NO. 26 JANUARY 1985 $3.00

THE #1 MAGAZINE FOR ATARI® COMPUTER OWNERS™

A.N.A.L.O.G. COMPUTING

ATARI GRAPHICS!

JACK TRAMIEL INTERVIEWED!

OSS WRITES ONLY PRECISION SOFTWARE... OUR CUSTOMERS WRITE OUR BEST ADS!

BASIC XL

"BASIC XL is a fast and powerful extension of Atari BASIC, totally compatible with virtually all software. Its many features make programming easy, especially games that require player/missile graphics. For people writing business software or translating existing programs from other computers, the new string arrays and other string-handling features make the task manageable. BASIC XL is a truly professional language that should become standard in all future Atari computers. Overall Rating—A." *The Addison-Wesley Book of Atari Software 1984*

BASIC XL SuperCartridge & Manual ***(Requires 16K Memory)*** **$99.00**

ACTION!

"For those who have found BASIC to be too slow or assembler to difficult, ACTION! is the logical alternative. ACTION! programs can increase speed from 50 to 200 times that of BASIC." *Jerry White, Antic, February 1984*

ACTION! SuperCartridge & Manual ***(Requires 16K Memory)*** **$99.00**

MAC/65

"For the serious machine language programmer or anyone interested in programming in 6502 machine language, this package is a must. A lot of the good professional software on the market, games or otherwise, was written using this brute. Coding machine language with anything else is like trying to swim upstream in quicksand." *ACE Of West Hartford, May 1984*

MAC/65 SuperCartridge & Manual ***(Requires 16K Memory)*** **$99.00**

THE WRITER'S TOOL was designed for WRITERS who want to WRITE!

When you want to write on your Atari® Computer—a letter, a business report, or a book—**you want to concentrate on writing and not on the word processor** that you are using.

You want to **write immediately without** having to spend hours **learning some fancy system or remembering complicated commands.**

You want to **write quickly...**as fast as your imagination can fly. Then **edit just as fast...**correct mistakes; change, add, or delete words and phrases; rearrange the sequence of paragraphs or entire pages **without fearing that one slip of your fingers might wipe-out hours of precious work.**

You want to be able to save or retrieve your text files **on any type of Atari compatible disk drive** using either single-density or double-density disks without worrying about accidentally erasing files.

You want to be able to **print out your finished piece right away,** using all of **your printer's capabilities** such as Pica, Elite, and condensed type; print two-columns on 8½" wide paper and proportional spacing; and print different types of characters: double-width, **boldface**, *italics*, underlined, superscript, and subscript.

Compare THE WRITER'S TOOL with the others: feature for feature, dollar for dollar—**if you can find a better word processor, buy it!**

THE WRITER'S TOOL*

SuperCartridge, Program Disk, Tutorial & Reference Manual **$129.95**

**Requires an Atari® Computer with 40K memory, disk drive, and any Atari compatible printer.*

ATARI® is the registered trademark of ATARI, INC.

Now Available At Your Software Dealer!

OSS™

Optimized Systems Software, Inc.

1221B Kentwood Avenue, San Jose, California 95129 (408) 446-3099

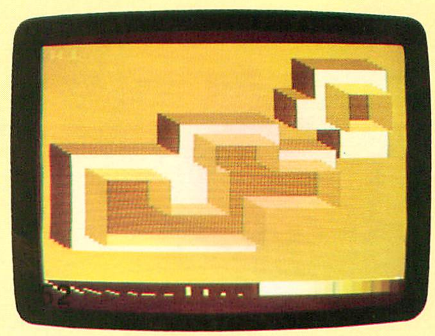
CIRCLE #101 ON READER SERVICE CARD

NO. 26 JANUARY 1985

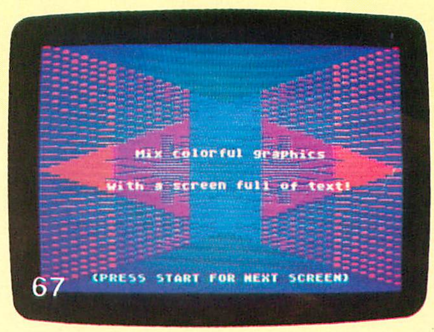
THE #1 MAGAZINE FOR ATARI® COMPUTER OWNERS

A.N.A.L.O.G. COMPUTING™

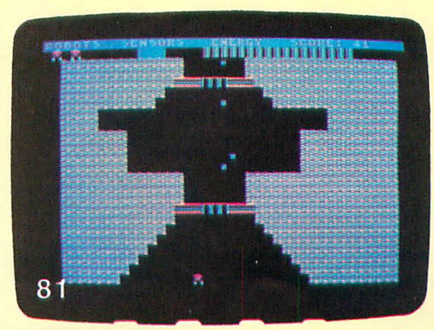
FEATURES



REVIEWS



COLUMNS



ANALOG COMPUTING (ISSN 0744-9917) is published monthly for $28 per year by ANALOG 400/800 Corp., 565 Main Street, Cherry Valley, MA 01611. Second class postage paid at Worcester, MA and additional mailing offices.

FOR COMMODORE AND ATARI

WHEN BATTERIES INCLUDED SET OUT TO DESIGN THE VERY BEST WORD PROCESSOR FOR ATARI® COMPUTERS... THEY FOUND THEY ALREADY HAD IT.

Features all standard word processing functions PLUS

- Full Screen Editing
- Dual text windows with cut and paste between windows.
- Video preview with scrolling, lets you see exactly what will be printed, *before* it is printed
- Transfer text between documents – easily, simply
- Horizontal scrolling – create wide documents with ease
- Re-arrange words, phrases, sentences, paragraphs, and columns with just a few keystrokes
- More than 40 different printer drivers – plus printer editor allows programming for any conceivable new printer
- Easy to learn, easy to use

WORKS WITH THE NEW ATARI B.I. 80 COLUMN DISPLAY ADAPTOR FOR XL SYSTEMS.

Compatible with Atari 400, 800, 600 XL, 800 XL, 1200 XL

PAPERCLIP
ATARI®

Few word processors have allowed Atari users to tap the full resources of their computer ***until Atari Paper Clip****...*
Atari Paper Clip is an extremely powerful, fully featured word processor that will allow your Atari to operate to the limits of its potential, with an ease of operation and speed you've never thought possible.

PAPER CLIP FOR ATARI® AND COMMODORE® OWNERS WHO WANT THE VERY BEST IN WORD PROCESSING.

BATTERIES INCLUDED

"The Energized Software Company!"

186 Queen St. West
Toronto, Ontario,
M5V 1Z1 Canada
(416) 596-1405

17875 Sky Park North,
Suite P, Irvine, California
USA 92714

FOR A FULL COLOR BROCHURE SEND A STAMPED SELF-ADDRESSED #10 ENVELOPE TO BATTERIES INCLUDED.

CIRCLE #102 ON READER SERVICE CARD

ANALOG COMPUTING STAFF

Editors/Publishers
MICHAEL DES CHENES
LEE H. PAPPAS

Managing Editor
JON A. BELL

Production Editor
DIANE L. GAW

Contributing Editors
DONALD FORBES
BRADEN E. GRIFFIN, M.D.
TONY MESSINA

East Coast Editor
ARTHUR LEYENBERGER

West Coast Editor
JIM DUNION

Art Director
BOB DESI

Contributing Artists
GARY LIPPINCOTT
LINDA RICE

Technical Division
CHARLES BACHAND
TOM HUDSON
KYLE PEACOCK

Advertising Manager
MICHAEL DES CHENES

Distribution
PATRICK J. KELLEY

Production/Distribution
LORELL PRESS, INC.

Contributors
PHILIP ALTMAN
JEFF BRENNER
JOEL GLUCK
MATT HILLMAN
CHARLES KORMOS
BERNARD MIKOWSKI
MICHAEL MIKOWSKI
KATHY SLOATMAN
MARK SLOATMAN
CHESTER C. WALTERS

Statement of Ownership,
Management and Circulation

Title .... ANALOG Computing
Frequency of issue .... Monthly
No. of issues annually 12
Annual subscription price $28.00
Mailing address .... 565 Main Street
Cherry Valley, MA 01611
Headquarters .... 565 Main Street
Cherry Valley, MA 01611
Publisher .... Michael J. Des Chenes
Editor .... Lee H. Pappas
Managing Editor .... Jon A. Bell
Owner .... ANALOG 400/800
Magazine Corporation,
Michael J. Des Chenes
Lee H. Pappas
Total no. copies printed 84,000
Circulation data (average per issue for preceding 12 months)
Dealers and vendors 57,000
Mail subscription 26,000
Total paid circulation 83,000
Free distribution 300
Total distribution 83,300
Copies not distributed 200
Return from news agents ... 500
Total 84,000

For subscription information and service call toll-free:

1-800-345-8112

in Pennsylvania call 1-800-662-2444

ADVERTISING SALES

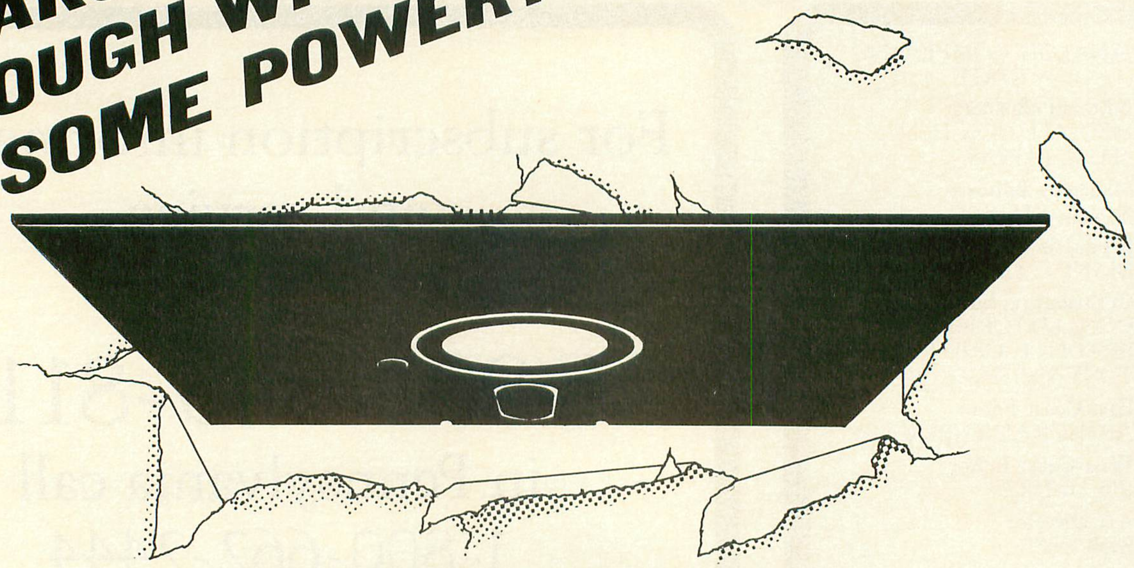
Gerald F. Sweeney & Associates
P.O. Box 662
New York, NY 10113
(212) 242-3540

ANALOG Computing
Home Office
Michael DesChenes
National Advertising
(617) 892-9230

Address all advertising materials to:
Michael DesChenes — Advertising Production
ANALOG Computing
565 Main Street, Cherry Valley, MA 01611

ANALOG Computing (ISSN 0744-9917) is published monthly for $28 per year by ANALOG 400/800 Corp., 565 Main Street, Cherry Valley, MA 01611, Tel. (617) 892-3488. Second-class postage paid at Worcester, MA and additional mailing offices. POSTMASTER: Send address changes to **ANALOG Computing**, P.O. Box 615, Holmes, PA 19043. No portion of this magazine may be reproduced in any form without written permission of the publisher. Program listings should be provided in printed form. Articles should be furnished as typed copy in upper and lower case with double spacing. By submitting articles to **ANALOG Computing**, authors acknowledge that such materials, upon acceptance for publication, become the exclusive property of **ANALOG Computing**. If not accepted for publication, the articles and/or programs will remain the property of the author. If submissions are to be returned, please supply self-addressed, stamped envelope. U.S.A. newstand distribution by Eastern News Distributors, Inc., 111 Eighth Ave., New York, NY 10011.

BREAKS THROUGH WITH AWESOME POWER!



TOP-DOS

POWER YOU COMMAND.

FRIENDLY POWER

TOP-DOS unleashes the latent power of your ATARI computer—an amazing machine. TOP-DOS puts this power under your control. Here are just a few of its friendly features:

COMMAND MENU & HELP FILES speed your mastery of the system.
SUPERIOR STATUS DISPLAY keeps you informed. Shows free memory, disk-drive configuration, and state of TOPS-DOS options.
ERROR-CODE TRANSLATOR deciphers numeric codes into English.
BREAK-KEY ABORT lets you change your mind in mid-command.
FULL SCREEN USE shows you what you've done. Keeps 23 ines of past operations on display.
UNDELETE COMMAND rescues an accidentally-deleted file.

POWER TO PLEASE

Upgrade to TOP-DOS. Owners are delighted. You will be too. Only $49.95. No risk, 30-day MONEYBACK GUARANTEE.

SPECIFICATIONS

Memory required: 32K
Computers: ALL ATARI
Disk drives: 1 to 8
5¼": Single-density, Double-density, Double-sided Double-density
8": Double-density.
Ramdisk: AXLON or MOSAIC.
Files: Single density: 64*
Double density: 128*
Sectors: Single density: Up to 944
Double density: Up to 1968
Memory–residency: 0700-1A80 (hex)
(Same as ATARI DOS-2)
Commands: 55
(All ATARI DOS-2 + 40 more)
Command options: 35

See TOP-DOS at your dealer. If not available, you may order direct from ECLIPSE SOFTWARE, 1058-G Marigold Court, Sunnyvale, CA 94086,(408) 246-8325.

DEALER INQUIRIES WELCOME

SOPHISTICATED POWER

TOP-DOS offers professional features found only in systems on much larger machines. Whatever your experience level, you will appreciate the flexibility an(power of this advanced system. Here are some examples of TOP-DOS's powerful features:

MACHINE LANGUAGE MONITOR & EDITOR allows yo to access and change bytes in memory.
COMMAND FILE CAPABILITY permits you to simply and rapidly execute a complex sequence of command:
"HELLO" FILE executes automatically on boot-up.
SET COMMAND enables you to customize your systen Configure disk drives and select TOP-DOS options.
FILE DIRECTORY COMMAND lets you choose: Alphabetization, the number of columns in the listin; and the inclusion of deleted & open files.
MEMORY MAP shows you the memory areas used by the Binary Load command.
ONE-LINE COMMANDS saves you time and conserve: screen space, once you are familiar with the comman(syntax.
DOS-RESIDENT OPTION speeds your transfer between TOP-DOS & BASIC, or other programs.

ECLIPSE

TOP-DOS includes all the features of its predecessor, DOS-MOD. TOP-DOS and DOS-MOD are trademarks of ECLIPSE. ATARI is a registered trademark of Atari, Inc. Prices are subject to change without notice. Shipping charges are prepaid. California residents add 6.5% sales tax ($49.95 + 3.25 = $53.20).

*An advanced version is available to TOP-DOS owners (at additional cost), which doubles the number of files, as well as adding a number of other features.

CIRCLE #145 ON READER SERVICE CARD

READER COMMENT

Faster floating points.

I was very interested to see an article in *Byte* (August 1984) on the 16-bit "65816" microprocessor as a code-compatible successor to the 6502, as I have been doing a certain amount of scientific computing on the 6502-based Atari 400 and 800XL computers.

The main advantage of Ataris is that interactive graphics is easy and nice, but there is a problem with speed: the BASIC and the floating-point hardware are pretty slow. Accessing the floating-point routines with machine language only seems to speed things up by about 40%, at most, and is pretty clumsy.

I am sure you can anticipate one of my two questions:

(1) Will it be possible to speed things up simply by replacing the 6502 chip with the new 65816 in the Ataris, or would something far more extensive be necessary?

(2) I remember seeing somewhere an ad for a thing called the "Fastchip," which would be a faster replacement for the floating-point hardware chip for the Atari, but that was a year or so ago. Are those still available, and do you know where I might get a couple?

Thanks for your help.
Sincerely yours,
W.J. Cocke
Tucson, AR

While the 65816 processor chip is code compatible, it is not pin compatible. Adding one to the Atari would probably involve a redesigning of the computer's circuitry. If you want to change processors, the only thing that I can suggest at this time is the new CMOS 65C02. Not only is it pin compatible, but it even includes a few extra instructions that will come in handy for any machine language programmers.

The "Fastchip" is still being manufactured by Newell Industries, and it does speed your floating-point calculations substantially. Contact:

Newell Industries
3340 Nottingham Lane
Plano, TX 75074
(214) 423-1781

—C.B.

A piece of the Action!

I have been programming in the Action! language by OSS for a couple of months now, and have found it very versatile and powerful. Being a newcomer to the language, I am eager to find more information on how to use it.

However, I have found only a few articles about it in magazines. The manual that comes with the cartridge does have a lot of information and examples, but, as is usually the case with manuals, parts of it are difficult to understand.

I think a tutorial about Action! is needed. I have heard that OSS is working on a book that will be a tutorial on the language, but it will be some time before it is ready for sale. I think it would be nice if **ANALOG Computing** could offer some information, as well.

Action! is so fast and powerful a language, I believe it deserves a col-

(continued on page 8)

Merry Christmas Specials From CDY For Your Atari 400/800/800XL

OMNIMON! Resident Monitor

OMNIMON! is a PC board which plugs into your 400/800 (soon to be available for the XLs also) and gives you complete control of your computer. Even though it is always available (by pressing SELECT and SYSTEM RESET) it takes up no user memory because it resides in the unused 4K block at $C000. Use it to interrupt, examine, and manipulate any program in memory whether it be disk, cassette, or cartridge based. It is especially good for program development or customization of existing programs. The flexible disk I/O allows you to write to or read from disk in either single or double density. You can edit raw sector data or even load a file without DOS. Many debugging tools are at your disposal: Display/Alter memory or 6502 registers, Disassemble memory, Search memory, Hex/Char modes, Single Step execution, JSR or GOTO address, Push/Pull stack, Printer dump, etc. After interrupting a program with OMNIMON!, many times it is possible to return to the program as if you had never left it (e.g., BASIC, DOS, etc.). Instructions are provided for the addition of a simple toggle switch to make OMNIMON! invisible, thus making it compatible with all software. An external cable is now provided to eliminate the need to solder directly on the board.

New 8K OMNIMON! Upgrade

This enhancement, which is available to all OMNIMON! users, includes a substantial number of features not available in the standard version. The 8K OMNI resides in an 8K ROM which has been modified by the addition of a switch for selecting either of two 4K banks. The additional features include Hex Conversion and Hex Arithmetic, Block Move, a Relocater, and a Line Assembler. A Binary Load command allows you to load any binary load file without DOS and doubles as a disk directory command which prints out the start sector of each file. Lockup recovery allows you to recover from system lockup, meaning that when your computer freezes, you can usually salvage the program or text file in memory by popping into 8K OMNI and dumping memory to disk. Advanced users will like the user extendibility feature which allows them to make use of the interface routines of 8K OMNI in their own software. One of the most exciting features of the 8K OMNI is the resident Ramdisk handlers. They allow AXLON Ramdisk owners to use this powerful device with any DOS which uses standard SIO calls and even with boot programs like word processors and games which access the disk a lot. Several additional features make this version very valuable for advanced programmers, but if you have a Ramdisk, 8K OMNI is a MUST!

New OMNIVIEW 80 Column Upgrade

Did you know that for most applications you do not need an expensive, slot consuming 80 column board to enjoy the power of 80 columns? Would you 400 owners like the convenience of 80 columns? OMNIVIEW takes advantage of the high resolution graphics mode built into the ATARI to generate an 80 column screen editor essentially identical to the ATARI screen editor (E:, S:). Thus, you can use OMNIVIEW in any environment where you would normally use the 40 column "E:" (e.g., BASIC, Assembler/Editor, etc.). The 80 column "E:" of OMNIVIEW has been optimized for speed so that it is not significantly slower than 40 column "E:". In addition, the character font was specially designed to be legible on an ordinary TV set! A monitor is recommended, but not really necessary for casual 80 column operation. The Bit-3 version of LJK's 80 column Letter Perfect has been modified to support OMNIVIEW and other programs are sure to follow. Lastly, the Ramdisk handlers described under 8K OMNI are also incorporated in OMNIVIEW.

New RAMROD-XL

800XL owners are now able to equip their computers with OMNIMON and OMNIVIEW. In addition, the Newell enhanced operating system and Fastchip floating point package will be included at no extra charge. This will essentially turn your 800XL back into a 400/800 compatible machine and allow it to run most of the software which the XL-OS will not. A switch will allow you to select the XL-OS when needed.

	OMNIMON Piggy-back 400/800	Ramrod OS Board 800	Upgrades for OMNIMON or Ramrod: 8K OMNI	8K VIEW	4K VIEW	Ramrod-XL Piggy-back 800XL	Addon for Ramrod-XL VIEWXL
Enhanced OS w/Fast Cursor		+				+	+
Includes FASTCHIP FP						+	+
80 Columns Emulation				+	+		+
AXLON Ramdisk Handlers			+	+	+		
OMNIMON Features:							
A:Alter Memory	+	+	+	+		+	
B:Boot (Ram) disk			+	+	+		
C:CPU Registers	+	+	+	+		+	
D:Display Memory	+	+	+	+		+	
E:Single Step Execution	+	+	+	+		+	
F:Fill Program Buffer			+			+	
G:Binary Load / Directory			+			+	
H:Hex Conversion			+	+		+	
H:Hex Arithmetic			+			+	
I:Install Ramdisk Handlers			+	+	+		
J:Jump Subroutine (JSR)	+	+	+	+		+	
L:Drive Selection/Control	+	+	+	+		+	
M:Move Block of Memory			+			+	
N:Relocate 6502 Code			+			+	
O:Operate from Program Buffer			+			+	
P:Printer Control	+	+	+	+		+	
R:Read Sector(s) from Disk	+	+	+	+		+	
S:Search Memory for Sequence	+	+	+	+		+	
T:Toggle Hex/Char Display Mode	+	+	+	+		+	
U:User's Custom Command			+				
V:Verify 2 Blocks of Memory			+	+		+	
W:Write Sector(s) to Disk	+	+	+	+		+	
X:Disassemble Memory	+	+	+	+		+	
Y:Line Assembler			+			+	
Z:Exit Monitor			+				
Lockup Recovery			+	+		+	
Redirection of Printer I/O			+	+		+	
Talk to Happy Ram Buffer			+	+		+	
80 Column ATRMON for ATR8000							+

★ XMAS Pricing ★

Hardware:	LIST	XMAS Special
Standard OMNIMON! Piggyback Board	$99.95	$79.95
RAMROD-XL with OMNIMON-XL (800XL)	$119.95	$99.95
OMNIVIEW-XL ADD-ON (RAMROD-XL)	$59.95	$45.00

Enhancements:		
8K OMNIMON Enhancement	$45.00	$35.00
8K OMNIVIEW Enhancement	$45.00	$35.00
4K OMNIVIEW Enhancement	$30.00	$25.00
Letter Perfect or Data Perfect	$99.95	$70.00
AXLON 128K Ramdisk	$299.95	$250.00
BASIC-XL, MAC65, ACTION!	$99.95	$70.00

Newell RAMROD OS Board

This is a new operating system board which replaces the existing OS board. It allows you to use EPROMs in place of the ATARI OS ROMs and comes with an enhanced OS which includes additional graphics modes and a fast cursor. It also has a socket which will accept any version of OMNIMON and thus is an alternative to the OMNIMON! piggyback board. For the 800 only.

RAMROD OS Board with Standard OMNIMON	$149.95	$119.95
RAMROD OS Board with 8K OMNIMON or 8K VIEW	$189.95	$149.95
Same as above with Fastchip Floating Point Package	$209.95	$169.95
RAMROD OS Board with 8K OMNIMON and 4K VIEW	$209.95	$169.95
Fastchip Floating Point Package by itself	$29.95	$24.95

How To Order

Add $1.50 shipping ($3.00 for 2-day delivery). Add $1.65 for C.O.D. We accept checks, money orders, or credit cards (Visa or MC). COD must be payed with cash or MO.

CDY Consulting
421 Hanbee
Richardson, Tx 75080
(214) 235-2146

DEALER INQUIRES SOLICITED

(continued from page 6)

umn of its own—as assembly and FORTH enjoy now. I hope you will agree, and give this wonderful language its own place in the pages of **ANALOG**.

Sincerely,
David Brundage
Pratt, KS

We've run programs in Action! in the past, but—rather than a single-viewpoint column on the language—we've decided to print individual Action! articles from various authors, on a frequent basis.

Back in issues 17 and 18, we ran a two-part series entitled **Introduction to Action!**, *by the author of* Action!, *Clinton Parker. Unfortunately, he's been busy on other projects and, thus, unable to continue the articles.*

ANALOG Computing *has received dozens of terrific Action! games and utilities, which you'll see in these pages in the new year. Never fear, we'll keep up the Action! —L.P.*

Crash course.

First, I would like to congratulate the entire **ANALOG** staff for producing the best computer magazine each month. It's really great!

Second, I found out from a reasonable source (*Compute!* magazine) that Atari has fixed the "dreaded system crash." It was just a bug in the BASIC cartridge. Is this true? If it is, will we old folks, who have Atari 400s and 800s, get to buy a new BASIC cartridge?

Third, how much is the **ANALOG Compendium** on disk?

Bill Boroson
Gloversville, NY

There is certainly a bug within the BASIC cartridge that causes the system to lock up, and it can be fixed in one of two ways.

The first is to send $15.00 to Atari and request a new cartridge. The second is to wait a few months. We will be publishing an article that explains how to convert the cartridge BASIC to an AUTORUN.SYS file and correct the bugs. We can legally do this, because you will need the original cartridge for the program to work. Believe me, it will be well worth it!

A set of disks for **Compendium** *programs (three double-sided disks) costs $35.00. —C.B.*

TV blues.

I've been using an Atari 800XL computer for a while now, and before that, the older Atari 800. We had them both hooked up to the TV at the same time, for a while there.

Sometimes when I want to use the computer, my sister wants to watch TV. . . so I'm wondering if there is a POKE that will allow you to use a printer as display screen instead of a monitor or TV—like a teletype machine or something of that nature. Do you know a POKE or set of POKEs that will do that?

William M. Shaw
Kenai, AK

As far as we know, there is no simple POKE that will let the printer behave like a teletype machine. There are several reasons why this will not work.

The built-in screen editor will send a line to BASIC only after you have hit the RETURN key on your computer. Also, the printer will print only when it receives a full line of information from your computer.

Basically, what this means is that, even if you could set your computer and printer up to do this, you wouldn't see what you have typed until after you hit the RETURN key. Maybe you should start saving up for a cheap little black and white TV set.

—C.B.

Controlling changes.

I use an Olivetti **Spark Jet** printer and **Ape-Face** interface with my 800XL. The combination is entirely satisfactory—with certain annoying drawbacks.

When using my **AtariWriter**, I cannot cue the printer to underline or change the number of characters per inch. Both my **AtariWriter** and the printer have the capability to do these functions separately, but not together.

I have tried to use the CTRL-O command to enter the appropriate binary printer code—the printer doesn't understand the CTRL-O! Is there a way to enter a program in conjunction with the **AtariWriter** that would make the printer understand my intentions?

Also, what has happened to the Atari support programs? I've tried both 800 numbers for the Atari Program Exchange (APX) and for customer service, and found that the lines have both been disconnected!

Thanks for your help.
Mike Johnson
Evanston, IL

*Your printer doesn't have to understand a CTRL-O code, but the **AtariWriter** must. CTRL-O is a signal for the word processor that the numbers following should be changed to a control code. This is similar to using the CHR$ function in BASIC!*

*Since I'm not familiar with codes used by the **Spark Jet** printer, we'll do an example with fictitious codes. Let's say that we want to send two bytes to the printer. In hex, they would be $1B and $41 (ESC A). The decimal equivalents would be 27 and 61. So, we would enter:*

O27O61

where inverse O means that you must hold down the control key and type the letter O at the same time. This should work.

APX, with its 800 numbers, was dropped when management of Atari was shifted over from Warner Communications to Jack Tramiel. Atari, Inc. is now known as Atari Corporation.

—C.B.

D:CHECK problem.

I'm writing you because I'm having a problem with **D:CHECK2**. During the time that the screen goes blank, the program stops. Is this a problem typical with Atari XL computers?

I would also like to know if a plotter utility such as **Solid States** (issue 16) can be converted for use with an Olivetti PR2300 printer/plotter, and, if it can, how do I go about doing so?

Sincerely,
Ben Luijt
Los Angeles, CA

*We have yet to receive any negative reports about our **D:CHECK2** program. It works with both the old 400s and 800s, as well as the newer XL models.*

There has probably been a mistake made when typing the program in, or there might even be a problem with the BASIC that's built into your machine!

*When the XLs were first shipped, it was found that some of the ROM chips were defective. Has any other program failed to work for you? If not, I would recheck spelling. CHECKSUM DATA was included along with **D:CHECK2**, but you can't use it unless the program works. . .and you can't be sure that the program works unless you use that CHECKSUM DATA.*

*You might also try using the new **Unicheck** program that appeared in issue 24. This is a machine language version of **D:CHECK2** that is a lot easier to use. Try it; I'm sure you'll like it.*

*As for the Olivetti PR2300 printer/plotter being used with **Solid States**, we just don't know, since we have no information here on the unit.*

—C.B.

Reader Comment

continued on page 48

New. . . Improved
WHAT IS CHECKSUM DATA?

Most program listings in **ANALOG Computing** are followed by a table of numbers appearing as DATA statements, called "CHECKSUM DATA." These numbers are to be used in conjunction with **D:CHECK** and **C:CHECK** (which appeared in **ANALOG Computing** issue 16 and the **ANALOG Compendium**) or with **UNICHECK** (from issue 24).

D:CHECK and **C:CHECK** (written by Istvan Mohos and Tom Hudson) and **UNICHECK** (by Tom Hudson) are designed to find and correct typing errors when readers are entering programs from the magazine. For those readers who would like copies of these articles, you may send for back issue 16 or 24 ($4.00 each) or the **ANALOG Compendium** ($14.95 plus $2.00 shipping and handling from:

ANALOG Computing
P.O. Box 615
Holmes, PA 19045

Some program listings reproduced in ANALOG may contain "strange" characters not shown on the Atari keyboard. These are special characters which use the CTRL, ESC and "ATARI LOGO" (INVERSE) keys. Shown below is a list of these characters and the keystrokes used to get them. □

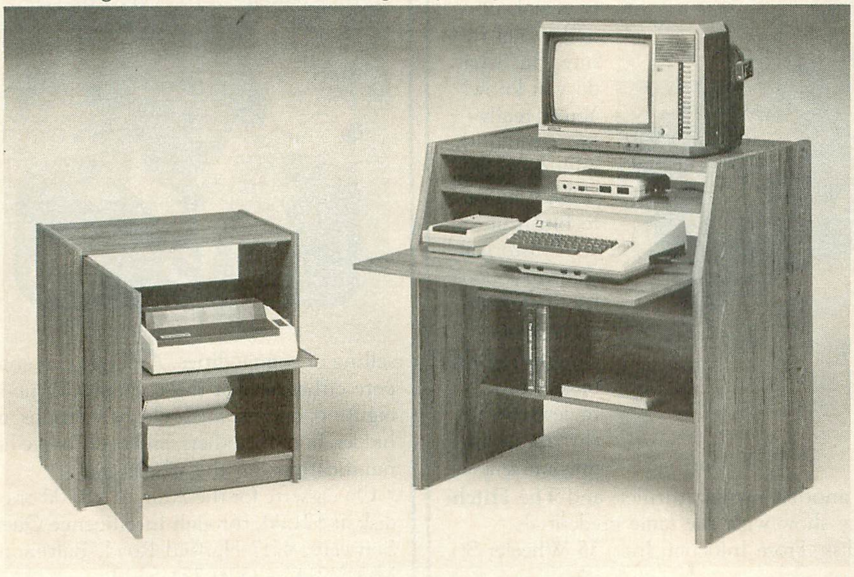
```
--- CTRL ,
--- CTRL A
--- CTRL B
--- CTRL C
--- CTRL D
--- CTRL E
--- CTRL F
--- CTRL G
--- CTRL H
--- CTRL I
--- CTRL J
--- CTRL K
--- CTRL L
--- CTRL M
--- CTRL N
--- CTRL O
--- CTRL P
--- CTRL Q
--- CTRL R
--- CTRL S
--- CTRL T
--- CTRL U
--- CTRL V
--- CTRL W
--- CTRL X
--- CTRL Y
--- CTRL Z
--- ESC ESC
--- ESC CTRL UP-ARROW
--- ESC CTRL DOWN-ARROW
--- ESC CTRL LEFT-ARROW
--- ESC CTRL RIGHT-ARROW
--- CTRL .
--- CTRL ;
--- ESC SHIFT CLEAR
--- ESC BACK S
--- ESC TAB
--- INVERSE CTRL ,
--- INVERSE CTRL A
--- INVERSE CTRL B
--- INVERSE CTRL C
--- INVERSE CTRL D
--- INVERSE CTRL E
--- INVERSE CTRL F
--- INVERSE CTRL G
--- INVERSE CTRL H
--- INVERSE CTRL I
--- INVERSE CTRL J
--- INVERSE CTRL K
--- INVERSE CTRL L
--- INVERSE CTRL M
--- INVERSE CTRL N
--- INVERSE CTRL O
--- INVERSE CTRL P
--- INVERSE CTRL Q
--- INVERSE CTRL R
--- INVERSE CTRL S
--- INVERSE CTRL T
--- INVERSE CTRL U
--- INVERSE CTRL V
--- INVERSE CTRL W
--- INVERSE CTRL X
--- INVERSE CTRL Y
--- INVERSE CTRL Z
--- ESC DELETE
--- ESC INSERT
--- ESC CTRL TAB (CLR)
--- ESC SHIFT TAB (SET)
--- INVERSE SPACE
--- INVERSE _
--- INVERSE CTRL .
--- INVERSE CTRL ;
--- INVERSE |
--- ESC CTRL 2
--- ESC CTRL BACK S
--- ESC CTRL INSERT
```

NEW PRODUCTS

by Lee Pappas

THE SMART DESK

This compact desk with 19 square feet of workspace utilizes a drop-leaf front for protected storage. The **Smart Desk** was originally designed for the Atari computer line and,



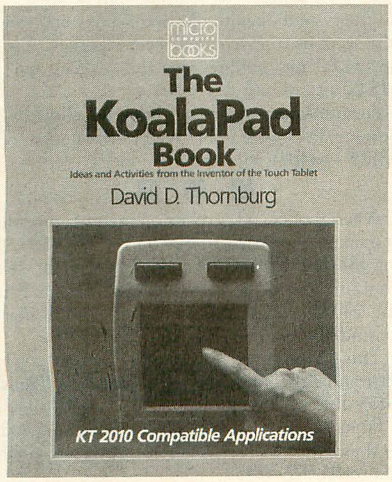
along with your computer, it will store a cassette, modem, disk drives, monitor, and other peripherals. Your computer's console slides onto the drop-leaf front at a typing height of 26 inches. The shelf in the center is adjustable, while the top shelf is fixed and will hold a monitor up to 19 inches or, with a smaller monitor, an 80-column printer. The **Smart Stand**, a matching printer stand, is also available.

Smart Desk costs $199.00; **Smart Stand**, $149.00, from The Furniture Byte, Inc., P.O. Box 1757, Longview, WA 98632 — (800) 426-5301.

THE KOALAPAD BOOK

The inventor of the touch tablet, David Thornburg, has now written a book on various uses of the **KoalaPad**—more than creating pretty pictures. For instance, chapter 4 describes some software products compatible with **KoalaPad**: **Spider Eater**, **Music Construction Set** and **Graphics Exhibitor**. Much of the book covers the Atari specifically.

The KoalaPad Book is priced at $12.95, 134 pages, from Addison-Wesley Publishing.



DESIGN YOUR OWN GAMES

Atari Graphics and Arcade Game Design is a hefty, 478-page book designed to teach you the building blocks of Atari graphics. Covered extensively are basic player/missile graphics, GTIA colors, scrolling, vertical blank and display list interrupts, animation and sound. Much of the book is devoted to

game design, with graphics and flowcharts evident throughout. Game particulars—bomb drops, laser fire, ship movement and missile firing —are explained, along with the mathematics and physics needed to derive those objects. Both BASIC and assembly language are covered, making the book handy for Atari users of all ranks.

By Jeffrey Stanton with Dan Pinal, softbound, $16.95 from Arrays, Inc., The Book Division, 11223 S. Hindry Avenue., Los Angeles, CA 90045.

BUFFER YOUR PRINTER

Digital Devices' latest release is the **Model P16 Printer Buffer**. Starting at $119.95, this unit contains 16K of buffer memory and is expandable to 64K, by the addition of more chips.

A printer buffer temporarily stores the data you've sent to the printer, thus freeing your computer from having to wait for your printer to stop. This low-cost buffer comes with a Centronics parallel connector, allowing compatibility with most printers—such as CItoh, Epson, NEC and others. RS232 models will also be available shortly.

For more information, contact: Digital Devices, 430 Tenth Street, Suite N205, Atlanta, GA 30318 — (404) 872-4430.

INFOCOM'S NEW ADVENTURE — NEW PACKAGING

Cutthroats by Infocom is presented in a striking, newly-designed box. Touted as "book-like," the more compact format is easier for retailers to display—and for consumers to store.

A bound-in booklet that the consumer can examine before purchasing gives details and a preview of the program, along with sample dialogue. The entire current and future Infocom adventure line is scheduled for this format.

The first release to reside in this new home is **Cutthroats**, and this time you're a "diver for hire," in search of underwater treasure. But what do you know? You're really just a run of the mill Atari user.

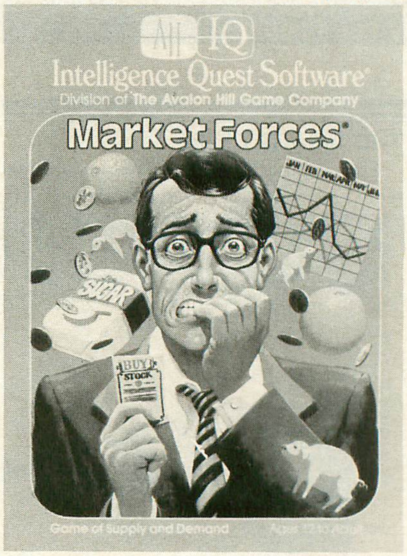
Well, be sure to brush up on your historical shipwreck booklet (included). **ANALOG Computing** will have a full review of **Cutthroats** in an upcoming issue.

Other new Infocom releases include **Suspect**, another mystery/thriller; and **The Hitchhiker's Guide to the Galaxy**, based on the crazy show with the same credentials.

Priced at $34.95, the adventures are for 48K disk. From Infocom, Inc., 55 Wheeler St., Cambridge, MA 02138 — (617) 492-1031.

MARKET FORCES

. . . is a new program designed to teach the inner workings of the economy. Set up as a stock-market economic environment, **Market Forces** simulates the buying and



selling of commodities. A player can compete either against the computer or up to eight opponents. Each player's actions are hidden from the others, and every move has ramifications on the whole game.

On cassette for the Atari at $16.00 or on disk at $21.00, through Intelligence Quest Software, 4517 Harford Road, Baltimore, MD 21214 — (301) 254-5300.

COMPUTER TRIVIA

What's really different about **PQ—The Party Quiz Game** is its hand-held controllers. The Atari-compatible version allows up to four players to compete *sans* keyboard (which is only used to pause the game). Six-foot cords connect to the computer joystick ports.

The game itself features over 2500 questions on disk, randomly chosen by the computer (even the categories). Players can choose the length of the game, the time allotted for the questions' responses and the number of players.

Other selections include: the competitive mode, where your first correct answer will win points; the social option, in which all correct responses gain you points; and the team scoring option, where players double up.

Additional disks will be released shortly, including **General Edition II**, covering entertainment, history, art, sports and literature. **General Edition III** will follow, with science, sports, music, geography and world records. The **Education Edition I** is also scheduled.

The cost is $69.95 from Suncom, 260 Holbrook Drive, Wheeling, IL 60090 — (312) 459-8000.

OTHER NEWS

Adventure International has signed an agreement with 20th Century Fox, involving ***The Adventures of Buckaroo Banzai across the 8th Dimension***. AI has announced that it will release at least two software products, one an adventure and the other an educational game.

Contact: Adventure International, Box 3435, Longwood, FL 32750 — (305) 862-6917.

■ ■ ■

If your Atari is used for practical applications, you now have the opportunity to access a **Yellow Page** service of over 6 million businesses in the U.S., using a modem and phone line.

The system operates 24 hours a day and takes just seconds to give the answer you seek. Costs are $1.00 per minute plus the hookup fee of $15.00 per month.

For information contact: Instant Yellow Page Service, P.O. Box 27347, Dept. N, Omaha, NE 68127 — (402) 331-7169.

■ ■ ■

A series of 600XL expansion memory modules is available, starting at $79.95 for 32K, $99.95 for 48K and $119.95 for 64K. These small, self-contained units plug into the expansion port in the 600XL's back and require only ¾" of space behind the computer.

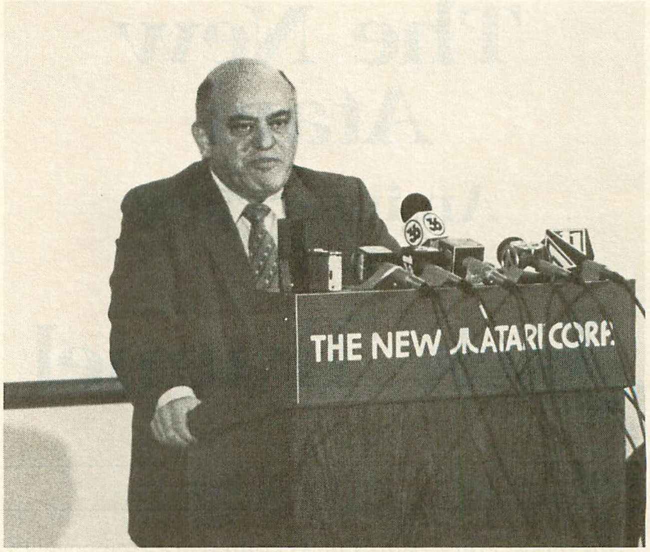
The **AM64** is fully compatible with most cassette and disk software, as well as the Atari **Translator** disk.

From RC Systems, Inc., 121 W. Winesap Road, Bothell, WA 98012 — (800) 227-1617; in California (800) 772-3545.

ATARI 1985

Atari Corporation's first press conference

by Jon A. Bell



Jack Tramiel, Chairman of the Board, Atari Corp.: "I want to make this business exciting again."

On Tuesday morning, November 13, 1984, Atari Corp. held a press conference in which they outlined their basic marketing strategy for 1985. A small army of newspaper, television and magazine reporters descended upon Atari's administrative headquarters, a dark brown building at 1265 Borregas Avenue. **ANALOG Computing** publisher Lee Pappas and I were in attendance at this press conference and were able to provide these two "scoops"—one of the first summaries of the press conference to be published in any monthly magazine, as well as an interview with Jack Tramiel, Chairman of the Board (see page 14).

At the meeting, Vice President of Marketing James Copland announced the news: price reductions on the 800XL computers, the 1050 disk drive, the 1010 cassette recorder and the 1025 and 1027 printers. The 800XL has had its price slashed from $179.00 retail to under $120.00. Exact prices on the Atari peripherals were not set, but will be reduced to "bring them more into line with that of the 800XL price point."

Producing the Atari hardware are two plants: one in Taiwan employing 1200 people, manufacturing products for North America; and one in Ireland employing 350 persons, manufacturing for the European market. This brings the total number of Atari employees to around 3000—less than half those employed two years ago, in Atari's more wasteful days.

Software prices are also going to be reduced, with no piece of Atari software to be sold "for more than $49.00." As stated in the press release, "these cost reductions now make it possible for the consumer to purchase a complete Atari computer system for well under $600.00."

To support these new products, Atari Corp. started a media blitz on Thursday, November 15—in newsprint. (Mr. Copland explains the differences between television and print advertising and marketing in our following interview with Mr. Tramiel.) No decision has been made yet on whether or not Alan Alda will continue as Atari's spokesman. However, when asked about Alda's future, Mr. Tramiel replied, "We are selling computers, not people." The theme of the Christmas ads, appearing in newspapers across the entire country, is for the personal computer "so affordable even Scrooge would have given it!"

Also announced at the meeting were Atari's plans to produce high-quality, inexpensive 16- and 32-bit machines for home use. According to Mr. Tramiel, these new high-end machines will be targeted to individual users. He emphasized that he would not try to compete with IBM and the business market, and the machines would not be IBM-compatible. Instead, Atari plans to use an operating system developed by the Monterey-based Digital Research, where industry insiders had reported seeing Atari technical personnel working with Digital software developers. Mr. Tramiel said he hoped the 16-bit machine would be ready in time for showing at the January CES, and that the 32-bit machine would be introduced in Hanover, West Germany in April.

Stay tuned for further details in the next issue of **ANALOG Computing**. ☐

The New Atari

An interview with

Jack Tramiel

by Lee H. Pappas and Jon A. Bell

Ed. note: After Atari's press conference (see page 13) and a brunch for press members, **ANALOG Computing** *publisher Lee Pappas and I were granted a short interview with Jack Tramiel, Chairman of the Board of the "new" Atari Corp. Also in attendance at the interview was James L. Copland, Vice President of Marketing for Atari Corp.*

AN.C: *In what directions will you be taking Atari—similar to or different from those of the previous administration? For instance, under Warner Communications, Atari was emphasizing video games over computers. It also suffered from the three divisions competing with each other—the arcade division, the home computer division and the home games division. What will you emphasize, home computers or games?*

JT: We will be emphasizing every product we manufacture, and we will have *one* company, not three companies.

AN.C: *Obviously, the latter was one of the major problems that the "old" Atari had. As stated in the press conference, Atari is going to be lowering the prices on both the 2600 game system and the 800XL. We were wondering what plans, if any, you had for releasing the 7800 game system.*

JT: Right now, we do not plan to release the 7800.

AN.C: *Do you plan to use any of the technology of the 7800 in new Atari products?*

JT: Could be. . . At the present time, we do not have any plans for it.

AN.C: *In terms of marketing, how are you going to sell your computers? Will you continue to court mass retailers, or try to distribute your products in computer stores?*

JT: We will definitely sell to anyone who pays their bills—if it's a mass marketer, or if it's a retailer. Our "star" is our product, the way James (Copland) mentioned before *(in the press conference—Ed.)*. From a marketing point of view, all our effort is going into the product, to tell the retailer what the product is. And I believe very much in the separation of religion and state; it's the same thing with me. . . manufacturing and retailing. We're the manufacturer; they're the retailer. Our job is to make the best product; their job is to sell it.

AN.C: *What about patching up the problems caused by the previous Atari administration, by dealers and mass marketers who felt "burned" by Atari?*

JT: I believe that every single retailer whom we are dealing with. . . has known us for the past twenty-some years, and they're not looking at us as the prior Atari, but they're looking at us as the new Atari. We have a relation in the way we do business. We know each other, and we have a brand-new start.

AN.C: *Nevertheless, you might be fighting an uphill battle with those dealers who simply see the name Atari and feel bitter towards the company and the way it's treated them in the past.*

JT: The beauty about it is, the person who buys the computer—or buys the game—is not the retailer. It's the individual who is going to *use* it.

AN.C: *In the press conference, you mentioned the number of 2600s sold and the fact that those buyers are a target audience for Atari computers. People want to trade up from their machines; they're not simply content to play games. What about the Atari veterans, those people who have owned Atari 400s and 800s for years now? Are you going to encourage them to trade up to new, more powerful Atari machines?*

JT: It's a nice group. . . It's not a very big group, but it's a nice group. There weren't that many Atari computers sold. . . compared with the company with which I was previously involved. Yes, we are definitely aiming at anyone who has an Atari product; yes, we will be able to offer them more powerful products; and I hope that they will upgrade their products.

AN.C: *We are heavily involved with the Atari SIG on CompuServe, and we try to keep in touch with the entire Atari community. As you said, the group may not be very large, but they are vocal. They are also very influential to their friends, who look to them for guidance about what computer to buy. One of the things we've noticed is that the real diehards are very anxious about new Atari products, and they're hoping that Atari is going to come out with something exciting to trade up to. They've plumbed the depths of what their Atari can do, and they want something more powerful. . . but they're waiting. Their loyalty is intense. They are not buying an IBM PC; they are not buying an Apple IIc. . .*

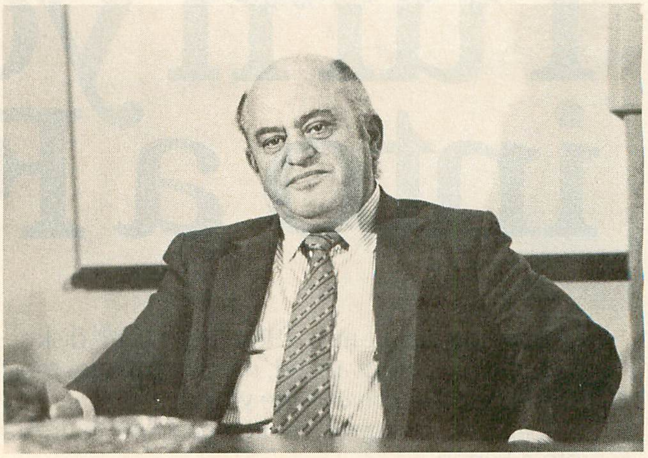
JT: Could they afford it?

AN.C: *Oh, yes. . . some of them, anyway.*

JT: Or are they just holding out?

(Laughter.)

We have to be proud of what we are buying and what we are selling.



AN.C: *We're talking about hackers here. . . the kind who would rather buy a new machine than eat dinner tomorrow.*

JT: But they're still out three thousand dollars. They bought the Atari because it was less than the Apple.

AN.C: *They figure that Atari's going to bring out a machine that can do even more than the three-thousand-dollar machines, for a lot less than half the price.*

JT: No question about that. That's absolutely right.

AN.C: *The good thing about that is: the money they might spend on other products, they'll spend on Atari.*

JT: It'll be worth the wait.

AN.C: *That's what we tell them.*

JC: What we'll do is—we'll offer a new advertising campaign next year. . .and with every Atari computer they've waited so long to buy, we'll throw in a free meal. How's that?

(Laughter.)

JT: You people can help—in encouraging your readers, Atari owners. . . Everything we do, we have to do for the long term. We have to be proud of what we are buying and what we are selling.

AN.C: *Would you say that was your policy. . .when you were working for the people in Pennsylvania?*

JT: It was my policy the day I opened up my doors. It will be my policy until I die.

AN.C: *Getting back to marketing, we were wondering if there was a specific reason for emphasizing print ads over television. Is it for monetary reasons, or are you going after a specific audience with print ads?*

JC: There is a phobia associated with television advertising, because it's beautiful and it's full color, and you can see it moving—that it's the most cost-efficient buy for introducing or exposing a message to the average American consumer. The fastest and most direct way to get immediate household coverage in any market is by newsprint. Our campaign that breaks on Thursday, November 15 (1984) will hit in excess of 75 percent of American households in the United States. The cost to run a single campaign on a national basis in this country, in print, will be less expensive than one 30-second commercial on this year's Super Bowl—in fact, almost 50 percent less.

AN.C: *That's an interesting comparison. The majority of the hype over Apple's incredibly expensive 1984 commercial was in print. That's why the ad was successful. It wasn't that so many people actually saw it, but the amount of paper that it generated.*

JT: There's no question. Again, one company believes that the end user is intelligent and knows what he's buying; the other one believes that he should get snowballed with marketing, with advertising.

AN.C: *They have to be bludgeoned into buying the products.*

(Laughter.)

JT: Intelligent companies are going after the long term —not to cheat the customer, not to be greedy. Our competitors are out there to make the buck right now. *(Tramiel's fist came down on the tabletop in cadence with the last phrase, for emphasis—Ed.)* Spend the money for advertising, have the image! You can't compute with image; you have to compute with the computer. Image you have to *earn*, not advertise.

AN.C: *That's what we've always said. . .*

(Laughter.)

A lot of people ask us, "What's the best inexpensive computer to buy?" We say it's Atari. And they'll say, "Well, you would say that, because you publish an Atari magazine." We reply, "No. We publish an Atari magazine because we like the computer and think it's the best computer for the money." No one works at **ANALOG Computing** *who isn't excited about what we are trying to promote. Our magazine has gotten where it is because of the spirit of the company, not hype.*

JT: I can assure you. . .You didn't ask me this question, but I will tell you. We will support the people who own Atari products—with everything we have. And we will make people like you, and users, proud of owning our products. That is our aim; that is the satisfaction I'm looking for. I'm definitely not doing it now for money. I'm doing it to create something, to insure that we have the best computer in the marketplace, for less. □

Turn your Atari into a Ferrari.

