

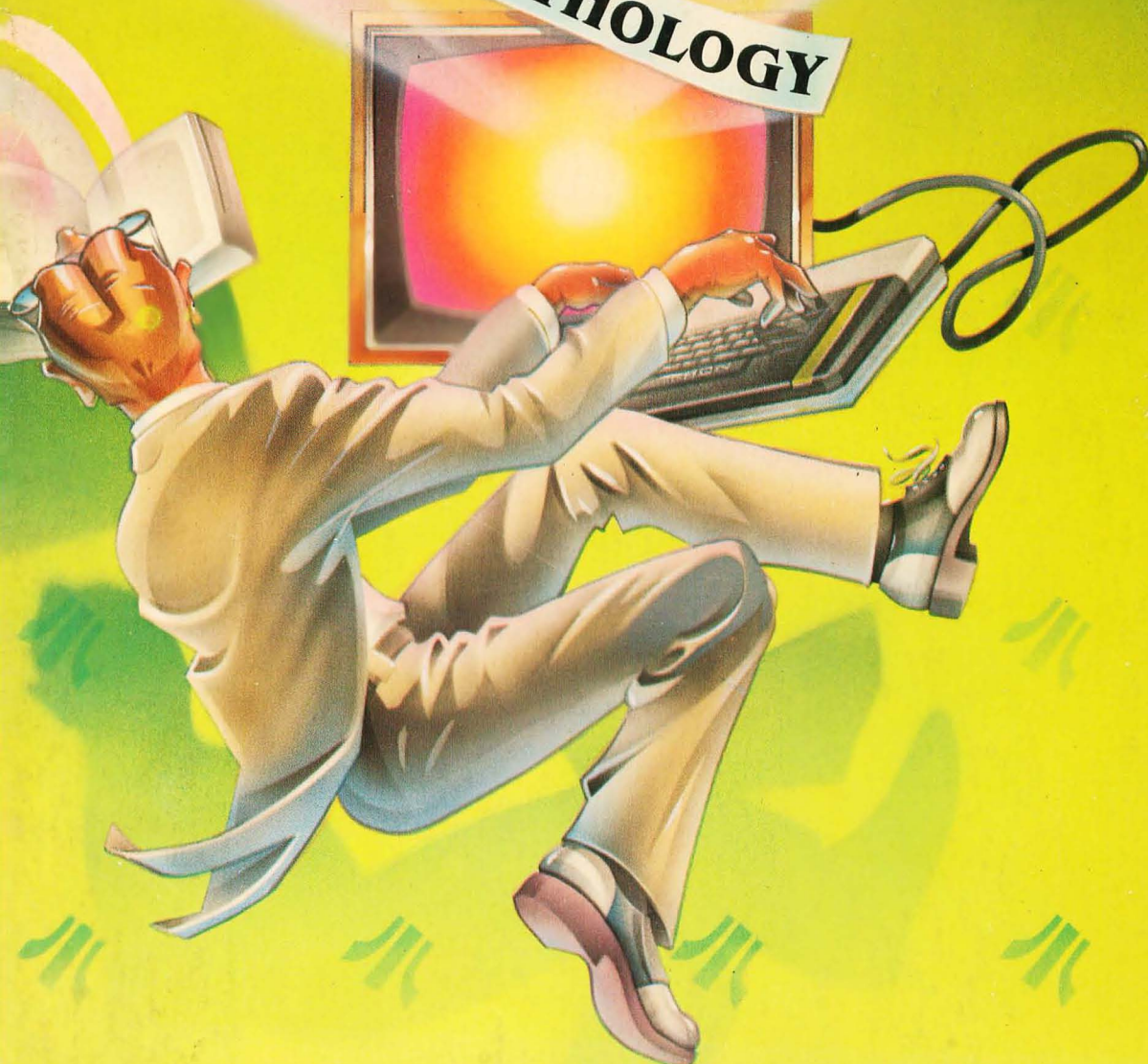
THE BEST OF

Antic™

VOLUME ONE

AN ANTHOLOGY

THE
ATARI
RESOURCE



PLUS SIX BONUS GAMES

Tired of Typing?

The BASIC programs in this book are available on cassette, tape and disk.

- ★ All ten games are yours for only \$20.00
- ★ Utilities and the other BASIC programs are just \$15.00

See order form on the other side

IF YOU OWN AN ATARI® ... YOU SHOULD BE READING **Antic**™

The ATARI® Resource Magazine

SAMPLE ISSUE (mailed 1st class) \$3.00

**Save Your Valuable Time . . . PLUS
Up to 33% off the Newstand Price**

SUBSCRIBE! Subscription order form on the other side

PLEASE SEND ME:

- ☐ 10 games from the Best of Antic
☐ Utilities and other BASIC programs

\$20 _____

\$15 _____

Shipping/Handling

\$1.50 _____

6½ % California Sales tax

Please Print

TOTAL

Name _____

or charge my VISA ☐ MasterCard ☐

Company Name _____

Acct. # _____

Address _____

Print name _____

City _____ State _____ Zip _____

Signature _____ Expires _____

I enclose payment of \$_____ by check or money order

Fold, then send this order to

THE BEST OF ANTIC

524 Second St.

San Francisco, CA 94107

YES! I want

Antic

The **ATARI** Resource

Please send me the following:

☐ 1 YEAR FOR \$24.00

SAVE 20%

CANADA & MEXICO

☐ 2 YEARS FOR \$44.00

SAVE 27%

1 year for \$30.00

☐ 3 YEARS FOR \$60.00

SAVE 33%

2 years for \$56.00

3 years for \$78.00

Please print. If information below is incomplete, magazine delivery cannot be guaranteed.

Name _____

or charge my VISA ☐ MasterCard ☐

Company Name _____

Acct. # _____

Address _____

Print name _____

City _____ State _____ Zip _____

Signature _____ Expires _____

I enclose payment of \$_____ by check or money order,

I own an ATARI A ☐ 800, B ☐ 400, C ☐ 2600, D ☐ 5200, P ☐ 1200XL, Q ☐ 600XL,
R ☐ 800XL, S ☐ 1400XL, T ☐ 1450XL, with cassette drive ☐, disk drive ☐,
printer ☐, modem ☐, interface ☐, memory

**To subscribe by phone, call toll free
FOR CREDIT CARD ONLY**

(800) 227-1617 Ext. 133 (outside California)

(800) 772-3545 Ext. 133 (inside California)

Please allow eight weeks for magazine delivery

Antic

524 Second St.

San Francisco, CA 94107

THE BEST OF



VOLUME ONE

AN ANTHOLOGY

Antic Publishing, Inc. 1983

This anthology was compiled by James Capparell,
edited by Robert DeWitt,
and designed and produced by Kim Gale,
with special thanks to the ANTIC Magazine staff.

Cover Illustration: Bud Thon
Inside Illustrations: John Musgrove

Copyright © 1983 by ANTIC Publishing Inc.
All rights reserved. No part of this book may be
reproduced in any form except for
individual entertainment or usage.

ANTIC Publishing Inc.
524 Second St.
San Francisco, CA 94107

ISBN: 0-914375-00-8

Printed in the USA

TABLE OF CONTENTS

What's in a name?	vi
Introduction <i>Robert DeWitt</i>	1
Listing Conventions	4
STARTING LINE	6
Help for the New User <i>James Capparell</i>	7
Screen Editing <i>Robert DeWitt</i>	10
Oh, Those Bugs <i>David Plotkin</i>	13
A Sound Introduction <i>James Capparell</i>	16
TYPO <i>Bill Wilkinson</i>	20
EDUCATION	24
Spin Colors With Spider <i>John & Mary Harrison</i>	25
Zahrcon <i>Linda Schreiber</i>	29
Tuning Your Atari <i>Linda Schreiber</i>	37
Candle, Candle, Burning Bright <i>Linda Schreiber</i>	42
SOUND AND MUSIC	48
Some Sound Advice <i>David Plotkin</i>	49
Audio While You CLOAD <i>John Victor</i>	51
Music With BASIC <i>Jerry White</i>	55
Ultra Sound <i>Thomas Krischan</i>	60
'Tari Talkers <i>Ken Harms</i>	63
COMMUNICATIONS	68
Modems <i>Jon Loveless</i>	69
Communications Software <i>Jon Loveless</i>	75
Dialing for Data <i>Robert DeWitt</i>	80
PRONTO <i>Deborah Burns</i>	87
GAMES	90
Chicken <i>Stan Ockers</i>	91
Attack on the Death Star <i>David Plotkin</i>	101
Speed Demon <i>John Magdziarz</i>	111
Bats <i>Stan Ockers</i>	117

TABLE OF CONTENTS

BONUS GAMES	124
Tie Fighter <i>Jimmy and Tommy Sa</i>	124
Tin Pan Alley Cats <i>Rick Bloom and Rob Glassman</i>	131
Drop <i>John Zakour</i>	141
Fallout <i>Scott McKissock</i>	145
Skull Chase <i>Dave Miller</i>	150
Crystal Caves <i>Thomas Edwards</i>	155
FEATURES	160
Translate <i>Jerry White</i>	161
Display Lists Simplified <i>Allan Moose and Marian Lorenz</i>	165
Tiny Text <i>Jim Carr</i>	173
Christmas Mailing Lister <i>Bill Lukerorth</i>	181
SYSTEMS GUIDE	196
Memory Map <i>James Capparell</i>	197
Scrolling <i>James Capparell</i>	212
ANTIC Disassembler <i>James Capparell</i>	217
Interrupts <i>James Capparell</i>	224
ASSEMBLY LANGUAGE	230
Move It <i>Jerry White</i>	231
Bubble Sort <i>Adrian Dery</i>	235
PILOT YOUR ATARI	244
Pilot Your Atari <i>Ken Harms</i>	245
Large Text <i>Ken Harms</i>	247
Colors for Your Pilot <i>Ken Harms</i>	250
The Musical Pilot <i>Ken Harms</i>	254
Holiday Trees <i>Ken Harms</i>	263
FORTH FACTORY	272
Turtle Graphics <i>Gordon Smith</i>	273

What's In A Name


Antic

Many new readers are curious about our name, ANTIC. They wonder what it has to do with computing, and especially with the ATARI. ANTIC is the name of one of the LSI chips designed by ATARI exclusively for its computers. ANTIC is an acronym formed from the words "Alpha-Numeric Television Interface Circuit." This chip controls the video display you see on your TV screen or monitor. ANTIC is a true microprocessor and has its own instruction set. By handling the screen display it relieves the Central Processing Unit—the 6502 chip—of about one-third of the load it would otherwise carry.

There are three other chips in the ATARI you often hear about. The GTIA (CTIA in machines manufactured before January, 1981) is the General Television Interface Adaptor; it enables Player/Missile graphics and Graphics Modes 0 through 11. The POKEY chip means "pots and keys." It monitors input from the keyboard and controls audio functions. The PIA chip is the controller for joystick ports and peripherals.

Atari

Atari, the name of the company that makes our favorite computer, was chosen by the founder of that company, Nolan Bushnell. It is now the second best-recognized product name in the world (next to "Coke"). Atari is a Japanese word, taken from the ancient game of GO. "Atari" has the approximate meaning of "check" in chess—the player who declares "atari" is reminding the opponent that territorial loss is imminent if an effective countermove is not made immediately.

Another Japanese word is sometimes heard in Atari circles. The three-legged Atari logo  is called the "fuji," and the key on the keyboard that bears the symbol is called the "fuji key." This is because the symbol looks like Mount Fuji, but the design is not a letter in the Japanese alphabet, nor is there a letter or ideogram called fuji. On the ATARI 400 and 800 computers, the fuji key switches the video display from regular to inverse.

Introduction

by Robert DeWitt
Managing Editor, ANTIC Publishing Inc.

This book is for people who own or have access to ATARI computers. It excerpts the best material from the first six issues of ANTIC magazine, and adds some extra articles, games and other programs ATARI owners may need and enjoy. We offer it primarily because our supply of back issues for Volume One of ANTIC is nearly exhausted. *ANTIC—The ATARI Resource* has become the largest monthly magazine exclusively serving the Atari community, and new readers constantly inquire about our early material. We hope this fills the bill.

If you, like us, own an ATARI, your computer is probably the first one you have ever owned. Many of us have used a computer in our office or classroom, and we may have taken courses in computing. Some of us may have studied programming—perhaps BASIC or COBOL. A few of us may have tinkered with electronics or even studied computer science, but most of us are rather new at all this.

Whatever our situations, computing sooner or later presents us with new terms, concepts, and ways of approaching and solving problems that baffle us. We struggle to understand, and gradually (or suddenly) the new ideas come clear. One of the qualities of computing is that it is supremely logical. If your computer or program doesn't work, some specific thing (or things) is wrong. Computers are built around the notion of error-free operation precisely to make it easier to find and fix problems. This frustrates new users because mistakes are common when you are learning. You don't yet have the experience of successful, pleasant use of the computer to encourage you, and you haven't yet absorbed the many pieces of information that will eventually lead you to quick solutions. You will probably conclude occasionally that your computer is broken (it seldom is) or that there is a program logic error. The solution is often simpler.

Our writers and editors have gone through your agonies. At various times we have connected our equipment incorrectly, scrambled procedures, misinterpreted instructions, overlooked the obvious, "lost" data and pro-

INTRODUCTION

grams, ruined diskettes, and made errors in programming. We have typed in a listing for hours only to find that the program doesn't work. Occasionally we have thrown up our hands in disgust. We know what you are going through.

We have also returned with a calmer mind, sought help, reread the instructions, persisted, and eventually enjoyed the personal satisfaction and some of the accomplishments that home computing brings.

Our primary purpose at ANTIC is to help you enjoy ATARI computing too. ANTIC magazine began as a labor of love, and although it has grown into a successful publishing business, it is still based on our personal interest in ATARI computers. Our first issue was dated April, 1982, and appeared just in time for that year's West Coast Computer Faire in San Francisco. That issue contained 40 pages and presented eight articles and a few other items. We printed about 12,000 copies (all we could afford) and stored them in our publisher's apartment. We sold 400 copies at the Faire, and hawked a few thousand more to computer stores. We offered the rest as back issues while they lasted, but the issue was sold out before the year was over.

ANTIC was published bimonthly for the first year, so there are six issues in Volume One. Issues number one through five are also now sold out. The last issue in the volume, February-March 1983, had 112 pages, 17 articles, and various other content. By then our circulation was about 60,000. With that issue we began monthly publication, and as of this writing have passed the milestone of 100,000 copies sold per issue.

