,ImageZeichnen
TST.B D1
BEQ Loop ;Karte schon offen ?
BCHG #0,Zeiger ;Nein
BEQ Image1 ;1.Karte ?
MOVE.B D1,Karte ;NEIN , 2.Karte
ImageZeigen: MOVEQ #0,D0
LEA ImageTabelle,A0 ;
MOVE.B ImageNum,D0

```

```

MOVE.B #0,(A0,D0) ;Karte offen
BSR Bild ;Karte laden
TST.B Zeiger
BNE Loop ;2.Karte holen
MOVEQ #50,D1 ;Zeit Karten löschen
Linksys Delay,DosBase ;Warte und lass andere
MOVE.B Karte ,D0 ;tasks arbeiten
CMP.B Karte1,D0 ;Karten vergleichen
BEQ Paar ;ist es ein Paar ?
LEA ImageTabelle,A0
MOVEQ #0,D0 ;NEIN ,ist kein Paar
MOVE.B ImageNum,D0
MOVE.B Karte,(A0,D0) ;Karten zurück in
MOVE.B ImageNum1,D0
MOVE.B Karte1,(A0,D0) ;Kartentabelle
MOVE MousePosX,XImage
MOVE MousePosY,YImage
LEA ImageDaten,A1 ;Karten auf Screen
BSR ZeigImage ;Löschen
MOVE MousePosX1,XImage
MOVE MousePosY1,YImage
LEA ImageDaten,A1
BSR ZeigImage
MOVE.B #0,KartenPaar ;
MOVE.L WindowBase,A0
Linksys ClearPointer,IntuitionBase ;alter Pointer
MOVEQ #0,D0 ;++
MOVE.B SpielerNR,D0
CMP.B Spieler,D0
BNE NextPL
MOVE.B #0,SpielerNR
LEA Pointer,A1
BSR SetSprite ;Wer ist am Zug
BRA ZugZL
NextPL: ADDQ.W #1,D0
MOVE.B D0,SpielerNR
MULU #44,D0
LEA Pointer,A1
ADD.L D0,A1
BSR SetSprite
BRA ZugZL ;++
Paar: MOVE.B #1,KartenPaar ;Paar Zähler
MOVE MousePosX,Image2X
MOVE MousePosY,Image2Y ;Paar Löschen
LEA Image2,A1
BSR ZeigImage
Paar2: MOVE MousePosX1,Image2X
MOVE MousePosY1,Image2Y
LEA Image2,A1
BSR ZeigImage
MOVEQ #0,D0
MOVEQ #0,D1
MOVEQ #1,D2
MOVE.B SpielerNR,D0
BSR PAARZL1
SUBQ.B #1,PaarZähler ;Paar gefunden
TST.B PaarZähler ;Alle Karten offen ?
BNE Loop ;NEIN , noch Karten dar
MOVE ZugNR,D0 ;Alle Paare gefunden
MOVE HiZL,D2
CMP D2,D0
BGE HiEintrag ;TagesHi ?
MOVE D0,HiZL ;Ja
BSR TagesHiZL
HiEintrag: MOVE ZugNR,D0
MOVE.L HiZahlTab,A0
MOVE.L HiTextTab,A1
MOVE 18(A0),D1
CMP D1,D0
BGE Loop ;Hi-Score?
MOVE.L 36(A1),A3 ;Ja
MOVE.L A3,StrInfo
ADDQ.L #6,StrInfo
MOVEM.L D0/D1/D2/A0,-(SP)
MOVE.L A3,A0
ADDQ.L #1,A0
BSR UmRechnen
MOVEM.L (SP)+,D0/D1/D2/A0
HiLoop: MOVE (A0)+,D1
MOVE.L (A1)+,A2
CMP D1,D0
BGE HiLoop ;Wo eintragen?

```



```

SUBQ.L #2,A0
SUBQ.L #4,A1
HiLoop1: MOVE.L (A1),A2
MOVE.L A3,(A1)+
MOVE.L A2,A3
MOVE (A0),D1
MOVE DO,(A0)+ ;Einträge aufrücken
MOVE D1,DO
TST (A0)
BNE HiLoop1
MOVE #34,ReqL
MOVE.L #StrText2,StrT ;Name holen
BSR Req1
MOVE.L HiTextTab,A0 ;Offset Speicher
MOVE.L HiTextSpe,DO ;minus
MOVEQ #9,D1
H1OF2: SUB.L DO,(A0)+
DBRA D1,H1OF2
MOVE.L #HiSpeicher,Buffer ;Hi auf Disk schreiben
MOVE.L #FileHI,D1
MOVE.L #566,D3
BSR Schreiben
MOVE.L HiTextTab,A0 ;Offset Plus
MOVE.L HiTextSpe,DO
MOVEQ #9,D1
H1OF3: ADD.L DO,(A0)+
DBRA D1,H1OF3
BRA HiBild ;Hi-Score Zeigen
Image1: MOVE.B D1,Karte1
MOVE MousePosX,MousePosX1
MOVE MousePosY,MousePosY1 ;1. Karte merken
MOVE.B ImageNum,ImageNum1
BRA ImageZeigen
Bild: MOVE mouseposx,pic2x
MOVE mouseposy,pic2y
MOVEQ #0,DO
MOVE.B ImageZeichnen,DO
MULU #768,DO ;Karte auf
SUBI.L #768,DO
MOVE.L imagepic1,a1
ADD.L DO,A1
MOVE.L A1,PicAdr ;KartenDaten
LEA Pic2,A1
BRA ZeigImage ;zeig sie
ZeigImage: MOVE.L RastBase,A0 ;RasPort_Zeiger
MOVEQ #0,D1 ;y Offset obere
MOVEQ #0,DO ;x linke Ecke im Screen
Linksys DrawImage,IntuitionBase
RTS
ZugZL: MOVE ZugNR,DO
ADDQ #1,DO
MOVE DO,ZugNR
BSR ZugZL1
BRA Loop
ZugZL1: LEA PL21,A0
BSR UmRechnen
LEA PL2,A2
MOVE.L #15000e4,(A2)
BSR Print
RTS
TagesHiZL: LEA PL21,A0
BSR UmRechnen
LEA PL2,A2
MOVE.L #21000e4,(A2)
BSR Print
RTS
PaarZL1: LEA PaarZL,A0
MOVE.B (A0,DO),D1
ADD D2,D1
MOVE.B D1,(A0,DO)
MULU #96,DO
SWAP DO
ADDI.L #f000f4,DO
MOVE.L DO,PL2
MOVE.L D1,DO
LEA PL21,A0
BSR UmRechnen
LEA PL21,A0
MOVE.B # " ",(A0)
LEA PL2,A2
BSR Print
RTS

```

```

UmRechnen: CMP #999,DO ;In ASCII Zeichen
BLS Rechnen
MOVE.L #999,DO ;A0 = Buffer
Rechnen: MOVEQ #1,D1 ;2*loop
MOVEQ #100,D2 ;Erster Teiler
R1: DIVU D2,DO
ADDI #30,DO
MOVE.B DO,(A0)+ ;Buffer Plus
DIVU #10,D2
MOVE #0,DO
SWAP DO
DBRA D1,R1
ADDI #30,DO
MOVE.B DO,(A0)
RTS
SetSprite: MOVE.L WindowBase,A0
MOVEQ #16,D1 ;Größe in Pixel
MOVEQ #9,DO
MOVEQ #-3,D2 ;HotSpod
MOVEQ #-3,D3
Linksys SetPointer,IntuitionBase
RTS
Finit: MOVE.L ScreenBase,A0
ADD.L #44,A0 ;Viewport
LEA CMap,A1 ;ColorData
MOVEQ #14,DO
Linksys LoadRGB4,GfxBase ;Color setzen
RTS
Print: MOVE (A2)+,DO
MOVE (A2)+,D1 ;X und Y Posi.
Print2: MOVE.L (A2)+,TPL ;welcher Text
LEA Plt,A1 ;Textstruk.
BRA PIText
Print1: MOVEQ #0,DO
MOVEQ #0,D1 ;X und Y Posi.
LEA Fts,A1 ;Textstruk.
PIText: MOVE.L RastBase,A0
Linksys PrintIText,IntuitionBase ;Text ausgeben
RTS
Loop3: MOVE D7,D6 ;IN Bits 0-4 Titel-NR.
AND #1f,D7 ;IN Bits 5-10 Punkt-NR
CMP #1f,D7 ;IN Bits 11-15 UnterPunkt
BEQ LOOP ;MenueTitelNR. keins = 1f
LSR #5,D6 ;MenuePunktNR 0-2
AND #3f,D6
TST D7
BEQ Menue1 ;0?
CMP #1,D7 ;### Welches von 3 Menus
BEQ Menue2 ;1?
BRA Menue3 ;kann nur 2 sein
Menue1: CMP #1,D6 ;Item Suchen
BEQ CLSP ;Neues Spiel?
CMP #3,D6
BEQ loop1 ;Quit?
TST D6
BNE HiBild ;Hi-Scores zeigen?
MOVE.B #1,ImageZeichnen ;About zeigen
BSR SpielBild2
LEA SB2T1,A2
MOVEQ #3,D2
SB2Text: BSR Print
DBRA D2,SB2Text
HiBild1: BSR Wait
MOVEM:L D6/D7,-(SP)
BSR SpielBild
MOVEM:L (SP)+,D6/D7
CMP #100,D6
BEQ LOOP3
BRA LOOP
Menue2: CMP.B Spieler,D6 ;Wieviel Spieler
BEQ Loop
MOVE.B #0,SpielerNR
MOVE.B D6,Spieler
CLSP: MOVE.L WindowBase,A0
Linksys ClearPointer,IntuitionBase
BRA Neu
Menue3: TST D6
BNE M32 ;War es Saven?

```

Listing 1. (Fortsetzung)


```

MOVE.L #StrBuffer,StrInfo ;Nein Laden
BSR Req1 ;Was ""
Bra IffRead ;Lade
M32: MOVE.L #StrBuffer1,StrInfo ;LeerBild ausgeben
BSR Req1 ;wohin welcher Name
MOVE.L #$1e00,D0
BSR Allocate ;Speicher für Bildleer
TST.L D0
BNE LoopDaten ;Speicher frei
MOVE.L #FehlerText1,FT ;nein
BSR FehlerR ;Fehler zeigen
BRA Loop
LoopDaten: MOVE.L D0,Buffer ;Daten aufbauen
MOVE.L D0,A0
MOVEQ #28,D0
LEA BildData,A1
LoopData: MOVE.L (A1)+,(A0)+
DBRA D0,LoopData
MOVEQ #47,D0
LoopData1: MOVE.L #$B100B100,(A0)+
DBRA D0,LoopData1
MOVEQ #2,D2
LoopData2: BSR Data2
MOVEQ #29,D0
LoopData3: MOVEQ #12,D1
MOVE #100,(A0)+
LoopData4: MOVE.B #$80,(A0)+ ;Vorsicht
MOVE.B #$fd,(A0)+ ;Speicher zugriff
MOVE.B #0,(A0)+ ;ungerade Adresse
MOVE.B #1,(A0)+
MOVE.B #1,(A0)+
DBRA D1,LoopData4
MOVE.B #0,(A0)+ ;wird gerade
BSR DATA3
DBRA D0,LoopData3
BSR Data2
DBRA D2,LoopData2
MOVE.L #159,D0
LoopData5: MOVE.L #$B100B100,(A0)+
DBRA D0,LoopData5
MOVE.L #StrBuffer1,D1
MOVE.L #$1e00,D3
BSR Schreiben ;Bild ablegen
MOVE.L Buffer,A1
MOVE.L #$1e00,D0
Callsys FreeMem ;Speicher zurück
BRA Loop ;Weiter
Schreiben: MOVE.L #1006,D2 ;+++
Linksys Open,DosBase
MOVE.L D0,FileHD
BNE Schreib
BSR FehlerRR
BRA SchreibenEnd
Schreib: MOVE.L FileHD,D1 ;Daten ablegen
MOVE.L Buffer,D2
Linksys Write,DosBase
BPL SchreibEnd
BSR FehlerRR
SchreibEnd: MOVE.L FileHD,D1
Linksys Close,DosBase
SchreibenEnd: RTS ;+++
FehlerRR: Linksys IoErr,DosBase ;***
LEA FehlerText,A0
BSR UmRechnen
MOVE.L #FehlerText,FT
FehlerR: BSR Print1 ;SchreibFehlerTexte
MOVEQ #100,D1
Linksys Delay,DosBase
BSR SpielBild1
RTS ;***
Data2: MOVE.L #$b3ff,(A0)+
MOVE #0,(A0)+
Data3: MOVE #$b100,(A0)+
MOVE.L #$b100b100,(A0)+
RTS
Req1: MOVE.L WindowBase,A1 ;
LEA Requester,A0
MOVE ReqL,ReqC
MOVEQ #0,D0
Linksys Request,IntuitionBase
TST.L D0
BPL Req2

```

```

MOVE.L WindowBase,A1
LEA Requester,A0
Linksys EndRequest,IntuitionBase
Req3: MOVE.L (SP)+,D6
BRA Loop
Req2: MOVE #$80,StOn
MOVE.L WindowBase,A1
LEA StGadget,A0 ;Requester ausgeben
LEA Requester,A2
Linksys ActivateGadget,IntuitionBase
BSR Wait
CMP #$1000,D6
BNE Req2
MOVE #50,ReqL
MOVE.L #StrText,StrT
MOVE.L StrInfo,A0
MOVE.B (A0),D0
TST.B D0
BEQ Req3
RTS ;

SpielBild: MOVE.L ScreenBase,A0 ;Alles Schwarz Setzen
ADD.L #44,A0
MOVE.L I2,A1
MOVEQ #20,D0
Linksys LoadRGB4,GfxBase
MOVE.L RastBase,A1 ;Bitmaps löschen
MOVEQ #0,D0
MOVEQ #0,D1
Linksys MOVE,GfxBase
MOVE.L RastBase,A1
Linksys ClearScreen,GfxBase
SpielBild1: Callsys Forbid ;Bild aufbauen
MOVE #0,xImage
MOVE #32,bImage ;Spielbild
MOVE #16,Half
MOVE.L #Ziegel_Karte,kImage
MOVEQ #19,D3
MOVEQ #19,D4
MOVEQ #32,D7
MOVEQ #16,D6
MOVEQ #0,D5
MOVEQ #0,D2
MOVE #0,yImage
BSR Image
MOVE #$e0,yImage
MOVEQ #1,D2
BSR Image
MOVE #48,bImage
MOVE #32,Half
MOVE.L #AH_Karte,kImage
MOVEQ #12,D3
MOVEQ #12,D4
MOVEQ #8,D5
MOVEQ #5,D2
MOVEQ #48,D7
MOVEQ #32,D6
MOVE #8,xImage
MOVE #24,yImage
LEA ImageTabelle,A3
BSR Image4
TST.B Zeiger
BEQ Weiter1 ;War Karte offen?
MOVE MousePosX1,MousePosX ;Ja
MOVE MousePosY1,MousePosY
MOVE.B Karte1,ImageZeichnen
BSR Bild
Weiter1: BSR SpText ;Texte Spielbild
BSR Finit ;Farben normal
Callsys Permit
RTS
SpielBild2: Callsys Forbid ;Bild Hi/About
MOVE #0,xImage
MOVE #0,yImage
MOVE #32,bImage
MOVE #16,Half
MOVE.L #Ziegel_Karte,kImage
MOVEQ #19,D3
MOVEQ #19,D4
MOVEQ #32,D7
MOVEQ #16,D6

```



```

MOVEQ #0,D5
MOVEQ #15,D2
BSR Image
BSR SpielBild21
Callsys Permit
RTS
SpielBild21: MOVE #8,MousePosX
MOVE #17,MousePosY
MOVEQ #12,D2
MOVEQ #1,D3
Bild21: BSR Bild
ADDQ.B #1,ImageZeichnen
ADDI #48,MousePosX
DBRA D2,Bild21
MOVE #8,MousePosX
MOVE #209,MousePosY
MOVEQ #12,D2
DBRA D3,Bild21
MOVE #49,MousePosY
MOVEQ #4,D2
MOVEQ #1,D3
Bild22: BSR Bild
ADDQ.B #1,ImageZeichnen
ADDI #32,MousePosY
DBRA D2,Bild22
MOVE #584,MousePosX
MOVE #49,MousePosY
MOVEQ #4,D2
DBRA D3,Bild22
RTS
Image: LEA ImageDaten,A1 ;Zeiger Image-Struktur
BSR ZeigImage
ADD D7,xImage ;x Offset Image erhöhen
DBRA D3,Image
MOVE D4,D3
MOVE D5,xImage
ADD D6,yImage ;y Offset Image erhöhen
DBRA D2,Image
RTS
Image4: TST.B (A3)+
BEQ NoImage
LEA ImageDaten,A1 ;Zeiger Image-Struktur
BSR ZeigImage
NoImage: ADD D7,xImage ;x Offset Image erhöhen
DBRA D3,Image4
MOVE D4,D3
MOVE D5,xImage
ADD D6,yImage ;y Offset Image erhöhen
DBRA D2,Image4
RTS
SpText: LEA SpT,A2 ;Texte Spielbild zeigen
MOVE.B Spieler,D0
ADDI #31,D0
MOVE.B D0,SpT1
BSR Print
LEA SpT2,A2
BSR Print
LEA Titel,A2
BSR Print
LEA HiT,A2
BSR Print
LEA ZugT,A2
BSR Print
MOVE ZugNR,D0
BSR ZugZL1
MOVE HiZL,D0
BSR TagesHiZL
MOVEQ #0,D3
MOVE.B Spieler,D3
Pzs: MOVEQ #0,D1
MOVEQ #0,D2
MOVEQ #0,D0
MOVE D3,D0
BSR PaarZL1
DBRA D3,Pzs
RTS
HiBild: MOVE.B #4,ImageZeichnen ;HiBild zeigen
BSR SpielBild2
MOVE.L HiTextTab,A2
MOVEQ #9,D3
MOVEQ #70,D2
Bild2Text: MOVE.L #159,D0 ;Texte zeigen

```

```

MOVE D2,D1
BSR PRINT2
ADDI #12,D2
DBRA D3,Bild2Text
BRA HiBild1 ;warte Taste

HiNeu: MOVE.L HiZahlTab,A0
MOVEQ #9,D1
Hil1: MOVE #999,(A0)+
DBRA D1,Hil1
MOVE.L HiTextTab,A0
MOVE.L HiTextSpe,A1
MOVEQ #9,D1
Hil2: MOVE.L A1,(A0)+
ADD.L #50,A1
DBRA D1,Hil2
MOVEQ #9,D1
MOVE.L HiTextTab,A0
Hil4: MOVE.L (A0)+,A2
LEA HiName,A1
MOVEQ #40,D0
Hil3: MOVE.B (A1)+,(A2)+
DBRA D0,Hil3
DBRA D1,Hil4
RTS

IffRead: MOVE.L #StrBuffer,D1 ;Filename
BSR OpenFile
MOVE.L #Buffer,D2 ;File gefunden
MOVEQ #8,D3
BSR Load ;Iff-Form?
MOVE.L #Buffer,A3
CMP.L # "FORM", (A3)
BNE FormFehler ;Fehler
MOVE.L 4(A3),A3
MOVE.L A3,D0
BSR Allocate ;Speicher Bilddaten
TST.L D0
BEQ AllocFehler ;Fehler fragen
MOVE.L D0,Buffer
MOVE.L A3,D3
MOVE.L Buffer,D2
BSR Load ;Lade Bild
BSR CloseFile ;File schliessen
IffTest: MOVE.L Buffer,A0 ;BildAnfang
MOVE.L A0,BildAnfang
MOVE.L (A0),D0
CMP.L # "ILBM",D0 ;Grafik?
BNE BildFehler ;Fehler
MOVE 12(A0),D0
CMP # $280,D0 ;640 Punkte BREIT ?
BNE BildFehler ;Fehler
MOVE.B 20(A0),D0
CMP.B #4,D0 ;4 BitPlane ?
BNE BildFehler ;Fehler
MOVE.B 22(A0),GePackt
CMaps: MOVE (A0)+,D0 ;===
CMP.L # "CMAP", (A0) ;Farben suchen
BNE CMaps
ADD.L #20,A0 ;5 Farbe im File
MOVEQ #11,D0
LEA CMap,A1 ;ColorTabelle Spiel
ADDQ.L #8,A1 ;5 Farbe in Tabelle
Farben: MOVEQ #0,D1
MOVE.B (A0)+,D1
LSL #4,D1
MOVE.B (A0)+,D1 ;Farben lesen
LSL #4,D1
MOVE.B (A0)+,D1
LSR #4,D1
MOVE D1,(A1)+
DBRA D0,Farben ;===
BSR IffBody ;Bilddaten Suchen
BSR BPlane ;In Bitplanes Schreiben
BSR KEinlesen ;Imagedaten Aufbauen
BSR Free ;Speicher frei
BRA Neu ;Ausprung IffRead
BPlane: MOVE.L ScreenBase,A0 ;Bitplaneadr. ermitteln

```

Listing 1. (Fortsetzung)


```

ADD.L #0,A0
LEA PlaneBuf,A1
MOVEQ #3,D0 ;4 Planeadressen
PlaneNum: MOVE.L (A0)+,(A1)+
DBRA D0,PlaneNum
BildAusgeben: MOVE.L BildAnfang,A0
MOVE #24,ZeileNum ;Ab hier Schreiben
LEA PlaneBuf,A1
Weiter: MOVE.L (A1),A2
BSR Zeile
MOVE.L 4(A1),A2
BSR Zeile
MOVE.L 8(A1),A2
BSR Zeile
MOVE.L 12(A1),A2
BSR Zeile
ADDQ #1,ZeileNum
CMPI #150,ZeileNum ;bis hier Einlesen
BNE Weiter
RTS ;Fertig IffBild
Zeile: MOVEQ #79,D0 ;IffDaten schreiben
MOVE ZeileNum,D1 ;ins Bild
MULU #80,D1
ADD.L D1,A2
TST.B GePackt
BNE PLoop
Zeile1: MOVE.B (A0)+,(A2)+
DBRA D0,Zeile1
RTS
PLoop: MOVEQ #0,D1
MOVE.B (A0)+,D1
BMI Packed
Plus: MOVE.B (A0)+,(A2)+
SUBQ #1,D0
DBRA D1,Plus
BRA Test
Packed: CMPI #128,D1
BEQ Ploop
NEG.B D1
MOVE.B (A0)+,D2
PackLoop: MOVE.B D2,(A2)+
SUBQ.B #1,D0
DBRA D1,packloop
Test: TST.B D0
BPL Ploop
RTS ;Zeile fertig
IffBody: MOVE.L BildAnfang,A0
IffLoop: MOVE (A0)+,D1
MOVE.L (A0),D1
CMPI.L #424f4459,D1 ;Body Chunk suchen
BNE IffLoop
ADDQ.L #8,A0
MOVE.L A0,BildAnfang
RTS
KEinLesen: MOVE.L #1,KAnfang ;1*8 Bit SeitenAbstand
MOVE #48,ZeileNum ;AnfangsZeile
MOVE.L ImagePic1,A3
MOVEQ #2,D4
LEA PlaneBuf,A0
EinLesen1: MOVEQ #12,D3 ;13* KartenScreen
EinLesen: MOVE.L (A0),A1
BSR Lesen
MOVE.L 4(A0),A1 ;ImageDaten aufbauen
BSR Lesen
MOVE.L 8(A0),A1
BSR Lesen
MOVE.L 12(A0),A1
BSR Lesen
ADDQ.L #6,KAnfang ;Nächste Karte 6*8 Bits
DBRA D3,EinLesen ;13 Karten
ADDI #32,ZeileNum
MOVE.L #1,KAnfang
DBRA D4,EinLesen1
RTS
Lesen: MOVEQ #0,D0
MOVE ZeileNum,D0
MULU #80,D0
ADD.L KAnfang,D0
ADD.L D0,A1
MOVEQ #31,D2 ;32 Bits KartenHöhe
Lesen1: MOVEQ #5,D1 ;6*8 Bits KartenBreite

```

```

Lesen2: MOVE.B (A1)+,(A3)+
DBRA D1,Lesen2
ADD.L #74,A1
DBRA D2,Lesen1
RTS
OpenFile: MOVE.L #1005,D2 ;REG D1 FileName
Linksys Open,DosBase
TST.L D0
BEQ OpenFehler ;Fehler Fragen
MOVE.L D0,FileHD
RTS
Load: MOVE.L FileHD,D1 ;Reg D3 Lange lesen
Linksys Read,DosBase ;Reg D2 Buffer
TST.L D0
BMI ReadFehler ;Fehler fragen
RTS
Allocate: MOVE.L #10000,D1
Callsys Allocmem
RTS
CloseFile: MOVE.L FileHD,D1
Linksys Close,DosBase
RTS
Free: LEA Buffer,A0
MOVE.L (A0)+,A1
MOVE.L (A0),D0
Callsys FreeMem
RTS
AllocFehler: MOVE.L #FehlerText1,FT ;!!!
BSR CloseFile
BSR Warte
BildFehler: BSR Free
BRA BFehler
FormFehler: BSR CloseFile ;FehlerTexte Load
BFehler: MOVE.L #FehlerText2,FT ;IffBild
BSR Warte
ReadFehler: BSR CloseFile
OpenFehler: Linksys IoErr,DosBase
LEA FehlerText,A0
BSR UmRechnen
MOVE.L #FehlerText,FT
Warte: MOVE.L (SP)+,D6
BSR Print1
BSR Wait
BRA Neu ;!!!
SetMenu: LEA MenuTabelle,A0 ; auf MenuePunkte
LEA Menu,A1 ; auf MenueStrukturBuffer
MOVEQ #10,D1 ;X Position
MenuLoop: MOVEQ #0,D2 ;Y
MOVE.L A1,A2 ;Buffer retten
TST.L (A0) ;Menues fertig ?
BEQ SetMenu1 ;JA =MenuesDarstellen
MOVE.L #0,(A1)+ ;MenueTitel fertig
MOVE D1,(A1)+
ADDI.L #80,D1 ;X Pos erhöhen
MOVE.L #80,(A1)+ ;Y Pos und breite
MOVE.L #a0001,(A1)+ ;höhe und Flags
MOVE.L (A0)+,(A1)+ ;MenueTitel
LEA 12(A1),A3 ; MenueItems
MOVE.L A3,(A1)+
MOVE.L #0,(A1)+
MOVE.L #0,(A1)+
ItemLoop: TST.L (A0) ;Letzter Eintrag ?
BEQ MenuEnd ;JA Menue fertig
LEA 54(A1),A3
MOVE.L A3,(A1)+ ; nächsten Item
MOVE.L D2,(A1)+ ;X Y Pos
ADDI #10,D2 ;Y Pos erhöhen
MOVE.L #70000a,(A1)+ ;breite höhe
MOVE #92,(A1)+ ;Flags
MOVE.L #0,(A1)+ ;kein Ausschluss
LEA 16(A1),A3
MOVE.L A3,(A1)+ ; TextStruk.
MOVE.L #0,(A1)+ ;keine FüllStruk.
MOVE.L #0,(A1)+ ;kein Kommand/UnterMenue
MOVE.L #0,(A1)+ ;keine fortsetzung
MOVE.L #3020100,(A1)+ ;TextStruk/Farbe/Mode
MOVE.L #50001,(A1)+ ;X Y Pos
MOVE.L #0,(A1)+ ;Font
MOVE.L (A0)+,(A1)+ ;Text Item
MOVE.L #0,(A1)+ ;keine fortsetzung
BRA ITEMLOOP ;nächstes Item
MenuEnd: MOVE.L #0,-54(A1) ;nächster Punkt löschen

```



```

TST.L (A0)+ ;Tabelle erhöhen
TST.L (A0) ;noch ein Menue ?
BEQ SetMenu1 ;nein fertig
MOVE.L A1,(A2) ;nächstes Menue
BRA MenuLoop ;aufgehts
SetMenu1: MOVE.L WindowBase,A0 ;Menue Initialisieren
LEA Menu,A1
MOVE.L #276,D0
MOVE #193,12(A1,D0)
MOVE.L #ffff,14(A1,D0)
ADDI #54,D0
MOVE #93,12(a1,D0)
MOVE.L #ffff,14(a1,D0)
ADDI #54,D0
MOVE #93,12(a1,D0)
MOVE.L #ffff,14(a1,D0)
ADDI #54,D0
MOVE #93,12(a1,D0)
MOVE.L #ffff,14(a1,D0)
LINKSYS SetMenuStrip,IntuitionBase
RTS

```

***** DATEN UND STRUKTUREN *****

```

Align.L
FileHD: DS.L 1
OldDir: DS.L 1
CDDir: DS.L 1
IntuitionBase: DS.L 1
WBenchMsg: DS.L 1
Initial_SP DS.L 1
DosBase: DS.L 1
GfxBase: DS.L 1
WindowBase: DS.L 1
UserPort: DS.L 1
RastBase: DS.L 1
ImagePic1: DS.L 1
Kanfng: DS.L 1
Planebuf: DS.L 6
Bildanfng: DS.L 1
PaarZL: DS.L 1
BUFFER: DS.L 2
HiZahlTab: DS.L 1
HiTextTab: DS.L 1
HiTextSpe: DS.L 1
ZeileNum: DS.W 1
MousePosX: DS.W .1 ;Speicher für
MousePosY: DS.W .1 ;Werte im
MousePosX1: DS.W 1 ;Programm_lauf
MousePosY1: DS.W 1
HiZL: DC.W 999
ZugNR: DS.W 1
SpPaare: DS.W 4
Karte: DS.B 1
Karte1: DS.B 1
ImageNum: DS.B 1
ImageNum1: DS.B 1
Zeiger: DS.B 1
PaarZähler: DS.B 1
KartenPaar: DS.B 1
Spieler: DC.B 0
SpielerNR: DS.B 1
Gp: DS.B 1
GePackt: DS.B 1

```

***** ScreenStruktur *****

```

Align.W
Screen: DC.W 0 ;obere linke Ecke x
DC.W 0 ;obere linke Ecke y
DC.W 640 ;Screen breite x Punkte
DC.W 256 ;Screen höhe y Punkte
DC.W 4 ;BitPlanes
DC.B 0 ;Pen
DC.B 0 ;Back Pen
DC.W $8000 ;ViewMode
DC.W 15 ;CustomScreen (1=Workb)
DC.L 0 ;Font (0=Systemfont)
DC.L STitel ;Titel
DC.L 0 ;Gadgets
DC.L 0 ;BitMaps

```

***** WindowStruktur *****

```

Window: DC.W 0,0,640,256 ;x/y -Offset, Breite, Höhe
DC.B 3,2 ;Pen_farbe
DC.L $41188 ;Flags
DC.L $9a00 ;IDCMP Flags
DC.L 0,0 ;Gadgets , Checkmark
DC.L PL2 ;Textzeiger
ScreenBase: DS.L 1 ;Screenzeiger
DC.L 0 ;Bitmap
DC.W 0,0,0,0 ;min/MAX Breite/Höhe
DC.W 15 ;Type 1=workb 15=custom

```

***** REQUESTER STRUKTUR *****

```

Requester: DC.L 0 ;Vorgänger
DC.W 200,100 ;Links/Oben
DC.W 240,40 ;Breite/Höhe
DC.W 0,0 ;Pointer bezugspunkt
DC.L StGadget ;Gadget
DC.L 0 ;Börder
DC.L StringText ;Intui.Text Struk.
DC.W $4 ;Flags
DC.B 1,0 ;Backpen/pad
DC.L 0 ;Layerstruk.
DS.B 32 ;Pad1 System use
DC.L 0 ;Custom BitMap
DC.L 0 ;RWindow
DS.B 36 ;Pad2 System use

```

***** String Gadget Struktur *****

```

STGADGET: DC.L 0 ;Nächstes Gadget
DC.W 20,20,200,10 ;X/Y Pos// Breit/höhe
StOn: DC.W $0,$204,$1004 ;Flags//Activation//Typ
DC.L 0 ;Borderstruk.
DC.L 0 ;select Zeichnung
DC.L 0 ;Textstruk.
DC.L 0 ;Execlute
DC.L StrInfo ;Stringinfostruk.
DC.W 1 ;Gadget ID
DC.L 0 ;User Data

```

```

StrInfo: DC.L StrBuffer ;Buffer
DC.L 0 ;undo Buffer
ReqC: DC.W 2 ;Cursor-Pos
ReqL: DC.W 50 ;Max.Char
DC.W 0,0,0,0,0,0 ;
DC.L 0,0,0

```

```

StringText: DC.B 0,1,1,0 ;Textstruk.
DC.W 8,8
DC.L 0
StrT: DC.L StrText
DC.L 0

```

```

StrBuffer: DC.B "MEMO_PIC",0
BLK.B 50
StrBuffer1: DC.B "MEMORY-LEER-BILD",0
BLK.B 50

```

```

StrText: DC.B "BITTE PFAD/FILENAME EINGEBEN",0
Align.W
StrText2: DC.B ">>> BITTE NAMEN EINGEBEN <<<",0
Align.W

```

***** Menue Daten *****

```

MenuTabelle: DC.L Menu1 ;MenueTitel 1
DC.L MP11,MP12,MP13,MP14 ;Item Punkte
DC.L 0 ;Menue 1 Endeflag
DC.L Menu2
DC.L MP21,MP22,MP23,MP24
DC.L 0
DC.L Menu3
DC.L MP31,MP32
DC.L 0
DC.L 0 ;Ende Flag ganzes Menue

```

***** Texte zu den Menues und Items *****

Listing 1. (Fortsetzung)




```

Menu1: DC.B " MEMORY ",0
MP11: DC.B " ABOUT ",0
MP12: DC.B " START ",0
MP13: DC.B " HI-SCORE ",0
MP14: DC.B " QUIT ",0

Menu2: DC.B " PLAYER ",0
MP21: DC.B " 1 SPIELER ",0
MP22: DC.B " 2 SPIELER ",0
MP23: DC.B " 3 SPIELER ",0
MP24: DC.B " 4 SPIELER ",0

Menu3: DC.B " KARTEN ",0
MP31: DC.B " IFF - LADEN ",0
MP32: DC.B " IFF - SAVEN ",0

;***** IMAGESTRUKTUR *****
Align.W
ImageDaten:
XImage: DS.W 1 ;x und y offset
YImage: DS.W 1
BImage: DC.W 48 ;Größe in Pixel
Half: DC.W 32
DC.W 2 ;Bitplanes (depht-farben)
KImage: DC.L AH_Karte ;Zeiger auf Image
DC.B 3 ;in welcher Bitplane
DC.B 0 ;Bitplane ausfüllen Farbe
DC.L 0 ;nächste ImageSTRUK.