Introducing the all-new 1984 Indus GT™ disk drive. The most advanced, most complete, most handsome disk drive in the world.

A flick of its "Power" switch can turn your Atari into a Ferrari.

Looks like a Ferrari.
The Indus GT is only 2.65" high. But under its front-loading front end is slimline engineering with a distinctive European-Gran flair.

Touch its LED-lit CommandPost™ function control AccuTouch™ buttons. Marvel at how responsive it makes every Atari home computer.

Drives like a Rolls.
Nestled into its soundproofed chassis is the quietest and most powerful disk drive power system money can buy. At top speed, it's virtually unhearable. Whisper quiet.

Flat out, the GT will drive your Atari track-to-track 0-39 in less than one second. And when you shift into SynchroMesh DataTransfer,™ you'll increase your Atari's baud rate an incredible 400%. (Faster than any other Atari system drive.)

And, included as standard equipment, each comes with the exclusive GT DrivingSystem™ of software programs. World-class word processing is a breeze with the GT Estate WordProcessor.™ And your dealer will describe the two additional programs that allow GT owners to accelerate their computer driving skills.

Also, the 1984 Indus GT is covered with the GT PortaCase.™ A stylish case that conveniently doubles as a 80-disk storage file.

Parks like a Beetle.
The GT's small, sleek, condensed size makes it easy to park.

And its low $449 price makes it easy to buy.

So see and test drive the incredible new 1984 Indus GT at your nearest computer dealer soon.

The drive will be well worth it.

INDUS™

The all-new 1984 Indus GT Disk Drive.
The most advanced, most handsome disk drive in the world.

For additional information, call 1-800-33-INDUS. In California, call 1-800-54-INDUS. 213/882-9600.

CIRCLE #132 ON READER SERVICE CARD

PRINTERS

Alphacom 40C w/Interface 99.95
Alphacom 80C w/Interface . . . 189.95
Axiom AT-550 . . 279.00
Epson . . . . Call
Prowriter I . . . . Call
Riteman . . . . Call
Silver Reed . . . . Call
Toshiba 1340 . . . . Call
Toshiba 1351 . . . . . Call
Silver Reed . . . . . . Call

ATARI MODEM SPECIAL

Hayes 300 & R-Verter 239
Hayes 1200 & R-Verter . . . . Call

No additional shipping for Modem packages in Cont. USA

SUPERPRINTER PACKAGES

Gemini 10X and Apeface 323
Prowriter and Apeface 409
Legend 880 and Apeface 308

No additional ship. charges on printer packages in Cont. USA

MOSAIC

48K RAM 94.00
64K RAM/400 . . 149.00
64K RAM/800 + Cable Kit #1 . . 169.00
64K Expander for 600 XL 99.95

Bring the trivia craze home with **P.Q. The Party Quiz Game** for the Atari 800 & 800XL (disk only) . . . . . . Call

ATARI

ATARI is a trademark of ATARI, INC.

Atari Inc. has cut all hardware and software prices plèase call for latest prices!!!

star

micronics • inc

THE POWER BEHIND THE PRINTED WORD.

Gemini 10X . . 249
Gemini 15X . . 389
Delta 10X 369
Delta 15X 499
Radix 10X 549
Radix 15X 629
Powertype 329

DISK DRIVES

Indus GT . . . . Call
Percom . . . . Call
Astra 2001 . . . . Call

MODEMS

Hayes Amart Modem 300 . . . . Call
Mark II 79.00
Mark VII/Auto Ans/ Auto Dial . . . . Call
Mark XII/1200 Baud . . . Call
MPP 1000 C . . . . Call
R-Verter Modem Adaptor 39.95
Prometheus . . . . Call

INTERFACES

Aid interfast I . . . . Call
Ape Face . . . . Call
R-Verter Modem Adaptor 39.95
MPP 1150 . . . . Call

☆ Call for prices on joysticks, printer cables, blank floppy disks, and other computer accessories. ☆

A T A R I S O F T W A R E

ACCESSORIES

Ape - Link 29.95
Gemini 10X 8K Upgrade . . . . Call
Koala Pad - D 69.95
Koala Pad - Cart 74.95
Humpty Dump - D 29.95
Monitors . . . . Call
Compuserve Starter 27.95
Vidtex 29.95
EIS Subscription Kit 64.95
Alien Voice Box II - D 99.95
Analog Compendium 9.95
Atari Assembler 14.95
Compute's Machine Lang / Beg 12.95
Inside Atari Dos 19.95
Mapping the Atari 14.95
Compute's 1st - 3rd Books Atari - EA 12.95
Printer Stand 15.95
Omnimon 82.95
Omniview 80 . . . . 39.95
Printer Ribbons . . . . Call
Ramrod XL . . . . Call
WICO Joysticks . . . . Call
MPP 64K Printer Buffer . . . . . Call
U-Print 16, 32, or 64K Buffer . Call
TAC III Joystick 12.95
Starfighter Joystick 9.95
Ramrod XL 99.95
Muppet Keys (XL Only) - D . 54.95

ACTIVISION

Decatholon - Cart 29.95
Beamrider - Cart 29.95
Hero - Cart 29.95
Pitfall II - Cart 29.95
River Raid - Cart 29.95
Designer Pencil - Cart 29.95
Space Shuttle - 29.95
Lone Ranger - Cart 29.95
Zenji - Cart 29.95

AVALON HILL

Call for items and prices'

BRODERBUND

Bank Street Writer - D 49.95
Loderunner - D 23.95
Mask of the Sun - D 27.95
Operation Whirlwind - D . . . 27.95
Spelunker - D 20.95
Stelth - D 20.95
Whistler's Brother 20.95
Print Shop - D 34.95
Serpent's Star - D 27.95

CBS SOFTWARE

Call for items and prices

DATASOFT

Bruce Lee - D / T 23.95
Micropainter - D 23.95
Lost Tomb - D / T 23.95
Mancopter - D / T 23.95
Meridan III - D / T 27.95
Letter Wizard + Spell - D . . 54.95
Conan the Barbarian - D / T . 27.95
Mr. Do - D / T 27.95

DISKETTES

Dysan . . . . Call
Verbatim . . . . For
Certron . . . . 10 pak
Elephant . . . . &
Maxell . . . . Quantity
Memorex . . . . Pricing.
Ultra Magnetics . . . .
BASF . . . .

ELECTRONIC ARTS

Archon - D 29.95
Pinball Construction - D . . . 29.95
M.U.L.E. - D 29.95
Murder / Zinderneuf - D 29.95
One on One - D 29.95
Archon II - D 29.95
Financial Cookbook - D 37.95
Music Construction - D 29.95
Realm/Impossibility - D 29.95

EPYX

Call for items and prices

INFOCOM

Deadline - D 29.95
Enchanter - D 23.95
Infidel - D 29.95
Planetfall - D 24.95
Sorcerer - D 29.95
Starcross - D 29.95
Suspended - D 29.95
Witness - D 29.95
Sea Stalker - D 24.95
Cutthroats - D 24.95
Suspect - D 24.95
Hitchiker - D 24.95
Zork I - D 24.95
Zork II or III - D 27.95

MICROPROSE

Chopper Rescue - D / T 23.95
Floyd / Jungle - D / T 23.95
Hellcat Ace - D / T 23.95
Mig Alley Ace - D / T 23.95
NATO Commander - D / T . . 23.95
Solo Flight - D / T 23.95
Spitfire Ace - D / T 23.95
Wingman - D / T 23.95
Air Rescue I - D / T 23.95
Challenger - D / T 23.95
F-15 Strike Eagle - D / T 23.95

MISCELLANEOUS ATARI

Diskey - D 34.95
Ultra Disassembler - D 34.95
Codewriter - D 69.95
Star League Baseball D / T 23.95
Star Bowl Football - D / T . . 23.95
Dino Eggs - D 27.95
Master Type - D / Cart 27.95
Flight Simulator II - D 37.95
S.A.M. - D 41.95
Castle Wolfenstein - D 20.95
Compuserve Starter Kit 27.95
Home Accountant - D 49.95
Megafont II - D 19.95
Monkey Wrench II - C 37.95
Movie Maker - D 37.95
Ultima III - D 41.95
Jupiter Mission - D 34.95
Boulder Dash - D / T 20.95
Scraper Caper - Cart 34.95
Miner 2049'er - Cart 34.95
Spy Hunter - Cart 29.95
Tapper - Cart 29.95
Up 'N Down - Cart 29.95
Beach Landing - D 27.95
Pic Builder - D 27.95
Astrochase - D 20.95
Flip - Flop - D / T 20.95
Sargon II - D / T 16.95
Odesta Chess - D 49.95
Millionaire - D 27.95
Spy vs. Spy - D 23.95
Adventure Writer -D 41.95
MMG Basic Compiler - D . . 69.95
Summer Games - D 27.95
Pitstop II - D 27.95
Gateway to Apshai - Cart . . 27.95
Montezuma's Revenge - D . 29.95

MISC. ATARI (cont'd.)

Dragon/Pern - D 27.95
Adventure Master - D 34.95
Get Rich Series - D 34.95
Q-Bert - Cart 34.95
MPP Modern Driver - D 19.95
Microfiler - Cart 34.95
Microcheck - D 34.95
Mr. Do's Castle - Cart 34.95
Frogger II - Cart 34.95
Net Worth - D 54.95
Stickybear - D 27.95
Fischer Price - Cart 19.95
Windham Classics - D 19.95
Omnitrend Universe - D 69.95
Space Beagle - D 23.95
Adventure Writer - D 41.95
Beachhead - D 23.95
Letter Perfect / Spell - D . . . 74.95
Harcourt / Bruce S.A.T. - . . 59.95
S.A.G.E. Graphics Editor - D 39.95
Strip Poker - D 23.95
Millionaire - D 27.95
Scroll of Abadon - D 23.95

OPTIMIZED SYSTEMS

Action - Cart 69.95
Basic XL - Cart 69.95
MAC/65 - Cart 69.95
MAC/65 Tool Kit - D 27.95
Action Tool Kit - D 27.95
DOS XL - D 27.95
Action Aid - D 27.95
C65 - D 59.95
Handy - Writer - D . . . . Call
Postal Tool - D 41.95
Print Tool - D 41.95

SIERRA ON-LINE

Dark Crystal - D 27.95
Homeword - D 49.95
Quest For Tires - D 23.95
Ultima II - D 41.95
Ultima I - D 23.95

SPINNAKER

Adventure Creator - D 22.95
Aerobics - D 27.95
All in the Color Caves - C . . 22.95
Alphabet Zoo - Cart 22.95
Delta Drawing - Cart 22.95

SPINNAKER (cont'd.)

Pacemaker - Cart 22.95
Fraction Fever - Cart 22.95
Grandma's House - D 19.95
Kids on Keys - Cart 22.95
Kindercomp - Cart 22.95
Search / Amazing Thing - D 22.95
Snooper #1 - D 22.95
Snooper #2 - D 22.95
Story Machine - Cart 22.95
Trains - D 22.95

SSI

Carrier Force - D 41.95
Combat Leader - D/T 27.95
Cosmic Balance II - D 27.95
Cosmic Balance - D 27.95
Broadsides - D 27.95
War in Russia - D 55.95
50 Mission Crush - D 27.95
Questron - D 34.95
Rails West - D 27.95
Bomb Alley - D 41.95
Computer Ambush - D 41.95
Galactic Adventures - D 41.95
Computer Baseball - D 27.95
Reforger 88 - D 41.95
Objective Kursk - D 27.95
Breakthrough / Ardennes - D 41.95
Field of Fire - D 27.95
Imperial Galactium - D 27.95

SYNAPSE

Air Support - D/T 23.95
Alley Cat - D/T 16.95
Blue Max - D/T 23.95
Dimension X - D/T 23.95
Drelbs - D/T 23.95
Electrician - D/T 23.95
Encounter - D 16.95
Fort Apocalypse - D/T 23.95
Necromancer - D/T 23.95
New York City - D/T 23.95
Pharoah's Curse - D/T 23.95
Quasimodo - D/T 23.95
Rainbow Walker - D/T 23.95
Relax Stress Reduction Sys. . 79.95
Shamus Case II - D/T 23.95
Zepplin - D/T 23.95
Synfile - D 49.95
Syncalc - D 49.95
Syntrend - D 49.95
Synchron - D 27.95
Syncomm - D 27.95
Synstock - D 27.95

D-Disk T-Cassette Cart-Cartridge

To Order Call Toll Free

800-558-0003

For Technical Info, Order Inquiries, or for Wisc. Orders -

414-351-2007

NEW LOWER PRICES

ComputAbility™

NEW PRODUCTS

NO SURCHARGE FOR MASTERCARD OR VISA

Est. 1982

ORDERING INFORMATION. Please specify system. For fast delivery send cashier's check, money order or direct bank transfers. Personal and company checks allow 2 weeks to clear. Charges for COD are $3.00. School Purchase Orders welcome. In CONTINENTAL USA, include $3.00 shipping per software order. Include 3% shipping on all Hardware orders, minimum $3.00. Mastercard & Visa please include card # and expiration date. WI residents please add 5% sales tax. HI, AK, FPO, APO, Canadian orders — add 5% shipping, minimum $5.00. All other foreign orders, please add 15% shipping, minimum $10.00. All goods are new and include factory warranty. Due to our low prices, all sales are final. All defective returns must have a return authorization number. Please call 414-351-2007 to obtain an RA# or your return will NOT be accepted for replacement or repair. Prices and availability are subject to change without notice.

COMPUTABILITY
P.O. Box 17882
Milwaukee, WI 53217

ORDER LINES OPEN
Mon-Fri 11 AM - 7 PM CST
Sat 12 PM - 5 PM CST

Ask Mr. Forth

by Donald Forbes

To demonstrate FORTH on the Atari, you'll want to stress the best features of each. Atari's superb graphics came to the forefront again when *Computer Animation Primer* at last hit the bookstores. Those who bought Atari for its graphics capabilities—then waited two years for this work by David Fox and Mitchell Waite—were well rewarded for their patience.

The first half of the book gives a vivid look at state-of-the-art computer-generated animated movies. The second half is all Atari: animation with character sets and color registers, player/missile graphics, scrolling and display lists, and more than fifty pages of Atari BASIC programs. The authors chose to work solely with the Atari computer because of its unique potential for graphics and animation.

The color registers on the Atari, as they put it, give programmers amazing animation capabilities, even in BASIC. "Unbeknownst to many people, the Atari 400/800 contains color-mapping hardware (called color registers), and this feature alone gives it awesome capabilities when compared to its competitors."

The book culminates in the Great Movie Cartoon—a scrolling street scene with five layers of depth, plus cars and strolling people in the foreground.

So where does this put FORTH?

FORTH stands at the forefront of the second generation of computer software—where the user writes his own software, instead of going through a middleman. Elimination of the GOTO construct (with its resulting spaghetti code) and the subsequent realization that all programs could be constructed with three structures (sequential, two-way branch and "do while") led to top-down design for large systems. Users could begin to exercise control.

FORTH, although it first achieved prominence in the days before cheap megabyte memories and nanosecond cycles as "an operating system for crippled micros," is moving with the new trend, along with LOGO in the educational field, and UNIX and the C language on the mainframes and minis. FORTH, with no GOTO, is ideally suited for structured programming, top-down program design and bottom-up program compilation. The user can build upon past software to solve new problems as they arise.

Back to the introductions.

The easy way for you to introduce Atari users to FORTH is via the chapter called Color Graphics and Sound in Bob Albrecht's *Atari BASIC*, which probably came wrapped in the same box with their Atari computer—and certainly belongs on the bookshelf of every Atari owner.

The Italians have a saying, *"Tradutore, traditore,"* that the translator is a traitor. But many BASIC statements have straightforward FORTH counterparts.

If you organize a demonstration for your user group, you might find a teammate who will display the BASIC code on one machine while you display the corresponding FORTH code on another.

The chapter on color graphics and sound provides a large number of illustrations. For example, *10 GR. 3* becomes *3 GR.* This short program:

```
10 GRAPHICS 3
20 PRINT "LINE 1"
30 PRINT "LINE 2"
40 PRINT "LINE 3"
50 PRINT "LINE 4"
60 GOTO 60
```

translates as follows:

```
: P285 ( for page 285 ) 3 GR.
  CR ." LINE 1" CR ." LINE 2"
  CR ." LINE 3" CR ." LINE 4" QUIT ;
```

The code *60 GOTO 60* gets rid of the *READY*, and *QUIT* gets rid of the *OK*. (Which brings to mind an oldie—BASIC to FORTH: "I'm READY! You're OK!")

This program plots a point:

```
: P286 3 GR. 2 5 PLOT
  ." This is plot position 2,5" ;
```

and this one plots four points:

```
: P287 3 GR. 1 COLOR
  5  5 PLOT 15  5 PLOT
  5 15 PLOT 15 15 PLOT ;
```

Remember, however, if you have **QS Forth**, or the **Extended fig-FORTH** written by Patrick L. Mullarky and once sold by APX, that both PLOT and DRAWTO take *COLOR* as their third input parameter. So, in this (and other) examples, you would have to use this code:

```
: P287 3 GR. 5 5 1 PLOT 15 5 1 PLOT
  5 15 1 PLOT 15 15 1 PLOT ;
```

This loop plots a line of points:

```
FOR I=0 TO 19 STEP 2:PLOT I,I:NEXT I
```

but in FORTH we must add 1 to the loop index:

```
: P289 3 GR. 1 COLOR 20 0 DO
  I I PLOT 2 +LOOP ;
```

This program displays a large X:

```
: P291 4 GR. 1 COLOR
  0 0 PLOT 78 39 DRAWTO
  78 0 PLOT 0 39 DRAWTO
  ." X marks the spot" ;
```

This program features an endless loop (until you hit an interrupt key) to replace *60 GOTO 60* and the use of *I* as the index for the loop:

```
: P292 4 GR. BEGIN 2 0 DO I COLOR
  0 40 DO 0 0 PLOT 78 I DRAWTO
  -13 +LOOP LOOP ?TERMINAL UNTIL ;
```

Here is the SETCOLOR statement:

```
: P293 3 GR. 1 COLOR 0 0 2 SETCOLOR
  0 3 PLOT 38 3 DRAWTO
  ." What color do you see?" ;
```

and here SETCOLOR uses the loop indexes, while we use an empty DO. . . LOOP to create a delay. Notice that *J* is a FORTH-79 word you may not have, which copies the third item of the return stack onto the parameter stack, and thus lets you access *the* index of the outer loop. It comes in handy for two-dimensional arrays.

```
: DELAY 30000 0 DO LOOP ;
: J >R >R >R R R# !
  >R >R >R R# @ ;
: P295 3 GR. 1 COLOR
  ." Watch my colors change"
  5 0 DO 16 0 DO J I 2 SETCOLOR
  5 5 PLOT 30 5 DRAWTO
  DELAY LOOP 2 +LOOP ;
```

Try this as a simple animation demo of a man approaching you.

```
: P296 3 8 DO I GR. 1 COLOR
  4 1 2 SETCOLOR
  5 5 PLOT 7 5 DRAWTO 6 6 PLOT
  6 10 DRAWTO 4 15 DRAWTO
  6 10 PLOT 8 15 DRAWTO 5 7 PLOT
  3 9 DRAWTO 7 7 PLOT 9 9 DRAWTO
  DELAY -2 +LOOP
  ." What a funny man!!!" ;
```

The next program draws rectangles in three sizes and shows how this code:

```
PLOT X-2,Y-2: DRAWTO X+2,Y-2
```

can be done, using FORTH constants:

```
80 CONSTANT X 48 CONSTANT Y
: P297 7 GR. 1 COLOR
  0 15 2 SETCOLOR X Y PLOT
  X 2 - Y 2 - PLOT
  X 2 + Y 2 - DRAWTO
  X 2 + Y 2 + DRAWTO
  X 2 - Y 2 + DRAWTO
  X 2 - Y 2 - DRAWTO
  X 4 - Y 4 - PLOT
  X 4 + Y 4 - DRAWTO
  X 4 + Y 4 + DRAWTO
  X 4 - Y 4 + DRAWTO
  X 4 - Y 4 - DRAWTO ;
```

We can combine three graphics modes in one program, as well as having the use of both variables and constants:

```
1 VARIABLE M 0 CONSTANT X 0 CONSTANT Y
: P298 BEGIN 1 ' M ! 3 8 DO I GR.
  1 COLOR 0 1 2 SETCOLOR
  80 M @ / ' X ! 48 M @ / ' Y !
  X Y PLOT
  X 2 - Y 2 - PLOT
  X 2 + Y 2 - DRAWTO
  X 2 + Y 2 + DRAWTO
  X 2 - Y 2 + DRAWTO
  X 2 - Y 2 - DRAWTO
  X 4 - Y 4 - PLOT
  X 4 + Y 4 - DRAWTO
  X 4 + Y 4 + DRAWTO
  X 4 - Y 4 + DRAWTO
  X 4 - Y 4 - DRAWTO
  DELAY M @ 2 * ' M !
  -2 +LOOP ?TERMINAL UNTIL ;
```

Here, incidentally, is a familiar screen from your *Atari BASIC Reference Manual.*

```
: P51 3 GR. 0 2 8 SETCOLOR 1 COLOR
  17 1 PLOT 17 10 DRAWTO
  9 18 DRAWTO 19 1 PLOT
  19 18 DRAWTO 20 1 PLOT
  20 18 DRAWTO 22 1 PLOT
  22 10 DRAWTO 30 18 DRAWTO
  1 752 C! CR 6 SPACES
  ." Atari Personal Computers"
  30000 0 DO LOOP QUIT ;
```

This brings us to the end of Bob Albrecht's section on graphics. As you can see, the FORTH version of the BASIC code produced relatively few surprises. The next section of the book explores Atari's four voice capabilities, providing us a chance to show off some new techniques.

Vocal FORTH.

First of all, you want a word to shut off the four voices at any time, just as *END* will do in BASIC.

```
: QUIET 4 0 DO I 0 0 0 SOUND LOOP ;
```

The **valFORTH** fix—for those of us who cannot remember that SOUND looks for channel, frequency, distortion and volume—is *CatFish Don't Vote*. This next program allows you to experiment with loudness by furnishing an integer from 0 to 15, and −5 to end it.

```
0 VARIABLE L
: INPUT# QUERY CR ?TERMINAL
  CR INTERPRET ;
: P299 BEGIN INPUT# L ! L @ -5 =
  NOT IF 0 115 10 L @ SOUND ELSE
  0 0 0 0 SOUND QUIT THEN
  ?TERMINAL UNTIL ;
```

You will find frequent use for the word *INPUT#* because many BASIC interactive programs ask for a signed integer from the keyboard. The following loop increases the loudness, step by step:

```
: P300 16 0 DO 0 115 10 I SOUND
  DELAY LOOP QUIET ;
```

and this loop varies the distortion levels:

```
: P300B 16 0 DO 0 115 I 8 SOUND
  DELAY LOOP QUIET ;
```

This program runs through all of the notes with a short time delay:

```
: DELAY_BRIEF 1000 0 DO LOOP ;
: P300C 256 0 DO 0 I 10 8 SOUND
  DELAY_BRIEF LOOP QUIET ;
```

Notice that you can use the underscore PL/1 style to separate words, as in *DELAY__BRIEF*, which looks neater than the COBOL-style hyphen.

The READ. . .DATA construct is used repeatedly in BASIC. The translation to FORTH is very easy . . .once you know how. Suppose we want to define an array with four locations, holding initial single precision values of *12*, *24*, *48* and *96*, and want to print the last two. Here is the code:

```
12 VARIABLE GROUP
24 , 48 , 96 ,
GROUP 4 + @ . <ret> 48 ok
GROUP 6 + @ . <ret> 96 ok
```

This word will fetch the *n*th element of the array:

```
: GET_GROUP ( stack: n1--n2 )
  2 *       ( find byte offset )
  GROUP     ( leave parm field start )
  +         ( calculate abs location )
  @ ;       ( fetch value )
0 GET_GROUP . <ret> 12 ok
3 GET_GROUP . <ret> 96 ok
```

With this technique, we can store the notes of a musical scale in a table and read them at will:

```
DECIMAL 145 VARIABLE DATA
129 , 122 , 109 , 97 , 92 ,
 82 ,  73 ,  65 , 61 , 54 ,
 48 ,  46 ,  41 , 36 , 32 ,
: WAIT_LONG 30000 0 DO LOOP ;
: WAIT_SHORT 1000 0 DO LOOP ;
: GET_DATA 2 * DATA + @ ;
: SCALE ( p. 302 ) 5 GR.
  16 0 DO 1 COLOR 0 I GET_DATA
  10 8 SOUND 0 I 2 SETCOLOR
  5 11 PLOT 70 11 DRAWTO
  WAIT_LONG 0 0 10 8 SOUND
  WAIT_SHORT LOOP ;
```

If we want to display warm colors with high notes and cold colors with low notes, we can use a similar technique. The trick here is to alternate the numbers in the table and retrieve them in pairs:

```
DECIMAL 122 VARIABLE DATA
9 , 109 , 10 , 97 , 13 ,
92 , 15 , 82 , 1 , 73 ,
2 , 65 , 3 , 61 , 5 ,
: GET_NOTE 4 * DATA + @ ;
: GET_COLOR 4 * DATA 2 + + @ ;
: P302B 3 GR. 8 0 DO 1 COLOR
  0 I GET_COLOR 2 SETCOLOR
  10 9 I - PLOT 20 9 I - DRAWTO
  0 I GET_NOTE 10 8 SOUND WAIT_LONG
  0 0 2 SETCOLOR 0 0 10 8 SOUND
  WAIT_SHORT LOOP ;
```

Here is the way to handle GOSUB and RETURN within a FORTH program, and play three notes:

```
0 VARIABLE N
: GOSUB100 0 N @ 10 8 SOUND
  WAIT_LONG 0 0 10 8 SOUND
  WAIT_SHORT ;
: P303 122 N ! GOSUB100
       109 N ! GOSUB100
        97 N ! GOSUB100 ;
```

You can play a one-octave scale and plot notes on the screen in color, using nested subroutines. Here is a way to do it with two FORTH screens:

```
SCR 1
0 VARIABLE N 0 VARIABLE X
0 VARIABLE Y
: GOSUB200 30000 0 DO LOOP ;
: GOSUB100 X @ Y @ PLOT
  0 N @ 10 8 SOUND GOSUB200 ; -->
SCR 2
: P304 3 GR. 1 COLOR 1 10 2 SETCOLOR
  14 5 DO 0 I PLOT 38 I DRAWTO
  0 15 PLOT 7 15 DRAWTO
  ( 30 15 PLOT 35 15 DRAWTO )
  2 +LOOP GOSUB200
  4 4 2 SETCOLOR 2 COLOR
   3 X ! 15 Y ! 122 N ! GOSUB100
   5 X ! 14 Y ! 109 N ! GOSUB100
   7 X ! 13 Y !  97 N ! GOSUB100
   9 X ! 12 Y !  92 N ! GOSUB100
  11 X ! 11 Y !  82 N ! GOSUB100
  13 X ! 10 Y !  73 N ! GOSUB100
  15 X !  9 Y !  65 N ! GOSUB100
  17 X !  8 Y !  61 N ! GOSUB100
  QUIET ;
```

You can show off two of the four voices with the following:

```
8 VARIABLE L0 0 VARIABLE L1
41 VARIABLE N0 129 VARIABLE N1
: GOSUB1000
  0 N0 @ 10 L0 @ SOUND
  1 N1 @ 10 L1 @ SOUND
  30000 0 DO LOOP ;
: P304B GOSUB1000
  8 L1 ! GOSUB1000
  0 L0 ! GOSUB1000
  0 L1 ! GOSUB1000 ;
```

Here is your chance to demonstrate a simple tune with two voices, using these three FORTH screens:

(continued on next page)

```
SCR 20
0 VARIABLE N0 0 VARIABLE N1
: DELAY50   5000 0 DO LOOP ;
: DELAY100 10000 0 DO LOOP ;
: DELAY150 15000 0 DO LOOP ;
: GOSUB1000 0 N0 @ 10 8 SOUND
  1 N1 @ 10 8 SOUND DELAY50 ;
: GOSUB2000 0 N0 @ 10 8 SOUND
  1 N1 @ 10 8 SOUND DELAY100 ;
: GOSUB3000 0 N0 @ 10 8 SOUND
  1 N1 @ 10 8 SOUND DELAY150 ;
              -->
SCR 21
: 1STHALF
  41 N0 ! 129 N1 ! GOSUB2000
  32 N0 ! 109 N1 ! GOSUB1000
  27 N0 !  92 N1 ! GOSUB3000
  31 N0 !  97 N1 ! GOSUB2000
  32 N0 ! 109 N1 ! GOSUB1000
  36 N0 ! 122 N1 ! GOSUB3000
  41 N0 ! 129 N1 ! GOSUB1000
  36 N0 ! 122 N1 ! GOSUB1000
  32 N0 ! 109 N1 ! GOSUB1000
  36 N0 ! 122 N1 ! GOSUB3000
  27 N0 !  92 N1 ! GOSUB2000
   0 N0 !   0 N1 ! GOSUB1000
  41 N0 ! 129 N1 ! GOSUB2000 ;
              -->
SCR 22
: 2NDHALF
  32 N0 ! 109 N1 ! GOSUB1000
  25 N0 !  92 N1 ! GOSUB3000
  36 N0 ! 122 N1 ! GOSUB2000
  32 N0 ! 109 N1 ! GOSUB1000
  31 N0 !  97 N1 ! GOSUB1000
  32 N0 ! 109 N1 ! GOSUB1000
  36 N0 ! 122 N1 ! GOSUB1000
  27 N0 !  92 N1 ! GOSUB2000
  41 N0 ! 129 N1 ! GOSUB1000
  32 N0 ! 109 N1 ! GOSUB2000
  36 N0 ! 122 N1 ! GOSUB1000
  41 N0 ! 129 N1 ! GOSUB3000
  0 GR. ;
: P305 1STHALF 2NDHALF QUIET ;
```

Now you see how Atari's sound statements can be demonstrated in FORTH.

Random numbers.

One important prerequisite for many Atari games is a random number generator to simulate chance events, such as a throw of the dice or a turn of the cards. Fortunately, Atari has a random number generator built into the hardware, in location 53770 (or 520A in hex), which will give you a random number between 0 and 255. In BASIC, all you need is:

```
10 PRINT PEEK(53770)
```

but in FORTH, it's better to define a word that will give a number between 0 and a chosen upper limit. For example, *10 RND#* will give a random number from 0 to 9. If you want a range of, say 5 to 15, then just add 5 to the result. Here is a definition which divides by the number at the top of the stack, drops the quotient and keeps the remainder as the random number:

```
: RND#   ( stack: n1--n2 )
  53770  ( hardware address )
  C@     ( fetch the byte )
  SWAP   ( swap for divide )
  /MOD   ( divide by n1 )
  DROP   ( discard quotient )
  ;      ( and keep remainder )
```

You will find many uses for this simple and elegant random number generator.

Animation.

What about computer animation? Well, that brings us to the point of this whole story. David Fox says (on page 35) that, "if you want to write computer games for personal computers that effectively use animation. . .you need to know a high-level computer language. . .You should also look at FORTH and C, two high-speed languages that are now available for many personal computers. . .You might also need to understand something called bit-slice microprocessors, as well as the FORTH language. FORTH is a tricky, powerful, exclusive (bordering on religious) language that is also extremely fast."

A few pages later, he adds, "You may also want to investigate FORTH as a graphics language. Although it is rather difficult to learn, it is a somewhat elegant language, and your own graphics instructions are easily added to it. Its advantages include high speed, immediate execution of programs (no compilation like in Pascal and C), ability to define your own commands, and very compact code."

So let us take David Fox at his own word and start translating some of his BASIC programs into FORTH.

(continued on next page)

Here are a few for openers. If you manage to translate some of the others, you will have the makings of a spectacular demo.

To simulate an explosion, you can flash ten random colors on the background with this BASIC code:

```
100 FOR I = 1 TO 10
200 POKE 712, RND(0)*255:NEXT I
300 POKE 712,0:REM FLASH BACKGRND
```

which you can render in FORTH as:

```
: DELAY 1000 0 DO LOOP ;
: EXPLODE 19 GR. 10 1 DO 255 RND#
  712 C! DELAY LOOP 0 712 C! ;
```

This program will flash through all the screen colors so fast that, if you leave out the delay loop, you'll hardly be able to see them. Here it is, in BASIC:

```
100 GRAPHICS 3 + 16
200 REM STEP THROUGH EVERY COLOR
300 FOR I=0 TO 254 STEP 2
400 POKE 712,I:REM CHANGE BACKGRND
500 FOR W=1 TO 50:NEXT W
600 NEXT I:GOTO TO 200
```

and here's the FORTH version:

```
: DELAY100 1000 0 DO LOOP ;
: FLASH 19 GR. 254 0 DO I 712 C!
  DELAY100 2 +LOOP ;
```

Here's an airplane, wingtip lights flashing, heading toward you in a lightning storm:

```
10 GRAPHICS 3 +16
20 COLOR 2
30 PLOT 10,8
40 COLOR 1
50 DRAWTO 29,8
60 COLOR 3
70 PLOT 30,8
80 COLOR 1
90 PLOT 20,5:PLOT 20,6
100 PLOT 19,7:DRAWTO 21,7
110 PLOT 19,9:DRAWTO 21,9
120 SETCOLOR 1,3,6:REM MAKE RED
130 SETCOLOR 2,12,6:REM MAKE GREEN
140 FOR I=1 TO 50:NEXT I:REM PAUSE
150 SETCOLOR 1,0,0:REM MAKE BLACK
160 SETCOLOR 2,0,0:REM MAKE BLACK
170 FOR I=1 TO 400
180 IF RND(0)*20<1 THEN SETCOLOR 4,0,1
4:SETCOLOR 4,0,0:REM RANDOM LIGHTNING
FLASH
190 NEXT I
200 GOTO 120
```

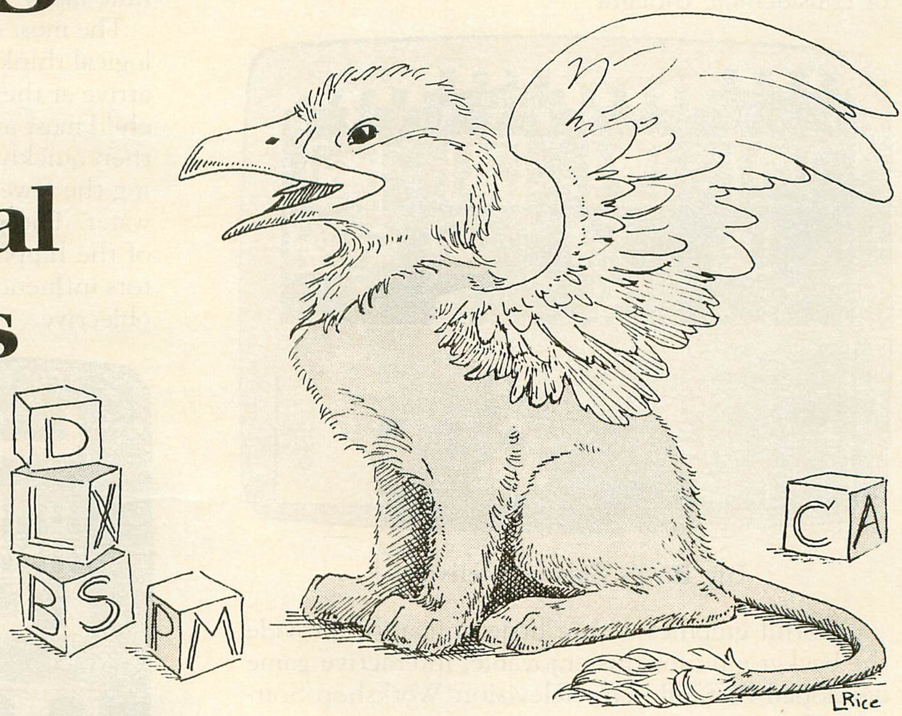
Here's the FORTH counterpart:

```
: SE. SETCOLOR ; : SEC DELAY100 ;
: AIRPLANE_IN_STORM 19 GR.
  2 COLOR 10 8 PLOT
  1 COLOR 29 8 DRAWTO
  3 COLOR 30 8 PLOT
  0 COLOR 20 8 PLOT
  1 COLOR 20 5 PLOT 20 6 PLOT
  19 7 PLOT 21 7 DRAWTO
  19 9 PLOT 21 9 DRAWTO
  BEGIN 1 3 6 SE. 2 12 6 SE.
  SEC 1 0 0 SE. 2 0 0 SE.
  400 1 DO 200 RND# 10 < IF
  4 0 14 SE. 4 0 0 SE. ENDIF
  LOOP ?TERMINAL UNTIL ;
```

As David Fox says, "We will show you how to bring the exciting world of animation into your own home. If you have an Atari microcomputer, you will be able to turn your computer into a fabulous animation machine." □

Griffin's Lair

Educational Programs Review



by Braden E. Griffin, M.D.

This month's column takes a look at five computer games from CBS Software specifically designed for children three to six years of age. Three of these programs are the result of a cooperative effort with Children's Television Network of "Sesame Street" fame. The other two were developed by Joyce Hakansson Associates. The high standard of quality one might expect is evident.

CBS Software's commitment to quality educational software for preschool children is refreshing. Who knows if these programs will make a child smarter, or school easier? On the other hand, I seriously doubt that playing them causes stupidity. I call to mind a properly-rephrased old adage. *What you don't know, can't help you.* No matter what we do, the kids are "alright."

BIG BIRD'S SPECIAL DELIVERY
CBS SOFTWARE
One Fawcett Place
Greenwich, CT 06836
Cartridge or Cassette $29.95

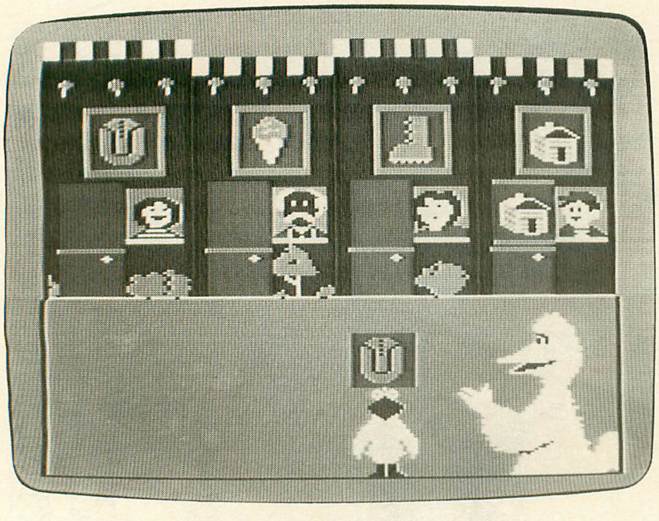
Object recognition is fundamental in the learning process. The ultimate realization of this skill is illustrated in the parable of the drunk who enters a pub accompanied by his duck. Although rewritten in the form of a morality play, the original version begins with the barkeep's query, "Where'd ya get the pig?" "That's no pig. It's a duck," belches the overly saturated swillpot. To which the very astute bartender replies, "I was talking to the duck!"

With **Big Bird's Special Delivery**, preschoolers from three to six years of age can practice object recognition and enhance their abilities to classify objects. Big Bird delivers a package to Little Bird, who is to deliver it to the appropriate store. Four stores are depicted on the screen with their respective owners and a picture of an object sold at the store.

Using the ARROW keys, Little Bird may be moved from store to store. If the match is incorrect, the storekeeper shakes his or her head in a negative fashion. A correct match elicits an approving nod, followed by Little Bird fluttering with glee to a happy little tune.

Two variations of the matching game are offered. In *The Same Game*, the matching picture is exactly like the one on the package.

Find the Right Kind requires the child to match objects which are in the same category. The categorical listing of objects is a simple one designed for this age group. When confronted with the delivery of the butterfly to either the airplane store (both fly) or the store with the picture of a bunny, the correct choice is the latter. The matches are at a very rudimentary level and, though sometimes confusing to overly insightful adults, this grouping of objects is the result of considerable thought.



Big Bird's Special Delivery.

Colorful graphics with a little animation provide the background for this enjoyable, interactive game developed by Children's Television Workshop Software Group.

DUCKS AHOY
CBS SOFTWARE
One Fawcett Place
Greenwich, CT 06836
Cartridge or Cassette Joystick $29.95

Preschoolers will enjoy this game, while developing a number of learning skills.

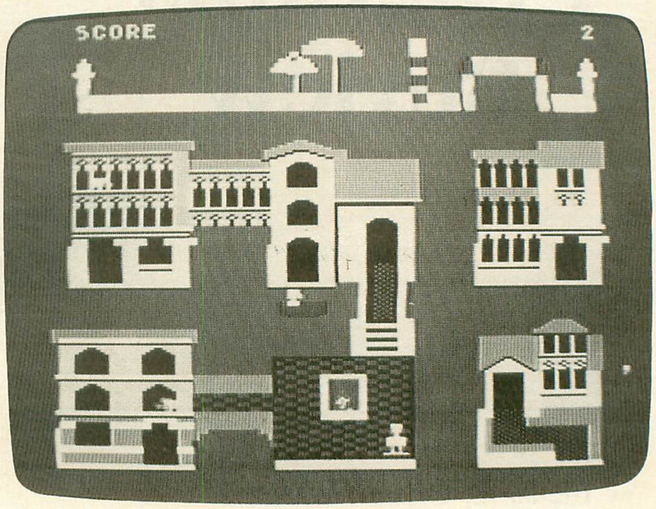
Guiding a gondola through the canals of Venice provides a classic setting for this well designed program from Joyce Hakansson Associates, Inc. That's Italy's Venice, not the one in California. The absence of roller skates and the paucity of ducks "pumping iron" makes this immediately apparent. Italian ducks seem to be better off riding in a boat than swimming in their natural aquatic habitat.

The object is to pick up the ducks as they jump from the docks of their homes, transporting them to the beachfront boardwalk. The ducks waddle through their homes, across bridges and around fountains, to arrive at one of a number of launching points.

A joystick is used to maneuver one's craft into the proper position for embarkation. One or two ducks may be carried at a time, depending on the location of a menacing hippo. The sight of bubbles surfacing from below temporarily reveals itself and helps avoid the sinking of the boat. Too many ducks in the boat, an improperly positioned boarding, or allowing ten ducks to jump into the water—all result in a quick visit to Davy Jones's locker.

The total number of ducks delivered is displayed and continues to increase until all four of one's boats are sunk. Counting skills are exercised as the child keeps track of how many ducks are at the beach and how many have jumped into the water.

The most emphasized skills in **Ducks Ahoy** involve logical thinking and planning ahead. Since the ducks arrive at their respective docks at different times, the child must anticipate just the right time to dock, and then quickly steer the boat to the other ducks, keeping the fewest ducks possible from jumping into the water. The presence of potential peril, in the form of the hippo, serves to develop an awareness of factors influencing outcome, exclusive of the immediate objective.



Ducks Ahoy.

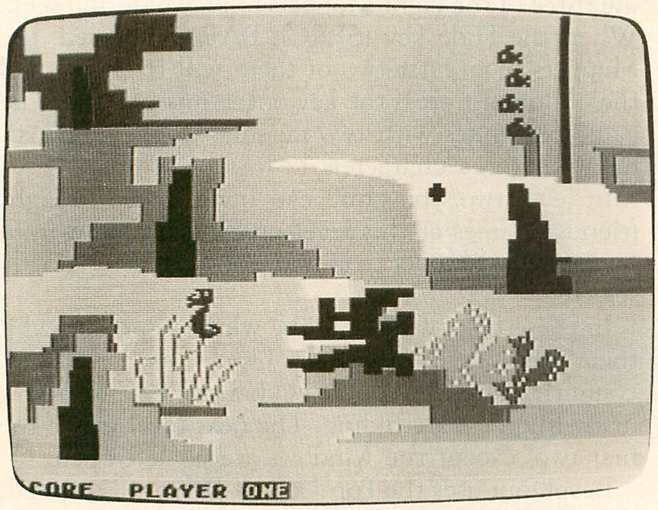
Pleasing musical interludes and lively graphics add extra appeal to this program. An activity book for use without the computer is included. Though designed for the very young, this game will be enjoyed by children of all ages, as they perfect strategic approaches. Described as the "ducky" discovery game of planning and surprise, **Ducks Ahoy** is just about everything it is "quacked up" to be.

SEAHORSE HIDE'N SEEK
CBS SOFTWARE
One Fawcett Place
Greenwich, CT 06836
Cartridge or Cassette Joystick $29.95

Five years ago, a game like **Seahorse Hide'n Seek** would have been an arcade hit. Three years ago, it would have been a popular home computer game for all ages. Now, games with far more sophisticated programming appear in computer magazines, available for the price of a few hours' typing.

Does this lessen the value of **Seahorse**? Not in the least. This program was not developed years ago for all age groups. It was developed by Joyce Hakansson Associates for preschoolers, with particular attention to age-specific developmental abilities. Simple as it may appear, there is ample evidence of thoughtful design.

The capsule summary on the packaging states that this is "an easy-to-fathom game of color camouflage and size relationships." I know that many of you, seeing such a clever play on words, must think that I'm ghosting for CBS Software. I, too, was astonished. Someday, no doubt, I'll come face to face with this crafty wordsmith. And, should I be bested by this pretender's badinage, I shall render in tribute one dozen cream-filled *bon mots*.



Seahorse Hide'n Seek.

But. . .the game. It is easy to play and understand. I am sure that, after a few minutes of play, any child asked how things are going will respond, "Swimmingly." (Gotcha!)

The object is to guide a sea horse across a coral reef to the safety of a shipwreck hideaway. Between the sea horse and its destination are a number of strange-looking lagoon fish. If caught by one of these denizens of the deep, the sea horse is chased out to sea—and out of the game. Although these hungry predators come very close, no violence occurs, as our heroes get their prehensile tails out of there.

The game begins with four sea horses. On occasion, the bad guys can be outmaneuvered, but there are other avenues of escape. In nature, the game tells us, sea horses use camouflage to hide. In **Seahorse**, one can do the same. By positioning the sea horse on a piece of coral, the horse's color can be changed to match that of the underlying coral. If the colors match and the coral totally encompasses the hiding sea horse, the fish will pass right by.

The coral caves will also prove a refuge. . .that is, *if* the sea horse fits through the opening, and the cave is unoccupied. In the former case, one's sea horse will appear at a distant, but identically-shaped cave opening. This distance may be nearer to or farther from the much sought-after shipwreck. The stipulation of occupancy refers to an octopus who hides in random caves, waiting to tickle the sea horse.

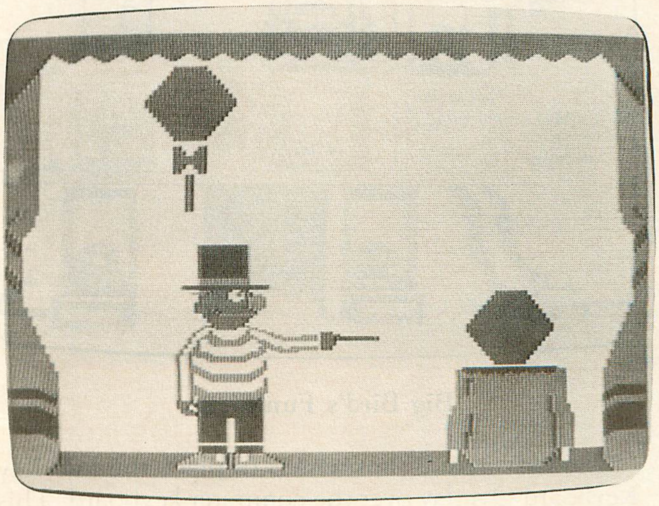
Guiding the sea horse into the hole in the shipwreck results in a score. After five successes have been recorded, the size of the next sea horse is doubled, making it much more difficult to hide. The larger and smaller variations alternate throughout the game.

The obvious emphasis of **Seahorse Hide'n Seek** is on the relationships of size and color. Deciding if a cave opening is large enough, or if the sea horse can be completely covered by a particular piece of coral, provides an excellent opportunity to practice these skills.

This is an enjoyable game which, like the others, would be an ideal setting for a child's initial computer experience. The game design is excellent; and the educational concepts, sound. This one is a keeper, especially with those cute little broncos of the brine. I wondered if these were the mounts sought after by Dick Butkus and Bubba Smith for their water polo match. And I got the answer straight from the horse's mouth: "Nayy. . .nayyy."

ERNIE'S MAGIC SHAPES
CBS SOFTWARE
One Fawcett Place
Greenwich, CT 06836
Cartridge or Cassette $29.95

Ernie of "Sesame Street" dons a magician's cloak and hat to perform a little prestidigitation, as he helps youngsters develop their abilities in visual discrimination. When Ernie waves his magic wand, a colorful geometric shape appears (as if by magic) above his head. A further gesture and—*voila!*—another shape materializes on the table next to him. If this shape matches the one above Ernie's head, and the appropriate ARROW key is pressed, *zap!*—it appears at the top beside the original. Shapes continue to appear on the table until the matching one is chosen.



Ernie's Magic Shapes.

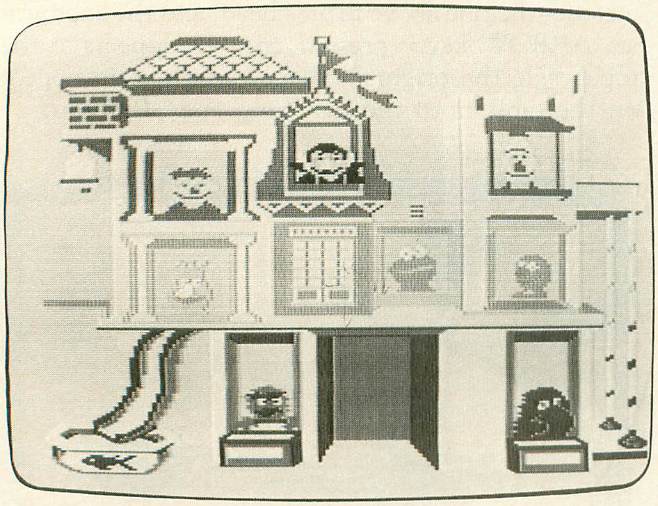
There are six variations of this activity. Single geometric forms of different colors are presented in the easier games, requiring the child to match the shape, size or color. More complex composites of shapes are found as the level of difficulty increases. The child must match the individual components by color, shape and size. A successful match is rewarded by the appearance of a hyperactive rabbit jumping on Ernie's wand.

Intended for preschoolers, this program does not overwhelm children with computer manipulations or create frustration as they play. Simplicity of design is a most important factor in developing these early skills. A thoughtfully prepared combination of this simplicity with eye-catching, colorful graphics is necessary to maintain the attention of this age group.

Ernie's Magic Shapes would be an excellent first introduction of the computer to children as young as three years old. The availability of an easy-to-use, hard-to-break cartridge adds further appeal to this worthwhile educational game for very young hackers.

BIG BIRD'S FUNHOUSE
CBS SOFTWARE
One Fawcett Place
Greenwich, CT 06836
48K Cartridge, Disk or Cassette $34.95

A trip down Sesame Street to visit **Big Bird's Funhouse**—filled with Muppets—is certain to attract a child's attention. This is a game of hide and seek. The Muppets are hiding, and Big Bird needs help finding them. The progression of difficulty through the five levels is well designed. The skills practiced and refined at each level establish the groundwork for succeeding levels.



Big Bird's Funhouse.

The first item of business is to place the EasyKey Keyboard Overlay over the computer keyboard. This flexible vinyl cover makes it simpler for children to use the keyboard. The overlay has pictures of the Muppet friends, each with its own key to punch. Other areas are designated for game-playing functions. The package includes a cover for both the 800 and the 800XL, because of dissimilar keyboards.

Neither of these covers works with the Atari 400 keyboard. There are still a lot of those durable 400s out there which, years from now, will be viewed with the same wonder as a '55 Chevy. There should have been some mention of the 400 keyboard, with at least a description of how to construct an overlay. It would be fairly simple to create one using construction paper and cannibalized parts from one of the vinyl covers.

The first level of play involves selecting and inviting three of the eight Muppets to play hide and seek. When the Hide key (SPACE BAR) is pressed, the Muppets hide. First two of them will reappear, and the child must press the key of the missing one. The game continues with two missing Muppets and, finally, all three characters to find.

In level two, the child can invite three to eight friends. Games at this level progress similarly, from one missing Muppet to as many of them as originally selected.

In these first two levels, the windows are closed if there is no one in that room. Using this, a child can count the number of open windows to remember how many Muppets are hiding. The next level is like the first two, except the windows are all left open.