Many ANTIC readers have requested back issues "to complete their set" or to get some specific article. It is gratifying to know our early efforts are in demand. We have gone back over Volume One and extracted the material we considered of greatest interest and continuing value. We have added some new material, useful especially to those who want to program. We have also added several games previously unpublished.

In spite of its growth in size and quality, in many ways ANTIC number six still resembles number one. Every issue featured at least one type-in game, placed conveniently at the centerfold. The first was *Chicken*, by Stan Ockers, a fine game then and now. In this book we repeat *Chicken* and several other games from ANTIC Volume One.

We also reprint *TYPO*, our checksum program by Bill Wilkinson, the buddah of BASIC. *TYPO*, which means Type Your Program Once, gives you a way to locate your entry errors in the BASIC listings that appear in ANTIC (and this book), and to verify your listings when correct. We still use *TYPO* in

INTRODUCTION

every issue of ANTIC and it alone will repay the price of this book in the time it will save you.

The Memory Map is another valuable resource. When you turn your ATARI on, the Operating System establishes values in memory that guide and direct hundreds of functions for your computer. The map tells you where these are, what they do, and how they work. This information has been gained by digging it out of the technical documentation for the ATARI 400 and 800 computers. Although it is not comprehensive, we believe it will be helpful. You should note it may not be valid in all respects for the new XL line,

Most of the programs we reprint will be in BASIC, a few in assembly language. The programs have been chosen for amusement and usefulness. We have reviewed them carefully to make sure they work. Each program has been RUN on our machines (400, 800, 600XL, and 1200XL). We have tried to keep RAM requirements within 16K. If more, we note it. Our computers were used to generate the listings, so they should work as published. Technical problems that appeared in the magazine version have been corrected here. Any new problems should be reported in writing, attention Technical Assistant, to the address below. If you want a personal response, include a self-addressed stamped envelope. We know from experience that most problems are due to entry errors, so use TYPO and review your work carefully before writing.


The listings in this book have been produced to show you exactly the same ATARI characters you will see on your video screen in the same place you will see them. In other words, the listing emulates the screen. This will help you type correctly and proof your work. The keystrokes needed to produce some of these characters may not be known to you. The Table of Listing Conventions that follows provides a chart of these obscure characters and tells you how to enter them.

If you would prefer not to type in these programs, you can obtain them on disk or cassette by using the form in the front of the book, or by sending your check, money order or credit card authorization to: ANTIC Anthology, 524 Second St. San Francisco, CA 94107

Listing Conventions

Table Information

Our custom font listings represent each ATASCII character as it appears on the video screen. You generate some characters by a single keystroke, for example, the regular alphabet. Others require a combination or sequence of keystrokes. In this table, ESC means *press and release* the escape key before pressing another key. CTRL or SHIFT means *press and hold* the control or shift key while simultaneously pressing the following key.

The Atari logo key () “toggles” inverse video alphanumeric and punctuation characters. Press the key once to turn it on; press again to turn it off. On the 1200XL there is no logo key; inverse video is controlled by a key on the function row. Decimal values are given for reference, and correspond to the CHR\$ values often used in BASIC listings.

LISTING CONVENTIONS

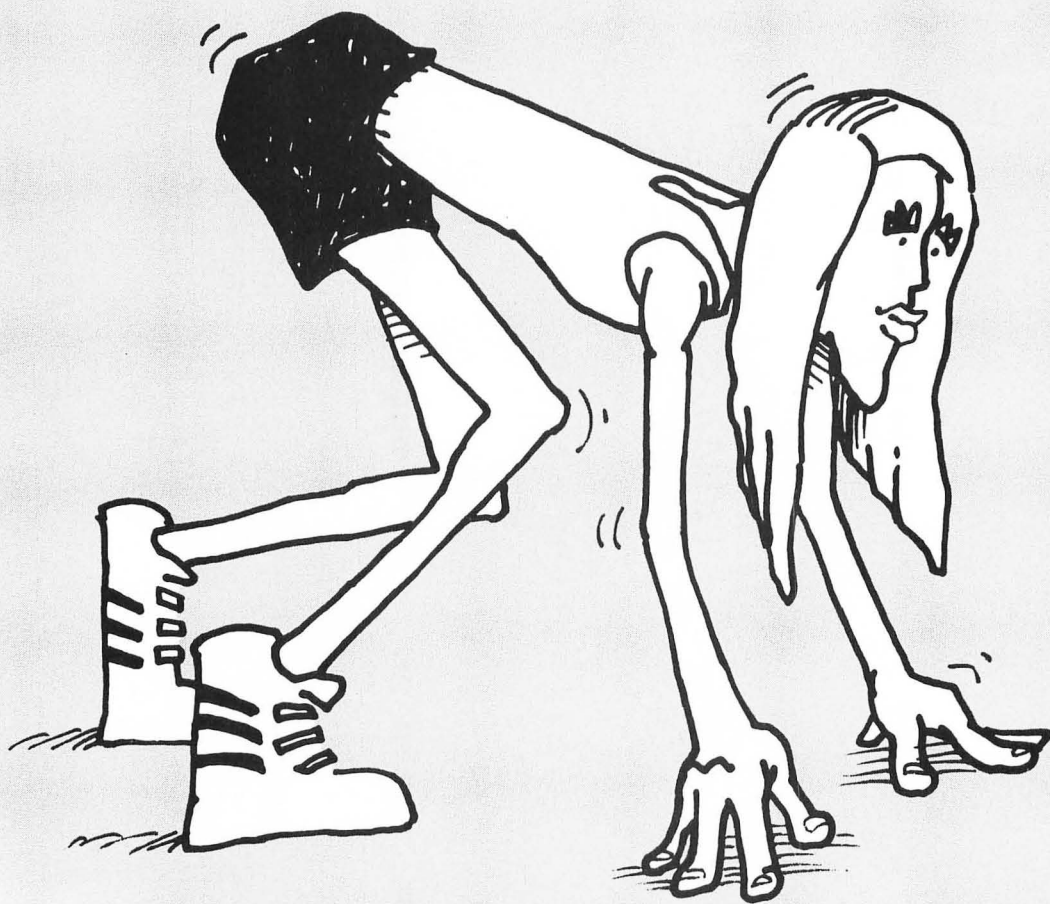
NORMAL VIDEO

FOR THIS	TYPE THIS	DECIMAL VALUE
	CTRL ,	0
	CTRL A	1
	CTRL B	2
	CTRL C	3
	CTRL D	4
	CTRL E	5
	CTRL F	6
	CTRL G	7
	CTRL H	8
	CTRL I	9
	CTRL J	10
	CTRL K	11
	CTRL L	12
	CTRL M	13
	CTRL N	14
	CTRL O	15
	CTRL P	16
	CTRL Q	17
	CTRL R	18
	CTRL S	19
	CTRL T	20
	CTRL U	21
	CTRL V	22
	CTRL W	23
	CTRL X	24
	CTRL Y	25
	CTRL Z	26
	ESC ESC	27
	ESC CTRL -	28
	ESC CTRL =	29
	ESC CTRL +	30
	ESC CTRL *	31
	CTRL .	96
	CTRL ;	123
	SHIFT =	124
	ESC	
	SHIFT	
	CLEAR	125
	ESC DELETE	126
	ESC TAB	127

INVERSE VIDEO

FOR THIS	TYPE THIS	DECIMAL VALUE
	^ CTRL ,	128
	^ CTRL A	129
	^ CTRL B	130
	^ CTRL C	131
	^ CTRL D	132
	^ CTRL E	133
	^ CTRL F	134
	^ CTRL G	135
	^ CTRL H	136
	^ CTRL I	137
	^ CTRL J	138
	^ CTRL K	139
	^ CTRL L	140
	^ CTRL M	141
	^ CTRL N	142
	^ CTRL O	143
	^ CTRL P	144
	^ CTRL Q	145
	^ CTRL R	146
	^ CTRL S	147
	^ CTRL T	148
	^ CTRL U	149
	^ CTRL V	150
	^ CTRL W	151
	^ CTRL X	152
	^ CTRL Y	153
	^ CTRL Z	154
	ESC	
	SHIFT	
	DELETE	156
	ESC	
	SHIFT	
	INSERT	157
	ESC	
	CTRL	
	TAB	158
	ESC	
	SHIFT	
	TAB	159
	^ CTRL .	224
	^ CTRL ;	251
	^ SHIFT =	252
	ESC CTRL 2	253
	ESC	
	CTRL	
	DELETE	254
	ESC	
	CTRL	
	INSERT	255

Starting Line



Help for the New User

As a regular feature in ANTIC we try to provide useful, jargon-free information for the new user. As time goes by there will be many more of you opening a silver box to the world of ATARI. We would like to relieve you of any unnecessary anxiety and help you verify that your equipment is operating correctly.

Can I hurt my machine?

No, there is nothing that you can do from your keyboard in normal operation that will harm your ATARI. Feel free to press any key in any order. Experiment, try it, learn by example and by trial and error. One note of warning, always save a copy of your program on cassette or disk prior to experimenting. This way you'll have a copy to reload should the experiment fail, or someone kicks the plug out of the socket. Keep food and drink away from your equipment, and disks and tapes away from magnetic fields and dust or ash.

What does 32K mean?

In computing circles, terms like 16K or 32K are frequently heard. Numbers with the suffix K are used to refer to the amount of memory available in your machine. K is a metric abbreviation and refers to the number 1000. The computer world adopted K to mean 1024, the value of two to the tenth power. So, 32K RAM would mean that 32,768 bytes of memory are available. Most ATARIs come factory equipped with 16K RAM. Both the ATARI 400 and 800 can be expanded to 48K RAM, or 49,152 characters of information. Consult your local dealer about memory expansion products.

How can I be sure that all my memory is there?

To verify that your installed memory is being recognized, type the command PRINT FRE(0), and press [RETURN].

James Capparell is the publisher and founder of ANTIC – The ATARI Resource.

STARTING LINE

With the BASIC cartridge installed you should read:

13326 (if 16K)

29710 (if 32K)

37902 (if 48K)

How many characters will fit on the screen?

A maximum of 40 text characters per line, by 24 lines, can be displayed on your TV screen by the ATARI. In normal operation only 38 characters are allowed because of the two-character margin. This can be changed by typing the command POKE 82,0 (press [RETURN]). This effectively moves the left margin two characters left, giving you the maximum of 40 characters. Press [SYSTEM RESET] to restore margins.

How long a line will BASIC accept?

BASIC can receive up to 120 characters per command line (three full 40-character lines). This is also called a "logical line." The normal margin reduces this to 114 characters. A warning buzzer sounds seven characters from the end. If you type more than the maximum, the excess characters are ignored.

Is there an easier or faster method of entering BASIC?

Yes, use abbreviations wherever possible (look at Appendix A of your BASIC Reference Manual). Using abbreviations will save typing time. For example use N. instead of NEXT or C. instead of COLOR. The BASIC cartridge will expand these abbreviations for you automatically. It is also legal to eliminate spaces wherever possible, once again BASIC will insert spaces for you. For example, 110REM is okay.

What color should my screen be when I turn on my ATARI?

Your screen should be blue when first turned on. This is one of 128 color possibilities available. There are 16 colors and eight hues on every 400/800. Look at page 50 of the BASIC Reference Manual for the color-range description.

What is Memo Pad Mode?

The ATARI will respond with this statement whenever you turn power on without either a language cartridge installed or the disk-system DOS installed. You can display characters on the screen with Memo Pad Mode, but that's about all.

What does screen editing mean?

This refers to the ability to insert and delete characters on the screen by moving the cursor around and by using several other keys. Press [CTRL] (the control key) and the up/down or left/right arrows simultaneously to see this effect. Errors can be corrected and lines inserted without the necessity of retyping entire lines. Look at Chapter Three of the BASIC Reference Manual for more edit features, and see the Screen Editing article in this book for an exercise.

Why does my screen change colors when I leave it for a while?

This is called attract mode. If there has been no input from your keyboard in the previous nine minutes, the colors begin to change on your television. This occurs to protect the color phosphors of your picture tube. Just press any key and the colors will return to normal for at least nine minutes more.

How can I be sure my equipment is operating properly?

Modern electronic equipment is extremely reliable. In almost all cases your computer either will fail in the first 50 hours of use, or continue operating for the next five years. Whenever you power-on your computer with the BASIC cartridge installed, the friendly message READY should appear in the upper-left corner of the screen. Almost always computer failure will be total. It will either run properly or it won't run at all. If you should develop trouble, read the instructions and recheck your power and connectors. Begin to eliminate probable causes one by one. Be methodical! If your cassette doesn't work, try it on your friend's computer. Try to isolate the pro-

STARTING LINE

blem. You can save yourself unneeded trips to the repair center by thinking through the problem, trying and retrying. These techniques work for professionals and they'll work for you.

by James Capparell

Screen Editing

As an ATARI owner, you will benefit from having its built-in "screen editor," one of the best available in the micro market. What's a screen editor? It's the built-in program that allows you to change words and letters after they have been keyed onto your display screen.

As you begin to program with your ATARI, you will come to appreciate this powerful tool. At first, though, it may seem strange to you, and you will make mistakes until you learn how it works.

The most important thing to do, in this or any other computer function, is to read the instructions. These are in your Operators Manual, and in the BASIC Reference Manual, under "Screen Editing" and "Editing." Read these, do the exercises, and experiment. Be bold. You cannot damage your computer by making keyboard errors.

Ground Zero

Connect your computer as instructed, insert the BASIC cartridge, and power-up. On a color television you will see a blue screen with black borders, the word READY, and the white cursor beneath the "R". Remember, this is not an exercise in BASIC, but in screen editing. The BASIC program used is just an example.

This blue screen is BASIC Graphics Mode 0, designed to display text. This mode divides the screen into 40 character positions across the screen and

Robert DeWitt is a journalist from San Francisco who began writing about computers in 1980 based on studies at Control Data Institute. He bought an ATARI as an apparent "best-buy" tool for his writing, and gravitated to the fledgling ANTIC through the local user group. He gradually assumed more of the editorial functions of the magazine, becoming Managing Editor in November, 1982.

24 lines down, i.e., a 40x24 grid yielding 960 character positions. Each position on the screen is the size of the cursor, and can be identified by its column and row number, beginning with 0,0 in the upper left corner and ending with 40,24 at the lower right. The first number, 40, indicates the column and the second number, 24, is the line number.

The content of each of the 960 positions is controlled by the Editor program, built into the ATARI Operating System. It takes one byte of memory to code the contents of each position. For the ATARI computers, this code is called the ATASCII code. You will find it in Appendix C of your *Atari BASIC Reference Manual*.