Image2:
Image2x: DS.W 1
Image2y: DS.W 1
DC.W 48,32
DC.W 1
I2: DS.L 1 ;Leer_Karte
DC.B 1,0
DC.L 0

Pic2:
Pic2X: DS.W 1
Pic2Y: DS.W 1
DC.W 48,32
DC.W 4
PicAdr: DS.L 1 ;Bilder_Karten
DC.B 15,0
DC.L 0

;***** Textstruktur *****
Plt: DC.B 3,2,1,0 ;PEN,BackPEN,MODE,?
DC.W 00,00 ; X und Y Pos
DC.L 0 ;FontStruk. 0=Systemfont
TPL: DS.L 1 ;TextZeiger **
DC.L 0 ;Nächste Textstruk.0=keine

FTS: DC.B 1,0,1,0 ;PEN,BackPEN,Mode,?
DC.W $C8,$74 ; X und Y Pos
DC.L 0 ;FontStruk. 0=Systemfont
FT: DC.L Fehlertext ;TextZeiger **
DC.L 0 ;Nächste Textstruk.0=keine

CMap: DC.W $000,$ddd,$e00,$fc8
DC.W $0d0,$f0f,$0ff,$0f0
DC.W $fd8,$d84,$ff0,$ddd
DC.W $879,$ecb,$999,$4ae
DC.W $000,$fff,$e00,$feb

BildData: DC.L $464f524d,$00001df8,$494c424d,$424d4844
DC.L $00000014,$028000e8,$00000000,$04020100
DC.L $000000a0b,$028000e8,$43414d47,$00000004
DC.L $00008000,$434d4150,$00000030,$000000d0
DC.L $d0d0e000,$00f0e080,$00d000f0,$00f000f0
DC.L $f00f0000,$f0d080d0,$8040f0f0,$00d0d0d0
DC.L $807090e0,$e0b09090,$9040a0e0
DC.L $424f4459,$00001d8c

PL2: DC.L $d8,PL21
SpT: DC.L $5200e4,SpT1
SpT2: DC.L $6000f4,SpT21
Titel: DC.L $d80004,Titel1
HiT: DC.L $1b000e4,HiT1

```

```

ZugT: DC.L $f000e4,ZugT1
Intui: DC.B "intuition.library",0
Dos: DC.B "dos.library",0
Gfx: DC.B "graphics.library",0
FileHI: DC.B "MEMORY-HI",0
HiName: DC.B "> <-> HI- SCORE NAME BITTE < <<<",0,0
FehlerText: DC.B " F E H L E R *** D O S ",0
FehlerText1: DC.B " KEIN SPEICHER FREI ",0
FehlerText2: DC.B " FALSCHES BILD (FORMAT) ",0
SpT1: DC.B " SPIELER ",0
SpT3: DC.B "****",0
PL21: DC.B " ",0
SpT21: DC.B " PAARE ",0
Titel1: DC.B " A M I G A - M E M O R Y ",0
HiT1: DC.B " TAGES-HI.: 000",0
ZugT1: DC.B " ZUG.NR.: 000",0
STitel: DC.B "SCREEN",0
SB2T12: DC.B "VERSION 2.0 14. 4. 1989",0
SB2T13: DC.B "GESCHRIEBEN VON ANDREAS HERCKNER",0
SB2T14: DC.B "COPYRIGHT BERLIN",0
ImageZeichnen: DC.B 0,0

```

```

Align.W
SB2T1: DC.L $d80064,Titel1
DC.L $d80074,SB2T12
DC.L $b80084,SB2T13
DC.L $f80094,SB2T14

```

;***** DATA muß ins ChipRam Gelinkt werden *****

```

Data
AH_Karte: DC.L $00000000,$00000000,$00000000,$3fffffff
DC.L $fffe3fff,$fffffffe,$30000000,$000e3000
DC.L $0000000e,$30000000,$000e3000,$0000000e
DC.L $30000000,$000e3000,$0000000e,$30000000
DC.L $000e3000,$0000000e,$30000000,$000e3000
DC.L $0000000e,$30000000,$000e3000,$0000000e
DC.L $30000000,$000e3000,$0000000e,$30000000
DC.L $000e3000,$0000000e,$30000000,$000e3000
DC.L $0000000e,$30000000,$000e3000,$0000000e
DC.L $30000000,$000e3000,$0000000e,$30000000
DC.L $000e3000,$0000000e,$3fffffff,$fffe3fff
DC.L $fffffffe,$00000000,$00000000,$00000000
DC.L $00000000,$00007fff,$fffffffe,$7fffffff
DC.L $fffe7fff,$fffffffe,$7fffffff,$fffe7fff
DC.L $fffffffe,$7fffffff,$fffe7fff,$fffffffe
DC.L $7fffffff,$fffe7fff,$fffffffe,$7fffffff7f
DC.L $fffe7fff,$fe7ffffe,$7fffe1c7f,$fffe7fff
DC.L $8e7ffffe,$7ffffc671,$fffe7fff,$a26e7ffe
DC.L $7fff705f,$3ffe7ffd,$e83f3ffe,$7ff9dc7f
DC.L $3ffe7ff9,$7e3f3ffe,$7ffdfelf,$3ffe7fff
DC.L $fe7f3ffe,$7ffffe7e,$7ffe7fff,$fffffffe
DC.L $7fffffff,$fffe7fff,$fffffffe,$7fffffff
DC.L $fffe7fff,$fffffffe,$7fffffff,$fffe7fff
DC.L $fffffffe,$7fffffff,$fffe0000,$00000000

```

```

Ziegel_Karte: DC.L $00000000,$00000000,$00000000,$02020800
DC.L $00000100,$00000400,$00400040,$00000000
DC.L $00000000,$00000000,$00000020,$01001010
DC.L $00000000,$04800400,$08000000,$00000000
DC.L $00000000,$fffffffd,$fffffffd,$fffeffff
DC.L $fffffffd,$fffffffd,$fffffffd,$fffffffd
DC.L $00000000,$fffdffff,$fffdffff,$fffdffff
DC.L $fffdffff,$fffdffff,$fffdffff,$fffdffff

```

```

Pointer: DC.W 0,0,$8810,$18,$5030,$30,$50,$78,$5090,$d8
DC.W $8910,$198,$10,$18,$10,$18,$10,$18,$10,$18,0,0
DC.W 0,0,$8878,$fe,$5084,$186,$84,$186,$5008,$c,$8810
DC.W $18,$20,$30,$40,$60,$80,$fe,$1fe,$1fe,0,0
DC.W 0,0,$88fe,$fe,$5102,$1fe,$2,$106,$5004,$c
DC.W $8878,$78,$4,$c,$2,$106,$102,$1fe,$fe,$fe,0,0
DC.W 0,0,$8810,$18,$5020,$30,$40,$60,$5088,$c,$8908
DC.W $18c,$208,$3ff,$7ff,$7ff,$8,$c,$c,$c,0,0

```

```

BSS
ImageTabelle: BLK.W 40
Menu: BLK.W 320
HiSpeicher: BLK.W 283

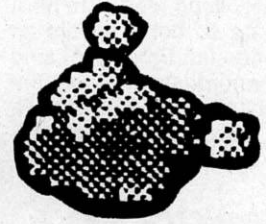
```

End

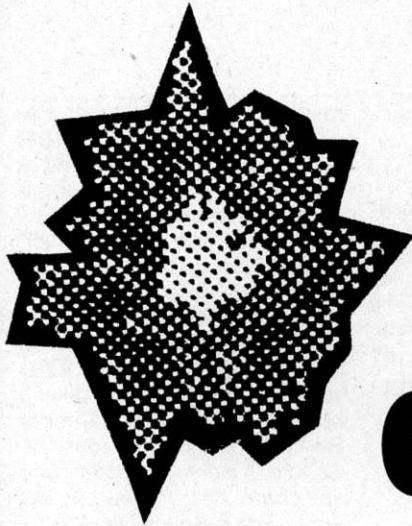
Listing 1. (Schluß)



Strategiespiel zu zweit



Krieg den Kernen



Die Fachwelt steht Kopf. Ein Durchbruch der Wissenschaft. Die erste strahlungsfreie Kettenreaktion im Homecomputer.

Von Hans Waldhäusel

Spaß beiseite, der AMIGA kann viel – es gibt sogar Leute, die davon überzeugt sind, daß es (fast) nichts gibt, was dieser Super-Computer nicht kann. Aber Kernspaltung? Nun, zumindest als Spiel gibt's das wirklich. Jetzt! Hier! Im AMIGA-Sonderheft!

»REAKTOR« ist ein Spiel für zwei. Ein Strategie-Spiel, bei dem die Action nicht zu kurz kommt. Ein Spiel, bei dem der Computer nur Werkzeug ist und gar nicht Gegner sein kann. Wenn Sie Gefallen an »REAKTOR« gefunden haben, kommen Sie nicht umhin, sich nach einem menschlichen Spielpartner (-gegner) umzusehen – einsam vor dem Computer sitzend hat es gar keinen Sinn, »REAKTOR« zu starten. Es macht wirklich nur Spaß, wenn man einen Partner hat (oder spielen Sie gegen sich selbst Schach?).

Auf einem quadratischen Spielfeld sind 36 Atomkerne gleichmäßig verteilt (Bild 1). Die Spieler können nun abwechselnd – durch Anklicken – jeweils einen Kern pro Spielzug für sich in Besitz nehmen. Die Besitzverhältnisse werden durch Einfärbung der Kerne in der Farbe des Spielers (Rot oder Blau) dokumentiert. Gleichzeitig wird mit jedem Anklicken dem Kern ein Neutron implantiert, wodurch die Masse des Kerns natürlich ansteigt. Und wie wir alle im Physik-Unterricht gelernt haben (oder noch ler-

nen werden): Wenn ein Atomkern seine kritische Masse erreicht – BUMM.

Bei REAKTOR gelten folgende »Naturgesetze«: Ein Kern erreicht dann seine kritische Masse, wenn er mit so vielen Neutronen gefüttert wurde, als er Nachbarn hat (wobei nur diejenigen zählen, die im rechten Winkel zum jeweiligen Kern stehen). Wir haben in Bild 1 zur Verdeutlichung einige signifikante Kerne mit ihren kritischen Neutronenzahlen gekennzeichnet. Macht es jetzt BUMM, so schleudert der explodierende Kern seine ihm zuvor aufge-

nommene Aufnahme des Fremdneutrons seine kritische Masse, kommt es auch bei ihm zu einer Reaktion – BUMM-BUMM! Und so weiter. Ein Spiel ist dann beendet, wenn es nur noch unbesetzte und/oder besetzte Kerne einer Farbe auf dem Spielfeld gibt, wenn also alle Kerne des Gegners übernommen wurden. Übrigens werden explodierte Kerne automatisch durch besitzneutrale ersetzt, so daß es schon eine Weile dauern kann, bis ein Spiel beendet ist. Erschwerend kommt noch hinzu, daß Sie nicht unbegrenzt Zeit zum Überlegen haben. Zu

es aus! »Mensch-ärgere-dich-nicht« ist auch simpel; und wie viele Stunden haben Sie damit schon verbracht? Uns hat »REAKTOR« ein wenig an Halma erinnert, weil man fremde »Kernstraßen«, die der Gegner für eine Kettenreaktion aufgebaut hat, bei klugem Taktieren für sich benutzen und übernehmen kann.

Und glauben Sie uns: Es gehört schon etwas Geistesakrobatik dazu, sich eine Kettenreaktion soweit vorzustellen, daß auch wirklich das geschieht, was man erreichen möchte. Allzu leicht passiert es, daß die eigene Reaktion in einem Stadium zum Erliegen kommt, in dem der Gegner durch eine geschickt platzierte Explosion alle soeben verlorenen Kerne wieder zurückerobert – und noch etliche mehr.

So einfach das Spiel, so kurz das Programm (Listing 1). Sie sehen, man braucht für ein unterhaltsames Spiel nur eine gute Idee und nicht unbedingt seitenlange Programme. Vermissen Sie lange DATA-Kolumnen oder ein nachzuladendes File für die Grafik? Sie vermissen richtig! Die Grafik wird »per Zufall« und mit Hilfe der Trigonometrie jedesmal neu entwickelt. Das spart Ihnen Tipparbeit und dem AMIGA Speicherplatz. Erkauft werden diese Vorteile durch eine etwas lästige Wartezeit beim ersten Programmstart. Aber wer früher einen C64 ohne Beschleuniger in Betrieb hatte, wird über die 35 Sekunden milde lächelnd hinwegsehen. Wir haben für Sie die Grafik-Routine aus dem Hauptprogramm herausgelöst und als lauffähiges Demo-Programm in Listing 2 abgedruckt, um Ihnen ein eventuelles Einbinden in Ihre eigenen Programme zu erleichtern. Mit der ausführlichen Kommentierung dürfte das keine Probleme bereiten. Die Grafik-Berechnungen erfolgen natürlich wesentlich



Bild 1. »REAKTOR« – viel Spaß mit einfachen Mitteln

pfropften Neutronen zu seinen Nachbarkernen, die diese in ihr Atomgefüge aufnehmen, so als wären sie angeklickt worden. So eine Neutronenaufnahme bedeutet aber auch, daß der beschossene Kern in den Besitz jenes Spielers übergeht, dessen Kern explodiert ist – die fremden Neutronen infizieren ihn gewissermaßen! Erreicht ein beschossener Kern durch

Spielbeginn müssen Sie die zulässige Bedenkzeit je Zug eingeben – und wenn diese abgelaufen ist, ohne daß Sie sich für einen Zug entschieden haben, kommt Ihr Gegner dran. Netterweise erinnert Sie ein akustisches Signal in den letzten fünf Sekunden Ihrer Zeit daran, endlich etwas zu unternehmen.

So simpel sich dieses Spielprinzip anhört – probieren Sie

schneller, wenn man dem Kern nicht eine so hohe Anzahl an Nukleonen (im Programm sind es 40) verordnet. Vielleicht findet sich auch ein REAKTOR-ianer, der die trigonometrischen Berechnungen in ein

Reaktor als Baukasten

Maschinensprache-Subprogramm umsetzt oder einen LIBRARY-Aufruf einbindet? – einen entsprechenden Nachtrag werden wir gerne veröffentlichen. Eine leichte Beschleunigung (Größenordnung ca. 20 Prozent) erreichen Sie, wenn Sie folgende Zeilen einbauen (diese Programmzeilen sind nicht Teil des Originalprogramms und daher auch nicht im File »REAKTOR« auf der Leserservice-Diskette enthalten):

```
DECLARE FUNCTION Find
Task& LIBRARY
LIBRARY "exec.library"
task&=FindTask&(0)
CALL SetTaskPri&
(task&,120)
```

Wenn Sie diese vier Programmzeilen an den Anfang des Programms stellen und auf Ihre »REAKTOR«-Diskette noch das File »exec.bmap« kopieren, so erhöhen Sie die Task-Priorität und lassen die Berechnungen daher auch schneller durchführen. Wichtig dabei ist, daß Sie ab nun keine zeitkritischen Vorgänge, wie z.B. Diskettenzugriffe, durchführen, da Sie bei der im Beispiel angegebenen hohen Task-Priorität von +120 bereits in die Bereiche des Betriebssystems hineinfuschen. Deshalb müssen Sie auch die Prioritätsverhältnisse

sobald als möglich wieder normalisieren, und zwar mit
CALL SetTaskPri(task&,0)
wobei unserer Meinung nach der empfohlene Einfügepunkt im Programm so aussieht:

```
.....
' einlesen
DATA 0,-1,1,0,0,1,-1,0
COLOR 2,0:GOSUB grafix
'Grafik erzeugen
CALL SetTaskPri(task&,0)
newgame: 'Neues Spiel
CLS
FOR i=0 TO 15
.....
```

Nochmals ca. 10 Prozent können Sie holen, wenn Sie die Kommentare entfernen oder auf das absolute Minimum kürzen. Die Redaktion ist der Meinung, daß in diesem Fall Information und Programm-Doku-

mentation vor Ablaufgeschwindigkeit zu setzen sind und hat daher die Kommentare nicht ausgefiltert – wie wir glauben, auch in Ihrem Interesse.

Um in den Genuß von »REAKTOR« zu kommen, müssen Sie einen von zwei möglichen Wegen beschreiten. Entweder Sie kopieren den Inhalt der Schublade »REAKTOR« von der Leserservice-Diskette auf Ihre Spiele-Diskette oder Sie tippen Listing 1 ab (der Aufwand hält sich in Grenzen) und speichern das Programm auf Ihrer Basic-Spiele-Diskette – fertig. Das Spiel starten Sie entweder durch Anklicken des Icons oder – aus dem CLI – mit run AmigaBASIC "Reaktor"

Und nicht vergessen: rechtzeitig Freundin oder Freund zum Spielen einladen. Viel Spaß!

```
DEFINT ne, pl2, pl, ex, ey, xp, yp, xoff, yoff, i, vv, aa, xx, yy,
vz, vz2, ke, add
SCREEN 1,640,256,4,2 'Hires-SCREEN mit 16 Farben, ohne Rand
WINDOW 2,"", (0,0)-(631,230),0,1 'oder Gadgets beim WINDOW !
RANDOMIZE TIMER 'Zufallsgenerator initialisieren
```

```
DIM r%(15), g%(15), b%(15), anz%(10,10), max%(10,10) 'Feldvariablen
DIM kern%(600,2), expl%(600), wav%(255)
```

```
pi=3.1415 'Konstante
FOR i=0 TO 255
wav%(i)=INT(RND*256-128) 'Zufallswellenf. fuer Rauschen
```

```
NEXT
FOR i=0 TO 15
READ r%(i), g%(i), b%(i) 'Farbwerte in Variablen lesen
PALETTE i,0,0,0 'Alle Farben schwarz
```

```
NEXT
PALETTE 2,5/15,0,12/15 'Bis auf Schriftfarbe
COLOR 2,0:LOCATE 14,31:PRINT "Bitte Warten..."
```

'Daten fuer RGB-Farbwerte im Bereich von 0 bis 15

```
DATA 0,0,0,0,0,0
DATA 9,8,5,13,12,9
DATA 13,13,13,10,10,10
DATA 7,7,7
DATA 14,5,5,10,5,5
DATA 6,5,5
DATA 5,5,14,5,5,10
DATA 5,5,6,14,13,0
DATA 0,0,5,0,5,11
```

```
FOR i=0 TO 3
READ xv(i), yv(i) 'Richtungsvektoren (x/y) fuer
NEXT 'rel.Positionen der Neutronen
' einlesen
```

```
DATA 0,-1,1,0,0,1,-1,0
COLOR 2,0:GOSUB grafix 'Grafik erzeugen
```

```
newgame: 'Neues Spiel
CLS
FOR i=0 TO 15
PALETTE i,r%(i)/15,g%(i)/15,b%(i)/15 'Farben richtig setzen
NEXT
COLOR 2,0:LOCATE 14,15
PRINT "Welche Bedenkzeit fuer jeden Zug (in Sekunden)...";
LINE INPUT t$:mt=VAL(t$):CLS 'max.Bedenkzeit festlegen
```

```
xan=6:yan=6:xoff=70:yoff=10 'Groesse und Position des Spielfelds
FOR xx=0 TO xan-1 'Spielfeld aufbauen
FOR yy=0 TO yan-1
xp=20+xx*60:yp=1+yy*30 'Bildschirmpos. des Kerns
max%(xx,yy)=4+(xx=0)+(xx=(xan-1))+(yy=0)+(yy=(yan-1))
anz%(xx,yy)=0 'max.Anzahl der Neutronen dieses Kerns
```

```
GOSUB kern 'berechnen (Vergl. geben -1 wenn wahr !)
```

```
NEXT
LOCATE 1,18:COLOR 2,0:PRINT "REAKTOR (c) 1988 by T.Bagdonat"
pl=1
LOCATE 25,28:COLOR 2,0:PRINT "Kerne Neutronen"
LOCATE 27,15:COLOR 11,0:PRINT "Player 1" 'Bildschirmaufbau
LOCATE 29,15:COLOR 8,0:PRINT "Player 2";
ke(0)=0:ke(1)=0:ne(0)=0:ne(1)=0:GOSUB kene
newplay: 'Naechster Zug
GOSUB kene:pl=ABS(pl-1):add=pl*2-1 'pl immer 0 oder -1
COLOR ABS(pl-1)*3+8:LOCATE 10,64:PRINT "Player";pl+1;
SOUND pl*200+250,2,60,1
LOCATE 14,64:COLOR 2,0:PRINT "Time"; 'Timerbehandlung
t=mt:COLOR 3,0:PRINT t 'Jede Sekunde ein Sprung nach
ON TIMER(1) GOSUB zeit:TIMER ON 'zeit
```

```
warte:
MOUSE ON:ON MOUSE GOSUB click:flag=0 'Mausbehandlung initialisieren
WHILE flag=0 AND t>0:WEND 'flag s.click
mx=MOUSE(1):my=MOUSE(2) 'Position des Mauszeigers
IF t=0 THEN newplay 'Wenn Zeit vorbei,dann weiter
```

```
xx=INT((mx-xoff)/60):yy=INT((my+yoff)/30) 'Mauspos.in Kernpos.
IF xx<0 OR xx>=xan OR yy<0 OR yy>=yan THEN warte 'Ausserhalb ?
vz=SGN(anz%(xx,yy)) 'Wem gehoert Kern ?
IF vz<>0 AND vz<>0 THEN warte 'Nicht dem Spieler ?
TIMER OFF 'Timerbehandlung abschalten
```

```
ne(pl)=ne(pl)+1:IF anz%(xx,yy)=0 THEN ke(pl)=ke(pl)+1
anz%(xx,yy)=anz%(xx,yy)+add:aender=0:GOSUB kern 'Neutron dazu
```

```
vorne: 'Hauptschleife fuer Kettenreaktion !
GOSUB kene:ne=ne(0)+ne(1)
pl2=ABS(pl-1):IF ke(pl2)=0 AND ne>1 THEN GOTO sieg 'Gegner k.o.
FOR ey=0 TO yan-1
FOR ex=0 TO xan-1 'Kern mit kritischer Masse suchen
IF ABS(anz%(ex,ey))<max%(ex,ey) THEN NEXT:NEXT:GOTO newplay
xp=20+ex*60+xoff:yp=ey*30+yoff 'Gefunden und Pos. auf Bildschirm
PUT (xp,yp),expl%,PSET:GOSUB exso 'Explosionsgrafik anzeigen
vv=0:aa=max%(ex,ey) 'vv ist Nr. des akt. Vektors
```

```
weiter3:
xx=ex+xv(vv):yy=ey+yv(vv) 'Pos des zu besetzenden Kerns ber.
IF xx<0 OR xx>=xan OR yy<0 OR yy>=yan THEN weiter2 'Ausserhalb ?
aender=0:vz2=SGN(anz%(xx,yy)) 'Wem gehoert Kern ?
IF vz2<>vz THEN 'Wenn nicht dem Spieler, dann
anz%(xx,yy)=anz%(xx,yy):aender=1 'Kern-und Neutr. neu ber.
ke(pl)=ke(pl)+1:ne(pl)=ne(pl)+ABS(anz%(xx,yy))
IF vz2<>0 THEN ke(pl2)=ke(pl2)-1:ne(pl2)=ne(pl2)-
```



```

ABS (anz% (xx,yy))
END IF
anz%(xx,yy)=anz%(xx,yy)+add:ne(pl)=ne(pl)+1
GOSUB kern 'Kern zeichnen
aa=aa-1:anz%(ex,ey)=anz%(ex,ey)-add 'Neutr.anz.alter Kern -1
IF aa=0 THEN 'Alle Neutr.verteilt ?
xx=ex:yy=ey:GOSUB kern:ne(pl)=ne(pl)-max%(ex,ey)
IF anz%(ex,ey)=0 THEN ke(pl)=ke(pl)-1 'Dann alten Kern zeichnen
GOTO vorne 'Naechster Expl.durchlauf und Neutr. bzw.Kernzahl
END IF 'neu berechnen
weiter2:
vv=vv+1:GOTO weiter3 'Naechsten Richtungsvektor bestimmen
END

grafix: 'Grafik erzeugen
FOR p=0 TO 2 'Drei Kernarten (0..2)
gx=40:gy=20+p*30 'Pos. auf Bildschirm
FOR m=1 TO 40 'Jeder Kern 40 Nukleonen
wi=wi+.35:1=RND*10+1 'Winkel erhoehen und Radius waehlen
x=gx+COS(wi)*1
y=gy+SIN(wi)*(1/2):r=3 'Pos.auf Kreis mit zufaell.Radius
co=INT(RND*1.5+.5) 'Farbe zufaell.ermitteln
IF p=2 THEN co=0 'p=2 ist neutraler Kern
IF co=1 THEN co=co+p 'Sonst Farbe entsprechend p ber.
GOSUB kugel 'Nukleon zeichnen
NEXT
GET (gx-22,gy-14)-(gx+22,gy+14),kern%(0,p) 'Grafik ausschneiden
NEXT
gx=40:gy=110:fa(1)=9:fa(2)=8:fa(3)=13 'Farben u.Pos. d. Ex.grafix
FOR mm=1 TO 3 'Drei Ringe
pl=(3-mm)*4+2:mu=(4-mm)*4 'max.und min. Radius
fa=fa(mm):wi=0:ax=gx+mu:ay=gy 'Zu verw. Farbe und Startpos.
FOR m=1 TO 30 '30 Pos.auf Ring
wi=wi+.21 '30x0.21=2pi (Einmal rum)
nx=gx+COS(wi)*(RND*mu+pl)
ny=gy+SIN(wi)*((RND*mu+pl)/1.5) 'neue Pos.im Bereich pl..mu
LINE (ax,ay)-(nx,ny),fa:ax=nx:ay=ny 'alte+neue Pos. verbinden
NEXT
LINE (nx,ny)-(gx+mu,gy),fa 'Zurueck zum Anfang
PAINT (gx,gy),fa 'Ring ausmalen
NEXT
GET (gx-22,gy-14)-(gx+22,gy+14),expl% 'Expl.grafix ausschneiden
RETURN

kugel: 'Nukleon zeichnen
fa=3*co+5:CIRCLE (x,y),3,fa 'Echte Farbe ber.und Kreis
PSET (x,y),fa:PSET(x-1,y),fa 'Statt ausmalen mit Punkten
PSET(x+1,y),fa:PSET (x-2,y),fa 'fuellen (schneller !!)
CIRCLE (x,y),r,fa-1,.4*pi,1.3*pi 'Schattierung durch farbl.
CIRCLE (x,y),r,fa+1,1.5*pi,.5*pi 'abgestufte Halbkreise
RETURN

kern: 'Kern zeichnen
xp=20+xx*60+xoff:yp=yy*30+yoff 'Bildschirmpos.ber.
anz=anz%(xx,yy) 'Anzahl der zus.Neutronen
IF ABS(anz)>=2 AND aender=0 THEN rand 'Kern nicht neu zeichnen
IF anz=0 THEN co=2:GOTO draw 'Farbe aus VZ von Neutr.anzahl
IF anz>0 THEN co=0:ELSE co=1
draw: 'Entspr.Kerngrafik auf Bild-
PUT (xp,yp),kern%(0,co),PSET 'schirm bringen
IF anz=0 THEN RETURN 'Keine zus. Neutronen,dann Ende
rand: 'Zusaetzhliche Neutr. zeichnen
v=0:gx=xp+22:gy=yp+14:i=0
again:
xn=xx+xv(v):yn=yy+yv(v) 'Pos.des Neutrons ber.
IF xn<0 OR xn>=xan OR yn<0 OR yn>=yan THEN weiter 'Ausserhalb ?
i=i+1:x=gx+xv(v)*18:y=gy+yv(v)*9 'Bild.pos.ber.
co=0:GOSUB kugel 'Nukleon zeichnen
weiter:
v=v+1:IF i=ABS(anz) THEN RETURN:ELSE GOTO again 'Naechstes Neutr.

click: 'Wird aufgerufen,wenn ein Maus-
a=MOUSE(0):flag=-1:MOUSE OFF 'ereignis stattfindet und
RETURN 'signalisiert das durch flag

zeit:
LOCATE 14,68:t=t-1:COLOR 3,0:PRINT t 'Wird jede Sekunde aufgerufen
IF t<5 THEN SOUND 700,1,100,1 'und zaehlt die Zeit runter
RETURN

kene: 'Schreibt die Anzahl der Kerne
FOR ii=0 TO 1 'bzw.Neutr.auf den Screen

```

```

COLOR 3,0:LOCATE 27+ii*2,29:PRINT ke(ii);
LOCATE 27+ii*2,38:PRINT ne(ii);
NEXT
RETURN

sieg: 'Wird am Spielende angesprungen
LOCATE 27+pl*2,45:COLOR 2,0:PRINT "hat gewonnen !!";
co=pl*3+8:a=MOUSE(0)
weiter5:
FOR i=5 TO 15 'Farblauf in Goldartigem
PALETTE co,i/17,1/19,1/24 'Ton fuer Siegerfarbe rauf
PALETTE co-1,i/15,1/17,1/22
NEXT
FOR i=15 TO 5 STEP -1 '...und wieder runter
PALETTE co,i/17,1/19,1/24
PALETTE co-1,i/15,1/17,1/22
NEXT
IF MOUSE(0)=0 THEN weiter5 'Bis Mausbutton gedruickt...
GOTO newgame 'Dann neues Game

exso: 'Expl.sound
WAVE 0,wav% 'Rauschwellen setzen
FOR i=0 TO 10
PALETTE 0,i/15,1/17,0:PALETTE 1,i/15,1/17,0 'Hintergr aufleuchten
SOUND INT(RND*20+10),.4,(10-i)*25 'Sound mit Zuf.freq
NEXT
PALETTE 0,0,0,0:PALETTE 1,0,0,0 'Hintergrund wieder schwarz
RETURN

```

Listing 1. »REAKTOR« - Kettenreaktion im Homecomputer

```

SCREEN 1,640,250,4,2
WINDOW 2,"",0,0)-(631,230),0,1
DIM kern%(600,2),expl%(600)

grafix: 'Grafik erzeugen
FOR p=0 TO 2 'Drei Kernarten (0..2)
gx=40:gy=20+p*30 'Pos. auf Bildschirm
FOR m=1 TO 40 'Jeder Kern 40 Nukleonen
wi=wi+.35:1=RND*10+1 'Winkel erhoehen und Radius waehlen
x=gx+COS(wi)*1
y=gy+SIN(wi)*(1/2):r=3 'Pos.auf Kreis mit zufaell.Radius
co=INT(RND*1.5+.5) 'Farbe zufaell.ermitteln
IF p=2 THEN co=0 'p=2 ist neutraler Kern
IF co=1 THEN co=co+p 'Sonst Farbe entsprechend p ber.
GOSUB kugel 'Nukleon zeichnen
NEXT
GET (gx-22,gy-14)-(gx+22,gy+14),kern%(0,p) 'Grafik ausschn.
NEXT
gx=40:gy=110:fa(1)=9:fa(2)=8:fa(3)=13 'Farben+Pos.der Expl.
FOR mm=1 TO 3 'Drei Ringe
pl=(3-mm)*4+2:mu=(4-mm)*4 'max.und min. Radius
fa=fa(mm):wi=0:ax=gx+mu:ay=gy 'Zu verw. Farbe und Startpos.
FOR m=1 TO 30 '30 Pos.auf Ring
wi=wi+.21 '30x0.21=2pi (Einmal rum)
nx=gx+COS(wi)*(RND*mu+pl)
ny=gy+SIN(wi)*((RND*mu+pl)/1.5) 'neue Pos.im Bereich pl..mu
LINE (ax,ay)-(nx,ny),fa:ax=nx:ay=ny 'alte+neue Pos. verb.
NEXT
LINE (nx,ny)-(gx+mu,gy),fa 'Zurueck zum Anfang
PAINT (gx,gy),fa 'Ring ausmalen
NEXT
GET (gx-22,gy-14)-(gx+22,gy+14),expl% 'Expl.grafix ausschneiden
END

kugel: 'Nukleon zeichnen
fa=3*co+5:CIRCLE (x,y),3,fa 'Echte Farbe ber.und Kreis
PSET (x,y),fa:PSET(x-1,y),fa 'Statt ausmalen mit Punkten
PSET(x+1,y),fa:PSET (x-2,y),fa 'fuellen (schneller !!)
CIRCLE (x,y),r,fa-1,.4*pi,1.3*pi 'Schattierung durch farbl.
CIRCLE (x,y),r,fa+1,1.5*pi,.5*pi 'abgestufte Halbkreise
RETURN

```

Listing 2. REAKTOR.graph - das Prinzip der Kern-Grafik

Grafik ist ein Markenzeichen des Amiga. Jede Menge toller Programme lassen das Thema zur Faszination werden. Doch wie können Sie Ihre mit DPaint & Co. gemalten Werke in das eigene Spiel einbauen? Fridtjof Siebert, Programmierer des Amok-Klubs aus Stuttgart, zeigt Ihnen, wie Sie Ihre Programme optisch aufwerten. Dabei unterstützen Sie die von »IFFSupport« (Listing 1) exportierten Prozeduren. Damit können Sie IFF-Bilder laden, speichern und auch Farbanimationen zeigen.

Anders als bei gängigen IFF-Unterstützungsroutinen üblich, werden bei »IFFSupport« die geladenen Daten in bekannten Formaten (Screens, Windows und BitMaps) übergeben und nicht so, wie sie im IFF-File vorliegen. Dadurch wird die Handhabung wesentlich vereinfacht (siehe unten).

Wir stellen hier die neueste Version der Routine vor, welche Sie auf der AMOK 21 finden. Sie kann mit »M2Amiga V3.2« compiliert werden. Neben Verbesserungen wurden auch einige Fehler gegenüber der alten Version von der AMOK-Diskette Nr. 6 behoben.

Die Prozeduren lassen sich nur in Modula-Bibliotheken einbauen. Im Beitrag auf Seite 30 finden Sie Tips, wie Modula- in C-Programme konvertiert werden können. Listing 2 zeigt das Definitionsmodul »IFFSupport.def«. Eine Beschreibung der Variablen und Typen finden Sie in nebenstehender Tabelle. In Listing 6 finden Sie die Definition der Maschinenroutine »LoadBody.def«, Listing 7 ist deren Definition.

Laden von IFF-Bildern

Mit der Prozedur

```
PROCEDURE ReadILBM  
(name: ARRAY OF CHAR;  
Flags: ReadILBMFlagSet;  
VAR Screen: ScreenPtr;  
VAR Window: WindowPtr):  
BOOLEAN;
```

laden Sie ein IFF-Bild mit dem Namen »name«. Tritt ein Fehler auf, wird »FALSE« zurückgegeben. Folgende Flags wählen Sie aus der Menge »ReadILBMFlagSet«:

- front: ist »front« gesetzt, wird der Screen, in den die Grafik

geladen wird, vor allen anderen Screens geöffnet, sonst dahinter - visible: ist »visible« nicht gesetzt, wird während dem Laden das Display mit »GfxMacros. OffDisplay()« abgeschaltet. Das erhöht die Ladegeschwindigkeit, besonders bei Bildern mit hoher Auflösung und/oder vielen Farben.

- dontopen: wenn gesetzt, wird kein Screen geöffnet. Der zurückgegebene »ScreenPtr« wird dann »NIL«. Die Grafik wird in eine Bitmap geladen, wobei der Zeiger auf diese in »IFFSupport.NuScreen.customBitmap« steht. Der Screen kann später mit »OpenScreen« geöffnet werden. Bei gewähltem »dontopen« muß der Speicher für die Bitplanes und die Bitmap-Struktur später wieder freigegeben werden.

- window: wenn gesetzt, wird vor dem Laden der Grafikdaten in den Screen ein Fenster mit der Größe der Grafik geöffnet. Dadurch können später Gadgets, Menüs etc. hinzugefügt werden, ohne die Grafik zu zerstören. Wenn »window« gelöscht ist, wird der »WindowPtr« zu »NIL«.

Variablen & Typen

Von IFFSupport werden folgende Variablen und Typen exportiert:

TYPE

```
IFFInfoTypePtr = POINTER TO IFFInfoType;  
IFFInfoType = RECORD  
...(* Mega-Record *)
```

END;

VAR

```
IFFInfo: IFFInfoType;
```

»IFFInfo« enthält Informationen über das zuletzt geladene Bild oder über ein zu speicherndes Bild. Wenn mehrere Bilder verwendet werden, müssen eigene Variablen vom Typ »IFFInfoType« definiert werden, da bei jedem neuen Laden »IFFInfo« überschrieben wird.

VAR

```
NuScreen: NewScreen;
```

```
NuWindow: NewWindow;
```

Diese können importiert werden, um später einen Screen mit der Grafik oder ein Window in diesem Screen zu öffnen (siehe auch »ReadILBM()«).

TYPE

```
IFFErrors = (iffNoErr, iffOutOfMem, iffOpenScreenfailed,  
iffOpenWindowfailed, iffOpenfailed, iffWrongIFF,  
iffReadWritefailed);
```

VAR

```
IFFError: IFFErrors;
```

»IFFErrors« enthält die Fehlerart, wenn das Laden oder Speichern eines Bildes erfolglos war.

iffNoErr:	kein Fehler
iffOutOfMem:	nicht genügend Speicher
iffOpenScreenfailed:	»OpenScreen« ist fehlgeschlagen
iffOpenWindowfailed:	es konnte kein Fenster geöffnet werden
iffOpenfailed:	File konnte nicht geöffnet werden
iffWrongIFF:	kein IFF-File
iffReadWritefailed:	Schreib-/Lesefehler

Pixelpracht le

Spiele auf dem Amiga leben von ihrer
Einbindung von Bildern in

Die Laderoutine unterstützt alle Auflösungen: Lores, Hires, Interlace, Hold and Modify (HAM) und Extra Half-Brite. Gepackte Bilder werden entsprechend entpackt. Da der Entpacker in Maschinensprache geschrieben ist, stehen gepackte Bilder am Ende schneller im Speicher als ungepackte.

Mit »DoCycle« kann das Color-Cycling für eine Grafik eingeschaltet werden:

```
PROCEDURE DoCycle  
(Info: IFFInfoTypePtr;  
Screen: ScreenPtr):  
BOOLEAN;
```

```
PROCEDURE EndCycle  
(Info: IFFInfoTypePtr);  
Color-Cycling, oder Farbani-
```

mation heißt, die Farben eines Bildes beginnen zu rotieren, wechseln zyklisch. Dazu wird ein »VBlank«-Interrupt initialisiert, das Cycling läuft also im Hintergrund Ihres Programms.

»DoCycle« benötigt einen »ScreenPtr« und die Adresse einer »IFFInfoType«-Variablen. Werden gleichzeitig mehrere Bilder mit Color-Cycling gezeigt, müssen auch verschiedene Variablen verwendet werden. »DoCycle()« gibt »FALSE« zurück, wenn ein Fehler auftritt. Ein solcher kommt dann zustande, wenn mehr als 32 Color-Cycling-Bilder gleichzeitig eingeschaltet sind. Da dies gewöhnlich nicht der Fall ist, kann »DoCycle« mit

```
IF DoCycle(ADR(IFFInfo),  
Screen) THEN END;  
aufgerufen werden.
```

Als die Farben laufen lernten

»EndCycle()« beendet das Color-Cycling. Dazu übergeben Sie den gleichen »IFFInfoTypePtr« wie bei »DoCycle«. »EndCycle« sollten Sie nicht vergessen, da ansonsten die Interrupts von »IFFSupport« nicht entfernt werden. Also muß »EndCycle« spätestens in der »TermProcedure«, kurz vor dem Schließen des Screens, aufgerufen werden!

Zum Speichern von IFF-Bildern sind drei ähnliche Prozeduren vorhanden:

```
PROCEDURE  
WriteILBM Screen  
(Name: ARRAY OF CHAR;  
Screen: ScreenPtr;  
Rect: RectanglePtr;  
CompressIt: BOOLEAN):  
BOOLEAN;
```

speichert den Screen unter »Name«. Trat ein Fehler (»iffOpenfailed« oder »iffReadWritefailed«) auf, wird »FALSE« zurückgegeben. Wenn »CompressIt« auf »TRUE« ist, wird ein gepacktes File erzeugt. »Rect« kann »NIL« sein, wenn der ganze Screen gespeichert werden soll. Sonst enthält es einen Zeiger auf ein »Graphics.Rectan-

leichtgemacht

Grafik. »IFFSupport« hilft bei der
Ihre eigenen Programme.

gle: »Rect.minX/minY« ist dann die linke obere und »Rect.maxX/maxY« die rechte untere Ecke des Ausschnitts. Dabei sollte man Ausschnitte, die größer als der Screen sind, vermeiden.

PROCEDURE WriteILBM

(Name:
ARRAY OF CHAR;
RP: RastPortPtr;
VP: ViewPortPtr;
Rect: RectanglePtr;
CompressIt:
BOOLEAN): BOOLEAN;

Speichert wie »WriteILBMScree« ein IFF-File. »Name«, »Rect«, »CompressIt« und der »RETURN«-Wert entsprechen dieser Prozedur. »RP« zeigt auf den »RastPort«, der die Grafikdaten enthält. »VP« weist auf einen »ViewPort«. Dieser enthält die Farben und »ViewModes« etc.

Mit »WriteILBM« speichern Sie ein Fenster folgendermaßen (das Fenster muß dabei vor allen anderen liegen):

VAR
Window: Intuition.
WindowPtr;
Rect: Graphics.

Rectangle;
OK: BOOLEAN;

...
BEGIN

WITH Window DO
WITH Rect DO
minX:=leftEdge;
minY:=topEdge;
maxX:=minX + width -1;
maxY:=minY + height -1;
END;
Error := WriteILBM
(Name,rPort,
ADR(wScreen.viewPort),
ADR(Rect),TRUE);
END;

Die dritte Prozedur, »WriteILBMAll«, kann auch Bilder mit Zusatzinformationen wie Color-Cycling etc. speichern.

PROCEDURE InitIFFInfo

(Info:
IFFInfoTypePtr;
RP: RastPortPtr;
VP: ViewPortPtr;
VAR Rect: Rectangle
Ptr);

PROCEDURE WriteILBMAll
(Name:

ARRAY OF CHAR;
Info: IFFInfoTypePtr;
BM: BitMapPtr;
FirstLine, LeftOffset:
INTEGER;
CompressIt: BOOLEAN):
BOOLEAN;

Dazu wird die Adresse einer vorher initialisierten »IFFInfoType«-Variablen benötigt. Um die Initialisierung zu erleichtern, kann die Prozedur »InitIFFInfo()« aufgerufen werden. Ihre Parameter entsprechen denen von »WriteILBM«. Danach sind die Subrecords »BMDH«, »CMAP« und »CAMG« initialisiert und deren Flags in »Info.IFFTitle« gesetzt. Weitere können »per Hand« bearbeitet werden.

»BM« zeigt auf jene Bitmap, welche die zu speichernden Planes enthält. Dabei kann »BM« auch zum Speichern neu initialisiert werden, etwa dann, wenn man eine zusätzliche »MaskPlane« speichern möchte. »FirstLine« und »LeftOffset« entsprechen »Rect.minY« und »Rect.minX«. »CompressIt« und der »RETURN«-Wert verhalten sich analog zu »WriteILBMScree«.

Beispiele

Abschließend stellen wir Ihnen noch einige Beispiele für die Anwendung der Prozeduren vor. Das Programm »ShowIFF« (Listing 3) lädt und zeigt IFF-Bilder. Es lädt das zusammen mit <SHIFT> vorher auf der Workbench angeklickte oder - beim Starten vom CLI

aus - das angegebene Bild. Bilder, die »ShowIFF« als Default-Tool haben (wird mit Workbench-Info bestimmt), können durch einfachen Doppelklick angezeigt werden. An diesem kurzen Programm können Sie die Verwendung der Ladeprozedur studieren.

»ShowCycle« (Listing 4) stellt das »ShowIFF« für Color-Cycling-Bilder dar. Normalerweise kann es auch für Bilder ohne Farbanimation verwendet werden, wenn die entsprechenden Informationen korrekt ausgeschaltet sind.

Mit »ShowOVSC« (Listing 5) werden Color-Cycling-Bilder im Overscan-Format, also bildschirmfüllend angezeigt. Bilder, die kleiner als die aktuelle Workbenchgröße sind, werden zentriert dargestellt. Dies sieht besonders bei NTSC-Bildern auf PAL-Amigas besser aus. Umgekehrt werden auf NTSC-Rechnern PAL-Bilder wie Overscan-Bilder angezeigt. Dieser Modus wurde aber noch nicht getestet.

Die Beispiele sollen Ihnen die Anwendung der Prozeduren von »IFFSupport« zeigen, Sie aber auch zur Programmierung eigener Module anregen. Verhelfen Sie damit Ihren Spielen zur wohlverdienten Pixelpracht. so

Martin Jobst studiert Publizistik und Kommunikationswissenschaft in Salzburg und arbeitet seit drei Jahren als freier Mitarbeiter für Zeitschriften des Markt & Technik Verlages. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Martin Jobst, Hans-Pinsel-Straße 2, 8013 Haar bei München).