Level four uses the Pop-Up key. When it's pressed, three Muppets will pop up in a special order, accompanied by their individual hiding theme. After they hide again, the child must select the missing Muppets in the exact sequence in which they originally appeared. The round continues until five of the characters have been identified in the correct order. Once this is achieved, a new set of playmates is generated.

The final level of play is definitely addicting. Pressing the Pop-Up key results in the appearance of a single Muppet and its musical riff. After it disappears, one presses the key of the Muppet seen. Then this same Muppet appears, followed by another. This is repeated, with the sequence growing by one each round. Individual Muppets may make several guest appearances in each round.

Visual and auditory discrimination skills are extensively involved throughout this program. It might be interesting to test these separately by turning off the sound or not looking at the screen. The fact that it is much easier using both stresses the importance of this perceptual coordination. Children will find many ways to memorize patterns and figure out the tricks that work best for them. Finally, the child's ability to place objects in sequence will improve markedly.

Many favorite Muppets are featured, including: Bert, Ernie, Oscar, Cookie Monster and the lovable Snuffle-Upagas. The sound and graphic enhancements are well done and appropriately geared for the younger child. The manual, like all the others, includes a section on additional activities.

I almost forgot. When the child succeeds, Big Bird skates gleefully in front of the house. Now, that's positive reinforcement. □

Popcorn

16K Cassette or 24K Disk

by Mark and Cathy Sloatman

The story. . .You're a concession worker at the local matinee. Since the day is over, you are busy cleaning under the tables. Then the worst happens! The popcorn popper's gone mad! Now, you must clean up (catch) all of the popcorn to make the boss happy. If you do well, he'll congratulate you. If not—you guessed it—looks like you're *fired!*

Instructions.

After **Popcorn!** is loaded, you can press START to begin in level 1, or press SELECT then START to begin in level 5. Levels 1 through 5 are for the little ones. The popcorn and the hand move more slowly, so the younger player won't get discouraged. Only twenty-five popcorn kernels need to be caught to advance to the next level (compared with fifty for higher levels), and, if the child finishes level 3, a happy boss appears.

To play **Popcorn!**, move your little on-screen hand to catch the kernels. The amount of points for each piece caught depends on what level you're in. There are nineteen levels of difficulty (if you finish level 19, let me know).

You start out with three chances. When you miss three kernels, the boss will pull you off the screen with his cane, and you'll lose a chance. Remaining chances are indicated by the number of pieces of popcorn in the upper right side of the screen. If you lose all your chances, you'll see the boss—and get fired.

There is good news, however. Making it through levels 3, 8, 12 or 16 will send you to the boss's office, too, but this time he'll thank you for a job well done.

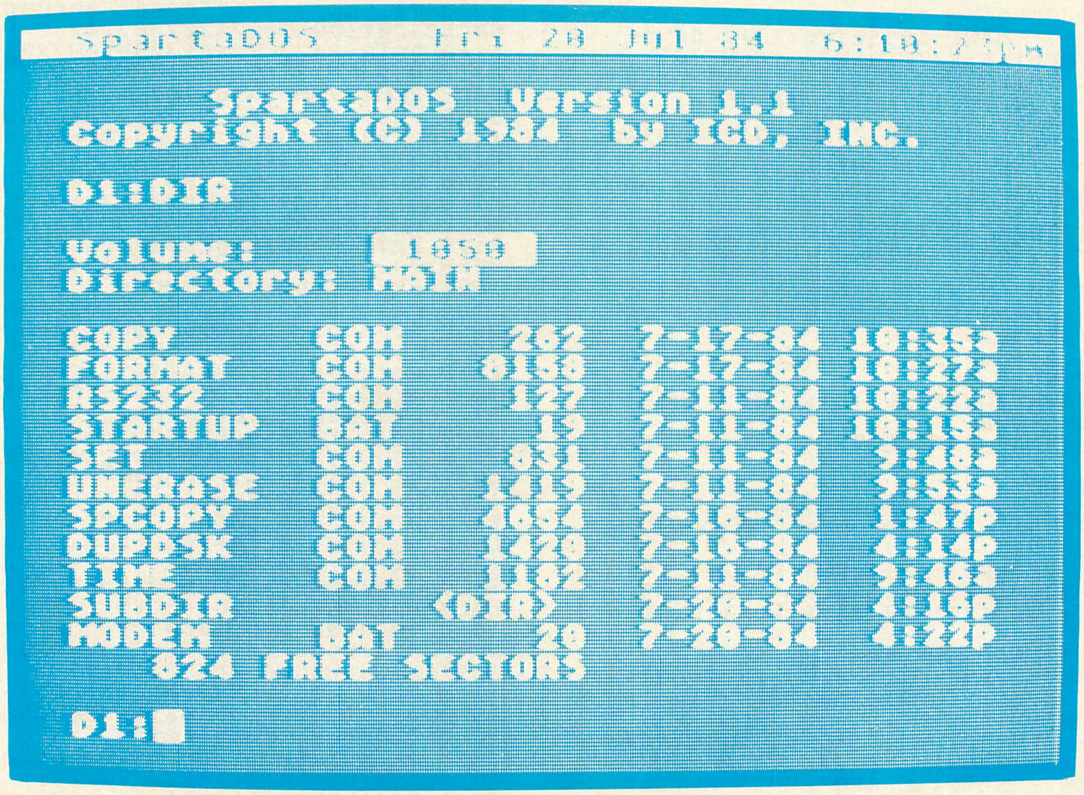
Every 10,000 points earns you another life (up to five). To pause the game, hit the SPACE BAR. To resume play, hit the SPACE BAR again. After the game is over, press START or the trigger button to play again. In the middle of a game, you may begin once more by pressing START.

Have some popcorn and play **Popcorn!** □

BASIC listing.

```
10 REM *** POPCORN ***
20 TRAP 20:? "MAKE CASSETTE (0), OR DI
SK (1)";:INPUT DSK:IF DSK>1 THEN 20
30 TRAP 40000:DATA 0,1,2,3,4,5,6,7,8,9
,0,0,0,0,0,0,0,10,11,12,13,14,15
40 DIM DAT$(91),HEX(22):FOR X=0 TO 22:
READ N:HEX(X)=N:NEXT X:LINE=990:RESTOR
E 1000:TRAP 120:? "CHECKING DATA"
50 LINE=LINE+10:? "LINE:";LINE:READ DA
T$:IF LEN(DAT$)<>90 THEN 220
60 DATLIN=PEEK(183)+PEEK(184)*256:IF D
ATLIN<>LINE THEN ? "LINE ";LINE;" MISS
ING!":END
70 FOR X=1 TO 89 STEP 2:D1=ASC(DAT$(X,
X))-48:D2=ASC(DAT$(X+1,X+1))-48:BYTE=H
EX(D1)*16+HEX(D2)
80 IF PASS=2 THEN PUT #1,BYTE:NEXT X:R
EAD CHKSUM:GOTO 50
90 TOTAL=TOTAL+BYTE:IF TOTAL>999 THEN
TOTAL=TOTAL-1000
100 NEXT X:READ CHKSUM:IF TOTAL=CHKSUM
 THEN 50
110 GOTO 220
120 IF PEEK(195)<>6 THEN 220
130 IF PASS=0 THEN 170
140 IF  NOT DSK THEN 160
150 PUT #1,224:PUT #1,2:PUT #1,225:PUT
 #1,2:PUT #1,0:PUT #1,32:CLOSE #1:END
160 FOR X=1 TO 117:PUT #1,0:NEXT X:CLO
SE #1:END
170 IF  NOT DSK THEN 200
180 ? "INSERT DISK WITH DOS, PRESS RET
URN";:DIM IN$(1):INPUT IN$:OPEN #1,8,0
,"D:AUTORUN.SYS"
190 PUT #1,255:PUT #1,255:PUT #1,0:PUT
 #1,32:PUT #1,226:PUT #1,45:GOTO 210
200 ? "READY CASSETTE AND PRESS RETURN
";:OPEN #1,8,128,"C:":RESTORE 230:FOR
X=1 TO 40:READ N:PUT #1,N:NEXT X
210 ? :? "WRITING FILE":PASS=2:LINE=99
0:RESTORE 1000:TRAP 120:GOTO 50
220 ? "BAD DATA: LINE ";LINE:END
230 DATA 0,29,216,31,255,31,169,0,141,
47,2,169,60,141,2,211,169,0,141,231,2,
133,14,169,56,141,232,2
240 DATA 133,15,169,0,133,10,169,32,13
3,11,24,96
1000 DATA A9018509A900850CA920850DA90E
8DC502A9328DC60220742CA9008D2F02859C8D
08D28D01D28D03D2A2299580,441
1010 DATA CA10FB20B72420E023A9D68580A9
0F8581A20520A52DA200BDAD279D7810E8D0F7
BDAD289D7811E8E018D0F5A2,285
1020 DATA 00BDC5289DFF12E8E01BD0F5A217
BDDF289DD70FCAD0F7204324A9308DF402A9FE
8D0002A9298D0102A9F78D30,22
1030 DATA 02A9288D3102A9C08D0ED4A93A8D
2F0220F922A901859AA90285A1A9118DF10FA9
0385A28D0FD2203D23AD1FD0,763
1040 DATA C906F010C905D007859AA9158DF1
0FAD8402D0E9A91E209D24AD1FD0C906F0F920
102420ED23A988859D8D0AD0,143
1050 DATA A9408D0ED4A9018D6F028D09D0A9
038D1DD0A9008D07D42024242004 24A977859C
8D1ED0A49AB974248593A904,844
1060 DATA A212A00920C22CA906A214A00920
C22CAD1FD0C906D00AAD1FD0C906F0F94C0020
ADFC023015A9FF8DFC02A5A4,956
1070 DATA 450185A4F008ADFC0230FB4C3421
206524207D2CA599D009AD0AD2C914901BB004
C699D012A908A001A21220C2,649
1080 DATA 2CA90AA001A21420C22C4CDE21A9
0CA212A00120C22CA90EA214A00120C22CA901
8584A59A0A1869048599A200,749
1090 DATA BD0006F009E8E8E00CF03E4C9521
8694A59A0A8596A201BD0006F004C5969029E8
E8E00DF0F1A694A00AA90E20,362
1100 DATA AA24C9109004C91690F19D000685
84A90CA00120AA24E89D000686A3AD05D0D003
4CCB228582205D2DA49AC006,100
1110 DATA 9002A0052009238D1ED0E69BA49A
A59BC005B007C919B0074CCB22C93290F9A908
A001A21220C22CA90AA001A2,925
1120 DATA 1420C22CA9068594A92885932065
24A9AF8596C694A594F01BA005200923C696A5
96C9A090E28D01D2A90C8D00,191
1130 DATA D22088244C3A22A900859B8D01D2
A49AC013F069E69AA49AC004F00CC009F008C0
0DF004C011D03D8C762620B7,493
1140 DATA 2420CE24A93A8D2F02201024A904
A212A00920C22CA906A214A00920C22CA908A0
01A21220C22CA90AA001A214,202
1150 DATA 20C22CA93C209D24AC7626C013F0
11C00AD00DA9108DF10FA9118DF00F4CC622EE
F10FB974248593A59FC903B0,432
1160 DATA 034C1E21205223A900859FC6A2A5
A2F00D203D23A977859C209D244C1E21A9018D
D72620CE244C0020A910A200,486
1170 DATA 9DDF0FE8E006F0034CFD2260ADE0
0F8597A204BDDF0FC919D009A9109DDF0FCA4C
1023FEDF0F88D0E8A597CDE0,499
1180 DATA 0FF00FA5A2C905F009E6A2203D23
A9C0859D60A205A9009DF50FCA10FAA6A2A941
9DF50FCAD0FA60A9A8859D20,433
1190 DATA B724A908A001A21220C22CA90AA0
01A21420C22CA914209D24A99F8597A9CF858B
A227A000208824B9DF269DB0,250
1200 DATA 1EC68BA58B8D02D0A59C38E92AC6
97C597B00BC8C00490E0A000CA4C7C23E68BBD
B01EF0174A4A9DB01E208824,911
1210 DATA E68BA58B8D02D038E906859C4CA4
23E8E02990DFA59C8596A596859CC9F5B00E18
69068D02D0E6962088244CC9,849
1220 DATA 2360A9008580A9048581A2054CA5
2DBDB32D9DD204BDC22D9DCD05BDD12D9DD706
E8E00FD0E9A207BD1D249DC0,382
1230 DATA 02CA10F760A9008580A9108581A2
0F4CA52D3A480F00280E00A9358D3002A9298D
3102A99F8D0002A9298D0102,124
1240 DATA A901209D24A9C08D0ED460A200BD
00E09D0030BD00E19D0031BD00E29D0032E8D0
EBBD232A9D0032E8E0A1D0F5,583
1250 DATA A000A220C8C493D0FBA000CAD0F6
6000C8967864504641 3C37322D28231E191612
0D0A9848A900A8186901C916,953
1260 DATA F006C8D0FD4C8D2468A8608594A9
008514A514C594D0FA608598AD0AD225988498
18659860A90F85A1207D2CA9,540
1270 DATA 00AA9D0006E8E00C90F8A90285A1
60A9408D0ED420E023A9008580A9108581A205
20A52DA230A97E9D9E06CA10,515
1280 DATA FAA200BDBC2A9D0030E8E030D0F5
A200BDEC2A9D0031E8E098D0F5A9018D6F02A9
038D0AD0A9E48DC202A9888D,539
1290 DATA C502ADD726D011A90DA00120AA24
0A0A0A0A18690E8DC602A9148D3002A9298D31
02A9138D0002A92A8D0102A9,131
1300 DATA C08D0ED4A9008594A9108595E695
18A594695D8594900 2E695ADD726D013AD0AD2
C906B0F9AABDD8268DD626A9,180
1310 DATA 254C8125A9008DD6268DC602A929
8DDE26A200A000BD77269194E8C8BD77269194
18A59469148594900 2E695E8,505
1320 DATA E00AD0E2A200A9019D5C11A9029D
5F11188A6914AAE064D0EDA9038D49118D4A11
A9048D4811A9058D4B11A970,163
1330 DATA 8D02D0209D24A9648D01D28D03D2
A9188D00D2A9198D02D2A97185962088 24A596
8D02D0C987F00B208824E696,411
1340 DATA 2088244CE925A9008D01D28D03D2
A91E209D24ADD726D004A98AD002A999859DAC
DE268C5D11C88C5E11C88C71,613
1350 DATA 11C88C7211AED626A000BD832699
D011C981F005E8C84C3126A9FF209D24A9008D
2F02A901209D24A9008D02D0,516
1360 DATA 8D0AD08DD726A9408D0ED420E023
20ED232043242004242024 24AE7526AC762660
000021222 3246D6E6F707172,335
1370 DATA 0101B9AFB587B2A580A6A9B2A5A4
8180A7B2A5A1B480AAAFA2818080B7A1B980B4
AF80A7AF81808080B7AFB7A5,155
1380 DATA A5818080A6A1AEB4A1B3B4A9A381
ACA9ABA58C80A1B7A5B3AFADA5818080A7AFAF
A480B7AFB2AB8100000D1924,617
1390 DATA 2E39470001051555790460 04A204
90028002790460 04A20490028002790460 0479
0460046C048004A204900280,371
1400 DATA 02790460 04A20490028002790460
04A20490028002790460046C04600479 08A204
8004D904C102AD02A2048004,460
```

. . . Until now, there was no reason to buy another DOS for your Atari® Computer



Introducing SpartaDOS™ $39.95

a resident DOS that supports all drives, all densities, 5¼ and 8 inch, single or double sided, time/date stamping of all files, unlimited multiple directories, user created batch files, automatic (intelligent) format selection, works with **all** Atari compatible drives including the ATR8000®.

. . . Finally, true double density for the Atari 1050 and it's affordable!

Introducing the US Doubler™ . . . $69.95

turns your Atari 1050 into the drive Atari **should** have made. Fully compatible with all existing software. When used with included SpartaDOS, UltraSpeed™ I/O reads and writes an amazing **3 times faster** than your present 1050 - plus in single density, density and one-half (Atari double), and true double density! Installation required.

other fine products from ICD include: The Chip with Archiver I™ . . $99.95, Happy Archiver™ . . $39.95, and coming soon Archiver II™.

ICD, Inc.
828 Green Meadow Avenue
Rockford, IL 61107
(815) 229-2999 (1-9 pm CST)

Distributors now in:
United Kingdom and Canada

order direct or call for the dealer or distributor nearest you.

CIRCLE #125 ON READER SERVICE CARD

```
1410 DATA D904C102AD02A2048004A2048004
9004AD04D904C102AD02A2048004D904C102AD
02A2048004D904C102AD02A2,503
1420 DATA 04800490048004A20800003C0235
022F043C04350440043C08005102510651086C
0460067910F31000F308D904,855
1430 DATA CC0CD90CF302FF02F302FF02F308
D904F308000021022502210225020000000041
414141004141414100414141,890
1440 DATA 4100414141410041414141004141
41410041000041004100000000410000410041
000041004100004100410000,385
1450 DATA 0000410000410041000041004100
00410041000000004100004100410000410041
000041004100000000410000,360
1460 DATA 4100410000410041000041004100
00000041414141004100004100414141410041
000000004100004100414141,790
1470 DATA 0000414100410041000000004100
00000041000041004100000000410000000041
000041004100004100410041,765
1480 DATA 4100410000000041000000004100
00410041000000004100000000410000410041
000041004100004100000000,610
1490 DATA 0000410000000041414141004100
00000041414141004141414100410000410041
000041004100002279002D61,142
1500 DATA 726B00616E640023617468790033
6C6F61746D616E0033232F3225000000000000
000000000000002C2536252C,378
1510 DATA 7070420010020202020202020202
0202028202020202020202020241F728707070 46
001006060606060606060606,688
1520 DATA 0606060606060606060686060670
4114297070C2D60F4D00100D0D0D0D0D0D0D0D
0D0D0D0D0D0D0D0D0D0D0D0D,132
1530 DATA 0D0D0D0D0D0D0D0D0D0D0D0D0D0D
0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D0D
0D0D0D0D0D0D0D0D0D0D0D0D,717
1540 DATA 0D0D0D0D0D0D0D0D0D0D0D0D0D0D
0D0D0D0D0D0D0D8D0D0D0D0D8D0D0D8D413529
48A9DA8D0AD48D18D0A9B48D,335
1550 DATA 0002A9298D0102684048A9E48D0A
D48D13D0ADC2028D16D0A9CF8D0002A9298D01
02684048A90A8D0AD48D12D0,821
1560 DATA A9048D1BD0A9E98D0002A9298D01
02684048A9C88D0AD48D1AD0A99F8D0002A929
8D0102684048A9E08D0AD48D,578
1570 DATA 09D4A90A8D17D0A9008D18D06840
48A9E08D0AD48D09D4A9088D1AD06840000000
000000000060F6FF7F3E7878,996
1580 DATA 0000000038EFEF2A0A2E2F0A0000
28AEFAE8C080000000AAAAAAAAAA55000000AA
282800AA55000000AAAAAAAA,821
1590 DATA AA55000000000000000000000000
00000000000055000000000000002855000000
000000000000550000000000,371
1600 DATA 0000000000005500000000000028
55000000000000000000550000000000000000
000000000000000000000000,666
1610 DATA 00000000F7F7F7007F7F7F38F8F8
F8387878781C1F1F1F1C1F1F1FFFFF00000000
00007FFFFCFC78787878FEFF,631
1620 DATA 3F3F1F1F1F1F0000000000000000
00070F193F3D7D7F00E0F098FCBCBEFE3F3C1F
0F07030300FC3CF8F0E0C0C0,394
1630 DATA 0000070D1B373D7D7F00E0B0D8EC
BCBEFE3B3C1F0F07030300DC3CF8F0E0C0C000
00070F1B3D3E7D7D00E0F0D8,949
1640 DATA BC7CBEBE3F3C1B0F07030300FC3C
D8F0E0C0C0003F7F7F6F6F6F6FFFCFEFEF6F6
F6F6F6606F0F6E6E0E0E0E06,547
1650 DATA F6F07676707070700E0E0E0E0E1E
3E707070707070787C8E098A0D86118215D8A5
8FF005C68F4CC62BA5904901,111
1660 DATA 8590D01BA69DBDE326F0218D04D2
A9AA8D05D2E8BDE326858FE8869D4CC62BA900
8D05D2A69DCABDE326858FAD,478
1670 DATA 7802C907F019C90BD02BA59CC932
9021A49AC0059002C69CC69CC69C4CF82BA59C
C9C3B00CA49AC0059002E69C,327
1680 DATA E69CE69CA900854DA59C8D01D018
69058D00D0A584F02BE68EA58EC902D023A68D
BD842B8D03D2E8BD842B8D02,101
1690 DATA D2A900858EE8868DE008D009A900
8D03D2858D8584A582F037E685A585C901D02F
A583D004A9F0858CA9AA8D01,152
1700 DATA D2A58C8D00D2F00F38E928858CA9
008585A9018583D00CA9F0858CA90085828583
85854C62E4A907A22BA08C4C,767
1710 DATA 5CE4A9008592A692BD0006D009E8
E88692E00C90F160BD00068586E8BD0006859E
C961900ECAA9009D0006E89D,279
1720 DATA 0006E69FD003FE0006E88692A5A1
858720C82CA692E00C90C0608587849E8686A9
01858AA93285A8A58AD004E6,39
1730 DATA 86E687A900858985A085A68595A5
9E0A26950A26950A2695A69586A685A50A2695
0A26951865A585A59565A6,241
1740 DATA 85A6A5A518658685A59002E6A618
A5A5690085A5A91065A685A6A90085 95A5870A
26890A26890A268918658785,176
1750 DATA A7A58965A885A8A4A0B1A7A00091
A5A5A518692885A59002E6A6E6A0A5A0C909D0
E5C68AA58AC9FFF0034CD02C,720
1760 DATA 60A59C38E9244A4A38E9048591A0
00A200BD0006F004C591F010E8E8E00CD0F1E6
91C8A200C006D0E860E8BD00,581
1770 DATA 06CAC94E90E7859EBD00068586A9
0F8587A9009D00069D01064CC82CA000989180
C8D0FBE681CAD0F6602070F0,203
1780 DATA F2F2FE7E3E0F0F0F0F0F0F0FFFFF
FF7E7E7E7E7E7E7E000000000000078F8808000
000000000000000000000000,505
1790 REM * 3555 BYTES
```

CHECKSUM DATA.

(see page 10)

```
10 DATA 942,351,496,811,423,729,200,60
3,555,573,694,613,29,205,214,7438
160 DATA 145,198,962,633,491,30,155,11
6,169,732,970,213,858,980,814,7466
1060 DATA 882,705,576,947,740,766,671,
763,513,975,716,370,96,760,982,10462
1210 DATA 31,105,613,89,595,879,794,11
5,703,73,759,845,643,925,743,7912
1360 DATA 461,993,111,4,96,527,33,169,
555,458,538,467,437,605,88,5542
1510 DATA 762,547,124,117,696,854,724,
798,486,403,128,808,95,541,866,7949
1660 DATA 260,967,160,923,982,793,964,
588,692,74,883,840,684,663,9473
```

Assembly language listing.

```
; +-------------------------+
; |         POPCORN!        |
; |                         |
; | Mark and Cathy Sloatman |
; |                         |
; |    ANALOG  Computing    |
; +-------------------------+
;
;SYSTEM EQUATES
;
BOOT     =  $09
DOSINI   =  $0C
ATRACT   =  $4D
VDSLST   =  $0200
DMA      =  $022F
SDLSTL   =  $0230
GPRIOR   =  $026F
STICK0   =  $0278
STRIG    =  $0284
PCOLR0   =  $02C0
PCOLR2   =  $02C2
COLOR1   =  $02C5
COLOR2   =  $02C6
CHSET    =  $02F4
CH       =  $02FC
HPOSP0   =  $D000
HPOSP1   =  $D001
HPOSP2   =  $D002
P1PF     =  $D005
SIZEP1   =  $D009
SIZEP2   =  $D00A
COLPM0   =  $D012
COLPM1   =  $D013
COLPF0   =  $D016
COLPF1   =  $D017
COLPF2   =  $D018
COLBK    =  $D01A
PRIOR    =  $D01B
GRACTL   =  $D01D
HITCLR   =  $D01E
CONSOL   =  $D01F
AUDF1    =  $D200
AUDC1    =  $D201
AUDF2    =  $D202
AUDC2    =  $D203
```

```
AUDF3    =   $D204
AUDC3    =   $D205
AUDCTL   =   $D208
RAND     =   $D20A
SKCTL    =   $D20F
PMBASE   =   $D407
CHBASE   =   $D409
WSYNC    =   $D40A
NMIEN    =   $D40E
CHORG    =   $E000
XITVBV   =   $E462
SETVBV   =   $E45C
;
;P/M AND SCRN MEMORY USE
;
CHARORG  =   $3000
CHARORG2 =   CHARORG+$0100
CHARORG3 =   CHARORG+$0200
SCRN     =   $1000
SCORELN  =   SCRN-$2A
CANESCRN =   SCRN+$0EB0
SHOWLEV  =   SCORELN+$1B
SHOWLIFE =   SHOWLEV+$04
;
PM       =   $00
PM0      =   PM+1234
PM1      =   PM+1485
PM2      =   PM+1751
POSITION =   $0600
;
;ZERO PAGE VARIABLES
;
         *=  $80
CLR      *=  *+2
CATCHSND *=  *+1
CATCHSND1 *= *+1
SNDFLAGPOP *= *+1
SNDDELAY *=  *+1
HORIZ    *=  *+1
CHR      *=  *+1
CHAROFS  *=  *+2
CHECK    *=  *+1
HPOSCANE *=  *+1
FREQ     *=  *+1
XFLAGPOP *=  *+1
DELAYPOP *=  *+1
LENGTH   *=  *+1
SNGFLAG  *=  *+1
HPOS1    *=  *+1
XREG     *=  *+1
SETDELAY *=  *+1
TEMP     *=  *+2
TEMP1    *=  *+1
TEMP2    *=  *+1
TEMPA    *=  *+1
TIMER    *=  *+1
LEVEL    *=  *+1
NUMBER   *=  *+1
HPOS     *=  *+1
SONGXREG *=  *+1
VERT     *=  *+1
MISSPOP  *=  *+1
PLUS     *=  *+1
LOADCHAR *=  *+1
LIVES    *=  *+1
LASTPOP  *=  *+1
PAUSEON  *=  *+1
OFFSET   *=  *+2
CHAR     *=  *+2
;
;INITALIZE GAME
;
         *=  $2000    ;RUN ADDRESS
;
START    LDA #1       ;TELL SYSTEM
         STA BOOT     ;DSK BOOT OCCURED
;
;FOR CASSETTE USERS
;
         LDA # <START ;NOW, HAVE
         STA DOSINI   ;SYSTEM RESET
         LDA # >START ;START THE
         STA DOSINI+1 ;GAME OVER!
         LDA #14      ;SET UP COVER
         STA COLOR1   ;COLORS
         LDA #50
         STA COLOR2
         JSR INITVBLANK ;TURN ON VBI
         LDA #0
         STA DMA      ;CLEANER SWITCHING
         STA HPOS     ;PLAYERS OFF SCRN
         STA AUDCTL   ;TURN OFF SOUND
         STA AUDC1
         STA AUDC2
         LDX #41      ;41 BYTES TO CLR
CLRZERO  STA $80,X    ;CLR
         DEX          ;ZERO PAGE
         BPL CLRZERO  ;VARIABLES
         JSR ERASEPOPCORN ;CLR SCRN
         JSR CLRPM    ;CLR PM GRAPHICS
;
;CLR TITLE PAGE
;
         LDA # <SCORELN ;LOW BYTE
         STA CLR
         LDA # >SCORELN ;HIGH BYTE
         STA CLR+1
         LDX #5       ;5 PAGES
         JSR CLR0     ;CLR IT!
         LDX #0
PUTPOP   LDA POPCORNSCRN,X ;PUTPOP
         STA SCRN+$78,X ;AND PUTPOP1
         INX          ;PUTS "POPCORN"
         BNE PUTPOP   ;TITLE
PUTPOP1  LDA POPCORNSCRN+$0100,X
         STA SCRN+$0178,X ;ON SCRN
         INX
         CPX #24
         BNE PUTPOP1
         LDX #0
PUTCREDS LDA MESSAGE,X ;GIVES
         STA SCRN+$02FF,X ;CATHY
         INX          ;AND I
         CPX #27      ;THE GLORY!
         BNE PUTCREDS
         LDX #23
PUTSCORE LDA MESSAGE1,X ;PUTS "SCORE"
         STA SCORELN+1,X ;AND "LEVEL"
         DEX          ;ON SCRN
         BNE PUTSCORE ;NOT DONE
         JSR PUTCHAR  ;REDEFINES CHAR. SET
         LDA # >CHARORG ;TELL ANTIC
         STA CHSET    ;WHERE NEW CHAR. SET IS
         LDA # <DLI   ;STORE LOW
         STA VDSLST   ;AND HIGH
         LDA # >DLI   ;BYTES OF DISPLAY
         STA VDSLST+1 ;LIST INTERRUPT
         LDA # <COVER ;DISPLAY LIST
         STA SDLSTL   ;OF TITLE SCRN
         LDA # >COVER
         STA SDLSTL+1
         LDA #192     ;ENABLE DISPLAY
         STA NMIEN    ;AND VB INTERRUPTS
         LDA #58      ;TURN SCRN
         STA DMA      ;BACK ON
         JSR SETSCORE ;PUT ZEROS IN SCORE
         LDA #1       ;START AT
         STA LEVEL    ;LEVEL ONE
         LDA #2       ;FALLING POPCORN
         STA LOADCHAR ;CHRS
         LDA #17      ;INTERNAL CODE
         STA SHOWLEV  ;FOR "0"
         LDA #3       ;3 LIVES
         STA LIVES
         STA SKCTL    ;INIT SOUND
         JSR SHOWLIVES ;PUT LIVES ON SCRN
;
;ALL SET, WANT TO PLAY A GAME?
;
WAIT     LDA CONSOL   ;START KEY
         CMP #6       ;PRESSED?
         BEQ INIT     ;YES!
         CMP #5       ;NO-HOW ABOUT SELECT?
         BNE DFD      ;NO,CHECK TRIGGER
         STA LEVEL    ;YES, START AT
         LDA #21      ;LEVEL 5
         STA SHOWLEV  ;SHOW IT!
DFD      LDA STRIG    ;TRIGGER PRESSED?
         BNE WAIT     ;NO DO IT AGAIN
INIT     LDA #30      ;DEBOUNCE
         JSR LONGPAUSE
INIT1    LDA CONSOL   ;
         CMP #6       ;START KEY RELEASED?
         BEQ INIT1    ;NO!
;
;SET UP GAME
;
         JSR CLRSCRN
         JSR WRITEPM  ;WRITE PLAYERS
         LDA #136     ;SHUT  OFF
         STA SONGXREG ;SONG
         STA SIZEP2
         LDA #64      ;DISABLE DLI'S
         STA NMIEN
         LDA #1       ;PRIORITY
         STA GPRIOR
         STA SIZEP1
         LDA #3       ;ENABLE PM
         STA GRACTL   ;GRAPHICS
         LDA # >PM    ;GET PM ADDRESS
         STA PMBASE   ;AND TELL ANTIC
         JSR SETDL    ;SET UP GAME DLI'S
         JSR WRITECOLORS ;SET UP COLORS
         LDA #119     ;CENTER OF SCRN
         STA HPOS
         STA HITCLR   ;CLR COLLISIONS
         LDY LEVEL    ;GET LEVEL
         LDA DELAYTABLE,Y AND DELAY NO.
         STA SETDELAY ;NEW SPEED OF GAME
;
;PRINT POT (BODY-LEFT SIDE)
;
         LDA #4       ;CHR VALUE
         LDX #18      ;X POSITION
         LDY #9       ;Y POSITION
         JSR PRINTCHR ;PRINT IT!
         LDA #6       ;NOW THE RIGHT
         LDX #20      ;SIDE
         LDY #9
         JSR PRINTCHR ;DITTO!
;
;ACTUAL GAME LOOP
;
GAME     LDA CONSOL   ;START KEYPRESSED?
         CMP #6
         BNE RR1      ;NO
ERT      LDA CONSOL   ;YES, NOW
         CMP #6       ;WAIT FOR
         BEQ ERT      ;RELEASE
         JMP START    ;GOTO TITLE PAGE
;
RR1      LDA CH       ;SPACE BAR PRESSED?
         BMI CONT     ;NOPE!
PAUSEING LDA #255     ;YES , LETS PAUSE
         STA CH       ;CLR KEYBOARD
         LDA PAUSEON  ;GET FLAG
         EOR $01      ;AND SWICH IT
         STA PAUSEON
         BEQ CONT     ;IF ZERO THEN CONT.
CHKPAUSE LDA CH       ;PAUSE UNTIL
         BMI CHKPAUSE ;TOUCHED AGAIN
         JMP PAUSEING
CONT     JSR DELAY    ;CONT. WITH GAME
         JSR CHARPOLL ;MOVE POPCORN
         LDA TIMER    ;LENGTH OF LID OPEN
         BNE WWQ      ;STILL COUNTING DOWN
         LDA RAND     ;MAYBE LETS POP SOME POPCORN
         CMP #20
         BCC AA       ;<20
         BCS WWA      ;>=20
WWQ      DEC TIMER
         BNE OUT4     ;STILL NOT READY TO POP
;
;PRINT LID DOWN  - LEFT SIDE
;
WWA      LDA #8       ;CHR VALUE
         LDY #1       ;Y POSISTION
         LDX #18      ;X POSITION
         JSR PRINTCHR ;PRINT
         LDA #10      ;RIGHT SIDE
         LDY #1
         LDX #20
         JSR PRINTCHR ;PRINT
OUT4     JMP OUT6
;
;PRINT  LID UP - RIGHT SIDE
;
AA       LDA #12      ;CHAR. VALUE
         LDX #18      ;X POS.
         LDY #1       ;Y POS.
         JSR PRINTCHR
         LDA #14      ;LEFT SIDE
         LDX #20
         LDY #1
         JSR PRINTCHR
         LDA #1       ;MAKE A VB
         STA SNDFLAGPOP ;"POP" SOUND
         LDA LEVEL    ;RESETS TIMER
         ASL A        ;SO THAT LID
         CLC          ;POPPING DOESN'T
         ADC #4       ;INCREASE TOO
         STA TIMER    ;MUCH IN SPEED
         LDX #0
CKCHAR   LDA POSITION,X ;ANY POPCORNS NEEDED?
         BEQ NN4      ;YES
         INX          ;NOT YET
         INX
         CPX #12      ;DONE CHECKING
         BEQ OUT6     ;YES,NONE NEEDED
         JMP CKCHAR   ;NO, CHECK AGAIN
;
NN4      STX TEMP     ;SAVE X REG
         LDA LEVEL    ;GET LEVEL
         ASL A        ;MULT. BY 2
         STA TEMP1    ;STORE IT
         LDX #1       ;GET POSITION
NNX      LDA POSITION,X
         BEQ NEXTONE  ;SKIP CHECK
         CMP TEMP1    ;LEVEL*2
         BCC OUT6     ;<LEVEL*2, DON'T POP
NEXTONE  INX          ;CHECK ALL
         INX          ;OTHER POPCORN
         CPX #13      ;DONE?
         BEQ NNX      ;YES!
         LDX TEMP     ;NO. POP SOME
NN5      LDY #10
         LDA #14
         JSR RANDOM   ;RND(0)*14+10
         CMP #16      ;MAKES SURE
         BCC NNZ      ;POP DOESN'T
         CMP #22      ;GET WIPED OUT
         BCC NN5      ;TRY AGAIN
;
NNZ      STA POSITION,X ;HORIZ. POS.
         STA SNDFLAGPOP ;"POP" SOUND
         LDA #12
         LDY #1
         JSR RANDOM   ;RND(0)*12+1
         INX
```