The important thing to know is that you can determine and change the content of these screen positions by using your keyboard. Editing deals mostly with changing and erasing the display.

You should now be running Atari BASIC and have the READY prompt on the screen. Type in the following program, beginning at "10 REM. . ." and be sure to include the misspelling of "capabilities." [RET] means press Return key. Begin!

```
10 REM * SCREEN EDITOR EXAMPLE * [RET]
20 PRINT "THIS IS AN EXAMPLE OF ATARI
   SCREEN EDITING CAPABILTIES" [RET]
RUN
```

Notice that as you typed line 20, the line "broke" between ATARI and SCREEN. This is an example of the "logical line" continuing over two "physical" lines. This phenomenon is called "wraparound." A logical line can be as long as three physical screen lines. The computer will "beep" when you are close to the logical line limit.

After the run, you should see on the screen:

```
THIS IS AN EXAMPLE OF SCREEN EDI
TING CAPABILTIES
READY
```

Now we will edit this material. Generally speaking, we edit by moving the cursor to the character position we wish to change and then changing the character. The cursor rests at the left margin below the R in READY. Find the Delete Back Space key (upper right corner), which we will represent as [DEL], and press it. The cursor does not move.

How can we move the cursor? Find the [CTRL] key. Press it down and

STARTING LINE

hold it there. Find the “up” arrow key and press it three times. Release the [CTRL]. The cursor will move up three lines and be superimposed over the T in TING. Notice that the T appears dark blue within the field of the cursor. This condition is called “inverse video.”

Press the space bar four times. The cursor moves across the letters of TING, erasing them as it passes.

To the right of the cursor is the word CAPABILTIES. Next, correct the spelling. Press and hold [CTRL], and press the right-pointing arrow key until the cursor is superimposed on the T. We want to insert the letter I. Holding the [CTRL], press the Insert key (top row, third from right).

Voila! A space opens between the L and the T. Release [CTRL] and type in the letter I. The cursor now rests over the letter T. To exit from the word without changing it further, press and hold [CTRL], and press the left-pointing arrow until you have backed out of the word. Release [CTRL].

Up and Over

Here is a surprise for you. Press [DEL] six times. This will be enough to make the cursor back up to the line above. This is due to wraparound. It would not be possible between logical lines without using the [CTRL].

We can now repair the damage done to the word EDITING by typing it again. When the cursor again rests between EDITING and CAPABILITIES, press and hold the [CTRL], press the down-arrow key three times, release [CTRL] and finally, press [RETURN].

Let's see if we have corrected the misspelling. Type LIST and press [RET]. This command rewrites the corrected program. You should have lines 10 and 20 come up, and the error is *still there*! That's because the correction was made to the “run,” and not the program. This time we will fix it for good. Press and hold [CTRL]. Press the up arrow three times till the cursor is over the S in SCREEN. Press the right-pointing arrow key till the cursor is over the T in CAPABILITIES. Still holding [CTRL], press the Insert key. Pop! Release [CTRL], then [RET]. Type RUN and press [RET]. A new line should appear on the screen. Read your correction. WHAT! The error is still there?

That's right. This is the trickiest part of screen editing in BASIC. Remember, changes to the screen do not necessarily change the stored program. Changes within lines (deferred mode) are made permanent by pressing the return key [RET] before you leave the logical numbered line on which the change was made.

Let's do it right this time. Hold [CTRL]. Move the cursor up until it is over the S in SCREEN. Hold the [CTRL] and press the right-arrow until the cursor is over our "I" (yes, it's still there is screen memory, but not in program memory). Release [CTRL]. This time, press [RET]. The cursor jumps down to the beginning of the next line, above the READY. Type RUN and [RET].

Aha! This time the change has been made in the program. Failure to use the screen editor correctly can cause you no end of grief. The main thing to remember is that all corrections to program (numbered lines) must be confirmed by pressing [RETURN] before moving the cursor from the corrected line.

by Robert DeWitt

Oh, Those Bugs

After the publication of "*Chicken*" and "*Attack on the Death Star*" in ANTIC we received many calls from puzzled readers who were unable to make the programs run. Since both listings were correct, we know many of you need help finding errors. This article will give some elementary guidance in debugging BASIC programs. Also see and use TYPO, in this book, to help you find entry errors.

The most important advice we can give is: *never* attempt to RUN a program prior to saving a copy on disk or tape. Should your newly typed program contain a fatal error it may possibly cause the computer to fail to respond to the keyboard. This forces you to turn the power off and then on again in order to reset, erasing computer memory and your program.

We assume you've corrected your normal typing errors, those which generate an error message when you press [RETURN] after a line. The remaining errors are more subtle and are not reported until the computer tries to execute the program.

Such things as NEXT with no FOR, or a RETURN with no GOSUB, are generally the result of a missing line. Tracing back through the program to the line where the command should be is fairly straightforward. More difficult are the errors which are not actually in the line the computer indicates. You have

STARTING LINE

to know where else to look for the error. Most notorious of these are errors which result from mistyping a DATA statement.

Most errors in typed computer programs stem from DATA statements. There are logical reasons for this. Human beings are not very good at duplicating long strings of numbers or letters separated by commas. Numbers get transposed or dropped, commas get left out, or periods are substituted for them. Secondly, the computer does not check DATA statements during input. You can put *anything* in a DATA statement and the computer won't protest — until you try to RUN the program.

A surprising variety of errors can be traced to DATA statements. There is, of course, "Out of Data" (Error 6), but often the following errors are due to DATA errors:

1. Value Error (outside a specified range; Error 3)
2. String length error (Error 5)
3. Number greater than 32767 (Error 7)
4. Input statement error (Error 8)
5. Cursor out of range (Error 141)

It is true that these error messages can also be caused by other mistakes besides DATA statements. Out of Data and Cursor Out of Range can be caused by an error in the parameters of a FOR-NEXT loop. Often there will be a series of FOR-NEXT loops in a program, and an error in typing the parameters of one FOR-NEXT loop won't be detected until one of the following loops is executed. A String Length error may be the result of mistyping the DIM statement. In general, this is not detected until you try to define or use a portion of the string past the dimensioned length.

Knowing that such a wide range of errors can indicate a mistyped DATA statement is half the battle. Finding out which DATA statement can be difficult because the computer reports the error as occurring in the line containing the READ statement. Thus, if line 10 says (in part), "READ X, Y: PLOT X, Y" followed by lines of DATA statements, any data error will be reported as occurring in line 10, even though line 10 is typed in perfectly! The problem of finding the erroneous DATA statement is somewhat simplified if each READ statement is followed by its DATA statements.

There are other ways to isolate the problems. One way is to check the line where the error is reported and ask the computer to print the value of the READ variable. Often the READ statement is executed in a FOR-NEXT loop, and you can ask the computer to print the value of the loop variable. For example, let's look at the following BASIC program:

```
10 DIM A$(10)
20 FOR N=1 to 10: READ Q: A$(N,N)=Q: NEXT N
30 DATA 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
40 FOR M=1 TO 5: READ X, Y, Z: PLOT X, Y:
   POKE 256+M, Z: NEXT M
50 DATA 10, 20, 24, 30, 40, 24, 50, 60, 24, 70, 80, 24,
   90, 100, 25
```

Suppose you accidentally typed A\$(5) instead of A\$(10) in line 10. You'll get the error message "ERROR 5 on Line 20". This is an example of a "misleading" error message. Next, suppose you made "8, 9" into "8.9" in line 30, by substituting a period for a comma. You'll get the message "ERROR 6 on line 40". Line 40? What's going on here? When line 20 is trying to READ Q 10 times (N=1 to 10), it runs out of data on line 30 because 8.9 is one data item, while 8, 9 is two. The computer gets the first data item on line 50. When line 40 executes, it starts reading at the second data item on line 50 and, consequently, runs out of data.

The first thing to do is type (in direct mode) PRINT M. The computer responds with 5. You can tell that line 40 did not finish executing, because if it had, the computer would respond with 6 — one *more* than the loop limits. Counting off the data items in line 50 reveals that the last values of X, Y, and Z should be 90, 100, and 25 respectively. Typing in PRINT X, Y, Z causes the computer to respond with 100, 25, and 90. Everything is off by one data item, but line 50 is typed in perfectly. Now go back to the READ statement executed on line 20 and type in PRINT Q. The computer responds with the last value of Q, which is 10. That's right, so you look at line 30 to find your error.

Leaving out commas is an easy way to get Cursor Out of Range, Value Error, and Input Statement Error. Leaving out the comma between 10 and 20 on line 50 would cause the computer to try to plot 1020, 24 — a non-existent position off the screen. Leaving out the comma between 24 and 30 would cause the computer to try to POKE the number 2430 into memory. Since 255 is the largest number a memory location can contain, this will also generate an error.

Finally, make sure that after you have made corrections and deletions to your program that you press [System Reset]. Sometimes errors cause critical memory locations to change in such a way that even error-free programs cannot run. Prior to every test run, press [System Reset]. Of course once your program runs correctly this is not necessary.

by David Plotkin

done

A Sound Introduction

Many new users have not realized the tremendous potential for music and sound hidden in their ATARI computers. After all, a computer that can produce phaser noise or let you hear Indianapolis cars racing down the straight-away, by altering a few simple commands, should be capable of more.

The following applies to both the 400 and 800 and is completely memory independent.

Sound on the ATARI is really made possible by the same technology that brought you hand-held calculators. I'm talking about the integrated circuit. In this case a special integrated circuit was designed and named POKEY (Pots and Keys). Every ATARI built has this special chip and therefore can play music and generate interesting sounds.

You might think of POKEY as a barber shop quartet, since there are four voices available. Each voice can be turned up loud, or so low it can barely be heard. Each barber (voice) can "sing," or sound, 255 different notes or pitches. Some of these are so similar your ear can't distinguish the differences. Among them are several that correspond to the musical scale (see Table 1). Each voice can be made to sound a pure tone — as if you were to whistle the note — or distort the tone. Distortion is one way of taking a familiar note and making it sound like a growl, hiss or rumble.

TABLE 1

E 193	C 121 Middle C
F 182	D 108
G 162	E 96
A 144	F 91
B 128	

Let's put this in the context of the standard ATARI BASIC statements. SOUND A,B,C,D is the general command format to generate sound, where:

A=Voice, one of the four barbers. A can equal any value from zero to three.

A SOUND INTRODUCTION

B=Pitch or note. This can equal any number from one to 255. The higher the value the lower the note.

C=Distortion. Any even number from zero to 14. Ten gives the purest tone with least distortion.

D=Volume. Any number from one to 15 is legal. A zero turns sound off.

That seems pretty easy, and so it is. Try this. SOUND 0,121,10,8 [RETURN]. This will cause the first barber (his number is zero) to sing middle C with as little distortion as possible. Now vary the volume; try a four and then a 14. Eight is a good volume value when more than one barber is singing. Experiment with distortion; change the 10 to a four, then a 14. Restore the sound statement as it is above. Now, add a second barber.

SOUND 1,72,10,8 This voice sings the note A above C.

SOUND 2,45,10,8 This voice sings the note F.

SOUND 3,193,10,8 This sings E below middle C.

Turn off each barber's voice by making the corresponding volume zero.

To turn off all voices, type END.

The legal abbreviation for the SOUND command is SO.; try it and save typing.

The following sounds should be experimented with. They are presented to get the wheels turning. I'm sure you can all do much better.

Our first sound is an explosion. Change the value of DUR in line 30. Experiment with volume changes in line 90.

```
10 REM EXPLOSION
20 REM DUR-LENGTH OF EFFECT, 1-10
30 DUR=6
40 PITCH=20:GOSUB 80
50 SOUND 1,0,0,0:SOUND 2,0,0,0
60 GOTO 30
70 REM *** SUBROUTINE ***
80 SOUND 2,75,8,15
90 ICR=0.79+DUR/100
100 V1-15:V3-15:REM VOLUME
110 SOUND 0,PITCH,8,V1
120 SOUND 2,PITCH+20,8,V2
```

STARTING LINE

```
130 SOUND 2,PITCH+50,8,V3
140 V1-V1*ICR
150 V2-V2*(ICR+0.05)
160 V3-V3*(ICR+0.08)
170 IF V3>1 THEN 110
180 SOUND 0,0,0,0:RETURN
```

Sound number two is a familiar siren. Change the DUR value in line 30. Try varying the step size in line 60.

```
10 REM SIREN
20 REM DUR=TIME IN SECONDS
30 DUR=10
40 LO=50:HI=35:STP=-1
50 FOR TIME=1 TO DUR
60 FOR PITCH=LO TO HI STEP STP
70 SOUND 0,PITCH,10,14
80 FOR WAIT=1 TO 15:NEXT WAIT
90 NEXT PITCH
100 XX=LO:LO=HI:HI=XX:STP=-STP
110 NEXT TIME
120 SOUND 0,0,0,0:GOTO 30
```

Sound number three is a European variation of the siren. Run it, you'll hear the difference. Experiment with the LO and HI values in line 40.

```
10 REM EUROPEAN SIREN
20 REM DUR=SECONDS RUN
30 DUR=5
40 LO=57:HI=45:PITCH=HI
50 FOR TIME=0 TO DUR*2
60 SOUND 0,PITCH,10,14
70 FOR WAIT=1 TO 180:NEXT WAIT
80 PITCH=LO:LO=HI:HI=PITCH
90 NEXT TIME
100 SOUND 0,0,0,0:GOTO 30
```

Sound four is the whistle and explosion of a falling bomb. Try to determine what makes the whistle sound and what part of the program makes the explosion sound.