```
(*-----*)
:Programm.  IFFSupport.mod
:Author.    Fridtjof Siebert
:Adresse.   Nobilweg 67, D-7000-Stuttgart-40
:Telefon.
:Shortcut.  [fbs]
:Version.   1.5
:Copyright. Markt & Technik Verlag AG
:Sprache.   Modula-II
:Compiler.  M2Amiga v3.2
:Imports.   LoadBody.asm [fbs]
:History.   V1.1 [fbs] 27-Jul-88: Erste veröffentlichte Version
:History.   V1.2 [fbs] 16-Nov-88: Fehlerkorrektur
:History.   V1.3 [fbs] 28-Dec-88: kleine Änderungen, angeregt
:History.   V1.4 [fbs] 23-Mar-89: Fehlerkorrektur (ExtraHB)
:History.   V1.5 [fbs] 03-Jun-89: Anpassung fuer v3.2, Fehler-
:History.   korrektur (Add/RemInt-
:History.   Server()-Bug (3.2))
:Inhalt.    PROCEDURen für IFF-Bilder (Load, Save,
:Inhalt.    ColorCycling).
(*-----*)

(* $S- $F- $N- $R- $V- this makes it a bit faster and shorter. *)
```

```
IMPLEMENTATION MODULE IFFSupport;

FROM SYSTEM IMPORT ADR, ADDRESS, SHIFT, BITSET, LONGSET, CAST,
                INLINE, REG, SETREG;
FROM Arts IMPORT TermProcedure, Assert, BreakPoint;

FROM Exec IMPORT AllocMem, FreeMem, MemReqs, MemReqSet, UByte,
                Interrupt, AddIntServer, RemIntServer, NodeType;
FROM Dos IMPORT FileHandlePtr, Open, Close, Read, oldFile,
                Write, DeleteFile, Seek, newFile, beginning;
FROM Intuition IMPORT NewScreen, ScreenPtr, OpenScreen, CloseScreen,
                ScreenToBack, ScreenFlags, ScreenFlagSet,
                customScreen, MoveScreen, WindowPtr, OpenWindow,
                CloseWindow, IDCMPFlags, IDCMPFlagSet,
                WindowFlagSet, WindowFlags;
FROM Graphics IMPORT SetRGB4, RastPortPtr, BitMapPtr, ViewModes,
                ViewModeSet, BitMap, InitBitMap, AllocRaster,
                BitClear, FreeRaster, ViewPortPtr,
                RectanglePtr, Rectangle, GetRGB4;
FROM GfxMacros IMPORT OffDisplay, OnDisplay;
FROM Hardware IMPORT vertb;

FROM Strings IMPORT Compare, first, last;

FROM LoadBody IMPORT LoadBody;
```

Listing 1. Der Quellcode von »IFFSupport« in Modula ▶


```

CONST
  MOVEMS = 48E7H; (* 68000-Instruction MOVEM to save Regs*)
  MOVEML = 4CDFH; (* MOVEM to load Registers *)

TYPE
  CyclingInfo = RECORD (* Needed Data for Cycle-Interrupt *)
    int: Interrupt; (* The Cycling's Interrupt *)
    VP: ViewPortPtr; (* The Cycling's ViewPort *)
    count: ARRAY[0..15] OF CARDINAL; (* Cycling-Positions *)
    speedCnt: ARRAY[0..15] OF CARDINAL; (* counts Speed *)
  END;

VAR
  InH, OutH: FileHandlePtr; (* Files *)
  i,j,k: LONGINT; (* can be used by everything *)
  LineLength: LONGINT; (* Bytes per Image-Line *)
  LineWidth: LONGINT; (* Bytes per Screen-Line *)
  BM: BitMapPtr; (* Screen's BitMap *)
  Compression: BOOLEAN; (* compressed ? *)
  MaskPlane: BOOLEAN; (* Is there a Mask-Plane ?? *)
  Buffer: ADDRESS; (* Buffer for Reading/Writing *)
  TextBuffer: POINTER TO ARRAY[0..63] OF ARRAY[0..3] OF CHAR;
  LONGBuffer: POINTER TO ARRAY[0..63] OF LONGCARD;
  WORDBuffer: POINTER TO ARRAY[0..127] OF INTEGER;
  BYTEBuffer: POINTER TO ARRAY[0..255] OF UBYTE;
  len: LONGINT; (* Length from Read/Write() *)
  BitMaps: ARRAY[0..7] OF ADDRESS; (* Pointer to Planes *)
  Line,Plane: LONGINT; (* Count Lines and Planes *)
  Location,Right: POINTER TO UBYTE; (* Used loading Buffer *)
  RQPos,RQlen: LONGCARD; (* Used by QuickRead-Procedure *)
  RQBuffer: POINTER TO ARRAY[0..511] OF UBYTE; (* ReadQu-Buffer *)
  Exit: BOOLEAN; (* Exit LOOP ? *)
  CycleInfos: ARRAY[0..31] OF CyclingInfo; (* Colorcyclings *)
  IntInfo: IFFInfoTypePtr; (* Interrupt's IFFInfo *)
  IntNum: CARDINAL; (* Interrupt's ID *)
  IntCount,IntCount2,IntCount3: CARDINAL; (* used for Cycling *)
  ColorConv: LONGCARD; (* converting Colors *)
  Address: ADDRESS;
  FileLength,BodyPos,BodyLength: LONGINT; (* Pos+Length in File *)
  ShiftBuffer: ARRAY[0..31] OF LONGSET; (* Buffer f Graphic-Shift *)
  ShiftSource: POINTER TO ARRAY[0..31] OF LONGSET; (* Points into Planes *)
  NeedToShift: BOOLEAN; (* is shifting really needed? *)
  ShiftWidth,BitsToShift: CARDINAL; (* how far+how many BitsToShift *)
  TrueLeftOffset,TrueWidth: INTEGER; (* Word-aligned Offset & Width *)

(*----- Procedure called by machineopde to get Data: -----*)

PROCEDURE Read512();

BEGIN
  len := Read(InH,RQBuffer,512);
END Read512;

(*
TYPE
  ReadILBMFlags = (front,visible,dontopen>window);
  ReadILBMFlagSet = SET OF ReadILBMFlags;
*)

PROCEDURE ReadILBM(name: ARRAY OF CHAR; Flags: ReadILBMFlagSet;
  VAR Screen: ScreenPtr;
  VAR Window: WindowPtr): BOOLEAN;

PROCEDURE OpenScrn();

BEGIN
  WITH NuScreen DO
    width := IFFInfo.BMHD.scrnWidth;
    IF width < IFFInfo.BMHD.width THEN
      width := IFFInfo.BMHD.width;
    END;
    height := IFFInfo.BMHD.scrnHeight;
    IF height < IFFInfo.BMHD.height THEN
      height := IFFInfo.BMHD.height;
    END;
    leftEdge := IFFInfo.BMHD.left;
    topEdge := IFFInfo.BMHD.top;
    depth := IFFInfo.BMHD.depth;
  END;

```

```

viewModes := ViewModeSet[];
IF (width>400) AND (depth<5) THEN INCL(viewModes,hires) END;
IF height>300 THEN INCL(viewModes,lace) END;
WITH IFFInfo.CAMG DO
  IF (Lace IN viewType) THEN INCL(viewModes,lace) END;
  IF (HoMod IN viewType) THEN INCL(viewModes,ham) END;
  IF (Hires IN viewType) THEN INCL(viewModes,hires) END;
  IF (DblPF IN viewType) THEN INCL(viewModes,dualpf) END;
  IF (Extra IN viewType) THEN viewModes := ViewModeSet[extra-Halfbrite] END;
END;
detailPen := 0; blockPen := 0;
type := customScreen+ScreenFlagSet[screenQuiet];
font := NIL;
defaultTitle := NIL;
gadgets := NIL;
customBitMap := NIL;
IF NOT(front IN Flags) THEN topEdge := 600 END;
END;
IF dontopen IN Flags THEN
  INCL(NuScreen.type,customBitMap);
  WITH NuScreen DO
    customBitMap := AllocMem(SIZE(BitMap),MemReqSet[public]);
    InitBitMap(customBitMap,depth,width,height);
    i:=0;
    REPEAT
      customBitMap^.planes[i] := AllocRaster(width,height);
      BitMaps[i] := customBitMap^.planes[i];
      IF BitMaps[i]=NIL THEN
        IFFError := iffOutOfMem;
      ELSE
        BitClear(BitMaps[i],width DIV 8 * height,0);
      END;
      INC(i);
    UNTIL (i=depth) OR (IFFError#iffNoErr);
    IF IFFError#iffNoErr THEN (* error: give Mem back: *)
      WHILE i>1 DO
        DEC(i);
        FreeRaster(BitMaps[i],width,height);
      END;
    END;
  END;
ELSE
  Screen := OpenScreen(NuScreen);
  IF Screen=NIL THEN
    IFFError := iffOpenScreenfailed;
  ELSE
    IF NOT(front IN Flags) THEN
      ScreenToBack(Screen);
      MoveScreen(Screen,0,-600);
    END;
    BM := Screen.rastPort.bitMap;
    FOR i:=0 TO NuScreen.depth-1 DO
      BitMaps[i] := BM^.planes[i];
    END;
    WITH IFFInfo.CMAP DO
      FOR i:=0 TO colorCnt-1 DO
        SetRGB4(ADR(Screen.viewPort),i,red[i],green[i],blue[i]);
      END;
    END;
  END;
END;
END;
WITH NuWindow DO
  leftEdge := 0;
  topEdge := 0;
  width := IFFInfo.BMHD.width;
  height := IFFInfo.BMHD.height;
  detailPen := 1;
  blockPen := 0;
  idcmpFlags := IDCMPFlagSet[];
  flags := WindowFlagSet[borderless,noCareRefresh];
  firstGadget := NIL;
  checkMark := NIL;
  title := NIL;
  screen := Screen;
  bitMap := NIL;
  type := customScreen;
END;
IF (window IN Flags) AND (Screen#NIL) THEN
  Window := OpenWindow(NuWindow);

```



```

IF Window=NIL THEN
  CloseScreen(Screen);
  Screen := NIL;
  IFFError := iffOpenWindowfailed;
END;
END;
IF NOT(visible IN Flags) THEN OffDisplay() END;
END OpenScrn;

PROCEDURE ReadQuick(To: ADDRESS; Count: CARDINAL);

VAR
  ToPtr: POINTER TO ARRAY[0..9999] OF UByte;
  i: CARDINAL;

BEGIN
  ToPtr := To;
  i := 0;
  REPEAT
    IF RQPos=RQLen THEN
      RQLen := Read(InH,RQBuffer,512);
      RQPos := 0;
    END;
    ToPtr[i] := RQBuffer[RQPos];
    INC(RQPos); INC(i);
  UNTIL i=Count;
END ReadQuick;

BEGIN
  IFFInfo.IFFTitle := IFFTitleSet[];

  IF NOT(visible IN Flags) THEN OffDisplay() END;
  IFFError := iffNoErr;
  Screen := NIL; Window := NIL;
  RQPos := 0; RQLen := 0;

  InH := Open(ADR(name),oldFile);
  IF InH=NIL THEN
    IFFError := iffOpenfailed;
  ELSE

(*----- File Header: -----*)

    len := Read(InH,Buffer,12);
    IF len#12 THEN IFFError := iffReadWritefailed END;
    IF (Compare(TextBuffer[0],first,4,"FORM",TRUE)#0) OR
       (Compare(TextBuffer[2],first,4,"ILBM",TRUE)#0) THEN
      IFFError := iffWrongIFF;
    END;

    Exit := FALSE;

(*----- Main Loop: -----*)

    WHILE (IFFError=iffNoErr) AND NOT(Exit) DO
      len := Read(InH,Buffer,4);

(*----- BMHD: -----*)

      IF Compare(TextBuffer[0],first,4,"BMHD",TRUE)=0 THEN
        INCL(IFFInfo.IFFTitle,BMHD);
        len := Read(InH,Buffer,4);
        len := Read(InH,Buffer,LongBuffer[0]);
        WITH IFFInfo.BMHD DO
          width := WORDBuffer[0];
          height := WORDBuffer[1];
          left := WORDBuffer[2];
          top := WORDBuffer[3];
          depth := BYTEBuffer[8];
          masking := BYTEBuffer[9];
          MaskPlane := masking-1;
          Compression := BYTEBuffer[10]-1;
          transCol := WORDBuffer[6];
          xAspect := BYTEBuffer[14];
          yAspect := BYTEBuffer[15];
          scrnWidth := WORDBuffer[8];
          scrnHeight := WORDBuffer[9];
        END;

(*----- CMAP: -----*)

```

```

      ELSIF Compare(TextBuffer[0],first,4,"CMAP",TRUE)=0 THEN
        INCL(IFFInfo.IFFTitle,CMAP);
        len := Read(InH,Buffer,4);
        i := LongBuffer[0];
        len := Read(InH,Buffer,i);
        WITH IFFInfo.CMAP DO
          colorCnt := i DIV 3;
          j := 0;
          FOR k:=0 TO colorCnt-1 DO
            red [k] := BYTEBuffer[j ] DIV 16;
            green[k] := BYTEBuffer[j+1] DIV 16;
            blue [k] := BYTEBuffer[j+2] DIV 16;
            INC(j,3);
          END;
        END;

(*----- CAMG: -----*)

      ELSIF Compare(TextBuffer[0],first,4,"CAMG",TRUE)=0 THEN
        INCL(IFFInfo.IFFTitle,CAMG);
        len := Read(InH,Buffer,8);
        IFFInfo.CAMG.viewType := CAST(ViewTypeSet,LongBuffer[1]);

(*----- GRAB: -----*)

      ELSIF Compare(TextBuffer[0],first,4,"GRAB",TRUE)=0 THEN
        INCL(IFFInfo.IFFTitle,GRAB);
        len := Read(InH,Buffer,8);
        IFFInfo.GRAB.hotX := WORDBuffer[2];
        IFFInfo.GRAB.hotY := WORDBuffer[3];

(*----- DEST: -----*)

      ELSIF Compare(TextBuffer[0],first,4,"DEST",TRUE)=0 THEN
        INCL(IFFInfo.IFFTitle,DEST);
        len := Read(InH,Buffer,12);
        WITH IFFInfo.DEST DO
          depth := BYTEBuffer[4];
          planePick := WORDBuffer[3];
          planeOnOff := WORDBuffer[4];
          planeMask := WORDBuffer[5];
        END;

(*----- CRNG: -----*)

      ELSIF Compare(TextBuffer[0],first,4,"CRNG",TRUE)=0 THEN
        IF NOT(CRNG IN IFFInfo.IFFTitle) THEN
          IFFInfo.CRNG.count := 0;
        END;
        INCL(IFFInfo.IFFTitle,CRNG);
        len := Read(InH,Buffer,12);
        WITH IFFInfo.CRNG.data[IFFInfo.CRNG.count] DO
          rate := WORDBuffer[3];
          on := 0 IN CAST(BITSET,WORDBuffer[4]);
          forward := NOT(1 IN CAST(BITSET,WORDBuffer[4]));
          low := BYTEBuffer[10];
          high := BYTEBuffer[11];
        END;
        (* this line is only to identify illegal data, that some IFF-
        Files contain:*)
        on := on AND (low<IFFInfo.CMAP.colorCnt)
              AND (high<IFFInfo.CMAP.colorCnt);
        END;
        INC(IFFInfo.CRNG.count);

(*----- BODY: -----*)

      ELSIF Compare(TextBuffer[0],first,4,"BODY",TRUE)=0 THEN
        INCL(IFFInfo.IFFTitle,BODY);
        OpenScrn();
        IF IFFError=iffNoErr THEN
          len := Read(InH,Buffer,4);
          LineLength := CAST(INTEGER,CAST(BITSET,IFFInfo.BMHD.width+
          15)
                        * {4..15}) DIV 8;
          LineWidth := CAST(INTEGER,CAST(BITSET,NuScreen.width+15)
                        * {4..15}) DIV 8;

```

Listing 1. (Fortsetzung)


```

IF Compression THEN
(*----- let's load the BitMap's Data: -----*)
  LoadBody(Read512, RQBuffer, ADR(BitMaps[0]), LineLength,
    LineWidth, IFFInfo.BMHD.height, NuScreen.depth,
    MaskPlane); (* this does all the work very
    quickly *)
ELSE (* not compressed *)
(*----- to load uncompressed Images is less time-
critical: *)
  FOR Line := 0 TO IFFInfo.BMHD.height-1 DO
    FOR Plane := 0 TO NuScreen.depth-1 DO
      ReadQuick(BitMaps[Plane]+ LineWidth*Line, LineLength);
    END;
    IF MaskPlane THEN
      ReadQuick(Buffer, LineLength);
    END;
  END;
END;
END; (* IF NoErr *)
Exit := TRUE;

(*----- Ignore unknown data: -----*)

ELSE
  len := Read(InH, Buffer, 4);
  i := LONGBuffer[0];
  WHILE i > 256 DO
    len := Read(InH, Buffer, 256);
    DEC(i, 256);
  END;
  len := Read(InH, Buffer, i);
END;

(*----- Detect ReadError: -----*)

IF len=0 THEN
  IFFError := iffReadWriteFailed;
END;

END; (* WHILE NOT(Exit DO *)
END; (* IF NoErr *)

IF InH#NIL THEN Close(InH); InH := NIL; END;
IF IFFError#iffNoErr THEN
  IF Window#NIL THEN CloseWindow(Window) END;
  IF Screen#NIL THEN CloseScreen(Screen) END;
END;
OnDisplay();
RETURN IFFError=iffNoErr;
END ReadILBM; (* that's it *)

(*----- Procedures for ColorCycling: -----*)

PROCEDURE CycleInterrupt();

BEGIN
  INLINE(MOVEMS, 3F3EH);

  IntInfo := ADDRESS(REG(9));
  IF CRNG IN IntInfo.IFFTitle THEN
    IntNum := IntInfo.Internal.CycleID;
    WITH CycleInfos[IntNum] DO
      IntCount := 0;
      WHILE IntCount < IntInfo.CRNG.count DO
        WITH IntInfo.CRNG.data[IntCount] DO
          IF ON THEN
            INC(speedCnt[IntCount], rate);
            IF speedCnt[IntCount] >= 4000H THEN
              (* this 4000H should be 8000H, but that's too slow. *)
              (* don't know why, but this way, it works correctly *)
              DEC(speedCnt[IntCount], 4000H);
              IF forward THEN
                IF count[IntCount] <= low THEN
                  count[IntCount] := high;
                ELSE
                  DEC(count[IntCount]);
                END;
              ELSE
                IF count[IntCount] >= high THEN
                  count[IntCount] := low;
                END;
              END;
            END;
          END;
        END;
      END;
    END;
  END;
END;

```

```

ELSE
  INC(count[IntCount]);
END;
END;
IntCount3 := count[IntCount];
IntCount2 := low;
WHILE IntCount2 <= high DO
  SetRGB4(VP, IntCount2, IntInfo.CMAP.red[IntCount3],
    IntInfo.CMAP.green[IntCount3],
    IntInfo.CMAP.blue[IntCount3]);
  INC(IntCount3);
  IF IntCount3 > high THEN IntCount3 := low END;
  INC(IntCount2);
END;
END;
END;
INC(IntCount);
END;
END;
END;

INLINE(MOVEML, 7CFCH);
END CycleInterrupt;

PROCEDURE DoCycle(Info: IFFInfoTypePtr; Screen: ScreenPtr): BOOLEAN;
(* this creates an interrupt, that does cycling. You needn't worry, *)
(* whether there's cycling data or not. Don't forget to call EndCycle
to *)
(* remove the Cycling-Interrupt !!! *)
(* If result is false, any error occurred. Don't call EndCycle in
this case! *)

BEGIN
  i := 0;
  LOOP
    IF CycleInfos[i].VP=NIL THEN EXIT END;
    INC(i);
    IF i=32 THEN RETURN FALSE END;
  END;
  Info.Internal.CycleID := i;
  WITH CycleInfos[i] DO
    VP := ADR(Screen.viewPort);
    IF CRNG IN Info.IFFTitle THEN
      FOR j:=0 TO Info.CRNG.count-1 DO
        count[j] := Info.CRNG.data[j].low;
        speedCnt[j] := 0;
      END;
    END;
  END;
  WITH int DO
    node.type := interrupt;
    node.pri := -60;
    node.name := NIL;
    data := Info;
    code := ADR(CycleInterrupt);
  END;
  SETREG(0,0); AddIntServer(vertb, ADR(int));
END;
RETURN TRUE;
END DoCycle;

PROCEDURE EndCycle(Info: IFFInfoTypePtr;
(* remove cycling-Interrupt *)

)

BEGIN
  i := Info.Internal.CycleID;
  SETREG(0,0); RemIntServer(vertb, ADR(CycleInfos[i].int));
  CycleInfos[i].VP := NIL;
END EndCycle;

VAR
  DefaultRect: Rectangle;

PROCEDURE InitIFFInfo(Info: IFFInfoTypePtr;
  RP: RastPortPtr;
  VP: ViewPortPtr;
  VAR Rect: RectanglePtr);

```



```

BEGIN

WITH RP DO
  IF Rect=NIL THEN
    Rect := ADR(DefaultRect);
    WITH DefaultRect DO
      minX := 0; minY := 0;
      maxX := bitMap.bytesPerRow * 8 - 1;
      maxY := bitMap.rows - 1;
    END;
  END;

(*----- Initialize BMHD: -----*)

Info.BMHD.width := Rect.maxX - Rect.minX + 1;
Info.BMHD.height := Rect.maxY - Rect.minY + 1;
WITH Info.BMHD DO
  depth := bitMap.depth;
  left := 0;
  top := 0;
  masking := 0;
  transCol := 0;
  scrnWidth := bitMap.bytesPerRow * 8;
  scrnHeight := bitMap.rows;
  IF scrnWidth < 640 THEN
    xAspect := 10;
  ELSE
    xAspect := 5;
  END;
  IF scrnHeight > 400 THEN
    INC(xAspect, xAspect);
  END;
  yAspect := 11;
END;
END; (* WITH RP DO *)

(*----- Initialize CMAP: -----*)

WITH Info.CMAP DO
  colorCnt := VP.colorMap.count;
  FOR i := 0 TO colorCnt-1 DO
    ColorConv := GetRGB4(VP.colorMap, i);
    IF ColorConv > OFFFH THEN ColorConv := 0 END;
    red[i] := UByte(CAST(CARDINAL, CAST(BITSET,
      CARDINAL(SHIFT(ColorConv, -8))) * {0..3}));
    green[i] := UByte(CAST(CARDINAL, CAST(BITSET,
      CARDINAL(SHIFT(ColorConv, -4))) * {0..3}));
    blue[i] := UByte(CAST(CARDINAL, CAST(BITSET,
      CARDINAL(ColorConv)) * {0..3}));
  END;
END;

(*----- Initialize CAMG: -----*)

WITH Info.CAMG DO
  viewType := ViewTypeSet[];
  IF lace IN VP.modes THEN INCL(viewType, Lace) END;
  IF hires IN VP.modes THEN INCL(viewType, Hires) END;
  IF dualpf IN VP.modes THEN INCL(viewType, DblPF) END;
  IF ham IN VP.modes THEN INCL(viewType, HoMod) END;
  IF extraHalfbrite IN VP.modes THEN INCL(viewType, Extra) END;
END;

Info.IFFTitle := IFFTitleSet[BMHD, CMAP, CAMG];

END InitIFFInfo;

PROCEDURE WriteILBMall(Name: ARRAY OF CHAR;
  Info: IFFInfoTypePtr;
  BM: BitMapPtr;
  FirstLine, LeftOffset: INTEGER;
  CompressIt: BOOLEAN): BOOLEAN;

TYPE
  BufPtr = POINTER TO ARRAY[0..255] OF Ubyte;

PROCEDURE Compress(At: BufPtr; Length: LONGINT): LONGINT;
VAR
  at, last, out, len: LONGINT;

  PROCEDURE CopyUnchanged(from, to: LONGINT);

```

```

BEGIN
  BYTEBuffer[out] := to - from - 1;
  INC(out);
  WHILE from < to DO
    BYTEBuffer[out] := At[from];
    INC(out);
    INC(from);
  END;
  END CopyUnchanged;

BEGIN
  at := 1;
  last := 0;
  out := 0;
  LOOP
    IF (At[at]=At[at-1]) AND (At[at]=At[at+1]) AND (at+1 < Length-
  ) THEN
    IF last # at-1 THEN
      CopyUnchanged(last, at-1);
    END;
    last := at-1;
    (*----- Repeat Byte: -----*)
    REPEAT
      INC(at)
    UNTIL (At[last] # At[at]) OR (at-last=128) OR (at=Length);
    BYTEBuffer[out] := 257+last-at;
    INC(out);
    BYTEBuffer[out] := At[last];
    INC(out);
    last := at;
    IF at=Length THEN EXIT END;
    ELSIF (at-last)=128 THEN
    (*----- Copy Unchanged: -----*)
    CopyUnchanged(last, at);
    last := at;
  END;
  INC(at);
  IF at=Length THEN EXIT END;
END;
IF at # last THEN CopyUnchanged(last, at) END;
len := Write(OutH, Buffer, out);
INC(BodyLength, out);
INC(FileLength, out);
RETURN len;
END Compress;

PROCEDURE ShiftLine(At: ADDRESS);

VAR
  sourcelong, sourcebit, destlong, destbit: CARDINAL;

BEGIN
  ShiftSource := At;
  sourcelong := 0;
  sourcebit := 31-ShiftWidth;
  destlong := 0;
  destbit := 31;
  ShiftBuffer[0] := LONGSET[];
  FOR i:=1 TO BitsToShift DO
    IF sourcebit IN ShiftSource[sourcelong] THEN
      INCL(ShiftBuffer[destlong], destbit);
    END;
    IF sourcebit=0 THEN
      sourcebit := 31;
      INC(sourcelong);
    ELSE
      DEC(sourcebit);
    END;
  END;
  IF destbit=0 THEN
    destbit := 31;
    INC(destlong);
    ShiftBuffer[destlong] := LONGSET[];
  ELSE
    DEC(destbit);
  END;
END;
END ShiftLine;

(*----- MAIN: -----*)

```

Listing 1. (Fortsetzung)

BEGIN

(*----- Open: -----*)

```

OutH := Open(ADR(Name),newFile);
IF OutH=NIL THEN
  IFFError := iffOpenfailed;
  RETURN FALSE;
END;
TextBuffer[0] := "FORM";
TextBuffer[2] := "ILBM";
len := Write(OutH,TextBuffer,12);
IF len#12 THEN
  Close(OutH);
  OutH := NIL;
  IF DeleteFile(ADR(Name)) THEN END;
  IFFError := iffReadWritefailed;
  RETURN FALSE;
END;
FileLength := 4;

```

(*----- BMHD: -----*)

```

IF BMHD IN Info.IFFTitle THEN (* in fact, BMHD MUST be set *)
  WITH Info DO
    TextBuffer[0] := "BMHD";
    LONGBuffer[1] := 20; (* Length *)
    WORDBuffer[4] := BMHD.width;
    WORDBuffer[5] := BMHD.height;
    WORDBuffer[6] := BMHD.left;
    WORDBuffer[7] := BMHD.top;
    BYTEBuffer[16] := BMHD.depth;
    BYTEBuffer[17] := BMHD.masking; (* special masking *)
    IF CompressIt THEN (* compression *)
      BYTEBuffer[18] := 1;
    ELSE
      BYTEBuffer[18] := 0;
    END;
    BYTEBuffer[19] := 0; (* pad *)
    WORDBuffer[10] := BMHD.transCol; (* transparent Color *)
    BYTEBuffer[22] := BMHD.xAspect;
    BYTEBuffer[23] := BMHD.yAspect;
    WORDBuffer[12] := BMHD.scrnWidth;
    WORDBuffer[13] := BMHD.scrnHeight;
    len := Write(OutH,Buffer,28);
    INC(FileLength,28);
  END;
END;

```

(*----- CMAP: -----*)

```

IF CMAP IN Info.IFFTitle THEN (* this has to be set, too *)
  WITH Info DO
    TextBuffer[0] := "CMAP";
    LONGBuffer[1] := CMAP.colorCnt * 3;
    IF ODD(LONGBuffer[1]) THEN INC(LONGBuffer[1]) END;
    FOR i:=0 TO CMAP.colorCnt-1 DO
      BYTEBuffer[8+3*i] := UByte(ORD(CMAP.red [i]) * 16);
      BYTEBuffer[9+3*i] := UByte(ORD(CMAP.green[i]) * 16);
      BYTEBuffer[10+3*i] := UByte(ORD(CMAP.blue [i]) * 16);
    END;
    len := Write(OutH,Buffer,LONGBuffer[1]+8);
    INC(FileLength,LONGBuffer[1]+8);
  END;
END;

```

(*----- GRAB: -----*)

```

IF GRAB IN Info.IFFTitle THEN
  TextBuffer[0] := "GRAB";
  LONGBuffer[1] := 8;
  WORDBuffer[4] := Info.GRAB.hotX;
  WORDBuffer[5] := Info.GRAB.hotY;
  len := Write(OutH,Buffer,12);
  INC(FileLength,12);
END;

```

(*----- DEST: -----*)

```

IF DEST IN Info.IFFTitle THEN

```

```

  TextBuffer[0] := "DEST";
  LONGBuffer[1] := 8;
  BYTEBuffer[8] := Info.DEST.depth;
  BYTEBuffer[9] := 0;
  WORDBuffer[5] := Info.DEST.planePick;
  WORDBuffer[6] := Info.DEST.planeOnOff;
  WORDBuffer[7] := Info.DEST.planeMask;
  len := Write(OutH,Buffer,16);
  INC(FileLength,16);
END;

```

(*----- CAMG: -----*)

```

IF CAMG IN Info.IFFTitle THEN
  TextBuffer[0] := "CAMG";
  LONGBuffer[1] := 4;
  LONGBuffer[2] := CAST(LONGCARD,Info.CAMG.viewType);
  len := Write(OutH,Buffer,12);
  INC(FileLength,12);
END;

```

(*----- CRNG: -----*)

```

IF CRNG IN Info.IFFTitle THEN
  i := 0;
  WHILE i<LONGINT(Info.CRNG.count) DO
    WITH Info.CRNG.data[i] DO
      TextBuffer[0] := "CRNG";
      LONGBuffer[1] := 8;
      WORDBuffer[4] := 0;
      WORDBuffer[5] := rate;
      IF on THEN
        WORDBuffer[6] := 1;
      ELSE
        WORDBuffer[6] := 0;
      END;
      IF NOT(forward) THEN
        INC(WORDBuffer[6],2);
      END;
      BYTEBuffer[14] := low;
      BYTEBuffer[15] := high;
      len := Write(OutH,Buffer,16);
      INC(FileLength,16);
    END;
    INC(i);
  END;
END;

```

(*----- BODY: -----*)

```

  BodyPos := FileLength;
  TextBuffer[0] := "BODY";
  len := Write(OutH,Buffer,8);
  INC(FileLength,8);
  BodyLength := 0;
  i := 0;
  TrueLeftOffset := CAST(CARDINAL,CAST(BITSET,LeftOffset) * {4..15});
  TrueWidth := CAST(CARDINAL,CAST(BITSET,Info.BMHD.width + 15) * {4..15});

  WHILE i<LONGINT(Info.BMHD.depth) DO
    BitMaps[i] := BM.planes[i];
    INC(BitMaps[i],FirstLine * LONGINT(BM.bytesPerRow) +
      TrueLeftOffset DIV 8);
    INC(i);
  END;

  LineLength := TrueWidth DIV 8;

  NeedToShift := (TrueLeftOffset # LeftOffset)
    OR (TrueWidth # Info.BMHD.width);
  IF NeedToShift THEN
    ShiftWidth := LeftOffset - TrueLeftOffset;
    BitsToShift := Info.BMHD.width;
  END;

  IF CompressIt THEN
    Line := 0;
    WHILE Line<Info.BMHD.height DO
      Plane := 0;

```



```

WHILE Plane<LONGINT(Info.BMHD.depth) DO
  IF NeedToShift THEN
    ShiftLine(BitMaps[Plane]);
    len := Compress(ADR(ShiftBuffer),LineLength);
  ELSE
    len := Compress(BitMaps[Plane],LineLength);
  END;
  INC(BitMaps[Plane],BM.bytesPerRow);
  INC(Plane);
  INC(Line);
END;
ELSE
  Line := 0;
  WHILE Line<Info.BMHD.height DO
    Plane := 0;
    WHILE Plane<LONGINT(Info.BMHD.depth) DO
      IF NeedToShift THEN
        ShiftLine(BitMaps[Plane]);
        len := Write(OutH,ADR(ShiftBuffer),LineLength);
      ELSE
        len := Write(OutH,BitMaps[Plane],LineLength);
      END;
      INC(FileLength,LineLength);
      INC(BodyLength,LineLength);
      INC(BitMaps[Plane],BM.bytesPerRow);
      INC(Plane);
    END;
    INC(Line);
  END;
END;
IF ODD(FileLength) THEN
  BYTEBuffer[0] := 0;
  len := Write(OutH,Buffer,1);
  INC(FileLength);
END;

len := Seek(OutH,BodyPos+12,beginning);
LONGBuffer[0] := BodyLength;
len := Write(OutH,Buffer,4);