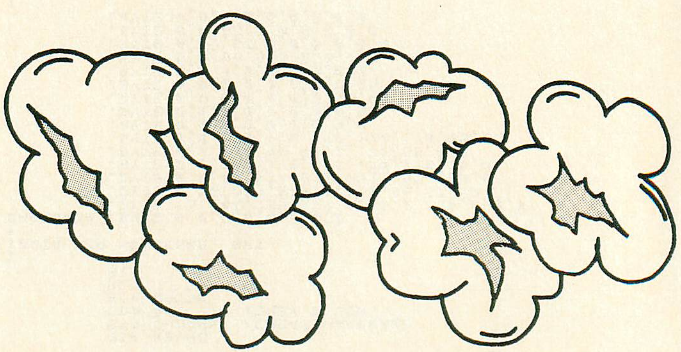
```
         STA POSITION,X ;VERT. POS.
         STX LASTPOP ;SAVE LAST POPCORN
OUT6     LDA P1PF    ;POPCORN CAUGHT?
         BNE NNØ     ;YES!
         JMP NN3     ;NOPE.
;
NNØ      STA CATCHSND ;MAKE SOUND(VB)
         JSR ERASECHR ;ERASE POPCORN
         LDY LEVEL   ;GET LEVEL
         CPY #6      ;6 OR HIGHER?
         BCC XXXX    ;NO
         LDY #5      ;YES,POINTS=5Ø
XXXX     JSR INCSCORE ;POINTS=LEVEL*1Ø
         STA HITCLR  ;CLR COLL.
         INC NUMBER  ;NUMBER OF CAUGHT CORN
         LDY LEVEL   ;LEVEL
         LDA NUMBER  ;GET NO. CAUGHT
         CPY #5      ;LEVEL 5 OR HIGHER?
         BCS FFFF    ;YES
         CMP #25     ;NO ,25 CAUGHT?
         BCS GGGG    ;YES!(NEXT LEVEL)
MMX      JMP NN3     ;A LONG WAY TO GO
;
FFFF     CMP #5Ø     ;5Ø CAUGHT?(NEXT LEVEL)
         BCC MMX     ;NO
;
;PRINT LID TO SCRN-DOWN
;
GGGG     LDA #8      ;LID DOWN
         LDY #1      ;Y POS
         LDX #18     ;X POS
         JSR PRINTCHR
         LDA #1Ø     ;2ND HALF OF LID
         LDY #1      ;SAME AS ABOVE
         LDX #2Ø
         JSR PRINTCHR
         LDA #6
         STA TEMP    ;TEMPORARY
BONUS    LDA #4Ø     ;SLOW IT DOWN
         STA SETDELAY
         JSR DELAY   ;BONUS GIVES
         LDA #175    ;25Ø POINTS
         STA TEMP1   ;IN 5 STEPS AND
         DEC TEMP    ;SOUNDS BELL
         LDA TEMP    ;EACH TIME
         BEQ GG1
         LDY #5      ;5Ø POINTS
         JSR INCSCORE ;ADDED EACH TIME
BELL     DEC TEMP1   ;RING BELL
         LDA TEMP1
         CMP #16Ø
         BCC BONUS
         STA AUDC1   ;MAKE BELL SOUND
         LDA #12     ;FREQUENCY
         STA AUDF1
         JSR PAUSE   ;PAUSE A LITTLE
         JMP BELL    ;NEXT SOUND
;
GG1      LDA #Ø
         STA NUMBER  ;Ø POPCORNS
         STA AUDC1   ;NO SOUND
         LDY LEVEL   ;GET LEVEL
         CPY #19     ;IF 19 THEN DON'T
         BEQ CHANGELEV ;INCREASE IT
         INC LEVEL   ;NEXT LEVEL
         LDY LEVEL
         CPY #4      ;LEVEL 3?
         BEQ GREETING ;SEE THE BOSS
         CPY #9      ;LEVEL 8
         BEQ GREETING ;SEE THE BOSS
         CPY #13     ;LEVEL 12
         BEQ GREETING ;SEE THE BOSS
         CPY #17     ;GUESS!
         BNE NOBOSS
GREETING STY TEMPYREG SAVE Y REG
         JSR ERASEPOPCORN ;CLR SCRN
         JSR SEEBOSS ;GET CONGRATULATED!
         LDA #58     ;TURN SCRN ON
         STA DMA
         JSR CLRSCRN
;
;PRINT POT BODY
;
         LDA #4      ;CHAR VALUE
         LDX #18     ;X POS
         LDY #9      ;Y POS
         JSR PRINTCHR ;PRINT IT!
         LDA #6      ;SAME AS ABOVE,
         LDX #2Ø     ;ONLY RIGHT
         LDY #9      ;SIDE OF POT
         JSR PRINTCHR ;DITTO!
;
;PRINT LID DOWN  - LEFT SIDE
;
         LDA #8      ;CHR VALUE
         LDY #1      ;Y POSISTION
         LDX #18     ;X POSITION
         JSR PRINTCHR ;PRINT
         LDA #1Ø     ;RIGHT SIDE
         LDY #1
         LDX #2Ø
         JSR PRINTCHR ;PRINT
         LDA #6Ø
         JSR LONGPAUSE ;PAUSE ONE SEC
         LDY TEMPYREG
NOBOSS   CPY #19     ;LEVEL 19?
         BEQ DS      ;YES. NO MORE TESTS
         CPY #1Ø     ;1Ø?
         BNE DS      ;NO
         LDA #16     ;YES, NEED TO
         STA SHOWLEV ;CHANGE 2ND DIGIT
         LDA #17     ;TO Ø AND
         STA SHOWLEV-1 ;PUT 1 IN FIRST
         JMP CHANGELEV
;
DS       INC SHOWLEV ;INCREASE LEVEL SHOWN
CHANGELEV LDA DELAYTABLE,Y ;GET NEW SPEED
         STA SETDELAY ;OF GAME
NN3      LDA MISSPOP ;MISSED THREE
         CMP #3      ;POPCORN
         BCS TT1     ;YEP. UH-OH!
         JMP GAME    ;NO GO ON PLAYING
TT1      JSR LOSECHANCE ;GET PULLED OFF SCRN
         LDA #Ø      ;RESET HOW MANY
         STA MISSPOP ;POPCORN MISSED
         DEC LIVES   ;ONE LESS CHANCE
         LDA LIVES   ;ANY LIVES LEFT?
         BEQ TT2     ;NO!
         JSR SHOWLIVES ;SHOW THEM
         LDA #119    ;CENTER HAND
         STA HPOS
         JSR LONGPAUSE ;PAUSE
         JMP GAME    ;PLAY GAME
;
TT2      LDA #1      ;TELL SEEBOSS
         STA BOSSFLAG ;THAT YOU LOST
         JSR SEEBOSS ;SE HIM!
         JMP START   ;START GAME OVER
;
;GAME ROUTINES
;
SETSCORE LDA #16     ;ZEROS SCORE
         LDX #Ø
ZSCORE   STA SCORELN+9,X ;PUT SIX
         INX         ;ZERO'S
         CPX #6      ;IN SCORELN
         BEQ GGG
         JMP ZSCORE  ;"Ø" IN NEXT POS.
;
GGG      RTS         ;ALL DONE
;
;INCREASES SCORE
;
INCSCORE LDA SCORELN+1Ø ;GET
         STA TEMP2   ;1ØØØØ AND STORE IT
INCSCOR1 LDX #4      ;5 POS.
LL       LDA SCORELN+9,X ;GET NO.
         CMP #25     ;FOR EACH POS.
         BNE FF      ;AND CHECK FOR 9
         LDA #16     ;ZERO NUMBER
         STA SCORELN+9,X
         DEX         ;NEXT POSITION
         JMP LL      ;AGAIN
;
FF       INC SCORELN+9,X ;ADD 1
         DEY         ;POINTS ADDED?
         BNE INCSCOR1 ;NO!
         LDA TEMP2   ;PASSED 1ØØØØ
         CMP SCORELN+1Ø ;POINTS?
         BEQ QW      ;NOPE. RETURN
;
;GAIN A ANOTHER CHANCE!
;
         LDA LIVES   ;GET LIVES
         CMP #5      ;5 YET?
         BEQ QW      ;YES. NO MORE
         INC LIVES   ;GAIN A LIFE
         JSR SHOWLIVES ;SHOW IT
         LDA #192    ;TELL PLAYER
         STA SONGXREG ;BY PLAYING SONG
QW       RTS
;
SHOWLIVES LDX #5     ;5 POSITIONS
         LDA #Ø
CLRLIFE  STA SHOWLIFE,X ;CLR
         DEX         ;LIFES OUT FIRST
         BPL CLRLIFE
         LDX LIVES   ;GET LIVES
         LDA #65     ;POPCORN CHAR.
NST      STA SHOWLIFE,X ;SHOW THEM
         DEX
         BNE NST     ;DONE?
         RTS         ;YES
;
;UH-OH! YOU BLEW IT!
;
LOSECHANCE LDA #168  ;PLAY DEFEAT
         STA SONGXREG ;SONG
         JSR ERASEPOPCORN ;CLR SCRN
         LDA #8      ;LID DOWN
         LDY #1
         LDX #18
         JSR PRINTCHR
         LDA #1Ø     ;2ND HALF OF LID
         LDY #1
         LDX #2Ø
         JSR PRINTCHR
         LDA #2Ø
         JSR LONGPAUSE ;PAUSE A LITTLE
         LDA #159    ;159 PIXELS
         STA TEMP2   ;STORE
         LDA #2Ø7    ;PLAYER STARTING
         STA HPOSCANE ;POSITION
         LDX #39     ;4Ø BYTES
         LDY #Ø
PUSHCANE JSR PAUSE   ;SLOW IT DOWN
         LDA VALUE,Y ;VALUES FOR CANE
         STA CANESCRN,X ;POKE SCRN
         DEC HPOSCANE
         LDA HPOSCANE ;MOVE
         STA HPOSP2  ;PLAYER
         LDA HPOS    ;GET HAND POS.
         SEC
         SBC #42     ;SUBTRACT 42
         DEC TEMP2   ;AT LOCATION
         CMP TEMP2   ;OF MAN YET?
         BCS PULLBACK ;YES!
         INY         ;NO. DONE WITH
         CPY #4      ;BYTE?
         BCC PUSHCANE ;NO!
         LDY #Ø      ;NEXT BYTE
         DEX
         JMP PUSHCANE ;PUSH CANE
;
PULLBACK INC HPOSCANE ;MOVE HOOK
PULLING  LDA CANESCRN,X
         BEQ OFF     ;IF DONE SHIFTING
         LSR A       ;MOVE CANE ONE POS.
         LSR A
         STA CANESCRN,X ;PUT ON SCRN
         JSR PAUSE
         INC HPOSCANE
         LDA HPOSCANE
         STA HPOSP2  ;MOVE HOOK
         SEC
         SBC #6      ;THEN MOVE
         STA HPOS    ;HAND
         JMP PULLING
;
OFF      INX
         CPX #41     ;4Ø BYTES PER SCRN
         BCC PULLING ;NOT DONE YET
         LDA HPOS    ;SAVE HAND POS
         STA TEMP1
OFFSCRN  LDA TEMP1   ;NOW WE
         STA HPOS    ;NEED TO
         CMP #245    ;PULL HAND & HOOK
         BCS ER      ;OFF OF SCRN
         CLC
         ADC #6
         STA HPOSP2  ;HOOK HORIZ POS
         INC TEMP1
         JSR PAUSE
         JMP OFFSCRN
;
ER       RTS
;
;CLR PLAYER MEMORY
;
CLRPM    LDA # <PM+1Ø24 ;LOW
         STA CLR
         LDA # >PM+1Ø24 ;HIGH
         STA CLR+1   ;BYTES
         LDX #5      ;5 PAGES
         JMP CLRØ    ;CLR THEM!
;
;PUT PLAYERS INTO MEMORY
;
WRITEPM  LDA PMDATAØ,X ;GET
         STA PMØ,X   ;PLAYER
         LDA PMDATA1,X ;DATA
         STA PM1,X   ;AND PUT
         LDA PMDATA2,X ;INTO P/M
         STA PM2,X   ;MEMORY
         INX
         CPX #15     ;15 BYTES
         BNE WRITEPM
;
;FILL COLOR REGS.
;
WRITECOLORS LDX #7   ;SET UP
COLORS   LDA COLORTABLE,X
         STA PCOLRØ,X ;COLORS
         DEX
         BPL COLORS
         RTS
;
;CLR GAME SCRN
;
CLRSCRN  LDA # <SCRN
         STA CLR     ;
         LDA # >SCRN ;CLR
         STA CLR+1   ;SCRN FOR PLAY
         LDX #15
         JMP CLRØ
;
COLORTABLE .BYTE 58,72,15,Ø,4Ø,14,Ø
;
;SET UP DL AND DLI'S
;
```

```
SETDL      LDA # <DLIST ;TELL ANTIC
           STA SDLSTL   ;WHERE DISPLAY
           LDA # >DLIST ;LIST IS
           STA SDLSTL+1
           LDA # <DLI1  ;AND WHERE
           STA VDSLST   ;DISPLAY LIST
           LDA # >DLI1  ;INTERRUPT IS
           STA VDSLST+1
           LDA #1       ;1/60 OF A SEC.
           JSR LONGPAUSE ;WAIT UNTIL
           LDA #192     ;NEXT VBI TO
           STA NMIEN    ;ENABLE DLI
           RTS
;
;LOAD REDEFINED CHR SET
;
PUTCHAR    LDX #0
PUTCHAR1   LDA CHORG,X ;LOAD
           STA CHARORG,X ;CHR
           LDA CHORG+$0100,X ;DATA AND PUT
           STA CHARORG2,X ;INTO
           LDA CHORG+$0200,X ;REDEFINED
           STA CHARORG3,X ;CHR SET
           INX
           BNE PUTCHAR1
REDEFINE   LDA CHARDATA,X ;NOW
           STA CHARORG3,X ;CHANGE
           INX          ;CHRS TO
           CPX #161     ;WHAT WE WANT
           BNE REDEFINE
;
;
DELAY      LDY #0
           LDX #32      ;SLIGHT DELAY
DELAY1     INY          ;USED MAINLY
           CPY SETDELAY ;TO TIME
           BNE DELAY1   ;FALLING POPCORN
           LDY #0
           DEX
           BNE DELAY1
           RTS
;
DELAYTABLE .BYTE 0,200,150,120,100,80,70,65,60
           .BYTE 55,50,45,40,35,30,25,22,18,13,10
;
PAUSE      TYA          ;A LONGER PAUSE
           PHA          ;THAT SAVES THE
           LDA #0       ;Y REG
           TAY
PAUSE1     CLC
           ADC #1
           CMP #22
           BEQ DONE7
PAUSE2     INY
           BNE PAUSE2
           JMP PAUSE1
;
DONE7      PLA
           TAY
           RTS
;
;JIFFY TIMER
;
LONGPAUSE STA TEMP      ;A REG
           LDA #0       ;CONTAINS JIFFIES
           STA 20       ;(1/60 SEC) FOR
WAIT1      LDA 20       ;LENGTH OF PAUSE
           CMP TEMP
           BNE WAIT1
           RTS
;
;GET A RANDOM NUMBER
;
RANDOM     STA TEMPA
           LDA RAND     ;RANDOM NUMBER
           AND TEMPA    ;KNOCK OFF
           STY TEMPA    ;THEN
           CLC          ;THEN ADD
           ADC TEMPA    ;SPECIFIED NUMBER
           RTS
;
;TAKE POPCORN OFF OF SCRN
;
ERASEPOPCORN LDA #15 ;BLANK CHAR
           STA LOADCHAR
           JSR CHARPOLL ;ALL OFF SCRN!
           LDA #0
           TAX
LOOP9A     STA POSITION,X ;NOW CLR
           INX          ;POS. TABLE
           CPX #12
           BCC LOOP9A
           LDA #2
           STA LOADCHAR
           RTS          ;ALL DONE
;
;YOU GET TO SEE THE BOSS!
;
SEEBOSS
           LDA #64      ;DISABLE DLI'S
           STA NMIEN
           JSR CLRPM
           LDA # <SCRN ;CLR SCRN
           STA CLR
           LDA # >SCRN
           STA CLR+1
           LDX #5
           JSR CLR0
           LDX #48      ;CHANGE PLAYER 2
           LDA #126     ;TO DOOR
WRITEDOOR STA PM+1694,X
           DEX
           BPL WRITEDOOR
           LDX #0
REDEFINE1 LDA BOSSCHAR1,X ;ALTERNATE
           STA CHARORG,X ;CHR
           INX          ;SET
           CPX #48
           BNE REDEFINE1
           LDX #0
REDEFINE2 LDA BOSSCHAR2,X
           STA CHARORG2,X
           INX
           CPX #152
           BNE REDEFINE2
           LDA #1       ;CHANGE PRIORITY
           STA GPRIOR
           LDA #3       ;PLAYER 2 SIZE
           STA SIZEP2
           LDA #228     ;PLAYER2 COLOR
           STA PCOLR2
           LDA #136
           STA COLOR1
           LDA BOSSFLAG
           BNE DONTCHANGE
           LDA #13      ;RND(0)*13+1
           LDY #1
           JSR RANDOM
           ASL A        ;*16
           ASL A
           ASL A
           ASL A
           CLC
           ADC #14
           STA COLOR2   ;COLOR OF LETTERS
DONTCHANGE LDA # <BOSSDLIST
           STA SDLSTL   ;NEW DISPLAY
           LDA # >BOSSDLIST ;LIST
           STA SDLSTL+1
           LDA # <BOSSDLI
           STA VDSLST   ;AND DLI
           LDA # >BOSSDLI
           STA VDSLST+1
           LDA #192     ;ENABLE INTERR.
           STA NMIEN
           LDA # <SCRN ;GET FIRST
           STA TEMP     ;BYTE OF SCRN
           LDA # >SCRN
           STA TEMP+1
           INC TEMP+1   ;ADD 255
           CLC
           LDA TEMP
           ADC #93      ;+93 =
           STA TEMP     ;SCRN+348
           BCC DC1
           INC TEMP+1
DC1        LDA BOSSFLAG
           BNE LOSEFACE
FINDNUM    LDA RAND     ;GET MESSAGE
           CMP #6       ;RANDOM NO.
           BCS FINDNUM
           TAX
           LDA BOSSTABLE,X ;FIRST BYTE
           STA BOSSXREG ;OF MESSAGE
           LDA #37      ;HAPPY FACE
           JMP STOREFACE
;
LOSEFACE LDA #0         ;SAD SONG
           STA BOSSXREG
           STA COLOR2   ;BLACK MESSAGE
           LDA #41      ;SAD FACE
STOREFACE STA FACE
           LDX #0
LOOPX      LDY #0       ;PUT CHRS
           LDA BOSS1,X ;THAT MAKE
           STA (TEMP),Y ;UP BOSS ON
           INX          ;SCRN
           INY          ;(2 WIDE*9 HIGH)
           LDA BOSS1,X ;NEXT CHAR
           STA (TEMP),Y ;PUT ON SCRN
           CLC
           LDA TEMP
           ADC #20      ;20 BYTES PER LINE
           STA TEMP     ;(ADD TO OFFSET)
           BCC LPX
           INC TEMP+1
LPX        INX
           CPX #10      ;9 HIGH
           BNE LOOPX
           LDX #0
FRAMEDOOR LDA #1
           STA SCRN+348,X ;LEFT SIDE
           LDA #2
           STA SCRN+351,X ;RIGHT SIDE
           CLC
           TXA
           ADC #20      ;20 BYTES PER LINE
           TAX
           CPX #100     ;DONE?
           BNE FRAMEDOOR ;NO!
           LDA #3       ;NOW FRAME IN
           STA SCRN+329 ;TOP OF DOOR
           STA SCRN+330
           LDA #4
           STA SCRN+328
           LDA #5
           STA SCRN+331
           LDA #112
           STA HPOSP2   ;CENTER DOOR
           JSR LONGPAUSE
           LDA #100     ;MAKE MECH.
           STA AUDC1    ;SOUND FOR DOOR
           STA AUDC2    ;BEAT VOICES
           LDA #24      ;ONE AND TWO
           STA AUDF1    ;TOGETHER
           LDA #25
           STA AUDF2
           LDA #113
           STA TEMP1
OPEN       JSR PAUSE
           LDA TEMP1
           STA HPOSP2   ;MOVE DOOR
           CMP #135     ;DONE?
           BEQ DONEA    ;YES!
           JSR PAUSE
           INC TEMP1
           JSR PAUSE
           JMP OPEN
;
DONEA      LDA #0       ;SHUT OFF SOUND
           STA AUDC1
           STA AUDC2
           LDA #30
           JSR LONGPAUSE
           LDA BOSSFLAG
           BNE PLAYSAD ;WHICH SONG?
           LDA #138     ;HAPPY!
           BNE SONGSTART
;
PLAYSAD    LDA #153     ;SAD
SONGSTART STA SONGXREG ;NOW PLAY
           LDY FACE     ;SONG AND PUT
           STY SCRN+349 ;CORRECT
           INY          ;FACE ON SCRN
           STY SCRN+350
           INY          ;EACH FACE MADE
           STY SCRN+369 ;UP OF FOUR
           INY          ;CHRS
           STY SCRN+370
           LDX BOSSXREG ;WHICH MESSAGE?
           LDY #0
PRINT1     LDA MESSAGE8,X
           STA SCRN+464,Y ;PUT ON SCRN
           CMP #129     ;INVERSE !
           BEQ GOBACK   ;NOT DONE!
           INX
           INY
           JMP PRINT1
;
GOBACK     LDA #255
           JSR LONGPAUSE
           LDA #0
           STA DMA      ;SHUT OFF DMA
           LDA #1       ;(CLEANER SWICHING)
           JSR LONGPAUSE ;PAUSE 1 JIFFY
;
;RESET TO ORIGINAL
;
           LDA #0       ;CLR VARIABLES
           STA HPOSP2   ;DOOR HORIZ
           STA SIZEP2
           STA BOSSFLAG
           LDA #64      ;DISABLE
           STA NMIEN    ;DLI'S
           JSR CLRPM    ;PUT PROPER
           JSR WRITEPM ;PLAYERS BACK
           JSR PUTCHAR ;AND CHAR SET
           JSR WRITECOLORS
           JSR SETDL    ;SET DISPLAY LIST
           LDX TEMPXREG ;GET BACK X AND
           LDY TEMPYREG ;Y REGS
           RTS
;
TEMPXREG .BYTE 0
TEMPYREG .BYTE 0
BOSS1      .BYTE 33,34,35,36,109,110
           .BYTE 111,112,113,114,1,1
;
;MESSAGES WHEN SEEING THE BOSS
;
MESSAGE8 .SBYTE +$80,"YOU'RE FIRED!"
           .SBYTE +$80," GREAT JOB! "
           .SBYTE +$80," WAY TO GO!"
           .SBYTE +$80,"   WOWEE! "
           .SBYTE +$80," FANTASTIC!"
           .SBYTE +$80,"LIKE, AWESOME!"
           .SBYTE +$80,"  GOOD WORK!"
```

```
BOSSXREG .BYTE 0
BOSSFLAG .BYTE 0      ;HAPPY OR SAD BOSS
BOSSTABLE .BYTE 13,25,36,46,57,71
FACE      .BYTE 0      ;HAPPY OR SAD
VALUE     .BYTE 1,5,21,85
;
;THEME SONG DATA
;
SONG      .BYTE 121,4,96,4,162,4,144,2
          .BYTE 128,2,121,4,96,4,162,4
          .BYTE 144,2,128,2,121,4,96,4
          .BYTE 121,4,96,4,108,4,128,4
          .BYTE 162,4,144,2,128,2,121,4
          .BYTE 96,4,162,4,144,2,128,2
          .BYTE 121,4,96,4,162,4,144,2
          .BYTE 128,2,121,4,96,4,108,4
          .BYTE 96,4,121,8
          .BYTE 162,4,128,4,217,4,193
          .BYTE 2,173,2,162,4,128,4
          .BYTE 217,4,193,2,173,2,162,4
          .BYTE 128,4,162,4,128,4,144,4
          .BYTE 173,4,217,4,193,2,173,2
          .BYTE 162,4,128,4,217,4,193,2
          .BYTE 173,2,162,4,128,4,217,4
          .BYTE 193,2,173,2,162,4,128,4
          .BYTE 144,4,128,4,162,8,0,0
;
;SONG VALUES WHEN CONGRADULATED!
;
          .BYTE 60,2,53,2,47,4,60,4,53
          .BYTE 4,64,4,60,8,0
;
;SONG  WHEN YOU LOST THE GAME!
;
          .BYTE 81,2,81,6,81,8,108,4
          .BYTE 96,6,121,16,243,16,0
;
;SONG WHEN YOU LOSE 3 CHANCES
;
          .BYTE 243,8,217,4,204,12,217
          .BYTE 12,243,2,255,2,243,2,255,2
          .BYTE 243,8,217,4,243,8,0,0
;
;DIDDLE WHEN A CHANCE IS GAINED
;
          .BYTE 33,2,37,2,33,2,37,2,0,0
;POPCORN! IN BIG LETTERS
;
POPCORNSCRN
          .BYTE 0,0,65,65,65,65,0,65
          .BYTE 65,65,65,0,65,65,65,65
          .BYTE 0,65,65,65,65,0,65,65
          .BYTE 65,65,0,65,65,65,65,0
          .BYTE 65,0,0,65,0,65,0,0
          .BYTE 0,0,65,0,0,65,0,65
          .BYTE 0,0,65,0,65,0,0,65
          .BYTE 0,65,0,0,0,0,65,0
          .BYTE 0,65,0,65,0,0,65,0
          .BYTE 65,0,0,65,0,65,0,0
          .BYTE 0,0,65,0,0,65,0,65
          .BYTE 0,0,65,0,65,0,0,65
          .BYTE 0,65,0,0,0,0,65,0
          .BYTE 0,65,0,65,0,0,65,0
          .BYTE 65,0,0,65,0,65,0,0
          .BYTE 0,0,65,65,65,65,0,65
          .BYTE 0,0,65,0,65,65,65,65
          .BYTE 0,65,0,0,0,0,65,0
          .BYTE 0,65,0,65,65,65,0,0
          .BYTE 65,65,0,65,0,65,0,0
          .BYTE 0,0,65,0,0,0,0,65
          .BYTE 0,0,65,0,65,0,0,0
          .BYTE 0,65,0,0,0,0,65,0
          .BYTE 0,65,0,65,0,0,65,0
          .BYTE 65,0,65,65,0,65,0,0
          .BYTE 0,0,65,0,0,0,0,65
          .BYTE 0,0,65,0,65,0,0,0
          .BYTE 0,65,0,0,0,0,65,0
          .BYTE 0,65,0,65,0,0,65,0
          .BYTE 65,0,0,65,0,0,0,0
          .BYTE 0,0,65,0,0,0,0,65
          .BYTE 65,65,65,0,65,0,0,0
          .BYTE 0,65,65,65,65,0,65,65
          .BYTE 65,65,0,65,0,0,65,0
          .BYTE 65,0,0,65,0,65,0,0
;
MESSAGE   .SBYTE "By Mark and "
          .SBYTE "Cathy Sloatman"
;
MESSAGE1 .SBYTE " SCORE                LEVEL"
;
;DISPLAY LISTS
;
;TITLE PAGE
;
COVER     .BYTE 112,112,66
          .WORD SCRN
          .BYTE 2,2,2,2,2,2,2,2
          .BYTE 2,2,2,2,130,2,2,2
          .BYTE 2,2,2,2,2,65
          .WORD COVER
;
;SCRN WHEN SEEING BOSS
;
BOSSDLIST .BYTE 112,112,112,70
          .WORD SCRN
          .BYTE 6,6,6,6,6,6,6,6
          .BYTE 6,6,6,6,6,6,6,6
          .BYTE 6,6,6,6,134,6
          .BYTE 6,112,65
          .WORD BOSSDLIST
;
;GAME SCRN
;
DLIST     .BYTE 112,112,194
          .WORD SCORELN
          .BYTE 77
          .WORD SCRN
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,13,13
          .BYTE 13,13,13,13,13,13,141,13
          .BYTE 13,13,13,141,13,13,141,65
          .WORD DLIST
;
;DISPLAY LIST INTERRUPTS
;
;CHANGE SCORELINE COLOR
;
DLI1      PHA
          LDA #218
          STA WSYNC    ;CLEAN BREAK
          STA COLPF2   ;NEW COLOR
          LDA # <DLI2  ;NEXT DLI
          STA VDSLST
          LDA # >DLI2
          STA VDSLST+1
          PLA
          RTI
;
;CHANGE POT COLOR
;
DLI2      PHA
          LDA #228
          STA WSYNC    ;CLEAN BREAK
          STA COLPM1   ;POT
          LDA PCOLR2   ;CANEHEAD COLOR
          STA COLPF0   ;PUT IN CANE
          LDA # <DLI3  ;NEW DLI
          STA VDSLST
          LDA # >DLI3
          STA VDSLST+1
          PLA
          RTI
;
;PUT A CUFF ON THE ARM
;
DLI3      PHA
          LDA #10
          STA WSYNC
          STA COLPM0   ;CUFF WHITE
          LDA #4       ;MAKE CANE PASS
          STA PRIOR    ;BEFORE PLAYER
          LDA # <DLI4  ;NEW DLI
          STA VDSLST
          LDA # >DLI4
          STA VDSLST+1
          PLA
          RTI
;
;MAKE GREEN TABLE
;
DLI4      PHA
          LDA #200
          STA WSYNC
          STA COLBK    ;BACKGRND
          LDA # <DLI1  ;NEW DLI
          STA VDSLST
          LDA # >DLI1
          STA VDSLST+1
          PLA
          RTI
;
;DLI FOR TITLE SCRN
;
DLI       PHA
          LDA #224     ;DEFAULT CHARSET
          STA WSYNC    ;CLEAN BREAK
          STA CHBASE   ;SWITCH CHAR SET
          LDA #10      ;NEW CHAR. COLOR
          STA COLPF1
          LDA #0       ;BACKGRND
          STA COLPF2
          PLA
          RTI
;
;FOR BOSS'S SCRN
;
BOSSDLI   PHA
          LDA #224
          STA WSYNC
          STA CHBASE   ;CHBAS HARDWARE
          LDA #8       ;GREY FLOOR
          STA COLBK
          PLA
          RTI
;
;DATA FOR NEW CHAR. SET
;
CHARDATA .BYTE 0,0,0,0,0,0,0,0
          .BYTE 96,246,255,127,62,120,120,0,0,0
          .BYTE 0,56,239,239,42,10,46,47,10
          .BYTE 0,0,40,174,250,232,192,128,0
          .BYTE 0,0,170,170,170,170,170,85,0
          .BYTE 0,0,170,40,40,0,170,85,0
          .BYTE 0,0,170,170,170,170,170,85,0
          .BYTE 0,0,0,0,0,0,0,0,0
          .BYTE 0,0,0,0,0,0,0,0,85
          .BYTE 0,0,0,0,0,0,0,40,85
          .BYTE 0,0,0,0,0,0,0,0,85
          .BYTE 0,0,0,0,0,0,0,0,0
          .BYTE 0,0,85,0,0,0,0,0,0
          .BYTE 0,40,85,0,0,0,0,0,0
          .BYTE 0,0,85,0,0,0,0,0,0
          .BYTE 0,0,0,0,0,0,0,0,0
;
;FOR BOSS'S CHAR. SET
;
          .BYTE 0,0,0,0,0,0,0,0,0
BOSSCHAR1 .BYTE 0,247,247,247,0,127,127,127
          .BYTE 56,248,248,248,56,120,120,120
          .BYTE 28,31,31,31,28,31,31,31
          .BYTE 255,255,0,0,0,0,0,0
          .BYTE 127,255,252,252,120,120,120,120
          .BYTE 254,255,63,63,31,31,31,31
BOSSCHAR2 .BYTE 0,0,0,0,0,0,0,0
          .BYTE 0,7,15,25,63,61,125,127
          .BYTE 0,224,240,152,252,188,190,254
          .BYTE 63,60,31,15,7,3,3,0
          .BYTE 252,60,248,240,224,192,192,0
          .BYTE 0,7,13,27,55,61,125,127
          .BYTE 0,224,176,216,236,188,190,254
          .BYTE 59,60,31,15,7,3,3,0
          .BYTE 220,60,248,240,224,192,192,0
          .BYTE 0,7,15,27,61,62,125,125
          .BYTE 0,224,240,216,188,124,190,190
          .BYTE 63,60,27,15,7,3,3,0
          .BYTE 252,60,216,240,224,192,192,0
          .BYTE 63,127,127,111,111,111,111,111
          .BYTE 252,254,254,246,246,246,246,246
          .BYTE 96,111,15,110,110,14,14,14
          .BYTE 6,246,240,118,118,112,112,112
          .BYTE 14,14,14,14,14,14,30,62
          .BYTE 112,112,112,112,112,112,120,124
;
;VERTICAL BLANK INTERRUPT ROUTINE
;
SNDVALUE .BYTE 142,9,138,13,134,17,130,21
;
VBROUTINE CLD
          LDA LENGTH   ;LENGTH OF NOTE
          BEQ NEXTNOTE ;TIME TO PLAY NEXT NOTE
          DEC LENGTH   ;DEC. TIMER
          JMP END1     ;SKIP THE REST
;
NEXTNOTE LDA SNGFLAG ;NOTE OR SILENCE
          EOR #1       ;FLIP IT
          STA SNGFLAG
          BNE SNDOFF   ;SILENCE
          LDX SONGXREG ;GET NOTE P
          LDA SONG,X   ;THEN NOTE
          BEQ END1     ;IF 0 DON'T PLAY
          STA AUDF3    ;AND SOUND OFF
          LDA #170     ;PURE TONE
          STA AUDC3    ;IN CONTROL
          INX          ;GET LENGTH
          LDA SONG,X   ;OF NOTE
          STA LENGTH   ;AND STORE IT!
          INX
          STX SONGXREG ;UPDATE NOTE POS.
          JMP END1     ;DONE
;
SNDOFF    LDA #0       ;SHUT OFF SOUND
          STA AUDC3    ;( REST)
          LDX SONGXREG ;POS. IN TABLE
          DEX          ;GET LENGTH
          LDA SONG,X   ;( SAME AS PREVIOUS
          STA LENGTH   ;NOTE)
END1      LDA STICK0   ;STICK POS.?
          CMP #7       ;RIGHT!
          BEQ FORWARD
          CMP #11      ;YOU GUESSED IT!
          BNE DONEX    ;IF NOT-DON'T MOVE
          LDA HPOS     ;POS. OF HAND
          CMP #50      ;TOO FAR LEFT?
          BCC DONE     ;YES!
          LDY LEVEL    ;GET LEVEL
          CPY #5       ;IF UNDER 5,
          BCC SLOWER   ;NEED TO SLOW IT
          DEC HPOS     ;DOWN
SLOWER    DEC HPOS     ;MOVE TWICE
          DEC HPOS
          JMP DONE
;
```

```
FORWARD   LDA HPOS
          CMP #195     ;TOO FAR RIGHT?
          BCS DONE     ;YES!
          LDY LEVEL    ;SAME AS ABOVE
          CPY #5
          BCC SLOWER1
          INC HPOS
SLOWER1   INC HPOS
          INC HPOS
DONE      LDA #Ø
          STA ATRACT   ;KILL ATTRACT MODE
DONEX     LDA HPOS
          STA HPOSP1   ;POT
          CLC
          ADC #5
          STA HPOSPØ   ;HAND PART TWO
          LDA SNDFLAGPOP ;POPCORN POPING?
          BEQ DONE9    ;NO!
          INC DELAYPOP ;YES, INC. COUNTER
          LDA DELAYPOP ;NEXT SOUND
          CMP #2       ;EVERY 2 VBI'S
          BNE DONE9    ;READY
          LDX XFLAGPOP ;GET NEXT VALUE
          LDA SNDVALUE,X
          STA AUDC2    ;SOUND 2
          INX          ;GET NEXT
          LDA SNDVALUE,X ;FREQUENCY
          STA AUDF2    ;VALUE
          LDA #Ø       ;RESET TIMER
          STA DELAYPOP
          INX
          STX XFLAGPOP ;NEXT VALUE
          CPX #8       ;DONE WITH POP?
          BNE DONE9    ;NOPE.
          LDA #Ø       ;RESET
          STA AUDC2    ;SOUND
          STA XFLAGPOP ;NEXT VALUE
          STA SNDFLAGPOP ;'POP' FLAG
DONE9     LDA CATCHSND ;ANY CAUGHT?
          BEQ NOTCAUGHT ;NOPE!
          INC SNDDELAY ;YES!
          LDA SNDDELAY ;EVERY TWO VBI'S
          CMP #1
          BNE NOTCAUGHT
          LDA CATCHSND1 ;IF IN MIDDLE OF
          BNE RR       ;POP, START
          LDA #24Ø     ;ONE
          STA FREQ
RR        LDA #17Ø     ;PURE TONE
          STA AUDC1
          LDA FREQ     ;NEXT VALUE
          STA AUDF1
          BEQ DONE2    ;DONE? YES.
          SEC          ;NEXT NOTE
          SBC #4Ø
          STA FREQ
          LDA #Ø       ;RESET DELAY
          STA SNDDELAY
          LDA #1       ;SHOW SOUND IS
          STA CATCHSND1 ;IN PROGRESS
          BNE NOTCAUGHT ;ALL DONE!
;
DONE2     LDA #24Ø     ;GET SOUND
          STA FREQ     ;READY FOR NEXT
          LDA #Ø       ;CATCH
          STA CATCHSND
          STA CATCHSND1
          STA SNDDELAY
NOTCAUGHT JMP XITVBV
;
INITVBLANK
          LDA #7       ;DEFFERRED VBI
          LDX # >VBROUTINE ;ADDRESS OF
          LDY # <VBROUTINE ;VBI PROGRAM
          JMP SETVBV   ;SET IT UP
;
;PLOT CHRS IN GR. 7
;
CHARPOLL  LDA #Ø
          STA XREG     ;KEEPS TRACK OF
CHKPOS    LDX XREG     ;CURRENT POPCORN
          LDA POSITION,X ;CHECKS
          BNE CHKMOVE  ;TO SEE
          INX          ;IF POPCORN
          INX          ;IS ON SCRN
          STX XREG     ;IF SO, MOVE
          CPX #12      ;IT DOWN
          BCC CHKPOS
          RTS
;
CHKMOVE   LDA POSITION,X ;TABLE
          STA HORIZ    ;DRIVEN HORIZ
          INX          ;AND VERT VALUES
          LDA POSITION,X
          STA VERT
          CMP #97      ;FALLEN OFFSCRN?
          BCC MOVEIT   ;NO!
          DEX          ;YES. PUT OUT OF PLAY
          LDA #Ø
          STA POSITION,X
          INX
          STA POSITION,X
          INC MISSPOP  ;ONE MORE MISSED
          BNE MOVED    ;DON'T PUT BACK IN PLAY
MOVEIT    INC POSITION,X :NEXT POS
MOVED     INX
          STX XREG     ;SAVE POPCORN POS
          LDA LOADCHAR ;POPCORN CHAR.
          STA CHR
          JSR PRINTCHAR
          LDX XREG     ;CHECK NEXT
          CPX #12      ;POSITION
          BCC CHKPOS
          RTS
;
;USED TO PRINT TO GR.7 SCRN
;
PRINTCHR  STA CHR      ;VARIABLES
          STY VERT
          STX HORIZ
PRINTCHAR LDA #1       ;1ST OR 2ND PART?
          STA CHECK    ;2 CHAR. PER POPCORN
          LDA # >CHARORG3 ;CTRL CHAR
          STA CHAR+1
PRINT     LDA CHECK
          BNE AAA
          INC HORIZ    ;2ND PART
          INC CHR
AAA       LDA #Ø
          STA CHAROFS+1 :RESET
          STA PLUS     ;VARIABLES
          STA OFFSET+1
          STA TEMP+1
          LDA VERT     ;VERTICAL VALUE
          ASL A        ;*2
          ROL TEMP+1   ;SAVE CARRY
          ASL A        ;*4
          ROL TEMP+1
          ASL A        ;*8
          ROL TEMP+1
          LDX TEMP+1
          STX OFFSET+1 ;SAVE HIGH&LOW
          STA OFFSET   ;BYTES
          ASL A        ;*16
          ROL TEMP+1
          ASL A        ;*32
          ROL TEMP+1
          CLC
          ADC OFFSET   ;ADD VERT*8
          STA OFFSET
          LDA TEMP+1   ;TO GET VERT*4Ø
          ADC OFFSET+1
          STA OFFSET+1
          LDA OFFSET
          CLC
          ADC HORIZ    ;SIMPLY ADD
          STA OFFSET   ;HORIZ. VALUE
          BCC BBB      ;OVERFLOW? NO.
          INC OFFSET+1
BBB       CLC
          LDA OFFSET
          ADC # <SCRN  ;ADD IN
          STA OFFSET   ;MEMORY LOC OF
          LDA # >SCRN  ;FIRST BYTE
          ADC OFFSET+1 ;ON SCRN
          STA OFFSET+1
          LDA #Ø
          STA TEMP+1
          LDA CHR      ;CHAR TO PRINT
          ASL A        ;OFFSET= CHAR*9
          ROL CHAROFS+1
          ASL A
          ROL CHAROFS+1
          ASL A
          ROL CHAROFS+1
          CLC
          ADC CHR
          STA CHAR
          LDA CHAROFS+1 ;ADD IN LOC
          ADC CHAR+1   ;OF CHR SET
          STA CHAR+1
LOOP2     LDY PLUS     ;BYTE OF CHAR.
          LDA (CHAR),Y ;NUMBER FOR BYTE
          LDY #Ø
          STA (OFFSET),Y ;PUT ON SCRN
          LDA OFFSET
          CLC
          ADC #4Ø      ;4Ø BYTES PER LINE
          STA OFFSET
          BCC CCC      ;OVERFLOW? NO.
          INC OFFSET+1
CCC       INC PLUS     ;9 BYTES PER
          LDA PLUS     ;CHR
          CMP #9
          BNE LOOP2
          DEC CHECK    ;NEXT CHR
          LDA CHECK
          CMP #255     ;WOULD BE THIRD
          BEQ MMM      ;CHR
          JMP PRINT    ;(TWO PER POPCORN)
;
MMM       RTS
;
;ROUTINE TO ERASE POPCORN
;THAT TOUCHED POT
;
ERASECHR  LDA HPOS     ;PLAYER HORIZ POS
          SEC
;
;ROUTINE FINDS CHAR POSITION
;BY HORIZ LOCATION OF PLAYER
;
          SBC #36
          LSR A        ;DIVIDE BY FOUR
          LSR A
          SEC
          SBC #4
          STA HPOS1
          LDY #Ø
          LDX #Ø
CHKPLAYER LDA POSITION,X
          BEQ CHECKED  ;IF Ø SKIP CHECK
          CMP HPOS1
          BEQ ERASE    ;ERASE POPCORN
CHECKED   INX
          INX
          CPX #12      ;COMP. ALL POPCORN
          BNE CHKPLAYER
          INC HPOS1    ;CHECK AROUND PLAYER
          INY
          LDX #Ø
          CPY #6       ;ALL DONE?
          BNE CHKPLAYER ;NO!
          RTS
;
ERASE     INX
          LDA POSITION,X ;VERT POS OF POPCORN
          DEX
          CMP #78      ;AT LOC OF POT (VERT)?
          BCC CHECKED  ;TO HIGH UP ON SCRN
          STA VERT     ;STORE IT
          LDA POSITION,X ;STORE
          STA HORIZ    ;HORIZ POS
          LDA #15      ;BLANK CHR
          STA CHR
          LDA #Ø
          STA POSITION,X ;PUT THAT POPCORN
          STA POSITION+1,X ;OUT OF PLAY
          JMP PRINTCHAR ;ERASE CHR
;
;CLR MEMORY ROUTINE
;
CLRØ      LDY #Ø
          TYA
CLR1      STA (CLR),Y
          INY          ;LOW BYTE OF
          BNE CLR1     ;BEG. MEM IN CLR
          INC CLR+1    ;HIGH BYTE IN
          DEX
          BNE CLR1     ;CLR+1
          RTS
;
;PLAYER DATA
;
PMDATAØ   .BYTE 32,112,24Ø,242,242,254,126
          .BYTE 62,15,15,15,15,15,15,15
PMDATA1   .BYTE 255,255,255,126,126,126
          .BYTE 126,126,126,126,Ø,Ø,Ø,Ø,Ø
PMDATA2   .BYTE Ø,12Ø,248,128,128,Ø,Ø,Ø
          .BYTE Ø,Ø,Ø,Ø,Ø,Ø,Ø
;
;RUN ADDRESS
;
          *=  $Ø2EØ
          .WORD START
          .END
```

●



An ANALOG Computing Tutorial

Painle$$ Player Mover

24K Cassette or 32K Disk

by Chet Walters

Aaahh, player/missile graphics! One of the Atari's finest features, and a powerful one, to be sure. Just displaying players is not at all difficult. Merely place the proper shapes in the correct area of memory, perform a series of POKEs to certain hardware registers and—bingo!—Antic will proceed to display your players, each with its own color, and each independent of the current display mode. Fantastic!

But try to move them through BASIC. The players must wait around for BASIC stick reads and POKEs to tell them where to go. The problematic result is jerky motion or jumps from one spot to another. So much program time must be devoted to simple player movement, it seems there's little left over for really creative playfield displays. It's a real problem.

What's the solution? Monitor input constantly and move the players at regular intervals. Is that possible in BASIC? Sure. With a dash of machine language and effective use of your Atari's leftover vertical blank interrupt (VBI) time, it's easy!

Painle$ help.

Enter the **Painles$ Player Mover**. During each VBI period (sixty times every second), **Painles$** reads *all four sticks* and moves *all four players* smoothly to the newly designated positions (positions, incidentally, within set limits or without limits), with no BASIC overhead. In fact, once **Painles$** is in place and activated, your BASIC program can stop completely, while **Painles$** goes on reading sticks and moving players (pointless, certainly, but it does).

Besides stick moves, **Painles$** will move players in response to simple POKEs. **Painles$** offers the means to instantly change any player's shape at any time. **Painles$** deals equally as well with double-line resolution as it does with single-line resolution. It will move your players at the speed of your choice, and that speed can be changed during the program run.

The **Painles$** machine code is completely relocatable (except a necessary data table that resides in the cassette buffer) and can be stored in a string or in that wasted memory in front of player memory. **Painles$** uses only four obscure page 0 memory locations, so the usual ones remain available for your use. After the initial setup, page 6 is completely free for your exclusive use. And, as mentioned, the program executes automatically every VBI, or **Painles$** itself can be turned off with a single POKE!

As you may have surmised, it will take some time to explain how to use such a versatile routine. This article will do just that. . .and *only* that (the editors have asked that I be brief). If you don't know what's needed to get Atari's player/missile graphics up and running, there are plenty of sources from which you can glean the necessary information. I suggest that you get one and familiarize yourself with player/missile graphics, because little of what the system requires will be provided here. If you are familiar with Atari's player/missile graphics and all of the features it offers, let's get to work!

(continued on page 39)

ATARI

600XL .... CALL
800XL .... CALL

© 1984 Atari, Inc. All rights reserved.

DISK DRIVES
- Rana 1000 $298
- Astra 2001 $549
- Indus GT $309
- Trak AT-D2 $329
- Trak AT-1 $319
- Trak AT-D4 Call
- Astra 1620 (Dual) $499
- Percom Call
- Atari 1050 $249

INTERFACES
- Axiom 846 Call
- Ape Face Call
- Atari 850 (In Stock) $169
- Interfast 1 $150
- Microbits 1150 Call
- R-Verter Call

DIRECT PRINTERS
- Axiom AT-100 $195
- Atari 1027 $269
- Axiom 550 AT $259
- Axiom 700 AT $469
- Atari 1025 $299

MEMORIES
- Microbits 64K (XL) $115
- Mosaic 48K (400) $98
- Mosaic 64K (400/800) Call
- Mosaic 32K $68

DIRECT MODEMS
- Microbits 1000C $128

OTHER ATARI
- 400 Keyboard Call
- Koala Pad $67
- Chalkboard Pad $75
- Bit-3 80 Column $228
- ATR-8000 (64K) $489
- ATR-8000 (16K) $359
- Alien Voice Box $98
- 1010 Recorder $74

GEMINI 10X $259.00

ATARI SOFTWARE

MISCELLANEOUS
- Syn Calc (D) $59
- Syn File (D) $59
- Syn Trend (D) $48
- Syn Com (D) $29
- Syn Chron (D) $29
- Decathlon (R) $29
- Drols (D) $23
- Gyruss (R) $31
- Heist (D) $23
- Bruce Lee (C/D) $27
- Universe (D) Call
- Questron (D) $34
- Koala Coloring I $20
- Koala Logo Design $27
- Bumble Games (D) $27
- Miles Accounting Call
- World Gtst. Baseball $23
- Gridrunner (R) $20
- Sargon II (C/D) $23
- Millionaire (D) $34
- Castle Wolfenstein (D) $20
- Odesta Chess (D) $46
- Financial Wizard (D) $41
- Ultima III (D) $39

ADVENTURE INT'L
- Adv. 1-12 each (C) $18
- Preppie (C/D) $20
- Preppie II (C/D) $23
- Diskey (D) $33
- Sea Dragon (C/D) $23
- Lunar Lndr. (C) $11 (D) $15
- Galactic Empire (C) $14

ADVENTURE INT'L
- Ultra Disassembler (D) $33
- Diskey (D) $33
- Adv. 1-12 (each) (C) $18
- Saga 1-12 (each) (D) $27

ATARI
- Atari Writer (R) $68
- Paint (D) $30
- Microsoft Basic II (R) $64
- Visicalc (D) $139
- Home File Mgr (D) $36
- Assembler Editor (R) $44
- Qix (R) $32
- Dig Dug (R) $32
- Atari Logo (R) $72
- Ms. Pac Man (R) $33
- Joust (R) $33
- Donkey Kong Jr. (R) $35
- Computer Chess (R) $24
- Galaxian (R) $30
- Defender (R) $30
- ET $34
- Pac Man (R) $30
- Centipede (R) $30
- Caverns of Mars (D) $28
- Star Raiders (R) $30
- Conv. Lang. Ea. (C) $42
- Asteroids (R) $27
- Space Invaders (R) $27
- Missile Command (R) $27
- Telelink (R) $21
- Family Finance (D) $35
- Prog. 1 (C) $18

AVALON HILL
- Telengard (C) $16 (D) $19
- Close Assault (C) 20 (D) 23
- TAC (D) $27
- Space Station Zulu (D) $17 (C) $14
- Flying Ace (D) $21 (C) $18
- GFS Sorceress (D) $23 (C) $20

AVALON HILL (Cont'd)
- Moon Patrol (C) $17
- B-1 Nuc. Bomber (C) $12
- Legionnaire (C) $23
- Empire of Overmind (D) $23 (C) $20
- Tanktics (D) $20 (C) $17
- Comptr Stock & Bonds (D) $17 (C) $14
- Galaxy (D) $17 (C) $14
- Nuke War (C) $12
- Andromeda Conquest (D) $16 (C) $13
- Close Asslt (D) $23 (C) $20

BRODERBUND
- Sky Blazer (D) $22
- Bank St. Writer (D) $46
- A.E. (D) $23
- Arcade Machine (D) $39
- Choplifter (D) $23 (R) $29
- Labyrinth (C/D) $20
- Serpentine (D) $23 (R) $27
- Steller Shuttle (C/D) $17
- Apple Panic (C/D) $20
- Genetic Drift (C/D) $20
- Oper. Whirlwind (D) $27
- Choplifter (D) $23 (R) $29
- Match Boxes (C/D) $20

CDY CONSULTING
- Pogoman (C/D) $27

CBS
- Mountain King (R) $27
- Boulders & Bombs (R) $27
- Krazy (each) (R) $27

CONTINENTAL
- Home Accountant (D) $48
- Tax Advantage (D) $45

COUNTERPOINT SW Call

DATASOFT
- Pooyan (C/D) $20
- Teletalk (D) $33
- Graphic Master (D) $23
- Micropainter (D) $23
- Zaxxon (C/D) $27
- Pac. Coast Hwy (C/D) $20
- Clowns/Balloons (C/D) $20
- Spell Wizard (D) $34
- Canyon Climber (C/D) $20

EASTERN HOUSE
- Monkey Wrench II $51

EDUCATIONAL SW
- Tricky 1,2,3 or 4 $15
- Tricky 5-13 $22

EDU-WARE
- Spelling Bee (D) $27
- Compu/Read (D) $21 (C) $15
- Compu/Math-FR (D) $27 (C) $21
- Compu/Math-Dec (D) $27 (C) $21

EPYX
- Dragon Riders (C/D) $27
- Temple APS (C/D) $27
- Jumpman (C/D) $27

FIRST STAR
- Boulder Dsh (C/D) 20 (R) 27
- Bristles (C/D) $20
- Flip Flop (C/D) $20

GAMESTAR
- Football (C/D) $21
- Baseball (C/D) $21

Printers/Etc.

DIABLO
- 630 Letter Qual. $1559

SILVER REED
- EXP 400 Ltr. Qual. $288
- EXP 500 Ltr. Qual. $355
- EXP 550 Ltr. Qual. $419
- EXP 770 Ltr. Qual. $849

STAR
- Gemini 10X $259
- Gemini 15X $378
- Delta 10 $378
- Delta 15 $519
- Radix 10 $549
- Radix 15 $645
- Power Type Call

TOSHIBA
- 1340 $739
- 1351 $1249

NEC
- 3510 $1215
- 3530 $1215
- 3550 $1498
- 7710/7730 $1648

CITOH
- Prowriter $315
- Prowriter II $498
- Starwriter $909
- Printmaster $1189

OKIDATA
- 82A Call
- 84P 799
- 92 $409
- 93 $639

DAISYWRITER
- 2000 $985

MANNESMANN
- 160L $559
- Spirit $299

JUKI
- 6100 $409

ABATI
- LQ 20P Ltr. Qual. $378

PANASONIC
- 1090 $255
- 1091 $309
- 1092 $458

MONITORS

AMDEK
- V300 G $119
- V300 A $139
- V310 G (IBM) $155
- V310 A (IBM) $159
- Color I+ $269
- Color II+ $399
- Color III $349
- Color IV (IBM) $699

PRINCETON GRAPHICS
- MAX 12 (Amber) $178
- HX 12 (RGB) $449
- SR 12 (RGB) $595

SUPER 5
- 100A (Amber) $99
- 500G (IBM with tilt) $126
- 500A (IBM with tilt) $126

SAKATA
- SC100 (Color) $239
- 1000G (Green) $99

TAXAN
- 100 Green $115
- 105 Amber $125
- 210 RGB/Composite $269
- 400 RGB Med-Res. $296
- 415 RGB Hi-Res. $429
- 420 RGB Hi-Res.(IBM) $479

NEC
- JB 1260 (Grn) $99
- JB 1201 (Grn) $145
- JB 1205 (Amber) $145

ZENITH
- Green $85
- Amber $95

MODEMS

NOVATION
- J-Cat $99
- Apple Cat II $259
- D-Cat $149

HAYES
- Smartmodem $199
- Smartmodem 1200 $479
- Micromodem II $259
- Micromodem IIe $225

PROMETHIUS
- Promodem 1200 $349

D = DISK
R = ROM (Cartridge)
C = CASSETTE
OPEN MON-SAT

COSMIC™

COSMIC COMPUTERS

727 BREA CANYON RD., SUITE 16
WALNUT, CA 91789

(800) 626-7642
PLEASE FOR ORDERS ONLY
SORRY, NO COD'S

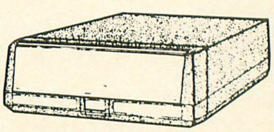
IN CALIF. **(714) 594-5204**
FOR TECHNICAL INFO, ORDER INQUIRIES,

Add $2.50 shipping per software order in continental U.S. Add $5.00 shipping per software order for AK, HI, FPO-APO. Add $10.00 or 15% (whichever is greater) per software order for non-U.S. Call for cost of hardware shipping. Calif. residents add 6½% sales tax. Cashiers checks or money orders filled within 24 hours for items in stock. Personal checks require 4 weeks to clear. MasterCard and Visa OK for software only within continental U.S., add 3% surcharge. Include card no., expiration date and signature. Due to our low prices, all sales are final. All defective returns must have a return authorization number. Please call to obtain one before returning goods for replacement or repair. Prices & availability subject to change.

ASTRA
DISK DRIVE SYSTEM

MORE FOR YOUR MONEY
DOUBLE OR SINGLE DENSITY
TWO DRIVES

2001 DISK DRIVE ... $549



INDUS GT...... $309

ATARI SOFTWARE

MISCELLANEOUS
- Financial Wizard (D) $41
- Castle Wolfenstein (D) $20
- Master Type (D) $27
- Astro Chase (D) $22
- Miner 2049er (R) $34
- Megalegs (C) $23
- Cap'n Cosmo (D) $19
- Spy's Demise (D) $14
- Galac. Chase (D) $20 (C) $18
- Kid Grid (C/D) $20
- Battle Trek (D) $20
- Air Strike (C/D) $27
- T&F-PMP Property Management (D) $149

GEBELLI
- Firebird (R) $27
- Embargo (R) $30
- Candy Factory (D) $20
- Match Racers (C/D) $20
- Dr. Goodcode (D) $20

HAYDEN
- Go (D) $23 (C) $20
- Sargon II (D) $23 (C) $20

HES
- Coco (C/D) $34
- Coco II (C/D) $27

INHOME
- Baseball (R) $23 (D) $23
- Crypts of Terror (D) $23 (C) $20

INNOVATIVE DESIGN
- Speedway Blast (D) $20 (R) $27
- Pool 400 (R) $27

JV SOFTWARE
- Jrny to Plnts (C/D) $20
- Action Quest (C/D) $20
- Ghost Encount. (C/D) $20

INFOCOM
- Zork I, II or III (D) $27
- Deadline (D) $34
- Starcross (D) $27
- Suspended (D) $34
- Witness (D) $34
- Planetfall (D) $34
- Enchanter (D) $34
- Infidel (D) $34

KRELL SAT Call

INTELL. STATEMENTS
- Prof. Blackjack (D) $46

LJK
- Letter Perfect (D) $74
- Data Perfect (D) $74
- Spell Perfect (D) $56
- Letter Perfect (R) $74

MICROPROSE
- Solo Flight (D) $26
- Hellcat Ace (C/D) $23

MONARCH
- ABC Compiler (D) $55

OPTIMIZED SYSTEMS
- Action (R) $65
- Basic XL (R) $65
- Mac 65 (D) $58
- C-65 (D) $58
- Bug 65 (D) $23

ODESTA
- Chess (D) $45
- Checkers (D) $34
- Odin (D) $34

PARKER BROS
- Astrochase (R) $33
- Death Star (R) $33
- Q-Bert (R) $33
- Popeye (R) $33

PHOENIX
- Birth of Phoenix (D) $16
- Adv. In Time (D) $20

QUALITY
- Name That Song (D) $13 (C) $11
- Return of Hercules (D) $22
- Ali Baba (D) $22
- Jeeper Creepers (D) $20

ROKLAN
- Gorf (D) $27 (R) $30
- Dlx Invaders (D) $23 (R) $27

RESTON
- Moviemaker (D) $45

SCARBOROUGH SYS.
- Mastertype (D/R) $27
- Songwriter (D) $27

SCHOOL WIZWARE Call

SENTIENT
- Gold Rush (D) $23
- Cyborg (D) $23

SIERRA ON-LINE
- Mission Asteroids (D) $17
- Ulys.& Gldn Fleece (D) $27
- Crossfire (D) $20 (R) $23
- Mouseattack (D) $23
- Jawbreaker (D) $20 (R) $23
- Threshold (D) $27
- Homeword (D) $46
- Ultima II (D) $39
- Dark Crystal (D) $26
- Wiz. & Princess (D) $22
- Lunar Leeper (D) $20
- Wiz & Princess (D) $22
- Frogger (C/D) $23

SIRIUS
- Bandits (D) $23
- Space Eggs (D) $20
- Sneakers (D) $20
- Way Out (D) $27
- Type Attack (D) $27
- Repton (D) $27
- Critical Mass (D) $27
- Fast Eddy (R) $23

SPINNAKER
- Snooper Troop 1,2 (D) $30
- Most Amazing (D) $27
- Kids on Keys (D) $20
- Trains (D) $27
- Delta Drawing (R) $27
- Aerobics (D) $34
- Hey Diddle Diddle (D) $20
- Srch Amzng Thngs (D) $27
- Story Machine (D) $23
- Face Maker (D) $23

STRATEGIC SIM.
- Broadsides (D) $27
- Carrier Force (D) $39
- Combat Leader (D) $27
- Rails West (D) $27
- Epidemic (D) $23
- Eagles (D) $27
- Cosmic Bal I or II (D) $27

SUBLOGIC
- Flight Simulator II (D) $36
- Pinball (C/D) $20

SWIFTY
- J. White Music Lessons (C/D) $20

SYNAPSE
- File Manager (R) $54
- Fort Apocalypse (C/D) $23
- Dimension X (C/D) $23
- Blue Max (C/D) $23
- Encounter (D/R) $23
- Zepplin (C/D) $23
- Pharoah's Curse (C/D) $23
- Protector II (D) $23 (R) $29
- Shamus (D) $23 (R) $29
- Fort Apocalypse (C/D) $23
- Shamus II (C/D) $23
- Necromancer (C/D) $23
- Pharoh's Curse (C/D) $23
- Drelbs (C/D) $23
- Shadow World (C/D) $23
- Survivor (C/D) $23

THORN EMI
- Soccer (R) $34
- Jumbo Jet (R) $34
- Submarine Comm. (R) $34

TRONIX
- S.A.M. (D) $39
- P.M. Animator (D) $29
- Juice (C/D) $20
- Chatterbee (D) $27

USA
- 3-D Sprgrphcs (C/D) $27
- Survival Adv. (C/D) $17

Initialization.

Look now at TAG, Listing 1. Line 10 tells us to go to 9000 to initialize, so we'll start there. That's where **Painless$** is born. The program itself consists of Lines 9000 through 10320, with a few specialized lines that concern only TAG. The first group of lines in the 9000 series (HAL, is that you?) is absolutely necessary for **Painless$** to work. In Line 9005, we determine where our player/missile memory starts. You can locate player/missile memory anywhere within the parameters of the system's requirements, and **Painless$** will comply with your choice, but it's vital that it be determined *before* **Painless$** is called to order.

The shape of strings to come.

Line 9010 DIMensions some necessary strings (the lengths are optional). **Painless$** will draw your player shapes from the data stored in the strings P0$, P1$, P2$, P3$, respectively (player 0, etc.). Why? So you can change the shape of any player at any time with a simple equate. P0$=SMILE$ will change player 0 from his original design to whatever shape is stored in SMILE$ instantaneously!

Lines 9015 to 9031 define these necessary player shapes. P0$ and friends *must* be DIMensioned to at least 1 and *must* be filled, even if only with 0s. **Painless$** draws its data from *only* these four strings. If you want to start with a short player, only to convert him to a longer player later on, DIM these strings to the maximum length you expect to use and fill the extra with 0s (not spaces!—CNTRL-COMMA or the " " are 0).

Lines 9040 through 9075, though not mandatory (**Painless$** doesn't need them to operate), are very useful. The first lines of this group set variables to the locations used by your program to communicate with **Painless$** (Line 9040) and some of the hardware registers that deal with player/missile graphics (Line 9045).

The lines following are icing on the cake. Line 9050 DIMensions the strings used to hold additional player shapes defined in Lines 9055 to 9070. P0HOLD$ and its brothers are the strings in which we store our original player shapes.

These are not necessary, but remember, when you change shapes by P0$=SMILE$, the original shape data is lost. If you want to use the **Painless$**' shape-changing capabilities, P0$ and family should be considered temporary data storage, at best. You must provide a means of storing shapes you don't want lost. P0HOLD$ (etc.) provides that means for TAG, and Line 9075 fills them with the original design.

The only lines in this group that **Painless$** *needs* for its operation are 9005 to 9031. The rest of the 9000 series is optional, but can serve a purpose.

Lines 10000 and beyond should not be changed in any way. When you've typed in TAG (and I recommend that you do so, to see how **Painless$** works), you can delete any unnecessary lines and LIST Lines 9000 to 10320 to any device. **Painless$** will then be easily available to use in a game of your own design, with ample space for your personal tailoring.

Setting the table.

Lines 10000 to 10041 set up a data table in the cassette buffer. This table is used by **Painless$** to communicate with your program. Though the values you place in these lines will change, the format *should* remain as presented (even the REMs are useful).

Bear in mind as you read that the locations mentioned here (1024 to 1075) are only half of the necessary data table. Locations 1076 to 1104 are used by **Painless$** for its own purposes, and you shouldn't go poking around in them.

And, yes, while **Painless$** is active, cassette I/O of any kind must be avoided. If it's attempted, your Atari will go into a deep slumber that even a SYSTEM RESET will not disturb. If you're writing a program and wish to save it onto cassette, press SYSTEM RESET to disable **Painless$**, perform an LPRINT statement (printer or no printer), and *then* SAVE your program. Disk and printer I/O, though they'll cause your players' motion to hesitate, work just fine.

Get into position.

Line 10022 contains the initial horizontal positions of players 0 to 3, respectively. When **Painless$** is activated, your players will initially be displayed in the positions declared here. Note that these values are read into locations 1024 to 1027, which are called XPOS.

These are the *only* locations that will change a player's horizontal position. POKEing the hardware registers (HPOSP0 or 53248-51) will have little effect. XPOS locations should be considered shadow registers, because that's exactly what they are. **Painless$** makes them such because it is a VBI routine. Use XPOS to change the horizontal positions of your players with POKEs, not the hardware.

YPOS works similarly to XPOS. Players will be displayed first in the positions (vertically) that are declared in Line 10024. The corresponding locations that can be POKEd to change a player's vertical position are 1028 to 1031, respectively. There *are* no vertical position hardware registers, so it is obvious that only these will affect vertical position.

If you're using the stick for a particular player's motion, XPOS and YPOS will take care of themselves, and you needn't bother with them (except in the special situations discussed in the "Limits" section of this article). The simplest method that can be used to address these locations is to set variables that correspond with those affecting player 0 (see Line 9040). Then, if you want to move player 3, say, use:

```
POKE XPOS+3,90:POKE YPOS+3,40
```

It's that simple!

Four sticks?

I know, you're saying, "So what do I have to do, have three friends around to move the other players?"

Atari XL owners must be bewildered as they search for the "other" two stick ports and thumb through their manuals with a puzzled look.

No, you needn't hire three stick jockeys. With **Painles$**, you can turn unused sticks off or you can lock them in any position! STKLCK, short for stick lock, indicates locations 1032 to 1035. These correspond to the stick ports on the front of your 400/800 (1 to 4) or the STICK(0-3) command in BASIC.

Location 1032 is STICK(0) or port 1, the others following in respective order. POKE STKLCK with any number above 15, and **Painles$** will read the stick plugged into port 1, moving player 0 accordingly. POKE STKLCK with any legitimate STICK(0) value (see Figure 1), and **Painles$** will respond as if there was a stick plugged into port 1 and held in that position, even if the stick is still in your drawer! If a stick *is* plugged into port 1, it will *not* be responded to.

POKE STKLCK with any value below 5, or with 8, 12 or 15, and the stick in port 1 will be effectively turned off. If you check Figure 1, you will see that these are not legitimate STICK(0) values (with the exception of 15, which is neutral, or centered). This works, respectively, with the other ports and locations 1033, 1034 and 1035. The starting STKLCK values are set by Line 10026. Notice that TAG starts with all sticks turned off (all zeroes), but you can start any way you like. Some easy ones to remember are: *POKE STKLCK,20* (STICK 0 on); *POKE STKLCK+1,0* (STICK 1 off); *POKE STKLCK+2,6* (player 2 moves diagonally up and to the right); *POKE STKLCK+3,9* (player 3 moves down and to the left); *ad infinitum.*

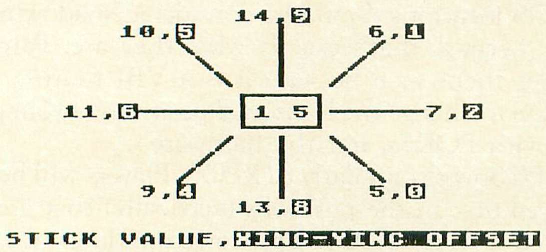


Figure 1.

Limits.

Remember my mentioning that the sticks will move players only within your set limits? These limits can range from 0 to 255, and can be changed any time you wish, with POKEs. Lines 10027 to 10034 set the initial limits.

Pay no particular attention to the values in the DATA statements, as they are peculiar to the TAG game (double-line resolution, too). Pay close attention, however, to the locations stated in the REMs. We'll discuss each in turn, starting at the top.

The values in TOP are where your respective players will stop when moving upward in response to a STICK or a STKLCK. Locations 1036 to 1039 hold the top limits. For single-line resolution, a top limit of 32 will stop your player at your Atari's text border. For double-line, a top limit of 16 will do the same.

A limit of 50 restricts a double-line player to movement in only the lower half of the screen (100 for single-line). A top limit of 1 will let a short player go out of sight at the top of the screen. *Notice* I said "1." top limit locations serve a dual purpose. If you set TOP to 0 for any player, that player will ignore *all* limits. Setting TOP to 0 turns off the limit feature of **Painles$**; it must be set to at least 1 for its respective player to obey *any* limits.

BOTTOM limits are held in 1040-1043. The correlation here for the two resolutions is akin to TOP, but there *are* certain differences. BOTTOM is 224 for single-line resolution and 112 for double. **Painles$** draws your players from the top down, so you must consider your player's overall length in determining bottom limits. Setting strictly a text border limit will allow extension of a player beyond the border by his length.