A SOUND INTRODUCTION

```
10 REM WHISTLE & BOMB
20 REM DUR=LENGTH OF EFFECT
30 DUR=5
40 V1=4:FOR PITCH=30 TO 75
50 SOUND 0,PITCH,10,V1
60 SOUND 1,PITCH+3,10,V1*.7
70 FOR WAIT=1 TO DUR*3:NEXT WAIT
80 V1=V1*1.03:NEXT PITCH
90 SOUND 2,35,8,12
100 V1=15:V2=15:V3=15
110 PITCH=DUR+5:ICR=.79+DUR/100
120 SOUND 0,PITCH,8,V1
130 SOUND 1,PITCH+20,8,V2
140 SOUND 2,PITCH+50,8,V3
150 V1=V1*ICR
160 V2=V2*(ICR+.05)
170 V3=V3*(ICR+.08)
180 IF V3>1 THEN 120
190 SOUND 0,0,0,0:SOUND 1,0,0,0
200 SOUND 2,0,0,0:GOTO 30
```

Sawing wood is sound five. Try changing the pitch and volume. Also eliminate the wait in line 180.

```
20 REM SAWING WOOD
30 REM DUR=SECONDS RUN
40 DUR=8
50 FOR TIME=1 TO DUR
60 ST=6:VL=12:GOSUB 90
70 ST=8:VL=8:GOSUB 90
80 NEXT TIME:RETURN
90 FOR PITCH=ST+5 TO ST STEP -1
100 GOSUB 160:NEXT PITCH
110 FOR PITCH=ST TO ST+5
120 GOSUB 170:NEXT PITCH
130 SOUND 0,0,0,0:SOUND 1,0,0,0
140 FOR WAIT=1 TO 25:NEXT WAIT
150 GOTO 40
160 SOUND 0,PITCH,2,VL
170 SOUND 1,PITCH,8,VL*.7
180 WAIT=(WAIT/5)*5:RETURN
```

STARTING LINE

There are many opportunities for the experimenter to use the sound command. Perhaps a program using the joystick to vary pitch or distortion would make your experimentation easier. Random notes and harmonies can be very interesting. Look up and use the RANDOM command in your BASIC Reference Manual. If you should write something interesting, let us know. ANTIC is always looking for new, interesting and helpful material.

by Jim Capparell

TYPO

Type Your Program Once

TYPO is designed to help you find typing errors made when entering BASIC programs published in ANTIC. When used properly, TYPO will produce a table of values which can be used to pinpoint where an error was made. ANTIC will publish a table with every BASIC listing, and the user may compare the two tables to ensure they are identical. If they are not, then the user presumably made a "typo" which needs to be corrected.

How To Use TYPO

1. Enter program listing *Exactly* as shown.
2. LIST this program to disk (LIST "D:TYPO.LIS") or cassette (via LIST "C:"). When using a cassette, use an entire blank cassette for just this program.
3. Type NEW to clear memory.
4. Type in a program from this book, or ANTIC magazine.
5. LIST this program to the disk (LIST "D:NAME") or cassette (LIST "C:"). Type NEW and reenter the program (ENTER "D: NAME" or ENTER "C;").

Bill Wilkinson is the president and founder of Optimized Systems Software, Inc., of Cupertino, California. He helped design the original Atari BASIC and has developed several other computer languages including Basic A+ and the new language, Action. Bill's work has been published in a number of publications including ANTIC. His checksum program, TYPO, is continuously used in ANTIC to help ATARI users verify their BASIC programs after typing them in.

6. Append the TYPO program onto the end of the program from the disk (ENTER "D:TYPO.LIS") or cassette (ENTER "C").
7. Type GOTO 32000 and a checksum table will be printed on your screen. Compare this table with the one published; if they agree you are finished and the program should run.
8. Note the value of the "variable checksum" printed on the screen, and keep it handy.
9. If the table does not agree with the published table, examine the lines which have codes and/or lengths which disagree. Correct any errors.
10. *If and only if* the variable checksum you noted agrees with that printed with the program, go to step 7 above and try again.
11. If the variable checksums do NOT agree, you MUST go to step 5 above and perform the listing and reentering ritual! You may skip step 6, however, since presumably you have the combined programs now LISTed together.

Follow these instructions exactly!

What TYPO Is Telling You

THIS PROGRAM IS FUSSY! It cares about every little period, comma, and even spaces. It also cares about the order in which you typed in program lines! The order in which the variable names are stored depends upon the order the lines were typed. Should this order be altered the values of the tokens and the subsequent checksums will be altered.

The "variable checksum" is used to correct for some of this by producing an (almost) unique checksum which depends on the order in which the variables are stored. If your checksum doesn't agree, you have either entered lines in the wrong order or misspelled a variable name. In either case, you *must* correct your error(s) and then go through the LIST/NEW/ENTER sequence to assure that the variables are put back in order.

The length shown is the number of bytes encountered by TYPO within the line number range shown. The two-letter code is essentially a checksum of "length" bytes within that same range. If the length is correct and the checksum is off, you have made a spelling or punctuation error. Watch out: since all keywords and operators (including two-character operators such as "=") are tokenized as one byte, the length might stay the same even though you type SET COLOR for CLR. Note!! You *may* use abbreviations for keywords as long as the LISTed result conforms to the published listing.

STARTING LINE

If the length bytes disagree, you have added or deleted characters. If nothing obvious shows, pay special attention to characters in quoted strings and/or REMark statements. It is easy to omit a space or punctuation in a REMark, thinking that "REMARKs don't matter"; but to TYPO they do.

This is a small but sophisticated program, use it and typing errors will be reduced.

by Bill Wilkinson

NOTE: TYPO asks for output file. Respond with S for television or P for printer.

```
32000 REM Type Your Program Once -- "T
YPO"
32100 CLR :DIM Q$(20):QF=7:CLOSE #QF:?
"File for output ";
32110 INPUT Q$:OPEN #QF,12,0,Q$:QREM=0
32130 QCNT=1:FOR QADDR=PEEK(130)+256*P
EEK(131) TO PEEK(132)+256*PEEK(133)-1
32140 QSUM=QSUM+PEEK(QADDR)*QCNT:QCNT=
QCNT+1:NEXT QADDR
32150 ? #QF;"Variable checksum = ";QSU
M:? #QF
32160 QADDR=PEEK(136)+256*PEEK(137):?
#QF;"    Line num range      Code  Length
"
32170 QLINE=PEEK(QADDR)+256*PEEK(QADDR
+1)
32180 IF QLINE>-32000 THEN END
32190 QLEN=0:QSUM=QLEN:QCNT=QLEN:? #QF
;"    ";QLINE,"- ";
32200 IF NOT (QCNT<12 AND QLEN<500 AN
D QLINE<32000) THEN 32270
32220 QLEN=QLEN+PEEK(QADDR+2):QCNT=QCN
T+1
32230 IF PEEK(QADDR+4)=0 AND QREM THEN
QADDR=QADDR+PEEK(QADDR+2):GOTO 32260
32240 FOR QADDR=QADDR TO QADDR+PEEK(QA
DDR+2)-1
32250 QSUM=QSUM+PEEK(QADDR):NEXT QADDR
32260 Q$=STR$(QLINE):QLINE=PEEK(QADDR)
+256*PEEK(QADDR+1):GOTO 32200
32270 QSUM=QSUM-676*INT(QSUM/676):QCNT
```

```

=INT(QSUM/26)
32280 ? #QF:Q$,CHR$(65+QCNT);CHR$(65+Q
SUM-26*QCNT);"      ";QLEN
32290 GOTO 32180
  
```

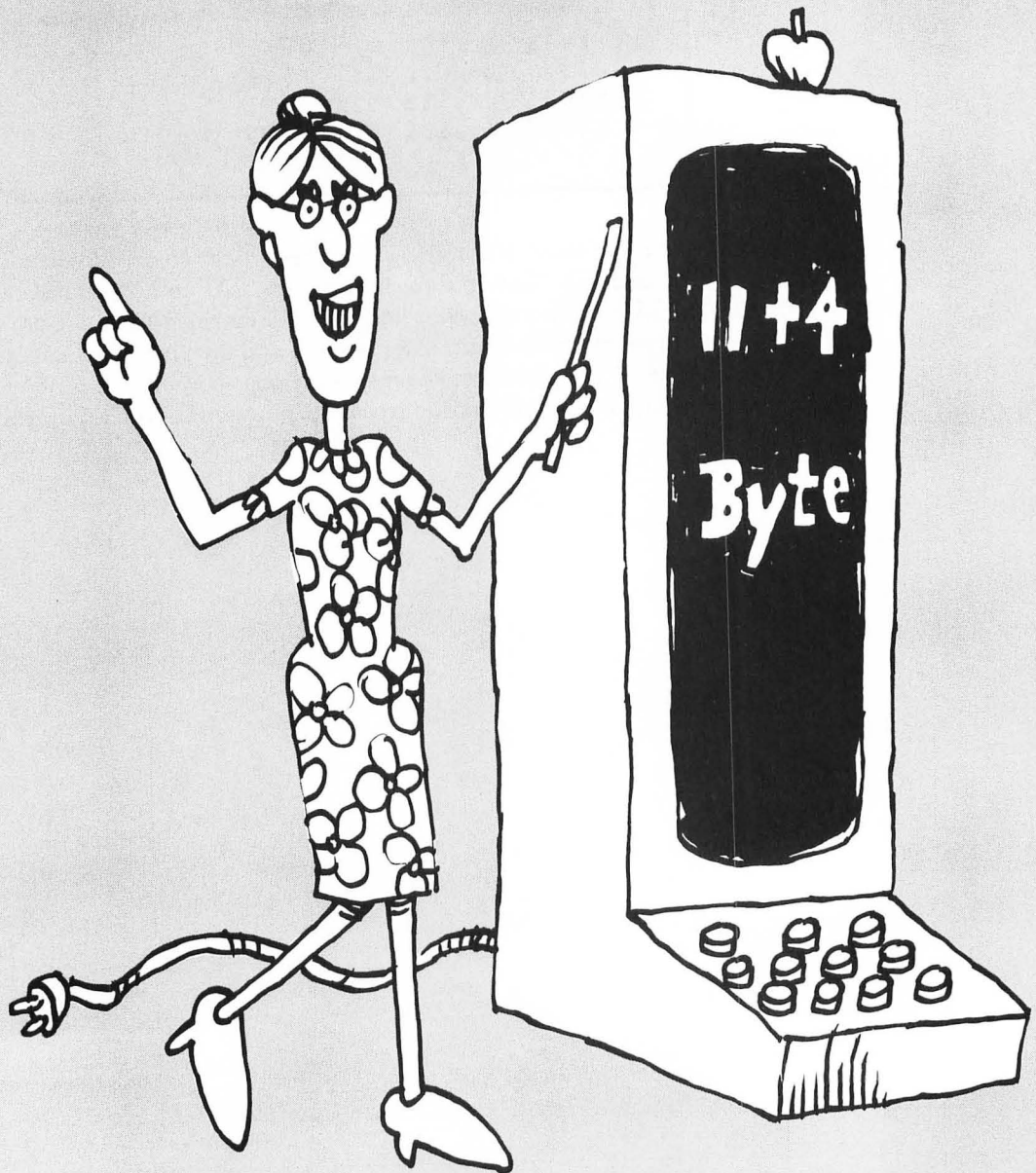
TYPO TABLE

```

Variable checksum = 50796
  Line num range      Code   Length
  32000 - 32200      PT      518
  32220 - 32290      WQ      310
  
```

This TYPO Table is the result of using TYPO to check itself. To do this you must change lines 32180 and 32200 first. In those lines change 32000 to 32500. Then type GOTO 32000 or RUN. Either of these commands will initiate TYPO, ask you to designate your output file (S for screen, P for printer), and then produce a table for itself. This, your first TYPO Table, should match the one above. If it does not, examine your program for typing errors and repeat the process until you get it right.

Education



Spin Colors With The Spider

Since our new ATARI 800 has the GTIA chip, we have been experimenting with it. **Spider** is a little BASIC program that lets you doodle colors with your joystick.

Mode 11 is our choice for this program because it gives 16 different colors in a high-resolution mode (80 pixels horizontally by 192 vertically). When you run the program, a white "spider" appears. The fire button changes the spider's color. As you move the joystick the spider leaves a trail of its color. When the spider is white it can be positioned without leaving a line (it actually draws in background color).

To start a new design, press [RESET] and type RUN.

by John and Mary Harrison

John and Mary Harrison are parents, teachers, and ATARI hobbyists. Mary teaches math and computer science at the high school level. John holds an M.S. degree in computer science and develops educational software. They live in Newport News, VA, and are Contributing Editors to ANTIC's Education Department.

```
10 REM ***** SPYDER *****
20 REM BY JOHN AND MARY HARRISON
40 REM FOR ANTIC JUNE 1982
60 DIM SPIDERS$(1),SSPIDERS$(8),ERASE$(1
0)
70 REM SET ASIDE MEMORY FOR PLAYER
80 REM AND SET GRAPHICS MODE.
90 MEMTOP=PEEK(106)-16
100 POKE 106,MEMTOP
110 REM CLEAR MEMORY FOR PLAYER
120 POKE 88,0:FOR I=0 TO 4:POKE 89,MEM
```


EDUCATION

```
TOP+8+I:? CHR$(125):NEXT I
130 GRAPHICS 7+16
140 MEMTOP=MEMTOP+8
150 REM START OF PLAYER MEMORY
160 POKE 54279, MEMTOP
170 REM DECIMAL ADDRESS OF PLAYER MEMO
RY
180 PMBASE=256*MEMTOP
190 REM SINGLE LINE PLAYER DOUBLE WIDT
H
200 POKE 559,46:POKE 53256,1
210 REM ENABLE PM GRAPHICS
220 POKE 53277,3
230 REM INITIAL PLAYER POSITION
240 XSPIDER=119
250 YSPIDER=48
260 REM CLEAR STRING FOR VERITCAL MOVE
MENT
270 ERASE$=""
280 REM POSITION PLAYER
290 POKE 53248,XSPIDER
300 REM DRAW PLAYER
310 FOR I=PMBASE+511+YSPIDER TO PMBASE
+518+YSPIDER
320 READ DAT
330 POKE I,DAT
340 NEXT I
350 DATA 36,36,90,60,60,90,36,36
360 REM ADDRESS OF ARRAY AND VARIABLE
370 REM TABLES. THIS SECTION OF CODE
380 REM ALLOWS YOU TO USE 128 BYTES
390 REM FOR SPIDERS$ WITHOUT RESERVING
400 REM THAT MUCH MEMORY IN THE DIM.
410 ATAB=PEEK(140)+256*PEEK(141)
420 VTAB=PEEK(134)+256*PEEK(135)
430 OFFSET=256*MEMTOP+512-ATAB
440 M3=INT(OFFSET/256)
450 M2=OFFSET-256*M3
460 POKE VTAB+2,M2:POKE VTAB+3,M3
470 POKE VTAB+4,128:POKE VTAB+5,0
480 POKE VTAB+6,128:POKE VTAB+7,0
490 REM SET UP SHADOW FOR SPIDER
500 SSPIDERS$(1,8)=SPIDERS$(YSPIDER,YSPI
```

```

DER+7)
510 REM INITIALIZE COLOR AND LINE CO-
520 REM ORDINATES.
530 C=0:XLINE=79:YLINE=35
540 SETCOLOR 4,0,0:POKE 704,8
542 SETCOLOR 0,3,8:SETCOLOR 1,6,8
544 SETCOLOR 2,9,8
550 GOSUB 880
560 COLOR C
570 PLOT XLINE,YLINE
580 REM CHANGE COLOR ROUTINE
590 IF STRIG(0)=1 THEN 660
600 C=C+1
610 IF C>3 THEN C=0
620 COLOR C:POKE 704,C*48+8
630 FOR DEL=1 TO 20:NEXT DEL
640 REM READ JOYSTICK AND SET PLAYER
650 REM PARAMETERS APPROPRIATELY.
660 ST=STICK(0)
670 IF ST=15 THEN 590
680 IF ST=6 OR ST=10 OR ST=14 THEN YSP
IDER=YSPIDER-1:YLINE=YLINE-1
690 IF ST>4 AND ST<8 THEN XSPIDER=XSPI
DER+1:XLINE=XLINE+1
700 IF ST=5 OR ST=13 OR ST=9 THEN YSPI
DER=YSPIDER+1:YLINE=YLINE+1
710 IF ST>8 AND ST<12 THEN XSPIDER=XSP
IDER-1:XLINE=XLINE-1
720 REM VERTICAL MOTION OF PLAYER
730 SPIDER$(YSPIDER,YSPIDER+7)=SSPIDER
$
740 SPIDER$(YSPIDER-8,YSPIDER-1)=ERASE
$
750 SPIDER$(YSPIDER+8,YSPIDER+15)=ERAS
E$
760 REM CHECK FOR CURSOR OUT OF RANGE
770 IF XSPIDER<41 THEN XSPIDER=41:XLIN
E=XLINE+1
780 IF XSPIDER>199 THEN XSPIDER=199:XL
INE=XLINE-1
790 IF YSPIDER<14 THEN YSPIDER=14:YLIN
E=YLINE+1
800 IF YSPIDER>108 THEN YSPIDER=108:YL