(*----- Done: -----*)

len := Seek(OutH,4,beginning);
LONGBuffer[0] := FileLength;
len := Write(OutH,Buffer,4);
Close(OutH);
OutH := NIL;
IF len#4 THEN
  IF DeleteFile(ADR(Name)) THEN END;
  IFFError := iffReadWriteFailed;
  RETURN FALSE;
ELSE
  RETURN TRUE;
END;
END WriteILBM;

PROCEDURE WriteILBM(Name: ARRAY OF CHAR;
  RP: RastPortPtr;
  VP: ViewPortPtr;
  Rect: RectanglePtr;
  CompressIt: BOOLEAN): BOOLEAN;

BEGIN
  InitIFFInfo(ADR(Info),RP,VP,Rect);
  RETURN WriteILBM(Info,ADR(Info),RP.bitMap,
    Rect.minY,Rect.minX,CompressIt);
END WriteILBM;

PROCEDURE WriteILBMScreen(Name: ARRAY OF CHAR;
  Screen: ScreenPtr;
  Rect: RectanglePtr;
  CompressIt: BOOLEAN): BOOLEAN;

BEGIN

```

```

WITH Screen DO
  RETURN WriteILBM(Name,ADR(rastPort),ADR(viewPort),Rect,CompressIt);
END;

END WriteILBMScreen;

(*----- TermProcedure: -----*)

PROCEDURE Cleanup();

BEGIN
  IF InH #NIL THEN Close(InH) END;
  IF OutH #NIL THEN Close(OutH) END;
  FreeMem(Buffer,768);
END Cleanup;

(*----- Initialization: -----*)

BEGIN
  Buffer := AllocMem(768,MemReqSet[memClear]);
  Assert(Buffer#NIL,ADR("Not enough ChipMem !!!"));
  TextBuffer := Buffer;
  LONGBuffer := Buffer;
  WORDBuffer := Buffer;
  BYTEBuffer := Buffer;
  RQBuffer := ADDRESS(Buffer+256);
  InH := NIL; OutH := NIL;
  FOR i:=0 TO 31 DO CycleInfos[i].VP:=NIL END;
  TermProcedure(Cleanup);
END IFFSupport.

```

Listing 1. (Schluß)

```

(*-----
:Programm.   IFFSupport.mod
:Autor.      Fridtjof Siebert
:Adresse.    Nobileweg 67, D-7000-Stuttgart-40
:Telefon.
:Shortcut.   [fbs]
:Version.    1.5
:Copyright.  Markt & Technik Verlag AG
:Sprache.    Modula-II
:Compiler.   M2Amiga v3.2
:Imports.    LoadBody.asm [fbs]
:History.    V1.1 [fbs] 27-Jul-88: Erste veroeffentlichte Version
:History.    V1.2 [fbs] 16-Nov-88: Fehlerkorrektur
              (NIL-RectanglePtr)
:History.    V1.3 [fbs] 28-Dec-88: kleine Aenderungen, angeregt
              von S. Salewski
:History.    V1.4 [fbs] 23-Mar-89: Fehlerkorrektur (ExtraHB)
:History.    V1.5 [fbs] 03-Jun-89: Anpassung fuer v3.2, Fehler-
              korrektur (Add/RemInt-
              Server()-Bug (3.2))
:Inhalt.     PROCEDURen fuer IFF-Bilder (Load, Save,
              ColorCycling).
-----*)

DEFINITION MODULE IFFSupport;

FROM Intuition IMPORT ScreenPtr, NewScreen, WindowPtr, NewWindow;
FROM Graphics  IMPORT BitMapPtr, RastPortPtr, ViewPortPtr,
  RectanglePtr;
FROM Exec      IMPORT UByte;

(*----- Types: -----*)

TYPE
  IFFTtitles = (BMHD,CMAP,GRAB,DEST,CAMG,CRNG,BODY,SPRT,CCRT,
    CMHD,DPPV);
  IFFTtitleSet = SET OF IFFTtitles;
  (* SPRT,CCRT,CMHD,DPPV not implemented !!! *)

```

**Listing 2. Das zugehörige Definitionsmodul
»IFFSupport.def«**


```

ViewTypes = (vt0,Ersy,Lace,LPen,vt4,vt5,vt6,Extra,Gaud,Color,
             DblPF,HoMod,vt12,vt13,vt14,Hires,v16);
ViewTypeSet = SET OF ViewTypes;
(* which ViewModes are selected *)

TYPE

(*----- The Structure that keeps all the data: -----*)

IFFInfoTypePtr = POINTER TO IFFInfoType;
IFFInfoType = RECORD
(* This contains all Data needed for a Picture *)

(*----- Which Data is available: -----*)
IFFTitle: IFFTitleSet; (* all Sub-Records, whose equally *)
(* named Flag is set here, contain readable data *)

(*----- Information on BitMap: -----*)
BMHD: RECORD

    width,height: INTEGER; (* the Picture's Size *)
    depth: UByte; (* it's Depth (how many BitPlanes) *)
    left,top: INTEGER; (* it's Location *)
    masking: UByte; (* Masking (see Documentation) *)
    transCol: INTEGER; (* Transparent Color *)
    xAspect,yAspect: UByte; (* Verzerrung *)
    scrnWidth,scrnHeight: INTEGER; (* The Image's Screen's Size *)
END;

(*----- Information on Colors: -----*)
CMAP: RECORD

    colorCnt: CARDINAL; (* Number of Colors used *)
    red,green,blue: ARRAY[0..63] OF UByte;
(* the Colors (I hope for 6 Bitplanes to be possible anytime) *)
END;

(*----- Information on HotSpot: -----*)
GRAB: RECORD

    hotX,hotY: INTEGER; (* Hot-Spot of this Image (if exists) *)
END;

(*----- Information on Destination-Bitmap: -----*)
DEST: RECORD

    depth: UByte; (* number of Planes *)
    planePick: CARDINAL;
    planeOnOff: CARDINAL; (* set or clear other Planes ? *)
    planeMask: CARDINAL; (* planes to be changed *)
END;

(*----- Information on any Special ViewMode: -----*)
CAMG: RECORD

    viewType: ViewTypeSet; (* ViewMode *)
END;

(*----- Information on ColorCycling: -----*)
CRNG: RECORD

    count: CARDINAL; (* Number of ColorCyclings *)
    data: ARRAY[0..15] OF RECORD

        rate: INTEGER; (* velocity, 800H is 60 per second *)
        on: BOOLEAN; (* decide, wether CRNG is active or not *)
        forward: BOOLEAN; (* Direction (DPaint) *)
        low,high: UByte; (* lower and upper Color of this Range *)
    END;
END;

(*----- Internal Information: -----*)
Internal: RECORD

    CycleID: CARDINAL; (* to distinguish different cyclings *)
END;
END;

VAR
    IFFInfo: IFFInfoType;

(*----- The NewScreen-Structure. -----*)
(* this can be used to open the Screen, if dontopen is specified *)

VAR
    NuScreen: NewScreen;

```

```

(*----- The NewWindow-Structure. -----*)
(* this can be used to open the Window later. Don't forget *)
(* to put ScreenPtr in NuWindow.screen !!! *)

VAR
    NuWindow: NewWindow;

(*----- Error-Message: -----*)
(* IFFError contains Error-Number if Read- or WriteILBM failed. *)

TYPE
    IFFErrors = (iffNoErr,iffOutOfMem,iffOpenScreenfailed,
                iffOpenWindowfailed,iffOpenfailed,iffWrongIFF,
                iffReadWritefailed);

VAR
    IFFError: IFFErrors;

(* ----- R e a d I L B M : ----- *)

TYPE
    ReadILBMFlags = (front,visible,dontopen>window);
    ReadILBMFlagSet = SET OF ReadILBMFlags;

PROCEDURE ReadILBM(name: ARRAY OF CHAR; Flags: ReadILBMFlagSet;
                  VAR Screen: ScreenPtr;
                  VAR Window: WindowPtr): BOOLEAN;

(* Name: The IFF-Filename *)
(* Flags: *)
(* -front: decides if Screen is first or last while loading *)
(* -visible: decides if display should be turned off (faster) *)
(* -dontopen: avoids to open the Screen. Returned value is NIL *)
(* -window: if set, an Window of the same size as the Image *)
(* is opened. So, Gadgets etc. can be added to it. *)
(* Screen: Pointer to Screen-structure of opened Screen *)
(* Window: Pointer to the opened Window or NIL if no window set. *)
(* Result: FALSE if error occurred. Then there's no Screen. *)

(* ----- Start Colorcycling: ----- *)

PROCEDURE DoCycle(Info: IFFInfoTypePtr; Screen: ScreenPtr): BOOLEAN;
(* this should create an interrupt, that does cycling. You *)
(* needn't worry, whether there's cycling data or not. Don't *)
(* forget to call EndCycle to remove the Cycling-Interrupt ! *)
(* if result is false, an error occurred. Don't call EndCycle *)
(* in this case! *)

(* ----- End Colorcycling: ----- *)

PROCEDURE EndCycle(Info: IFFInfoTypePtr);
(* remove cycling-Interrupt *)

(* --- Initialize BMHD, CMAP & CAMG for WriteILBMall: --- *)

PROCEDURE InitIFFInfo(Info: IFFInfoTypePtr;
                    RP: RastPortPtr;
                    VP: ViewPortPtr;
                    VAR Rect: RectanglePtr);

(* Initialize essential parts of IFFInfoType-Variable. *)
(* used to simplify the initialization of an IFFInfoType *)
(* RP: RastPort containing the BitMap etc. *)
(* VP: ViewPort containing the Colors, ViewModes etc. *)
(* Rect: The Rectangle Region in your RastPort, that *)
(* should be saved or NIL to save complete RastPort *)

(* ----- Save an ILBM-File: ----- *)

PROCEDURE WriteILBMall(Name: ARRAY OF CHAR;
                    Info: IFFInfoTypePtr;
                    BM: BitMapPtr;
                    FirstLine, LeftOffset: INTEGER;
                    CompressIt: BOOLEAN): BOOLEAN;

(* Saves IFF-File named Name *)
(* Low-Level Procedure. You should use it to save Pictures *)
(* with ColorCycling and things like that. *)
(* To save Screens, Windows or so use the other Procedures ! *)
(* Info.IFFTitle must have set the Flags of all initialized *)

```



```
(* Sub-Records *)
(* BM: contains the Graphicdata. In fact BM doesn't *)
(* have to be part of a RastPort. It can be used *)
(* to save a MaskPlane. Then BM has to contain *)
(* one extra Plane and BM.depth and *)
(* Info.BMHD.depth have to be increased by 1. *)
(* FirstLine: is the TopEdge within BM *)
(* LeftOffset: LeftEdge within BM. Must be a Word-Address, *)
(* i.e. 16*n *)
(* an example to call this is the Implementation of WriteILBM() *)
(* ----- Save a RastPort and ViewPort ILBM-File: ----- *)
```

```
PROCEDURE WriteILBM(Name: ARRAY OF CHAR;
  RP: RastPortPtr;
  VP: ViewPortPtr;
  Rect: RectanglePtr;
  CompressIt: BOOLEAN): BOOLEAN;
```

```
(* Creates an ILBM-File *)
(* Name: File's Name *)
(* RP: RastPort containing the BitMap etc. *)
(* VP: ViewPort containing the Colors, ViewModes etc. *)
(* Rect: The Rectangle Region in your RastPort, that *)
(* should be saved or NIL to save hole RastPort *)
(* CompressIt: Create compressed ILBM-File or not ? *)
(* Result is FALSE if any Error occurred. *)
(* example to save a Window: *)
(* OK := WriteILBM("Test.iff", *)
(* MyWindow.rPort, *)
(* ADR(MyWindow.screen.viewPort, *)
(* TRUE); *)
```

```
(* ----- Save a Screen as ILBM-File: ----- *)
```

```
PROCEDURE WriteILBMScreen(Name: ARRAY OF CHAR;
  Screen: ScreenPtr;
  Rect: RectanglePtr;
  CompressIt: BOOLEAN): BOOLEAN;
```

```
(* This creates an ILBM-File from a Screen *)
(* Name: File's Name *)
(* Screen: Screen to be saved *)
(* Rect: The Rectangle Region in your Screen, that *)
(* should be saved or NIL to save hole Screen *)
(* CompressIt: Create a Compressed ILBM-File *)
(* Returns TRUE if no Error occurred. *)
(* example: OK:=WriteILBMScreen("Test.iff",MyScreen,NIL,TRUE); *)
```

END IFFSupport.

Listing 2. (Schluß)

```
(*-----*)
:Programm. ShowIFF.mod
:Autor. Fridtjof Siebert
:Adresse. Nobileweg 67, D-7-Stgt-40
:Telefon.
:Shortcut. [fbs]
:Version. 1.0
:Datum. 20.04.88
:Copyright. Markt & Technik Verlag AG
:Sprache. Modula-II
:Compiler. M2Amiga
:Imports. IFFSupport [fbs].
:UpDate. none.
:Inhalt. IFF-Ladeprogramm fuer IFF (ILBM)-Bilder.
:Bemerkung. Syntax: ShowIFF <filename>
(*-----*)

MODULE ShowIFF;

FROM SYSTEM IMPORT ADR, ADDRESS, SHIFT, BITSET, LONGSET, CAST;

FROM Arguments IMPORT NumArgs, GetArg;

FROM Intuition IMPORT ScreenPtr, CloseScreen, DisplayBeep, WindowPtr;
```

```
FROM IFFSupport IMPORT ReadILBM, ReadILBMFlags, ReadILBMFlagSet;
```

```
FROM Dos IMPORT Delay;
```

```
VAR
  MyScreen: ScreenPtr; (* the Picture's ScreenPointer *)
  WindowDummy: WindowPtr; (* only a dummy for NIL-Windowpointer *)
  Name: ARRAY[0..79] OF CHAR; (* the Picture's Name *)
  length: INTEGER; (* dummy for receiving Name's Length *)
  Ciapra [OBFE001H]: SET OF (s0,s1,s2,s3,s4,s5,lmb);
  (* Ciapra is a Hardware-Register. Bit #6 contains left Button *)
```

BEGIN

```
(*----- Get Name: -----*)
```

```
IF NumArgs() # 0 THEN
  GetArg(1, Name, length);
ELSE
  HALT; (* a silly guy didn't say which pic I should load *)
END;
```

```
(*----- Read and Show Pic: -----*)
```

```
IF ReadILBM(Name, ReadILBMFlagSet{front, visible}, MyScreen, WindowDummy) THEN
```

```
  WHILE lmb IN Ciapra DO Delay(3) END;
  (* that's bad style, but it's the easiest way to wait for left Button *)
```

```
  CloseScreen(MyScreen); (* close the Screen *)
```

```
ELSE
  DisplayBeep(NIL); (* any error occurred: lets display a Beep ! *)
END;
```

END ShowIFF. That's all. Wasn't it easy ???

Listing 3. »ShowIFF«, die Laderoutine für IFF-Bilder ■

```
(*-----*)
:Programm. ShowCycle.mod
:Autor. Fridtjof Siebert
:Adresse. Nobileweg 67, D-7-Stgt-40
:Telefon.
:Shortcut. [fbs]
:Version. 1.0
:Datum. 20.04.88
:Copyright. Markt & Technik Verlag AG
:Sprache. Modula-II
:Compiler. M2Amiga
:Imports. IFFSupport [fbs].
:UpDate. none.
:Inhalt. Demo zu IFFSupport für Colorcycling.
:Bemerkung. Syntax: ShowCycle <filename>
(*-----*)
```

MODULE ShowCycle;

```
FROM SYSTEM IMPORT ADR, ADDRESS, SHIFT, BITSET, LONGSET, CAST;
```

```
FROM Arguments IMPORT NumArgs, GetArg;
```

```
FROM Intuition IMPORT ScreenPtr, CloseScreen, DisplayBeep, WindowPtr;
```

```
FROM IFFSupport IMPORT ReadILBM, ReadILBMFlags, ReadILBMFlagSet,
  DoCycle, EndCycle, IFFInfo;
```

```
FROM Dos IMPORT Delay;
```

```
VAR
  MyScreen: ScreenPtr; (* the Picture's ScreenPointer *)
  WindowDummy: WindowPtr; (* only a dummy for NIL-Windowpointer *)
```

Listing 4. Bilder mit Farbanimationen werden mit »ShowCycle« geladen ►


```

Name: ARRAY[0..79] OF CHAR; (* the Picture's Name *)
length: INTEGER; (* dummy for receiving Name's Length *)
Ciapra [OBFEO01H]: SET OF (s0,s1,s2,s3,s4,s5,lmb);
(* Ciapra is a Hardware-Register. Bit #6 contains left Button *)

BEGIN

(*----- Get Name: -----*)

IF NumArgs() # 0 THEN
  GetArg(1,Name,length);
ELSE
  HALT; (* a silly guy didn't say which pic I should load *)
END;

(*----- Read and Show Pic: -----*)

IF ReadILBM(Name,ReadILBMFlagSet[front,visible],MyScreen,WindowDummy) THEN

  (*----- start ColorCycling: -----*)

  IF DoCycle(ADR(IFFInfo),MyScreen) THEN END;

  WHILE lmb IN Ciapra DO Delay(3) END;
  (* bad style, but the easiest way to wait for left Button *)

  (*----- stop ColorCycling (this mustn't be forgotten !!!) -----*)

  EndCycle(ADR(IFFInfo));

  CloseScreen(MyScreen); (* close the Screen *)

ELSE
  DisplayBeep(NIL); (* any error occurred: lets display a Beep ! *)
END;

END ShowCycle. That's all. Wasn't it easy ???

```

Listing 4. (Schluß)

```

(*-----
:Programm. ShowOvsc.mod
:Author. Fridtjof Siebert
:Adresse. Nobileweg 67, D-7000 Stuttgart 40
:Telefon.
:Shortcut. [fbs]
:Version. 1.0
:Datum. 26-Feb-89
:Copyright. Markt & Technik Verlag AG
:Sprache. Modula-II
:Compiler. M2Amiga v3.1d
:Imports. IFFSupport [fbs].
:Update. none.
:Inhalt. Demo zu IFFSupport für Overscan-Colorcycling-Bilder.
:Bemerkung. Syntax: ShowOvsc <filename>
*)

```

MODULE ShowOvsc;

```

FROM SYSTEM IMPORT ADR;
FROM Arguments IMPORT NumArgs, GetArg;
FROM Graphics IMPORT GfxBasePtr, ViewModes;
FROM Intuition IMPORT ScreenPtr, CloseScreen, DisplayBeep,
  WindowPtr, MakeScreen, RethinkDisplay;
FROM IFFSupport IMPORT ReadILBM, ReadILBMFlags, ReadILBMFlagSet,
  DoCycle, EndCycle, IFFInfo, NuScreen;
FROM Dos IMPORT Delay;
IMPORT Graphics;

CONST
  minvx = -31;
  maxvx = 0;
  minvy = -19;
  maxvy = 12;
  author = "ShowOvsc, 1989 by Fridtjof Siebert / AMOK Stuttgart";

```

```

VAR
  MyScreen: ScreenPtr; (* the Picture's ScreenPointer *)
  WindowDummy: WindowPtr; (* dummy for NIL-Windowpointer *)
  Name: ARRAY[0..79] OF CHAR; (* the Picture's Name *)
  length: INTEGER; (* dummy for Name's Length *)
  Ciapra [OBFEO01H]: SET OF (s0,s1,s2,s3,s4,s5,lmb);
  gfx: GfxBasePtr;

BEGIN

(*----- Get Name: -----*)

IF NumArgs() # 0 THEN GetArg(1,Name,length) ELSE HALT END;
gfx := ADR(Graphics);

(*----- Read and Show Pic: -----*)

IF ReadILBM(Name,ReadILBMFlagSet[front],MyScreen,WindowDummy) THEN

  WITH MyScreen.viewPort DO
    dxOffset := IFFInfo.BMHD.width;
    dyOffset := IFFInfo.BMHD.height;
    IF NOT(hires IN NuScreen.viewModes) THEN INC(dxOffset,dxOffset)
    END;
    IF lace IN NuScreen.viewModes THEN dyOffset := dyOffset
    DIV 2 END;
    dxOffset := (ORD(gfx.normalDisplayColumns) - dxOffset) DIV 2;
    dyOffset := (ORD(gfx.normalDisplayRows) - dyOffset) DIV 2;
    IF dyOffset < -16 THEN dyOffset := -16 ELSEIF dyOffset > 200
    THEN dyOffset := 200 END;
    IF dxOffset < -32 THEN dxOffset := -32 ELSEIF dxOffset > 704
    THEN dxOffset := 704 END;
    IF NOT(hires IN NuScreen.viewModes) THEN dxOffset := dxOffset
    DIV 2 END;
    IF lace IN NuScreen.viewModes THEN INC(dyOffset,dyOffset)
    END;
  END;
  MakeScreen(MyScreen); RethinkDisplay();

  (*----- start ColorCycling: -----*)

  IF DoCycle(ADR(IFFInfo),MyScreen) THEN END;

  WHILE lmb IN Ciapra DO Delay(3) END;
  (* that's bad style, but it's the easiest way to wait for left
  Button *)

  (*----- stop ColorCycling (this mustn't be forgotten !!!) -----*)

  EndCycle(ADR(IFFInfo));

  CloseScreen(MyScreen); (* close the Screen *)

ELSE
  DisplayBeep(NIL); (* any error occurred: lets display a Beep ! *)
END;

END ShowOvsc. That's all. Wasn't it easy ???

```

Listing 5. »ShowOVSC« nutzt die gesamte Bildschirmfläche zur Darstellung

```

DEFINITION MODULE LoadBody;

FROM SYSTEM IMPORT BYTE,WORD,ADDRESS,BITSET,LONGSET,FFP;

(*----- LoadBody -----*)

PROCEDURE LoadBody(a: PROC; b,c:ADDRESS; d,e:LONGINT; f,g:INTEGER;h:
  BOOLEAN);

(* PROCEDURE LoadBody(GetData: PROC; Buffer, BitMapPtrs: ADDRESS;
  LineLength, LineWidth, Height, Depth: INTEGER;
  ExtraPlane: BOOLEAN);

*)

END LoadBody.

```

Listing 6. »LoadBody.def«: die Definition der Maschinenroutine


```
IMPLEMENTATION MODULE LoadBody;

FROM SYSTEM IMPORT BYTE, WORD, ADDRESS, BITSET, LONGSET, FFP,
    INLINE, SETREG, REG, CAST;

CONST
    MOVEMS = 48E7H;
    MOVEML = 4CDFH;

(*----- LoadBody -----*)

PROCEDURE LoadBody(a: PROC; b,c:ADDRESS; d,e:LONGINT; f,g:INTEGER;h:
    BOOLEAN);

(* PROCEDURE LoadBody(GetData: PROC; Buffer, BitMapPtrs: ADDRESS;
    LineLength, LineWidth, Height, Depth: INTEGER;
    ExtraPlane: BOOLEAN);
*)

BEGIN

    INLINE(MOVEMS,OFFFEH);

    SETREG(10,CAST(ADDRESS,a));
    SETREG(11,ADDRESS(b));
    SETREG(12,ADDRESS(c));
    SETREG( 2,ADDRESS(d));
    SETREG( 3,ADDRESS(e));
    SETREG( 5,ADDRESS(f));
    SETREG( 4,ADDRESS(g));
    SETREG( 6,ADDRESS(h));
```

```
    INLINE(041FAH,000D2H,03086H,041FAH,000CEH,03085H,048E7H,03F3EH);
    INLINE(04E92H,04CDFH,07CFCH,0323CH,00000H,03A3CH,00000H,03C3CH);
    INLINE(00000H,03003H,0C0C5H,02A40H,0E546H,0DBF4H,06000H,0E446H);
    INLINE(02C4DH,0DDC2H,06100H,00084H,0B03CH,00080H,06212H,06722H);
    INLINE(01E00H,06100H,00076H,01A80H,0528DH,05307H,06AF4H,06012H);
    INLINE(01E00H,06100H,00066H,01A80H,0528DH,05207H,0BE3CH,00001H);
    INLINE(066F4H,0BDCDH,062CEH,05246H,0BC44H,065B6H,041FAH,00066H);
    INLINE(04A50H,06738H,02A7CH,00000H,00000H,06100H,0003EH,0B03CH);
    INLINE(00080H,06210H,06722H,01E00H,06100H,00030H,0528DH,05307H);
    INLINE(06AF6H,06014H,01E00H,06100H,00022H,0DBFCH,00000H,00101H);
    INLINE(0CEBCH,00000H,000FFH,09BC7H,0B48DH,062CEH,05245H,041FAH);
    INLINE(00026H,0BA50H,06500H,0FF68H,0601EH,01033H,01000H,05241H);
    INLINE(0B27CH,00200H,0650CH,048E7H,0FF3EH,04E92H,04CDFH,07CFFH);
    INLINE(04241H,04E75H,00001H,00001H);
```

```
    INLINE(MOVEML,07FFFFH);
```

```
END LoadBody;
```

```
BEGIN
END LoadBody.
```

Listing 7. »LoadBody.mod«: die Implementation der Maschinenroutine

Kompaktor für IFF-Bilder

Packen wir's an

Wenn Sie viel mit Bildern arbeiten, sind die langen Ladezeiten oft störend. Wenn dann auch noch der Platz auf der Diskette knapp wird, gibt es nur eines: Die Daten müssen gepackt werden.

In Sonderheft 6 veröffentlichten wir den Müllmann »Waste-Hunter«. Damit lassen sich Programmfiles bis auf die Hälfte ihrer ursprünglichen Länge reduzieren. Der Autor von »Waste-Hunter«, Lutz Vieweg, hat sich nicht ausgeruht und präsentiert in diesem Heft einen IFF-Kompaktor. Wie der Name schon verrät, verarbeitet diese IFF-Daten, aber auch andere Datenfiles. Lediglich für Musik-Daten, genauer Sound-Sample-Daten, ist er nicht geeignet. Zukünftige Versionen sollen aber auch mit diesen Daten fertig werden.

Der Kompaktor besteht aus zwei Teilen: dem Packer »WaHuD« und der Routine »decompd«, welche die Daten wieder dekomprimiert.

Die Anwendung von »WaHuD« ist dabei wie folgend: Sie komprimieren die Daten mit Hil-

Der Platz auf Disketten ist immer zu wenig. Bilder und andere Daten verschlingen besonders viel Raum. Mit dem IFF-Kompaktor gehen diese Zeiten zu Ende.

Von Martin Jobst

fe von »WaHuD«. Rufen Sie den Packer dazu vom CLI aus mit

```
WaHuD [-n] filename
```

auf. Mit »-n« geben Sie die Größe der Suchtiefe (Werte von 0 bis 6), also die Stärke der Komprimierung, an. In den meisten Fällen wird Suchtiefe 1 ausreichend sein. Das Fenster des Packers öffnet sich. Unter »READ« sehen Sie die Anzahl der Bytes des Files, die bereits gelesen wurde. »WRITE« gibt die Byteanzahl an, die nach dem Packen übrig geblieben ist. Ganz rechts sehen Sie, wieviel Prozent das File noch von seiner ursprünglichen Länge besitzt. Solange der Packer in Aktion ist, blinkt der Mauszeiger.

Wenn der Packer seine Arbeit beendet hat, können Sie

mit »Y/N« entscheiden, ob das File gespeichert werden soll. Ist das der Fall, wird es unter dem gleichen Namen mit angehängtem »d« gespeichert.

Wie werden komprimierte Daten entpackt? Dazu benötigen Sie »decompd«, welches im Assembler-Sourcecode vorliegt. Dieses assemblieren Sie und laden es mit dem Programm, das die gepackten Daten lesen soll. Das können Assembler- aber auch C- oder Basic-Programme sein. Der Linker muß nur das B-Link-Verfahren unterstützen. Das betrifft etwa den Lattice-C-Compiler, den Hisoft-Basic-Compiler oder die meisten Assembler (mit Ausnahme des Profimat). Bevor Sie »decompd« assemblieren, müssen Sie im Sourcecode die Packtie-

fe der zu ladenden komprimierten Files (0 bis 6; siehe Sourcecode unter »optionequ«) einstellen. Aus Platzgründen finden Sie übrigens »WaHuD« und »decompd« nur auf der Programmservice-Diskette.

Beim Aufruf von »decomp« aus Ihrem Programm sollten Sie folgendes beachten:

Die Variable »_decsourc« sollte mit der Adresse versehen werden, an die das komprimierte Datenfile geladen wurde, »_decdest« mit der Adresse, an der die entpackten Daten später stehen sollen.

Dabei werden diese beiden Variablen im 32-Bit-Format angelegt und ebenso wie »_decomp« für »public« erklärt. Quell- und Zielbereich dürfen sich durchaus überschneiden, nur sollten Sie dafür Sorge tragen, daß die komprimierten Daten ans Ende – oder besser ein paar Words darüber hinaus – des Zielbereichs geladen werden.

Wenn all das geschehen ist, kann die Dekomprimier-Routine mit »bsr _decompd« oder »jsr _decompd« gestartet werden.

so

Der gute Ton

Ob Action-, Denk-
oder Aben-
teuerspiel: alle
profitieren
von einem guten
Sound. Mit
»IFF8SVXLoad«
vertonen
Sie Ihre Spiele.



Von Fridtjof Siebert
und Martin Jobst

Der Amiga MOdula Klub (AMOK) steht für eine Menge guter Programme aus der Modula-Ecke. Der Club betreibt eine eigene PD-Serie, auf der ausschließlich in Modula geschriebene Programme veröffentlicht werden. Wir stellen Ihnen »IFF8SVXLoad« von Amok-Programmierer Fridtjof Siebert vor.

Wollen Sie Ihre Spielprogramme mit einer Klangaura versehen, ist »IFF8SVXLoad« genau das Richtige. Damit lassen sich Sound-Samples von Modula-Programmen aus abspielen. Die Prozeduren (Listing 1) lassen sich nur in Modula-Bibliotheken einbauen. Im Beitrag von Frank Staudte auf Seite 130 finden Sie einige Anregungen, wie Sie Modula- in C-Programme konvertieren können.

Im folgenden wollen wir die verschiedenen Prozeduren von »IFF8SVXLoad« beschreiben. Das zugehörige Definitions-Modul finden Sie in Listing 2.

Laden...

Beim Laden von gesampelten Sounds verwenden Sie folgende Prozedur:

```
PROCEDURE Read8SVX  
(Name: ARRAY OF CHAR;  
MaxChunkSize: LONGCARD;  
ChipMem: BOOLEAN);  
IFF8SVXInfoPtr;
```

Diese lädt die Datei »Name«. Die Sounddaten werden dabei in Blöcken der Größe von »MaxChunkSize« (Byte) geladen. Ist »MaxChunkSize« gleich Null, wird versucht, alle Daten in einen einzigen Block zu laden. »ChipMem« entscheidet, ob die Datenblöcke im Chip-Memory liegen sollen, oder nicht. Dadurch würde das Abspielen erleichtert, da man die Blöcke nicht einzeln ins Chip-Memory kopieren muß.

Der Rückgabewert der Prozedur ist ein Zeiger auf die Struktur »IFF8SVXLoad.IFF8SVXInfo«, die alle geladenen

Daten enthält, oder auf »NIL«, wenn ein Fehler auftritt. Über den Fehler informiert die Variable »IFF8SVXLoad.IFF8SVXErrs«.

Probleme mit dieser Prozedur kann es geben, wenn ein langes Sound-File geladen, »MaxChunkSize« dabei relativ klein und außerdem der Stack nicht sonderlich groß ist. Es kann dann zu Fehlermeldungen

...freigeben...

gen (Adreßfehler, Stack Überlauf oder Guru) kommen.

Ist der Sound einmal im Speicher, wird er mit

```
PROCEDURE Dealloc8SVX  
(Info: IFF8SVXInfoPtr);
```

freigegeben.

Damit der Sound abgespielt wird, muß das Audio-Device geöffnet werden. Dies übernimmt die Prozedur »OpenAudio«:

```
PROCEDURE OpenAudio  
(Channels: CARDINAL;  
Priority: Byte);  
BOOLEAN;
```

»Channels« gibt an, wie viele Soundkanäle angesprochen werden sollen, wobei hardwaremäßig maximal vier möglich sind. »Priority« bestimmt die Priorität, mit der die Kanäle aktiviert werden sollen. Ihr Wertebereich liegt zwischen -128 (niedrigste) und 127 (höchste Priorität). Bei erfolgreicher Ausführung von OpenAudio() ist »TRUE« das Ergebnis, »FALSE« signalisiert einen aufgetretenen Fehler. Genauer über den Fehler erfahren Sie auch hier von »IFF8SVXError«.

Nach dem Öffnen der Soundkanäle kann der Sound gespielt werden. Dies geschieht mit:

```
PROCEDURE PlaySample  
(Info: IFF8SVXInfoPtr;  
Octave: INTEGER;  
Repeat: CARDINAL;  
Channel: CARDINAL);  
BOOLEAN;
```

»Info« ist der von »Read8SVX()« übergebene Zeiger, »Octave« bestimmt die Oktave des abgespielten Sounds und kann Werte zwischen 0 und 7 anneh-

men. Mit der Ausnahme von Instrumenten besitzen gesampelte Sounds meist nur eine Oktave (Nummer 0).

»Repeat« gibt an, wie oft der Sound wiederholt werden soll. Mit »Channel« bestimmen Sie den Audio-Kanal, auf dem der Sound ausgegeben wird (Werte zwischen 0 und dem bei »OpenAudio()« angegebenen Wert minus 1).

Bei erfolgreicher Ausführung gibt PlaySample »TRUE« zurück, ansonsten »FALSE«. Wieder finden Sie genauere Informationen zum Fehler in »IFF8SVXError«.

»PlaySample« spielt den

...und abspielen

Sound in einem eigenen Task, der die Daten aus dem Fast-Memory im »Doublebuffering«-Verfahren in das Chip-Memory kopiert und abspielt. Der bei jedem Durchgang kopierte Block hat maximal die bei »Read8SVX« angegebene Größe. Bei zu kleinen Blocks kann es zu Timing-Problemen kommen. Da ein neuer Task gestartet wurde, gibt »PlaySample« nach dem Aufruf sofort die Kontrolle an das aufrufende Programm zurück.

Rufen Sie »PlaySample« nacheinander mit verschiedenen Kanälen auf, können Sie verschiedene Sounds gleichzeitig spielen. Wird es auf einen Tonkanal angewandt, in dem bereits ein Sound läuft, wird dieser abgebrochen und durch den neuen ersetzt. Es ist auch möglich, auf allen vier Kanälen gleichzeitig ein und denselben Klang abzuspielen. Dabei können zu hohe Samplingraten oder zu klein gewählte »MaxChunkSize« zu leichten Störgeräuschen führen. Hier hilft nur eines: Experimentieren.

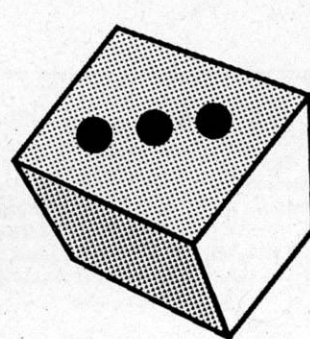
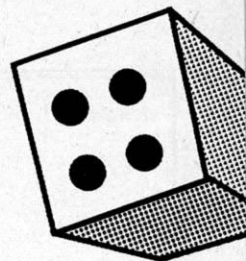
Während ein Sound gespielt wird, darf sein Speicherbereich nicht mit »Dealloc8SVX()« freigegeben werden.

Mit

```
PROCEDURE WaitPlay  
(Channel: CARDINAL);
```

wartet der Amiga, bis der mit »PlaySample()« gestartete

zum Spiel



Sound in Kanal »Channel« fertig gespielt ist und springt erst danach in den aufrufenden Task zurück. Wurde für »Channel« »IFF8SVXLoad.AllChannels« angegeben, wird gewar-

tet, bis alle Kanäle ihre Ausgabe beendet haben.

Genau gesagt springt »Wait-Play« aber schon zurück, kurz bevor der letzte Block gespielt wird. Dies ermöglicht es, direkt

danach ohne Zwangspause einen anderen Sound zu spielen.

Das Schließen des »Audio-Device« schließlich geschieht mit

```
PROCEDURE CloseAudio();
```

Der Task zum Abspielen der Sounds wird gestoppt und alle gerade gespielten Sounds werden abgebrochen.

In Tabelle 1 finden Sie eine Übersicht über die Bedeutung der von »IFF8SVX-Load« verwendeten Variablen und Typen. »IFF8SVX« und die folgenden Beispiele finden Sie auf der AMOK-Diskette Nr. 8.

Beispiele

Einige Beispiele sollen die Anwendung der beschriebenen Prozeduren verdeutlichen:

PlaySample (Listing 3) spielt von der Workbench den (mit <SHIFT linke Maustaste>) ge-

wählten oder im CLI als Argument angegebenen Sound auf einem Kanal einmal ab.

PlayRep (Listing 4) wird wie »PlaySample« gestartet und spielt den Sound endlos ab.

PlayKeys (Listing 5) wird analog gestartet. Danach kann man den Sound durch Drücken von verschiedenen Tasten in unterschiedlichen Tonlagen abspielen.

PlayKanon (Listing 6) gibt einen gewählten Sound auf allen vier Kanälen – jeweils um eine halbe Sekunde zeitversetzt – einmal aus. SO

Martin Jobst studiert Publizistik und Kommunikationswissenschaft in Salzburg und arbeitet seit drei Jahren als freier Mitarbeiter für Zeitschriften des Markt & Technik Verlages. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Martin Jobst, Hans-Pinsel-Straße 2, 8013 Haar).

Variablen und Typen

IFF8SVXInfo:

In einer Variablen dieses Typs werden die Daten eines geladenen Sounds gespeichert. Dabei handelt es sich um einen großen Record, der viele Subrecords enthält. Wer lediglich Sounds abspielen möchte, braucht über diese Variable nichts Genaueres zu wissen.

IFF8SVXInfo.loadedChunks	ist ein »SET«, das Flags enthält, die für jedes gültige (geladene) Subrecord gesetzt sind
IFF8SVXInfo.next/prev	wird nicht benutzt und kann zum Linken von mehreren »IFF8SVXInfos« verwendet werden
IFF8SVXInfo.VHDR	enthält wichtige Daten wie Geschwindigkeit, Größe, Lautstärke etc. des geladenen Sounds
IFF8SVXInfo.NAME/COPY/AUTH/ANNO	enthalten die Größe und einen Zeiger auf ein »ARRAY OF CHAR« des Soundnamens, Copyrights etc. Die Größe des Arrays beträgt maximal 1000 Zeichen.
IFF8SVXInfo.ATAK/RLSE	speichert Werte für die Lautstärke beim Start und Ausklingen von Samples. »PlaySample()« unterstützt »ATAK & RLSE« nicht.
IFF8SVXInfo.BODY	enthält die eigentlichen Sounddaten und Informationen, ob je Oktave ein einziger Block alloziert wurde (MaxChunkSize=0). Außerdem enthält es die Größe des umfangreichsten Blocks sowie ein Flag, das angibt, ob alle Daten im Chip-Memory liegen oder nicht. Es können bis zu acht Oktaven gespeichert werden. Pro Oktave wird eine Liste von »DataChunk«-Records angelegt, die einen Zeiger auf die Daten enthält.

IFF8SVXError:

Enthält die letzte Fehlernummer (IFF8SVXErrs). Mögliche Fehler:	
iff8OK	letztes Kommando war erfolgreich
iff8OutOfMem	nicht genügend Speicher frei
iff8Openfailed	Dos.Open() war erfolglos -> Dos.IOErr()
iff8Readfailed	Dos.Read() war erfolglos -> Dos.IOErr()
iff8NoIFF	Datei ist kein gesampelter IFF-Sound
iff8NoChannel	Kein Soundkanal mehr frei
iff8OpenDevicefailed	OpenDevice war erfolglos
Genauere Informationen zu den Variablen und Typen können »IFF8SVXLoad.def« (Listing 2) entnommen werden.	