Extra care must be used in setting BOTTOM for double liners. An extremely long player may extend into his neighbor's memory area, and you'll find that he takes on the color of his neighbor and responds to that horizontal positioning, as well. **Painles$** has a built-in safeguard to ensure that your entire double liner will not inadvertently be dumped into the adjacent area, but this only applies to his first (top) byte. Any bytes of his length will still reach into no-man's-land. BOTTOM for double liners should not exceed 128 (more on this later).

The LEFT and RIGHT text border limits are the same for single- or double-line resolution. The limit for LEFT (1044 to 1047) is 48 for a player's left side . . . *Period!* The text border limit of RIGHT (1048 to 1051) for a normal width player is 200.

A player's horizontal position is read from his left side, so RIGHT's limit is actually 208 (a player is 8 bits wide). This limit changes as a player's width is set, by POKEing *SIZEPO* or *53256-9*. If a player is set to double or quadruple width with the size registers, he does not expand symmetrically, but to the right—so set your limits accordingly.

All of these limits will only affect players moved by the STICK or STKLCK features of **Painles$**. The program also affords you the luxury of POKEing any position to any player through your program. This can create some interesting anomalies.

For example: say you have set TOP to 100 for a single liner, and "border" values for the other limits. His movement in response to STICK or STKLCK is now restricted to a window in the lower half of the screen. Everything is copacetic, until your program POKEs him to a vertical position of 50. That's beyond his upper limit! What happens? He's within RIGHT and LEFT limits, so he'll respond to the stick and move right or left. He's out of range for up movement, but this also precludes any down movement, so he's stuck.

He will only move right or left if the stick is held in a diagonal, right or left position. He will not move *at all* if the stick is up, down or centered. If you now POKE him beyond his LEFT or RIGHT limits, he'll be frozen in his tracks!

The only way to recover from this (and have him respond to the stick) is to POKE him back within his limits, or POKE his TOP to 0, as noted.

Under wraps.

A limited player moved by the stick will *not* wrap around to appear on the other side of the screen. If you do want your players to wrap, you must set TOP to 0, or your program must wrap them on its own.

This is accomplished (for player 0) by:

```
IF PEEK(XPOS)>250 THEN POKE XPOS,5
```

There are exceptions to this rule, however. You will recall that a double liner's territory is only 128 bytes long. As I mentioned, **Painless$** takes care that he's not inadvertently dumped into his neighbor's area.

Consequently, if BOTTOM for a double liner is set to more than 128, and your program—with POKE or STICK—takes him beyond 128, **Painless$** subtracts 128 from the called-for position, displaying him accordingly. If you POKE a double liner to 220, for instance, he will be wrapped and displayed at a vertical position of 92 (YPOS will still hold the 220, however). Another exception to the limit rule is discussed below.

Speed.

Lines 10037 to 10040 are. . .Wait a minute. What about Lines 10035 and 10036, you ask? Well, I was going to *skip* them for now, but a short discussion won't hurt.

It's simple. SKIP is location 1052. POKE SKIP,0 and **Painless$** will read the STICKs and draw your players every VBI period. POKE SKIP with anything *but* 0, and **Painless$** will take a break, effectively freezing all player motion. Your players will be displayed in the last positions—and with the shapes they had before you told **Painless$** to skip itself. The program starts off in a SKIP state, irrespective of the value you place here; your program must turn **Painless$** on (see Line 10320).

Now, back to Lines 10037 to 10040. This is where you set the speed of your players' response to sticks. When **Painless$** interprets a STICK or "legitimate" STKLCK (refer to Figure 1), it reads a number from 5 to 15. If a movement value is returned, **Painless$** subtracts 5 from it, yielding 0 through 9.

Using this remainder as an offset into the XINC locations, **Painless$** then adds the appropriate XINC byte to the affected player's XPOS. The player's TOP is then checked for a "no limit" condition. If TOP is non-zero (i.e., limits in effect), the result is compared to your RIGHT or LEFT settings. If the new XPOS is found to be within these limits, it's saved, and the player is moved to the new horizontal position. The same takes place with YINC and YPOS, with the result compared to your TOP (if non-zero) and BOTTOM settings. If all is okay, your player is moved to the new vertical position.

For example: you're using the stick to move a player diagonally up and to the right. **Painless$** reads 6, subtracts 5 and now has 1. Location 1057 (XINC+1) holds a 1, which is added to the present XPOS. Your player is moved one increment to the right. Location 1067 (YINC+1) holds 255. Adding 255 to the present YPOS essentially "subtracts" 1. Your player is moved one increment *up*.

If you want your players to respond only to the four cardinal directions, fill the proper XINC and YINC bytes with 0s. If you want your players to move at double speed, make all the 1s into 2s and all the 255s into 254s (i.e., +2 or −2). You could conceivably have your players moving faster up than down, or faster right than left, depending on what values have been placed in these locations. But don't forget to subtract 5 from the respective STICK values (see Figure 1).

Do you recall the earlier discussion about wrapping players around the screen? The speed you choose will affect this phenomenon in the following manner. If your limits are set to maximum, meaning 0 and 255 (except TOP), a speed of more than 1 will allow your players to wrap occasionally. If speed is set even faster, then wraps will occur at smaller limits proportionately. Check Lines 200 to 220 for a short subroutine to change the speed of play during program run.

Final preparation.

Lines 10100 to 10182 hold the actual **Painless$** VBI routine. The routine is completely relocatable and can be stored anywhere. TAG stores the routine in the string **Painless$**. If you want to store it in the wasted memory in front of player memory, change Line 10101 to read:

```
10101 TRAP 10160:READ A:POKE PMBAS*256
+I,A:I=I+1:GOTO 10101
```

There are some other requirements to do this, and they'll be covered shortly.

Lines 10160 to 10182 comprise the machine language setup routine which creates the second half of the cassette buffer data table, telling the system that we want **Painless$** to execute right along with housekeeping chores. This routine is also completely relocatable. We read it into page 6 for memory conservation only. Once the routine has run, you can overwrite page 6 with any data you wish.

The call to arms.

Lines 10203 and 10205 are the USR calls that run the setup routine and create the remainder of the data table. Here's where you must do a little work for **Painless$**. You *use only one* of these calls, and *they should not be changed from their present form*. We'll dissect them, piecemeal, shortly. Note the REM statements that precede each call.

For single-line resolution, you'd run Line 10203. TAG uses double-line resolution, so this call is REMed out. It will be ignored. Line 10205 is for double-line resolution. This is the call that's made for TAG.

It won't hurt to type in *both* lines when you LIST **Painless$** to use in your own programs. You can then REM the one that isn't to be used. This simplifies switching from single- to double-line—just delete the REM and insert it into the line that you *don't* want to run. If you run with both lines active, **Painless$** will default to whichever line is listed last.

Let's examine the calls. They're quite long, but don't be intimidated. The setup routine does some minimal error trapping for you. If your Atari goes into a SYSTEM RESET condition, you've made an error and should examine your call line. We'll go step by step through Line 10205, as this is the line used by TAG. It reads:

```
10205 X=USR(1536,128,(PMBAS*256)+512,A
DR(P0$),8,ADR(P1$),8,ADR(P2$),8,ADR(P3
$),8,ADR(PAINLES$)):POKE 559,46
```

The first parameter in the call is 1536. This is the address at which the setup routine starts. If you've decided to locate setup elsewhere in memory, the starting address goes here in place of 1536.

The second parameter is 128. This tells **Painless$** that you've chosen double-line players. If you check Line 10203, you can see that this value is a 0 for single-line resolution. The setup routine checks to see if you have either a 0 or a 128 here. Any other value will cause SYSTEM RESET.

The next parameter tells **Painless$** where the start of player 0 memory is located. Note that this is *not* the start of player/missile memory, or PMBAS. Your total player/missile memory must start at a certain distance from the start of player 0 memory (1024 bytes for single-line resolution, 512 for double). Setup only checks to see that you've located player/missile memory on a page boundary, not whether it's located on the *proper* page boundary.

The next six parameters are identical in each call. ADR(Pn$) will tell **Painless$** where the string Pn$ resides in memory. Remember that these are the strings where shape data is stored for your players, and **Painless$** must know where to find them. These strings *must* be DIMed before the USR call is made, or you'll get an ERROR 9.

The number that follows each *ADR(Pn$)* is the length of the player. It can be any value above 0 and needn't be the same for each player. Nor must it be the actual length of the Pn$ string, but it ought to equal the DIM value for Pn$ in Line 9010. After the setup has been run, **Painles$** will draw your players to the lengths you've placed here. If your players are too short or too long when they appear, you've made an error in these values. The setup routine will only SYSTEM RESET if you've placed a 0 here.

The final parameter is the address of **Painles$** itself. TAG uses a string, hence the ADR( ) function. If you have chosen to locate **Painles$** elsewhere, the starting address goes here. If you make an error in this value, your Atari will lock up, forcing you to COLDSTART it.

That pretty well covers the initialization of **Painles$**. One more thing. . .setup does a last favor for you. It clears each player's memory. If you want other shapes in player memory that are not controlled by **Painles$**, insert them after setup has run. As mentioned, setup also sets the SKIP flag.

The remaining POKEs merely tell the system that you want players. The necessary 559 POKEs for resolution are conveniently set within their matching USR call lines. And, finally, Line 10320 turns **Painles$** on, with a POKE SKIP,0!

Caveat.

We've all heard the dentist's classic line, "Now open wide—this won't hurt a bit," with justifiable disbelief. **Painles$** isn't without its price, either. I don't want you to get the program up and running, only to discover hidden problems without any warning.

First of all, VBI time is not free by any means; it's merely automatic. What we've done here is set up a "USR" call that takes place in every VBI period.

Also, **Painles$** is a long routine. When it is active, you'll find that your program will run more slowly than usual. **Painles$** will cost approximately 1% of your processing time per average drawn player byte. Just total up all four player lengths and divide by 4. It's pretty cheap overall. And remember, if you have a routine that doesn't require player moves or shape changes, POKE SKIP to a positive value, so that **Painles$** costs nothing. I've found that, if your four players total more than 80 bytes in length, players 2 and 3 will be drawn in conjunction with the screen. This causes them to do funny things when they're near the top. You have to go to about 100 total player bytes (each player 25 bytes long) for them to invade text border limits, though.

Due to the relocatable nature of **Painles$**, you *cannot* experiment with it in immediate mode. If you hit the BREAK key and attempt to enter a program line, whether immediate or deferred, say good-night to your Atari. A good rule to follow is: if your players are

(continued on next page)

visible, *don't!* To make program changes, press SYSTEM RESET.

One final problem to consider: if you are detecting collisions from BASIC, you may find that BASIC will fall behind when your speeds are high. Players will then overrun certain areas before a collision is reacted to. If you come up with a viable solution to this problem, send it to **ANALOG Computing**!

These problems can't be avoided, but I think you'll agree that they seem insignificant when compared with the power that **Painless$** offers.

Alternatives.

Should you wish to change some of **Painless$**' features, here are a few alternatives.

If you only want to read one or two sticks, check Line 10118 and the REM statement above it. The 4 with an asterisk above it is the number of sticks read by **Painless$**. You can change this to the number you want (1 to 4 *only*). If you place a 2 there, STICK(0) and STICK(1) (ports 1 and 2) alone will be read. You must then control players 2 and 3 with POKEs from your program. The "shadow" characteristics of the XPOS registers for these players will still be in effect, YPOS will respond to POKEs in the normal way, and **Painless$** will still change their shapes. It's easier to simply turn off the unused sticks through STCKLCK, however, and that's what I recommend.

If you only want one or two players active, the 4 in Line 10134 (with the asterisk) is the number of players that **Painless$** will move. If a 2 is put here, **Painless$** will only draw and move players 0 and 1. You must draw and move the others. The "shadow" characteristics of XPOS will be disabled for the eliminated players, and YPOS will do nothing. *You* are now responsible for the motion of the eliminated players, nor will **Painless$** change their shapes for you.

Painless$**, by design, carefully avoids the page 0 locations that are used by most player/missile mover routines (like Tom Hudson's). You could easily incorporate one into your program. A player still under **Painless$' control should *not* be moved by these other means. Trying it will only clutter up his memory area with trash, and he'll just move back to where **Painless$** has him located, anyway.

Notes on the XLs.

Painless$** will work quite well on XLs, with a few qualifications. The first and most obvious problem is that the XLs have only two stick ports. **Painless$ reads *all four sticks*, working well if sticks 2 and 3 are off or locked. If you try to run **Painless$** with all sticks in an *unlocked* condition (i.e., ask the program to read nonexistent sticks), a very interesting thing happens. Player 2 will mimic the movements of player 0, and player 3 will mirror player 1.

If you're an XL owner who wants to use **Painless$**, you might try the changes mentioned in the previous paragraph, or keep the "other" sticks locked, or—think of it!—create "stacked" or multi-colored players with the "mirror" effect. It all depends on how you set their initial positions. Be creative!

Up and running.

"Right. Now that my eyes are blurred from reading all of this, how do I make it work?"

The TAG game gives a few hints as to how you can communicate with **Painless$**. The game is very simple, with a main loop only eight lines long. Player 0 is "it." He must avoid contact with any other player. If he's touched, he explodes. . . game over. He can score points only if he is moving; standing still gets him nothing. If he is into the "safe zone" (border), his score is reduced by two points each time through the loop. . . but the other players can't reach him.

The game, though simple, is for one to four players. With four sticks, four of you can play simultaneously. Port 1 controls player 0, etc. If you only have one or two players, the computer controls the unused sticks. Whoever "tags" player 0 should then take over stick 1 and see if he or she can beat the score.

Tag, line by line.

Line 15 shuts off all four sticks.

Line 20 waits for the number of players playing (XL owners should change the 4s to 2s in this line). Note that Line 20 also sets the SKIP flag (we don't want any movement while waiting for input).

Line 25 contains the "quick" variable, a flag used to determine when we will increase speed.

Line 30 gets the starting speed of your choice.

Line 35 will GOSUB 200 if it is not already in effect.

Line 40 concerns "attract," which I will describe later.

Line 45 turns on only sticks to be used and tells **Painless$** we're ready to move players.

Line 50 begins the main game loop, which is contained in Lines 50 to 85. Line 50 checks to see if player 0 is touching "safe" and reduces the score accordingly.

Line 55 checks for movement and increases the score.

Line 60, if we have less than four people playing, will POKE the STKLCKs of all players not under human control with random "legitimate" STICK(0) values, giving these players motion.

Line 65 is used for one-player games. It will place the current direction of player 0's motion into the STKLCK for player 3. This makes player 3 a sort of "nervous shadow" of player 0, adding challenge to a single-player game.

Line 70 checks for START order during the attract run.

Line 75 sees if we're ready to increase speed.

Line 80 prints the score and skips player collisions if attract is in effect and speed is not at maximum.

Line 82 checks to see whether or not player 0 has been tagged; if so, exit.

Lines 90 to 105 handle the explosion. First, players 1 through 3 are changed to smile faces. In the FOR/NEXT loop, the explosion is produced rather simply. Random 8-character substrings are chosen from the **Painless$** string and dumped into P0$ (try changing this to sequential substrings, using *EXPLODE = EXPLODE + 1* —the effect is quite interesting). Player 0's color is randomized to make it flash, and the SIZEP0 register is POKEd to make him shrink or swell. The hardware HPOSP0 is POKEd to the left of the original position, to make player 0 appear to expand to the left, as well.

Line 110 then resets player 0 to normal size, giving him back his original color. The timer is set, and all players are given "play" shapes to show which one tagged P0$.

Lines 120 to 135 wait for input to restart the game.

Line 130 will remain inactive for about 20 seconds, then it will cause a crude "attract" to begin. In this attract mode, the players retain their title shapes (T$, A$, G$), and all are given random legitimate STKLCKS until either maximum speed is attained (Line 80) or you press START (Line 70).

Line 140, in attract *or* restart, will reset all startup values by rereading the original DATA into the table from Lines 10022 to 10040.

Line 200 increases the speed routine.

Lines 300 and 320 change player shapes.

TAG is really rather simple and was meant to be so, because it merely demonstrates the powers of **Painles$**. Some of TAG's lines are redundant, but have been included for demonstrative purposes, too.

A bonus!

As mentioned, **Painless$** creates shadow registers for each of the four hardware "horizontal position registers" affecting players. These are updated every VBI, just like the systems color registers. Give you any ideas?

Listing 2 is a short routine that creates a display list interrupt (DLI) below the bottom limit of player 3. After you have TAG up and running, type in these few lines and see how a DLI will affect the display. You can, conceivably, have up to *eight* moving player shapes on the screen at one time with this shadow feature. Try it.

Listing 3 is a simple debugger to use in experimenting with limits and positions. Type in this short routine while writing your game. Check Line 1. Remove the REM, and the debugger will be ignored. Retype it, and the debugger will be activated. Here's a method you can use for debugging. Draw your playfield and set up **Painles$** with player shapes, etc. (all limits should, at first, be set to maximum and all sticks unlocked). Then get a grease pencil (or soapstick) and trace the significant areas (such as playfield hazards or limits) onto the glass of the screen. Replace the REM and RUN. The positions of all four players are displayed, and you can move each with the stick, recording playfield coordinates (by player standards) and limits according to your tracings. Delete the REM and RUN your game. It works!

Some final words (at last).

At the risk of redundancy, I wish to repeat that **Painles$** consists of Lines 9000 through 9031 and all lines following 10000 in Listing 1. Any other lines in this listing are unnecessary but convenient, or are peculiar to TAG. **Painles$** will cost a bit less than 2K to code in *sans* REMs.

I've been deliberately arcane about player/missile requirements of the system, but I feel that the information provided here will enable anyone (with at least a moderate understanding of BASIC) to take advantage of what **Painless$** has to offer. Coupled with a thorough knowledge of the Atari's unique OS (player/missile graphics in particular), the **Painless$ Play-**

er **Mover** becomes a powerful tool, limited only by your imagination.

If you have any problems, refer to the text and memory map for possible solutions or write to me (send a self-addressed, stamped envelope):

Chet Walters
c/o NON-STANDARD MAGIC
P.O. Box 45
Girard, OH 44420

I'll try to answer any questions. And don't forget: send your game ideas to **ANALOG Computing**, P.O. Box 23, Worcester, MA 01610. □

Painless$ memory map.

Label	Location	Function
XPOS	1024-27	Horizontal position; shadows for hardware; can be POKEd.
YPOS	1028-31	Vertical position; can be POKEd.
STKLCK	1032-35	POKE above 16 to turn sticks on; POKE below 5 to turn sticks off; POKE with "legitimate" STICK(0) value to lock.
TOP	1036-39	Entering 0 here disables *all* limits for respective player; 1 or more sets upper limit for player motion in response to STICK or STKLCK. Text border is 16 for double line; 32 for single.
BOTTOM	1040-43	Lower limit for player motion by stick. Text border is 112 for double; 224 for single (player length should be subtracted).
LEFT	1044-47	Left limit for player motion by stick. Text border is 48 for both resolutions.
RIGHT	1048-51	Right limit for player motion by stick. Varies with player width, but 200 is normal width (see text).
SKIP	1052	POKEd with a value, and **Painless$** will not execute motion or shape changes. POKEd with 0, and **Painless$** will execute full functions. Used to save time or freeze player motion (see text).
XINC	1056-65	Increment that player will move horizontally in response to stick. Offset is determined by subtracting 5 from the STICK(0) value in effect.
YINC	1066-75	As above, but for vertical motion. Speed can be doubled by replacing 1s with 2s and 255s with 254s in XINC and YINC.

Note: *never* POKE locations 1076-1104—*ever!*

The author would like to thank Perfect Computers of Niles, Ohio, for valuable assistance in the creation of this article.

Listing 1.

```
3 REM ****************************
4 REM *  A SIMPLE GAME OF TAG    *
5 REM *  DEMONSTRATING WAYS TO   *
6 REM *  COMMUNICATE WITH THE    *
7 REM *  PAINLES$ PLAYER MOVER   *
8 REM *  by CHET WALTERS 8/84    *
9 REM ****************************
10 GRAPHICS 3:POKE 710,0:POKE 752,1:?
:? ,"INITIALIZING":GOSUB 9000:GOSUB 30
0
14 REM YOUR GAME GOES HERE!!
15 FOR I=0 TO 3:POKE STKLCK+I,0:NEXT I
:IF ATTRACT THEN PLYRS=0:GOTO 35
20 TRAP 20:POKE SKIP,1:? CHR$(125):? "
ENTER NUMBER OF PLAYERS";:INPUT PLYRS:
IF PLYRS<1 OR PLYRS>4 THEN 20
25 COLOR 1:PLOT 0,0:DRAWTO 39,0:DRAWTO
 39,19:DRAWTO 0,19:DRAWTO 0,1:QUICK=40
0
30 TRAP 30:? CHR$(125):? "  ENTER SPEE
D (1 TO 4)";:INPUT SPD:SPEED=SPD:IF SP
EED<1 OR SPEED>4 THEN 30
35 TRAP 40000:IF PEEK(1056)<>SPEED THE
N GOSUB 200
40 IF  NOT ATTRACT THEN GOSUB 320
45 FOR I=0 TO PLYRS-1:POKE STKLCK+I,20
:NEXT I:POKE SKIP,0:REM TURN ON STICKS
 AND PLAYERS
48 REM
49 REM ** MAIN GAME LOOP **
50 IF PEEK(P0PF) THEN SKR=SKR-2:SOUND
0,100,10,8:POKE HITCLR,0:GOTO 60
55 IF STICK(0)<>15 THEN SKR=SKR+2+SPEE
D:SOUND 0,5,10,10
60 IF PLYRS<4 THEN FOR I=3 TO PLYRS ST
EP -1:POKE STKLCK+I,RND(I)*10+5:NEXT I
:REM RANDOM PLAYER MOVEMENT
65 IF PLYRS=1 THEN POKE STKLCK+3,STICK
(0):REM **CREATE "SHADOW"**
70 IF ATTRACT THEN SKR=SKR+7:IF PEEK(5
3279)=6 THEN ATTRACT=0:STRT=15:GOTO 12
0
75 SOUND 0,0,0,0:IF SKR>QUICK THEN SPE
ED=SPEED+1:QUICK=QUICK+300:GOSUB 200
80 ? CHR$(125):? "SCORE ";SKR;"  ",,"S
PEED ";SPEED:IF ATTRACT THEN IF SPEED<
4 THEN 50
82 IF PEEK(P0PL) THEN FOR I=0 TO 3:POK
E STKLCK+I,0:NEXT I:POKE HTCLR,0:GOTO
90
85 GOTO 50
88 REM
89 REM **CHANGE SHAPES AND EXPLODE**
90 P1$=SMILE$:P2$=SMILE$:P3$=SMILE$
95 FOR I=1 TO 50:EXPLODE=INT(RND(I)*16
0)+1:P0$=PAINLES$(EXPLODE,EXPLODE+8)
100 SOUND 0,EXPLODE,8,8:POKE 704,EXPLO
DE:WOBBLE=RND(I)*3
105 POKE HPOSP0,PEEK(XPOS)-WOBBLE*3:PO
KE SIZEP0,WOBBLE:NEXT I:SOUND 0,0,0,0
110 POKE SIZEP0,0:POKE 704,40:POKE 19,
0:POKE 20,0:GOSUB 320
115 ? CHR$(125);"GAME OVER   YOUR SCOR
E ";SKR:? "PRESS START FOR MORE PLAYER
S":? "PRESS TRIGGER FOR SAME PLAY"
120 IF PEEK(53279)=6 THEN STRT=15:ATTR
ACT=0:GOTO 140
125 IF STRIG(0)=0 THEN STRT=35-(20*ATT
RACT):ATTRACT=0:GOTO 140
130 IF PEEK(19)=5-ATTRACT*3 THEN ATTRA
CT=1:STRT=15:SPD=1:GOTO 140
135 GOTO 120
140 GOSUB 300:RESTORE 10022:FOR I=1024
 TO 1075:READ A:POKE I,A:NEXT I:GOSUB
320-(20*ATTRACT)
145 ? "K":SPEED=SPD:QUICK=400:SKR=0:PO
KE HITCLR,0:POKE 764,255:GOTO STRT
198 REM
199 REM ***INCREASE SPEED*****
200 SETCOLOR 0,PEEK(20),6:IF PEEK(1056
)=4 THEN RETURN :REM SPEED LIMIT OF 4
205 FOR I=1056 TO 1075:IF  NOT PEEK(I)
 THEN 220
210 IF PEEK(I)<10 THEN POKE I,SPEED
```

```
215 IF PEEK(I)>200 THEN POKE I,256-SPE
ED
220 SOUND 0,PEEK(I),10,8:NEXT I:SOUND
0,0,0,0:RETURN
298 REM
299 REM ***SET SHAPES FOR TITLE***
300 SETCOLOR 0,PEEK(53770),6:P0$=SMILE
$:P1$=T$:P2$=A$:P3$=G$:RETURN
318 REM
319 REM ***RESET PLAYERS TO NORMAL***
320 P0$=P0HLD$:P1$=P1HLD$:P2$=P2HLD$:P
3$=P3HLD$:RETURN
8999 REM
9000 REM PAINLESS IS BORN HERE
9001 REM
9002 REM *THESE ARE NECESSARY LINES*
9003 REM
9004 REM *FIRST DETERMINE P/M MEM*
9005 PMBAS=PEEK(106)-16
9008 REM
9009 REM *PLAYER DATA STRINGS*
9010 DIM P0$(8),P1$(8),P2$(8),P3$(8),P
AINLESS$(205)
9013 REM
9014 REM **DEFINE PLAYER SHAPES**
9015 RESTORE 9016:FOR I=1 TO 8:READ A:
P0$(I)=CHR$(A):NEXT I:POKE 704,40
9016 DATA 0,127,103,103,103,103,127,12
7
9020 FOR I=1 TO 8:READ A:P1$(I)=CHR$(A
):NEXT I:POKE 706,148
9021 DATA 0,14,28,28,28,12,12,62
9025 FOR I=1 TO 8:READ A:P2$(I)=CHR$(A
):NEXT I:POKE 705,202
9026 DATA 0,127,115,3,127,79,64,127
9030 FOR I=1 TO 8:READ A:P3$(I)=CHR$(A
):NEXT I:POKE 707,70
9031 DATA 0,127,79,31,14,103,127,62
9032 REM
9033 REM **FOLLOWING LINES OPTIONAL**
9034 REM
9039 REM *** PAINLESS$ REGISTERS ***
9040 XPOS=1024:YPOS=1028:STKLCK=1032:S
KIP=1052
9044 REM *** SYSTEM REGISTERS ***
9045 HPOSP0=53248:SIZEP0=53256:P0PF=53
252:P0PL=53260:HITCLR=53278
9048 REM
9049 REM **ADDITIONAL PLAYER SHAPES**
9050 DIM T$(8),A$(8),G$(8),SMILE$(8),P
0HLD$(8),P1HLD$(8),P2HLD$(8),P3HLD$(8)
9055 FOR I=1 TO 8:READ A:SMILE$(I)=CHR
$(A):NEXT I:DATA 129,126,90,255,189,19
5,126,60
9060 FOR I=1 TO 8:READ A:T$(I)=CHR$(A)
:NEXT I:DATA 255,255,189,60,60,60,60,1
26
9065 FOR I=1 TO 8:READ A:A$(I)=CHR$(A)
:NEXT I:DATA 24,60,126,231,231,255,231
,231
9070 FOR I=1 TO 8:READ A:G$(I)=CHR$(A)
:NEXT I:DATA 127,255,255,224,239,231,2
55,127
9074 REM **STORE SHAPE DATA**
9075 P0HLD$=P0$:P1HLD$=P1$:P2HLD$=P2$:
P3HLD$=P3$
10000 REM  NOW SET UP DATA TABLE
10001 REM LOCATED IN CASS. BUFFER
10020 RESTORE 10022:FOR I=1024 TO 1075
:READ A:POKE I,A:NEXT I
10021 REM 1024 TO 1027=XPOS
10022 DATA 90,110,130,150
10023 REM 1028 TO 1031=YPOS
10024 DATA 20,58,68,78
10025 REM 1032 TO 1035=STKLCK
10026 DATA 0,0,0,0
10027 REM 1036 TO 1039=TOP/NO LIMIT
10028 DATA 16,25,25,25
10029 REM 1040 TO 1043=BOTTOM LIMIT
10030 DATA 88,79,79,79
10031 REM 1044 TO 1047=LEFT LIMIT
10032 DATA 48,57,57,57
10033 REM 1048 TO 1051=RIGHT LIMIT
10034 DATA 200,191,191,191
10035 REM 1052=SKIP  THREE SPARES
10036 DATA 0,0,0,0
10037 REM 1056 TO 1065=XINC
10038 DATA 1,1,1,0,255,255,255,0,0,0
10039 REM 1066 TO 1075=YINC
10040 DATA 1,255,0,0,1,255,0,0,1,255
10041 REM
10050 REM CREATE OUR VBLANK INTERRUPT
10051 REM     STRING CALLED PAINLESS
10100 RESTORE 10102:I=0
10101 TRAP 10160:I=I+1:READ A:PAINLESS$
(I)=CHR$(A):GOTO 10101
10102 DATA 173,28,4,240,3,76,98,228,21
6,162,0,189
10104 DATA 8,4,201,16,144,3,189,120,2,
201,15,240
10106 DATA 74,56,233,5,48,69,168,185,3
2,4,24,125
10108 DATA 0,4,141,78,4,189,12,4,240,1
3,173,78
10110 DATA 4,221,20,4,144,11,221,24,4,
176,6,173
10112 DATA 78,4,157,0,4,185,42,4,24,12
5,4,4
10114 DATA 141,78,4,189,12,4,240,13,17
3,78,4,221
10116 DATA 12,4,144,11,221,16,4,176,6,
173,78,4
10117 REM ---# OF STICKS---*----
10118 DATA 157,4,4,232,224,4,208,163,1
62,0,142,76
10120 DATA 4,189,56,4,133,0,189,60,4,1
33,1,189
10122 DATA 64,4,133,2,189,68,4,133,3,1
89,0,4
10124 DATA 157,0,208,188,52,4,189,72,4
,170,169,0
10126 DATA 141,77,4,145,0,200,202,208,
250,174,76,4
10128 DATA 189,4,4,45,79,4,157,52,4,14
1,78,4
10130 DATA 189,72,4,170,172,77,4,177,2
,172,78,4
10132 DATA 145,0,238,77,4,238,78,4,202
,208,237,238
10133 REM ---# OF PLAYERS----*-----
10134 DATA 76,4,174,76,4,224,4,208,164
,76,98,228,0,-1
10135 REM
10150 REM  READ IN SETUP DATA
10160 RESTORE 10162:FOR I=1536 TO 1646
:READ A:POKE I,A:NEXT I
10162 DATA 216,104,201,11,240,3,76,116
,228,104
10164 DATA 104,141,79,4,133,2,240,10,2
01,128
10166 DATA 208,240,169,0,133,3,240,4,1
69,1
10168 DATA 133,3,104,133,1,104,133,0,2
08,222
10170 DATA 170,240,13,165,0,24,101,2,1
33,0
10172 DATA 165,1,101,3,133,1,165,0,157
,56
10174 DATA 4,165,1,157,60,4,172,79,4,1
69
10176 DATA 0,145,0,136,208,251,104,157
,68,4
10178 DATA 104,157,64,4,104,104,240,17
4,157,72
10180 DATA 4,232,224,4,208,203,142,28,
4,206
10182 DATA 79,4,104,170,104,168,169,7,
76,92,228
10200 REM CALL PAINLESS$ TO ATTENTION
10202 REM FOR SINGLE LINE RESOLUTION
10203 REM X=USR(1536,0,(PMBAS*256)+102
4,ADR(P0$),8,ADR(P1$),8,ADR(P2$),8,ADR
(P3$),8,ADR(PAINLESS$)):POKE 559,62
10204 REM FOR DOUBLE LINE RESOLUTION
10205 X=USR(1536,128,(PMBAS*256)+512,A
DR(P0$),8,ADR(P1$),8,ADR(P2$),8,ADR(P3
$),8,ADR(PAINLESS$)):POKE 559,46
10280 REM
10281 REM PAGE SIX AND THE USUAL
10282 REM PAGE ZERO POINTERS ARE
10283 REM ALL NOW AVAILABLE FOR
10284 REM YOUR EXCLUSIVE USE!!!!!
10285 REM
10286 REM YOUR PAGE SIX ROUTINES
```

```
10287 REM CAN BE READ IN HERE!!!
10288 REM
10300 REM  TURN ON PLAYERS 
10310 POKE 54279,PMBAS:POKE 623,33:POK
E 53277,3
10311 REM
10315 REM  TURN ON PAINLESS AND GO!! 
10320 POKE SKIP,0:RETURN
```

•

CHECKSUM DATA.

(see page 10)

```
3 DATA 259,986,448,349,355,264,271,656
,419,549,928,978,686,267,555,7970
45 DATA 98,281,500,121,353,58,542,113,
641,648,224,634,289,946,579,6027
95 DATA 441,420,257,291,748,955,877,59
6,703,383,692,109,754,440,663,8329
210 DATA 563,277,688,111,637,381,89,73
,156,343,509,293,41,297,756,5214
9005 DATA 820,307,241,623,298,472,592,
221,270,500,254,742,415,748,298,6801
9033 DATA 385,302,451,220,384,158,311,
170,843,42,149,134,641,626,942,5758
10000 DATA 301,415,907,816,981,824,724
,649,135,100,717,106,745,122,729,8271
10033 DATA 681,860,325,137,815,901,825
,879,550,292,814,846,336,388,74,8723
10106 DATA 167,106,857,788,181,92,752,
374,20,568,195,232,699,3,287,5321
10133 DATA 80,521,561,68,941,47,633,51
4,595,461,337,465,821,109,756,6909
10182 DATA 943,727,705,849,724,396,557
,307,352,452,510,572,582,848,581,9105
10300 DATA 83,609,547,108,575,1922
```

•

Listing 2.

```
10280 RESTORE 9050:FOR I=0 TO 7:READ A
:POKE (PMBAS*256)+896+108+I,A:NEXT I
10282 RESTORE 10282:FOR I=1536 TO 1546
:READ A:POKE I,A:NEXT I:DATA 72,165,20
,141,21,208,141,3,208,104,64
10284 DL=PEEK(560)+256*PEEK(561):POKE
DL+23,136:POKE 512,0:POKE 513,6:POKE 5
4286,192
```

•

Listing 3.

```
1 REM GOTO 10
2 GRAPHICS 0:POKE 710,0:POKE 712,6:POK
E 752,1:GOSUB 9000
3 FOR I=0 TO 3:POKE STKLCK+I,20:NEXT I
:POKE SKIP,0
4 POSITION 2,2:? "PNUM","XPOS","YPOS"
5 POKE 84,4:FOR I=0 TO 3:? I,PEEK(XPOS
+I);" ",PEEK(YPOS+I);" ":? :NEXT I:GOT
O 5
```

•

Reader Comment

(continued from page 9)

Friends of the C.

I'm an Atari 800 user and a data processing professional who is hooked on the C programming language. If you want an excellent programming language, read on. . .and if you already use C on an Atari, please drop me a line, so we can swap functions and tricks.

Here are three very good reasons to program in C:

1. In the business world, a lot of people are making big bucks writing in C. AT&T and IBM are betting that C will be the language of the future for business micros, and a lot of IBM followers are already on the bandwagon. The problem is that there aren't very many C programmers out there. So lately, in the computer industry's trade journals, there have been a lot of ads for programmers knowledgeable in C and UNIX—offering salaries at $30-$50,000! And this is with little other DP experience. A BASIC, assembler or COBOL programmer is lucky to get a job for half that much.

2. It's a high-level language that produces compiled code which runs rings around BASIC programs, yet is infinitely easier to use than assembler.

3. It has many of the familiar BASIC verbs, like PEEK and POKE, as functions that are used in a similar fashion. Yet—since these functions are written in C, and you are provided the source code—you can make them act differently by modifying the C code.

So, if you want to program in the language of the future—that uses many of your favorite BASIC verbs, that allows modular development, that produces fast programs, and that costs very little—buy a copy of *Deep Blue* C. Then write to me, Don Vuckovich at: Friends of the C, 584 Crescent Ave., East Aurora, NY 14052.

Don Vuckovich
East Aurora, NY

BASIC highlighter.

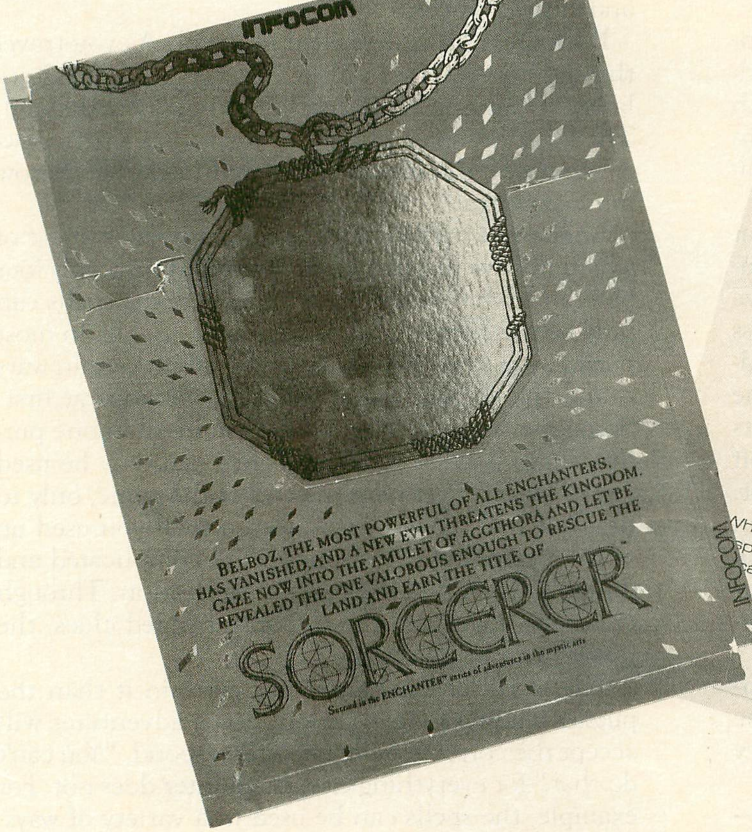
Here's a quickie little BASIC program that uses display list interrupts in a unique way. When run, it will highlight whatever line the cursor is on in dark blue.

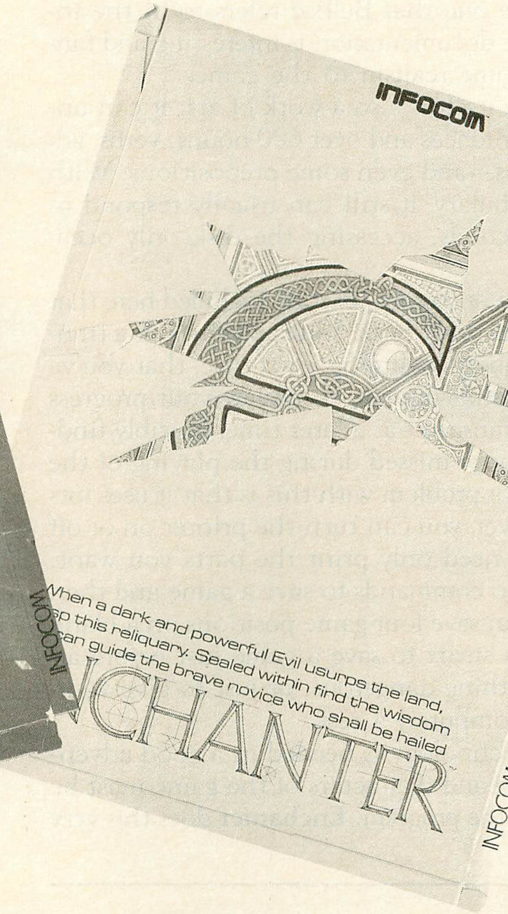
```
10 FOR I=1536 TO 1578:READ A
:POKE I,A:NEXT I
20 DLIST=PEEK(560)+PEEK(561)
*256
30 POKE DLIST+2,240:POKE DLI
ST+3,194:FOR I=DLIST+6 TO DL
IST+28:POKE I,130:NEXT I
40 POKE 512,0:POKE 513,6:POK
E 54286,192
50 DATA 72,173,11,212,233,12
,74,74,197,84,208,13,169,146
,141,10,212,141,24,208,133,2
03,76,41
60 DATA 6,165,203,201,146,20
8,10,169,148,141,10,212,141,
24,208,133,203,104,64
```

For some interesting effects, POKE 710 with random values from 0 to 255, then move the cursor up and down your screen.

Yours truly,
Angelo Giambra
Cheektowaga, NY

Send letters to:
Reader Comment
P.O. Box 23
Worcester, MA 01603





ENCHANTER and SORCERER
INFOCOM, Inc.
55 Wheeler Street
Cambridge, MA 02138
32K Disk $49.95

by Matt Hillman

It must be the warlock Krill. The odd disappearances, the mysterious dissolution of regions sacred to the Circle, the lessening of the Powers—these could only be his handiwork. The Circle gathers, and the leader, the esteemed Belboz, reveals to them an ancient document which portends evil days much like our own.

"Krill's evil must be unmade," he begins, "but to send a powerful Enchanter is ill-omened. It would be ruinous to reveal oversoon our full powers." A ripple of concern spreads over the face of each Enchanter. Belboz pauses and collects his resolve. "Have hope! This has been written by a hand far wiser than mine!"

He recites a short spell, and you appear. Belboz approaches, transfixing you with his gaze, and hands you the document. The other Enchanters await his decree. "These words, written ages ago, can have only one meaning. You, a novice Enchanter with but a few, simple spells in your Book, must seek out Krill, explore the Castle he has overthrown and learn his secrets. Only then may his vast evil be lessened or, with good fortune, destroyed."

The Circle rises to intone a richly-woven spell whose textures imbue the small, darkened chamber with warmth and hope. There is a surge of power; you are Sent.

Thus begins **Enchanter**, the first adventure in a new three-part saga from Infocom. The games are set in a magical world on the shore of a great sea—across which lies the Great Underground Empire of **Zork**.

For those adventurers who have made their way through **Zork**, **Enchanter** is like an old friend. While it is not really a sequel to the **Zork** trilogy, there are some scenes and characters that are familiar. If you haven't played **Zork**, you won't lose much; you just won't see a few jokes that only **Zork** players would recognize.

As you explore the world of **Enchanter**, the methods of conquering Krill slowly become evident, and, eventually, you gain enough knowledge to assault Krill and win the game.

Many adventures, like **Enchanter**, have good plots. However, **Enchanter** is a notch or two above most of the competition. There are a number of factors that make this, like most of Infocom's games, one of the best adventures around.

What makes it good.

The first thing one notices about **Enchanter** is the superb documentation. It comes with a Guild Directory, which humorously explains the rules and gives playing hints. And there's a genuine, sealed "ancient document," the one that Belboz refers to in the introduction. The documentation is interesting and funny, and adds some realism to the game.

The program itself is also a work of art. It can understand full sentences and over 600 nouns, verbs, adjectives, articles—and even some prepositions. With this huge vocabulary, it still can usually respond in a couple of seconds, accessing the disk only occasionally.

There are also numerous "extras" included here that many adventures don't have. You can print out a transcript of your quest (assuming, of course, that you've got a printer), allowing you to review your progress or peruse the transcript at a later time, possibly finding clues that were missed during the playing of the game. The major problem with this is that it uses lots of paper; however, you can turn the printer on or off at will, so you need only print the parts you want.

There are also commands to save a game and then recall it. You can save four game positions on a blank disk. It is often smart to save a game just before attempting something dangerous, or just to give yourself (and the computer) a rest.

More than technology is needed in a good adventure; the setting and characters of the game must be as developed as the program. **Enchanter** does this very well, making you feel as if you really are wandering through a world of strange creatures, magical spells and hostile forces.

Each location is described in detail. As you travel through the deserted village, you can *sense* the desolation. . .and, in Krill's castle, you can *feel* the evil. The locations even change over time; as Krill's power increases, the darkness and evil in the land grow more powerful.

Enchanter is filled with excellent puzzles, most of which involve magical spells. You begin with only four but can find more during the game. Some spells can only be used once, while others are reusable. In most puzzles, you must manipulate the spells in certain ways—and the correct way usually isn't obvious at first. Some spells can be used to solve more than one puzzle, which poses a problem if they can only be used once! You may get near the end of the game, only to find out that, to win, you need a spell you used up at the beginning. Most puzzles are complicated and difficult, but not to the point of frustration. Through creativity, logic and the use of imbedded clues, the puzzles *can* be overcome.

The world of **Enchanter** has more in it than the puzzles leading to its solution. Some adventures will accept the correct commands and respond, "You can't do that," for everything else; **Enchanter** does not. For example, the spells can be used in a variety of ways. One spell, Blorb, seals whatever you cast it on in a strongbox. You can cast it on anything you want, including yourself (not too healthy). There are also random events that are inconsequential to solving the puzzles but add realism to the game, making it more interesting.

Enchanter has some problems, though none really detracts from the game. At one point, a seemingly-simple action becomes complicated because of the way you have to word it; the synonym recognition of the program could be better here. Also, sometimes when you try to cast a spell on something (such as Blorbing a turtle), the computer doesn't respond with anything. It should at least say, "Nothing happens." One other problem is that you have to eat and drink; while this adds to the realism of the game, eventually it just becomes a pain in the neck.

When the warlock Krill is finally conquered—and you become a hero—you are ready for the next challenge. This is **Enchanter**'s sequel, **Sorcerer**.

Onward to Sorcerer.

In **Sorcerer**, Belboz has been kidnapped. It is up to you to rescue him. You'll have to explore the Guild Hall, an ancient fort and castle, and an underground amusement park—finally gaining the ability to defeat the demon Jeearr, who holds Belboz prisoner.

Unlike many sequels, **Sorcerer** does live up to the precedent set for it by **Enchanter** and, in many respects, surpasses it. The documentation consists of an issue of *Popular Enchanting Magazine* (or rule book) and a rotating cardboard Infotater, which gives information about the many creatures of the land, like Brogmoids and Dorns.

Game play is essentially the same as **Enchanter**'s. **Sorcerer** has all the special commands and features of its predecessor, with a much larger vocabulary (consisting of over 1000 words). There seems to be a drawback, however, in that—occasionally—the computer will spend a long time interpreting your sentence (possibly because it's searching through its vocabulary).

One small improvement is in the routine after the player's death. In previous Infocom games, in order to use a saved game after a death, you had to restart the game (which takes a little while) and then restore your old game. In **Sorcerer**, you have the option to restore a game right after your death. This can save a lot of time if you are in a place where you are continually dying.

The best improvement is in the game world itself, its setting and puzzles. **Sorcerer**'s very well developed. As you play the game, the history and geography of the land becomes evident. There is even an encyclopedia which contains entries about the history and culture of **Sorcerer**'s world. The scenario is realistic, not only because of its detailed description, but also because of its depth; there's more to this game than puzzles. Even after the game has been solved, there are many interesting things that can be done.

The puzzles *are* still the heart of the game, and **Sorcerer**'s are excellent. In general, they're not *too* difficult (**Enchanter**'s are harder) but are very interesting and, in a few cases, very complicated. Some puzzles, especially one quite complex puzzle toward the end, seem difficult at first but never reach the point of impossibility. There's always something else to try.

Like **Enchanter**, most **Sorcerer** puzzles are dependent on magical spells. Some are new, while others were used in the previous game. There are also potions, which affect the drinker in various ways. Magic works much the same way it did in **Enchanter**.

The problems in **Sorcerer** are few. One could complain that it is too easy, but that's really dependent on the player's tastes and abilities. The program seems to take a long time between commands, as mentioned earlier. The problem of food and water is taken care of early; in **Sorcerer** you don't have to carry it around with you. All in all, **Sorcerer** is one of the best adventures to be found.

Once **Sorcerer** is solved, aspiring **Enchanters** will have to wait until later this year to continue questing. **Sorcerer** drops a few hints about its sequel, but the plot remains a mystery. One can only wait. □

Magic Palette

24K Cassette or 32K Disk

by Michael and Bernard Mikowski

When I first got my 600XL, visions of complex and detailed graphics screens filled my head. For the first week or so, I messed around with PLOTs, DRAWTOs and that clumsy OS Fill. Soon eyestrain, a sore set of fingers and cold dinners caught up with me. Something had to be done.

I knew that the Atari was capable of some amazingly detailed displays, but this wasn't the way to go about doing it. What I needed was some sort of graphics program that could speed up the process. I decided to design one myself. After a few days of work, I perfected the program. **Magic Palette** is the result.

I chose to write the program in graphics mode 9 because of its many shades and its fairly high resolution. Graphics 9 allows sixteen shades of one color. You must have the GTIA chip to use this mode and this program.

Magic Palette is structured to make the designing of screens quick, easy and fun. Using the joystick, trigger and keyboard, you can draw freehand with one of the eleven brushes, draw lines between points, draw open or filled ellipses, use sixteen shades of color, draw quadralaterals, and fill in almost any nook or cranny imaginable.

Using your palette.

When you RUN **Magic Palette**, there will be a momentary pause while the data areas are initialized. When this process is complete, you will be asked for the name of the picture file you want to work with. If you're using cassette, type C: and press RETURN. Disk users should enter *D:filename.ext* and press RETURN. After you've entered this information, the program displays the drawing screen.

Once in the drawing mode, you can move the drawing cursor around on the screen with a joystick which is plugged into port 1. By using the joystick and several keyboard commands, you can create and save your own art masterpieces. Below is a list of commands and their meanings.

Joystick: Use the joystick to move the blinking cursor in one of eight directions. The cursor is "accelerating." When the cursor is moved in one direction for a while, it will pick up speed. Thus it can move around the screen more quickly, while still allowing for intricate artwork.

Brush or Color: To select a particular brush or color, simply move the cursor over the desired brush or color at the top of the screen and press the trigger.

Line: To draw a line between two points, press the SPACE BAR at the first point and then again at the second. The line will be drawn with the current color and brush size.

Rubberband: This mode allows you to draw a "continuous" line. Press *R* to enter the mode. When you enter the mode, press the trigger to establish a starting point. From then on, press the trigger to draw from the last plotted pixel position to the current cursor position. Exit the mode by pressing *R* again.

Open Circle or Ellipse: To draw an open circle or ellipse, press C once for the outer vertical edge, once again for the center of the circle, and one last time for the horizontal edge (Figure 1).

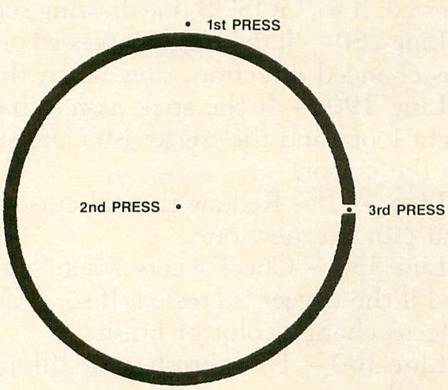


Figure 1.

Solid Circle or Ellipse: This command is typed as *S* and works the same way as the C command, but the circle is filled.

Quadralateral: Press *Q* once for the top left-hand corner, again for the bottom left-hand corner, and one last time for the bottom right-hand corner (see Figure 2).

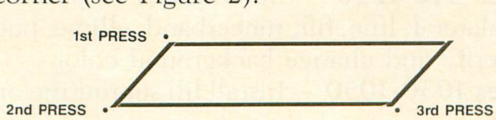


Figure 2.

Erase Screen: Type *CONTROL E*. Note: use quadralateral mode with the background color to erase smaller portions of the screen.

Fill: Any closed area may be filled in with the current color by placing the cursor in the location desired and pressing *F*. **Magic Palette** uses a modified version of the fill routine used in **Graphics 10/7 Painter** by Peter C. Budgell. For details on the subroutine, see **ANALOG Computing** issue 16.

Quick Position Cursor: Press a number (1-9) to quickly move the cursor. See Figure 3 for the positions.

1	2	3
4	5	6
7	8	9

Figure 3.

Put Screen to File: This command enables you to save a screen to the file you specified at the

(continued on next page)

Bopotron! the new levels

by ANALOG Computing readers

In **ANALOG Computing**'s issue 24, Kyle Peacock unveiled **Bopotron!**, a BASIC-and-assembly-language game. The **Bopotron Construction Set**, also published in that issue, allowed readers to create their own screens for the game. Here are two levels created by Jerry Petersen, of Portland, Oregon.

The levels are listed separately, so you can add them one at a time, if you like. When you add a level, however, be sure to change the MAXLEVEL variable in Line 160 to reflect the highest level available.