```

```

INE=YLINE-1
810 REM HORIZONTAL MOTION OF PLAYER
820 POKE 53248,XSPIDER
830 REM DRAW SPIDER TRAIL
840 DRAWTO XLINE,YLINE
850 GOTO 590
860 REM ROUTINE TO DRAW FRAME. THIS
870 REM SHOWS LIMITS OF SCREEN.
880 COLOR 1
890 PLOT 0,0:DRAWTO 0,95:DRAWTO 159,95
: DRAWTO 159,0:DRAWTO 0,0:RETURN

```

TYPO TABLE

Variable checksum = 726158

Line num	range	Code	Length
10	- 140	DS	373 ✓
150	- 260	AV	318 ✓
270	- 380	WM	258 ✓
390	- 500	NU	395 ✓
510	- 600	UB	325 ✓
610	- 720	IR	458
730	- 840	KV	361 ✓
850	- 890	JD	184 ✓

dede except
for chartr at end

Zahrcon

Zahrcon is a modification of the familiar game of Hang-Man. This article shows you how to write it in BASIC with your ATARI computer. The game of Hang-Man has been written for every computer on the market today, but as an educational game it has a major flaw. It rewards the player (child) for failing to guess the word. The kids like to see the little "man" get "hung," especially when the computer enhances this outcome with a special graphics display and noises.

When developing educational games for children, we should save the positive reinforcement for correct behavior. There should not be a reward for wrong answers, especially when deliberate. Zahrcon attempts to improve on Hang-Man by rewarding the player for guessing correctly the letters comprising the secret word generated by the computer. Each proper letter helps build an animated "creature," accompanied by special graphics and sound.

Since Zahrcon is designed for children, some as young as five or six, the letters displayed on the screen should be large and clear. Only one word needs to be displayed at any time, so Graphics Mode 2 is a good choice. Upper case letters with numbers and symbols will be better than lower case, and we will need to redefine some of the symbols into graphics characters that will build the creature.

To redefine a character set, we must decide which characters will not be needed in the program. We must also create our new characters to replace the old ones. Each letter, character, or symbol that is on the screen is made up of eight bytes. Since each byte is eight bits, a character occupies an 8×8 matrix. If a bit is "on" (set to "1"), the corresponding pixel will be lit on the screen. Next, we must calculate the place in the character set where we will be putting our new characters. Figure 1 illustrates how the character set is placed in ROM.

To change the character set, we must first move the character set from

Linda Schreiber, from Garden City, Michigan, is President and co-owner of T.H.E.S.I.S., a company specializing in educational software for microcomputers including the ATARI. Her programs have appeared in many publications, and she has written two books about ATARI programming, both available from TAB Books.

EDUCATION

ROM into RAM, then replace the old characters with the new ones. In this program, we will replace the character set from the quotation mark to the period. To calculate the RAM location of the first character that will be changed, we multiply its position in the character set by eight. The space, which occupies the first eight bytes, is counted as the zero position. The exclamation point is the first position, etc. The quotation marks begin with the sixteenth byte of the character set. This is where our new characters will begin. Figure 2 shows the old character set and the new character set that will replace it.

Once we have redefined our characters, we can begin our program. Our menu will offer two choices: to play the game or to end it. By moving the joystick forward and backward, we can move the arrow up or down on the screen. Press the red button on the joystick when the choice has been made.

While the player is deciding whether or not to play, our creature displays some life. The winking and blinking is obtained by changing the character that is used for the creature's eyes. The character that replaces the apostrophe is used for both eyes, the quotation mark has been replaced with the left eye, the slash is now the right eye, and the asterisk is for no eyes. If the red button has not been pressed after a given amount of time, the program will choose one of the three options and PRINT it in the location of both eyes. After another set amount of time, both eyes will appear again on the screen. This same principle is used at the end of the game when the child wins. Even though it doesn't seem like much, this kind of enhancement can make the difference between a mediocre program and a good one.

The game essentially plays like Hang-Man. The player is rewarded with another part of the creature whenever a letter is guessed that belongs in the secret word. If the child solves the word within a certain number of tries, the creature winks and blinks, and there is a graphics and sound reward.

FIGURE 1

CHARACTER	DECIMAL CODE	POSITION IN CHARACTER SET
[SPACE]	20	0
!	21	1
"	22	2
#	23	3
\$	24	4
%	25	5

.	.	.
[ctrl] A	1	64
[ctrl] B	2	65
[ctrl] C	3	66
.	.	.
.	.	.
.	.	.
a	97	97
b	98	98
c	99	99

The placement of the characters in the character set does not follow their decimal or ATASCII codes.

FIGURE 2

OLD CHARACTER	NEW CHARACTER
"	left eye
#	antennae
\$	left top ear
%	left bottom ear
&	left top head
'	both eyes
(right top head
)	left bottom head
*	no eyes
+	right bottom head
,	neck also part of leg
—	right top ear
.	right bottom ear
/	right eye

The character set on the left will be replaced with the character on the right.

by Linda M. Schreiber

```

10 REM ***** ZAHRCN *****
20 REM BY L.M. SCHREIBER FOR ANTIC MAY
  1982
40 DIM WORD$(10),YWORD$(10),A(26)
50 GRAPHICS 18:REM GRAPHICS 2 WITH NO

```

TEXT WINDOW

```

60 TOP=PEEK(106):REM FIND OUT HOW MUCH
  MEMORY IS AVAILABLE
70 CHBASE=TOP-4:REM PLACE CHARACTER SE
  T 1024 BYTES BELOW TOP OF MEMORY
80 OLDCH=57344:NWCH=CHBASE*256:REM ST
  ARTING BYTES OF OLD CHARACTER SET AND
  NEW CHARACTER SET
90 FOR X=0 TO 511:REM MOVE THE NUMBERS
  , SYMBOLS AND UPPER CASE LETTERS
100 C=PEEK(OLDCH+X):REM GET A BYTE OF
  THE CHARACTER SET FROM ROM
110 POKE NWCH+X,C:REM RELOCATE IT IN R
  AM
120 NEXT X
130 NWCH=NWCH+16:REM DO NOT REPLACE TH
  E SPACE OR THE EXCLAMATION POINT
140 FOR X=NWCH TO NWCH+111:REM BYTES I
  N THE CHARACTER SET TO BE REPLCED
150 READ C:REM READ THE NEW BYTE FROM
  THE DATA BASE
160 POKE X,C:REM REPLACE THE OLD BYTE
  WITH THE NEW ONE
170 NEXT X
180 DATA 255,255,255,255,255,255,63,63
181 DATA 129,66,66,36,36,36,24,24
182 DATA 128,224,120,62,31,31,15,15
183 DATA 15,31,63,127,255,0,0,0
184 DATA 63,63,127,127,255,255,255,255
185 DATA 255,255,255,255,255,255,60,60
186 DATA 252,252,254,254,255,255,255,2
  55
187 DATA 255,255,255,255,127,127,63,63
188 DATA 255,255,255,255,255,255,255,2
  55
189 DATA 255,255,255,255,254,254,252,2
  52
190 DATA 255,126,60,24,24,60,126,255
191 DATA 1,7,30,124,248,248,240,240
192 DATA 240,248,252,254,255,0,0,0
193 DATA 255,255,255,255,255,255,252,2
  52
200 POKE 756,CHBASE:REM CHANGE TO THE
  
```

NEW CHARACTER SET

```

210 POKE 77,0: ? #6;"☒": POSITION 4,3: ?
#6;"#"
220 POSITION 2,4: ? #6;"$&' (- PLAY "
230 POSITION 2,5: ? #6;"%) *+."
240 POSITION 4,6: ? #6;"", "
250 POSITION 3,7: ? #6;"&* ("
260 POSITION 2,8: ? #6;"& , ( end"
270 POSITION 2,9: ? #6;" , * , "
280 POSITION 6,1: ? #6;"ZAHRCOIN": P=4: P1
=4: C=0: GOSUB 990
290 POSITION 10,P: ? #6;" ": POSITION 10
,P1: ? #6;">": REM ERASE THE LAST ARROW
AND POSITION THE NEW ONE
300 C=C+1: IF C<75 THEN S=39: GOTO 330
310 IF C=75 THEN S=42: X=INT(RND(0)*3):
IF X=1 THEN S=34
320 IF X=2 THEN S=47
330 POSITION 4,4: ? #6:CHR$(S): IF C>100
THEN C=0
340 IF STRIG(0)=0 THEN 390: REM CHOICE
HAS BEEN MADE
350 IF STICK(0)=15 THEN 300: REM CHECK
FOR MOVEMENT ON JOYSTICK
360 POKE 77,0: IF STICK(0)=14 THEN P=P1
: P1=P1-4: IF P1<4 THEN P1=4: REM CHECK F
OR TOP OF MENU
370 IF STICK(0)=13 THEN P=P1: P1=P1+4: I
F P1>8 THEN P1=8: REM CHECK FOR BOTTOM
OF MENU
380 GOSUB 990: GOTO 290
390 POKE 77,0: IF P1=8 THEN FE=FRE(S): E
ND
400 ? #6;"☒": FOR X=2 TO 10: POSITION 1,
X: ? #6:CHR$(63+X): NEXT X: REM PRINT LET
TERS ON THE RIGHT
410 FOR X=2 TO 10: POSITION 2,X: ? #6:CH
R$(72+X): NEXT X
420 FOR X=2 TO 9: POSITION 3,X: ? #6:CHR
$(81+X): NEXT X
430 X=INT(RND(0)*15): RESTORE 1000+X
440 READ WORD$
450 L=LEN(WORD$): P=10-L/2: REM POSITION

```



```

TO CENTER THE QUESTION MARKS
460 FOR X=P TO P+L-1:POSITION X,11:? #
6;"?":NEXT X:REM QUESTION MARKS FOR TH
E LETTERS
470 FOR X=1 TO 26:A(X)=0:NEXT X:YWORDS$
(1)="- " :YWORDS$(10)="- " :YWORDS$(2)-YWORD
$:X=1:P1=2:LT1=0:LT2=0
480 LT=0
490 S=(X-1)*9+95+P1:POSITION X,P1:? #6
:CHR$(S):GOSUB 990
500 IF STICK(0)-14 THEN POSITION X,P1:
? #6:CHR$(S-32+A(S-96)):P1=P1-1:IF P1=
1 THEN P1=10:X=X-1
510 IF X=0 THEN X=3:P1=9
520 IF STICK(0)-13 THEN POSITION X,P1:
? #6:CHR$(S-32+A(S-96)):P1=P1+1:IF P1=
11 THEN P1=2:X=X+1:IF X=4 THEN X=1
530 IF P1=10 AND X=3 THEN P1=2:X=1
535 IF STICK(0)-13 OR STICK(0)-14 THEN
POKE 77,0
540 IF STRIG(0)=0 THEN 560
550 GOTO 490
560 POKE 77,0:IF A(S-96)=128 THEN 480
570 A(S-96)=128:REM KEEP LETTER BLUE
ON SCREEN
580 S=S-32:REM GET TRUE CHARACTER VALU
E
590 FOR C=1 TO LEN(WORDS$):IF ASC(WORDS$
(C,C))=S THEN POSITION P+C-1,11:? #6:C
HR$(S):YWORDS$(C,C)-CHR$(S):LT=1
600 NEXT C
610 IF LT=1 THEN 630
620 GOTO 740
630 SN=50:LT1=LT1+1:GOTO 630+LT1*10
640 POSITION 5,7:? #6;"&":POSITION 5,8
:? #6;" ":GOTO 880
650 POSITION 7,7:? #6;" ":POSITION 7,8
:? #6;"*":GOTO 880
660 POSITION 9,7:? #6;"(":POSITION 9,8
:? #6;" ":GOTO 880
670 POSITION 6,6:? #6;"&*(":GOTO 880
680 POSITION 7,5:? #6;" ":GOTO 880
690 POSITION 6,4:? #6;"")*+":POSITION 6