```
(*-----*)
:Program.      IFF8SVXLoad.mod
:Author.       Fridtjof Siebert
:Address.      Nobileweg 67, D-7-Stgt-40
:Shortcut.     [fbs]
:Version.      .9
:Date.         18-Sep-88
:Copyright.    PD, no commercial use !!!
:Language.     Modula-II
:Translator.   M2Amiga
:Imports.      MemSystem [bne], Amok#5
:UpDate.       none.
:Contents.     Procedures to load and play sampled sounds.
:Remark.       Contact me if you want to use this in your own
               commercial
:Remark.       Software !
(*-----*)
```

IMPLEMENTATION MODULE IFF8SVXLoad;

```
(*----- IMPORTS: -----*)

(*----- SYSTEM: -----*)
FROM SYSTEM      IMPORT ADR, ADDRESS, LONGSET, SHIFT, CAST;
FROM Arts        IMPORT Assert, TermProcedure, BreakPoint;

(*----- Libraries: -----*)
FROM Audio       IMPORT IOAudioPtr, IOAudio, audioName, allocFailed,
                    perVol;
```

Listing 1. Die Prozeduren von »IFF8SVXLoad«

Tabelle 1. Die Funktionen der Variablen von »IFF8SVXLoad«


```

FROM Dos      IMPORT Open, Close, oldFile, Read, FileHandlePtr,
                Seek, current;
FROM Exec     IMPORT MsgPortPtr, DevicePtr, UByte, Byte, write,
                IOFlagSet, ReplyMsg, GetMsg, OpenDevice, CopyMem,
                CloseDevice, WaitPort, AddTask, RemTask, TaskPtr,
                Forbid, Permit, NodeType, MsgPortAction, FindTask,
                Wait, Signal, AllocSignal, FreeSignal;
FROM ExecSupport IMPORT CreatePort, DeletePort, NewList, BeginIO,
                AbortIO;

(*----- Standard: -----*)
FROM Strings  IMPORT Compare, first, last;

(*----- Modules: -----*)
FROM MemSystem IMPORT ErrHeader, Allocate, AllocMem, Deallocate,
                YesNoRequest, RETRY, CANCEL, ExitQuiet;

(*----- CONSTANTS: -----*)

CONST
unity = 10000H;    (* This is Fixed(1) = 1.0000H *)

sCompNone = 0;    (* no compression *)
sCompFibDelta = 1; (* Fibonacci-Delta encoding *)

(*----- TYPES: -----*)

TYPE
Fixed = LONGINT; (* Fixed Point Value (xxxx.xxxxH) *)
IFF8SVXChunks = (VHDR, NAME, COPY, AUTH, ANNO, ATAK, RLSE, BODY);
IFF8SVXChunkSet = SET OF IFF8SVXChunks;

(*----- Type for Data-Chunks: -----*)
DataChunkPtr = POINTER TO DataChunk;
DataChunk = RECORD
    next: DataChunkPtr; (* to link them *)
    prev: DataChunkPtr;
    size: LONGCARD;      (* size of this Chunk *)
    data: ADDRESS;       (* where to find it *)
END;

(*----- String: -----*)
String = ARRAY[0..999] OF CHAR;
(* This is used for smaller strings ! So don't modify them
(insert sth.) ! *)

(*----- Type to contain loaded data: -----*)
IFF8SVXInfoPtr = POINTER TO IFF8SVXInfo;
IFF8SVXInfo = RECORD
    loadedChunks: IFF8SVXChunkSet;
    (* all Sub-RECORDs whose flag is set here contain legal data *)
    next, prev: IFF8SVXInfoPtr;
    (* unused. Can be used to link IFF8SVXInfo's *)
    VHDR: RECORD (* 8SVX's header chunk *)
        oneShotHiSamples: LONGCARD; (* # of Samples in
            shotpart Oct.1 *)
        repeatHiSamples: LONGCARD; (* # of Repeatsamples in
            Oct. 1 *)
        samplesPerHiCycle: LONGCARD; (* # Samples/Cycle or 0 *)
        samplesPerSec: CARDINAL; (* Samplingrate *)
        countOctave: Byte; (* counts octaves *)
        sCompression: Byte; (* Compression type or
            0 if none *)
        volume: Fixed; (* Volume (0..10000H) *)
    END;
    NAME: RECORD (* Sound's Name *)
        size: LONGCARD; (* Length *)
        string: POINTER TO String; (* Name *)
    END;
    COPY: RECORD (* Sound's Copyright *)
        size: LONGCARD;
        string: POINTER TO String;
    END;
    AUTH: RECORD (* Sound's Author *)
        size: LONGCARD;
        string: POINTER TO String;
    END;
    ANNO: RECORD (* Author's Annotation *)
        size: LONGCARD;
        string: POINTER TO String;
    END;
END;

```

```

ATAK, RLSE: RECORD (* Attack & Decay duration *)
    duration: CARDINAL; (* Duration in milliseconds *)
    dest: Fixed; (* Destination Volume *)
END;
BODY: RECORD (* Data of sampled Voice *)
    oneChunk: BOOLEAN; (* Just one Chunk per
        Octave ? *)
    maxChunkSize: LONGCARD; (* Highest Chunk-Size *)
    dataInChip: BOOLEAN; (* Chunks in Chip-Memory ? *)
    soundData: ARRAY[0..7] OF DataChunkPtr; (* <= 8 Octaves
        of Data *)
END;

(*----- Errors: -----*)
IFF8SVXErrors = (iff8OK, iff8OutOfMem, iff8Openfailed, iff8Readfailed,
    iff8NoIFF,
    iff8NoChannel, iff8OpenDevicefailed);

(*----- Variables: -----*)
VAR
    IFF8SVXError: IFF8SVXErrors; (* Last Error *)

(*----- internal Variables: -----*)

TYPE
    ExtIOAudio = RECORD
        ioa: IOAudio;
        id: CARDINAL;
    END;
    ExtIOAudioPtr = POINTER TO ExtIOAudio;
VAR
    InH: FileHandlePtr;
    Buffer: ADDRESS; (* Buffer for Reading / Writing *)
    TextBuffer: POINTER TO ARRAY[0..63] OF ARRAY[0..3] OF CHAR;
    LONGBuffer: POINTER TO ARRAY[0..63] OF LONGCARD;
    WORDBuffer: POINTER TO ARRAY[0..127] OF CARDINAL;
    LongPtr, LONGPtr: POINTER TO LONGCARD;
    i, j: LONGCARD;
    len: LONGCARD; (* number of bytes read from file *)
    ChunkSize: LONGCARD; (* size of loaded octave *)
    AllocPort: MsgPortPtr;
    AllocIOB: IOAudioPtr;
    Device: DevicePtr;
    SoundPort: MsgPortPtr;
    SoundIOB: IOAudioPtr;
    AllocationMap: ARRAY[0..3] OF Byte;
    SoundXIOA: ARRAY[0..3], [0..1] OF ExtIOAudioPtr;
    SoundInfo: ARRAY[0..3] OF RECORD
        info: IFF8SVXInfoPtr;
        chunk: DataChunkPtr; (* active Chunk *)
        oct: CARDINAL; (* octave to play *)
        repCnt: CARDINAL; (* how often to repeat *)
        bufSize: LONGCARD; (* size of Buffer *)
        playing: BOOLEAN; (* playing or ready ? *)
        done: BOOLEAN; (* sound played ? *)
        dblBuf: BOOLEAN; (* DoubleBuffering ? *)
    END;
    ChannelDone: ARRAY[0..3] OF ExtIOAudioPtr;
    PlayTask: TaskPtr;
    PlayStack: ADDRESS;
    SoundSignal: LONGINT;
    SoundTask: TaskPtr;

(*----- Load an Octave: -----*)
PROCEDURE LoadChunk(Size, MaxSize: LONGCARD;
    Prev: DataChunkPtr;
    ChipMem: BOOLEAN): DataChunkPtr;
VAR
    Chunk: DataChunkPtr;
BEGIN
    Allocate(Chunk, SIZE(DataChunk));
    IF Chunk=NIL THEN
        IFF8SVXError := iff8OutOfMem;
        RETURN NIL;
    END;
    WITH Chunk DO
        prev := Prev;
        IF (Size <= MaxSize) OR (MaxSize=0) THEN
            size := Size;
        ELSE
            size := MaxSize;
        END;
    END;
END;

```



```

DEC(Size,size);
AllocMem(data,size,ChipMem);
IF data=NIL THEN
  IFF8SVXError := iff8OutOfMem;
  Deallocate(Chunk);
  RETURN NIL;
END;
len := Read(InH,data,size);
IF len#size THEN
  IFF8SVXError := iff8Readfailed;
  Deallocate(data);
  Deallocate(Chunk);
  RETURN NIL;
END;
IF Size#0 THEN
  next := LoadChunk(Size,MaxSize,Chunk,ChipMem);
  IF next=NIL THEN
    Deallocate(data);
    Deallocate(Chunk);
    RETURN NIL;
  END;
ELSE
  next := NIL;
END;
END; (* WITH Chunk DO *)
RETURN Chunk;
END LoadChunk;

(*-----*)
(*
(*          Load Sampled Sound:
(*
(*-----*)

PROCEDURE Read8SVX(Name: ARRAY OF CHAR;
  MaxChunkSize: LONGCARD;
  ChipMem: BOOLEAN): IFF8SVXInfoPtr;
(* Name:      Sound's Name
(* MaxChunkSize: MaxSize of Data-Chunk or 0 to load to a single
(* chunk
(* ChipMem:    TRUE if you want all chunks in ChipMem.
(* Result:     Pointer to info of loaded sample or NIL if any
(* error
(* occurred. IFF8SVXError contains errortype.
VAR
  Info: IFF8SVXInfoPtr;
BEGIN
  IFF8SVXError := iff8OK;
  Allocate(Info,SIZE(Info));
  IF Info=NIL THEN
    IFF8SVXError := iff8OutOfMem;

    RETURN NIL;
  END;

(*----- Open File: -----*)

  LOOP
    InH := Open(ADR(Name),oldFile);
    IF InH=NIL THEN
      IFF8SVXError := iff8Openfailed;
      EXIT;
    END;

(*----- File Header: -----*)

    len := Read(InH,Buffer,12);
    IF len=0 THEN
      IFF8SVXError := iff8Readfailed;
      EXIT;
    END;
    IF (Compare(TextBuffer[0],first,4,"FORM",TRUE)#0) OR
      (Compare(TextBuffer[2],first,4,"8SVX",TRUE)#0) THEN
      IFF8SVXError := iff8NoIFF;
      EXIT;
    END;

(*----- Main Loop: -----*)

    LOOP
      len := Read(InH,Buffer,8);

      IF len#8 THEN

```

```

      IFF8SVXError := iff8Readfailed;
      EXIT;
    END;

(*----- VHDR: -----*)
    IF Compare(TextBuffer[0],first,4,"VHDR",TRUE)=0 THEN
      INCL(Info.loadedChunks,VHDR);
      len := Read(InH,Buffer,LONGBuffer[1]);
      LONGPtr := Buffer;
      LongPtr := ADR(Info.VHDR);
      FOR i:=0 TO 4 DO
        LongPtr := LONGPtr;
        INC(LongPtr,4);
        INC(LONGPtr,4);
      END;

(*----- NAME: -----*)
    ELSIF Compare(TextBuffer[0],first,4,"NAME",TRUE)=0 THEN
      INCL(Info.loadedChunks,NAME);
      WITH Info.NAME DO
        size := LONGBuffer[1]+1;
        AllocMem(string,size,TRUE);
        IF string=NIL THEN IFF8SVXError := iff8OutOfMem; EXIT END;
        len := Read(InH,string,size-1);
        IF ODD(size-1) THEN len := Read(InH,Buffer,1) END;
      END;

(*----- COPY: -----*)
    ELSIF Compare(TextBuffer[0],first,4,"(c)",TRUE)=0 THEN
      INCL(Info.loadedChunks,COPY);
      WITH Info.COPY DO
        size := LONGBuffer[1]+1;
        AllocMem(string,size,TRUE);
        IF string=NIL THEN IFF8SVXError := iff8OutOfMem; EXIT END;
        len := Read(InH,string,size-1);
        IF ODD(size-1) THEN len := Read(InH,Buffer,1) END;
      END;

(*----- AUTH: -----*)
    ELSIF Compare(TextBuffer[0],first,4,"AUTH",TRUE)=0 THEN
      INCL(Info.loadedChunks,AUTH);
      WITH Info.AUTH DO
        size := LONGBuffer[1]+1;
        AllocMem(string,size,TRUE);
        IF string=NIL THEN IFF8SVXError := iff8OutOfMem; EXIT END;
        len := Read(InH,string,size-1);
        IF ODD(size-1) THEN len := Read(InH,Buffer,1) END;
      END;

(*----- ANNO: -----*)
    ELSIF Compare(TextBuffer[0],first,4,"ANNO",TRUE)=0 THEN
      INCL(Info.loadedChunks,ANNO);
      WITH Info.ANNO DO
        size := LONGBuffer[1]+1;
        AllocMem(string,size,TRUE);
        IF string=NIL THEN IFF8SVXError := iff8OutOfMem; EXIT END;
        len := Read(InH,string,size-1);

        IF ODD(size-1) THEN len := Read(InH,Buffer,1) END;
      END;

(*----- ATAK: -----*)
    ELSIF Compare(TextBuffer[0],first,4,"ATAK",TRUE)=0 THEN
      INCL(Info.loadedChunks,ATAK);
      len := Read(InH,Buffer,LONGBuffer[1]);
      Info.ATAK.duration := WORDBuffer[0];
      LONGPtr := ADDRESS(LONGCARD(Buffer) + 2);
      Info.ATAK.dest := LONGPtr;

(*----- RLSE: -----*)
    ELSIF Compare(TextBuffer[0],first,4,"RLSE",TRUE)=0 THEN
      INCL(Info.loadedChunks,RLSE);
      len := Read(InH,Buffer,LONGBuffer[1]);
      Info.RLSE.duration := WORDBuffer[0];
      LONGPtr := ADDRESS(LONGCARD(Buffer) + 2);
      Info.RLSE.dest := LONGPtr;

```

Listing 1. (Fortsetzung)


```

(*----- BODY: -----*)
ELSIF Compare(TextBuffer[0],first,4,"BODY",TRUE)=0 THEN
  IF NOT(VHDR IN Info.loadedChunks) THEN
    IFF8SVXError := iff8NoIFF;
    EXIT;
  END;
  INCL(Info.loadedChunks,BODY);
  WITH Info.BODY DO
    oneChunk := TRUE;
    dataInChip := ChipMem;
    WITH Info.VHDR DO
      ChunkSize := oneShotHiSamples + repeatHiSamples;
      FOR i:=0 TO Info.VHDR.countOctave-1 DO
        soundData[i] := LoadChunk(ChunkSize,MaxChunkSize,
          NIL,ChipMem);
        IF soundData[i]=NIL THEN EXIT END;
        maxChunkSize := ChunkSize;
        INC(ChunkSize,ChunkSize);
      END;
      IF maxChunkSize>MaxChunkSize THEN
        maxChunkSize := MaxChunkSize;
      END;
    END;
  END;
  EXIT;
(*----- Unknown Chunk: -----*)
ELSE
  IF ODD(LONGBuffer[1]) THEN
    len := Seek(InH,current,LONGBuffer[1]+1);
  ELSE
    len := Seek(InH,current,LONGBuffer[1]);
  END;
  END;
END;

END; (* LOOP *)

(*----- EXIT & Error check: -----*)
IF NOT((VHDR IN Info.loadedChunks) OR (BODY IN Info.loaded
Chunks)) THEN
  IFF8SVXError := iff8NoIFF;
  END;

  EXIT;

END; (* LOOP *)

IF IFF8SVXError#iff8OK THEN
  Dealloc8SVX(Info);
  Info := NIL;
END;

Close(InH); InH := NIL;

RETURN Info;

END Read8SVX;

(*----- Free DataChunk's Memory: -----*)

PROCEDURE FreeChunks(first: DataChunkPtr);

BEGIN
  WITH first DO
    IF next#NIL THEN FreeChunks(next) END;
    IF data#NIL THEN
      Deallocate(data);
    END;
  END;
  Deallocate(first);
END FreeChunks;

(*-----*)
(* Free Sound's Memory: *)
(*-----*)

```

```

PROCEDURE Dealloc8SVX(Info:IFF8SVXInfoPtr);

(* Info: Sound's Info-
Record *)

VAR
  i: CARDINAL;

BEGIN
  IF Info#NIL THEN
    IF NAME IN Info.loadedChunks THEN
      WITH Info.NAME DO
        IF string#NIL THEN Deallocate(string) END;
      END;
    END;
    IF COPY IN Info.loadedChunks THEN
      WITH Info.COPY DO
        IF string#NIL THEN Deallocate(string) END;
      END;
    END;
    IF AUTH IN Info.loadedChunks THEN
      WITH Info.AUTH DO
        IF string#NIL THEN Deallocate(string) END;
      END;
    END;
    IF ANNO IN Info.loadedChunks THEN
      WITH Info.ANNO DO
        IF string#NIL THEN Deallocate(string) END;
      END;
    END;
    IF BODY IN Info.loadedChunks THEN
      FOR i:=0 TO Info.VHDR.countOctave-1 DO
        IF Info.BODY.soundData[i]#NIL THEN
          FreeChunks(Info.BODY.soundData[i]);
        END;
      END;
    END;
    Deallocate(Info);
  END;
END Dealloc8SVX;

(*----- Task to play Sound: -----*)

PROCEDURE PlayTaskProc(); (* $S- *)

VAR
  XIOA: ExtIOAudioPtr;
  i: CARDINAL;

BEGIN
  LOOP
    WaitPort(SoundPort);
    XIOA := GetMsg(SoundPort);
    IF (XIOA#NIL) AND (XIOA.request.error=0) THEN
      WITH SoundInfo[XIOA.id] DO
        Forbid();
        IF done THEN
          ChannelDone[XIOA.id] := NIL;
        END;
        IF playing THEN
          IF chunk.next#NIL THEN
            chunk := chunk.next;
          ELSIF repCnt>1 THEN
            DEC(repCnt);
            chunk := info.BODY.soundData[oct];
          ELSE
            done := TRUE;
            ChannelDone[XIOA.id] := XIOA;
            playing := FALSE;
            Signal(SoundTask, LONGSET{SoundSignal})
          END;
        END;
      END;
    IF NOT(done) THEN
      playing := TRUE;
      IF dblBuf THEN
        CopyMem(chunk.data,XIOA.ioa.data,chunk.size);
      ELSE
        XIOA.ioa.data := chunk.data;
      END;
      XIOA.ioa.length := chunk.size;
    END;
  END;

```



```

BeginIO(XIOA);
END;
Permit();
END; (* WITH SoundInfo[XIOA.id] *)
END; (* IF Msg ok THEN *)
END; (* endless LOOP *)
END PlayTaskProc;

(*-----*)
(*
(*           Open Audio Device:
(*
(*-----*)

PROCEDURE OpenAudio(Channels: CARDINAL; Priority: Byte): BOOLEAN;

(* Channels: Number of soundchannels to allocate
(* Priority: Allocation Priority

BEGIN
    IFF8SVXError := iff8OK;

(*----- Allocation Precedence and Channel: -----*)

    AllocIOB.request.message.node.pri := -40;
    AllocIOB.request.message.replyPort := AllocPort;

CASE Channels OF
1: AllocationMap[0] := 1;
   AllocationMap[1] := 2;
   AllocationMap[2] := 4;
   AllocationMap[3] := 8;
2: AllocationMap[0] := 3;
   AllocationMap[1] := 6;
   AllocationMap[2] := 12;
   AllocationMap[3] := 5;
3: AllocationMap[0] := 14;
   AllocationMap[1] := 13;
   AllocationMap[2] := 11;
   AllocationMap[3] := 7;
4: AllocationMap[0] := 15;
   AllocationMap[1] := 15;
   AllocationMap[2] := 15;
   AllocationMap[3] := 15;
END;
AllocIOB.data := ADR(AllocationMap);
AllocIOB.length := SIZE(AllocationMap);

(*----- Open Audio-Device: -----*)

OpenDevice(ADR(audioName),0,AllocIOB,LongSet{});
(* Why doesn't OpenDevice() return it's error ??? *)

CASE AllocIOB.request.error OF
-1,allocFailed:
    IFF8SVXError := iff8OpenDeviceFailed;
    RETURN FALSE;
ELSE
END;

Device := AllocIOB.request.device;

(*----- Initialize ReplyPort: -----*)

WITH SoundPort DO
    flags := signal;
    node.type := msgPort;
END;
NewList(ADR(SoundPort.msgList));

(*----- Initialize I/O-Block: -----*)

j := 0;
FOR i:=0 TO 3 DO
    WITH SoundXIOA[i,0] DO
        id := i;
        ioa.request.message.replyPort := SoundPort;
        ioa.request.device := Device;
        WHILE (j<4) AND NOT(j IN CAST(LongSet,AllocIOB.request.unit))
        DO
            INC(j);

```

```

END;
IF j<4 THEN
    ioa.request.unit := CAST(ADDRESS,LongSet{j});
    DEC(Channels,1);
ELSE
    ioa.request.unit := NIL;
END;
INC(j);
ioa.request.command := write;
ioa.request.flags := IOFlagSet{4}; (* perVol *)
ioa.allocKey := AllocIOB.allocKey;
END;
SoundXIOA[i,1] := SoundXIOA[i,0];
SoundInfo[i].playing := FALSE;
SoundInfo[i].done := TRUE;
ChannelDone[i] := NIL;
END;

(*----- Start PlayTask: -----*)

Allocate(PlayTask,SIZE(PlayTask));
IF PlayTask=NIL THEN ExitQuiet END;
SoundPort.sigTask := PlayTask;
WITH PlayTask DO
    spLower := PlayStack;
    spUpper := ADDRESS(LongCard(PlayStack) + 1000);
    spReg := spUpper;
    node.type := task;
    node.name := ADR("SamplePlayTask");
END;
AddTask(PlayTask,ADR(PlayTaskProc),NIL);

IF Channels>0 THEN
    CloseAudio();
    IFF8SVXError := iff8NoChannel;
    RETURN FALSE;
END;

RETURN TRUE;

END OpenAudio;

(*-----*)
(*
(*           Close Audio Device:
(*
(*-----*)

PROCEDURE CloseAudio();

BEGIN
    RemTask(PlayTask);
    Deallocate(PlayTask);
    PlayTask := NIL;
    CloseDevice(AllocIOB);
    Device := NIL;
    FOR i:=0 TO 3 DO
        IF SoundInfo[i].dblBuf THEN
            WITH SoundXIOA[i,0].ioa DO
                IF data#NIL THEN Deallocate(data) END;
            END;
        WITH SoundXIOA[i,1].ioa DO
            IF data#NIL THEN Deallocate(data) END;
        END;
    END;
END;
END CloseAudio;

(*-----*)
(*
(*           Play sampled Sound:
(*
(*-----*)

PROCEDURE PlaySample(Info: IFF8SVXInfoPtr;
    Octave: INTEGER;
    Repeat: CARDINAL;
    Channel: CARDINAL): BOOLEAN;

```

Listing 1. (Fortsetzung)


```

(* Info:      Sound's IFF8SVXInfo                      *)
(* Octave:    Octave to play (0..7)                    *)
(* Repeat:    how often to repeat sound                *)
(* Channel:    Channel to play sound                   *)

VAR
  Reply: BOOLEAN;

(*----- Fun Starts: -----*)

BEGIN

  IF Info=NIL THEN
    IFF8SVXError := iff8NoIFF;
    RETURN FALSE;
  END;

  IF Octave >= Info.VHDR.countOctave THEN RETURN TRUE END;

  WITH SoundInfo[Channel] DO
    Forbid();
    Reply := done;
    AbortIO(SoundXIOA[Channel,0]);
    AbortIO(SoundXIOA[Channel,1]);
    WITH Info DO
      info := Info;
      chunk := BODY.soundData[Octave];
      oct := Octave;
      repCnt := Repeat;
      playing := FALSE;
      done := FALSE;
      bufSize := Info.BODY.maxChunkSize;
      IF dblBuf THEN
        FOR i:=0 TO 1 DO
          WITH SoundXIOA[Channel,i].ioa DO
            IF data#NIL THEN Deallocate(data) END;
          END;
        END;
      END;
      dblBuf := NOT(BODY.dataInChip);
      FOR i:=0 TO 1 DO;
        WITH SoundXIOA[Channel,i].ioa DO
          period := LONGINT(3584200) DIV LONGINT
            (VHDR.samplesPerSec);
          volume := SHIFT(VHDR.volume,-10);
          cycles := 1;
          IF dblBuf THEN
            AllocMem(data,bufSize,TRUE);
            IF data=NIL THEN
              IFF8SVXError := iff8OutOfMem;
              RETURN FALSE;
            END;
          ELSE
            data := NIL;
          END;
        END;
      END;
      IF Reply THEN
        IF ChannelDone[Channel] = NIL THEN
          ReplyMsg(SoundXIOA[Channel,0]);
          ReplyMsg(SoundXIOA[Channel,1]);
        ELSE
          ReplyMsg(ChannelDone[Channel]);
        END;
      END;
    END;
  END;
  Permit();
END;

IFF8SVXError := iff8OK;

RETURN TRUE;

END PlaySample;

(*-----*)
(* Wait for Play to finish: *)
(*-----*)

```

```

(*)
CONST
  AllChannels = 4;
*)

PROCEDURE WaitPlay(Channel: CARDINAL);

(* This waits for a channel to complete. *)
(* WaitPlay(AllChannels) waits for all channels to finish. *)

VAR
  i: CARDINAL;
  x: BOOLEAN;

BEGIN
  IF Channel<AllChannels THEN
    IF NOT(SoundInfo[Channel].done) THEN
      IF Wait(LONGSET[SoundSignal])=LONGSET[] THEN END;
    END;
  ELSE
    LOOP
      x := TRUE;
      FOR i:=0 TO 3 DO
        x := x AND SoundInfo[i].done;
      END;
      IF x THEN EXIT END;
      IF Wait(LONGSET[SoundSignal])=LONGSET[] THEN END;
    END;
  END;
END WaitPlay;

(*----- Cleanup: -----*)

PROCEDURE Cleanup();

BEGIN
  IF PlayTask #NIL THEN
    RemTask(PlayTask);
    Deallocate(PlayTask);
  END;
  IF Device#NIL THEN CloseDevice(AllocIOB) END;
  IF Buffer#NIL THEN Deallocate(Buffer) END;
  IF InH#NIL THEN Close(InH) END;
  IF AllocIOB #NIL THEN Deallocate(AllocIOB) END;
  IF AllocPort#NIL THEN DeletePort(AllocPort) END;
  IF SoundPort#NIL THEN Deallocate(SoundPort) END;
  IF PlayStack#NIL THEN Deallocate(PlayStack) END;
  FOR i:=0 TO 3 DO
    IF SoundInfo[i].dblBuf THEN
      WITH SoundXIOA[i,0].ioa DO
        IF data#NIL THEN Deallocate(data) END;
      END;
      WITH SoundXIOA[i,1].ioa DO
        IF data#NIL THEN Deallocate(data) END;
      END;
    END;
    IF SoundXIOA[i,0]#NIL THEN Deallocate(SoundXIOA[i,0]) END;
    IF SoundXIOA[i,1]#NIL THEN Deallocate(SoundXIOA[i,1]) END;
  END;
  FreeSignal(SoundSignal);
END Cleanup;

(*----- Initialization: -----*)

BEGIN
  ErrHeader := "Error loading Sampled Sound:";
  AllocMem(Buffer,768,TRUE);
  Assert(Buffer#NIL,ADR("Not enough ChipMem !!!"));
  TextBuffer := Buffer;
  LONGBuffer := Buffer;
  WORDBuffer := Buffer;
  InH := NIL;
  AllocPort := CreatePort(ADR("Sampled SoundPort"),0);
  IF AllocPort=NIL THEN ExitQuiet END;
  Allocate(AllocIOB,SIZE(AllocIOB));
  IF AllocPort=NIL THEN ExitQuiet END;
  FOR i:=0 TO 3 DO
    Allocate(SoundXIOA[i,0],SIZE(ExtIOAudio));
    IF SoundXIOA[i,0]=NIL THEN ExitQuiet END;
    Allocate(SoundXIOA[i,1],SIZE(ExtIOAudio));
    IF SoundXIOA[i,1]=NIL THEN ExitQuiet END;
    SoundInfo[i].dblBuf := FALSE;
  END;

```



```

SoundInfo[i].done := TRUE;
END;
Allocate(SoundPort,SIZE(SoundPort));
IF SoundPort=NIL THEN ExitQuiet END;
Allocate(PlayStack,1000);
IF PlayStack=NIL THEN ExitQuiet END;
PlayTask := NIL;
SoundSignal := AllocSignal(-1);
Assert(SoundSignal>0,ADR("No more Signalbits !!!"));
SoundTask := FindTask(0);
Device := NIL;
TermProcedure(CleanUp);
END IFF8SVXLoad.

```

Listing 1. (Schluß)

```

(*-----*)
:Program.    IFF8SVXLoad.mod
:Author.     Fridtjof Siebert
:Address.    Nobileweg 67, D-7-Stgt-40
:Shortcut.   [fbs]
:Version.    .9
:Date.       18-Sep-88
:Copyright.  PD, no commercial use !!!
:Language.   Modula-II
:Translator. M2Amiga
:Imports.    MemSystem [bne], Amok#5
:UpDate.     none.
:Contents.   Procedures to load and play sampled sounds.
:Remark.     Contact me if you want to use this in your own
:Remark.     commercial
:Remark.     Software !
(*-----*)

DEFINITION MODULE IFF8SVXLoad;

(*----- IMPORTS: -----*)

(*----- SYSTEM: -----*)

FROM SYSTEM IMPORT ADR, ADDRESS;

(*----- Libraries: -----*)

FROM Exec IMPORT Byte;

(*----- CONSTANTS: -----*)

CONST

unity = 10000H; (* This is Fixed(1) = 1.0000H *)

sCompNone = 0; (* no compression *)
sCmpFibDelta = 1; (* Fibonacci-Delta encoding *)

(*----- TYPES: -----*)

TYPE

Fixed = LONGINT; (* Fixed Point Value (xxxx.xxxxH) *)

IFF8SVXChunks = (VHDR,NAME,COPY,AUTH,ANNO,ATAK,RLSE,BODY);
IFF8SVXChunkSet = SET OF IFF8SVXChunks;

(*----- Type for Data-Chunks: -----*)

DataChunkPtr = POINTER TO DataChunk;
DataChunk = RECORD
    next: DataChunkPtr; (* to link them *)
    prev: DataChunkPtr;
    size: LONGCARD; (* size of this Chunk *)
    data: ADDRESS; (* where to find it *)
END;

```

```

(*----- String: -----*)

String = ARRAY[0..999] OF CHAR;
(* This is used for smaller strings ! So don't modify them
(insert sth.) ! *)

(*----- Type to contain loaded data: -----*)

IFF8SVXInfoPtr = POINTER TO IFF8SVXInfo;

IFF8SVXInfo = RECORD
    loadedChunks: IFF8SVXChunkSet;
    (* all Sub-RECORDs whose flag is set here contain legal data *)

    next,prev: IFF8SVXInfoPtr;
    (* unused. Can be used to link IFF8SVXInfo's *)

VHDR: RECORD (* 8SVX's header chunk *)
    oneShotHiSamples: LONGCARD; (* # of Samples in
                                shotpart Oct.1 *)
    repeatHiSamples: LONGCARD; (* # of Repeatsamples in
                                Oct. 1 *)
    samplesPerHiCycle: LONGCARD; (* # Samples/Cycle
                                or 0 *)
    samplesPerSec: CARDINAL; (* Sampling-rate *)
    countOctave: Byte; (* counts octaves *)
    sCompression: Byte; (* Compression type or 0
                        if none *)
    volume: Fixed; (* Volume (0..10000H) *)
END;

NAME: RECORD (* Sound's Name *)
    size: LONGCARD; (* Length *)
    string: POINTER TO String; (* Name *)
END;

COPY: RECORD (* Sound's CopyRight *)
    size: LONGCARD;
    string: POINTER TO String;
END;

AUTH: RECORD (* Sound's Author *)
    size: LONGCARD;
    string: POINTER TO String;
END;

ANNO: RECORD (* Author's Annotation *)
    size: LONGCARD;
    string: POINTER TO String;
END;

ATAK,RLSE: RECORD (* Attack & Decay duration *)
    duration: CARDINAL; (* Duration in milliseconds *)
    dest: Fixed; (* Destination Volume *)
END;

BODY: RECORD (* Data of sampled Voice *)
    oneChunk: BOOLEAN; (* Just one Chunk per Octave ? *)
    maxChunkSize: LONGCARD; (* Highest Chunk-Size *)
    dataInChip: BOOLEAN; (* Chunks in Chip-Memory ? *)
    soundData: ARRAY[0..7] OF DataChunkPtr; (* <= 8 Octaves
                                                of Data *)
END;

END;

(*----- Errors: -----*)

IFF8SVXErrors = (iff8OK,iff8OutOfMem,iff8Openfailed,iff8Readfailed,
iff8NoIFF,iff8NoChannel,iff8OpenDevicefailed);

(*----- Variables: -----*)

VAR
    IFF8SVXError: IFF8SVXErrors; (* Last Error *)

```

Listing 2. Das Definitionsmudul zu »IFF8SVXLoad«


```

(*-----*)
(* *)
(* *)
(* *)
(* *)
(*-----*)
PROCEDURE Read8SVX(Name: ARRAY OF CHAR;
  MaxChunkSize: LONGCARD;
  ChipMem: BOOLEAN): IFF8SVXInfoPtr;

(* :Input. Name = Sound's Name and Path *)
(* :Input. MaxChunkSize = Maximum Size of Data-Chunks or 0 to
  load to a *)
(* :Input. single chunk *)
(* :Input. ChipMem = TRUE if you want all chunks in ChipMem *)
(* :Result. Pointer to Info of loaded sample or NIL if any error
  occurred. *)
(* :Result. IFF8SVXError contains errortype. *)
(* :Semantic. Loads Sampled sound. *)

(*-----*)
(* *)
(* *)
(* *)
(* *)
(*-----*)

PROCEDURE Dealloc8SVX(Info: IFF8SVXInfoPtr);

(* :Input. Sound's Info-Record *)
(* :Semantic. Frees Sound's Memory. *)

(*-----*)
(* *)
(* *)
(* *)
(* *)
(*-----*)

PROCEDURE OpenAudio(Channels: CARDINAL; Priority: Byte): BOOLEAN;

(* :Input. Channels = Number of soundchannels to allocate *)
(* :Input. Priority = Allocation Priority *)
(* :Result. FALSE in case of any error *)
(* :Semantic. Opens Audio Device and allocates Soundchannels *)
(* :Note. Detects whether OpenDevice() fails or not! *)

(*-----*)
(* *)
(* *)
(* *)
(* *)
(*-----*)

PROCEDURE CloseAudio();
(* :Semantic. Closes Audio Device. Don't forget this! *)

(*-----*)
(* *)
(* *)
(* *)
(* *)
(*-----*)

PROCEDURE PlaySample(Info: IFF8SVXInfoPtr;
  Octave: INTEGER;
  Repeat: CARDINAL;
  Channel: CARDINAL): BOOLEAN;