Next issue, we'll be presenting many more **Bopotron!** levels sent in by other creative readers. So, if you have a **Bopotron!** level or two that you've designed, why not send it in? □

Bopotron's Boogie Brigade
ANALOG Computing
P.O. Box 23
Worcester, MA 01603

Level 6.

```
2200 DATA 0,5,500,500
3200 DATA 0,10
4200 DATA 12,0,8,5,17,22,5,28,37,5,0,1
2,10,17,23,10,26,38,10,0,12,15,17,26,1
5,30
4201 DATA 38,16,32,38,22,0,15,23,18,26
,23
5200 DATA 5,5,10,9,5,10,32,15,23,6,15,
23,27,16,22,36
6200 DATA 1,18,23
7200 DATA 4,37,10,0,15,32,22,0,23
8200 DATA 2,5,15,15,15,10,2,15,10,15,5
,2,15,5,13,10,2,13,10,13,15,2,13,15,15
8201 DATA 15,2,3,24,10,26,5,2,26,5,23,
5,2,23,5,24,10,2
```

•

Level 7.

```
2220 DATA 0,23,500,500
3220 DATA 38,5
4220 DATA 13,0,5,5,15,20,5,32,38,5,27,
30,6,1,4,10,4,7,11,18,21,11,29,38,11,7
4221 DATA 10,12,0,10,18,18,22,18,28,38
,22,0,20,23
5220 DATA 3,5,11,32,12,18,11,18,23,18
6220 DATA 1,37,11
7220 DATA 4,0,5,29,6,0,18,37,22
8220 DATA 2,5,23,18,26,22,2,26,22,27,1
1,2,27,11,22,11,2,22,11,25,6,2,25,6,23
8221 DATA 18,2,5,13,5,16,11,2,16,11,14
,12,2,14,12,16,11,2,16,11,6,5,2,6,5
8222 DATA 5,10,2
```

•

Magic Palette

(continued from previous page)

start of the program. When you press *P*, the program will sound a short tone and wait for verification. Ready your file device (cassette or disk) and press the joystick button. If you decide not to save the screen, you can abort by moving the joystick in any direction.

Get Screen from File: This command loads a screen from the specified device. Pressing the button starts the load; moving the stick aborts the load and returns you to drawing mode.

New Color: To choose a new background color, press *N*. Then, by moving the joystick to the right or left, you can change the color. When you have selected the color you want, press the joystick button to return to the drawing mode. Note: to use **Magic Palette** in multi-color, change the graphics 9 to graphics 11 in Line 240. The new color command will then change the luminance of the sixteen colors on the screen.

I hope that you'll enjoy **Magic Palette**. □

Lines 100-190 — Set up arrays, flags, etc.
Lines 200-230 — Accept name of picture file.
Line 240 — Graphics 9. Put bar of shades on the screen.
Line 250 — Load the arrays and draw the brushes on the screen.
Line 270 — Main loop. Check to see if we are in rubberband mode and if the trigger is pressed. If so, GOSUB line drawing subroutine.
Line 280 — If the trigger is pressed or the stick has changed direction, slow down the cursor.
Line 290 — If the stick *hasn't* changed for eight loops and the trigger isn't pressed, speed up the cursor.
Line 340 — Redraw the old cursor position and plot the new one.
Line 350 — Check if cursor is in "menu" area and if the trigger is pressed. If so, GOSUB the line to change color or brush.
Line 360 — Draw brush. If no fill, jump over the fill routine.
Lines 370-380 — Read keyboard. Check for numbers for instant cursor position.
Lines 390-480 — Check for keys to certain subroutines (e.g., *F* for fill) and branch to them.
Line 490 — Back to main loop.
Lines 500-530 — Subroutine to change brush or color.
Lines 540-560 — Quick position subroutine.
Lines 570-1020 — Subroutines to perform quadralateral, line, fill, rubberband, ellipse, put, get, verify and change background color.
Lines 1030-1050 — Install fill subroutine on page 6.

```
100 GRAPHICS 18:POSITION 3,2:? #6;"MAG
IC PALETTE":POSITION 0,6:? #6;"one mom
ent please..."
110 CC=10:DIM XP(11),YP(11),NN(35),SX(
15),SY(15),A$(256),B$(256),C$(256),D$(
256),FI$(20):X2=999:X3=999:GOSUB 1030
120 FOR I=5 TO 15:READ Z:SX(I)=Z:READ
Z:SY(I)=Z:NEXT I
130 DATA 1,1,1,-1,1,0,0,0,-1,1,-1,-1,-
1,0,0,0,0,1,0,-1,0,0
140 J=INT(ADR(A$)/256):I=ADR(A$)-J*256
:POKE 203,I:POKE 204,J
150 J=INT(ADR(B$)/256):I=ADR(B$)-J*256
:POKE 205,I:POKE 206,J
160 J=INT(ADR(C$)/256):I=ADR(C$)-J*256
:POKE 207,I:POKE 208,J
170 J=INT(ADR(D$)/256):I=ADR(D$)-J*256
:POKE 1790,I:POKE 1791,J
180 FOR I=1 TO 35:NN(I)=0:NEXT I:FOR I
=1 TO 9:READ Z:NN(Z-24)=I:NEXT I:XP(11
)=4:YP(11)=8:Q=1
190 DATA 31,30,26,24,29,27,51,53,48
200 BC5=0:GRAPHICS 34
210 ? "KWHAT FILE DO YOU WANT TO WORK
WITH?":INPUT FI$:IF LEN(FI$)<2 THEN 21
0
220 IF FI$(1,2)="C:" THEN 240
230 IF FI$(1,1)<>"D" THEN 210
240 GRAPHICS 9:FOR I=0 TO 15 STEP 0.5:
COLOR I:PLOT I*2,0:DRAWTO I*2,10:NEXT
I:COLOR 7:PLOT 0,10:DRAWTO 79,10
250 RESTORE 260:FOR I=0 TO 10:READ A,B
:XP(I)=A:YP(I)=B:PLOT I*4+35,0:DRAWTO
I*4+35+XP(I),YP(I):NEXT I
260 DATA 0,0,0,1,0,4,0,6,1,0,2,0,1,1,2
,2,3,3,2,4,4,8
270 IF RR=1 THEN STT=ST:ST=STRIG(0):IF
 STT=1 AND ST=0 THEN GOSUB 610
280 IF SS<>S OR STRIG(0)=0 OR S=15 THE
N LL=0:Q=1:GOTO 300
290 LL=LL+1:IF LL>8 THEN LL=8:Q=3
300 SS=S:S=STICK(0):X=X+SX(S)*Q:Y=Y+SY
(S)*Q:IF Y>191 THEN Y=0:GOTO 320
310 IF Y<0 THEN Y=191
320 IF X>79 THEN X=0:GOTO 340
330 IF X<0 THEN X=79
340 COLOR C:PLOT X1,Y1:LOCATE X,Y,C:X1
=X:Y1=Y:COLOR 7+C:PLOT X,Y:IF STRIG(0)
=1 THEN 370
350 IF Y<11 THEN GOSUB 500:GOTO 370
360 C=CC:COLOR CC:TRAP 370:PLOT X1,Y1:
DRAWTO X1+XP,Y1+YP
370 TRAP 40000:IF PEEK(753)=0 THEN ZZ=
1:GOTO 270
380 P=PEEK(754):IF P>23 AND P<60 THEN
IF NN(P-24)>0 AND ZZ=1 THEN GOSUB 740:
GOSUB 540:ZZ=0
390 IF P=47 AND ZZ=1 THEN GOSUB 740:GO
SUB 570:ZZ=0
400 IF P=33 AND ZZ=1 THEN GOSUB 740:GO
SUB 610:ZZ=0
410 IF P=170 THEN CC=0:GOTO 240
420 IF P=18 AND ZZ=1 THEN GOSUB 740:GO
SUB 750:ZZ=0
430 IF P=62 AND ZZ=1 THEN GOSUB 740:GO
SUB 920:ZZ=0
440 IF P=56 AND ZZ=1 THEN GOSUB 740:GO
SUB 700:ZZ=0
450 IF P=10 THEN GOSUB 740:GOSUB 930
460 IF P=61 THEN GOSUB 740:GOSUB 940
470 IF P=35 THEN GOSUB 740:GOSUB 990
480 IF P=40 AND ZZ=1 THEN GOSUB 740:GO
SUB 720:ZZ=0
490 GOTO 270
500 IF X<35 THEN CC=C:RETURN
510 XP=XP((X-35)/4):YP=YP((X-35)/4):IF
 X+XP>79 THEN X=79-XP
520 IF Y+YP>191 THEN Y=191-YP
530 RETURN
540 U=NN(P-24):Y=(INT(U/3-0.3))*90+1
550 IF U>3 THEN U=U-3:GOTO 550
560 U=U-1:X=U*35+5:X=ABS(X):Y=ABS(Y):R
ETURN
570 IF X2=999 THEN X2=X:Y2=Y:RETURN
580 IF X3=999 THEN X3=X:Y3=Y:RETURN
590 COLOR CC:FOR I=X3 TO X:PLOT I,Y:DR
AWTO X2,Y2:X2=X2+1:IF X2>79 THEN X2=79
600 NEXT I:X2=999:X3=999:C=CC:RETURN
610 IF Y<10 THEN RETURN
620 IF X2=999 THEN X2=X:Y2=Y:RETURN
630 COLOR CC:FOR Z=0 TO XP:FOR Z1=0 TO
 YP:IF X+Z>79 THEN X=79-Z
640 IF X2+Z>79 THEN X2=79-Z
650 IF Y+Z1>191 THEN Y=191-Z1
660 IF Y2+Z1>191 THEN Y2=191-Z1
670 PLOT X+Z,Y+Z1:DRAWTO X2+Z,Y2+Z1
680 NEXT Z1:NEXT Z:IF RR=0 THEN X2=999
:C=CC:RETURN
690 X2=X1:Y2=Y1:RETURN
700 IF Y<11 THEN RETURN
710 AA=USR(1536,CC):C=CC:RETURN
720 IF RR=1 THEN RR=0:X2=999:RETURN
730 RR=1:RETURN
740 SOUND 0,60,10,10:SOUND 1,150,10,10
:FOR TM=0 TO 10:NEXT TM:SOUND 0,0,0,0:
SOUND 1,0,0,0:RETURN
750 FFF=0
760 IF X3=999 THEN X3=X:Y3=Y:RETURN
770 IF X2=999 THEN X2=X:Y2=Y:RETURN
780 X1=X:Y1=Y:XR=ABS(X2-X1):YR=ABS(Y2-
Y3):COLOR CC:FOR I=0 TO 3.15 STEP 0.05
790 X1C=X2+XR*SIN(I):X2C=X2-XR*SIN(I):
YC=Y2+YR*COS(I)
800 IF X1C>79 THEN X1C=79
810 IF X1C<0 THEN X1C=0
820 IF X2C>79 THEN X2C=79
830 IF X2C<0 THEN X2C=0
840 IF YC<11 THEN YC=11
850 IF YC>191 THEN YC=191
860 IF RX1C=0 THEN 900
870 IF FFF=1 THEN 890
880 PLOT RX1C,RYC:DRAWTO X1C,YC:PLOT R
X2C,RYC:DRAWTO X2C,YC:GOTO 900
890 FOR IY=YC TO RYC:PLOT X2C,IY:DRAWT
O X1C,IY:NEXT IY
900 RX1C=X1C:RYC=YC:RX2C=X2C
910 NEXT I:COLOR 7+CC:PLOT X2,Y2:X2=99
9:X3=999:RX1C=0:C=CC:RETURN
920 FFF=1:GOTO 760
930 COLOR CC:PLOT X,Y:GOSUB 950:OPEN #
1,8,0,FI$:FOR I=DMEM TO 7239+DMEM:PUT
#1,PEEK(I):NEXT I:CLOSE #1:RETURN
940 GOSUB 950:OPEN #1,4,0,FI$:FOR I=DM
EM TO 7239+DMEM:GET #1,A:POKE I,A:NEXT
 I:CLOSE #1:LOCATE X,Y,C:RETURN
950 DMEM=PEEK(88)+256*PEEK(89)+440
960 S=STICK(0):S1=STRIG(0):IF S<15 THE
N POP :RETURN
970 IF S1=1 THEN 960
980 RETURN
990 S=STICK(0):S1=STRIG(0):IF S1=0 THE
N GOSUB 740:RETURN
1000 IF S=7 THEN BC5=BC5+1:IF BC5>15 T
HEN BC5=0
1010 IF S=11 THEN BC5=BC5-1:IF BC5<0 T
HEN BC5=15
1020 SETCOLOR 4,BC5,0:GOTO 990
1030 RESTORE 1040:FOR I=1536 TO 1791:R
EAD Z:POKE I,Z:NEXT I:RESTORE :RETURN
1040 DATA 104,104,104,133,209,173,254,
6
1050 DATA 133,212,173,255,6,133,213,16
9
1060 DATA 0,133,231,160,1,165,85,145
1070 DATA 203,133,233,133,214,165,84,1
45
1080 DATA 205,133,234,133,215,132,232,
32
1090 DATA 184,6,133,230,197,209,208,1
1100 DATA 96,32,221,6,177,203,133,214
1110 DATA 177,205,133,215,230,214,165,
214
1120 DATA 201,80,176,8,32,184,6,208
1130 DATA 3,32,164,6,198,214,198,214
1140 DATA 165,214,201,255,240,8,32,184
1150 DATA 6,208,3,32,164,6,230,214
1160 DATA 230,215,165,215,201,192,176,
8
1170 DATA 32,184,6,208,3,32,164,6
1180 DATA 198,215,198,215,165,215,201,
11
```

```
1190 DATA 144,8,32,184,6,208,3,32
1200 DATA 164,6,198,232,240,4,200,76
1210 DATA 52,6,164,231,240,95,132,232
1220 DATA 177,212,145,205,177,207,145,
203
1230 DATA 136,208,245,160,1,169,0,133
1240 DATA 231,76,52,6,32,221,6,230
1250 DATA 231,240,75,164,231,165,214,1
45
1260 DATA 207,165,215,145,212,164,235,
96
1270 DATA 32,200,6,169,7,157,66,3
1280 DATA 32,86,228,164,235,197,230,96
1290 DATA 132,235,165,214,133,85,165,2
15
1300 DATA 133,84,162,96,169,0,157,72
1310 DATA 3,157,73,3,96,32,200,6
1320 DATA 169,11,157,66,3,165,209,32
1330 DATA 86,228,164,235,96,165,234,13
3
1340 DATA 84,165,233,133,85,96,32,237
1350 DATA 6,104,104,96,0,0,0,0
```

•

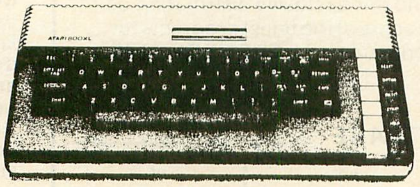
CHECKSUM DATA.
(see page 10)

```
100 DATA 640,558,148,3,96,111,126,327,
276,885,880,195,217,280,671,5413
250 DATA 540,549,24,335,377,377,766,64
4,574,342,354,533,131,229,432,6207
400 DATA 377,173,408,398,401,647,666,6
90,399,739,831,245,910,598,417,7899
550 DATA 962,949,983,991,336,199,323,9
70,457,554,739,85,904,901,354,9707
700 DATA 325,147,901,508,986,342,989,9
87,983,865,460,2,468,12,15,7990
850 DATA 392,982,827,368,455,129,870,5
03,886,494,424,88,512,621,582,8133
1000 DATA 872,106,988,88,221,229,943,4
99,467,200,185,310,740,998,18,6864
1150 DATA 919,240,616,509,618,941,216,
332,177,849,506,509,620,88,531,7671
1300 DATA 983,482,972,294,249,301,3281
```

•

Always the Lowest Prices on the Finest Quality at. . . COMPUTER CREATIONS

ATARI®



OKIMATE 10 COLOR PRINTER

$139.95

With purchase of printer interface!!

ATARI HARDWARE

Atari 800XL™ Home Computer (64K RAM) .....
Atari 410 Recorder (Special) ....
Atari 1010™ Program Recorder ....
Atari 1020™ 40-Column Color Printer/Plotter ...
Atari 1025™ 80-Column Printer ....
Atari 1027™ Letter Quality Printer ....
Atari 1030™ Direct Connect Modem ....
Atari 1050™ Disk Drive ....
CX77 Atari Touch Tablet™ ....
Atari Light Pen ....

CALL FOR UPDATED PRICES

*Screen Dump Program **FREE!!** with purchase of CX77

GENERIC DISKS AT FANTASTIC PRICES!!

GENERIC DISKS AS LOW AS 99¢ ea.
Generic 100% Defect-Free/Guaranteed.
Includes sleeves, labels, write protect tabs, reinforced hub rings, lifetime warranty.

		SS/SD	SS/DD	DS/DD
DISKETTES	1 or 2 boxes	12.99	14.99	17.99
(1 box minimum)	3 - 9 boxes	11.50	13.50	16.50
10 per box	10+ boxes	9.90	11.99	14.99

DISK DRIVES

Indus GT (Free Software) ....
Astra 1620 ....
ATR-8000 ....
Happy Enhancement for Atari 810 and 1050 Drives

CALL FOR UPDATED PRICES

SUPER SOFTWARE PRICES

Atari
Electronic Arts
EPYX
American Educational
Infocom
Syn-Series
Spinnaker
OSS
LJK
Others

DISKETTE/CARTRIDGE/ CASSETTE FILES

Flip 'N' File 10	3.95
Flip 'N' File 15	6.95
Data Defender	16.99
Flip 'N' File/The Original	16.95
Disk Bank	3.99
Colored Library Case	2.99
Disk Bank/5 (holds 50)	12.99
Power Strip (6 outlet)	16.95
Lineguard Spike Suppressor	13.99
Disk Drive Cleaning Kit	11.99
Ring King Wallet (holds 10 disks)	4.99
Ring King Wallet (holds 20 disks)	7.99

PRINTERS

GEMINI 10X (80 Column)	259.00
GEMINI 15X (136 Column)	379.00
DELTA 10 (80 Column)	360.00
RADIX 10 (80 Column)	539.00
RADIX 15 (136 Column)	649.00
POWERTYPE Daisywheel	339.00
EPSON RX-80 (80 Column)	279.00
EPSON RX-80 FT (80 Column)	359.00
EPSON FX-80 (80 Column)	429.00
OKIDATA 92P	449.00
EPSON RX 100 (135 Column)	459.00
TTX LETTER Quality Printer (Includes Tractor Feed)	359.00
Printer Interface Cables	
MPP-Microprint	49.95
MPP-1150 Parallel Printer Interface	59.95
Printer Ribbons	
GEMINI Printers (Black/Blue/Red/Purple)	3.00
EPSON Printers	8.95
RAM (Memory) BOARDS	
Microbits 64K (600XL) Expansion	89.95
MONITORS	
Sakata SC 100 Color Screen	239.00
Monitor Cable	4.95
Sanyo 12" Green Screen	84.95
Sanyo 12" Amber Screen	89.95
Sanyo Color Screen	229.95
Sanyo 9" Green Screen	69.96
MODEMS	
MPP-1000C Modem	114.95
Signalman Mark XII Modem with R-Verter	299.95
Mark X with R-Verter	169.95

AUTHORIZED SERVICE CENTER
ATARI
STAR MICRONICS (GEMINI)
Call for prices and services.

To order call TOLL FREE
1-800-824-7506

ORDER LINE ONLY

COMPUTER CREATIONS, Inc.
P.O. Bo 292467 - Dayton, Ohio 45429
For information call: (513) 294-2002 (Or to order in Ohio)

No extra charge for MasterCard or Visa

Order Lines Open 8:30 a.m. to 8:00 p.m. Mon.-Fri.; 10:00 a.m. to 4:00 p.m. Sat. (Eastern Standard Time). Minimum $10 per order. C.O.D. (add $2.50). All prices are subject to change without notice. Call toll free number to verify prices and availability of product. Actual freight will be charged on all hardware. Software and accessories add $3.50 shipping and handling in Continental United States. Actual freight will be charged outside U.S. to include Canada, Alaska, Hawaii, Puerto Rico. Ohio residents add 6% sales tax. NO CREDITS! Return must have authorization number.

LOTSABYTES DECLARES WAR !

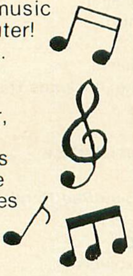
WAR on high prices ! We're going to put an end to the software price 'ripoff'. And **YOU** can help ! Just keep those orders coming while you continue to enjoy the **quality, quantity, selection** and **low prices** that you deserve. Our National Public Domain Copy Service will save you time, tedious work, and money. And our **exclusive** distribution of **sharply discounted** commercial programs will bring you some of the finest programs for the lowest possible price, usually 50% off retail! You continue to get **FREE BONUSES** with each purchase of three or more disks.

PUBLIC DOMAIN SOFTWARE

#1 **GAMES**	#2 **UTILITIES**	#3 **AMS MUSIC**	#4 **GAMES**	#5 **EDUCATION**
Two full disk sides packed with over 25 games including some Arcade quality. **$7.95**	25 powerful programs to help you get the most out of your Atari computer. **$7.95**	25 Advanced Musicsystem files including a new Player program. 2 sides. **$7.95**	All different ! 14 more better games on 2 disk sides. Some Arcade types. **$7.95**	Loaded with 28 programs on 2 disk sides Fun learning for the whole family. **$7.95**
#6 **AMS MUSIC**	#7 **GAMES**	#8 **UTILITIES**	#9 **GAMES**	#10 **UTILITIES**
25 all-time favorites with a Player program. Two sides. **$7.95**	Two disk sides packed with 14 more great games. Some Arcade types. **$7.95**	17 more power-packed utilities to help unleash the full potential of your Atari. **$7.95**	*NEW !* Two full sides filled with some of the best and most recent. Some Arcade. **$7.95**	*NEW !* A new assortment of great and powerful programs. Don't miss it ! **$7.95**

LotsaBytes EXCLUSIVES

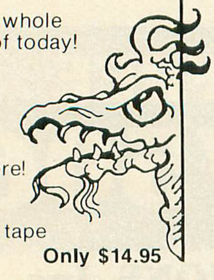
ADVANCED MUSICSYSTEM II
by LEE ACTOR
Allows you to create music with your Atari computer! All new machine code.



* Control over pitch duration, envelope dynamic level, meter, tempo and key.
* 4 independent voices
* 5½ octaves per voice
* Save up to 8200 notes
* Custom DOS
* FULL instructions
* 24K disk

Originally $29.95 **Only $14.95**

ORIGINAL ADVENTURE
by Bob Howell
For all Atari computers. The Original Colossal Cave Adventure faithfully reproduced from the 'main-frames'. This is the one that launched the whole Adventure craze of today!



* Two mazes
* 130 rooms
* Deadly Dragons
* Nasty Dwarves
* Tenacious Troll
* The Pirate & More!
* 86 coded hints
* SAVE/RESUME
* 40K disk or 32K tape

Originally $29.95 **Only $14.95**

QUALITY WORD PROCESSING

ESI WRITER! At last a brand-new Word Processor that has more features and is easier to use than anything else available for the Atari. Easy for the beginner to use, it asks questions and remembers the answers. ESI WRITER is so sophisticated that it has many more features we don't even have room to mention ! Works with ANY Atari.

* Reads any text file * Built in Help screen
* Very fast ! * Works with ANY printer
* Instant top, bottom or text location without scrolling!
* Every printer feature * DISK ONLY (Any Atari)
* Search and replace * Block move text
* Page eject/start * Set margins/lines etc.
* Full justification * Print headers etc.
* Block delete etc. * Change video color
* Over 50 pages of docs and tutorials

TRUST US ON THIS ONE ! YOU WILL LOVE IT !
Originally $49.95 ***LotsaBytes* price $24.95**

* * FREE BONUSES * *

If you purchase any 3 or more disks at a time you may choose any 1 of the following disks **FREE !!**

a. The Atari XL TRANSLATOR DISK that enables XL owners to use most 400/800 software. **FREE !!**

-- or --

b. An all different AMS MUSIC disk with Player. **FREE !!**

--or --

c. Your choice of one of the P.D. disks -- #1, #2, #3, #4, #5, #6, #7, #8, #9, or #10 (specify one) **FREE !!**

Full 100% Replacement guarantee.Any disk found to be defective will be replaced free and we will also refund your return postage. All orders shipped by First Class U.S. Mail. Add $1.95 shipping and handling for 1 to 5 disks. Add $2.95 for 6 to 12 disks. California residents add 6% sales tax. Outside of U.S.A. and Canada add 15%. U.S. Funds only. We accept checks or Money Orders. Sorry, no COD or Charge Cards. Allow three weeks for personal checks to clear.

LOTSABYTES

15445 Ventura Blvd., Suite 10H
Sherman Oaks, CA 91413

CIRCLE #114 ON READER SERVICE CARD

GREAT GAMES !

LotsaBytes Exclusives !

SPACE GAMES: Three games for **one** low price!. In **Aliens** you can't get them all and the pace keeps getting faster. When you do get rid of most of them, you are left in a space quadrant peppered with mines. Will you **Survive**? If you do, you must penetrate the alien's spaceship, survive a **Robot Attack**, and get back your stolen 'cloaking' device! Interested?
$24.95 list
***LotsaBytes price:* $12.95**

THE BEAN MACHINE by Steve Robinson is an Award Winning Arcade game that will drive you crazy balancing a series of beams while trying to get all the beans to roll down, without touching, all the while avoiding 'strange creatures' who drop in to steal the beans. It's addicting!
$24.95 list
***LotsaBytes price:* $12.95**

DIGGERBONK, another Award Winning game by Steve Robinson, challenges you to find your way through a continuously scrolling maze while avoiding some really strange creatures. Along the way you will need to Bonk some of them, but watch out for the bombs.
$24.95 list
***LotsaBytes price:* $12.95**

GUESS WHAT'S COMING TO DINNER lets you try to manuever a snake through 7 levels if you can keep it from starving or being electrocuted. Lots of surprises! One or two players.
$24.95 list
***LotsaBytes price:* $12.95**

* * * *

CREATIVE LEARNING ADVENTURES

Ages 4 to 10 — Disk only

1. Hours of **educational** fun playing 3 exciting creative adventures with a friendly alien learning about our planet Earth. Hand/eye co-ordination, drawing, and music skills are emphasized.
$24.95 list
***LotsaBytes price:* $12.95**

2. Four challenging learning games that are the favorites of our friendly alien. Helps your child to develop logical reasoning ability.
$24.94 list
***LotsaBytes price:* $12.95**

3. These 3 Fun-Day learning games will help with intellectual development, hand/eye co-ordination, logic, spatial, and analytical abilities.
$24.95 list
***LotsaBytes price:* $12.95**

An Atari BASIC Tutorial PART 2.

by Philip Altman

In the first article of this tutorial, we reviewed the structure, function and location of the memory tables that BASIC constructs as a program is entered. We looked at the process of tokenization and the format of tokenized program lines in the statement table. A program was presented to display lines in tokenized form. Now we'll analyze a few simple lines using this program, then focus on how BASIC builds and executes a program.

Program lines — examples.

First, type *NEW* to clear BASIC's tables. Now enter *Program 3*. When *READY* appears, type the following:

```
10 X=5
20 A$="TEST"
30 LIST
40 X=5:LIST
```

Run the program with *GOTO 32000* and display Line 10. This is what you'll see:

```
10-0-15-15-54-134-45-14-64-5-0-0-0-0-2
2
```

Refer to the discussion of a tokenized line in Part 1. Recall that the first two bytes are the line number in two-byte form, in this case, *10*. The next two are the line and statement lengths, respectively. Note that they are equal, since there is only one statement in the line. Token 54 is the implied *LET*, which BASIC inserts before the variable X. This variable is represented by token 134, since it is placed in the variable name table after the six variables in Program 2 (128-133). The "=" operator separates the variable from its BCD value, a six-byte numeric constant preceded by a 14 token. The line then ends with the terminator *22*.

Now do the same for Line 20, yielding the following:

```
20-0-14-14-54-135-46-15-4-84-69-83-84-
22
```

Here the line number is *20*, and the length is *14*. Note that the string name, *A$*, has been assigned the next available token, *135*, and that BASIC uses a different "=" operator (46) for string equality. The string "value," *TEST*, is preceded by a *15* (denoting a string constant) and a *4* (string length). The following bytes are the ATASCII values for *T*, *E*, *S*, *T*, as can be seen from the character dump. The line again ends with a 22 terminator.

Continuing with Line 30, we get:

```
30-0-6-6-4-22
```

Note that the statement, *LIST*, is represented by a 4 token. Since a one-statement line is the shortest one allowable in BASIC, each line must have at least six bytes.

Lastly, view Line 40 and see what happens when

(continued on page 61)

ATARI®

Vastly SUPERIOR to any translation programs available! FOR ATARI 1200XL/600XL/800XL with 64K. (Please specify computer model number!)

$69.95 (Rom)
$49.95 (D or C)

THE XL "FIX"! ®

The Atari XL series computers represent power, sophistication, and flexibility virtually unrivalled in todays Home Computer Market.

With "approximately" 30-40% of existing software being "incompatable", a real, and serious problem exists. Because of this we have developed THE XL "FIX"!

ADVANTAGES over cheaper "translation products":

1. The XL "FIX"! is capable of fixing more software . . . an estimated **30% more software!**
2. The XL "FIX"! is available in **DISK, CASSETTE,** and now **ROM!**
3. XL "FIX"! versions fix ALL THREE types of software (Disk - Cassette - and Cartridges!)
4. The XL "FIX"! (disk or cassette) adds **OVER 4K** of usable RAM to your computer (anyone using Data bases or Word processors will **really** appreciate this feature!)
5. You never have to hold the OPTION button down on 600XL or 800XL computers!
6. VERY IMPORTANT! You need to load the XL "FIX"! only **once** . . . you can **change** disks, cassettes, or cartridges **without** rebooting the XL "FIX"! each time (disk or cassette)!
7. The **ROM** version is instantaneous upon computer power up, has a high speed cursor, is instantly switchable to your original operating system, will work with 16K 600XL's, and more!

The XL "FIX"! . . . . another SUPERIOR product! **64K required!**

DISTRIBUTOR/DEALER inquires welcome

Mastercard-Visa-Money Order or Cashier Check.
Phone **(716) 467-9326**
Please specify computer model number!

Send $49.95 ($69.95 for Rom) plus $4 shipping and handling (N.Y.S. residents please add 7%) to:
COMPUTER SOFTWARE SERVICES
P.O. Box 17660
Rochester, New York 14617

ATARI ONLY **$49.95**

PROTECT your DISK programs and files BEFORE lending them out!

THE "PROTECTOR"! ™

Includes hardware and software! The "PROTECTOR" produces a true BAD SECTOR GENERATOR which will allow you to create BAD SECTORS wherever you wish (approximately 10 per second!). You'll never have to fool with ridiculous speed adjustment or tape jerking schemes again! Simple do it yourself installation requires 15 to 20 minutes!

The DISK software is the most versatile that we've ever seen and it's lightning FAST! Allows you to move and rearrange data anywhere on the disk, scrambles directories making them unaccessible to others, and offers INSTANT mapping of file disks (requires one second for ENTIRE disk!). Simple operation.

All these features are done from a 720 sector FULL VIEW map for total operator viewing and simplicity!

- Multiple drives
- Digitial SECTOR indicator
- Directory scrambler
- Moves/arranges data
- Selectable read/write
- Selectable start/end
- Hex conversion
- Disk Duping
- Disk mapping
- Instant map
- Compaction
- Fast formating
- Auto-formating
- Bad sector memory
- Instant restart
- Multiple copy function

DEALER/DISTRIBUTOR INQUIRIES WELCOME!
Our other fine products include
THE "PILL" and **THE "SILENCER".**

Send $49.95 plus $4 shipping and handling (N.Y.S. residents add 7% sales tax) to:
COMPUTER SOFTWARE SERVICES
P.O. Box 17660
Rochester, New York 14621
Phone Order:
(716) 467-9326

Mastercard-Visa-Money Orders or Bank Checks. Atari is a TM of Atari Inc. The "PROTECTOR" is a TM of COMPUTER SOFTWARE SERVICES (division of S.C.S.D., Inc.) 100% WARRANTY (replacement only - no refund policy.)

ATARI ONLY **$149.95**

For years they said it couldn't be done . . .

THE "IMPOSSIBLE"! ®

they claimed!

Backup almost any disk currently available (even **heavily protected** programs) with an **UNMODIFIED** disk drive! Works with **ANY** disk drive!

PURPOSE: The **"IMPOSSIBLE"** was developed in response to the estimated half million disk drive users that own a drive other than the Atari 810 (Indus, Percom, Trak, Rana, Astra, etc.) that wish to **BACK UP** their **protected software.** Due to a radically new technology developed by Computer Software Services, modification to your disk drive has been eliminated! The advantages are obvious! Drive warranties are not violated, the chance accidental damage has been eliminated, etc., etc.

OPERATION: The **"IMPOSSIBLE"**! consists of a disk program (**unprotected** so you can make as many backups as you wish) and a 4K STATIC RAM pack which is inserted into your computer (no soldering!) The **"IMPOSSIBLE"**! will read your program disk and then **re-write it in an unprotected format!** You may make additional backup copies using a sector copier or even regular DOS! Because your backup copy no longer has BAD SECTORS or EXOTIC FORMATS, the program data can now be manipulated into DOS compatable files (even double density!), transfered to cassette, etc. (with the aid of our Satellite programs!) No user programming knowledge required. A few programs require logical thinking.

FEATURES:
1. Backup protected disks
2. Handles most MULTI-LOAD programs
3. Makes DOS files (with Satellite option)
4. Up to 90K data input capable
5. AFSD-Automatic FUZZY Sector Discriminator
6. Expands computer memory to 52K usable
7. Simple NO SOLDER installation
8. Satellite expandable

PROJECTED SATELLITES: A **"COMPACTOR"** program which will convert your program into DOS compatable files (double density compatable!) for the storage of several programs on one disk. A **"COLUMN 80"** program for Word Proccessing, etc. It allows 80 columns on the screen! The "XL-MATE" will allow programs made with your 400/800 "IMPOSSIBLE"! to now play on your XL Computer! The **METAMORPHOSES II** program will allow you to convert your **protected CASSETTES** into disk DOS files and vice-versa. All satellite programs must be used with inconjunction with The **"IMPOSSIBLE"**!

REQUIREMENTS: The **"IMPOSSIBLE"** diskette, the 4K STATIC RAM pack, a 400 or 800 computer (please specify!) with 48K and "B" Rom's. NOTE! The very old ATARI computers were shipped with "A" Rom's which had some serious "Bugs". Even if you don't own an **"IMPOSSIBLE,"** you should upgrade to "B" Rom's (simple to install!) We have them available at a very inexpensive price. CALL US! "XL" version available soon!

NOT A PIRATING TOOL: We at C.S.S. did not design The **"IMPOSSIBLE"**! to put Software Manufactures out-of-business overnight! Nearly all of **our** products have been "ripped-off" by industry parasite who have little or no ability to develop a product of their own so we can sympathize with their dilemma. All C.S.S. products have built-in safe guards which prohibit their use for flagrant pirating. The "IMPOSSIBLE"! is no exception! While The **"IMPOSSIBLE"**! backup the most heavily protected programs, it also checks to see that the 4K STATIC RAM pack is installed before allowing the backup copy to execute!

EXAMPLES: The **"IMPOSSIBLE"**! has been tested on 300 of the most popular and heavily protected programs we could find. With nearly 4000 programs for Atari, we DO NOT guarantee that it will backup all programs in the past-present-and future! We will supply updates at $6 each (non-profit!) if and when necessary. Programs we have successfully backed up include: Blue Max, Visi-cal, Archon, Mule, File Manager 800 +, Syn Calc, Syn File, One on One, 7 Cities of Gold, Super Bunny, Load Runner, Drol, and Gumball just to name a few!

Mastercard-Visa-Money Orders or Cashier Check.
Phone: **(716) 467-9326**
Please specify computer model number!

Send $149.95 plus $4 shipping and handling (N.Y.S. residents please add 7%)

COMPUTER SOFTWARE SERVICES
P.O. BOX 17660
ROCHESTER, N.Y. 14617

we combine two lines into one with two statements. We get:

```
40-0-18-15-54-134-45-14-64-5-0-0-0-0-2
0-18-4-22
```

What previously required a total of 21 bytes now uses 18. Thus, consolidating short program lines saves memory. Since this line contains two statements, the line and statement length bytes are no longer the same. Token 20, end-of-statement, separates the two statements in the line. The next byte (*18*) is the cumulative statement length, which now equals the line length, because this is the final statement in the line.

The editor.

BASIC can be divided into two functional parts, the program editor and the program executor. The editor is normally in control when no program is running. The user interacts directly with the program editor, which waits for a line, usually from the keyboard. When a carriage return is entered, the operating system turns the line over to BASIC. Characters are placed one by one in a 128-byte input buffer located at address 1408. Note that any line longer than 128 bytes will spill over into page 6 (1536-1791). Page 6 is free memory but may be used by some programs. As the tokenized line is built, tokens are placed in a 256-byte output buffer beginning just above LOMEM and extending to the start of the variable name table. When tokenization is complete, the line will be moved from here to the statement table and executed if in immediate mode.

Want to call the editor on your own? BASIC's coldstart entry point is at 40960. Jumping to this address initializes BASIC and clears any program in memory. Just enter the following line: A=*USR(40960)*. If you want to preserve your program, simply enter BASIC at 41037.

Once BASIC has a line, it checks the BREAK key. If pressed, the line is ignored, and BASIC returns to the input mode. Otherwise, tokenizing begins. First, the line number is processed and inserted in the first two positions in the output buffer. A line consisting of a number only is searched for in the statement table and deleted when found. Lines with no number (LIST, RUN, etc.) are in immediate mode and are assigned line number 32768, higher than any legal BASIC line.

This line number flags the line for execution once tokenization is complete. The next two output buffer bytes are reserved for the as yet incomplete line and statement length values. BASIC then compares the first statement in the line with all the entries in the ROM statement name table. If the statement is found, the corresponding token is placed in the output buffer. Any unfound statement is assumed to be an implied "LET" preceding a variable (token 54). Control is now temporarily turned over to the pre-compiler.

In Part 1, we mentioned that Atari BASIC preprocesses program lines. Thus it differs from some other interpreted languages and can detect certain errors before execution. This is the function of the precompiler.

The pre-compiler is made up of a complex set of logical rules which form BASIC's grammar. Testing statements for grammatical accuracy is called "syntaxing" and proceeds through a sequence of pass/fail tests. The rules governing each statement are contained in a ROM table beginning at 42509 (?42501 for XLs). The complete operation of the pre-compiler is covered in detail in the *Atari BASIC* source book, but is beyond our scope here.

One of the pre-compiler's tasks is to identify and process each variable. First, the pre-compiler tests to see if the variable name is a valid one (begins with an upper case letter, etc.). If not, processing stops. Otherwise, the variable name table is searched for the variable. If the name is present, the variable is already known to BASIC, and its token is inserted in the output buffer. Any name not found must be new, so the name is inserted in the variable name table, an eight-byte variable value table entry is created, and the variable is assigned the next unused token (>128). This token is then placed in the output buffer.

When its tasks are complete, the pre-compiler re-

(continued on next page)

turns pass or fail. Let's first assume a syntax error was detected (fail). BASIC processes this line as an error line, converting the character at the current position in the line to inverse video. Token 55 (ERROR) is next inserted at the start of the line in the output buffer. The rest of the line is then transferred, untokenized, to the output buffer and then to the statement table, where it is saved, along with the rest of the program. When listed, this line will be preceded by *ERROR -*. Next, any variable names defined in the line are removed from the variable name table and, finally, the bad line is listed. The editor is now ready for the next line.

If "pass" is returned, the pre-compiler detected no problems. Tokenizing then continues, as we've described, until BASIC reaches the line's end. When it does, the job is done. The line length byte is updated, and the line is inserted in its correct numerical position in the statement table. Now, BASIC determines if the line is to be executed. If so, control is transferred to the program executor.

The program executor.

The program editor has created BASIC's tables and checked each line for correct syntax. The role of the program executor is to carry these statements out. The process of execution is overseen by execution control, a routine beginning at 43359 (43358 for XLs).

Execution control directs the execution of program statements one by one, beginning with the immediate mode line. Like the editor, it tests for BREAK to see if you've changed your mind. If not, it retrieves the first statement token, using it as an index into the statement execution table at 43520 (43515 for XLs). In the table, in token order, are addresses of the instructions which tell BASIC how to execute its statements. Control is transferred to the appropriate address, where the statement is processed. After processing, BASIC usually returns to execution control.

Let's consider two simple examples. First, take the immediate mode line, *LIST* <*CR*>. We've seen how this line appears when tokenized. Because the line's in immediate mode, the editor sends it to execution control for processing. Execution control skips to the first statement token, 4 (LIST), retrieves address number 4 from the statement execution table, and goes there to execute LIST. The program is now listed. BASIC then returns to execution control.

Compare this with the immediate mode line, *RUN* <*CR*>. In tokenized form, this line is identical, except that the token for *RUN*, 37, replaces the one for *LIST*. Execution control gets address number 37 from the statement execution table as its destination.

RUN directs execution control to execute each statement in every program line, from the beginning of the statement table. Statements which invoke execution control and operate in a similar way include: CONT, GOTO, GO TO and GOSUB.

At the end of each statement, execution control checks for the line's next statement, to see if it has more to do. If it's at the line's end, it tests to see if the line was in immediate mode. If so, execution control *is* done, so control returns to the program editor. Otherwise, execution control moves to the next line, continuing to the program's end. Then BASIC returns to the editor and waits for your next command.

Statement execution.

We can't detail the steps involved in the processing of each BASIC statement. But, to clarify, let's trace through the execution of *LIST*, one of BASIC's more complex statements. We've seen how execution control uses the statement token to get the execution routine's address in the statement execution table. Instructions for *LIST* begin at 46211 (46261 for XLs).

In carrying out *LIST*, BASIC reverses tokenization, converting tokenized lines back to characters (a relatively simple task, as no syntax checks are needed). BASIC can LIST a group of lines—or an entire program—to any one of several devices (e.g., E:, P:, D:). So, the most general form of the *LIST* statement would be: *LIST <device> <filename> <start line>, <endline>*. In executing *LIST*, then, BASIC must determine which, if any, of these parameters is specified. It assumes that the entire program is to be listed to screen. Then it examines the LIST statement. If a device/filename is given, IOCB #7 is opened to the device and used. If either or both line numbers are found, they're saved as starting and ending lines.

Now BASIC lists each line, statement by statement. Each token is categorized by type. Statement, operator and variable tokens are used as indices into the statement name, operator name and variable name tables. From these, their corresponding names are retrieved. Numeric constants beginning with a 14 token are converted from BCD to character format and displayed. String constants start with a 15 and are not stored with quotes. String length, which follows, tells BASIC how many characters to allow before printing the listing's final double quote. When processing is complete, BASIC returns to execution control.

Wrapping up.

We've arrived at the end of the **Tutorial** but only scratched the surface of Atari BASIC. Hopefully, this introduction has shed light on the structure and operation of BASIC. There are details not covered here, so, if you're eager to learn more, consult one of the texts mentioned earlier. The more you learn about your computer, the more you'll enjoy it. ☐

YOU HAVE ALREADY MADE YOUR FIRST MISTAKE!

You thought that cassette recorder would handle your storage needs.

WRONG

DON'T MAKE ANOTHER ONE!

You think you need a disk drive to solve your storage problems.

WRONG

YOU NEED 2 DISK DRIVES!

Any serious application practically demands at least 2 drives.

WORD PROCESSING
SPREADSHEET
DATA BASE MANAGEMENT
MAILING LIST SOFTWARE

All of these are made more powerful and, at the same time, easier to use if you have two disk drives.

So now it will cost twice as much, right?

WRONG

You need an Astra single or double density dual disk drive. Two drives in one

OKIMATE 10
OKIDATA
532 Fellowship Road
Mt. Laurel, NJ 08054
(609) 235-2600
$239.00

by Charles Bachand

At last! Someone has brought out a color printer for the Atari. My prayers have been answered. No more wasted packs of Polaroid film to throw away, nor will I ever again have to run down to Radio Shack for a set of pens for the 1020 plotter. Nope, never again. I found the ultimate printer. . .or so I thought.

The **Okimate** has to be one of the neatest printers, technology—wise, that I've seen in a long time. It's small, well styled and highly versatile for a unit that takes full size perforated or single sheet paper. This printer will actually work with both the Atari and Commodore lines of computers; you merely specify which Okimate "PLUG 'n PRINT" interface module you need when ordering. If you want to use the **Okimate 10** with both brands, you need buy only one printer with two interface modules.

You're getting warm!

What you end up with is a thermal printer that doesn't use thermal paper. Let me explain that a little better.

Most thermal printers use special paper that's chemically treated to turn dark when exposed to heat. You must use this treated paper for the system to work—which means you can't use personal or business stationary on these units. Also, since the paper only comes in one color (usually black or blue), multi-color screen dumps are definitely out.

The **Okimate 10** does not have these limitations. It uses a ribbon. . .a very special ribbon. The printer literally "melts" the ink (a wax-based substance, containing solid pigments) off the ribbon and onto the paper.

Even though the printhead can become quite warm during extended periods of operation, you need not worry that the printer will burst into flames. When the operating temperature of the printhead becomes too high, the operating speed is automatically reduced, allowing the head to cool off. By the way, the printhead on the **Okimate 10** is replaceable, just like those found on Epson printers.

Feature parade.

The **Okimate 10** does many of the things that other printers can do—like European characters, graphics, adjustable line spacing and multiple font sizes—so I won't talk about them here. Instead, we shall examine the features that are unique to this printer.

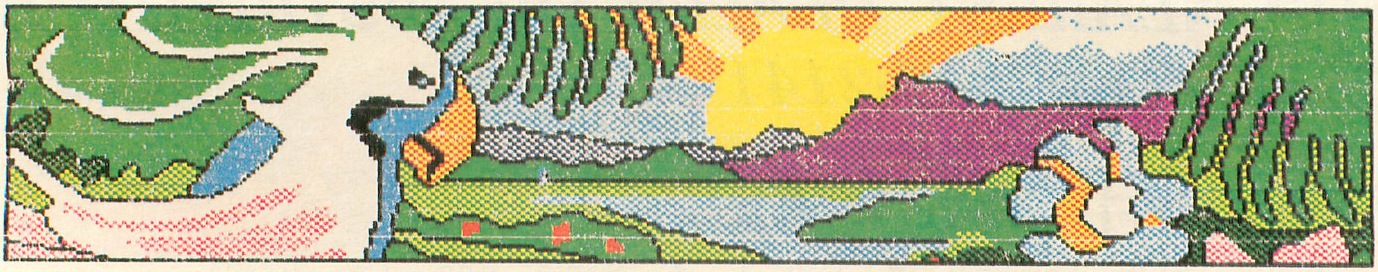
Reverse printing — The **Okimate 10** will output in inverse when the appropriate control code is sent to it. This can be used to highlight your text by printing white letters on a black background.

Repeating graphics — Here is a strange one that I'm surprised I haven't seen in other printers. When printing repeating graphics, graphics that do not change for an entire line (as in a border), you normally have to send a code to enter the printer's graphics mode and then output a byte that represents the graphic pattern *x* number of times. With the **Okimate 10**, things are greatly simplified through the repeat graphics command. The printing of hundreds of bytes is now reduced to the printing of only three—when you send a *CHR$(154)* after entering the **Okimate**'s graphics mode, followed by a byte representing the repeat count and a last byte for the graphic image. *Voila*, a nice border in only three bytes.

Fine print positioning — This one allows you to start printing anywhere on a line with resolution down to 1/60th of an inch. You can almost think of this as a very accurate TAB instruction that will work with text as well as graphics.

Full color printing — The **Okimate 10** can print both text and graphics in full color through the use of a special ribbon that contains alternating areas of the three primary colors—yellow, red (magenta) and blue (cyan). The printer makes three passes, one for each color, to produce a line in full color. Because of this, color ribbons are consumed three times faster than black ribbons.

(continued on page 78)



Atari Graphics Overlay

16K Cassette or 24K Disk

by Jeff Brenner

The ability to intermix graphics and text on the same screen is a powerful, useful computer feature. On the Atari, the mixing of the two modes has been accomplished in several ways, but these methods have limitations or problems. The most popular method is to change the Atari's display list.

The display list is a program which instructs the video chip to display specified graphics modes. As the display list is contained in RAM, it can be reprogrammed to display a combination of graphics modes on the same screen. This technique is the easiest for putting graphics and text on the screen. However, it is only a partial solution. The display list allows lines of graphics and lines of text to coexist on the screen, but it does not provide for mixing the two on one screen line. Thus, this technique is severely limited.

Another method involves rapidly switching the display from a graphics screen to a text screen so that one will appear superimposed on the other. While this permits both text and graphics to lie on the same screen line, an eye-straining flicker results from the alternating screens. In addition, the display seems faded, as each screen loses one-half of its normal luminosity.

This article presents the ultimate method for combining the two modes. The **Graphics Overlay** program listed here allows text and graphics to appear simultaneously on the same screen and line. The display is of full intensity and is flicker-free, while the graphics are completely independent of the text.

How it works.

The secret to this ideal graphics and text mixing program lies in the Atari's celebrated player/missile graphics capability. The superimposed graphics are actually the Atari's display-independent players. The term "player" denotes an 8-bit wide object (also known as a "sprite"), which can be placed at any position on the screen without interfering with the normal display.

Apparent problems.

While using the players to display graphics seems like a great idea, you might recall that each player has a maximum length of 8 pixels. The screen is composed of 160 of these pixels, or color clocks, horizontally. Furthermore, the Atari has only four players. Therefore, displaying each player side by side would yield a maximum of 32 pixels horizontally; apparently inadequate for reaching across an entire screen.

MOVING?

DON'T MISS A SINGLE ISSUE.

Let us know your new address right away. Attach an old mailing label in the space provided and print your new address where indicated.

QUESTION ABOUT YOUR SUBSCRIPTION?

Check the appropriate boxes below:

☐ New subscription. Please allow 4-8 weeks for your first copy to be mailed. ☐ Renewal subscription. Please include a current address label to insure prompt and proper extension. ☐ 1-year $28.00. This rate limited to the U.S.A. and its possessions. ☐ Payment enclosed or ☐ Bill me.

MAIL TO: ANALOG COMPUTING
P.O. Box 615, Holmes, PA 19043

Name ______
Address ______
City ______ State ______ Zip ______

ATTACH LABEL HERE

(If label is not handy, print OLD address in this space.)

Player 5.

Creating a fifth player would be of some help. This fifth player is composed of the Atari's four 2-bit missiles. It is given a single color register by setting bit #4 of GPRIOR (623/$26F).

The fifth player gives us 8 more pixels—a total of 40 horizontally. This still doesn't seem sufficient for a full screen of graphics. To cover the whole screen, we need to expand the players.

Stretching across the screen.

Locations 53256 through 53259 ($D008-$D00B) are the size registers for four of the five players. POKEing all of these locations with a *3* (setting bits #0 and #1) quadruples the width of these players. The size of the last player is quadrupled by POKEing 53260 ($D00C) with a *255*. This expands each missile that makes up the player. Line 25140 of Program 1 does this expansion. Quadrupling the width of each player gives us the screen width necessary to cover the entire screen.

The step that follows is to position each player adjacent to the next. Locations 53248 through 53251 ($D000-$D003) are the horizontal position registers for the first four players. The fifth player has four separate position registers at locations 53252 through 53255 ($D004-$D007), one for each 2-bit missile.

When each player has been expanded, it is 8*4 (8 *times* 4) pixels wide. Thus, a player should be located 32 pixels from the beginning of the previous one. The fifth player is placed on the left side of the screen, before any of the others, as its memory area resides in front of the others. The first four players are then placed after the fifth one, in order. Figure 1 shows the layout of these players. Line 25150 of Program 1 puts the players in their proper positions.

The height of each player is not a problem, since players normally extend from the top of the screen to the bottom. Single-line resolution is used to obtain 192 lines on the screen. The resolution of the new graphics screen is therefore 40 × 192 (40 *by* 192) pixels.

Plotting on the overlay.

Surprisingly, a routine to plot on this new graphics overlay is much faster and less complex than one for the normal bit-mapped graphics modes. The main reason is that each player is exactly 256 bytes away from the previous one. Thus, the proper horizontal region can be reached by merely incrementing the high byte of the memory pointer.

Specifying a vertical position is even simpler. Since each player consists of a stack of bytes, moving ahead 1 byte moves the position down one line. Thus, the vertical position is reached by incrementing the low byte of the memory pointer.

Lines 25170 through 25220 contain a machine language routine which plots on the overlay at specified coordinates. Assembly language programmers might want to study the assembly listing of this routine, which follows Program 2.

To get an idea of how the plotting is performed, we'll use coordinates (35,100) as an example. First, the player which contains this point must be found. This is done by dividing the horizontal (X) coordinate by 8 and taking the integer of the result. So, 35 divided by 8 is equal to 4.375, the integer of which is 4. So we see that the point is located in the fourth player. A result of 0 would indicate the fifth player, which precedes the others.