```

```

,3:? #6;"&*"(:GOTO 880
700 POSITION 5,3:? #6;"$":POSITION 5,4
:? #6;"%":GOTO 880
710 POSITION 9,3:? #6;"-":POSITION 9,4
:? #6;".":GOTO 880
720 POSITION 7,3:? #6;""':GOTO 880
730 POSITION 7,2:? #6;"#":POSITION X,P
1:? #6;CHR$(S+128):GOTO 880
740 SN=90:LT2=LT2+1:GOTO 740+LT2*10
750 POSITION 14,7:? #6;"&":POSITION 14
,8:? #6;"":GOTO 880
760 POSITION 16,7:? #6;"":POSITION 16
,8:? #6;"*":GOTO 880
770 POSITION 18,7:? #6;"(":POSITION 18
,8:? #6;"":GOTO 880
780 POSITION 15,6:? #6;"&*"(:GOTO 880
790 POSITION 16,5:? #6;"":GOTO 880
800 POSITION 15,4:? #6;")*+":POSITION
15,3:? #6;"&*"(:GOTO 880
810 POSITION 14,3:? #6;"$":POSITION 14
,4:? #6;"%":GOTO 880
820 POSITION 18,3:? #6;"-":POSITION 18
,4:? #6;"":GOTO 880
830 POSITION 16,3:? #6;"":GOTO 880
840 POSITION 16,2:? #6;"#":POSITION X,
P1:? #6;CHR$(S+128)
850 POSITION P,11:? #6;WORDS$:SOUND 0,2
00,10,10:GOSUB 990:SOUND 0,0,0,0
860 IF STRIG(0)=0 THEN GOSUB 990:POSIT
ION 0,0:GOTO 210
870 GOTO 860
880 FOR SS=16 TO 0 STEP -2:SOUND 0,SN,
10,SS:NEXT SS
890 IF YWORDS$(1,L)<>WORDS$ THEN 480
900 IF LT1<10 THEN 630:REM FINISH BODY
910 POSITION 6,1:? #6;"":GOSUB
990:POSITION 6,1:? #6;"HURRAY!!":X=IN
T(RND(1)*3)+1:ON X GOTO 920,930,940
920 S=42:GOTO 950
930 S=47:GOTO 950
940 S=34
950 IF STRIG(0)=0 THEN GOSUB 990:GOTO
210

```

EDUCATION

```
960 POSITION 7,3:? #6;CHR$(S):REM PRIN
T AN EYE
970 GOSUB 990
980 POSITION 7,3:? #6;""':GOSUB 990:GO
TO 910
990 FOR TIME=1 TO 50:NEXT TIME:RETURN
1000 DATA COULD
1001 DATA COUPLE
1002 DATA KENNEL
1003 DATA KINDS
1004 DATA CROCODILE
1005 DATA FRECKLES
1006 DATA BACKWARDS
1007 DATA PACKAGE
1008 DATA NICKEL
1009 DATA MECHANIC
1010 DATA LEPRECHAUN
1011 DATA ORCHESTRA
1012 DATA SKUNK
1013 DATA TRAGIC
1014 DATA ANTIQUE
```

TYPO TABLE

Variable checksum = 438736

Line	num	range	Code	Length
10	-	100	NJ	505
110	-	184	RE	432
185	-	220	GQ•	481
230	-	310	ZJ•	523
320	-	400	ZZ•	588
410	-	490	UQ•	556
500	-	560	ZF•	508
570	-	660	XL•	527
670	-	740	FD•	506
750	-	820	GM•	572
830	-	910	MJ	571
920	-	1003	BU	306
1004	-	1014	DM•	150

done

Tuning Your ATARI

There's something about music that fascinates kids. Give them a small piano, drums, harmonica, and they will sit for hours creating their own melodies. A few years ago there was a toy piano on the market that contained a tape recorder. This was a big hit with my daughter. Now, she could not only make her own music, but listen to it afterward.

Tuning Your ATARI uses this idea. It is a musical game for children. Type it in and run it, and you will see a simple menu. Choice #3 demonstrates the program. Choice #1 allows you to compose a tune, and Choice #2 will play it back. The tones appear to be made by little figures jumping on a bellows.

Above each figure is the letter name of the tone which that bellows will produce. To operate the bellows, press [1], then press any key from [1] to [8]. Key [1] corresponds to the low C; Key [8] to high C. When a number is pressed, the character will jump down on the bellows, flapping his arms as the bellows is compressed. Once the tone is played, he bounces back up to his original position. The program can hold up to 100 notes. If your melody is less than 100 notes, press the [ESC] key and the menu will reappear on the screen. Press [2] to hear your melody.

Young children will enjoy this program just to see the characters jump up and down while they are playing the tunes. Slightly older children will enjoy listening to the tunes that they have created. The letters above the characters do not attract attention, but are a subtle reminder of the names of the notes. After a while, children will begin to associate the letters with the tones of the character. Don't be surprised if you hear your child singing "A-G-F-G-A-A-A!"

Once again, in this program, we will move the character set out of ROM and into RAM so that we can change some of the characters. In line 70, P1\$ should equal h, reverse quotation marks, control D, reverse space, control comma, reverse l, reverse M, reverse control Q. The characters from K to R are all in reverse. The last character in the string is control period. This string is the machine language subroutine that moves the characters.

EDUCATION

Variables Used in This Program

P1\$	=	machine language subroutine.
M\$	=	string holds the melody played.
A	=	location of the new character set. This value is POKEd in- to 756 to change to the new character set.
TONE	=	line number that starts the tone for the key pressed.
WAIT	=	line number for the timing routine.
Q	=	no function.
CHBS	=	first decimal location of the new character set.
X	=	no function — used in FOR . . . NEXT loops.
C	=	used in READ for new character set, used for value of key pressed, and for position of character.
K	=	counter for the note being entered or played.
T	=	value of the tone to be played.
TL	=	value used in timing loop.
ROUTINE	=	the line number that the program goes to when entering the melody, or playing one back.

by Linda M. Schreiber

```
10 REM ***** TUNING YOUR ATARI ****  
****  
20 REM BY L.M.SCHREIBER FOR ANTIC AUGU  
ST 1982  
40 DIM P1$(20),M$(100)  
50 GRAPHICS 18:POKE 711,PEEK(710):POKE  
710,100  
60 A=PEEK(106)-8:POKE 204,A:POKE 206,2  
24:REM STORE THE BEGINNING OF NEW & OL  
D CHARACTER SETS  
70 P1$="hIM-KHPyINfLJPr":TONE=430  
:WAIT=500:REM MACHINE LANGUAGE SUBROUT  
INE MOVES THE CHARACTER SET  
80 Q=USR(ADR(P1$)):CHBS=A*256:POKE 756  
,A:REM INSTALL NEW CHARACTER SET  
90 FOR X=CHBS+8 TO CHBS+71:READ C:POKE  
X,C:NEXT X:REM CHANGE THE CHARACTERS  
FROM I TO $
```

```

100 DATA 0,254,124,254,124,254,124,254
,108,0,254,254,124,254,124,254,40,108,
0,254,254,254,124,254
110 DATA 186,40,108,0,254,254,254,254,
56,108,56,16,254,56,40,108,0,56,108,56
,146,124,56,40
120 DATA 0,0,56,108,56,16,254,56,0,0,0
,56,108,56,16,124
130 OPEN #2,4,0,"K":REM OPEN THE KEYB
OARD FOR READ
140 POSITION 2,9:? #6;"! ! ! ! ! ! ! !
":POSITION 2,8:? #6;"% % % % % % % %"
REM THE ! AND % ARE THE NEW CHARACTERS
150 POSITION 2,6:? #6;"c d e f g a b c
":REM PLACE THE TONE NAMES
160 POKE 710,100:REM RESTORE THE MENU
170 POSITION 2,0:? #6;"1. PLAY KEYBOAR
D"
180 POSITION 2,2:? #6;"2. REPEAT MELOD
Y"
190 POSITION 2,4:? #6;"3. PLAY EXAMPLE
"
200 K=0:GET #2,C:POKE 710,0:REM GET TH
E KEY PRESSED-REMOVE THE MENU
210 IF C>127 THEN C=C-128:POKE 694,0:R
EM INVERSE FLAG IS ON RESET IT TO NORM
AL
220 IF C<49 OR C>52 THEN POKE 764,255:
GOTO 200:REM NOT A NUMBER FROM 1 TO 4
230 C=C-48:REM GET THEN NUMBER
240 ON C GOTO 250,540,520,560
250 M$="":REM REMOVE CONTENTS OF THE S
TRING
260 ROUTINE-260:K=K+1:IF K=101 THEN 16
0:REM ONLY ACCEPT 100 NOTES
280 GET #2,C:IF C=27 THEN 160:REM GET
THE KEY PRESSED-RETURN TO MENU ON ESCA
PE KEY
290 IF C>127 THEN C=C-128:POKE 694,0:R
EM INVERSE FLAG IS ON RESET IT TO NORM
AL
300 IF C<49 OR C>56 THEN POKE 764,255:
GOTO 200:REM NOT A NUMBER FROM 1 TO 8

```

EDUCATION

```
310 C-C-48:M$(K,K)-STR$(C):REM GET THE
N NUMBER-PUT IT IN THE STRING
320 C-C*2:REM OFFSET IT FOR THE PROPER
POSITION
330 ON C/2 GOSUB 350,360,370,380,390,4
00,410,420
340 GOTO ROUTINE
350 T-121:GOTO TONE:REM 'C'
360 T-108:GOTO TONE:REM 'D'
370 T-96:GOTO TONE:REM 'E'
380 T-91:GOTO TONE:REM 'F'
390 T-81:GOTO TONE:REM 'G'
400 T-72:GOTO TONE:REM 'A'
410 T-64:GOTO TONE:REM 'B'
420 T-60:REM 'C'
425 REM LINES 430-450 MAKE THE CHARACT
ER APPEAR TO PUSH DOWN ON THE BELLOW A
ND MAKE THE TONE
430 TL-10:POSITION C,8:? #6:CHR$(134):
POSITION C,9:? #6:CHR$(130):SOUND 0,T,
10,6:GOSUB WAIT
440 POSITION C,8:? #6:CHR$(135):POSITI
ON C,9:? #6:CHR$(131):SOUND 0,T,10,8:G
OSUB WAIT
450 POSITION C,8:? #6:CHR$(136):POSITI
ON C,9:? #6:CHR$(132):GOSUB WAIT
460 SOUND 0,T,10,10
470 POSITION C,8:? #6:CHR$(135):POSITI
ON C,9:? #6:CHR$(131):SOUND 0,T,10,8:G
OSUB WAIT
475 REM LINES 470-490 RETURN THE CHARA
CTER AND BELLOW TO THE CORRECT POSITIO
N
480 POSITION C,8:? #6:CHR$(134):POSITI
ON C,9:? #6:CHR$(162):SOUND 0,T,10,6:G
OSUB WAIT
490 POSITION C,8:? #6:"%":POSITION C,9
:? #6:"!":SOUND 0,0,0,0:RETURN
500 FOR X-1 TO TL:NEXT X:RETURN :REM T
IMING LOOP
510 REM PLAY A SAMPLE TUNE
520 M$-"11556654433221"
530 REM ROUTINE TO PLAY BACK THE MELOD
```


Y ENTERED

```
540 ROUTINE=540:K=K+1:IF K<=LEN(M$) TH  
EN C=VAL(M$(K,K)):GOTO 320:REM KEEP PL  
AYING UNTIL THE END OF THE STRING  
550 GOTO 160  
560 END
```

TYPO TABLE

Variable checksum = 225282

Line num	range	Code	Length
10	- 90	AA	524
100	- 150	SI	504
160	- 240	ZA	517
250	- 330	UO	571
340	- 440	GF	511
450	- 500	RF	519
510	- 560	RN	217

done

Candle, Candle, Burning Bright

Most computers owned by schools are used in the math department, a recent survey showed. Computer science ranked second. The prime use for computers in any school is drill and practice.

In drill and practice, the computer gives the student questions. If the questions are answered correctly, the student is rewarded. If the answer is wrong, the correct answer appears on the screen. Some educators frown on this, calling it “electronic flash cards.” Others praise such programs, stating that they aid the teacher by reinforcing facts that children need to know.

Another type of educational software is the tutorial, where the computer “teaches” a particular lesson. Some tutorial programs make the computer an electronic page-turner; others allow the students to learn at their own pace, test the students, then review material or present new material based on the results of the test.

Some programs are advertised as educational games. They present learning as a *fun* experience. Some vendors will advertise a game as educational if any single thing is learned. Arcade games are even called educational because they teach “hand-eye coordination.” Maybe they do, but does this mean that they are truly educational games?

There is another educational category — simulation. This is one area where computers could be used to better advantage. There are very few good simulation programs available.

This program simulates a science experiment. A candle is drawn on the screen, and a jar is hovering above it. The program is very simple. To light the candle, press [SELECT]. To lower or raise the jar, press [START]. The candle cannot be lit if the jar has been lowered, but the jar can be lowered or raised whether or not the candle is lit. The white dots that move around on the screen represent the oxygen in the air.

This is a fairly standard experiment, and with a program like this, young children can learn about their environment safely. To light the candle, press

CANDLE, CANDLE, BURNING BRIGHT

[SELECT] and hold it down until the flame appears above the candle. The oxygen dots will move around on the screen. The flame on the candle will flicker because of the air movement.

Hold down the [SELECT] button until the jar starts to move. Once the jar is over the candle, the oxygen will begin to disappear. The oxygen still moves in the jar and the flame will flicker. When all the oxygen is used up, the flame will go out.

Hold down the [START] button until the jar starts to move up again. Notice that the oxygen dots will appear around the candle. If the jar is raised just before all the oxygen is used up, more oxygen dots will gather around the candle, and the flame will not go out.

This program uses the Player/Missile graphics for the jar, candle and the flame. Lines 50 and 60 contain the machine language to move the player (jar) up and down. Be sure that these lines are typed in exactly, or the program will not work correctly.