(* :Input. Info = Sound's IFF8SVXInfo *)
(* :Input. Octave = Octave to play (0..7) *)
(* :Input. Repeat = how often to repeat sound *)
(* :Input. Channel = Channel to play sound *)
(* :Result. FALSE if any error occurred *)
(* :Semantic. Plays sampled sound *)
(* :Note. Sounds are played with a second task feeding the audio
  device. *)
(* :Note. If you call PlaySample with a sound actually playing

```

```

on the *)
(* :Note. specified Channel, the old sound will be stopped. *)

(*-----*)
(* *)
(* *)
(* *)
(* *)
(*-----*)

CONST
  AllChannels = 4;
PROCEDURE WaitPlay(Channel: CARDINAL);

(* :Input. Channel = Channel to wait for or AllChannels *)
(* :Semantic. This waits for a channel to complete. *)
(* :Semantic. WaitPlay(AllChannels) waits for all channels to
  finish. *)

END IFF8SVXLoad.

```

Listing 2. (Schluß)

```

(*-----*)
:Program. PlaySample.mod
:Author. Fridtjof Siebert
:Address. Nobileweg 67, D-7-Stgt-40
:Shortcut. [fbs]
:Version. 1.0
:Date. 19-Sep-88
:Copyright. PD
:Language. Modula-II
:Translator. M2Amiga
:Imports. IFF8SVXLoad [fbs], MemSystem [bne]
:UpDate. none.
:Contents. Program to play a sampled IFF-Sound.
:Remark. Usage: PlaySample <iff-soundfile>
(*-----*)

MODULE PlaySample;

FROM SYSTEM IMPORT ADR, ADDRESS, SHIFT, BITSET, LONGSET, CAST;
FROM Arts IMPORT Terminate;
FROM Arguments IMPORT NumArgs, GetArg;
FROM Dos IMPORT Delay;

FROM IFF8SVXLoad IMPORT Read8SVX, Dealloc8SVX, IFF8SVXInfoPtr,
  PlaySample, OpenAudio, CloseAudio, WaitPlay, AllChannels;

CONST
  ProgName = "PlaySample, 1988 by Fridtjof Siebert [Amok]";

VAR
  Name: ARRAY[0..79] OF CHAR; (* the Sound's Name *)
  length: INTEGER; (* dummy for receiving Name's Length *)
  Info: IFF8SVXInfoPtr; (* Sound's Data *)

BEGIN

(*----- Get Name: -----*)

IF NumArgs() # 0 THEN
  GetArg(1, Name, length);
ELSE
  Terminate(0);
END;

(*----- Read and Play Sound: -----*)

Info := Read8SVX(Name, 16000, FALSE); (* load sound *)

IF Info # NIL THEN

```



```

IF OpenAudio(1,0) THEN          (* open audio *)

  IF PlaySample(Info,0,1,0) THEN (* play sound *)

    WaitPlay(0);

  END;

  Delay(100);      (* wait 2s to get last samples before
                    closing audio *)
  CloseAudio();    (* close audio *)

END;

Dealloc8SVX(Info);      (* free sound *)

END;

END PlaySample.

```

Listing 3.
»PlaySample« spielt einen Sound einmal durch ■

```

(*-----*)
:Program.   PlayRep.mod
:Author.    Fridtjof Siebert
:Address.   Nobileweg 67, D-7-Stgt-40
:Shortcut.  [fbs]
:Version.   1.0
:Date.      19-Sep-88
:Copyright. PD
:Language.  Modula-II
:Translator. M2Amiga
:Imports.   IFF8SVXLoad [fbs], MemSystem [bne]
:Update.    none.
:Contents.  Program to play a sampled Sound repeatedly.
:Remark.    Usage: PlayRep <iff-soundfile>
(*-----*)

MODULE PlayRep;

FROM SYSTEM IMPORT ADR, ADDRESS, SHIFT, BITSET, LONGSET, CAST;
FROM Arts IMPORT Terminate, Assert;
FROM Arguments IMPORT NumArgs, GetArg;
FROM Exec IMPORT WaitPort;
FROM Intuition IMPORT DisplayBeep, NewWindow, WindowPtr,
WindowFlags, WindowFlagSet, ScreenFlags,
ScreenFlagSet, IDCMPFlags, IDCMPFlagSet,
OpenWindow, CloseWindow;

FROM IFF8SVXLoad IMPORT Read8SVX, Dealloc8SVX, IFF8SVXInfoPtr,
PlaySample,
OpenAudio, CloseAudio, WaitPlay, AllChannels;

CONST
  ProgName = "PlayRep, 1988 by Fridtjof Siebert [Amok]";

VAR
  Name: ARRAY[0..79] OF CHAR; (* the Sound's Name *)
  length: INTEGER; (* dummy for receiving Name's Length *)
  Info: IFF8SVXInfoPtr; (* Sound's Data *)
  Window: WindowPtr;
  NuWindow: NewWindow;
  Channel: CARDINAL; (* Active Channel *)

BEGIN

(*----- Get Name: -----*)

  IF NumArgs() # 0 THEN
    GetArg(1,Name,length);

```

```

ELSE
  Terminate(0);
END;

(*----- Open Window: -----*)

WITH NuWindow DO
  leftEdge := 400;
  topEdge := 40;
  width := 240;
  height := 24;
  detailPen := 0;
  blockPen := 1;
  idcmpFlags := IDCMPFlagSet{closeWindow};
  flags := WindowFlagSet{windowSizing, windowDrag, windowDepth,
    windowClose, activate};

  firstGadget := NIL;
  checkMark := NIL;
  title := ADR("Sound loader AMOK");
  screen := NIL;
  bitMap := NIL;
  minWidth := 30;
  minHeight := 16;
  maxWidth := -1;
  maxHeight := -1;
  type := ScreenFlagSet{wbenchScreen};
END;
Window := OpenWindow(NuWindow);
Assert(Window # NIL,ADR("Shit!"));

(*----- Read and Play Sound: -----*)

Info := Read8SVX(Name,16000,FALSE); (* load sound *)

IF Info # NIL THEN

  IF OpenAudio(1,0) THEN (* open audio *)

    IF PlaySample(Info,Channel,65535,Channel) THEN END;

    WaitPort(Window.userPort);

    CloseAudio(); (* close audio *)

  ELSE
    DisplayBeep(NIL);
  END;

  Dealloc8SVX(Info); (* free sound *)

ELSE
  DisplayBeep(NIL); (* error *)
END;

CloseWindow(Window);

END PlayRep.

```

Listing 4.
Mit »PlayRep« wird der Sound endlos wiederholt ■

```

(*-----*)
:Program.   PlayKeys.mod
:Author.    Fridtjof Siebert
:Address.   Nobileweg 67, D-7-Stgt-40
:Shortcut.  [fbs]
:Version.   1.0
:Date.      19-Sep-88
:Copyright. PD
:Language.  Modula-II
:Translator. M2Amiga
:Imports.   IFF8SVXLoad [fbs], MemSystem [bne]
:Update.    none.
:Contents.  Program to play a sampled Sound with different

```

Listing 5. »PlayKeys« läßt das Sample in verschiedenen Tonhöhen erklingen ►


```

samplerates.
:Remark.      Usage: PlayKeys <iff-soundfile>
-----*)

MODULE PlayKeys;

FROM SYSTEM      IMPORT ADR, ADDRESS, SHIFT, BITSET, LONGSET, CAST;
FROM Arts        IMPORT Terminate, Assert;
FROM Arguments   IMPORT NumArgs, GetArg;
FROM Dos         IMPORT Delay;
FROM Exec        IMPORT GetMsg, WaitPort, ReplyMsg;
FROM Intuition   IMPORT DisplayBeep, NewWindow, WindowPtr,
WindowFlags,
                    WindowFlagSet, ScreenFlags, ScreenFlagSet,
                    IDCMPFlags, IDCMPFlagSet, OpenWindow,
                    CloseWindow,
                    IntuiMessagePtr;

FROM IFF8SVXLoad IMPORT Read8SVX, Dealloc8SVX, IFF8SVXInfoPtr,
PlaySample, OpenAudio, CloseAudio, WaitPlay, AllChannels;

CONST
  ProgName = "PlayKeys, 1988 by Fridtjof Siebert [Amok]";

VAR
  Name: ARRAY[0..79] OF CHAR; (* the Sound's Name *)
  length: INTEGER; (* dummy for receiving Name's Length *)
  Info: IFF8SVXInfoPtr; (* Sound's Data *)
  Window: WindowPtr;
  NuWindow: NewWindow;
  Samples: CARDINAL; (* IFF's samplespersec *)
  s: LONGINT; (* My SamplesPerSec *)
  Msg: IntuiMessagePtr;
  Channel: CARDINAL; (* Active Channel *)

BEGIN

(*----- Get Name: -----*)

  IF NumArgs() # 0 THEN
    GetArg(1, Name, length);
  ELSE
    Terminate(0);
  END;

(*----- Open Window: -----*)

  WITH NuWindow DO
    leftEdge := 80;
    topEdge := 40;
    width := 320;
    height := 80;
    detailPen := 0;
    blockPen := 1;
    idcmpFlags := IDCMPFlagSet{closeWindow, vanillaKey};
    flags := WindowFlagSet{windowSizing, windowDrag, windowDepth,
                          windowClose, activate};

    firstGadget := NIL;
    checkMark := NIL;
    title := ADR("Press some Keys !!!");
    screen := NIL;
    bitMap := NIL;
    minWidth := 30;
    minHeight := 16;
    maxWidth := -1;
    maxHeight := -1;
    type := ScreenFlagSet{wbenchScreen};
  END;
  Window := OpenWindow(NuWindow);
  Assert(Window # NIL, ADR("Shit!"));

(*----- Read and Play Sound: -----*)

  Info := Read8SVX(Name, 16000, FALSE); (* load sound *)

  IF Info # NIL THEN

```

```

    IF OpenAudio(4, 0) THEN (* open audio *)

      Samples := Info.VHDR.samplesPerSec;

      Channel := 0;

      LOOP
        WaitPort(Window.userPort);
        Msg := GetMsg(Window.userPort);
        IF closeWindow IN Msg.class THEN EXIT END;
        s := LONGINT(Samples) * LONGINT(Msg.code) DIV 64;
        ReplyMsg(Msg);
        IF (s < 20000) AND (s > 200) THEN
          Info.VHDR.samplesPerSec := s;
          IF NOT(PlaySample(Info, 0, 1, Channel)) THEN END;
          IF Channel < 3 THEN
            INC(Channel);
          ELSE
            Channel := 0;
          END;
        END;
      END;
      ReplyMsg(Msg);

      CloseAudio(); (* close audio *)

    ELSE
      DisplayBeep(NIL);
    END;

    Dealloc8SVX(Info); (* free sound *)

  ELSE
    DisplayBeep(NIL); (* error *)
  END;

  CloseWindow(Window);

END PlayKeys.

```

Listing 5. (Schluß)

```

(*-----*)
:Program.      PlayKanon.mod
:Author.       Fridtjof Siebert
:Address.      Nobileweg 67, D-7-Stgt-40
:Shortcut.     [fbs]
:Version.      1.0
:Date.         19-Sep-88
:Copyright.    PD
:Language.     Modula-II
:Translator.   M2Amiga
:Imports.      IFF8SVXLoad [fbs], MemSystem [bne]
:UpDate.       none.
:Contents.     Program to play a sampled Sound on 4 times with
               0.5s delay.
:Remark.       Usage: PlayKanon <iff-soundfile>
-----*)

MODULE PlayKanon;

FROM SYSTEM      IMPORT ADR, ADDRESS, SHIFT, BITSET, LONGSET, CAST;
FROM Arts        IMPORT Terminate;
FROM Arguments   IMPORT NumArgs, GetArg;
FROM Dos         IMPORT Delay;

FROM IFF8SVXLoad IMPORT Read8SVX, Dealloc8SVX, IFF8SVXInfoPtr,
PlaySample, OpenAudio, CloseAudio, WaitPlay, AllChannels;

CONST
  ProgName = "PlaySample, 1988 by Fridtjof Siebert [Amok]";

VAR
  Name: ARRAY[0..79] OF CHAR; (* the Sound's Name *)

```



```
length: INTEGER;          (* dummy for receiving Name's Length *)
Info: IFF8SVXInfoPtr;      (* Sound's Data *)
i: CARDINAL;

BEGIN

(*----- Get Name: -----*)

IF NumArgs() # 0 THEN
  GetArg(1,Name,length);
ELSE
  Terminate(0);
END;

(*----- Read and Play Sound: -----*)

Info := Read8SVX(Name,20000,FALSE);      (* load sound *)

IF Info#NIL THEN

  IF OpenAudio(4,0) THEN                  (* open audio *)
```

```
FOR i:=0 TO 3 DO
  IF PlaySample(Info,0,1,i) THEN          (* play sound *)
    Delay(25);
  END;
END;
WaitPlay(AllChannels);

Delay(100);          (* wait 2s to get last samples before
                     closing audio *)
CloseAudio();        (* close audio *)

END;

Dealloc8SVX(Info);   (* free sound *)

END;

END PlayKanon.
```

Listing 6. Das Demo »PlayKanon« aktiviert alle vier Soundkanäle

Für Programmierer: das IFF-Gratik-Format

Der Standard

Nichts hat sich so durchgesetzt wie das IFF-Format auf dem Amiga. Wir verraten Ihnen, was Sie als Programmierer bei der Arbeit mit IFF-Grafikdateien wissen sollten.

Von Fridtjof Siebert
und Martin Jobst

Fast jeder Computer-Anwender kennt das Problem: Programm x kann die Daten von Programm y nicht verarbeiten. Wenn überhaupt, machen erst mühselige Konvertierungen die Verarbeitung möglich. Amiga-User haben es da besser. Bereits seit den Urzeiten des Amiga existiert das von Electronic Arts ins Leben gerufene IFF-Format (Interchange File Format). Es vereinfacht den Austausch von Grafik- genauso wie von Sound- oder Animations-Dateien. Gerade als Programmierer sollten Sie über dessen Aufbau genau Bescheid wissen. Wir geben Ihnen daher im folgenden eine genaue Beschrei-

schreibung der Struktur des IFF-Bildformats.

IFF-Dateien beginnen mit der Kennung »FORM«. Darauf folgt die Länge des Datenblocks. Dieser enthält die Kennung der Daten, welche bei Bildern »ILBM« ist. Dahinter liegen beliebig viele Datenblöcke, welche jeweils mit vier Byte beginnen. Diese kennzeichnen die Daten. Darauf folgt ein Langwort, das wiederum die Länge des Datenblocks angibt. Den genauen Aufbau einer ILBM-Datei entnehmen Sie bitte Tabelle 1. »Byte« ist die Nummer des jeweils beschriebenen Bytes.

Der Block »BMHD« bildet die Basis einer IFF-Grafik und enthält wichtige Daten, wie die Größe des Bildes oder die Anzahl der Bitplanes. »CMAP« legt die Farben fest, während »GRAB« die Möglichkeit bietet, einen bestimmten Punkt (»hot spot«) der Grafik zu markieren.

»DEST« gibt an, in welche Bitplanes die Grafikdaten geschrieben werden. Im Block »CAMG« werden besondere Modi wie Interlace, HAM etc. festgelegt. »CRNG« dient zur Steuerung einer Farbanimation, »BODY« schließlich enthält die eigentlichen Daten der Gra-

fik und ist damit zusammen mit »BMHD« zwingender Bestandteil eines IFF Grafik-Files.

Ein Ladeprogramm für IFF-Bilder sollte Daten, die unbekannte Kennungen haben, einfach überlesen. Die Länge dieser Daten steht jeweils im darauf folgenden Langwort. Einen unbekannten - oder besser: nicht benötigten - Datenblock finden Sie beispielsweise am Ende von Tabelle 1 unter dem Namen »CMHD«. Dieser wird vom Programm »Graphiccraft« verwendet.

Ein gültiges ILBM-File muß die Blöcke »BMHD« und »BODY« enthalten, »CMAP« sollte vorhanden sein. Alle anderen Strukturen sind optional. In Bild 1 sehen Sie ein Beispiel für eine Lores-Grafik mit der Auflösung 320 x 200 (14H: 0140H; 16H: 0C8H) als Hex-Dump. Die Tiefe beträgt 5 Bitplanes (1CH), deshalb besteht sie aus 32 Farben (in 2CH steht 60H, durch 3 also 20H=32). Die Daten in »DPPV« sollten überlesen werden. Die vier »CRNG«-Blöcke sind ebenfalls meist unwichtig. so