To determine exactly which bit of the fourth player is to be set, we subtract 8 times the resulting integer from the original X coordinate (in effect, we find the remainder). In other words, 8*4 subtracted from 35 is equal to 3. Thus, the precise horizontal position is located 3 bits to the right of the leftmost bit of player four. Figure 2 illustrates this position. The decimal equivalent of a byte with this bit set (binary 00010000) is 16.

The vertical position of 100 is then easily found by moving ahead 100 bytes in the fourth player's memory area. At this memory location, the value of 16 could be stored, and the proper pixel would be illuminated. The plotting routine actually merges (logically ORs) the value of 16 with whatever it finds at that location.

Resetting, or unplotting, a point is done in a similar way, but the bit is removed instead of added.

Adding color.

Since each player has its own color registers, up to five additional colors can be added to the acreen. The graphics overlay has five vertical color regions, one for each player. The color registers, with their associated horizontal positions, are listed below and are also illustrated in Figure 1.

Color Register Location	X Positions
711 ($2C7)	0 through 7
704 ($2C0)	8 through 15
705 ($2C1)	16 through 23
706 ($2C2)	24 through 31
707 ($2C3)	32 through 39

Thus, all points with horizontal coordinates from 0 through 7 take their color from location 711; those from 8 through 15 take their color from location 704, etc. To convert the SETCOLOR values into a value that can be POKEd into these locations, use the following formula:

```
SETCOLOR _,H,L = POKE _,H*16+L
where H = Hue and L = Luminance.
```

Line drawing and screen clearing.

In addition to the machine language plotting routine, a line-drawing routine and a routine to clear the overlay are provided. The line-drawing routine is based on a popular BASIC algorithm but has been converted to machine language for fast operation. The routine calculates the positions for the line to be drawn and then jumps to the plotting routine to plot the points.

Lines 25230 through 25340 of Program 1 contain the data for the line-drawing routine. The screen-clearing routine merely stores 0s in the player/missile graphics memory area. Lines 25350 and 25350 contain its machine language data. The assembly listings of the line-drawing and screen-clearing routines follow the PLOT assembly listing.

Using the program.

Enter Program 1 and LIST it to cassette or disk (so you can merge it with your BASIC programs). To use the overlay, your program must first GOSUB 25000. This subroutine sets up the players and loads the machine language routines into memory. BASIC memory is set back by 4K to make room for the player/missile graphics memory. This subroutine also sets up three variables: PLOT, DRAWTO and CLS. Each of the mnemonic variable holds the address of the particular function: PLOT contains the location of the plotting routine; DRAWTO contains the location of the line-drawing routine; and CLS contains the location of the overlay-clearing routine.

Using the routines is simple. To plot a point on the overlay at position (X,Y), use the following:

```
A=USR(PLOT,X,Y)
```

To draw a line from the most recently plotted point to position (X,Y), use the following:

```
A=USR(DRAWTO,X,Y)
```

The horizontal positions (X) are 0 through 39. The vertical positions are 0 through 191. Coordinates which exceed these ranges are ignored.

Location 203 ($CB) is used to determine whether to plot or erase the points. POKE 203 with a *0* to erase points. POKE 203 with a *1* to plot them. This is similar to the COLOR statement for a one-color mode (like graphics 8): COLOR C = POKE 203,C.

To clear the overlay, use the following:

```
A=(CLS)
```

This only clears the overlay; it does not clear the regular graphics screen.

These variables should not, of course, be changed by your program. Also, if your program executes a CLR command (which clears all of the variables), it will have to GOSUB 25000 again to reset them. String variables PL$, DR$ and CLS$ should not be used by your program, since they contain the actual routines.

Whenever a graphics call is executed (such as graphics 0), you must reset the player/missile graphics registers. A short subroutine at Lines 25130 through 25160 is provided to do this. Simply GOSUB 25130 after a graphics statement, and the overlay will remain intact.

Other graphics modes.

Although **Graphics Overlay** was designed primarily for use with graphics 0 screens, there is no reason why it cannot be used with other graphics modes, as well. Graphics can be added to graphics 1 and 2 text, or **Graphics Overlay** can be used to add more colors to graphics 3 through 11 (and 12 through 15 on the XL models).

The only problem created by using these modes is with the priority of the overlay. You might want the overlay to be an "underlay," letting the other graphics appear on top of the overlay graphics. Fortunately, GPRIOR at 623 ($26F) lets us give precedence to either the overlay or the playfield (the normal graphics/text area). To make the overlay appear underneath the playfield graphics, POKE 623 with a *20*. To make the overlay appear on top of the playfield graphics, POKE 623 with a *17*.

It should be noted that, when using other graphics modes, playfield 3 (the graphics using SETCOLOR 3,__,__) uses the same color register as the leftmost player. Therefore, changing the color of one will result in a change of color in the other.

Demonstration.

Enter Program 2 without erasing Program 1 from memory. Program 2 demonstrates a few of the many possible applications of **Graphics Overlay**: mixing designs and text, underlining and highlighting words, creating colorful graphs with labels, etc. Since the overlay functions like a regular graphics mode, animated graphics and fast-moving games can be created, as well.

The ability to mix graphics and text freely on the screen is a luxury afforded by very few personal computers. **Graphics Overlay** offers an excellent method of simulating this feature. Use it in your BASIC programs to create impressive, attractive and colorful displays. □

Summary of features.

Command	Effect
A = USR(PLOT,X,Y)	Plot a point at x,Y.
A = USR(DRAWTO,X,Y)	Draw a line to X,Y.
POKE 711,C	Give points in first region color C (C = HUE * 16 + LUM).
POKE 704,C	C = color for second region.
POKE 705,C	C = color for third region.
POKE 706,C	C = color for fourth region.
POKE 707,C	C = color for fifth region.
POKE 203,1	Prepare to plot points.
POKE 203,0	Prepare to erase points.
A = USR(CLS)	Clear the overlay screen.

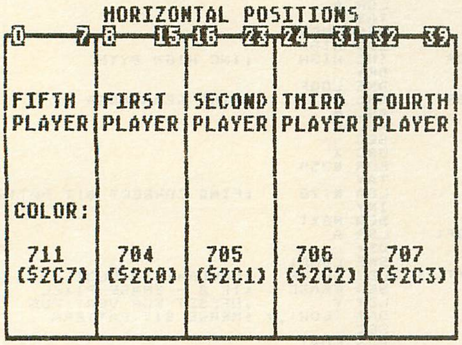


Figure 1.

Layout of the players and color registers.

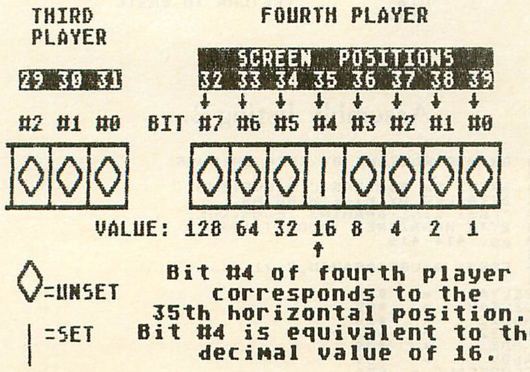


Figure 2.

Horizontal overlay positions related to bit structure and binary values.

Program 1.

```
25000 REM ATARI GRAPHICS OVERLAY
25010 REM (C) 1984 Jeff Brenner
25020 DIM PL$(83),DR$(192),CLS$(24)
25030 RESTORE 25170:FOR I=1 TO 83:READ
 NUM:PL$(I)=CHR$(NUM):NEXT I
25040 FOR I=1 TO 192:READ NUM:DR$(I)=C
HR$(NUM):NEXT I
25050 FOR I=1 TO 24:READ NUM:CLS$(I)=C
HR$(NUM):NEXT I
25060 LET PLOT=ADR(PL$):LET DRAWTO=ADR
(DR$):CLS=ADR(CLS$):I=PLOT+21
25070 H=INT(I/256):L=I-H*256:DR$(177,1
77)=CHR$(L):DR$(178,178)=CHR$(H)
25080 DR$(191,191)=CHR$(L):DR$(192,192
)=CHR$(H)
25090 IF PEEK(106)/32>INT(PEEK(106)/32
) THEN 25110
25100 POKE 106,PEEK(106)-16:GRAPHICS P
EEK(87)
25110 GOSUB 25130:I=PEEK(106)*256:H=IN
T(I/256):L=I-H*256
25120 POKE 204,L:POKE 205,H:RETURN
25130 POKE 54279,PEEK(106):POKE 711,0:
POKE 559,62:POKE 53277,3:POKE 623,17
25140 FOR I=53256 TO 53259:POKE I,3:NE
XT I:POKE 53260,255
25150 FOR I=0 TO 3:POKE 53248+I,80+I*3
2:POKE 53252+I,72-8*I:NEXT I
25160 RETURN
25170 DATA 104,104,104,201,40,176,71,1
05,24,133,206,104,104,201,192,176
25180 DATA 63,105,32,133,207,166,205,1
65,206,74,74,74,168,240,7,166
25190 DATA 205,230,205,136,208,251,10,
10,10,56,229,206,73,255,168,169
25200 DATA 128,200,240,4,74,136,208,25
2,164,203,240,7,164,207,17,204
25210 DATA 56,176,6,164,207,73,255,49,
204,145,204,134,205,96,104,104
25220 DATA 104,104,96
25230 DATA 104,104,104,48,111,201,40,1
76,107,105,24,133,97,104,104,201
25240 DATA 0,144,99,201,192,176,95,105
,32,133,96,162,0,134,214,134
25250 DATA 217,232,134,219,134,215,134
,216,165,97,56,229,206,176,9,198
25260 DATA 216,198,216,165,206,56,229,
97,133,212,165,96,56,229,207,176
25270 DATA 9,198,215,198,215,165,207,5
6,229,96,133,213,165,212,197,213
25280 DATA 176,23,166,212,165,213,133,
212,138,133,213,165,216,133,214,165
25290 DATA 215,133,217,169,0,133,216,1
33,215,165,212,74,133,218,165,212
25300 DATA 240,67,208,3,104,104,96,165
,206,24,101,216,133,206,165,207
25310 DATA 24,101,217,133,207,230,219,
165,218,24,101,213,133,218,165,212
25320 DATA 197,218,176,21,165,218,56,2
29,212,133,218,165,206,24,101,214
25330 DATA 133,206,165,207,24,101,215,
133,207,165,212,197,219,144,6,32
25340 DATA 0,0,56,176,194,165,96,133,2
07,165,97,133,206,76,0,0
25350 DATA 104,165,106,133,209,169,0,1
33,208,162,16,160,0,145,208,200
25360 DATA 208,251,230,209,202,208,244
,96
```

•

CHECKSUM DATA.

(see page 10)

```
25000 DATA 449,721,950,522,709,743,110
,180,335,888,669,723,409,566,998,8972
25150 DATA 856,70,251,219,207,0,20,488
,381,896,495,521,514,772,296,5986
25300 DATA 189,469,305,455,463,173,388
,2442
```

•

Program 2.

```
10 REM DEMONSTRATION PROGRAM
20 GRAPHICS 0:PRINT "STAND BY..."
30 GOSUB 25000:POKE 752,1:POKE 203,1
40 A=USR(CLS)
50 FOR Y=0 TO 186 STEP 6
60 A=USR(PLOT,0,96)
70 A=USR(DRAWTO,39,Y)
80 A=USR(PLOT,39,96)
90 A=USR(DRAWTO,0,Y)
100 NEXT Y
110 POKE 711,4*16+2
120 POKE 704,6*16+2
130 POKE 705,8*16+2
140 POKE 706,6*16+2
150 POKE 707,4*16+2
160 SETCOLOR 2,10,0
170 POSITION 10,10:PRINT "Mix colorful
 graphics"
180 POSITION 7,13:PRINT "with a screen
 full of text!"
190 POSITION 5,22:PRINT "(PRESS START
FOR NEXT SCREEN)"
200 IF PEEK(53279)<>6 THEN 200
210 A=USR(CLS):PRINT CHR$(125):SETCOLO
R 2,9,4:SETCOLOR 1,0,0
220 POSITION 2,5:PRINT "Underline word
s."
230 POKE 711,104:POKE 704,104:POKE 705
,104
240 A=USR(PLOT,2,48)
250 A=USR(DRAWTO,17,48)
260 POSITION 24,18:PRINT "Highlight wo
rds."
270 POKE 706,236:POKE 707,236
280 FOR I=0 TO 7
290 A=USR(PLOT,24,144+I)
300 A=USR(DRAWTO,39,144+I)
310 NEXT I
320 POSITION 5,22:PRINT "(PRESS START
FOR NEXT SCREEN)"
330 IF PEEK(53279)<>6 THEN GOTO 330
340 A=USR(CLS):PRINT CHR$(125):SETCOLO
R 2,0,0:SETCOLOR 1,0,10
350 PRINT "Create colorful graphs."
360 POKE 711,70:POKE 704,102:POKE 705,
134:POKE 706,166:POKE 707,198
370 SETCOLOR 1,0,12:SETCOLOR 2,0,0
380 FOR I=0 TO 32 STEP 8
390 FOR J=1 TO 6:A=USR(PLOT,I+J,175):A
=USR(DRAWTO,I+J,128-I*4):NEXT J
400 NEXT I
410 POSITION 2,21:PRINT "1980    1981
  1982    1983    1984"
420 POSITION 0,8:PRINT "S":POSITION 0,
9:PRINT "A":POSITION 0,10:PRINT "L"
430 POSITION 0,11:PRINT "E":POSITION 0
,12:PRINT "S"
440 POSITION 5,23:PRINT "(PRESS START
FOR NEXT SCREEN)";
450 IF PEEK(53279)<>6 THEN GOTO 450
460 A=USR(CLS):PRINT CHR$(125)
470 POSITION 5,12:PRINT "PRODUCE DYNAM
IC VIDEO DISPLAYS"
480 POSITION 10,23:PRINT "(PRESS START
 TO END)";
490 POKE 711,66:POKE 704,54:POKE 705,1
0*16+12:POKE 706,54:POKE 707,66
500 A=USR(PLOT,4,95):A=USR(DRAWTO,35,9
5)
510 A=USR(DRAWTO,35,104):A=USR(DRAWTO,
4,104)
520 A=USR(DRAWTO,4,96):C=1
530 FOR X=0 TO 39 STEP 2:A=USR(PLOT,4,
95):A=USR(DRAWTO,X,X*2)
540 A=USR(PLOT,35,95):A=USR(DRAWTO,39-
X,X*2)
550 IF PEEK(53279)=6 THEN 590
560 A=USR(PLOT,35,104):A=USR(DRAWTO,39
-X,191-X*2)
570 A=USR(PLOT,4,104):A=USR(DRAWTO,X,1
91-X*2)
580 NEXT X:C=C=0:POKE 203,C:GOTO 530
590 END
```

•

CHECKSUM DATA.

(see page 10)

```
10 DATA 464,596,465,122,125,809,92,645
,824,760,525,534,544,542,540,7587
160 DATA 714,981,347,599,806,815,405,8
,649,992,974,484,312,252,319,8657
310 DATA 735,582,384,56,127,169,574,13
2,847,734,573,639,904,460,397,7313
460 DATA 41,638,572,294,504,605,368,24
1,764,560,494,858,777,61,6777
```

•

Assembly listing 1.

```
; ATARI GRAPHICS OVERLAY
; PLOTTING ROUTINE
; (C) 1984 JEFF BRENNER
;
; FORM: A=USR(PLOT,X,Y)
;
COLOR     =    $CB
LOW       =    $CC
HIGH      =    $CD
X         =    $CE
Y         =    $CF
;
          *=   $0600    ;RELOCATABLE
;
PLOT      PLA
          PLA
          PLA
          CMP #40       ;HORIZ POS>39 OR <0
          BCS RET1
          ADC #24
          STA X
          PLA
          PLA
          CMP #192      ;VERT POS>191 OR <0
          BCS RET2
          ADC #32
          STA Y
          LDX HIGH
          LDA X
          LSR A         ;DIVIDE BY EIGHT
          LSR A
          LSR A
          TAY
          BEQ OVER
          LDX HIGH
LOOP      INC HIGH      ;INC HIGH BYTE
          DEY
          BNE LOOP
OVER      ASL A         ;FIND REMAINDER
          ASL A
          ASL A
          SEC
          SBC X
          EOR #255
          TAY
          LDA #128      ;FIND CORRECT BIT PATTERN
          INY
          BEQ NEXT
LOOP1     LSR A
          DEY
          BNE LOOP1
NEXT      LDY COLOR     ;COLOR=1 TO PLOT
          BEQ ERASE     ;IF 0 - ERASE PIXEL
          LDY Y         ;OFFSET FOR VERT POS
          ORA (LOW),Y   ;MERGE BIT PATTERN
          SEC
          BCS CONT
ERASE     LDY Y
          EOR #255
          AND (LOW),Y
CONT      STA (LOW),Y   ;SUBTRACT BIT PATTERN
RETURN    STX HIGH      ;FROM CURRENT TO ERASE
          RTS
RET1      PLA
          PLA
RET2      PLA
          PLA
          RTS           ;RETURN TO BASIC
          .END
```

•

Assembly listing 2.

```
; DRAWTO ROUTINE BY JEFF BRENNER
;
; BASED ON DDA BASIC ALGORITHM
; DETAILED BY MIKE HIGGINS,
; "FAST LINE-DRAWING TECHNIQUE,"
; BYTE MAGAZINE, (AUGUST 1981),
; pp. 414-416
;
; FORM: A=USR(DRAWTO,X,Y)
;
DELTAX    =    $D4
DELTAY    =    $D5
ADDX2     =    $D6
ADDY2     =    $D7
ADDX      =    $D8
ADDY      =    $D9
SLOPECALC =    $DA
COUNT     =    $DB
OLDX      =    $CE
OLDY      =    $CF
DRAWTOX   =    97
DRAWTOY   =    96
;
          *=   $0600    ;RELOCATABLE
;
DRAWTO    PLA
```

```
        PLA
        PLA
        BMI RET1
        CMP #40       ;HORIZ POS>39 OR <0?
        BCS RET1
        ADC #24
        STA DRAWTOX   ;X TO DRAW TO
        PLA
        PLA
        CMP #0
        BCC RET2
        CMP #192      ;VERT POS>191 OR <0?
        BCS RET2
        ADC #32
        STA DRAWTOY   ;Y TO DRAW TO
        LDX #0
        STX ADDX2
        STX ADDY
        INX
        STX COUNT
        STX ADDY2
        STX ADDX
        LDA DRAWTOX
        SEC
        SBC OLDX      ;DRAWTOX-OLDX
        BCS SKIP1
        DEC ADDX
        DEC ADDX
        LDA OLDX
        SEC
        SBC DRAWTOX
SKIP1   STA DELTAX
        LDA DRAWTOY
        SEC
        SBC OLDY      ;DRAWTOY-OLDY
        BCS SKIP2
        DEC ADDY2
        DEC ADDY2
        LDA OLDY
        SEC
        SBC DRAWTOY
SKIP2   STA DELTAY
        LDA DELTAX
        CMP DELTAY
        BCS SKIP3
        LDX DELTAX    ;DELTAX<DELTAY? SWITCH
        LDA DELTAY
        STA DELTAX
        TXA
        STA DELTAY
        LDA ADDX
        STA ADDX2
        LDA ADDY2
        STA ADDY
        LDA #0
        STA ADDX
        STA ADDY2
SKIP3   LDA DELTAX
        LSR A
        STA SLOPECALC
        LDA DELTAX
        BEQ RETURN
        BNE MAIN
RET1    PLA
        PLA
RET2    RTS
MAIN    LDA OLDX      ;ADD X OFFSET
        CLC
        ADC ADDX
        STA OLDX
        LDA OLDY      ;ADD Y OFFSET
        CLC
        ADC ADDY
        STA OLDY
        INC COUNT
        LDA SLOPECALC
        CLC
        ADC DELTAY
        STA SLOPECALC
        LDA DELTAX
        CMP SLOPECALC
        BCS PLOT
        LDA SLOPECALC
        SEC
        SBC DELTAX
        STA SLOPECALC
        LDA OLDX
        CLC
        ADC ADDX2     ;SIMILAR TO ADDX
        STA OLDX
        LDA OLDY
        CLC
        ADC ADDY2     ;SIMILAR TO ADDY
        STA OLDY
PLOT    LDA DELTAX
        CMP COUNT     ;COUNT>DELTAX? DONE
        BCC RETURN
        JSR $FFFF     ;REPLACED WITH PLOT ADDR
        SEC
        BCS MAIN
RETURN  LDA DRAWTOY
        STA OLDY      ;PLOT LAST POINT
        LDA DRAWTOX
        STA OLDX
        JMP $FFFF     ;PLOT ROUTINE
        .END
```

Assembly listing 3.

```
; CLS ROUTINE, BY JEFF BRENNER
; FORM: A=USR(CLS)
;
        *= $0600      ;RELOCATABLE
;
CLS     PLA
        LDA 106       ;P/M AREA HI BYTE
        STA $D1
        LDA #0        ;LOW BYTE = 0
        STA $D0
        LDX #16       ;DO 16 PAGES
LOOP2   LDY #0
LOOP1   STA ($D0),Y
        INY
        BNE LOOP1
        INC $D1
        DEX
        BNE LOOP2
        RTS
        .END
```

BASIC Training

by Tom Hudson

In this issue of **BASIC Training**, we're going to do a little exploring in the world of computerized artificial intelligence, by flowcharting the computer's logic for the game we're developing, **High Seas**. Once we've finished outlining this logic, we'll be ready to write the game itself.

Ships ahoy!

Last issue, we began by looking at the routine which would allow the human player to place his ships on the playing grid. As you recall, we had to devote a good deal of code to error-trapping. That is, we didn't want the player to be able to unintentionally (or intentionally) cheat.

Fortunately, computers don't cheat unless they're programmed to, and ship placement is relatively easy. Figure 1 (page 76) shows the code necessary to place each ship on the playing grid.

You will notice that this flowchart is very similar to Figure 5 from last issue, except that it does not print out any error messages. If the ship doesn't fit in the space the computer has tried, it simply loops back to location 10 and tries again.

One special point of interest in the flowchart is the third decision diamond. This part of the code checks to see if the ship will touch any others if it's placed in that location. Having ships placed right next to each other is usually a rather undesirable tactic, and the computer tries to avoid it. However, people sometimes put their ships next to one another to confuse their opponent. For this reason, about five percent of the time, the computer will allow this placement, just to make it seem more human.

Obviously, this flowchart is a simplified version of what goes on inside the computer. If we drew a detailed flowchart, it would be much larger and more complicated. For the purpose of this column, this flowchart will do for now.

Let the attack begin.

Figure 2 (pages 76-78) shows the most important routine in **High Seas**, the computer's shooting logic. Without effective shooting, the game would be worthless. Therefore, a large part of the memory **High Seas** uses is devoted to artificial intelligence routines.

The first thing the AI routine does is determine the largest and smallest ships the human player has left. This is done so that the computer won't try to shoot at a place where a ship couldn't fit. For example, if the aircraft carrier (5 units long) is the smallest ship the human player has remaining, the computer shouldn't bother shooting at any places where there are less than five units of open area.

After determining the sizes of ships that are left, the computer will go into one of two shooting modes: semi-random or selective.

Semi-random shooting is used whenever there are no unresolved hits on the game grid. An unresolved hit is where a human's ship has been hit by one or more shots, but is not yet sunk. If there are no hits on the board, the computer is free to shoot wherever it wants.

I call this type of shooting semi-random because it works within special parameters. Totally random shooting would waste a lot of firepower, because you may be shooting at places where the smallest ship

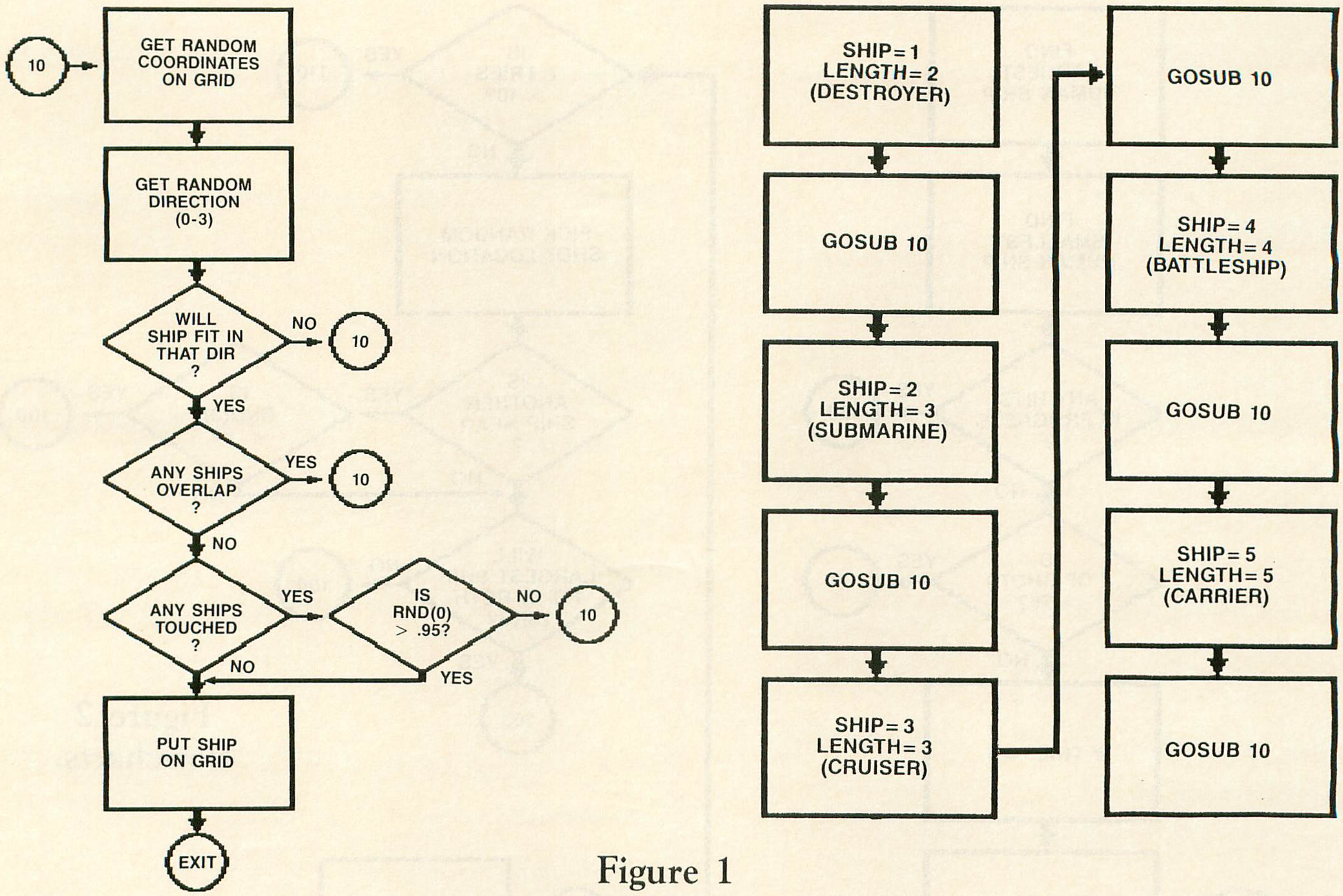


Figure 1
flowcharts.

wouldn't fit! Since the smallest ship is the destroyer (2 units), the computer is set to shoot at *every other* grid location in semi-random mode. There are two possible patterns, and the computer chooses one at random. Figure 3 shows one of these patterns.

	0	1	2	3	4	5	6	7	8	9
A	X		X		X		X		X	
B		X		X		X		X		X
C	X		X		X		X		X	
D		X		X		X		X		X
E	X		X		X		X		X	
F		X		X		X		X		X
G	X		X		X		X		X	
H		X		X		X		X		X
I	X		X		X		X		X	
J		X		X		X		X		X

Figure 3.

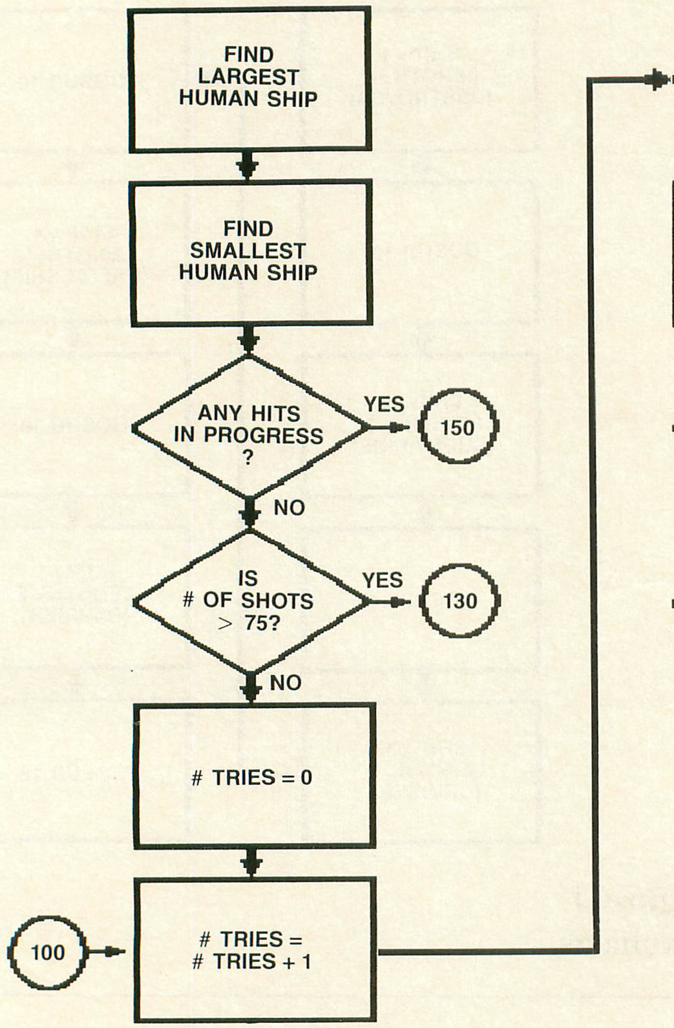
Within semi-random mode, the computer always tries to maximize its chances of hitting a ship. First, it tries to find a location where the largest ship would fit both horizontally and vertically. If it can't find such a place, it tries to find a location where the largest ship would fit either horizontally or vertically. Both these searches are done at random. After about 85 tries, the computer stops searching at random and performs the same two tests sequentially on each position of the grid. When a good candidate is found, the program branches to the firing routine (location 160 in Figure 2).

The second type of shooting, selective mode, is entered when there are unresolved hits on the game board. In this mode, the computer must locate the unresolved hit on the board, shooting at the locations around the hit until the ship is sunk.

If there is only one unresolved hit on the board, the computer does a search around the hit, based on the smallest ship left, to see if that ship could fit in the direction being tested. If the ship could fit, the shot is taken.

If more than one unresolved hit is present, the computer follows the line formed by the hits, looking for an open area to shoot at. It will keep shooting along the line of hits until it sinks the ship or runs out of open areas to shoot at.

Usually, the computer will succeed in sinking the ship. However, if the player has placed two ships next



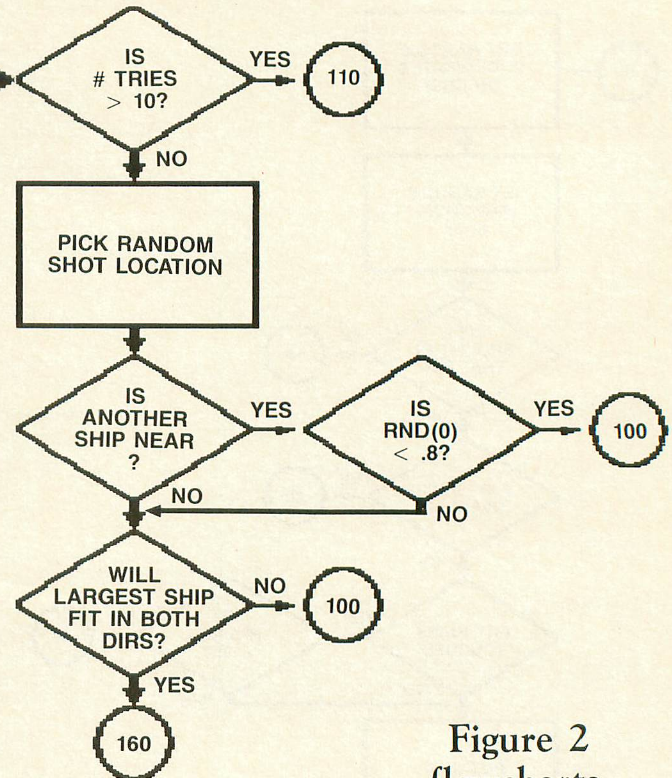
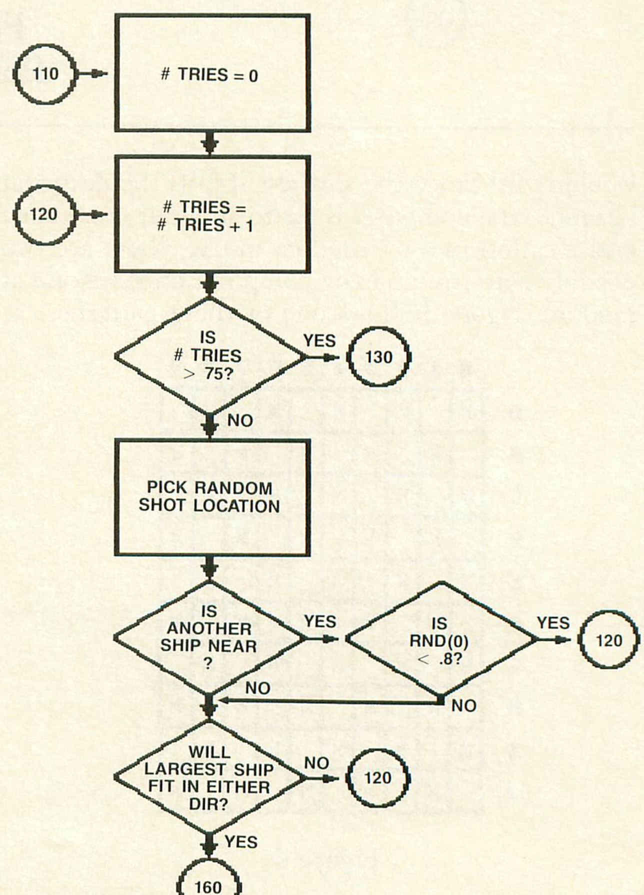


Figure 2 flowcharts.



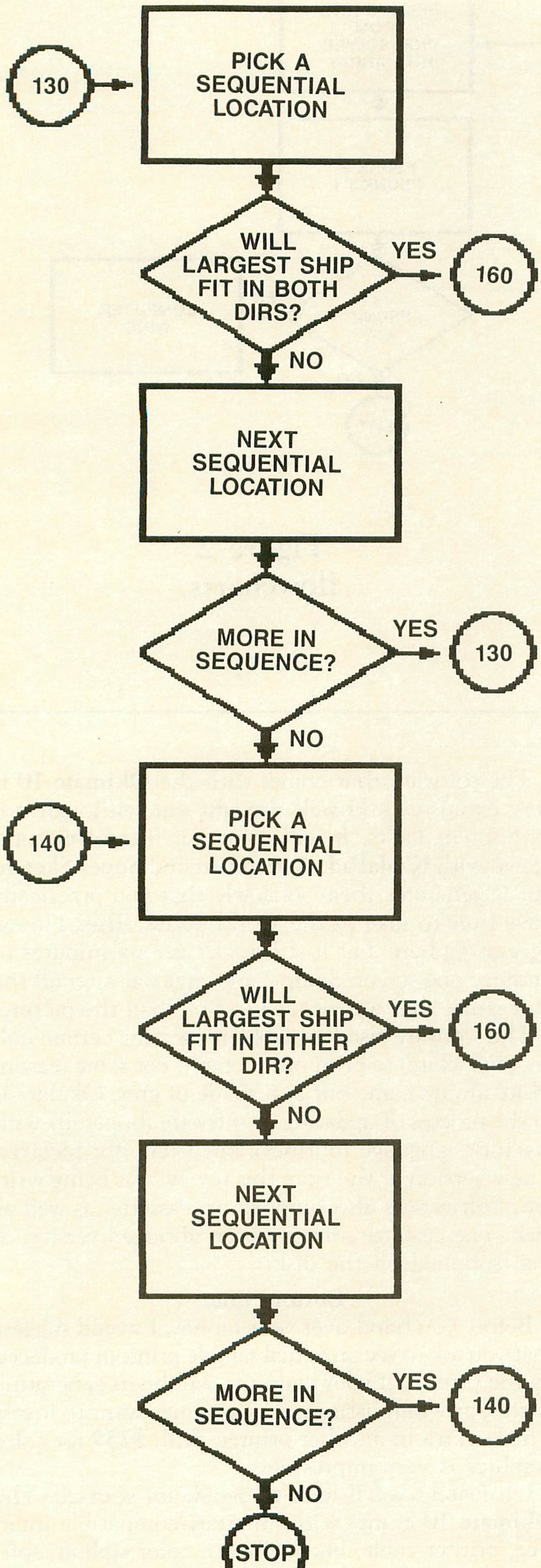
to each other, the computer could score a hit on each ship and then run out of areas to shoot at. At this point, the computer will begin using the code, starting at location 150 in Figure 2, which allows it to search around the existing hits in order to sink the ships that are clustered together.

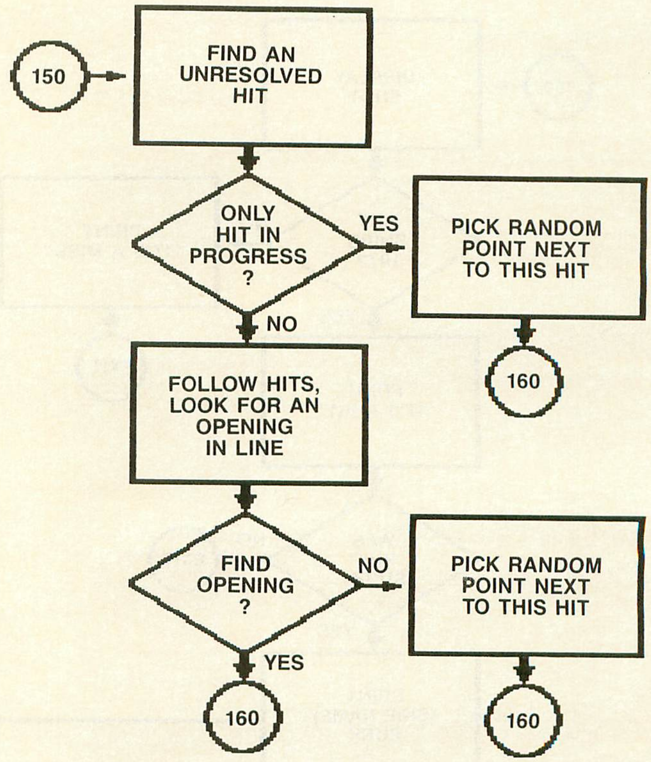
When the decision is made to shoot at a location, the code starting at 160 in Figure 2 is used to determine if the shot was a hit or a miss. If the shot is a miss, the program simply goes to the player's shooting routine. If the shot was a hit, the computer decides whether or not the ship that was hit has been sunk. If not, the program goes to the player's shooting routine.

Once a player's ship is sunk, the computer must remove the hits that sunk it from the "in progress" hit array, so that they won't be considered as active hits. Then the player's sunk ship count is incremented. If all five ships have been sunk, the game is over, and the computer wins.

We've covered all the primary routines used by **High Seas**, in some detail. Next issue's **BASIC Training** will present the full BASIC listing of **High Seas**, with a breakdown of every major section of code. □

(Figure 2 flowcharts continue on page 77)





(Figure 2 flowcharts conclude on page 78)

ATTENTION ATARI XL USERS

P/M Creator/Animator Fix

The **P/M Creator/Animator** program, as listed in **ANALOG Computing** issue 23, does not work properly in the XL computer series. To make the program operate, change the following line:

```
20012 DATA 208,2,169,160,32,176,
242,198,4,208,239,198,84,208,221
,96
```

The **ANALOG Computing** staff would like to thank Dave Tammi, of Sault Ste. Marie, Ontario for bringing this fix to our attention.

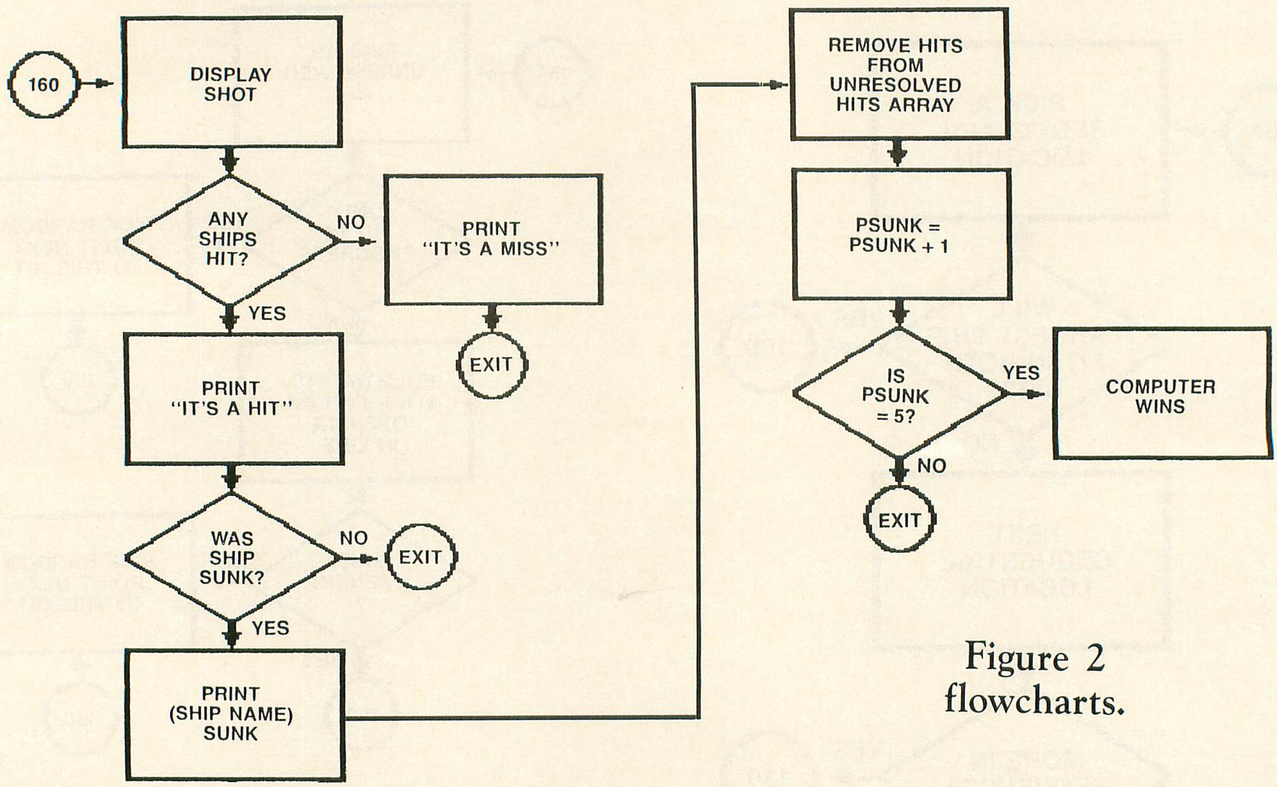


Figure 2 flowcharts.

Review *(continued from page 57)*

Not all red and rosy.

While I think that the **Okimate 10** printer is a fine piece of machinery, I'm afraid that I cannot say the same for the "Thermal Transfer Printing Technology" that this printer uses. I would in no way consider the text output produced to be of "letter quality" as stated in Okidata's press release.

The fault seems to be in the somewhat uneven deposit of ink along the length of the ribbon. This leads to letters that are too light and may have segments missing. Remember, I'm comparing this against a letter quality (a.k.a. Daisy Wheel at $600+) printer, which cannot possibly do black and white graphics, let alone color.

Before we get off the subject of ribbons, I just want to say that they don't last as long as I had hoped. A black ink ribbon will print about 120,000 characters (or about 75 whole pages of text) before needing to be replaced—and I can live with that.

Okimate's ribbons, however, are a horse of a different color (sorry about that). Since a color ribbon gets used up three times faster (remember, it has to make three passes for each line), you can get only about 35,000 characters from one ribbon. Or, in terms of color screen dumps (which is what I assume you might want the **Okimate 10** for), this works out to about ten pictures per ribbon. At $6.69 per color ribbon, you end up paying about 67 cents per picture. Ouch!

The software that comes with the **Okimate 10** is very extensive and well thought out, although not without its faults. It will print out files of pictures drawn with **KoalaPad**, **AtariArtist** and **Super Sketch**, but it generates them so slowly that you practically have time to take in the World Series. True, I'm exaggerating here, but it still took over six minutes to produce one screen dump—and that was after all the processing that was needed to first load the picture.

The software also had problems getting certain colors (like white) to print out properly. For some reason, white always came out as a shade of grey. Okidata is in the process of updating its software (hopefully with machine language routines), but I had not received a new version at the time this review was being written. Software is also supplied on cassette, as well as disk. The cassette software is an abridged version of that supplied on the disk.

Bottom line.

Before you hand over your money, I would suggest that you ask to see an actual sample printout produced by the printer. If your main use will be in generating letters, program listings, etc., you may want to invest a little extra in another printer. Still, $239 for color graphics is very impressive.

Ultimately, you'll have to decide for yourself. The **Okimate 10** comes with an Atari-compatible interface, printer cable, black ribbon, color ribbon, software on cassette and disk, printer paper and, of course, an instruction manual. □

PuLse in Action!

16K Cassette or 24K Disk

by Joel Gluck

It's not easy to write an article to accompany a simple graphic demo, and it's probably just as tough to read one. But before you race to your computer, plug in your Action! cartridge, type in **puLse** and run it, why not take a few seconds to read what it's taken a few hours for me to write?

For starters, I'll admit that what **puLse** does is beautiful. The program draws several brightly-colored horizontal lines on the screen and makes them expand and contract at various rates. This sounds simple, but the combinations are almost infinite—and frequently complex. When I first wrote **puLse** and ran it, I shut off the room lights and spent a good long time staring at the screen. No, the effect is not hypnotic, and, no, I wasn't on drugs at the time—the fact is that, as a human being, I appreciate beauty, and **puLse** gave me a sizable dose.

Despite its beauty, however, **puLse** is not a work of art. To me, a work of art must relate to the human experience, and **puLse** is merely a random and abstract visual creation. It does not affect me deeply, the way a good novel, play or piece of music can. The best way to describe it is "emptiness"—**puLse** is empty. Rated as art, it is bad art.

I'm not saying that I'm going to stop fooling around with graphic demos on my Atari 800. All I'm saying is that there is more to art than beauty or simple emotional effects, and this fact is a challenge to myself and to all creators of "computer art." Translation: I'm still thinking. I hope you are, too.

Using the program.

Plug in an Action! cartridge and type the source code as written into the editor. Save it to disk or tape and then run.

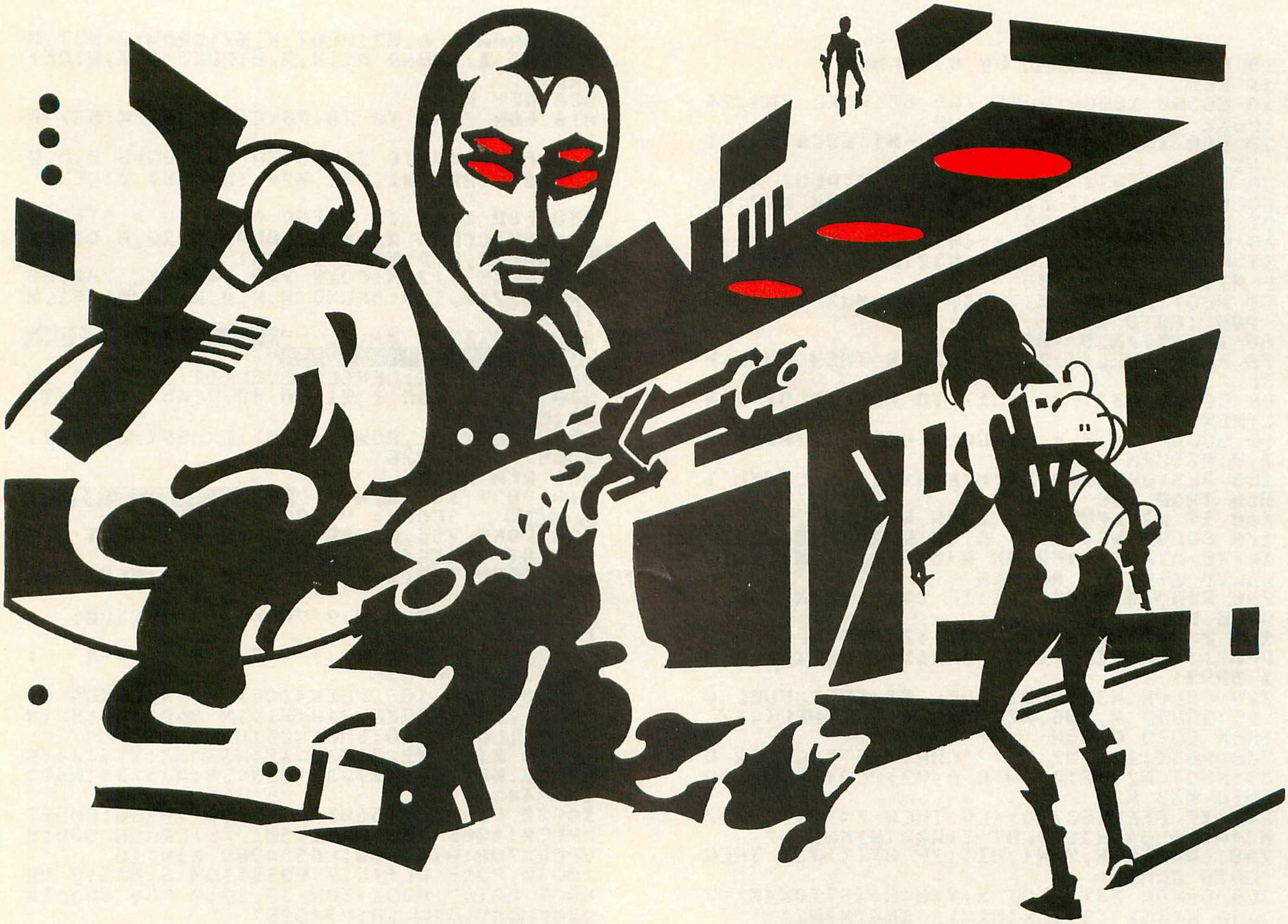
PuLse is a simple graphic demo. For the greatest effect, run the program with all room lights extinguished. Press RETURN to exit the program. Press any other keyboard key for new patterns. The program will automatically display a new pattern every 15-20 seconds. □

Action! listing.

```
; puLse - joeL gLuck - anaLog

BYTE ARRAY x(192),y(192),c(192)
BYTE num=[10]
INT ARRAY xd(192)
CARD ARRAY linept(192)


PROC pauz(CARD n)
CARD i

FOR i=0 TO n+n
DO OD
RETURN


PROC intro()
BYTE i,COLOR1=709

Graphics(0)
Poke(710,0)
Poke(752,1)
Print(" ")
pauz(30000)
Position(17,9)
Print("puLse")
```

```
pauz(30000)
Position(13,11)
Print("by joeL gLuck")
pauz(30000)
Position(9,13)
Print("from ANALOG COMPUTING")
pauz(60000)
FOR i=0 TO 15 DO
  COLOR1=15-i
  pauz(2000)
OD
RETURN


PROC gr10init()
CARD line,reg,col,lum,scrn=88
BYTE ARRAY gtiacol(8)=705

Graphics(10)
Poke(704,0)
FOR reg=0 TO 7 DO
  col=Rand(16)
  lum=Rand(9)+4
  gtiacol(reg)=col*16+lum
OD
FOR line = 0 TO 191 DO
  linept(line)=scrn+40*line
OD
RETURN


PROC plot10(BYTE x,y,col)
BYTE POINTER pixel
BYTE ARRAY
     colfil= [0 17 34 51 68 85 102
             119 136 0 0 0 0 0 0 0],
     mask= [15 240],
     mask2= [240 15]

pixel=linept(y)+(x RSH 1)
pixel^=pixel^ & mask(x & 1)
             % (colfil(col)
             & mask2(x & 1))
RETURN


BYTE FUNC locate10(BYTE x,y)
BYTE POINTER pixel
BYTE ARRAY mask= [240 15]

pixel= linept(y)+(x RSH 1)
RETURN((pixel^ & mask(x & 1)) RSH
     (((x & 1) XOR 1) LSH 2))


PROC drawline(BYTE a,b,c)
BYTE i

FOR i=a TO 79-a DO
  plot10(i,b,c)
  plot10(i,191-b,c)
OD
RETURN


PROC init()
BYTE i,j,s

gr10init()
FOR i=0 TO num-1 DO
  x(i)=Rand(40)
  DO
    y(i)=Rand(96)
    s=0
    IF i=0 THEN
      EXIT
    FI
    FOR j=0 TO i-1 DO
      IF y(i)=y(j) THEN
        s=1
        EXIT
      FI
    OD
    UNTIL s=0
  OD
  c(i)=(i MOD 8)+1
  drawline(x(i),y(i),c(i))
  xd(i)=(Rand(2)*2-1)*(Rand(3)+1)
OD
RETURN


PROC squeeze(BYTE n)
BYTE a,i
INT d

d=xd(n)
IF d<0 THEN
  d=-1
ELSE
  d=1
FI
FOR i=1 TO (xd(n)/d) DO
  a=x(n)+d
  IF a>39 THEN
    xd(n)=-xd(n)
    EXIT
  FI
  IF d>0 THEN
    plot10(x(n),y(n),0)
    plot10(79-x(n),y(n),0)
    plot10(x(n),191-y(n),0)
    plot10(79-x(n),191-y(n),0)
  ELSE
    plot10(x(n),y(n),c(n))
    plot10(79-x(n),y(n),c(n))
    plot10(x(n),191-y(n),c(n))
    plot10(79-x(n),191-y(n),c(n))
  FI
  x(n)=a
  plot10(x(n),y(n),c(n))
  plot10(79-x(n),y(n),c(n))
  plot10(x(n),191-y(n),c(n))
  plot10(79-x(n),191-y(n),c(n))
OD
RETURN


PROC puLse()
BYTE i,CH=764,AT=77,MS=19,LS=20

intro()
DO
  num=Rand(6)+5
  init()
  CH=255
  AT=77
  LS=0
  MS=0
  DO
    FOR i=0 TO num-1 DO
      squeeze(i)
    OD
    UNTIL CH<>255 OR MS=4
  OD
  UNTIL CH=12
OD
CH=255
Graphics(0)
RETURN
```

•



ROBOT RAID

16K Cassette or 24K Disk

by Charles Kormos

You are the commander of a group of forces rebelling against the evil galactic empire and have just received great news from one of your most trusted spies. The empire's mother ship is in for repairs at a dock inside an asteroid. You know of a long tunnel leading to the repair bay where the ship sits helpless. The old tunnel is littered with energy pods, unpenetrable force fields, airlocks, cave-ins, and twists and turns far too difficult for any manned craft. But, luckily, you have three robot ships you can use with your Atari computer, TV and joystick. This is your *only* chance to destroy the empire's mother ship.