Variables Used in This Program

UP\$	=	machine language subroutine to move player up.
DOWN\$	=	machine language subroutine to move player down.
A	=	free memory less 8K.
PMBASE	=	beginning of the memory for players and missiles.
CANDLE	=	memory location of where the candle will be drawn.
FLAME	=	memory location of where the flame will be drawn.
JAR	=	memory location of where the jar will be drawn.
C	=	column where oxygen will be plotted.
R	=	row where oxygen will be plotted.
OS(50,2)	=	column and row of oxygen on screen.
OJ(10,2)	=	column and row of oxygen under jar.
FL	=	state of flame (1=flame lit, 0=flame out).
JU	=	state of jar (1=jar down, 0=jar up).
OX	=	amount of oxygen visible.
F	=	which of the three flames to draw.
B	=	data being read.
X,Q,M	=	dummy variables.

by Linda M. Schreiber

EDUCATION

```
10 REM *** BURNING CANDLE SIMULATION
***
20 REM BY LINDA M. SCHREIBER FOR ANTIC
DEC. 1982
30 DIM OS(50,2),OJ(10,2),UP$(13),DOWNS
(13)
40 A=PEEK(106)-32:REM SET ASIDE 2K FOR
PLAYER/MISSILE GRAPHICS - GRAPHICS 7
NEEDS 4K
50 UP$="--h"
60 DOWNS="--h"
70 GRAPHICS 7:REM HIGH RESOLUTION WITH
TEXT WINDOW
80 POKE 54279,A:PMBASE=A*256:REM TELL
ANTIC WHERE P/M GRAPHICS BEGIN
90 POKE 559,62:POKE 53277,3:REM ENABLE
P/M GRAPHICS FOR SINGLE LINE RESOLUTI
ON
100 POKE 704,104:REM COLOR OF FLAME
110 POKE 705,200:REM COLOR OF CANDLE
120 POKE 706,120:REM COLOR OF JAR
130 POKE 708,154:REM COLOR OF OXYGEN
140 POKE 709,8:REM COLOR OF DISH
150 FOR X=PMBASE+1024 TO PMBASE+2043:P
OKE X,0:NEXT X:REM CLEAR MEMORY FOR GR
APHICS
160 COLOR 2:PLOT 100,75:DRAWTO 110,70:
DRAWTO 40,70:POSITION 50,75
170 POKE 765,2:XIO 18,#6,0,0,"S:"
180 CANDLE=PMBASE+1426:REM LOCATION OF
CANDLE IN P/M MEMORY
190 RESTORE 510:FOR X=0 TO 25:READ B:P
OKE CANDLE+X,B:NEXT X:REM READ IN THE
DATA FOR CANDLE
200 POKE 53249,120:REM PUT CANDLE ON S
CREEN
210 FLAME=PMBASE+1157:REM LOCATION OF
FLAME IN P/M MEMORY
220 JAR=PMBASE+1606:POKE 206,INT(PMBAS
E+1536)/256:POKE 205,(PMBASE+1536)-INT
((PMBASE+1536)/256)*256:REM JAR IN P/M
230 POKE JAR,255:FOR X=1 TO 50:POKE JA
R+X,129:NEXT X:REM DRAW THE JAR
```

CANDLE, CANDLE, BURNING BRIGHT

```
240 POKE 53258,3:POKE 53250,107:REM PU
T THE JAR ON THE SCREEN
250 COLOR 1:FOR X=1 TO 50:REM PUT OXYG
EN ON SCREEN
260 C=INT(RND(1)*160):REM COLUMN OF OX
YGEN
270 R=INT(RND(1)*80):REM ROW OF OXYGE
N
280 IF C>60 AND C<90 THEN IF R>43 THEN
270:REM DON'T PLACE IT IN THE JAR
290 IF C>40 AND C<110 THEN IF R>69 THE
N 270:REM OR ON SAUCER
300 OS(X,1)=C:OS(X,2)=R:REM PLACE THE
OXYGEN LOCATION IN THE ARRAY
310 PLOT C,R:NEXT X:REM DO IT 50 TIMES
320 FOR X=1 TO 10:REM OXYGEN IN JAR
330 C=INT(RND(1)*23)+63:R=INT(RND(1)*
23)+46:REM AREA OF JAR
340 OJ(X,1)=C:OJ(X,2)=R:REM PLACE IN J
AR ARRAY
350 PLOT C,R:NEXT X:OX=10:REM DO IT 10
TIMES
360 POKE 752,1:? "PRESS START TO MOVE
JAR":? :? "PRESS SELECT TO LIGHT CANDL
E":REM INSTRUCTIONS
370 IF PEEK(53279)=7 THEN 400:REM NO K
EY PRESSED - MOVE OXYGEN & FLAME IF LI
T
380 POKE 77,0:IF PEEK(53279)=5 AND FL=
0 AND JU=0 THEN 410:REM TURN OFF ATTRAC
T - LIGHT FLAME?
390 IF PEEK(53279)=6 THEN GOSUB 430:IF
JU=0 THEN COLOR 1:FOR X=1 TO 10:PLOT
OJ(X,1),OJ(X,2):NEXT X
400 IF FL=0 THEN GOSUB 540:GOTO 370:RE
M FLAME NOT LIT
410 FL=1:POKE 53248,120:GOSUB 520:REM
ANIMATE FLAME ON SCREEN
420 GOTO 370
430 IF JU=0 THEN FOR Q=1 TO 51:M=USR(A
DR(DOWN$)):NEXT Q:JU=1:RETURN :REM MOV
E JAR DOWN
440 FOR Q=1 TO 51:M=USR(ADR(UP$)):NEXT
```

```

Q:JU=0:OX=10:RETURN:REM MOVE JAR UP
500 REM DATA FOR CANDLE
510 DATA 8,8,12,28,28,30,62,62,126,126
,126,126,126,126,126,126,126,126,1
26,126,126,126,126,126,126
520 F=INT(RND(1)*3)+1:REM PICK ONE OF
THREE FLAME POSITIONS
530 RESTORE 530+F:FOR X=0 TO 9:READ B:
POKE FLAME+X,B:NEXT X:REM READ IN THE
DATA FOR FLAME
531 DATA 16,8,12,28,62,62,28,24,8,4
532 DATA 8,4,6,12,60,60,28,48,16,8
533 DATA 32,16,24,56,30,30,12,12,4,2
539 REM DECREASE THE OXYGEN IF FLAME I
S ON AND JAR IS DOWN. FLAME GOES OUT W
HEN THERE IS NO OXYGEN
540 IF JU=1 AND FL=1 THEN COLOR 4:PLOT
OJ(OX,1),OJ(OX,2):OX=OX-1:IF OX=0 THE
N FL=0:POKE 53248,0:RETURN
550 IF OX=0 THEN 580:REM NO OXYGEN IN
JAR
560 FOR X=1 TO OX STEP 2:R=INT(RND(1)*
23)+63:C=INT(RND(1)*23)+46:COLOR 4:PLO
T OJ(X,1),OJ(X,2):OJ(X,1)=R:OJ(X,2)=C
570 COLOR 1:PLOT R,C:NEXT X
580 FOR X=1 TO 50 STEP 5:COLOR 4:PLOT
OS(X,1),OS(X,2):C=INT(RND(1)*160):REM
GET A NEW COLUMN
590 R=INT(RND(1)*80):IF C>60 AND C<90
THEN IF R>43 THEN 590:REM IN THE JAR!
600 IF C>40 AND C<110 THEN IF R>69 THE
N 590:REM ON THE SAUCER!!
610 OS(X,1)=C:OS(X,2)=R:COLOR 1:PLOT C
,R:NEXT X:RETURN

```

TYPO TABLE

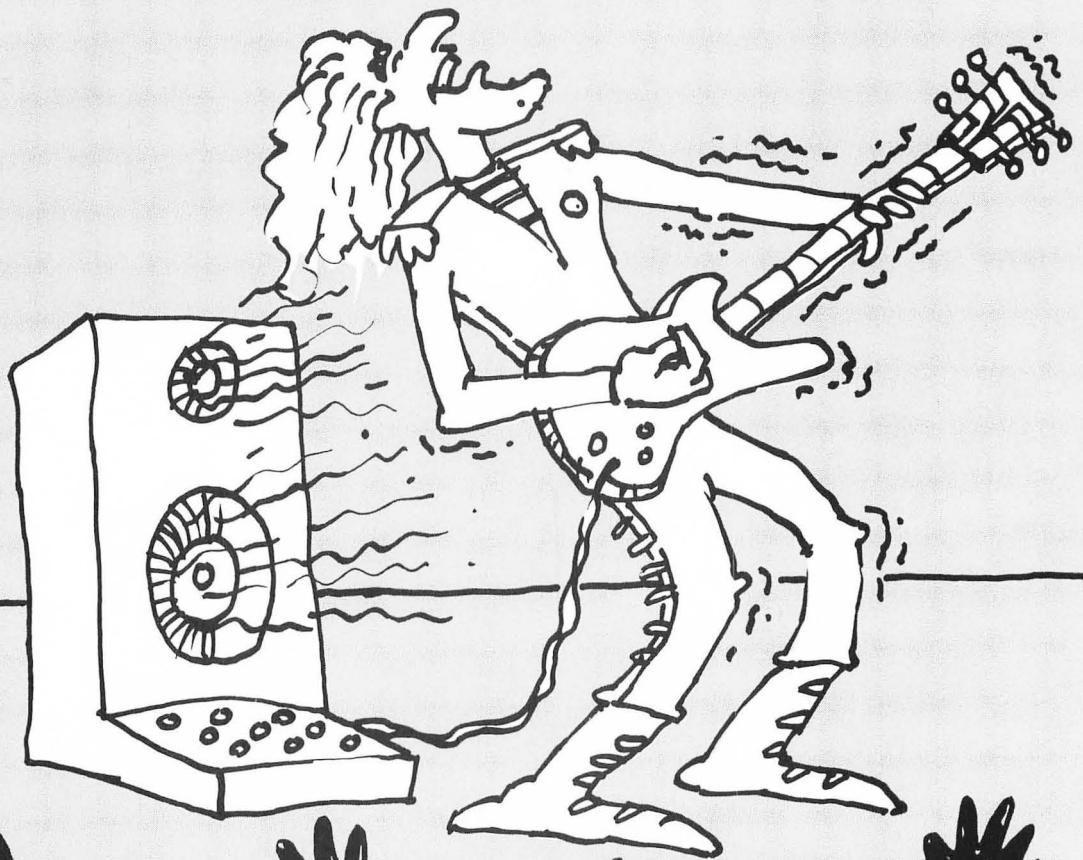
Variable checksum = 367255

Line num range	Code	Length
10 - 100	MN	529
110 - 200	IQ	543

CANDLE, CANDLE, BURNING BRIGHT

210	-	280	PO	522
290	-	370	MW	571
380	-	440	XH	507
500	-	540	QN	587
550	-	610	ZS	552

Sound and Music



Some Sound Advice

The SOUND statement in Atari BASIC is very powerful. Its ability to modify tone, distortion, and volume for each of four voices has been put to good use elsewhere in this book. One of the problems with the SOUND statement is that using it extensively slows down program execution. While this is true of BASIC statements in general, with the SOUND statement there is an easy alternative — SOUND registers. SOUND registers are memory locations which control properties (tone, distortion and volume) of the ATARI's sound.

Memory Location	Function
53760	Tone of Voice 1 (SOUND 0)
53761	Distortion and Volume of Voice 1
53762	Tone of Voice 2 (SOUND 1)
53763	Distortion and Volume of Voice 2
53764	Tone of Voice 3 (SOUND 2)
53765	Distortion and Volume of Voice 3
53766	Tone of Voice 4 (SOUND 3)
53767	Distortion and Volume of Voice 4
53768	Tone "clock" control

The even-numbered memory locations (53760, 62, 64, 66) control the TONE, i.e., which note the ATARI will play. This is identical to the second number in a SOUND statement. For example, to get the same tone as SOUND 0, 100, 10, 8 you would POKE 53760, 100. This specifies Voice 0, note 100. But what about distortion and volume? The odd-numbered memory locations (53761, 63, 65, 67) take care of these two characteristics for each voice via the following relation:

$$16 * \text{DISTORTION} + \text{VOLUME}$$

where DISTORTION is the third number in the SOUND statement (10 in our example) and VOLUME is the fourth number (8 in our example).

The equivalent POKE in our example is $16 * (10) + 8 = 168$, and you would specify POKE 53761, 168. Try it. Type in: POKE 53760, 100:POKE 53761, 168 [RET]. The other pairs of registers work the same way.

SOUND & MUSIC

You can turn off the note by specifying zero in either TONE or DISTORTION-and-VOLUME registers.

Memory location 53768 is an interesting one. The ATARI maintains two internal "clocks" which it uses to measure the frequency of the sound wave it generates. The two clocks run at different speeds. Switching clocks changes the frequency (and thus the tone) of the sound. Bit 1 of memory location 53768 controls which "clock" the ATARI uses to produce its sound. Normally bit 1 is off, and the ATARI's sounds correspond to the tables in the reference manual. Turning bit 1 on (POKE 53768, 1) selects the slower clock, and alters the tone produced. Toggling bit 1 off and on will switch all four voices up and down for a pretty good "alarm" effect. Try this loop:

```
FOR N=0 TO 255:POKE 53768, N: NEXT N
```

This turns bit 1 off and on very nicely without having to worry about setting and resetting the bit. The reason this works is that the values jump back and forth from odd to even, turning bit 1 on and off.

How much faster is POKE than SOUND? Well, let's try an example. The following program downloads the ROM character set into RAM so it can be modified. With no sound (leave out the SOUND statements in line 30), this process takes 15.7 seconds. There are much faster ways to do this, but you can use this method until you feel confident. Fifteen seconds is a long time to sit looking at a computer doing nothing visible. Most people start getting nervous and wondering if "Lockup" has struck again. Let's add some sound to assure the user that something is happening.

```
10 POKE 106,PEEK(106)-4:POKE 53761,168:POKE  
   53763,168:GRAPHICS 0  
20 CHBASE=PEEK(106):OLDCH=57344:NWCH=  
   CHBASE*256  
30 FOR X=0 TO 1024:C=PEEK (OLDCH+X):POKE  
   NWCH+X,C:SOUND 0,C,10,4:SOUND 1,X,10,4  
40 NEXT X
```

Downloading the character set now takes some 25 seconds. If we try the following instead, substituting POKes, the character set loads in about 20 seconds.

```
10 POKE 106,PEEK(106)-4:POKE 53761,168:POKE  
   53763,168:GRAPHICS 0  
20 CHBASE=PEEK(106):OLDCH=57344:NWCH=
```

```
CHBASE*256
30 FOR X=0 to 1024:C=PEEK(OLDCH+X):POKE
  NWCH+X,C:POKE 53760,C:POKE 53762,140
40 NEXT X
```

Note that X, which varies from 0 to 1024, can be used as an input to the SOUND statement — each time it rolls over a multiple of 255, it starts over at 0 (thus 256 is 0, as is 513 and 769). This is *not* true of the POKE statement, so a constant was used. Doing a calculation to keep everything in range (such as POKE 53762, X/4) slows things down still further (about 28 seconds), and isn't a good idea.

Finally, various sources give the equations that relate tone to the internal clocks and note frequency. While these equations are beyond the scope of this article, they can be useful to those composing music on their computer.

by Dave Plotkin

Audio While You CLOAD

“Your mission, Jim, if you choose to accept it . . .”

There is no question that the microcomputer community dislikes computer cassettes — and with good reason. In the early days of computing when hobbyists had no other storage medium, hours of frustration were spent trying to save or load programs from cassettes. When disk storage became available, many hobbyists gladly junked their cassettes. Some manufacturers have quietly stopped supporting their cassette systems.