```
0000: 464F524D 0000567A 494C424D 424D4844  FORM..VzILBMHMD
0010: 00000014 014000C8 00000000 05020100  .....
0020: 00020A08 014000C8 434D4150 00000060  .....CMAP..
0030: 00000090 10000000 00200020 30003040  .....0.0
0040: 10305010 40602050 70305080 4060A050  .P..PpOP..P
0050: 70806070 C07080D0 9090E0A0 A0F0C0C0  p..p.p.....
0060: F06020D0 5020B040 10903010 60301040  .P..0..0.0.
0070: 20103000 00000080 0000A000 00A00000  .0.....
0080: A00000A0 0000A000 00A00000  A00000A0  .....
0090: 44505056 00000068 00000000 00000000  .....
00A0: 01680000 014000C8 0002005A 00020000  .h...Z....
00B0: 00020000 00020000 00000000 00000000  .....
00C0: 00000000 00000000 00000000 00000000  .....
00D0: 00000000 00000000 00000000 00010002  .....
00E0: 00000000 00000000 00000000 00010002  .....
00F0: 00000000 00000000 00000000 00010002  .....
0100: 43524E47 00000008 19961800 0002020F  CRNG.....
0110: 43524E47 00000008 29304E78 00039D26  CRNG....)ONX.&
0120: 43524E47 00000008 DD41AB4A FFFF7FFF  CRNG....A.J...
0130: 43524E47 00000008 0047FFF4 00000001  CRNG....G....
0140: 424F4459 00005539 11043336 01EFD03F  BODY..U9..36..?
0150: DCC00000 0161000D FFFF40F9 00026130  ....a...a0
0160: 03FE0004 B5C0C890 E1FE0011 FDC337FF  ....7.
```

Bild 1. Beispiel einer IFF-Lores-Grafik als Hex-Dump

Martin Jobst studiert Publizistik und Kommunikationswissenschaft in Salzburg und arbeitet seit drei Jahren als freier Mitarbeiter für Zeitschriften des Markt & Technik Verlages. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Martin Jobst, Hans-Pinsel-Straße 2, 8013 Haar).

Grundstruktur		
Byte	Bezeichnung	Funktion
0	"FORM"	kennzeichnet IFF-Datei
4	LONGCARD	Länge des Datenblocks, entspricht der Filelänge 8, da die Daten beim achten Byte beginnen
8	"ILBM"	kennzeichnet die Grafikdatei (Interleaved BitMap)
12	Unterblock 1	die verschiedenen Daten
x	Unterblock 2	
y	etc.	

Die Unterblöcke		
Byte	Bezeichnung	Funktion
"BMHD", BitMapHeader:		
0	"BMHD"	Kennung des Blocks
4	LONGCARD(20)	Länge der Daten: 20 Bytes
8	CARDINAL(dx)	Breite der Grafik
10	CARDINAL(dy)	Höhe der Grafik
12	INTEGER(x)	x-Position der Grafik (LeftEdge)
14	INTEGER(y)	y-Position der Grafik (TopEdge)
16	UBYTE(Depth)	Anzahl der Bitplanes
17	UBYTE(Masking)	Maske für Grafik, normalerweise nicht vorhanden (0). 1: Maske vorhanden, in diesem Fall enthält "BODY" eine zusätzliche Bitplane mit der Maske, diese muß dann auch geladen werden. 2: Durchsichtige Farbe, es steht in »TransparentColor« eine Farbnummer, die als durchsichtig gelten soll. 3: Eine Maske kann erzeugt werden, indem man eine Art Lasso um das Image wirft. Dazu zieht man eine Linie außen um das Bild und füllt den Innenraum mit »transparentColor« aus. Alle Punkte, die daraufhin die Farbe »transparentColor« verwenden, werden »durchsichtig«.
18	UBYTE	0: Daten nicht gepackt
	(Compression)	1: Daten gepackt, siehe "BODY"
19	UBYTE(0)	nicht benutzt, muß 0 sein
20	CARDINAL(transparentColor)	durchsichtige Farbe, siehe »Masking«
22	UBYTE(xAspect), UBYTE(yAspect)	Verzerrung des Bildes, bei einer Auflösung von 320x200 im Verhältnis 10:11
24	INTEGER(dx)	Breite des Screens
26	INTEGER(dy)	Höhe des Screens
"CMAP", ColorMap:		
0	"CMAP"	Kennung des Blocks
4	LONGCARD(x)	Länge des Blocks, entspricht der Anzahl der Farben geteilt durch drei. Für gewöhnlich ist die Anzahl der Farben eine gerade Zahl, weshalb kein Füllbyte eingefügt werden muß.
8	UBYTE(Red)	Rotanteil Farbe 0
9	UBYTE(Green)	Grünanteil Farbe 0
10	UBYTE(Blue)	Blauanteil Farbe 0
11	UBYTE(Red)	Rotanteil Farbe 1
12	etc.	
x+7	UByte(Blue)	Blauanteil Farbe (x DIV 3)

"GRAB", hot spot:		
0	"GRAB"	Kennung
4	LONGCARD(4)	Länge 4 Byte
8	INTEGER(x)	x-Position des markierten Punktes
10	INTEGER(y)	y-Position
"DEST", destination:		
0	"DEST"	Kennung
4	LONGCARD(8)	Länge
8	UBYTE(Depth)	Anzahl der Planes in Quell-Grafik
9	UBYTE	Füllbyte: 0
10	CARDINAL (PlanePick)	siehe Intuition Ref. Manual, S.192 ff
12	CARDINAL (PlaneOnOff)	
14	CARDINAL (PlaneMaks)	zum Schreiben benutzte Bitplanes
"CAMG", besondere ViewModes:		
0	"CAMG"	
4	LONGCARD(4)	
8	LONGSET (ViewMode)	entspricht nicht »ViewModeSet{ }« {2}: Interlace {10}: DoublePlayField {17}: Hold and Modify {31}: HiRes
"CRNG", Colorcycling:		
0	"CRNG"	
4	LONGCARD(8)	
8	INTEGER	nicht benutzt:0
10	INTEGER(rate)	Geschwindigkeit: 60mal/s=8000H 30mal/s=4000H 1mal/ec= 8000H
12	INTEGER	eingeschaltet: ungleich 0
14	UBYTE(low)	unterste und
15	UBYTE(high)	oberste Grenze der Farbe
"BODY", enthält die eigentlichen Bilddaten:		
0	"BODY"	
4	LONGCARD(x)	Länge (Höhe•Breite•Tiefe/8)
8	Daten	wenn in "BMHD" »Compressed« gewählt wurde, in gepacktem Format. Ungepackte Bitmap-Daten; Daten sind der Reihe nach gespeichert: 1. Zeile, 1. Bitplane; 1. Zeile 2. Bitplane etc. Danach: 2. Zeile 1. Bitplane, 2. Zeile 2. Bitplane etc. Gepackte Bitmap-Daten: gepackt wird nur innerhalb einer Zeile. Die Zeilen sind der Reihe nach (analog wie bei den ungepackten Daten) gespeichert. Gepackt wird byteweise: Liegt ein Byte im Bereich von 0 bis 127, bedeutet das: die nächsten n+1 Bytes unverändert in die Grafik laden. 129 bis 255 heißt: das folgende Byte wird 257-n Mal hintereinander in die Grafik kopiert. 128: keine Funktion.
Weitere Datenblöcke:		
"SPRT"	für Sprites	
"CCRT"	Graphiccraft: Colorcycling	
"CMHD"	Graphiccraft (?)	
"DPPV"	Graphiccraft (?)	

Tabelle 1. Die Struktur eines IFF-Grafik-Files mit den einzelnen Unterblöcken

Gängige Spielearten für den Amiga

Spiele und kein Ende

Computer sind universelle Werkzeuge. Erledigen sie für den Anwender vor allem nützliche Aufgaben, sind sie auch für andere Bereiche wie geschaffen: Spiele. Gerade der Amiga – ursprünglich als reine Spielmaschine konzipiert – ist ideal für diesen Zweck. Wir haben versucht, die gängigen Spiele grob nach Genres einzuteilen, um Ihnen einigermaßen Überblick zu verschaffen. Repräsentative Vertreter werden dabei genauso genannt, wie Reiz und wesentliche Elemente der verschiedenen Gattungen.

Adventures

Die Abenteuerlust im Computer. Sie sind der Held – oder Antiheld – der sich durch eine Welt jenseits von Gut und Böse schlagen muß. Sie kämpfen mit realen oder irrationalen Problemen. Manchmal brennt der Hut, ein anderes Mal ist High-Life angesagt. Fast wie im richtigen Leben.

Abenteuerspiele faszinieren durch die große Abwechslung, die sie dem Spieler bieten. Wie Sie das Abenteuer überstehen, hängt von Ihrer Kreativität, Ihrem Ehrgeiz und Forschergeist ab. Mittels eines sogenannten Parsers teilen Sie dem Computer Ihre Entscheidungen mit. Wie gut dieser Parser Sie versteht, hängt stark davon ab, ob Sie ein Text- oder ein Grafik-Adventure spielen.

Text-Adventures arbeiten rein auf Textbasis und mit Ihrem Vorstellungsvermögen. Sie bekommen die Informationen über Ihre Umgebung nur verbal mitgeteilt, erhalten detaillierte Schilderungen der Welt im Computer. Genauso detailliert können Sie dem Computer Ihre Vorstellungen und Anweisungen mitteilen. Text-Adventures besitzen sehr gute Parser und verstehen – zwar meist in Englisch – auch komplexe Sätze. Ungeschlagen in diesem Genre sind immer noch die Adventures von Infocom. »Hitchhikers Guide to the Galaxy« etwa fasziniert

Der Spieltrieb des Menschen ist unersättlich. Selbst vor Computern macht er nicht halt. Was haben diese Maschinen zu bieten?

Von Martin Jobst

niert durch Detailreichtum und seine spritzige Art.

Grafik-Adventures haben optisch wesentlich mehr zu bieten. Der Amiga als prädestinierte Grafikmaschine ist für diese Art Spiele besonders geeignet. »Fish« oder »The Pawn« von Magnetic Scrolls sind gute Beispiele für Grafik-Adventures auf dem Amiga. Mit einer Unmenge an schönen Bildern verwöhnen sie den Spieler und lassen ihn mit der Computerwelt eins werden. Der Nachteil bei dieser Art Abenteuer liegt meist am relativ schlechten Parser. Durch den großen Speicherbedarf der Grafiken, bleibt für die Schnittstelle Mensch-Computer oft wenig Platz.

Rollenspiele

Stark verwandt mit den Adventures ist die nächste Gattung: die Rollenspiele. Sie befinden sich auch hier in einer in Silikon geformten Welt und kämpfen ums Überleben. Im Unterschied zu den Abenteuerspielen kommt es aber hier stärker auf Ihre Person an, auf die Person, die Sie im Computer verkörpern. Wie der Name schon sagt, nehmen Sie bestimmte Rollen ein: als Zauberer, Gnom oder was auch immer. Jedenfalls ist Gedeih oder Verderb stark von den Eigenschaften abhängig, die Sie besitzen oder sich im Laufe des Spieles – wie auch immer – aneignen. Stark abhängig sind Sie auch von Ihren Komparten samt deren Eigenschaften. Nur wenn Sie diese geschickt auswählen, haben Sie auch Chancen, zu überleben.

Rollenspiele arbeiten wie Grafik-Adventures stark mit grafischen Elementen. Wieder

können wir zwei Gattungen unterscheiden. Während Rollenspiele wie »Bard's Tale« stark mit symbolischen Mitteln arbeiten, haben Sie Kämpfe und andere Aktionen bei »Dungeon Master« in »Realtime« auszutragen. Geben Sie acht, daß Sie Ihre Rolle nicht ins richtige Leben mitnehmen: »Nein, glauben Sie mir, Sie sind kein Zauberer!«

Strategiespiele

Taktik ist gefragt. Auch wenn Sie Strategiespiele auf dem Computer spielen, kommen Sie um das Kopfzerbrechen nicht herum. Der Sinn, diese Art von Spielen auf Rechnern zu spielen, ist der fehlende Partner, den diese Ihnen ersetzen können. Nicht nur das, starke Programme stellen Gegner, die Sie – was die Spielstärke betrifft – sonst nicht so schnell finden. Sie können auf dem Amiga jede Menge spielen: ob Dame, Schach, Knobelspiele oder chinesische Brettspiele – alles ist möglich.

Besonders interessant sind natürlich Schachprogramme. Wenn Sie das spielstärkste suchen, ist Ihnen mit »Colossus Chess X« gedient. Das Programm ist erst seit kurzem auf dem Markt, schlägt aber seine Konkurrenten bei weitem. Ganz nett, was die Optik betrifft, ist auch »Battle Chess«. Der Reiz dieser Schachvariante liegt darin, daß alle Züge als Kämpfe ausgetragen werden, die prächtig animiert auf dem Bildschirm dargestellt werden. Was dieses Programm allerdings nicht kann, ist Schachspielen. Wenn Sie einen kompetenten Gegner suchen, ist Ihnen mit diesem Programm keinesfalls geholfen.

Ein Brettspiel, das Sie vor den Monitor fesseln wird, ist »Shanghai«. Bei der Variante eines chinesischen Brettspiels, geht es darum, von Türmen aus Spielsteinen nach bestimmten Regeln immer zwei gleiche wegzunehmen. Hört sich einfach an? Ist es aber ganz bestimmt nicht. Der Computer ist bei diesem Spiel nicht Ihr unmittelbarer Gegner, er ersetzt nur das Spielbrett und legt die Spielsteine auf.

Sportspiele

Tor! Tor! Tor! So klingt es nicht nur zuzeiten von Fußballmeisterschaften aus Ihrem Fernsehlautsprecher. Auch Ihr Amiga kann da mithalten. Fußballsimulationen wie »Kick off« holen den Ball auf den Bildschirm. Aber warum Sport auf dem Computer? Das kann viele Gründe haben. Aber ob nun aus Faulheit, Bequemlichkeit oder schlicht schlechtem Wetter vor der Tür, Sport ist auch auf dem Computer sehr reizvoll. Gerade dann, wenn Sie das Vergnügen mit einem Partner teilen und gegeneinander zur Eishockey-Weltmeisterschaft antreten. Und von wegen unsportlich: passen Sie auf, daß Sie sich beim Body-Check auf dem Schirm Ihre Finger nicht verrenken.

Auch bei diesem Genre greifen die tollen Grafikfähigkeiten unseres Amiga. Sportspiele stehen und fallen mit der Realitätsnähe der grafischen Darstellung. Daher kann man Spiele wie »Kick off« fast schon als Simulationen bezeichnen. Auch futuristische Ballspiele wie »Ballistix« oder »Speed-Ball« üben großen Reiz aus. Wenn Sie's amerikanisch wollen: »California Games« von Epyx, zeigt die sportliche Realität an den Stränden Kaliforniens.

Rennspiele

Lange Zeit schauten Amiga-User neidisch auf Spielautomaten oder Besitzer von Videospiel-Konsolen. Die Mißgunst

richtete sich auf die tollen Renn-Spiele, die es für diese Geräte gab. Daß diese Zeiten endgültig vorbei sind, bestätigen eine Reihe von Neuererscheinungen für den Amiga.

Wettrennen im Computer, mit schneller Grafik und noch schnelleren Boliden – reizvoll für viele. Wettrennen auf unseren Straßen mit schnellen Autos und noch größerem Risiko – tödlich für viele. Dann doch lieber der Crash im Computer.

Geben Sie Ihrem Drang nach schnellen Fahrzeugen nach und erleben Sie einen Geschwindigkeitsrausch – vor Ihrem Computer. Rasen Sie bei »Super Hang On« mit einem schnellen Motorrad über Grand-Prix-Strecken dieser Welt und riskieren dabei nicht Ihr Leben. Machen Sie mit »Crazy Cars II« amerikanische High-Ways unsicher und finden Sie den Weg gen Süden. Brausen Sie in »Stunt Car Racer« auf achterbahn-mäßigen Strecken und schlagen Sie Ihren Gegner. Ergeben Sie sich dem Gefühl von Freiheit und Schnelligkeit – hautnah und rasend schnell, aber dennoch risikolos und preiswert: der Ferrari, den Sie mit »Crazy Cars II« erwerben, kostet Sie 0,01 Prozent des realen Vorbilds...

Simulationen

Die Realität in ihre Kiste zu verfrachten – kein unbekannter Traum für viele Computer-Freaks. Der Wunsch nach Simulation von realen oder fiktiven Vorgängen ist präsent. Klassisch und zugleich aktuell ist die Umsetzung eines alten Menschheitstraums, des Traums vom Fliegen. Bereits zu ihren Urzeiten erhoben sich Computer in die Lüfte. Unter mehr oder minder realen Bedingungen steuert man ein Flugzeug durch die Wolken.

Klassisch und zugleich etwas veraltet ist der »Flight II«. Unter lebensnahen Umständen fliegen Sie mit einem Sportflugzeug der Sonne entgegen. So realistisch die Bedingungen auch sein mögen, die langsame Grafik läßt verwöhnte Amiga-Fans schnell ermüden. Flotter geht's da schon bei »Interceptor« zu: rasende Flüge mit einem amerikanischen Kampfflugzeug. Abgedroschener dafür leider das Ziel des Spieles: die Jagd nach russischen Flugzeugen vom Typ MIG 13.

Wesentlich anspruchsvoller geht es dagegen bei »Ports of Call« zu. Sie sind Reeder, und

müssen sich vor geldgierigen Konkurrenten hüten. Halten Sie Ihre Flotte dicht und lassen Sie sich nicht beim Schmuggeln erwischen. Eine Wirtschaftssimulation, die es in sich hat. Kennen Sie schon die Preise für die neuen Schiffe mit 10 000 Bruttoregistertonnen?

Breakout-Spiele

Fang den Ball! Alles was Ihnen dazu zur Verfügung steht, ist ein kleiner Schläger. Ein ein-

faches, aber fesselndes Spieleprinzip. Entstanden aus dem Ur-Klassiker der Computerspiele bieten Breakout-Varianten auf dem Amiga dem Spieler viel Abwechslung.

Die verschiedenen Gimmicks wie Klebstoff-Schläger, Laser, mehr Bälle und die bestechende Grafik schalten Distanz zum Klassiker – und Abwechslung. Die Spitzenreiter sind »Arkanoid II« und »Crystal Hammer«.

Laufen, hüpfen, sammeln.

Jump'n Run-Spiele

Das ist in gebotener Kürze das, was bei diesen Spielen angesagt ist. Durchstöbern Sie Level für Level nach wichtigen Gegenständen und treten Sie nicht in diese oder jene Falle.

Waren auf einem C64 Affen wie »Donkey Kong« die Stars, geht's auf dem Amiga schon wesentlich »gewichtiger« zu. »Hard'n Heavy« ist wohl aussa-

Vermarkungsstrategien für Spiele

Sie haben ein Spiel programmiert und wollen es an den Mann/die Frau bringen. Wir zeigen Ihnen den ersten Schritt auf dem Weg zum Erfolg.

Von Martin Jobst

Ein Spiel sauber zu programmieren ist eine Sache, es erfolgreich zu vermarkten eine andere. Sie investieren viel Zeit, Arbeit und nicht zuletzt Nerven in Ihr Projekt. Wie hoch das letztlich honoriert wird, hängt stark davon ab, welchen Weg Sie zur Vermarktung Ihres Spiels beschreiten. Sie sollten daher alles daran setzen, das Beste aus Ihrer Arbeit zu machen. Dazu müssen Sie nur Möglichkeiten finden, Ihr Produkt – im buchstäblichen Sinne – richtig zu verkaufen. Viele Wege führen dahin und sicherlich gibt es keinen, der immer gültig wäre. Wir wollen Ihnen einige Tips geben und einen möglichen Weg zeigen, wie Ihr Produkt den Weg auf die Ladentische der Software-Händler findet.

Wenn Sie Empfehlungen suchen, wie man ein Spiele-Projekt von der Programmentwicklung her am besten durchzieht, lesen Sie unseren Beitrag dazu ab Seite 6.

»Viele Köche verderben den Brei« lautet ein altes Sprichwort. Daß es sich bei der Programmentwicklung nicht so verhält, ist längst bewiesen: die meisten guten Programme entstehen aus der Zusammenarbeit von Teams. Daß Ihnen als

»kleiner« Programmierer keine Crew aus Grafikern, Designern und Musikern zur Verfügung steht, ist uns schon klar. Was wir damit sagen wollen ist nur, daß gute Programme meist nicht in einem Kopf entstehen. Auch wenn Sie Ihr Spiel alleine programmieren, sollten Sie es vor der endgültigen Fertigstellung möglichst vielen Leuten zugänglich machen.

Versuchen Sie daher, Vorversionen Ihres Spiels unter die Freaks zu bringen. Zehn Leute finden schneller die Bugs in einem Programm, zehn Leute haben mehr Ideen für Verbesserungen, zehn Leute können mehr aus Ihrem Spiel machen. Tja, aber: wie finde ich die zehn Leute, die sich für mein Programm interessieren, es auf Fehler testen etc.? Nicht nur zehn, hundert, ja tausend Freaks, die Ihr Programm liebend gerne testen und auseinandernehmen, lassen sich jederzeit finden. Wo? Im PD-Pool. Etwas Besseres kann Ihrem neuen Spiel gar nicht passieren, als (in einer Vorversion) über PD verbreitet zu werden. Sie glauben gar nicht, welche Vorteile sich daraus ergeben. Achten Sie dabei aber wirklich darauf, nur eine Demo-Version über PD zu verbreiten, wenn Sie beabsichtigen, das Programm später noch kommerziell zu vermarkten. Bauen Sie also »Sperrern« ein, die das Programm nur bedingt nutzbar machen. Der Zweck soll ja nur sein, die wesentlichen Elemente des Programmes zu demonstrieren.

Ganz abgesehen davon, daß möglichst viele Freaks auf Ihr

Produkt aufmerksam werden und potentielle Kunden für das »Endprodukt« sein können, erhalten Sie von diesen bestimmt auch eine Menge Anregungen. Wo Bugs in Ihrem Programm sind, was verbessert werden sollte und was daran toll ist, erfahren Sie dann als Reaktion von Anwendern oder besser »Spielern«. Daß Sie sich ganz nebenbei in der Szene einen Namen machen können, ist ein angenehmer Nebeneffekt. Viele Beispiele geben Zeugnis dafür. Warum etwa arbeitet ein Leo Schwab plötzlich bei einer Walt Disney-Software-Produktion namens »Roger Rabbit« mit? Richtig geraten, weil die Szene ihn über PD durch seine wunderbaren Grafik-Demos kennt.

Vielleicht wird eine Software-Firma über PD auf Ihr Produkt aufmerksam und interessiert sich für dessen Endversion. Daß Ihre Arbeit finanziell dann besser belohnt werden wird, liegt auf der Hand. Erstens können Sie mit der Vorversion in der Hand womöglich einen besseren Vertrag abschließen. Zweitens erfahren Sie vielleicht durch das Software-Haus Unterstützung bei der Weiterentwicklung Ihres Produktes.

Ein anderer Weg, mit der Vorab-Version Ihres Spieles eine möglichst große Gruppe anzusprechen, wäre der Weg über die Veröffentlichung in einem Computer-Magazin. Auch hier sind Ihnen die Reaktionen vieler User sicher und auch Kontakte zu Produzenten könnten sich so ergeben. Ganz nebenbei könnten Sie sich auch gleich ein paar Mark verdienen,

gekräftigt genug für ein Spiel einer neuen Generation dieses Genres. Kämpfen Sie sich durch 24 mit Gegnern gespickten Levels und hüten Sie sich vor herumschwirrenden Gruselviechern. Sammeln Sie soviel Sterntaler wie möglich und lassen Sie sich nicht von getarnten Schatzkammern verlocken.

Ballerspiele

Ping, Peng, Pong, Knatatata. Ballern und kein Ende: das

Motto unseres letzten Genres. Fast so alt wie der Computer selbst, bieten diese Spiele immer Unglaublicheres. Begann alles mit einfachen Space Invader-Varianten, haben heutige Klasse-Spiele nur noch das Prinzip mit den Ahnen gemeinsam. Sie führen den Spieler in neue Welten und holen grafisch das Letzte aus dem Amiga heraus.

Der Weltraum-Commander fliegt durch bizzare, futuristische Landschaften, ballert mit unglaublichen Laser-Phasern

durch die Gegend und hat gegen Wesen der fünften bis achten Art anzukämpfen. »Katakis«, »Hybris«, »Menace«, »R-Type«: das sind die High-Lights unter den Ballerspielen. Ideale Games, um den Alltagsstreß hinter sich zu lassen und einmal so richtig reinzuhaufen.

Sie meinen, unsere Übersicht wäre nicht vollständig? Sie haben recht. Um alle Spielarten für den Amiga vollständig zu erfassen, wären etliche Kategorien mehr nötig. Wir ha-

ben nur versucht, die größten Bereiche zu erfassen. So gibt es viele Spiele, die aus dem Rahmen fallen und eine eigene Sparte bilden würden. Wir denken dabei etwa an Tetris & Co. Viele Spiele schienen uns aber auch dank ihrer subtilen Handlung nicht erwähnenswert: Denken Sie etwa an Ballereien im Rambo-Stil. Andere Genres beginnen sich erst zu bilden. Vielleicht setzen Sie mit Ihrem nächsten Spiel einen neuen Meilenstein? so

Idee zum Erfolg

die Sie in die Weiterentwicklung Ihres Produktes investieren können, die Sie aber zumindest anspornen werden.

Wie Sie Ihr Spiel auch reifen lassen mögen, früher oder später wird es zur Vertragsanbahnung mit einem Softwarehaus kommen. Bevor es aber so weit ist, sollten Sie als ersten Schritt mit allen in Frage kommenden Partnern Kontakt aufnehmen. Informieren Sie sich über deren Angebote und lassen Sie sich konkrete Honorarvorschläge

geben. Nur so können Sie den Marktwert Ihres Spiels herausfinden und das beste Angebot erkennen.

Seien Sie mit Umsatzbeteiligungen vorsichtig. Ansonsten kann der Deal mit Ihrem Spiel sehr schnell selber zu einem solchen werden – Kategorie Glücksspiel. Besteht das zu erwartende Honorar zum Großteil aus Umsatzerlösen, stehen Sie sehr schnell mit leeren Händen da. Bedenken Sie auch, daß Sie die verkauften Stückzahlen

selbst nie wirklich kontrollieren können. Sie sind dabei zum großen Teil auf die Seriosität des Softwarehauses angewiesen. In untenstehendem Kasten finden Sie noch einige Tipps, wie Sie negative Ergebnisse vermeiden können, Tipps von Leuten, die in diesem Bereich bereits Erfahrung gesammelt haben.

Universallösung für die Vermarktung eines Programms gibt es keine, dazu sind die Voraussetzungen von Fall zu Fall

zu verschieden. Beachten Sie aber unsere Tipps und Hinweise, wird es Ihnen gelingen, Ihr Spiel nach Ihren Vorstellungen vermarkten zu können – Erfolg damit zu haben. so

Martin Jobst studiert Publizistik und Kommunikationswissenschaft in Salzburg und arbeitet seit drei Jahren als freier Mitarbeiter für Zeitschriften des Markt & Technik Verlages. Sie können ihn erreichen über unsere Verlagsanschrift (Markt & Technik Verlag AG, Redaktion Sonderhefte, z. Hd. Martin Jobst, Hans-Pinsel-Straße 2, 8013 Haar).

Profis geben Tips

Grundsätzliches zur Vermarktung:

– Es ist nur dann zu empfehlen, ein Spiel kommerziell (also über ein Softwarehaus) zu vermarkten, wenn die Honorare, die Sie erwarten können, die 10000-DM-Grenze überschreiten. Andernfalls ist es sinnvoller, zu versuchen, das Produkt über eine Zeitschrift (etwa als Programm des Monats) zu verwerfen. Es fällt dann der beträchtliche Aufwand, der Ihnen mit einem professionellen Projekt entsteht, weg. Allerdings können Sie nur mit einem Honorar von höchstens 2000 DM rechnen.

– Überlegen Sie sich, bevor Sie an entsprechende Softwarehäuser herantreten, für welchen Markt Ihr Produkt eigentlich geeignet ist.

– Aus Copyright-Gründen sollten Sie darauf achten, ob nicht bereits ein Produkt auf dem Markt ist, das den gleichen Namen trägt wie Ihr Spiel.

– Bevor Sie einen Vertrag mit Ihrem Partner anbahnen, sollten Sie sich unbedingt von dessen Seriosität überzeugen. Schwarze Schafe gibt es in diesem Markt genug. Oft fehlt diesen – meist Kleinstfirmen – das nötige Kapital und auch das Know-how für den Vertrieb Ihres Produkts. So mancher dieser Verlage versucht nur eine schnelle Mark zu machen – auf Ihre Kosten. Sehr schnell stehen Sie mit leeren Händen da. Im Extremfall verlangt man von Ihnen sogar Zahlungen, für den Verlust, den Ihr Produkt verursacht...

Der Vertrag:

Wenn es soweit ist und der Vertragsabschluß kurz bevorsteht, sollten Sie den Vertrag inhaltlich genau prüfen. Achten Sie auch auf das

klassische »Kleingedruckte«. Überlegen Sie sich eventuelle Konsequenzen bestimmter Klauseln. Im folgenden einige Hinweise zu eventuellen Hürden in Verträgen.

– Eine Klausel, die bestimmt, daß Sie sämtliche Unterlagen und Daten über Ihr Programm Ihrem Partner »bedingungslos« überlassen, sollte der Vertrag auf keinen Fall enthalten. Mit diesem Schritt verkaufen Sie sich nämlich vollends an das Softwarehaus.

– Sollte im Vertrag die Rede von Vorkaufsrecht sein, bedenken Sie, daß Sie dann eventuelle weitere Produkte aus Ihrer Software-Schmiede zuerst immer Ihrem Vertragspartner anbieten müssen. Solche Klauseln sollten Sie, ohne entsprechende vertragliche finanzielle Absicherung dieses Vorverkaufsrechtes zu Ihren Gunsten, nicht akzeptieren.

– Viele Programmierer arbeiten mit einer eigenen Modulbibliothek, das heißt sie verwenden gleiche Programmteile in verschiedenen Programmen. Wenn Ihr Partner von Ihnen verlangt, daß Sie alle im betreffenden Produkt verwendeten Programmteile in anderen Publikationen nicht mehr verwenden dürfen, sollten Sie von einem Vertrag Abstand halten.

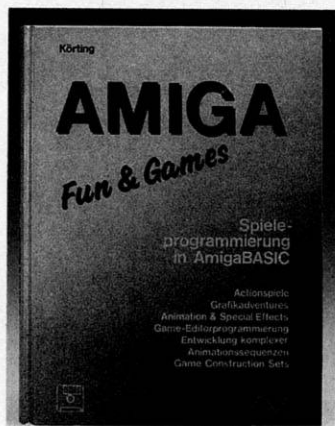
– Es ist grundsätzlich zu empfehlen – vor allem bei Summen unter 30000 DM – für einen festen Honorarsatz zu arbeiten. Sie gehen dann kein unnötiges Risiko ein. Denn erstens ist es sehr schwer, den Umsatz, den Ihr Produkt erzielt, zu kontrollieren. Zudem kann es passieren, daß Ihr Partner versucht, Ihnen Unkosten – etwa für Werbung – anzulasten.

– Haben Sie Ihr Programm fertiggestellt, lassen Sie es von Ihrem Vertragspartner gründlich testen. Lassen Sie sich schriftlich bestätigen, daß das Programm einwandfrei läuft. Nur so können Sie vermeiden, daß eventuell später auftauchende Fehler Ihnen angelastet werden und Sie die Konsequenzen daraus zu tragen haben.

Amiga Fun & Games

Wie der Titel dieses Buches verrät, wird ein spezielles Gebiet der Amiga-Basic-Programmierung behandelt, die Spieleprogrammierung. Dieses Werk setzt Grundkenntnisse in der Basic-Programmierung voraus. Es ist daher nicht für den absoluten Einsteiger gedacht. Um den Umfang in einem akzeptablen Rahmen zu halten, wurde darauf verzichtet die Befehle genauer zu erklären und dafür das eigentliche Thema intensiv behandelt. Wie auch im »Amiga BASIC-Know-How«-Buch dieses Verlages werden insbesondere Lösungswege für bestimmte Probleme aufgezeigt. Das Hauptaugenmerk lag darauf, Lösungswege für die Spieleprogrammierung zu entwickeln, die universell einsetzbar sind.

Das gesamte Buch unterteilt sich in drei Hauptthemengebiete, die Strategie- und Taktikspiele, die Adventures und die Actionspiele. Anhand eines Projektes für jede dieser Spielgattungen wird Schritt für Schritt eine brauchbare Lösung entwickelt. Wobei alle Spiele auf Diskette beiliegen.



In einem 3D-Tic-Tac-Toe-Spiel wird gezeigt, mit welchen Algorithmen dem Computer Strategie und Taktik beigebracht werden. Die Erklärungen dieser Algorithmen sind stets verständlich, nachvollziehbar und lassen sich auf diese Weise auch leicht durch eigene ergänzen.

An einem Wallbreaker-Spiel wird der Aufbau und Nutzen eines Game-Construction-Sets erklärt. Da für jedes Themengebiet ein recht umfangreiches Programm aufgebaut wird, läßt sich die technische Umsetzung der Programmidee gut nachvollziehen. In einigen Unterkapiteln überwiegt jedoch das Listing gegenüber der Beschreibung. Um die Programme leistungsfähig zu machen, wurden die Betriebssystemroutinen einbezogen und ausführlich deren Verwendung erklärt.

Im Kapitel über die Grafik-Adventure-Programmierung wird von der optimalen Planung über das Schreiben eines Drehbuchs bis zur Programmumsetzung alles erklärt. So erfahren Sie, wie man einen leistungsfähigen Parser programmiert. Wie mit Raummodulen ein vielseitiges Spiel entsteht oder wie IFF-Grafiken in Adventure integriert werden. Weitere Anregungen fordern den Leser dazu auf, eigene Erweiterungen zu programmieren.

Im Kapitel über Actionspiele erfahren Sie viel über Libraries und Structures, da nur auf diese Weise Geschwindigkeit erreicht wird. Dies ist auch das Kapitel, in dem alles über Sprites und deren Animation sowie die schnelle Grafikausgabe zu finden ist. Wie auch in den vorherigen Kapiteln wird ein Spiel in den einzelnen Schritten geplant (Spielprinzip, Planung der Grafik, Hauptprogramm, verschiedene Level etc.).

Im sechsten Kapitel werden noch weitere Ausblicke auf die Möglichkeiten in Amiga-Basic und anderen höheren Programmiersprachen gegeben. Das Buch ist gut gegliedert und stellt die wichtigsten Spielgattungen und deren Lösung vor. Jedes Kapitel beginnt mit der Planung und Ausarbeitung der Spielidee. Daraufhin werden in einzelnen Phasen eine Reihe von Unterprogrammen erstellt und zuletzt das koordinierende Hauptprogramm programmiert. Diese Programmstellung dient dem übersichtlichen und professionellen Programmieren. Es wurde besonderer Wert darauf gelegt, dem Leser ein möglichst abstraktes und auf einzelne Prozeduren verteiltes Programmieren nahelegen. Dies ist nicht nur für die Spieleprogrammierung, sondern für jedes andere Projekt sinnvoll. Der in Basic ungeliebte »Spaghetticode« wird, wenn möglich, vermieden.

»Amiga Fun & Games« ist für denjenigen zu empfehlen, der sich intensiver mit der Programmierung von Spielen jeglicher Art beschäftigen will. Man sollte aber schon Erfahrung mit Amiga-Basic haben, da das Wissen über die einzelnen Kommandos und deren Funktionsweise vorausgesetzt wird.

Aus unserem

Ein Programmierer wie ein Fisch ohne nicht auf dem schwimmen, haben für Sie ausgewählt.

Es wird Wert darauf gelegt, die speziellen Probleme der Spieleprogrammierung global zu erklären und danach in einen Programmcode umzusetzen. Daher sind einige Kapitel nicht nur auf Amiga-Basic beschränkt. Der Parser z.B. kann leicht auch in andere Programmiersprachen umgesetzt werden.

Die auf Diskette beiliegenden Programme beweisen, daß auch in Basic gute Spiele geschrieben werden können. Es kommt nur darauf an, wie man die Fähigkeiten des Amiga-Basic nutzt. *Andreas Reguly/pe*

Amiga Fun & Games; Körtling; AV Alpha Verlag
AG; 336 Seiten; Preis 59,90 Mark.

Amiga-BASIC-Know-How

Das Amiga-Basic-Buch von Bonau/Körtling ist ein weiterführendes Buch in der Basic-Programmierung auf dem Amiga. Es wendet sich daher auch nicht an den Einsteiger, sondern ist für denjenigen gedacht, der sich schon mit dieser Programmiersprache beschäftigt hat und den Befehlssatz und deren Funktion kennt. Das Werk zeigt auf, was sich mit Amiga-Basic erreichen läßt und wo die Grenzen des Interpreters liegen.

Im ersten Kapitel befassen sich die Autoren mit der richtigen Planung und Ausarbeitung einer Programmidee. Außerdem wird großer Wert auf guten Programmierstil gelegt. Das zweite Kapitel enthält alles Wissenswerte über Grafik und Animation. Hier werden Balken- und Tortengrafiken, Animation durch Farbrotation, Scrolling, räumliche Darstellungen, Fraktale, Sprites und verschiedene Zeichensätze erklärt. Beispielprogramme, die auch auf der beiliegenden Diskette zu finden sind, verdeutlichen die Funktionen und stellen sinnvolle Hilfsprogramme dar.

Im dritten Kapitel geht es dann an die Hardware. Sie lernen, wie Peripheriegeräte direkt über Hardwareadressierung gesteuert werden und wie Sie von Amiga-Basic aus einen neuen Task starten. Hierbei wird intensiver Gebrauch von den Betriebssystemroutinen gemacht.

Das vierte Kapitel beschäftigt sich mit den Hilfsprogrammen

(Tools und Utilities). Für die professionelle Programmierung erfahren Sie, welche Vorteile Universal-Tools bieten. Weiterhin werden in diesem Kapitel drei größere Tools entwickelt und beschrieben. Im folgenden Kapitel sind einige Besonderheiten der Programmierung enthalten. Neben dem Aufbau eines Passwort-Schutzes werden Sortierverfahren erläutert und ein Beispiel zur recht unbekannten Autoprogrammierung (Programmtext entwickelt sich selbst) gegeben.

Im sechsten Kapitel befinden sich Tips und Tricks, die häufig auftretende Fehler oder Probleme mit dem Editor des Amiga-Basic erklären. Außerdem finden Sie hier Anregungen und Hinweise ähnlich den »Tips und Tricks« im AMIGA-Magazin.

Insgesamt wurde in diesem Buch sehr viel Wissen auf etwa 300 Seiten komprimiert. Die Autoren gehen davon aus, daß die Funktionsweise von Basic-Befehlen dem Leser keine Schwierigkeiten bereitet. Deshalb wurde darauf verzichtet, die reine programmtechnische Lösung zu erläutern. Vielmehr steht die Lösung eines übergeordneten Problems, wie die 3D-Projektion auf dem Bildschirm, die realistische Darstellung mit Hilfe schattierter Drahtgittermodelle oder die Fraktaldarstellung



Bücher-Regal

ohne Bücher ist
Wasser. Damit Sie
Trocknen
wir einige Bücher



lung, im Vordergrund. Im Kapitel zur 3D-Projektion ist ein gutes mathematisches Verständnis von Vorteil. Da das Buch auf relativ wenigen Seiten sehr viel Wissen vermittelt, ist die Schreibweise zum Teil etwas trocken. Im Vergleich zum Amiga-Basic-Buch von Data Becker fehlen auflockernde Einschübe. Das Buch richtet sich jedoch an einen anderen Leserkreis. Es ist für denjenigen empfehlenswert, der sich mit Amiga-Basic bereits ausgiebig beschäftigt hat und mit Hilfe dieses Werkes professionell arbeiten will. Hierzu bietet es viele Anregungen und zeigt in Beispielen, wie die beschriebenen Verbesserungen verwirklicht werden können. Wer diese Funktionen in seine Programme einbaut, kann eine Benutzerfreundlichkeit erreichen, die Programmen in anderen höheren Programmiersprachen in nichts nachsteht.

Andreas Regulpe

Amiga-BASIC-Know-How; Bonau/Körting; AV-Alpha Computer Verlag AG; 304 Seiten; Preis 59,90 Mark.

Amiga-Basic

Das »Amiga-Basic«-Buch von Hannes Rügheimer und Christian Spanik stellt ein umfangreiches Standardwerk für jeden

Amiga-Basic-Programmierer dar. Es ist sowohl für den Einsteiger als auch für Fortgeschrittene geschrieben. Um beiden Ansprüchen gerecht zu werden, schreiben die Autoren zwei Einleitungen, eine, welche dem Anfänger die Grundbegriffe vermittelt, und eine zweite, die dem Fortgeschritten den schnellen Einstieg ermöglicht.

Jeder einzelne Themenbereich wird ausführlich und nachvollziehbar dargestellt. Dabei verfallen die Autoren nicht in eine trockene und anonyme Schreibweise, sondern sie lockern den Text durch kleine Kommentare auf.

Dies ist auch das herausragendste Merkmal des Buches. Im Gegensatz zu anderen Werken versuchen die Autoren mit Hilfe von kleinen Karikaturen, witzigen Einschüben oder eigenen Kommentaren den manchmal doch recht trockenen Stoff auf amüsante Weise zu vermitteln. Hin und wieder schildern sie auch die kleineren und größeren Probleme, die sie während des Schreibens hatten. All dies und auch der recht witzige Dialog der beiden Autoren ganz zu Anfang nimmt dem Buch das Lehrwerkhafte.

Das ganze Werk ist Schritt für Schritt gegliedert, d.h. zunächst werden die Grundlagen geschaffen und danach in aufstei-

gendem Schwierigkeitsgrad die weiteren Themen besprochen. Deshalb ist es am sinnvollsten, das Buch Seite für Seite durcharbeiten, da in den späteren Kapiteln auf das aufgebaute Wissen zurückgegriffen wird.

Im ersten Kapitel wird die Animation von Bobs und Sprites sehr ausführlich erläutert. Danach werden die Grafikfähigkeiten des Amiga (Auflösungsstufen, Screen- und Windowdefinition) erklärt, wobei anschauliche Beispielprogramme das Verstehen erleichtern. Das dritte Kapitel befaßt sich mit der Verwaltung von Daten auf Disketten und kleinen Tools zu diesem Thema. Im Kapitel über Bilderverarbeitung erfahren Sie nebenbei auch alles über die IFF-Austauschformate des Amiga. Auch die herausragenden Soundfähigkeiten werden gebührend behandelt.

Das vierte Hauptkapitel geht noch etwas über das Amiga-Basic hinaus. Hier werden die Unterschiede von Interpretern, Compilern und der Maschinensprache erklärt. Außerdem erhalten Sie eine äußerst ausführliche Anleitung des Basic-Compilers von AC/Basic.

Jedes Kapitel des Buches enthält gut dokumentierte Beispielprogramme, welche, um Ihnen das Abtippen zu ersparen, auf Diskette beiliegen. Zum Teil werden auch größere Pro-

werden erläutert und schließlich sorgt ein Fachwortlexikon für den nötigen Durchblick bei Fremdwörtern aus dem Computerbereich.

Trotz des sehr ausführlichen und verständlich geschriebenen Textes, soll jedoch auf zwei Kritikpunkte hingewiesen werden. In der momentan aktuellsten 5. Auflage des Buches wird nur der AC/Basic-Compiler beschrieben. Es wäre wünschenswert, wenn die Autoren für die nächste Auflage eine kleine Ergänzung vornehmen und auch den recht neuen Hisoft-Basic-Compiler berücksichtigen. Ein kleiner Vergleichstest der beiden Compiler wäre für den Leser eine wünschenswerte Kaufhilfe.

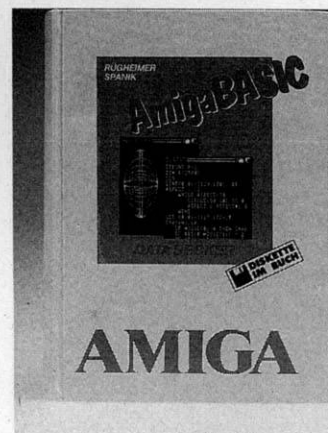
Wichtiger ist jedoch der zweite Kritikpunkt: Da sich das Buch laut Autoren auch an die fortgeschrittenen Basic-Programmierer wendet, sollte auch eine professionellere Programmierweise berücksichtigt werden. Hierzu gehört speziell die Verwendung von Betriebssystemroutinen.

In dem sehr umfangreichen Werk wird mit keinem Wort der Aufruf von Funktionen und Befehlen aus den LIBRARY-Dateien beschrieben. Um jedoch professionelle Programme auch mit Amiga-Basic programmieren zu können, sind diese Bibliotheken unerlässlich. Mit ihrer Hilfe wird es zum Teil erst möglich, bestimmte Programmfunktionen zu verwirklichen.

Dieses Buch ist besonders für den Einsteiger geeignet, der auf diesem Wege erste Erfahrungen im Programmieren sammeln will. Die verständliche und teilweise auch amüsante Schreibweise sorgt dafür, daß auch nach den ersten 200 Seiten der Spaß am Lesen nicht verlorengeht. Aber auch für den etwas fortgeschrittenen Basic-Programmierer finden sich hier nützliche Tips und eine Menge Hintergrundwissen. »Amiga-Basic« von Data-Becker ist daher sowohl für den Anfänger als auch für den Fortgeschrittenen empfehlenswert.

Andreas Regulpe

Amiga Basic; Rügheimer/Spanik; Data Becker Verlag; 792 Seiten; Preis 59,00 Mark.



jekte Schritt für Schritt entwickelt und so die Arbeitsweise eines Programmierers gezeigt.

Im Ausarbeiten des Anhangs haben sich die beiden Autoren große Mühe gegeben. In den einzelnen Kapiteln werden die Amiga-Basic-Fehlermeldungen erklärt und Vorschläge für deren Lösung gegeben, alle Befehle des Interpreters werden mit Parameterangabe und Funktionsweise beschrieben, die Beispielprogramme auf der Commodore-Extras-Diskette

Sollten Sie Interesse an Literatur in den Programmiersprachen C und Assembler haben, bitten wir Sie, in unserem Bücherregal der Ausgabe 4 »C und Assembler« zu stöbern.

Fortsetzung von Seite 27.

```

struct MathBase      *MathBase;
struct MathTransBase *MathTransBase;
struct Screen        *VGscreen;
struct RastPort      *VGrastrport;
struct ViewPort      *VGviewport;
struct VGLine        *VGLinesMem;
struct VGPoint       *VGPointsMem;
char VGErrFile[80]="";
ULONG VGLinesMemSize,VGPointsMemSize,
      VGNumLines=0,VGNumPoints=0,
      VGMaxLines,VGMaxPoints,
      VGxorigin,VGyorigin;
union VGzahl VGscalex,VGscaley;

/* Ausgabe von Fehlermeldungen nach stderr und Errorfile */
BOOL VGerror(Funcname,errname)
char *errname,*Funcname;
{
    FILE *fp;
    time_t tm;
    char timestr[80];
    int length;

    tm=time(NULL);
    length=strlen(ctime(&tm))-5;
    strncpy(timestr,ctime(&tm),length);
    timestr[length]=0;
    fprintf(stderr,"%s %12s: %s ERROR.\n",timestr,Funcname,errname);
    if (strlen(VGErrFile)!=0)
    {
        if (!(fp=fopen(VGErrFile,"a")))
        {
            fp=fopen(VGErrFile,"w");
        }
        if (!fp) { return (FALSE); }
        fprintf(fp,"%s %12s: %s ERROR.\n",timestr,Funcname,errname);
        fclose(fp);
    }
    return (TRUE);
}

/* Setzen der Farbpalette */
BOOL VGcolor(Pos,Anzahl,Colors)
unsigned int Pos,Anzahl;
struct VGColor *Colors;
{
    int i;

    if (!VGscreen)
    {
        VGerror("VGcolor",ERR_FIRSTOPEN);
        return FALSE;
    }
    if ((Pos+Anzahl) > 32)
    {
        VGerror("VGcolor",ERR_FALSESEPAR);
        return FALSE;
    }
    for (i=0;i<Anzahl;i++)
    {
        if ((Colors[i].R > 15) || (Colors[i].G > 15) ||
            (Colors[i].B > 15))
        {
            VGerror("VGcolor",ERR_FALSESERGB);
        }
        SetRGB4(VGviewport,Pos+i,Colors[i].R&15,Colors[i].G&15,
            Colors[i].B&15);
    }
    return TRUE;
}

/* Alle Libraries schließen und Speicher wieder freigeben */
void VGclose(end)
BOOL end;
{
    if (mask & SCREEN) { CloseScreen(VGscreen); }
    if (mask & LINEMEM) { FreeMem(VGLinesMem,VGLinesMemSize); }
    if (mask & POINTMEM) { FreeMem(VGPointsMem,VGPointsMemSize); }
    if (mask & MATHTRANS) { CloseLibrary(MathTransBase); }

```

```

    if (mask & MATHFFFP) { CloseLibrary(MathBase); }
    if (mask & GRAPHICS) { CloseLibrary((struct Library *)GfxBase); }
}

    if (mask & INTUITION) { CloseLibrary((struct Library *)
    IntuitionBase); }
    mask=0;
    if (!end) exit(10);
}

/* Libraries und Screen öffnen, Speicher reservieren, Initialisierung */
void VGopen(AuflsgX,AuflsgY,BitPlanes,MaxLines,MaxPoints,ErrFile)
int AuflsgX,AuflsgY,BitPlanes;
unsigned int MaxLines,MaxPoints;
char *ErrFile;
{
    if (mask)
    {
        VGerror("VGopen",ERR_ALREADYOP);
        VGclose(FALSE);
    }
    strncpy(VGErrFile,ErrFile,79);
    VGErrFile[79]=0;
    VGLinesMemSize=MaxLines * sizeof(struct VGLine);
    VGPointsMemSize=MaxPoints * sizeof(struct VGPoint);
    VGMaxLines=MaxLines;
    VGMaxPoints=MaxPoints;
    switch (AuflsgX)
    {
        case 640: VGnewscreen.Width=640;
                  VGnewscreen.ViewModes=HIRES;
                  break;
        case 320: break;
        default : VGerror("VGopen",ERR_DISPX);
    }
    switch (AuflsgY)
    {
        case 256: VGnewscreen.Height=256;
                  break;
        case 400: VGnewscreen.Height=400;
                  VGnewscreen.ViewModes=LACE;
                  break;
        case 512: VGnewscreen.Height=512;
                  VGnewscreen.ViewModes=LACE;
                  break;
        case 200: break;
        default : VGerror("VGopen",ERR_DISPY);
    }
    if ((BitPlanes>0) && (BitPlanes<=5)){VGnewscreen.Depth=BitPlanes; }
    else { VGerror("VGopen",ERR_DISPB); }
    IntuitionBase=(struct IntuitionBase *)OpenLibrary
        ("intuition.library",0);
    if (!IntuitionBase)
    {
        VGerror("VGopen",ERR_OPINTUITION);
        VGclose(FALSE);
    }
    mask |= INTUITION;
    GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",0);
    if (!GfxBase)
    {
        VGerror("VGopen",ERR_OPGFX);
        VGclose(FALSE);
    }
    mask |= GRAPHICS;
    MathBase=(struct MathBase *)OpenLibrary("mathfp.library",0);
    if (!MathBase)
    {
        VGerror("VGopen",ERR_OPMATHFFFP);
        VGclose(FALSE);
    }
    mask |= MATHFFFP;
    MathTransBase=(struct MathTransBase *)OpenLibrary
        ("mathtrans.library",0);
    if (!MathTransBase)
    {
        VGerror("VGopen",ERR_OPMATHTRANS);
        VGclose(FALSE);
    }
    mask |= MATHTRANS;
    if (VGLinesMemSize)

```



```

{
    VGLinesMem=(struct VGLine *)
        AllocMem (VGLinesMemSize, MEMF_PUBLIC | MEMF_CLEAR);
    if (!VGLinesMem)
    {
        VGerror("VGopen", ERR_OUTMEM);
        VGclose(FALSE);
    }
    mask |= LINEMEM;
}
if (VGPointsMemSize)
{
    VGPointsMem=(struct VGPoint *)
        AllocMem (VGPointsMemSize, MEMF_PUBLIC | MEMF_CLEAR);
    if (!VGPointsMem)
    {
        VGerror("VGopen", ERR_OUTMEM);
        VGclose(FALSE);
    }
    mask |= POINTMEM;
}
if (!(VGLinesMemSize+VGPointsMemSize))
{
    VGerror("VGopen", ERR_NOTHING);
}
VGscreen=(struct Screen *) OpenScreen(&VGnewscreen);
if (!VGscreen)
{
    VGerror("VGopen", ERR_OPSCR);
    VGclose(FALSE);
}
mask |= SCREEN;
VGviewport=&(VGscreen->ViewPort);
VGrastport=&(VGscreen->RastPort);

VGcolor(0,32,VGDefaultcolors);
VGxorigin = VGnewscreen.Width/2;
VGyorigin = VGnewscreen.Height/2;
if ((VGxorigin/2) > VGyorigin)
{
    VGscalex.f=(float)VGnewscreen.Height;
    VGscaley.f=(float)VGyorigin;
}
else
{
    if (VGxorigin > VGyorigin)
    {
        VGscalex.f=(float)VGyorigin;
        VGscaley.f=(float)VGyorigin;
    }
    else
    {
        VGscalex.f=(float)(VGyorigin/2);
        VGscaley.f=(float)VGyorigin;
    }
}

/* Darstellungsroutine */
void VGdisplay(clear)
BOOL clear;
{
    register int i,startx,starty,endx,endy;

    if (clear) { SetRast(VGrastport,0); }
    SetDrMd(VGrastport,JAM1);
    for (i=0; i < VGNumLines; i++)
    {
        SetAPen(VGrastport,VGLinesMem[i].Color);
        startx = VGxorigin+SPFix(SPMul(
            VGPointsMem[VGLinesMem[i].StartPoint].Koord[0].i,
            VGscalex.i));
        starty = VGyorigin+SPFix(SPMul(
            VGPointsMem[VGLinesMem[i].StartPoint].Koord[1].i,
            VGscaley.i));
        endx = VGxorigin+SPFix(SPMul(
            VGPointsMem[VGLinesMem[i].EndPoint].Koord[0].i,
            VGscalex.i));
        endy = VGyorigin+SPFix(SPMul(
            VGPointsMem[VGLinesMem[i].EndPoint].Koord[1].i,

```

```

            VGscaley.i));
        if(startx>=VGnewscreen.Width){startx=VGnewscreen.Width-1;}
        if(endx >=VGnewscreen.Width){endx =VGnewscreen.Width-1;}
        if(starty>=VGnewscreen.Height){starty=VGnewscreen.Height-1;}
        if(endy >=VGnewscreen.Height){endy =VGnewscreen.Height-1;}
        if (startx < 0) { startx=0; }
        if (endx < 0) { endx =0; }
        if (starty < 0) { starty=0; }
        if (endy < 0) { endy =0; }
        Move(VGrastport,startx,starty);
        Draw(VGrastport,endx,endy);
    }
}

/* Linie definieren und übernehmen */
void VGLine(Num,Anfang,Ende,Color)
int Num;
short Color;
int Anfang,Ende;
{
    if ((Anfang>=VGNumPoints) || (Ende>=VGNumPoints))
    {
        VGerror ("VGLine",ERR_UNDEFPPOINT);
    }
    else
    {
        if (VGNumLines >= VGMaxLines)
        {
            VGerror ("VGLine",ERR_TOOMLINES);
        }
        else
        {
            VGLinesMem[VGNumLines].Number = Num;
            VGLinesMem[VGNumLines].StartPoint = Anfang;
            VGLinesMem[VGNumLines].EndPoint = Ende;
            VGLinesMem[VGNumLines++].Color = Color;
        }
    }
}

/* Punkt definieren und übernehmen */
void VGpoint(Obj,Koord)
int Obj;
union VGzahl *Koord;
{
    if (VGNumPoints >= VGMaxPoints)
    {
        VGerror ("VGpoint",ERR_TOOMPOINTS);
    }
    else
    {
        VGPointsMem[VGNumPoints].Object=Obj;
        VGPointsMem[VGNumPoints].Koord[0]=Koord[0];
        VGPointsMem[VGNumPoints].Koord[1]=Koord[1];
        VGPointsMem[VGNumPoints].Koord[2]=Koord[2];
        VGPointsMem[VGNumPoints++].Koord[3].f=FPONE;
    }
}

/* Zuvor definiertes Objekt löschen */
void VGeraseobj(PointObj)
int PointObj;
{
    register int i=0,j,a,b;

    if (PointObj!=0)
    {
        while (i<VGNumPoints)
        {
            if (PointObj==VGPointsMem[i].Object)
            {
                for (j=i+1;j<VGNumPoints;j++)
                {
                    VGPointsMem[j-1].Object=VGPointsMem[j].Object;

```

Listing 1. (Fortsetzung)


```

VGPointsMem[j-1].Koord[0]=VGPointsMem[j].Koord[0];
VGPointsMem[j-1].Koord[1]=VGPointsMem[j].Koord[1];
VGPointsMem[j-1].Koord[2]=VGPointsMem[j].Koord[2];
VGPointsMem[j-1].Koord[3]=VGPointsMem[j].Koord[3];
}
VGNumPoints--;

for (a=0;a<VGNumLines;a++)
{
    if ((VGLinesMem[a].StartPoint==1)|| (VGLinesMem
        [a].EndPoint==1))
    {
        for (b=a+1;b<VGNumLines;b++)
        {
            VGLinesMem[b-1].Number=VGLinesMem
                [b].Number;
            VGLinesMem[b-1].StartPoint=VGLinesMem
                [b].StartPoint;
            VGLinesMem[b-1].EndPoint=VGLinesMem
                [b].EndPoint;
            VGLinesMem[b-1].Color=VGLinesMem
                [b].Color;
        }
        VGNumLines--;
    }
    if (VGLinesMem[a].StartPoint>1)
    {
        VGLinesMem[a].StartPoint-= 1;
    }
    if (VGLinesMem[a].EndPoint>1)
    {
        VGLinesMem[a].EndPoint-= 1;
    }
    else { i++; }
}
else
{
    VGNumPoints=0;
    VGNumLines=0;
}