Along the way, you will use the joystick to move left or right. Pressing the stick forward will engage the hyperdrive, which will move your robot inexorably forward—even through cave walls. Just be careful not to end up in rock when you come *out* of hyperspace.

You will use the joystick button to fire torpedoes which can blast energy pods and give energy to your ship, blast cave walls or airlocks. Force fields, however, cannot be destroyed by torpedo fire. Watch your energy gauge and listen to your engines. If you run out of energy, that robot is destroyed.

Your robot ship must pass through twenty-four screens—without bumping into anything or running out of energy—to get to the repair bay.

Now use your scanners. Position your ship so the scanner line is in the center of the gauge, and you'll be on target to blast through to the defective nuclear reactor. One torpedo hit there will nuke the ship. But the mother ship is not completely defenseless during this time. It will be moving toward your robot, trying to crush it.

When you finish this mission, you'll receive three more robots and a thirty-six screen mission, with a forty-eight screen mission after that one. The author has yet to be successful at the final mission, so good luck. □

(Listing starts on page 82)

```
10 REM ROBOT RAID by C.Kormos
11 REM
15 GOSUB 10000:POKE 708,68:POKE 709,24
:POKE 710,148:GOTO LOOP
20 FOR I=Y-N1 TO 4 STEP -N1:LOCATE X,I
,HIT
30 IF HIT<N34 THEN COLOR N33:PLOT X,I:
COLOR N32:PLOT X,I:NEXT I:COLOR N32:PL
OT X,I-N1:GOTO 90
40 POP :SOUND N1,64-(HIT=N35)*N35,8+(H
IT=N34)*N2,N10:COLOR N32:PLOT X,I+N1:I
F HIT=N34 THEN 90
50 PLOT X,I:IF HIT=N40 AND NUKE=X THEN
 POP :GOTO WIN
60 IF HIT>N35 THEN 90
70 FUEL=FUEL+4:IF FUEL>N38 THEN FUEL=N
38
80 COLOR N37:FOR I=17 TO FUEL:PLOT I,N
1:NEXT I
90 SOUND N1,N,N,N:FUEL=FUEL-0.3:POKE 7
7,N:RETURN
150 RESTORE MAP(M):M=M+N1:IF M=LEVEL T
HEN TRAP 400
160 IF M>=LEVEL-N2 THEN RETURN
170 COLOR N34:PLOT A,N2:DRAWTO B,N2:Z=
INT(B-A)/N2+A:COLOR N38:PLOT Z-N1,N2:D
RAWTO Z+N1,N2:RETURN
200 READ A,B:C=C+N1:IF C=12 THEN C=N:G
OSUB MAP
210 Z=USR(ML,1570,POW1):Z=USR(ML,POW1,
POW2):Z=USR(ML,1580,BAR1):Z=USR(ML,BAR
1,BAR2)
220 COLOR N32:PLOT FUEL,N1:FUEL=FUEL-0
.15:SOUND N,200,N2,FUEL/6:IF FUEL<=17
THEN GOTO CRASH
230 POSITION N2,N2:? CHR$(157):COLOR N
39:PLOT N2,N2:DRAWTO A,N2:PLOT B,N2:DR
AWTO N38,N2
240 IF PEEK(53770)<LV THEN Z=INT(B-A)/
N2+A:COLOR N35:PLOT Z+RND(N)*N2,N2
250 LOCATE X,Y-N1,HIT:IF HIT<>N32 THEN
 GOTO CRASH
260 COLOR N32:PLOT X,Y:X=X+(STICK(N)=7
 AND X<38)-(STICK(N)=11 AND X>N2)
270 COLOR N36:PLOT X,Y:IF  NOT STRIG(N
) THEN GOSUB SHOT
280 IF STICK(N)=14 THEN GOSUB HYPER
290 POSITION N34,N:? SCORE;:SCORE=SCOR
E+N1:Z=USR(ML,POW1,1570):Z=USR(ML,BAR1
,1580):GOTO LOOP
400 A=N1:B=N35
410 POSITION N2,N2:? CHR$(157);SHIP$(A
,B):A=A+N35:B=B+N35:IF A>491 THEN A=49
1:B=525
420 COLOR N32:PLOT FUEL,N1:FUEL=FUEL-0
.15:IF FUEL<=19 THEN GOTO CRASH
430 LOCATE X,Y-N1,HIT:IF HIT<>N32 THEN
 GOTO CRASH
440 COLOR N32:PLOT X,Y:X=X+(STICK(N)=7
 AND X<38)-(STICK(N)=11 AND X>N2):COLO
R N36:PLOT X,Y
450 COLOR N44:PLOT N10,N1:DRAWTO 12,N1
:IF  NOT STRIG(N) THEN GOSUB SHOT
470 COLOR 43:PLOT 11+SGN(NUKE-X),N1:GO
TO 410
600 FOR I=N10 TO 200 STEP 5:SOUND N,12
1,8,N10:SOUND N1,I,8,N10:SOUND N2,255-
I,8,N10:POKE 712,I:NEXT I
610 SOUND N,N,N,N:SOUND N1,N,N,N:SOUND
 N2,N,N,N:POKE 712,N:Z=USR(ML,POW1,157
0):Z=USR(ML,BAR1,1580)
620 POSITION SHIP,N1:? " ":SHIP=SHIP-N
2:IF SHIP<N1 THEN GOTO LOSE
630 FOR I=N1 TO 500:NEXT I:FOR I=N1 TO
 24:POSITION N2,N2:? CHR$(157):NEXT I:
RESTORE 20000:M=N1:FUEL=N38:C=N
650 POSITION N10,N1:? "***":COLOR N37:
FOR I=19 TO FUEL:PLOT I,N1:NEXT I:SOUN
D N1,200,N,N2:GOTO LOOP
700 FOR I=200 TO 80 STEP -N10:SOUND N1
,I,N,15:POKE 712,PEEK(53770)
710 READ A,B:C=C+N1:IF C=12 THEN C=N:G
OSUB MAP
720 COLOR N32:PLOT FUEL,N1:FUEL=FUEL-0
.3:POSITION N2,N2:? CHR$(157):COLOR N3
9:PLOT N2,N2
730 DRAWTO A,N2:PLOT B,N2:DRAWTO N37,N
2:NEXT I:SOUND N1,N,N,N:POKE 712,N:RET
URN
800 REM WIN
810 FOR Z=N1 TO 20:POKE 712,PEEK(53770
)
820 FOR D=N TO 255 STEP 20:SOUND N,D,N
10,15:SOUND N1,D+2,N10,15:NEXT D:NEXT
Z
830 FOR Z=N1 TO 5:FOR D=15 TO N STEP -
0.5:SETCOLOR N2,N,D:SOUND N,20,N,D:SET
COLOR N1,N,D:NEXT D:NEXT Z
840 POKE 712,N:POKE 709,24:POKE 708,68
:POKE 710,148:SOUND N,N,N,N:SOUND N1,N
,N,N
850 POSITION N1,N:? "               MISSION
ACCOMPLISHED!          "
860 LV=LV-N2:LEVEL=LEVEL+12:RESTORE 19
070+LEVEL:FOR I=N1 TO 48:READ A:MAP(I)
=A:NEXT I
870 Z=USR(ML,POW1,1570):Z=USR(ML,BAR1,
1580):GOTO 920
900 REM LOSE
910 POSITION N,N:? "   ALL ROBOTS DESTR
OYED        SCORE: ";SCORE:SCORE=N
920 POKE 755,N:FOR D=N TO N32:NEXT D:I
F  NOT STRIG(N) THEN 950
940 POKE 755,2:FOR D=N TO N32:NEXT D:G
OTO 920
950 POSITION N,N:? "ROBOTS  SENSORS
ENERGY   SCORE:          "
960 POSITION N,N1:? " $ $ $     ***  ":
SHIP=5:GOTO 630
10000 POKE 106,PEEK(106)-8:GRAPHICS N:
POKE 752,1:POKE 710,N:DIM MAP(48):X=18
:Y=23:LOOP=200:SHOT=20:MAP=150
10010 N1=1:N2=2:N32=32:N33=33:N34=34:N
35=35:N36=36:N37=37:N38=38:N39=39:N40=
40:N44=44
10020 DL=PEEK(560)+PEEK(561)*256:CHSET
=PEEK(106)*256:M=N1:SHIP=5:CRASH=600:H
YPER=700:WIN=800:LOSE=900:N10=10
10030 POKE DL+15,7:POSITION 5,N10:? "R
OBOT RAID":POSITION 30,12:? "by Charle
yKormos":DIM SHIP$(525)
10040 FOR I=N TO 1023:POKE CHSET+I,PEE
K(57344+I):NEXT I
10050 NUSET=CHSET+16:POSITION 5,N10:?
"              ":POSITION 30,12:? "
               "
10060 POW1=CHSET+24:BAR2=CHSET+104:POW
2=CHSET+112:BAR1=NUSET:FOR I=N TO 103:
READ A:POKE NUSET+I,A:NEXT I
10070 FOR I=DL+6 TO DL+28:POKE I,4:NEX
T I:POKE 756,CHSET/256:POKE 710,148
10090 ML=1536:FOR I=ML TO 1560:READ A:
POKE I,A:NEXT I:LV=29:LEVEL=25
10100 FOR I=N1 TO 48:READ A:MAP(I)=A:N
EXT I:RESTORE 20000
10110 POSITION N,N:? "ROBOTS  SENSORS
  ENERGY   SCORE:"
10120 POSITION N,N1:? " $ $ $     ***
":FUEL=N38:COLOR N37:FOR I=17 TO FUEL:
PLOT I,N1:NEXT I
10125 SHIP$=" ":SHIP$(525)=" ":SHIP$(N
2)=SHIP$
10130 SHIP$(13)="* *        * *":SHIP$(4
8)="***         ***"
10140 SHIP$(84)="*          *":SHIP$(114
)="**********   *********"
10150 SHIP$(149)="*********** *********
         ****************************
**"
10160 SHIP$(211)="+++)+)+)+)+)+)+)+)+)
+)+)+)+)+)+)+++  ********************
***********"
10170 SHIP$(289)="*********************
":SHIP$(324)="*********** *********"
10180 SHIP$(364)=",,          ,,":SHIP$(3
98)=",,,,,,,,,,,,,,,,"
10190 SHIP$(431)=",(((((((((((((((,":S
HIP$(472)=",,,,,"
10200 SHIP$(525)=" ":NUKE=INT(RND(N)*1
5)+13:RETURN
19034 DATA 0,0,0,255,255,0,0,0
```

(Listing continues on next page)

(continued from previous page)

```
19035 DATA 170,170,130,130,130,130,170
,170
19036 DATA 20,85,85,20,20,130,130,130
19037 DATA 27,27,27,27,27,27,27,27
19038 DATA 170,195,195,195,195,195,195
,195
19039 DATA 175,245,95,250,175,245,90,2
50
19040 DATA 85,85,170,175,175,170,85,85
19041 DATA 170,170,85,95,95,85,170,170
19042 DATA 255,255,255,255,255,255,255
,255
19043 DATA 170,170,170,170,170,170,170
,170
19044 DATA 85,85,85,85,85,85,85,85
19045 DATA 85,85,0,255,255,0,85,85
19046 DATA 0,0,60,60,60,60,0,0
19059 DATA 104,104,133,204,104,133,203
,104,133,207,104,133,206,160,0,177,206
,145,203,200,192,8,208,247,96
19060 DATA 20040,20050,20100,20150,201
60,20110,20030,20120,20130,20080,20100
,20110,20060,20120,20130
19070 DATA 20070,20140,20090,20010,200
20,20040,20170,20180,20190,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
19107 DATA 20040,20010,20050,20030,200
70,20100,20110,20120,20130,20160,20150
,20140,20090,20080,20040,20060,20020
19108 DATA 20010,20100,20150,20050,200
60,20100,20110,20120,20130,20080,20070
,20080,20120,20090,20100,20110,20170
19109 DATA 20180,20190,0,0,0,0,0,0,0,0
,0,0,0,0
19119 DATA 20170,20180,20190,20000,200
10,20050,20010,20050,20030,20070,20030
,20070,20100,20110,20120,20090
19120 DATA 20100,20150,20140,20130,200
80,20070,20060,20050,20040,20030,20020
,20010,20170,20000,20040,20080
19121 DATA 20010,20100,20150,20100,201
10,20120,20130,20120,20130,20160,20110
,20040,20170,20180,20190,20190
20000 DATA 4,32,5,31,6,30,7,29,8,28,9,
27,10,26,11,25,12,24,13,23,14,22,14,22
20010 DATA 14,22,14,22,11,24,8,22,5,20
,3,18,3,16,3,15,5,16,8,18,11,20,14,22
20020 DATA 14,22,5,23,6,24,7,25,8,26,9
,27,10,28,11,29,12,30,13,31,14,22,14,2
2
20030 DATA 14,22,14,22,13,23,12,24,12,
24,14,22,14,22,10,18,11,19,12,20,13,21
,14,22
20040 DATA 14,22,14,22,8,28,8,28,8,28,
8,28,5,31,5,31,11,25,12,24,13,23,14,22
20050 DATA 14,22,11,20,8,18,5,16,3,15,
3,16,3,18,5,20,8,22,11,24,14,22,14,22
20060 DATA 14,22,14,22,13,31,12,30,11,
29,10,28,9,27,8,26,7,25,6,24,5,23,14,2
2
20070 DATA 14,22,13,21,12,20,11,19,10,
18,14,22,14,22,12,24,12,24,13,23,14,22
20080 DATA 14,22,13,23,12,24,11,25,5,3
1,5,31,8,28,8,21,8,21,8,21,14,22,14,22
20090 DATA 3,12,3,12,4,13,5,14,6,15,7,
16,8,17,9,18,10,19,11,20,12,21,13,22
20100 DATA 14,23,15,24,16,25,17,26,18,
27,19,28,20,29,21,30,22,31,23,32,24,33
,24,33
20110 DATA 24,33,24,33,23,32,22,31,21,
30,20,29,19,28,18,27,17,26,16,25,15,24
,14,23
20120 DATA 13,22,12,21,11,20,10,19,9,1
8,8,17,7,16,6,15,5,14,4,13,3,12,3,12
20130 DATA 3,12,3,12,3,13,3,14,3,15,3,
16,3,17,3,18,3,19,3,20,3,21,14,22
20140 DATA 14,22,3,21,3,20,3,19,3,18,3
,17,3,16,3,15,3,14,3,13,3,12,3,12
20150 DATA 24,33,24,33,22,33,21,33,20,
33,19,33,18,33,17,33,16,33,15,33,14,33
,14,22
20160 DATA 14,22,14,33,15,33,16,33,17,
33,18,33,19,33,20,33,21,33,22,33,24,33
,24,33
20170 DATA 14,22,13,23,12,24,11,25,10,
26,9,27,8,28,7,29,6,30,5,31,4,32,3,33
20180 DATA 3,33,3,33,3,33,3,33,3,33,3,
33,3,33,3,33,3,33,3,33,3,33,3,33
20190 DATA 3,33,3,33,3,33,3,33,3,33,3,
33,3,33,3,33,3,33,3,33,3,33,3,33
```

•

CHECKSUM DATA.

(see page 10)

```
10 DATA 342,254,823,500,267,592,339,83
6,690,440,815,640,536,69,118,7261
210 DATA 869,140,489,376,761,60,808,84
8,465,786,275,259,759,788,259,7942
470 DATA 693,373,641,708,695,757,922,1
31,412,554,555,922,744,286,995,9388
850 DATA 898,572,967,797,190,797,724,6
63,919,459,227,590,576,217,630,9226
10060 DATA 726,304,731,381,349,45,751,
601,858,309,12,421,937,878,425,7728
19034 DATA 333,543,161,926,675,595,105
,117,625,567,935,926,350,750,144,7752
19070 DATA 150,298,305,954,735,706,730
,886,611,869,346,901,631,742,787,9651
20080 DATA 879,539,418,420,425,170,141
,403,405,654,339,341,5134
```

•

CIRCLE #135 ON READER SERVICE CARD

CIRCLE #136 ON READER SERVICE CARD

CIRCLE #137 ON READER SERVICE CARD

CIRCLE #138 ON READER SERVICE CARD

CIRCLE #139 ON READER SERVICE CARD

Boot Camp

by Tom Hudson

This month, **Boot Camp** continues its coverage of BASIC USR calls, with a couple of new twists, including variable-argument calls. So, if you haven't read the last **Boot Camp**, I urge you to check it out. The information it contains is vital to this installment.

An "average" challenge.

I hope all **Boot Camp** readers tried to solve last issue's problem, which was to write a USR call that would accept two arguments and find their average. This is done by adding the two arguments and dividing the total by two. Figure 1 shows one possible solution.

```
10 RESLO    =      $D4
20 RESHI    =      $D5
30          *=     $0600
0100        CLD          ;DECIMAL MODE!
0110        PLA          ;DISCARD # ARGS
0120        PLA          ;PULL ARG 1 HI
0130        STA ARG1H    ;AND SAVE IT
0140        PLA          ;PULL ARG 1 LO
0150        STA ARG1L    ;AND SAVE IT
0160        PLA          ;PULL ARG 2 HI
0170        STA ARG2H    ;AND SAVE IT
0180        PLA          ;PULL ARG 2 LO
0190        STA ARG2L    ;AND SAVE IT
0200        LDA ARG1L    ;ADD...
0210        CLC          ;ARGUMENT 1...
0220        ADC ARG2L    ;TO...
0230        STA TEMPL    ;ARGUMENT 2...
0240        LDA ARG1H    ;AND...
0250        ADC ARG2H    ;PLACE IT...
0260        STA TEMPM    ;IN TEMP!
0270        LDA #0
0280        ADC #0
0290        STA TEMPH
0300        LSR TEMPH    ;DIVIDE...
0310        ROR TEMPM    ;ARG1 + ARG2...
0320        ROR TEMPL    ;BY 2
0330        LDA TEMPL    ;AND PUT...
0340        STA RESLO    ;FINAL RESULT...
0350        LDA TEMPM    ;IN BASIC'S...
0360        STA RESHI    ;ANSWER AREA!
0370        RTS          ;ALL DONE!
0380 ARG1L  *=*+1        ;ARGUMENT 1
0390 ARG1H  *=*+1
0400 ARG2L  *=*+1        ;ARGUMENT 2
0410 ARG2H  *=*+1
0420 TEMPL  *=*+1        ;TEMPORARY HOLD
0430 TEMPM  *=*+1
0440 TEMPH  *=*+1
0450        .END
```

Figure 1.

This solution is probably very close to the one most beginning assembly programmers would come up with and is, therefore, less efficient than it could be. We'll look at a better solution in a moment, but first let's analyze this one.

Line 100 clears the decimal mode. I know I repeat myself a lot on this point, but it's important: *always* be sure of the decimal mode setting!

Line 110 pulls the first byte from the stack. This is the number of parameters passed to the subroutine by BASIC, and we'll simply discard it, assuming the BASIC program sent two parameters.

Lines 120-150 pull the first argument from the stack and store it in the locations ARG1L and ARG1H for later processing.

Lines 160-190 pull and store the second argument, placing it in the locations ARG2L and ARG2H.

All right, so far we've placed the two numbers we're going to process in the proper memory locations, and we're ready to average them.

One factor to consider at this point is an important thing to remember in assembly language: data overflow. In this averaging subroutine, we must add the two arguments, then divide this number by two in order to get our answer. We must make sure that the place we use to store the result of the addition is large enough to hold it! For example, if we add the two-byte values 45960 and 37902, we get the result 83862, which requires a *three*-byte storage area. In this "beginner's" average routine, we have set up a three-byte area—TEMPL, TEMPM and TEMPH—to hold the result of the addition.

Lines 200-290 add the two arguments and place them in the temporary hold area.

Lines 300-320 divide the value found in the temporary storage area by two. Remember that performing a logical shift right (LSR) on a byte will divide its contents by two. The rotate right (ROR) operation is used on subsequent bytes of a multiple-byte shift. If you are unclear about this process, re-read issue 19's **Boot Camp**. After this divide is complete, the average of the two values is present in the TEMPL and TEMPM bytes.

Lines 330-360 move the final result to RESLO and RESHI, the two-byte location which returns the USR call's result to BASIC.

Finally, **Line 370** executes an RTS instruction, returning to BASIC.

Figure 2 shows the BASIC code needed to execute this USR call. When two values are entered, the subroutine returns the average of the two numbers to BASIC, which prints them.

```
10 FOR X=1536 TO 1599:READ N:POKE X,N:
NEXT X
20 ? "ENTER VALUE1, VALUE2";:TRAP 20:I
NPUT VAL1,VAL2
30 A=USR(1536,VAL1,VAL2)
40 PRINT A
50 GOTO 20
60 DATA 216,104,104,141,65,6,104,141,6
4,6,104,141,67,6,104,141,66,6,173,64,6
,24,109,66,6
70 DATA 141,68,6,173,65,6,109,67,6,141
,69,6,169,0,105,0,141,70,6,78,70,6,110
,69,6
80 DATA 110,68,6,173,68,6,133,212,173,
69,6,133,213,96
```

Figure 2.

As I mentioned earlier, the solution shown above is a typical beginner's answer to the problem. Let's examine a more efficient example, shown in Figure 3.

```
10 RESLO   =    $D4
20 RESHI   =    $D5
30         *=   $0600
0100       CLD            ;DECIMAL MODE!
0110       PLA            ;DISCARD # ARGS
0120       PLA            ;PULL ARG 1 HI
0130       STA ARG1H      ;AND SAVE IT
0140       PLA            ;PULL ARG 1 LO
0150       STA ARG1L      ;AND SAVE IT
0160       PLA            ;PULL ARG 2 HI
0170       STA RESHI      ;SAVE IT
0180       PLA            ;PULL ARG 2 LO
0190       CLC            ;ADD IT TO
0200       ADC ARG1L      ;ARG 1 LO
0210       STA RESLO      ;STORE ANSWER
0220       LDA RESHI      ;GET ARG2 HI
0230       ADC ARG1H      ;ADD TO ARG 1 HI
0240       STA RESHI      ;STORE ANSWER
0250       ROR RESHI      ;DIVIDE...
0260       ROR RESLO      ;RESULT BY 2
0270       RTS            ;AND EXIT!
0280 ARG1L *=*+1
0290 ARG1H *=*+1
0300       .END
```

Figure 3.

Looking at this new example, you will notice that it is much shorter than Figure 1 and has only two bytes reserved for data storage. Let's see why.

Line 100 clears the decimal mode, as usual.

Line 110 pulls the number of arguments off the stack.

Lines 120-150 perform the same function as in Figure 1, pulling the first argument off the stack and placing it in the ARG1L and ARG1H locations.

From this point on, the program's operation is quite different from that of Figure 1. We no longer use the TEMP area to hold the results of the add. Instead, we use the two-byte BASIC result area, RESLO and RESHI.

"Wait a minute," you say, "what about the overflow problem?" You're right; we'll have to consider that. However, because of the way this problem is structured, we can use a simple trick involving the carry flag to eliminate the danger of an overflow occurring.

In our problem, the largest argument values possible (using our two-byte format) are 65535 ($FFFF) and 65535 ($FFFF), which, when added, give a result of 131070 ($1FFFE). If we're going to store this value (as we did in Figure 1), we must have a three-byte field in which to do it. Figure 4 shows the three-byte field before and after the shift operation. Note that, after the shift, the two lower-order bytes contain $FFFF, which is, of course, the average of the values $FFFF and $FFFF.

```
BEFORE SHIFT:
 TEMPH      TEMPM      TEMPL
|00000001|11111111|11111110| = $1FFFE

AFTER SHIFT:
 TEMPH      TEMPM      TEMPL
|00000000|11111111|11111111| = $FFFF
```

Figure 4.

(continued on next page)

The thing that makes this particular problem easier to solve is the fact that, as soon as we add the values, we perform the right-shift operations to divide the answer by two. As you recall, when multi-byte values are added, the carry flag holds the bits that are to be carried to the next byte. In this case, only one bit ever gets carried to the third byte of TEMP, and, therefore, we can simply leave that bit in the carry flag—rather than storing it in a third byte. Since that bit stays in the carry flag, we can use the *rotate* instruction to shift the bit out of the carry flag and into the high-order byte of our result. Figure 5 shows the use of this technique with the same problem as Figure 4. Note, once again, that the result is $FFFF, the correct average.

```
BEFORE ROTATE:
CARRY    RESHI      RESLO
[1]     [11111111|11111110]

AFTER ROTATE:
CARRY    RESHI      RESLO
[0]     [11111111|11111111]
```

Figure 5.

Lines 160-170 pull the high-order byte of argument 2 from the stack and place it in the location labeled RESHI. Remember, values stored in RESHI and RESLO will be returned to BASIC automatically.

Line 180 pulls the low-order byte of argument 2 from the stack, leaving it in the accumulator.

Lines 190-200, instead of storing the low byte of argument 2, add it to the low byte of argument 1, which was stored earlier. The result of this add (the first part of a two-byte add) is stored in RESLO at Line 210.

Lines 220-240 complete the addition process by adding ARG1H (the high byte of argument 1) to RESHI (the high byte of argument 2, which we stored earlier), and storing the result in RESHI. At this point—if we were averaging $FFFF and $FFFF—RESLO, RESHI and the carry flag would look exactly like the first part of Figure 5. We're now ready to divide this answer by two for our average.

Line 250 performs a rotate right (ROR) on RESHI. This is the same as a shift operation, except that the carry flag is shifted into the leftmost bit, and the rightmost bit is shifted into the carry flag. Internally, the operation looks like this:

```
CARRY     RESHI      CARRY
[1]--->[11111111]--->[1]
```

Line 260 performs another rotate right, but this time the byte RESLO is rotated. The carry from the previous rotate (Line 250) is shifted into the leftmost bit, and the rightmost bit is placed in the carry flag, like so:

```
CARRY     RESLO      CARRY
[1]--->[11111110]--->[0]
```

After this instruction, RESLO, RESHI and the carry flag will look like the second part of Figure 5, and all the average calculations are complete!

At this point, all the program needs to do is return to BASIC, and it does so in Line 270 with an RTS instruction.

Once again, this routine is written to reside on page 6 of computer memory, and the BASIC code needed is shown in Figure 6.

```
10 FOR X=1536 TO 1567:READ N:POKE X,N:
NEXT X
20 ? "ENTER VALUE1, VALUE2";:TRAP 20:I
NPUT VAL1,VAL2
30 A=USR(1536,VAL1,VAL2)
40 PRINT A
50 GOTO 20
60 DATA 216,104,104,141,33,6,104,141,3
2,6,104,133,213,104,24,109,32,6,133,21
2,165,213,109,33,6
70 DATA 133,213,102,213,102,212,96
```

Figure 6.

(continued on next page)

Talk to ANALOG Computing

We're happy to announce that three members of our staff can now be regularly found on CompuServe. If you're a CompuServe member, you can contact Tom Hudson, Charles Bachand or Art Leyenberger by leaving a message on the Atari SIG, which can be accessed by typing GO PCS-132 at any menu page.

The Atari SIG has logged over 100,000 calls—with over 60,000 messages posted! They have a staff of highly competent SYSOPs, headed up by Ron Luks, who are more than happy to help you. Their program database contains well over a megabyte (that's one million bytes, folks!) of Atari programs that can be downloaded into your computer.

So, if you need to get in touch with **ANALOG Computing**, you can now do it through CompuServe. Our user numbers are:

Tom Hudson	70775,424
Charles Bachand	73765,646
Art Leyenberger	71266,46

Run the program and try some different input. The program will print the average just like Figure 2 did, but with far less memory needed to perform the operation. The moral: don't accept your first solution to any problem. Think it through and try to find the most efficient way to get the job done.

Square off.

Here's a routine that's a little different: an algorithm to calculate integer square roots! Figure 7 shows a flowchart of the algorithm, which is very simple.

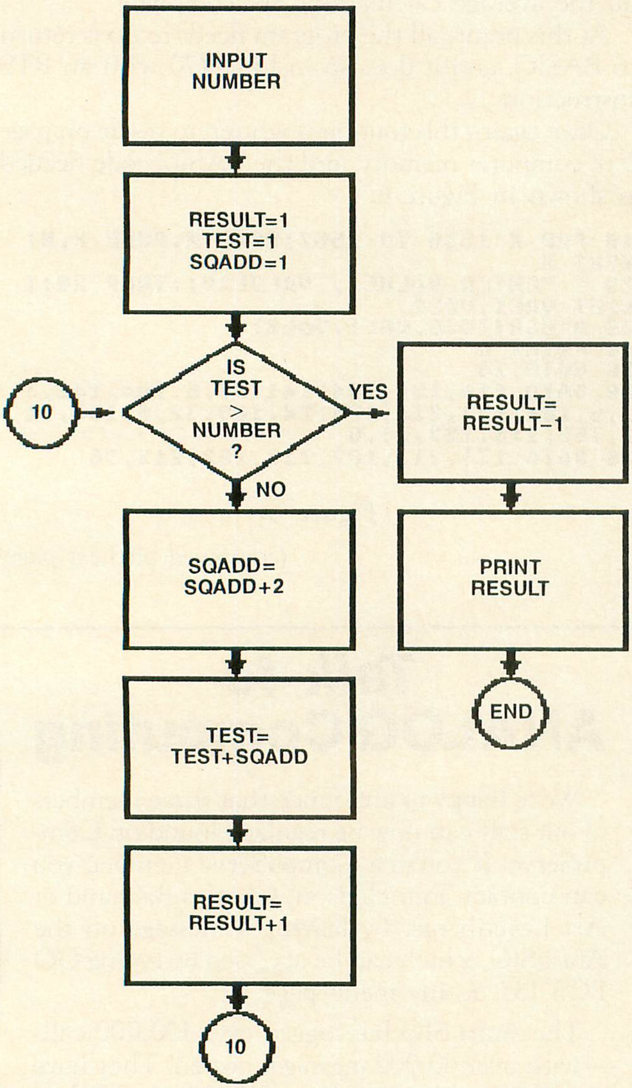


Figure 7.

Let's convert this flowchart into BASIC first, so we can get a "feel" for what's going on. Figure 8 shows the BASIC code corresponding to the flowchart.

```
10 INPUT NUMBER
20 RESULT=1:TEST=1:SQADD=1
30 IF TEST>NUMBER THEN RESULT=RESULT-1
:? RESULT:GOTO 10
40 SQADD=SQADD+2:TEST=TEST+SQADD:RESUL
T=RESULT+1:GOTO 30
```

Figure 8.

Type in this program and RUN it. When you are prompted for a value, type 9 and press RETURN. The computer will print out an answer of three, which is, of course, the square root of nine. Try typing *12* and pressing RETURN. Once again, the computer will give an answer of three. What happened? Well, this is an *integer* square root routine, and it will return only the integer portion of the answer. Since the actual square root of 12 is approximately 3.464, the integer result will be 3.

Now let's write the same routine in assembly language. The subroutine will only require one argument, the number we wish to find the square root of. Let's keep the routine simple and limit the arguments to a range of 0-255. If we do this, all the program variables can be one-byte areas, except for TEST. This is the variable compared to the argument to test for the end of the routine. If TEST is greater than the argument, the square root has been found. Since we are allowing arguments up to 255, TEST must be able to exceed 255 and, so, must be a two-byte value.

Figure 9 shows the assembly language version of the square root routine.

```
10 RESLO   =    $D4
20 RESHI   =    $D5
30         *=   $0600
0100       CLD         ;DECIMAL MODE!
0110       PLA         ;DISCARD # ARGS
0120       PLA         ;DISCARD ARG HI
0130       PLA         ;PULL ARG LO
0140       STA NUMBER  ;AND SAVE IT
0150       LDA #1      ;PUT 1 IN...
0160       STA RESLO   ;RESULT,
0170       STA TESTL   ;TEST VALUE,
0180       STA SQADD   ;SQRT ADD FIELD
0190       LDA #0      ;ZERO OUT...
0200       STA RESHI   ;RESULT HI BYTE
0210       STA TESTH   ;AND TEST HI
0220 SQLP  LDA TESTH   ;IF TESTH > 0,
0230       BNE END     ;WE'RE DONE!
0240       LDA NUMBER  ;GET NUMBER,
0250       CMP TESTL   ;< TEST?
0260       BCS NOSQ    ;NO, KEEP TRYING
0270 END   DEC RESLO   ;SUB 1 FROM ANS.
0280       RTS         ;AND EXIT!
0290 NOSQ  INC SQADD   ;SQADD =
0300       INC SQADD   ;SQADD + 2
0310       LDA TESTL   ;GET TEST VALUE,
0320       CLC         ;ADD THE
0330       ADC SQADD   ;ADD FACTOR
0340       STA TESTL   ;AND STORE IT
0350       LDA TESTH   ;NOW HI BYTE
0360       ADC #0
0370       STA TESTH
0380       INC RESLO   ;INC RESULT
0390       JMP SQLP    ;AND LOOP BACK!
0400 NUMBER *=*+1
0410 TESTL  *=*+1
0420 TESTH  *=*+1
0430 SQADD  *=*+1
0440       .END
```

Figure 9.

Line 100 clears the decimal mode.

Line 110 pulls the number of arguments from the stack. Since this routine will always be a single-argument one, this value is discarded.

Lines 120-140 pull the argument off the stack. Since we are only allowing values from 0-255, the high byte of the argument is discarded. The low byte is placed in the location labeled NUMBER.

Lines 150-210 initialize the subroutine variables. Note that Lines 190-210 set the high bytes of the result and the variable TEST to zero. Always be sure your multi-byte variables are properly initialized.

Lines 220-260 test for the end of the routine. In Lines 220-230, if the high byte of TEST is greater than zero, then we know TEST is greater than NUMBER (which is only a one-byte variable), and the program branches to END. If the high byte of TEST is zero, Lines 240-260 will compare NUMBER to the low byte of TEST. If NUMBER is greater than TESTL, the branch to NOSQ is taken and processing continues.

Lines 270-280 are used to exit the subroutine. Line 270 decrements the result by one, just like the BASIC program did in Line 30 of Figure 8. Line 280 executes an RTS instruction.

Lines 290-300 add two to SQADD by incrementing it twice.

Lines 310-370 add SQADD to TEST. Note that, since SQADD is a single-byte value and TEST is a two-byte value, a dummy value of zero is added to the high byte of TEST.

Line 380 adds one to the answer, stored in RESLO. This is done using the increment memory (INC) instruction.

Line 390 jumps back to SQLP to test the new square root value.

Figure 10 shows the BASIC code used to execute the machine language version of the subroutine. Try values from 0-255 to verify that the program works properly.

```
10 FOR X=1536 TO 1604:READ N:POKE X,N:
NEXT X
20 ? "ENTER VALUE";:TRAP 20:INPUT VALU
E
30 A=USR(1536,VALUE)
40 PRINT A
50 GOTO 20
60 DATA 216,104,104,104,141,69,6,169,1
,133,212,141,70,6,141,72,6,169,0,133,2
13,141,71,6,173
70 DATA 71,6,208,8,173,69,6,205,70,6,1
76,3,198,212,96,238,72,6,238,72,6,173,
70,6,24
80 DATA 109,72,6,141,70,6,173,71,6,105
,0,141,71,6,230,212,76,24,6
```

Figure 10.

All of the programs we've written so far are fixed-argument subroutines. That is, the square root routine never uses more than one argument, and the average routine never uses less than two. Now let's take a look at a short program which will take as many arguments as you send it!

Fun with arguments.

Let's say you want to add a list of numbers, using an assembly language subroutine. There's *no telling* how many numbers there will be, because it's different every time.

Fortunately, the BASIC USR call mechanism will tell our subroutine exactly how many arguments it needs to process. Up until now, we've ignored this value, which is always the first number pulled off the stack. Figure 11 shows a program which uses this information to add a variable number of values, returning the total to BASIC.

```
10 RESLO    =    $D4
20 RESHI    =    $D5
30          *=   $0600
0100        CLD         ;DECIMAL MODE!
0110        PLA         ;GET # ARGS
0120        STA ARGCT   ;AND SAVE IT
0130        LDA #0      ;ZERO OUT...
0140        STA RESLO   ;RESULT AREA
0150        STA RESHI
0160 ARGLP  PLA         ;GET ARG HI
0170        STA ARGHI   ;SAVE IT
0180        PLA         ;GET ARG LO
0190        CLC         ;AND ADD...
0200        ADC RESLO   ;TO TOTAL,
0210        STA RESLO   ;STORE NEW TOTAL
0220        LDA ARGHI   ;GET HI BYTE,
0230        ADC RESHI   ;ADD TO TOTAL,
0240        STA RESHI   ;SAVE NEW TOTAL
0250        DEC ARGCT   ;1 LESS ARGUMENT
0260        BNE ARGLP   ;BRANCH IF MORE
0270        RTS         ;ALL DONE!
0290 ARGHI  *=*+1
0300 ARGCT  *=*+1
0310        .END
```

Figure 11.

Let's look at the program and follow its logic.

Line 100 clears the decimal mode, as usual.

Lines 110-120 pull the number of arguments from the stack and store this value in the variable ARGCT.

Lines 130-150 zero out the total area, RESLO and RESHI. This ensures that the adding routine will start with zero.

Lines 160-170 pull the high byte of the argument from the stack and place it in the location labeled ARGHI.

Lines 180-210 pull the low byte of the argument from the stack, add it to the low byte of the total, and store the result back in RESLO.

Lines 220-240 add the high byte of the argument to the high byte of the total, storing the result back in RESHI.

Line 250 decrements the argument counter by one.

Line 260 will branch back to ARGLP if the result of the decrement was not zero (more arguments to process).

If there are no more arguments (ARGCT=0), the program exits with the RTS instruction at Line 270.

That's all there is to it! The BASIC code to use this USR call is shown in Figure 12. As it is, the routine will add the numbers from 1 to 10, returning the result 55. Try using different values. Just be sure you have at least one value to add and that the total will not exceed 65535.

```
10 FOR X=1536 TO 1569:READ N:POKE X,N:
NEXT X
20 A=USR(1536,1,2,3,4,5,6,7,8,9,10)
30 PRINT A
40 END
50 DATA 216,104,141,35,6,169,0,133,212
,133,213,104,141,34,6,104,24,101,212,1
33,212,173,34,6,101
60 DATA 213,133,213,206,35,6,208,234,9
6
```

Figure 12.

The use of multiple-argument USR calls is almost unrestricted. The main thing to remember, as far as system limitations go, is that you can't have more than about 27 arguments. Otherwise, BASIC will give you an ERROR-10 (argument stack overflow). Actually, 27 arguments is more than you'll probably ever try, but I just thought I'd warn you!

Until next time. . .

Between now and next issue's **Boot Camp**, try using and modifying these simple USR calls. Next time, we'll go even further into this interesting area of assembly language. ☐

CONTEST!

Here's a little contest that should keep all the code-crackers out there occupied.

The numbers below, when decoded, are a message in standard Atari ASCII. The numbers are in the proper sequence, and have been encrypted using a simple algorithm.

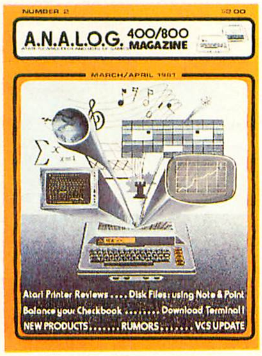
```
145 211 145 185 255 186 112 88
183 174 224 34 145 126 226 178
51 207 191 129 188 234 4 191
199 175 178 243 197 16 118 43
210 198 166 241 237 194 211 94
213 171 252 246 233 178 12 218
210 203 172 129 133 219 23 186
206 170 203 141 126 246 117 203
190 250 212 206 22 160 197 161
182 183 246 20 53 141
```

Decode the message, if you can, and send your solution to:

Code-Cracker Contest
c/o **ANALOG Computing**
P.O. Box 23
Worcester, MA 01603

The first five entries we receive with the correct solution will win a ***free one-year cassette or disk subscription.*** All entries must be postmarked before January 1, 1985.

BACK ISSUES

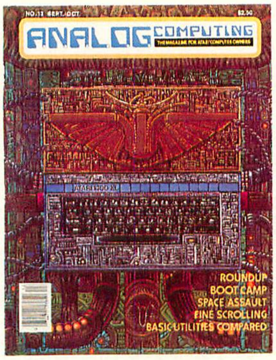


ISSUE 2 • Wasting Arrays • Atari's CPU • Download Terminal • Converting BASIC • Disk Files

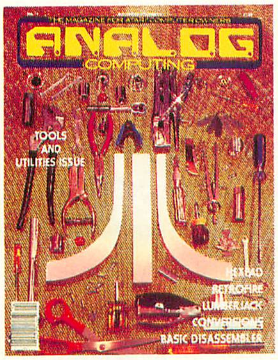
ISSUE 8 • GTIA Graphics • Audio in Your Programs • NOREM • Graphic Violence • Color Slot Machine

ISSUE 9 • Build Your Own 400 Keyboard • Harvey Wallbanger • Forth-DOS • Letter Writer

ISSUE 11 • Strings in BASIC • C:CHECK • Disk Cataloging Utility • Adventure in the Fifth Dimension • Moving Missiles in BASIC

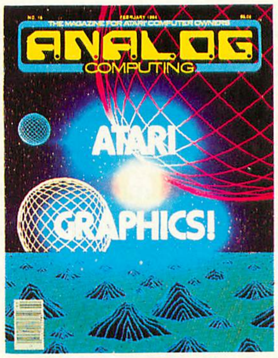


ISSUE 13 • Fine Scrolling Part 1 • Roundup • Space Assault • Observational Astronomy • CIO Routines



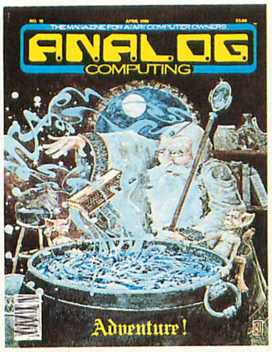
ISSUE 14 • Fine Scrolling Part 2 • Disassembler in BASIC • Hexpad • Lumberjack • Retrofire!

ISSUE 15 • Fine Scrolling Part 3 • Knights & Chalices • Music Synthesizer • Bricklayer's Nightmare • Keyboard Handler

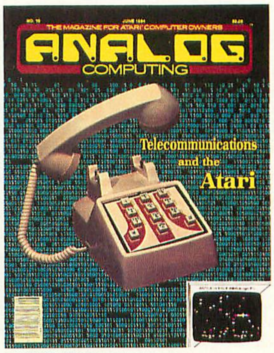


ISSUE 16 • Fine Scrolling Part 4 • Create-A-Font • Bar Chart Subroutine • Shooting Stars • 3-D Object Rotation

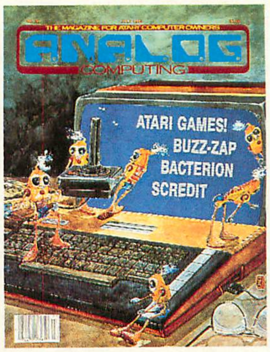
ISSUE 17 • Planetary Defense • Disk Miser • Live without DOS • Boot Camp • D:CHECK2



ISSUE 18 • Crash Dive • File'em • Munch'in Climb'in • H:BUG • Adventure Reviews



ISSUE 19 • Battle in the B-Ring • A Look at Modems • Bulletin Board Systems • Siege • Touch-Tone® Dialer



ISSUE 20 • Buzz-zap! • Scredit • AlterDOS • Bacterion! • ConTEXT

All back issues $4.00 each

Send check or money order to:
ANALOG Computing Back Issues
P.O. Box 615, Holmes, PA 19043

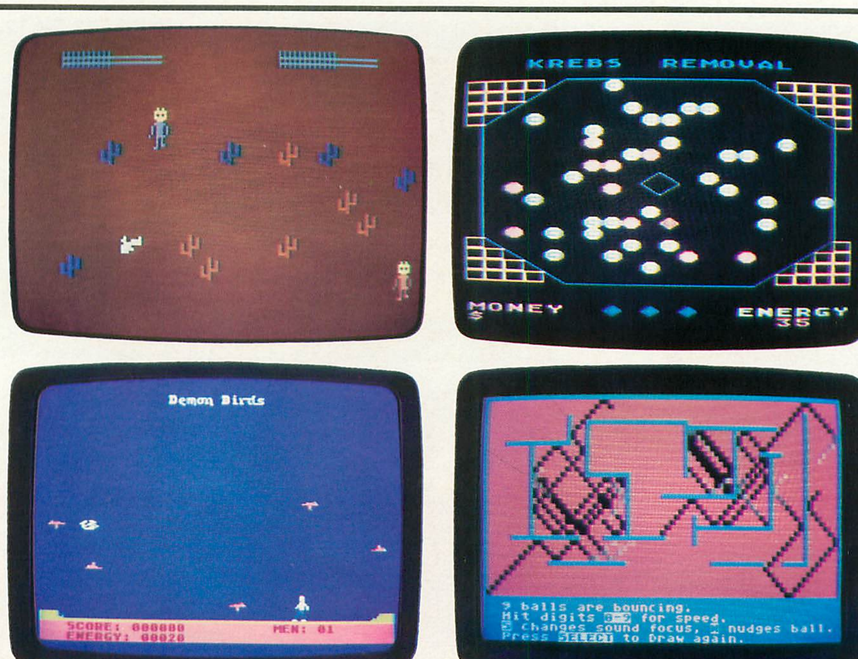
Issues 2, 21, 22 and 23 are also still available.
Order now—limited quantities available.

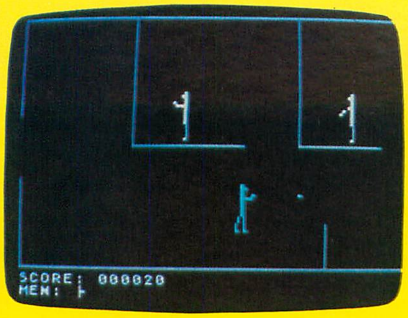
MasterCard and VISA orders call:
1-800-345-8112
in PA, 1-800-662-2444

INDEX TO ADVERTISERS

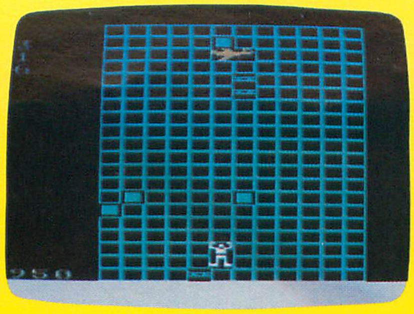
This index is an additional service. While every effort is made to provide a complete and accurate listing, the publisher cannot be responsible for inadvertent errors.

Coming soon in future issues of **ANALOG Computing!**

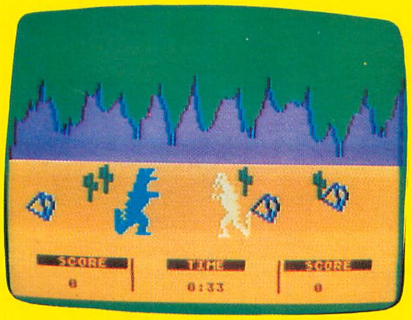




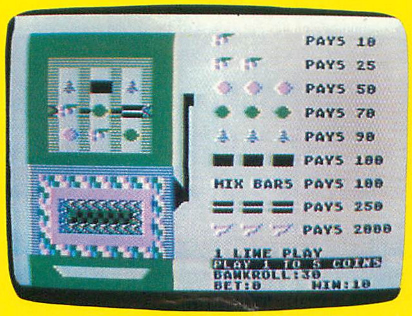
Maniac!


Stuntman

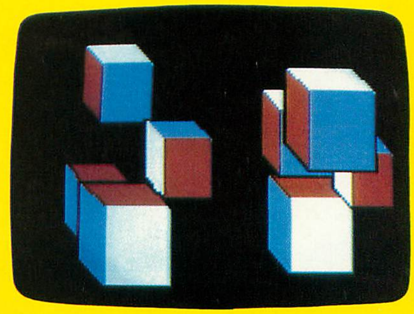
Fill 'Er Up



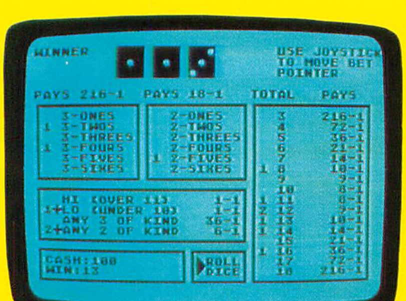
Dino Battle



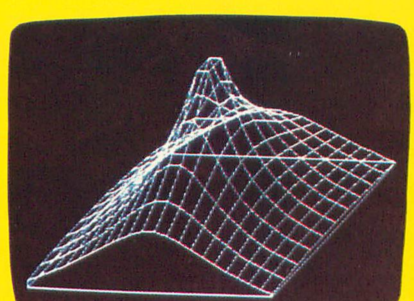
Color Slot Machine

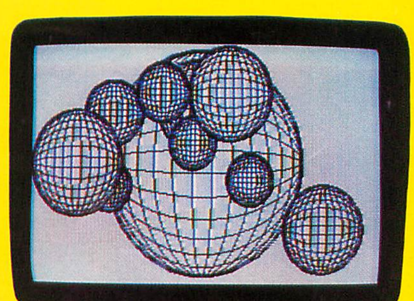

Cubes

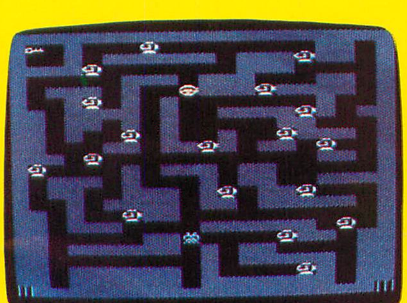
Where can you get all of these programs (and dozens more!) for only $14.95?



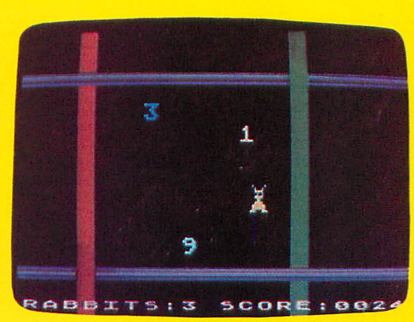
Triple Threat Dice


3-D Graphs


Sphere Demo


Leprechaun King

Sketch Pad



Harvey Wallbanger