/* Einzelne Linie löschen */
void VGeraseline(LineNum)
int LineNum;
{
    register int i,j;

    if (LineNum!=0)
    {
        i=0;
        while (i<VGNumLines)
        {
            if (LineNum==VGLinesMem[i].Number)
            {
                for (j=i+1;j<VGNumLines;j++)
                {
                    VGLinesMem[j-1].Number=VGLinesMem[j].Number;
                    VGLinesMem[j-1].StartPoint=VGLinesMem
                        [j].StartPoint;
                    VGLinesMem[j-1].EndPoint=VGLinesMem[j].EndPoint;
                    VGLinesMem[j-1].Color=VGLinesMem[j].Color;
                }
                VGNumLines--;
            }
            else { i++; }
        }
    }
    else { VGNumLines=0; }
}

/* Datenfile einlesen */
BOOL VGreaddata(FileName)
char *FileName;
{
    int i,PointNum,LineNum;
    struct VGPoint Point;

```

```

struct VGLine Line;
FILE *fp;

if (!(fp=fopen(FileName,"r")))
{
    VGError ("VGreaddata",ERR_CANTOPENFILE);
    return (FALSE);
}
fscanf(fp,"%d",&PointNum);
fscanf(fp,"%d",&LineNum);
if ((PointNum > VGMaxPoints) || (LineNum > VGMaxLines))
{
    VGError ("VGreaddata",ERR_LOWRESMEM);
    fclose(fp);
    return (FALSE);
}
for (i=0;i<PointNum;i++)
{
    fscanf(fp,"%d%d%d%d",&(Point.Object),&(Point.Koord[0]),
        &(Point.Koord[1]),&(Point.Koord[2]));
    VGpoint(Point.Object,Point.Koord);
}
for (i=0;i<LineNum;i++)
{
    fscanf(fp,"%d%d%d",&(Line.StartPoint),&(Line.EndPoint),
        &(Line.Color));
    VGLine(i+1,Line.StartPoint,Line.EndPoint,Line.Color);
}
fclose(fp);
return (TRUE);
}

/* Matrix auf Objekt(e) anwenden */
void VGtransform(Obj,Matrix)
union VGzahl Matrix[][VGDIM];
int Obj;
{
    register int a;
    union VGzahl MH[VGDIM];

    for (a=0;a<VGNumPoints;a++)
    {
        if ((!Obj)|| (VGPointsMem[a].Object==Obj))
        {
            MH[0].f=FPZERO;
            MH[0].i=SPAdd(MH[0].i,SPMul(Matrix[0][0].i,VGPointsMem
                [a].Koord[0].i));
            MH[0].i=SPAdd(MH[0].i,SPMul(Matrix[0][1].i,VGPointsMem
                [a].Koord[1].i));
            MH[0].i=SPAdd(MH[0].i,SPMul(Matrix[0][2].i,VGPointsMem
                [a].Koord[2].i));
            MH[0].i=SPAdd(MH[0].i,SPMul(Matrix[0][3].i,VGPointsMem
                [a].Koord[3].i));
            MH[1].f=FPZERO;
            MH[1].i=SPAdd(MH[1].i,SPMul(Matrix[1][0].i,VGPointsMem
                [a].Koord[0].i));
            MH[1].i=SPAdd(MH[1].i,SPMul(Matrix[1][1].i,VGPointsMem
                [a].Koord[1].i));
            MH[1].i=SPAdd(MH[1].i,SPMul(Matrix[1][2].i,VGPointsMem
                [a].Koord[2].i));
            MH[1].i=SPAdd(MH[1].i,SPMul(Matrix[1][3].i,VGPointsMem
                [a].Koord[3].i));
            MH[2].f=FPZERO;
            MH[2].i=SPAdd(MH[2].i,SPMul(Matrix[2][0].i,VGPointsMem
                [a].Koord[0].i));
            MH[2].i=SPAdd(MH[2].i,SPMul(Matrix[2][1].i,VGPointsMem
                [a].Koord[1].i));
            MH[2].i=SPAdd(MH[2].i,SPMul(Matrix[2][2].i,VGPointsMem
                [a].Koord[2].i));
            MH[2].i=SPAdd(MH[2].i,SPMul(Matrix[2][3].i,VGPointsMem
                [a].Koord[3].i));
            MH[3].f=FPZERO;
            MH[3].i=SPAdd(MH[3].i,SPMul(Matrix[3][0].i,VGPointsMem
                [a].Koord[0].i));
            MH[3].i=SPAdd(MH[3].i,SPMul(Matrix[3][1].i,VGPointsMem
                [a].Koord[1].i));
            MH[3].i=SPAdd(MH[3].i,SPMul(Matrix[3][2].i,VGPointsMem
                [a].Koord[2].i));
            MH[3].i=SPAdd(MH[3].i,SPMul(Matrix[3][3].i,VGPointsMem
                [a].Koord[3].i));

```



```

    VGPointsMem[a].Koord[0]=MH[0];
    VGPointsMem[a].Koord[1]=MH[1];
    VGPointsMem[a].Koord[2]=MH[2];
    VGPointsMem[a].Koord[3]=MH[3];
}
}

/* Umrechnung von Grad in Bogenmaß */
float VGdeg_rad(VGdegree)
int VGdegree;
{
    union VGzahl VGangle;

    VGangle.i = SPMul(SPDIV(SPFLT(180),VGPI),((float)VGdegree));
    return(VGangle.f);
}

/* Matrix kopieren */
void VGcopylongmatrix(MatrixA,MatrixB)
union VGzahl *MatrixA,*MatrixB;
{
    register int i;

    for (i=0;i<(VGDIM*VGDIM);i++) { *(MatrixB+i) = *(MatrixA+i); }
}

/* Matrizenmultiplikation */
void VGMullongmatrix(MatrixA,MatrixB,MatrixE)
union VGzahl MatrixA[][VGDIM],MatrixB[][VGDIM],*MatrixE;
{
    union VGzahl MatrixC[VGDIM][VGDIM];
    register int i,j;

    for (i=0;i<VGDIM;i++)
        for (j=0;j<VGDIM;j++)
        {
            MatrixC[i][j].i =
                SPAdd(SPMul(MatrixB[i][0].i,MatrixA[0][j].i),
                    SPAdd(SPMul(MatrixB[i][1].i,MatrixA[1][j].i),
                        SPAdd(SPMul(MatrixB[i][2].i,MatrixA[2][j].i),
                            SPMul(MatrixB[i][3].i,MatrixA[3][j].i) ));
        }
    VGCopylongmatrix(MatrixC,MatrixE);
}

/* Verschiebungsmatrix erzeugen */
void VGtranslate(RMatrix,E1,E2,E3,length,clear)
union VGzahl RMatrix[][VGDIM];
float E1,E2,E3,length;
BOOL clear;
{
    union VGzahl TMatrix[VGDIM][VGDIM];

    E1 *= length;
    E2 *= length;
    E3 *= length;
    TMatrix[0][0].f=FPONE;
    TMatrix[0][1].f=FPZERO;
    TMatrix[0][2].f=FPZERO;
    TMatrix[0][3].f=E1;
    TMatrix[1][0].f=FPZERO;
    TMatrix[1][1].f=FPONE;
    TMatrix[1][2].f=FPZERO;
    TMatrix[1][3].f=E2;
    TMatrix[2][0].f=FPZERO;
    TMatrix[2][1].f=FPZERO;
    TMatrix[2][2].f=FPONE;
    TMatrix[2][3].f=E3;
    TMatrix[3][0].f=FPZERO;
    TMatrix[3][1].f=FPZERO;
    TMatrix[3][2].f=FPZERO;
    TMatrix[3][3].f=FPONE;
    if (clear) { VGCopylongmatrix(TMatrix,RMatrix); }
    else { VGMullongmatrix(TMatrix,RMatrix,RMatrix); }
}

/* Vergrößerungs- oder Verkleinerungsmatrix erzeugen */

```

```

void VGglobalscale(RMatrix,scalefactor,clear)
union VGzahl RMatrix[][VGDIM];
float scalefactor;
BOOL clear;
{
    union VGzahl SMatrix[VGDIM][VGDIM];

    SMatrix[0][0].f=scalefactor;
    SMatrix[0][1].f=FPZERO;
    SMatrix[0][2].f=FPZERO;
    SMatrix[0][3].f=FPZERO;
    SMatrix[1][0].f=FPZERO;
    SMatrix[1][1].f=scalefactor;
    SMatrix[1][2].f=FPZERO;
    SMatrix[1][3].f=FPZERO;
    SMatrix[2][0].f=FPZERO;
    SMatrix[2][1].f=FPZERO;
    SMatrix[2][2].f=scalefactor;
    SMatrix[2][3].f=FPZERO;
    SMatrix[3][0].f=FPZERO;
    SMatrix[3][1].f=FPZERO;
    SMatrix[3][2].f=FPZERO;
    SMatrix[3][3].f=FPONE;
    if (clear) { VGCopylongmatrix(SMatrix,RMatrix); }
    else { VGMullongmatrix(SMatrix,RMatrix,RMatrix); }
}

/* Gerichtete Vergrößerungs- oder Verkleinerungsmatrix erzeugen */
void VGlocalscale(RMatrix,E,scalefactor,clear)
union VGzahl RMatrix[][VGDIM];
float E[],scalefactor;
BOOL clear;
{
    union VGzahl SMatrix[VGDIM][VGDIM];

    SMatrix[0][0].i=SPAdd(SPMul(SPMul(SPSub(scalefactor,FPONE),E[0]),
        E[0]),FPONE);
    SMatrix[0][1].i=SPMul(SPMul(SPSub(scalefactor,FPONE),E[1]),E[0]);
    SMatrix[0][2].i=SPMul(SPMul(SPSub(scalefactor,FPONE),E[2]),E[0]);
    SMatrix[0][3].f=FPZERO;
    SMatrix[1][0].i=SPMul(SPMul(SPSub(scalefactor,FPONE),E[0]),E[1]);
    SMatrix[1][1].i=SPAdd(SPMul(SPMul(SPSub(scalefactor,FPONE),E[1]),
        E[1]),FPONE);
    SMatrix[1][2].i=SPMul(SPMul(SPSub(scalefactor,FPONE),E[2]),E[1]);
    SMatrix[1][3].f=FPZERO;
    SMatrix[2][0].i=SPMul(SPMul(SPSub(scalefactor,FPONE),E[0]),E[2]);
    SMatrix[2][1].i=SPMul(SPMul(SPSub(scalefactor,FPONE),E[1]),E[2]);
    SMatrix[2][2].i=SPAdd(SPMul(SPMul(SPSub(scalefactor,FPONE),E[2]),
        E[2]),FPONE);
    SMatrix[2][3].f=FPZERO;
    SMatrix[3][0].f=FPZERO;
    SMatrix[3][1].f=FPZERO;
    SMatrix[3][2].f=FPZERO;
    SMatrix[3][3].f=FPONE;
    if (clear) { VGCopylongmatrix(SMatrix,RMatrix); }
    else { VGMullongmatrix(SMatrix,RMatrix,RMatrix); }
}

/* Dreidimensionale Drehmatrix erzeugen */
void VGrotate(RMatrix,Xangle,Yangle,Zangle,clear)
union VGzahl RMatrix[][VGDIM];
int Xangle,Yangle,Zangle;
BOOL clear;
{
    union VGzahl VGRx,VGRy,VGRz,XMatrix[VGDIM][VGDIM],
        YMatrix[VGDIM][VGDIM],ZMatrix[VGDIM][VGDIM];

    if (Xangle)
    {
        if (Xangle >= 0) { VGRx.f=VGdeg_rad(Xangle); }
        else { VGRx.f=VGdeg_rad(360+Xangle); }
        XMatrix[0][0].f=FPONE;
        XMatrix[0][1].f=FPZERO;
        XMatrix[0][2].f=FPZERO;
        XMatrix[0][3].f=FPZERO;
        XMatrix[1][0].f=FPZERO;
        XMatrix[1][1].i=SPCos(VGRx.i);
        XMatrix[1][2].i=SPSin(SPNeg(VGRx.i));
    }
}

```

Listing 1. (Fortsetzung)


```

XMatrix[1][3].f=FPZERO;
XMatrix[2][0].f=FPZERO;
XMatrix[2][1].f=SPSin(VGRx.1);
XMatrix[2][2].f=SPCos(VGRx.1);
XMatrix[2][3].f=FPZERO;
XMatrix[3][0].f=FPZERO;
XMatrix[3][1].f=FPZERO;
XMatrix[3][2].f=FPZERO;
XMatrix[3][3].f=FPONE;
}
if (Yangle)
{
    if (Yangle >= 0) { VGRy.f=VGdeg_rad(Yangle); }
    else { VGRy.f=VGdeg_rad(360+Yangle); }
    YMatrix[0][0].f=SPCos(VGRy.1);
    YMatrix[0][1].f=FPZERO;
    YMatrix[0][2].f=SPSin(VGRy.1);
    YMatrix[0][3].f=FPZERO;
    YMatrix[1][0].f=FPZERO;
    YMatrix[1][1].f=FPONE;
    YMatrix[1][2].f=FPZERO;
    YMatrix[1][3].f=FPZERO;
    YMatrix[2][0].f=SPNeg(SPSin(VGRy.1));
    YMatrix[2][1].f=FPZERO;
    YMatrix[2][2].f=SPCos(VGRy.1);
    YMatrix[2][3].f=FPZERO;
    YMatrix[3][0].f=FPZERO;
    YMatrix[3][1].f=FPZERO;
    YMatrix[3][2].f=FPZERO;
    YMatrix[3][3].f=FPONE;
}
if (Zangle)
{
    if (Zangle >= 0) { VGRz.f=VGdeg_rad(Zangle); }
    else { VGRz.f=VGdeg_rad(360+Zangle); }
    ZMatrix[0][0].f=SPCos(VGRz.1);
    ZMatrix[0][1].f=SPSin(SPNeg(VGRz.1));
    ZMatrix[0][2].f=FPZERO;
    ZMatrix[0][3].f=FPZERO;
    ZMatrix[1][0].f=SPSin(VGRz.1);
    ZMatrix[1][1].f=SPCos(VGRz.1);
    ZMatrix[1][2].f=FPZERO;
    ZMatrix[1][3].f=FPZERO;
    ZMatrix[2][0].f=FPZERO;
    ZMatrix[2][1].f=FPZERO;
    ZMatrix[2][2].f=FPONE;
    ZMatrix[2][3].f=FPZERO;
    ZMatrix[3][0].f=FPZERO;
    ZMatrix[3][1].f=FPZERO;
    ZMatrix[3][2].f=FPZERO;
    ZMatrix[3][3].f=FPONE;
}
if (Xangle)
{
    if (clear) { VGCopylongmatrix(XMatrix,RMatrix); }
    else { VGMullongmatrix(XMatrix,RMatrix,RMatrix); }
    if (Yangle) { VGMullongmatrix(YMatrix,RMatrix,RMatrix); }
    if (Zangle) { VGMullongmatrix(ZMatrix,RMatrix,RMatrix); }
}
else
{
    if (Yangle)
    {
        if (clear) { VGCopylongmatrix(YMatrix,RMatrix); }
        else { VGMullongmatrix(YMatrix,RMatrix,
                                RMatrix); }
        if (Zangle) { VGMullongmatrix(ZMatrix,RMatrix,
                                RMatrix); }
    }
    else
    {
        if (Zangle)
        {
            if (clear) { VGCopylongmatrix(ZMatrix,RMatrix); }
            else { VGMullongmatrix(ZMatrix,RMatrix,
                                    RMatrix); }
        }
    }
}
}

```

Listing 1. (Schluß)

```

/*****\
|   VG - Vector Graphics   |
|*****|
|   Heiko Schlichting     |
|   Dirk Mehren           |
|   Berlin / Cuxhaven     |
|*****\
#define VG_H
#define VG_H
#include <intuition/intuitionbase.h>
#include <graphics/gfxbase.h>
#include <exec/memory.h>
#include <ctype.h>
#include <stdio.h>
#include <time.h>
#ifdef AZTEC_C
#include <functions.h>
#endif
#define INTUITION 0x00000001
#define GRAPHICS 0x00000002
#define SCREEN 0x00000004
#define LINEMEM 0x00000008
#define POINTMEM 0x00000010
#define MATHFFP 0x00000020
#define MATHTRANS 0x00000040
#define VGDIM 4
#define VGPI ((float)3.1415926525)
#ifdef LIBRARIES_MATHFFP_H
#define FPONE ((float)1.0)
#define FPZERO ((float)0.0)
extern int SPFFix();
extern int SPFFit();
extern int SPNeg();
extern int SPAdd();
extern int SPSub();
extern int SPMul();
extern int SPDiv();
extern int SPSin();
extern int SPCos();
#endif
/* ERRORMESSAGES */
#define ERR_OPINTUITION "OpenLibrary (Intuition)"
#define ERR_OPGFX "OpenLibrary (Gfx)"
#define ERR_OPMATHFFP "OpenLibrary (MathFFP)"
#define ERR_OPMATHTRANS "OpenLibrary (MathTrans)"
#define ERR_OPSCR "OpenScreen"
#define ERR_OUTMEM "Out of Memory"
#define ERR_DISPX "Invalid Viewresolution (X)"
#define ERR_DISPY "Invalid Viewresolution (Y)"
#define ERR_DISPB "Bitplane"
#define ERR_NOTHING "Nothing to do: LOGIC"
#define ERR_FIRSTOPEN "Requires VGopen"
#define ERR_FALSERGB "Out of RGB bounds"
#define ERR_FALSEPAR "Out of Colortable"
#define ERR_ALREADYOP "Second VGopen without VGclose"
#define ERR_TOOMLINES "Number of Lines > MaxLines"
#define ERR_TOOMPOINTS "Number of Points > MaxPoints"
#define ERR_UNDEFPPOINT "Line with undefined Point"
#define ERR_CANTOPENFILE "File not found, can't open it"
#define ERR_LOWRESMEM "Out of Line/Point-Memory"
union VGzahl {
    float f;
    int i;
};
struct VGLine {
    int Number;
    int StartPoint;
    int EndPoint;
    short Color;
};
struct VGPoint {
    int Object;
    union VGzahl Koord[VGDIM];
};
struct VGColor {
    unsigned short R,G,B;
};
#endif

```

Listing 2. Das von »VG« benötigte Include-File »VG.h«


```

/*****\
|   VG - Vector Graphics   |
|*****\
|   Heiko Schlichting     |
|   Dirk Mehren           |
|   Berlin / Cuxhaven     |
\*****/

#include "VG.h"
#define ALLOBJ 0
#define OBJ1 1
#define OBJ2 2
#define OBJ3 3
#define OBJ4 4
extern struct Screen *VGscreen;
struct VGColor FarbTab1[4]={0x0,0x0,0x4,0xf,0xf,0xf,0xb,0xb,0xb,0x7,0x7,0x7};
struct VGColor FarbTab2[4]={0x8,0x8,0x8,0x0,0x0,0xf,0x0,0xf,0x0,0xf,0x0,0x0};
union VGzahl XMatrix[40][VGDIM][VGDIM],
             YMatrix[40][VGDIM][VGDIM],
             AMatrix[VGDIM][VGDIM],
             BMatrix[VGDIM][VGDIM];

main()
{
    register int i;
    int    DeltaX,DeltaY,OldX,OldY,Grad;
    float   SVektor[3];

    DeltaX=DeltaY=0;
    Grad=OldX=OldY=1;
    printf("\nWenn die Bildschirmfarbe GRAU ist,\nkann das Objekt mit
    der Maus bewegt werden!\n");
    printf("PROGRAMMENDE durch positionieren der Maus in der linken
    oberen Ecke!\n");
    printf("\nWeiter mit RETURN\n");
    getchar();
    VGopen(320,256,2,100,100,"RAM:SYSPRINT");
    VGrotate(XMatrix[20],-1,0,0,TRUE);
    VGrotate(YMatrix[20],0,-1,0,TRUE);
    VGrotate(XMatrix[19],+1,0,0,TRUE);
    VGrotate(YMatrix[19],0,+1,0,TRUE);
    for (i=0;i<19;i++)
    {
        VGrotate(XMatrix[18-i],+Grad,0,0,TRUE);
        VGrotate(XMatrix[i+21],-Grad,0,0,TRUE);
        VGrotate(YMatrix[18-i],0,+Grad,0,TRUE);
        VGrotate(YMatrix[i+21],0,-Grad,0,TRUE);
        Grad++;
    }
    VGreaddata("VGTEST.DAT");
    VGdisplay(TRUE);
    SVektor[0]=0.0;
    SVektor[1]=0.5;
    SVektor[2]=0.0;
    VGlocalscale(AMatrix,SVektor,0.25,TRUE);
    for (i=0;i<20;i++)
    {
        VGtransform(ALLOBJ,AMatrix);
        VGdisplay(TRUE);
    }
    VGrotate(AMatrix,0,3,0,TRUE);
    for (i=0;i<180;i+=3)
    {
        VGtransform(ALLOBJ,AMatrix);
        VGdisplay(TRUE);
    }
    VGlocalscale(AMatrix,SVektor,-0.08,FALSE);
    for (i=0;i<60;i++)
    {
        VGtransform(ALLOBJ,AMatrix);
        VGdisplay(TRUE);
    }
    for (i=0;i<18;i++) { VGline(36+i,1,i+18,2); }
    VGrotate(AMatrix,0,3,4,TRUE);
    VGrotate(BMatrix,3,4,0,TRUE);
    for (i=0;i<360;i+=5)
    {
        VGtransform(OBJ1,AMatrix);
        VGtransform(OBJ3,AMatrix);
        VGtransform(OBJ2,BMatrix);

```

```

        VGtransform(OBJ4,BMatrix);
        VGdisplay(TRUE);
    }
    VGcolor(0,4,FarbTab1);
    VGrotate(AMatrix,2,2,1,TRUE);
    VGglobalscale(AMatrix,0.967,FALSE);
    VGtranslate(AMatrix,0.2,0.3,-0.2,0.05,FALSE);
    for (i=0;i<20;i++)
    {
        VGtransform(ALLOBJ,AMatrix);
        VGdisplay(TRUE);
    }
    VGtranslate(AMatrix,0.2,0.3,-0.2,-0.04,TRUE);
    for (i=0;i<20;i++)
    {
        VGtransform(ALLOBJ,AMatrix);
        VGdisplay(TRUE);
    }
    VGglobalscale(AMatrix,1/0.967,TRUE);
    for (i=0;i<20;i++)
    {
        VGtransform(ALLOBJ,AMatrix);
        VGdisplay(TRUE);
    }
    VGrotate(AMatrix,0,1,0,TRUE);
    for (i=0;i<30;i++)
    {
        VGtransform(OBJ3,AMatrix);
        VGtransform(OBJ4,AMatrix);
        VGdisplay(TRUE);
    }
    VGrotate(AMatrix,0,-1,0,TRUE);
    for (i=0;i<30;i++)
    {
        VGtransform(OBJ3,AMatrix);
        VGtransform(OBJ4,AMatrix);
        VGdisplay(TRUE);
    }
    VGtranslate(AMatrix,0.2,0.2,0.0,-0.02,TRUE);
    for (i=0;i<10;i++)
    {
        VGtransform(ALLOBJ,AMatrix);
        VGdisplay(FALSE);
    }
    VGeraseobj(ALLOBJ);
    VGdisplay(TRUE);
    VGreaddata("VGTEST.DAT");
    for (i=0;i<18;i++) { VGline(36+i,1,i+18,2); }
    VGcolor(0,4,FarbTab2);
    VGdisplay(TRUE);
    while (OldX || OldY)
    {
        DeltaX = OldX-(unsigned int)(VGscreen->MouseX);
        DeltaY = OldY-(unsigned int)(VGscreen->MouseY);
        OldX = (unsigned int)(VGscreen->MouseX);
        OldY = (unsigned int)(VGscreen->MouseY);
        if ((DeltaX < 20) && (DeltaX > -20) && DeltaX)
        {
            VGtransform(ALLOBJ,YMatrix[19+DeltaX]);
        }
        if ((DeltaY < 20) && (DeltaY > -20) && DeltaY)
        {
            VGtransform(ALLOBJ,XMatrix[19+DeltaY]);
        }
        if (DeltaY || DeltaX) { VGdisplay(TRUE); }
    }
    VGclose(FALSE);

```

Listing 3.
»VGTest« demonstriert
die fantastischen
Fähigkeiten von »VG«


```

{
    REGISTER SHORT i;
    for(i=0; i<VS; i++)
        FreeVSprite(VSP[i], RP, VP, 1);
    if(Ball)
        FreeMem(Ball, sizeof(BallDat));
    if(GelsInfo)
        FreeGelsInfo(GelsInfo);
    if(win)
        CloseWindow(win);
    if(GfxBase)
        CloseLibrary(GfxBase);
    if(IntuitionBase)
        CloseLibrary(IntuitionBase);
    exit(0);
}

VOID OpenW()
{
    REGISTER SHORT i, c, hm, x, y;
    if(!IntuitionBase=(struct IntuitionBase *)
        OpenLibrary("intuition.library", 0)) CloseW();
    if(!GfxBase=(struct GfxBase *)
        OpenLibrary("graphics.library", 0)) CloseW();
    if(!win=OpenWindow(&win)) CloseW();
    VP=ViewPortAddress(win);
    RP=win->RPort;
    if(!Ball=(WORD *)AllocMem(sizeof(BallDat), MEMF_CHIP|MEMF_PU-
        BLIC)) CloseW();

    CopyMem(BallDat, Ball, sizeof(BallDat));
    if(!GelsInfo=GetGelsInfo(RP, 0xFE, 202, 55, 435, 185)) CloseW();
    SetCollision(0, BorCon, GelsInfo);
    for(i=0; i<5; i++) SetCollision(i+1, MarCon, GelsInfo);
    for(c=i=0; i<VS; i++, c++){
        if(c>4) c=0;
        hm=2<<c;
        x=210+i*10;
        y=60+i*5;
        if(!VSP[i]=GetVSprite(RP, x, y, 7, hm, hm+1, Ball, Col[c],
            1))) CloseW();
    }
    DrawGels(RP, VP);
}

ULONG GetMessage(code)
    USHORT *code;
{
    struct IntuiMessage *msg;
    ULONG class=0;

    if(msg=(struct IntuiMessage *)GetMsg(win->UserPort)){
        class=msg->Class;
        *code=msg->Code;
        ReplyMsg(msg);
    }
    return(class);
}

VOID main()
{
    REGISTER SHORT i;
    USHORT code;
    ULONG class;
    OpenW();
    while(1){
        class=GetMessage(&code);
        if(class==CLOSEWINDOW) break;
        WaitTOF();
        for(i=0; i<VS; i++){
            if(VSP[i]){
                VSP[i]->X+=vX[i];
                VSP[i]->Y+=vY[i];
            }
        }
        DrawGels(RP, VP);
        DoCollision(RP);
        if(Id1+Id2){
            DisplayBeep(0);
            FreeVSprite(VSP[Id1], RP, VP, 0);
            VSP[Id1]=NULL;
            Id1=Id2=0;
        }
    }
    CloseW();
}

```

Listing 4. Das Demo-Programm »VS« stellt 20 VSprites gleichzeitig in 15 Farben dar

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Redaktionsdirektor: Richard Kerler

Chefredakteur: Hans-Günther Beer

Stellv. Chefredakteur: Gottfried Knechtel (kn) – verantwortlich für den redaktionellen Teil

Chef vom Dienst: Susanne Kirmaier

Redaktion: Klaus Sonnenleiter (so), Albert Petryszyn (pe)

Mitarbeiter dieser Ausgabe: Ralph Babel, Arno Gözler, Martin Jobst, Dirk Mehren, Andreas Regul, Heiko Schlichting, Fridtjof Siebert, Jürgen Singer, Frank Staudte, Lutz Vieweg, Hans Waldhäusel

Redaktionsassistent: Brigitte Bobenstetter, Sylvia Derenthal, Helga Weber (202)

Telefax: 089/4613-778, Hotline: Donnerstag 14 bis 17 Uhr

Alle Artikel sind mit dem Kurzzeichen des Redakteurs und/oder mit dem Namen des Autors/Mitarbeiters gekennzeichnet

Manuskripteinsendungen: Manuskripte und Programm Listings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten worden sein, muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt & Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programm Listings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt & Technik Verlag AG verlegten Publikationen. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Art-director: Friedemann Porscha

Layout: Erich Schulze (Cheflayout), Marian Schwarz, Johanna Schneider

Bildredaktion: Janos Feitser (Ltg.), Sabine Tennstaedt; Roland Müller (Fotografie); Ewald Standke, Norbert Raab (Spritzgrafik); Werner Nienstedt (Computergrafik)

Bildnachweis: Südde. Zeitung (Titelmotiv)

Anzeigenleitung: Alicia Clees (313) – verantwortlich für die Anzeigen

Telefax: 089/4613-775

Anzeigenverwaltung und Disposition: Monika Burseg (147)

Auslandsrepräsentation:

Anzeigenverkaufsleitung Ausland: Thomas Schlüter (299)

Auslandsniederlassungen:

Schweiz: Markt & Technik Vertriebs AG, Kollerstr. 3, CH-6300 Zug,

Tel. 042-41 56 56, Telex: 862 329 mut ch

USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City, CA 94063, Telefon: (415) 366-3600, Telex 752-351

Österreich: Markt & Technik Ges. mbH
Große Neugasse 28, A 1040-Wien
Telefon: 0222/5871393, Telex: 047-132532

Anzeigen-Auslandsvertretung:

England: F. A. Smyth & Associates Limited, 23a, Aylmer Parade, London, N2 0PQ.

Telefon: 0044/1/3405058, Telefax: 0044/1/341 9602

Israel: Baruch Schaefer, Haeskel-Str. 12, 58348 Holon, Israel, Tel. 00972-3-5562256

Taiwan: Third Wave Publishing Corp., 1-4 Fl. 977 Min Shen E. Road, Taipei 10581,

Taiwan, R.O.C., Tel. 00886/2/7630052, Telefax: 00886/2/7658767, Telex: 078529335

Vertriebsleitung: Helmut Grünfeldt (189)

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Buchhandelsbuchhandel) sowie Österreich und Schweiz: ip Internationale Presse, Hauptstätter Straße 96, 7000 Stuttgart 1, Tel. 0711/6483-110

Bezugsmöglichkeiten: Leser-Service: Telefon (089) 4613-366. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.

Preis: Das Einzelheft kostet DM 16,-

Produktion: Technik: Klaus Buck (Ltg./180), Wolfgang Meyer (Stellv./887);

Herstellung: Otto Albrecht (Ltg./917)

Druck: SOV Graphische Betriebe, Laubanger 23, 8600 Bamberg

Urheberrecht: Alle in diesem Heft erschienenen Beiträge sind urheberrechtlich geschützt. Alle Rechte, auch Übersetzungen, vorbehalten. Reproduktionen, gleich welcher Art, ob Fotokopie, Mikrofilm oder Erfassung in Datenverarbeitungsanlagen, nur mit schriftlicher Genehmigung des Verlages. Aus der Veröffentlichung kann nicht geschlossen werden, daß die beschriebenen Lösungen oder verwendeten Bezeichnungen frei von gewerblichen Schutzrechten sind.

Haftung: Für den Fall, daß in diesem Heft unzutreffende Informationen oder in veröffentlichten Programmen oder Schaltungen Fehler enthalten sein sollten, kommt eine Haftung nur bei grober Fahrlässigkeit des Verlages oder seiner Mitarbeiter in Betracht.

Sonderdruck-Dienst: Alle in dieser Ausgabe erschienenen Beiträge sind in Form von Sonderdrucken zu erhalten. Anfragen an Reinhard Jarczok, Tel. 089/4613-185, Fax 4613-776.

© 1989 Markt & Technik Verlag Aktiengesellschaft

Vorstand: Otmar Weber (Vors.), Bernd Balzer, Richard Kerler

Leiter Produktmarketing: Hans-Günther Beer

Direktor Zeitschriften: Michael M. Pauly

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen: Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon 089/4613-0, Telex 522052, Telefax 089/4613-773

Telefon-Durchwahl im Verlag: Wählen Sie direkt: Per Durchwahl erreichen Sie alle Abteilungen direkt. Sie wählen 089/4613 und dann die Nummer, die in den Klammern hinter dem jeweiligen Namen angegeben ist.

Stoff für Ihren Amiga



HiSoft-Basic-Compiler

Mit HiSoft Basic gibt es endlich auch einen Basic-Compiler für den Amiga. Der interaktive Editor-, Kompilier- und Laufzeitzyklus entspricht dem eines Interpreters.

Der integrierte Editor erlaubt eine komfortable Eingabe. HiSoft Basic unterstützt die Eigenschaften des Amiga mit Fenstern, Grafik-Kommandos, Sprite-Handling und Maschinenzugriffen während des Gebrauchs von Bibliotheken. Es ist voll kompatibel mit Amiga-Basic, andere Standarddialekte für den PC, wie z.B. Microsoft QuickBasic, und den Atari ST können mit geringen Modifikationen kompiliert werden. Rekursive

Unterprogramme und Funktionen sind möglich. Eine Anzahl strukturierter Ausdrücke wie z.B. WHILE...WEND, DO...LOOP UNTIL und SELECT...CASE lassen Sie jede Programmierhürde meistern. Die Größe von Variablen ist nicht beschränkt. Aussagekräftige Fehlermeldungen und Korrekturmöglichkeiten tragen zur komfortablen Handhabung bei.

Hardware-Anforderungen: Amiga 500, 1000 oder 2000 mit Kickstart 1.2 oder höher. 3 1/2"-Diskette, Bestell-Nr. 54127

DM 179,-*
(sFr 161,- /öS 1790,-*)

Devpac Assembler 2.0

Devpac 2.0 ist ein Entwicklungspaket für den Amiga mit komfortablem Editor/Assembler, symbolischem Debugger und Linker zum Einbinden von Hochsprache-Modulen.

GenAm ist ein 68000er-Makroassembler mit integriertem Bildschirmditor, der bis zu 75.000 Zeilen pro Minute assemblieren kann. Der 2-Paß-Assembler erzeugt sowohl linkbaren als auch direkt ausführbaren Code. Er unterstützt lokale Labels, die Signifikanz beträgt bis zu 127 Zeichen.

Makros können bis zu 36 Parameter beinhalten und – Rekursion inbegriffen – so tief verschachtelt werden, wie Speicherplatz vorhanden ist. MonAm, der Debugger, erlaubt das Setzen von Breakpoints, das Disassemblieren auf Diskette und noch weitere Features, die das Debuggen zum Vergnügen machen.

Hardware-Anforderungen: Amiga 500, 1000 oder 2000 mit mindestens 512 Kbyte, ein Diskettenlaufwerk. 3 1/2"-Diskette, Bestell-Nr. 54131

DM 149,-*
(sFr 135,- /öS 1490,-*)

Devpac-Assembler 2.0 im Test! Amiga-Magazin 4/89: Eines der besten Programme seiner Art für den Amiga.

SuperED C

Multitasking-fähiger Editor als Programmierumgebung für den Aztec-C-Compiler (V 3.6). Bestell-Nr. 54139

DM 39,-*
(sFr 35,- /öS 390,-*)

* Unverbindliche Preisempfehlung

INFO-COUPON

Bitte senden Sie mir weiteres Informationsmaterial zu

☐ Devpac-Assembler ☐ HiSoft-Basic-Compiler

Name _____

Straße _____

PLZ/Ort _____

Bitte ausschneiden und einsenden an: Markt&Technik Verlag AG, Buchverlag, Frau Brosien, Hans-Pinsel-Straße 2, 8013 Haar bei München

Ami SH 7

2211/907



clevere Amiga-User verwenden gleich

ProMigos

denn wir verwenden nur ausgesuchte Qualitätsware für unsere Produkte, die auch höchsten Ansprüchen gerecht werden. Und das bei einem sehr guten Preis-Leistungs-Verhältnis!!

ProMigos Zubehör rund um den Amiga:

elektronischer Bootselektor

wahlweise booten von DF0: – DF2; (optional auch DF3:)

nur 58,00 DM

ROM/ROM Kickstart-Umschaltplatine

Zwischen zwei original ROMs (40polig) umschalten

nur 49,00 DM

ProMigos 3,5"-Floppystation

abschaltbar, durchgeführter Bus, extrem leise, NEC-Drive

nur 279,00 DM

ProMigos Boot-Strap-Platine

beliebige Kickstart wie beim A1000 aus RAM booten (Lieferung ohne RAM & A1000-BootROMs)

nur 98,00 DM

ProMigos Autoboot-Harddisk-Station für A500 & A1000

Kapazität von 20 bis 110 MByte, ALF2-Treibersystem inkl. Backup-Software

ab 1168,00 DM

ProMigos 512 KByte RAMCARD

mit eingebauter Uhr, abschaltbar, für A500

nur 319,00 DM

ProMigos Kickstart-Umschaltplatine

zwischen einem ROM & zweimal Eprom umschalten

nur 68,00 DM

ProMigos 5,25"-Floppystation

40/80 Track, abschaltbar, durchgeführter Bus, TEAC-Drive

nur 349,00 DM

Autoboot-Upgrade-Kit

Autoboot-Upgrade-Kit für vorhandene *ProMigos* & *Amigos* Harddiskstationen inkl. ALF2-Treiber und Backup-Software

nur 248,00 DM

ProMigos Autoboot-FileCard für A2000

Kapazität von 20 bis 50 MByte (intern), ALF2-Treibersystem inkl. Backup-Software

ab 1120,00 DM

- bitte ausführliche Info anfordern
- *ProMigos*-Produkte erhalten Sie direkt bei uns oder im guten Fachhandel
- deutsche Qualitätsfertigung
- Ausländische Distributoren gesucht
- Händleranfragen erwünscht!!

Amiga und Commodore sind eingetragene Warenzeichen der Firma Commodore Inc.

Flesch & Hörnemann Computer OHG
Schlägel & Eisen Straße 46
4352 Herten
Telefon: 049-0 23 66-5 51 76
Telefax: 049-0 23 66-5 34 50