Unfortunately, this has prejudiced software developers against the use of the Atari cassette system. However, I consider this component one of Atari's strongest points. The Atari system, unlike most others, uses a cassette player made specifically to run on the ATARI.

John Victor is the president of Program Design, Inc., a software company in Greenwich, Connecticut, specializing in games and educational software. Several of their cassette-based products use a voice track on the cassette to enhance the program during loading or play of the game.

SOUND & MUSIC

The advantage is this: the Atari cassette is recorded in stereo. The digital information for programs is stored on the right track. Sound recorded on the left track is played back through the user's TV set. The existence of the left-side sound track means that recorded voice or music can be played at any time while the computer is on — either during the running of a program or during the loading of a program.

One technique that we use at **PDI** is to put voice instructions on the left sound track to play while a cassette is loading. This means that we do not have to put all instructions for using the program in the program itself, reducing the memory requirements. At least half of the Atari market consists of 16K ATARI 400 computers. By keeping memory requirements within 16K (and providing programs in cassette format) a software publisher will reach a greater percentage of the Atari market.

The existence of a voice track gives the program user something to do in the time it takes to load the program. This can set the mood for the game itself. In **MOONBASE IO** we use the voice to give the player a "recorded message" from Earthbase control as to the nature of the mission (just as in "Mission Impossible"). Most of the four and a half minutes it takes to load the program is spent doing something related to playing the game.

To create and use the voice track during the cassette load, several things have to be done. First, the sound that the computer makes during a cassette load has to be turned off. This is done with a POKE 65,0. This can be put in a loader program placed first on the cassette. This loader program will contain a visual display, the POKE 65,0, and a CRUN routine that will automatically load and run the main program.

The following is the CRUN routine. POKE 764,32 will automatically pro-

```
1000 REM ROUTINE TO CRUN NEXT PROGRAM
1002 DIM A$(20)
1005 POKE 65,0
1010 POKE 764,32
1020 FOR LOOP=1 TO 19
1022 READ X:A$(LOOP,LOOP)=CHR$(X)
1024 NEXT LOOP
1030 X=USR(ADR(A$))
1040 DATA 162,253,154,169,183,72,169,8
4,72,169,4,32,182,187,169,255,76,4,187
```

duce a carriage return so that the next program will begin loading. The ASCII values in the REM statement are those for the machine language CRUN routine found in the USR routine. (USR routines are used to run machine language from BASIC.)

After the first program is loaded and run, instructions will be put on the screen and the next program load started. Any recorded sound in the left channel will now be heard clearly in the TV set. Positioning of the recording tape is important.

Atari programs have a two-second string of zeros recorded at the end of each cassette program. The programs stop loading two seconds before the recorded program ends. This means that the recorded voice or music can begin just before the first program ends, but must end two seconds before the main program's record track. Otherwise the computer is going to turn off the voice track before it finishes.

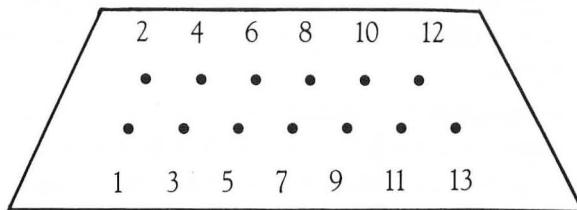
The Atari 410 Program Recorder can play back voice and music but cannot be used to record it. This must be done on a stereo tape deck or a reel-to-reel recorder. For the sake of quality, master tapes from which cassettes are going to be manufactured should be made on reel-to-reel recorders *only*! Cassette recorders do not produce good enough sound to be copied. There is just too much speed variation and lack of separation between the two stereo tracks on cassette masters. If the user only wants a few copies, then a stereo tape deck is okay, but this is not acceptable for commercial software producers.

The first step in making the master tape is to record the programs. The ATARI computer makes no provision to connect the ATARI to a stereo recorder, so the programmer will have to rig up something. This is not very difficult. The "data out" and the "ground" pins in the peripheral connector are the ones that send the program signal to the recorder.

These can be connected to the recorder with a cable that has alligator clips on one side and an RCA connector on the other. A local Radio Shack or audio dealer may have this, or an audio technician can make one. The alligator clips are then connected to the ATARI pins 5 and 6, and the RCA is plugged into the right recording jack of the stereo unit. It's not a bad idea to put tape over the alligator clips to keep them from touching the wrong points.

Before recording, start the computer outputting and set the VU meter on the recorder at between 7 and 5. Also note the reading on the tape counter.

SERIAL I/O PORT CONNECTOR



1. Clock In
2. Clock Out
3. Data In to Computer
4. Ground
5. Data Out of Computer
6. Ground

-
7. Command

8. Motor Control

-
9. Proceed
 10. + 5 / Ready
 11. Audio In
 12. + 12

-
13. Interrupt

Record the loader program. The computer will lay down 18 seconds of pilot tone before the program is recorded. However, after the program is loaded, the computer will continue to output pilot tone. Listen to the computer for an indication of when the program stops, and immediately shut off the recorder. Next, record the main program. Using the tape counter, keep track of where on the tape the second program is.

The voice (or music) can now be recorded. To record voice, connect a microphone to the left-side "mic" jack. The recorder must be one that will not erase the right track while the left one is being recorded. This can be determined quite simply — there must be a separate record button for each track.

Using the tape counter as a guide, rewind the tape. Then begin recording voice instructions and/or music on the left track. This must be finished two seconds before reaching the end of the recorded program (because that is where the computer is going to stop when the program is loading).

It will also help to have an appropriate graphic on the screen while the main program is being loaded. If directions are being given, the directions might also appear on the screen at the same time.

This technique can enhance a program and make it more interesting. It also adds a "professional" touch to cassettes.

by John Victor

Music with BASIC

Two songs and a tutorial for would-be composers.

This tutorial and example program demonstrates one of the many ways of playing music using Atari BASIC. Those of you with no knowledge of music may simply type in the program and follow the instructions on the screen. If you have some knowledge of music, and you'd like further information on how this program works, read on.

The program begins with a GOTO 310. This bypasses the main program loop, subroutines, and song DATA, and brings us to our setup and screen display. Here we specify GRAPHICS 0, set the background color at random, turn off the cursor, set the left margin at 5, set the print tab width at 7, and NP=0. The numeric variable NP will be used to count the Notes Played. Lines 320-360 display our program description, author name, and user options. POKE 764,255 tells the computer to ignore the last key pressed.

The routine beginning at line 370 and ending at line 390 waits for the user to press a normal video 1, 2, or 3. Nothing will happen until one of these keys is pressed. The checking is done by PEEKing at location 764 until it contains a 31, 30, or 26. These are the internal keycodes for 1, 2, and 3. By checking the last key pressed, we eliminate the need to press the [RETURN] key.

Once we have a valid key, we position the cursor at the appropriate option number on the screen, and print that number using inverse video. The numeric variable PLAY is used to store the number of notes we are about to play.

If option 1 was selected, we do not have to use a RESTORE command since the DATA for this song preceeds any other DATA. If either of the other options has been chosen, we use the RESTORE command to point to the line number where the appropriate DATA begins.

*Jerry White lives in Levittown, New York, and is a prolific writer of BASIC and assembly language programs for the ATARI. He has many commercial products on the market, including **Poker SAM** and **Chatterbee**, from Don't Ask Software, that use the intriguing voice-synthesis-on-a-disk known as Software Automatic Mouth (or S.A.M. for short).*

SOUND & MUSIC

If the number 3 key was pressed, we also must set a flag to indicate a special condition. Since this program reruns itself when a song is over, we set the variable EXIT=1 in line 390 before GOTO 40 instruction.

Look at line 40. In English, it says that if the number of Notes Played is equal to the number of notes we wanted to PLAY, then go to line 420. Line 420 begins with "IF EXIT." This is the same as saying "IF EXIT < > 0". So IF EXIT=0, the program falls through to line 430 where we have a RUN command. If EXIT < > 0 then we reset the left margin, turn the cursor back on, tell the user that BASIC has control, and END the program.

Now that we know how the program starts and how it ends, let's see what happens in between. Let's assume you have chosen option number 3. As you pressed the number 3 key, an ASCII 26 was automatically stored in location 764. At line 390 we hit a true condition and highlight the number 3 on the screen, set PLAY=10, RESTORE 290, set EXIT=1, and GOTO 40.

The routine from line 40 through 170 is called the main program loop. We haven't played any notes yet so NP=0 and we fall through to line 50. Here we read two bytes of DATA. This will result in the variable PITCH being set to 91 and DUR being set to 12. Remember, we are reading the DATA that begins in line 290. Also in line 50 we add 1 to NP.

In line 60, we see if PITCH=0, and if it is, we GOTO our REST routine which begins at line 90. PITCH=91 so we GOTO our SOUND routine at line 130.

We will POKE the value of DUR into a countdown timer at RAM location 540. Countdown timers count backwards at the rate of 60 per second until zero is reached. In other words, when we POKE 540,DUR, since DUR=12, exactly 12/60 of a second later, the countdown timer will reach zero. In that same line we calculate the pitches we will use in SOUND registers 1 and 2, and store the value of PITCH+1 in P1 and PITCH-1 in P2.

At line 140 we turn the tables and set DUR=PEEK(540), and check to see if it is equal to zero. At this point it isn't zero yet, so we continue on to line 150 and see if DUR > 6. Six will be our maximum volume of each of three SOUND commands. In any case, we continue on to execute three SOUND commands, then go back to line 140 and check the value in our countdown timer again. We stay in this loop until we find that our countdown timer has reached zero.

When PEEK(540)=0, we GOTO line 170 where all sounds are turned off, and we can finally go back to where this whole thing started, line 40.

Remember line 40? That's the main program loop. We have played one note and have nine to go. But what if the PITCH is a 0? When we want no sound for a period of time (a REST), we enter a zero as the pitch, and use the routine beginning at line 90 to rest for the period of time specified by DUR. By the way, 60ths of a second are also known as "jiffies."

By using DUR as the volume value in the SOUND commands, we get a slight decay or decreasing volume at the end of each note. By using two additional SOUND channels, and setting their frequency levels slightly higher and lower than the desired pitch, we achieve a richer, fuller sound.

This program demonstrates only one method of playing music on your computer. BASIC can be used to play true four-part harmony and even display the lyrics of your songs on the screen at the same time. This is demonstrated by Swifty Software's **Singalong Sound & Music Tutorial** package.

Atari's **Music Composer** provides another way to play music and displays musical notes on your screen. Unfortunately, you can't put the Music Composer Cartridge and BASIC in at the same time. But I found a way around that problem too.

P.D.I.'s **Music Box** will convert your Music Composer files and play them for you using vertical blank assembler subroutine. This is done while the BASIC cartridge is installed. The best part is that once the music begins, BASIC is at your disposal. You can even write a BASIC program while the music continues to play.

The possibilities provided by your computer's audio channels are almost limitless. Take advantage of this and let us know what you come up with.

by Jerry White

```
10 REM ATARI BASIC MUSIC by Jerry White
0 5/4/82
20 GOTO 310
30 REM MAIN PROGRAM LOOP
40 IF NP=PLAY THEN 420
50 READ PITCH,DUR:NP=NP+1
60 IF PITCH=0 THEN 90
70 GOTO 130
80 REM REST TIME DELAY SUBROUTINE
90 POKE 540,DUR
100 IF PEEK(540)<>0 THEN 100
```

SOUND & MUSIC

```
110 GOTO 40
120 REM PLAY NOTE SUBROUTINE
130 POKE 540,DUR:P1-PITCH+1:P2-PITCH-1
140 DUR-PEEK(540):IF DUR=0 THEN 170
150 IF DUR>6 THEN DUR-6
160 SOUND 0,PITCH,10,DUR:SOUND 1,P1,10
,DUR:SOUND 2,P2,10,DUR:GOTO 140
170 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND
2,0,0,0:GOTO 40
180 REM DATA FOR POP GOES THE WEASEL
190 DATA 121,6,91,6,0,6,91,6,81,6,0,6,
81,6,72,6,60,6,72,6,91,6,0,6
200 DATA 121,6,91,6,0,6,91,6,81,6,0,6,
81,6,72,18,0,6,91,6,0,6
210 DATA 121,6,91,6,0,6,91,6,81,6,0,6,
81,6,72,6,60,6,72,6,91,6,0,18
220 DATA 53,12,0,12,81,12,0,6,68,6,72,
18,0,6,91,12
230 REM DATA FOR TEN LITTLE INDIANS
240 DATA 121,18,121,6,121,6,121,18,121
,6,121,6,96,18,81,6,81,6,96,6,96,6,121
,18
250 DATA 108,18,108,6,108,6,108,18,108
,6,108,6,128,18,108,6,108,6,128,6,128,
6,162,18
260 DATA 121,6,121,6,121,6,121,6,121,1
8,121,6,121,6,96,18,81,6,81,6,96,6,96,
6,121,18
270 DATA 108,18,108,6,108,6,162,6,162,
6,162,18,121,48
280 REM DATA FOR EXIT ROUTINE
290 DATA 91,12,0,6,121,6,128,6,121,6,1
08,24,121,24,0,24,96,24,91,24
300 REM SETUP/DISPLAY/OPTIONS
310 GRAPHICS 0:SETCOLOR 2,RND(0)*16,0:
POKE 752,1:POKE 82,5:POKE 201,7:NP=0
320 ? :? :? ,"ATARI BASIC MUSIC"
330 ? :? :? ," by Jerry White":? :?
340 ? :? "Type 1 for POP GOES THE WEAS
EL"
350 ? :? "Type 2 for TEN LITTLE INDIAN
S"
360 ? :? "Type 3 for PROGRAM EXIT":PO
```

```

KE 764,255
370 IF PEEK(764)-31 THEN POSITION 10,8
: ? "1" : POKE 764,255:PLAY-43:GOTO 40
380 IF PEEK(764)-30 THEN POSITION 10,1
0: ? "2" : PLAY-44:RESTORE 240:GOTO 40
390 IF PEEK(764)-26 THEN POSITION 10,1
2: ? "3" : POKE 764,255:PLAY-10:RESTORE
290:EXIT-1:GOTO 40
400 GOTO 370
410 REM EXIT/RERUN
420 IF EXIT THEN POKE 82,2:POKE 752,0:
? : ? : ? "BASIC": ? "IS" : :END
430 RUN

```

TYPO TABLE

Variable checksum - 153160			
Line num range	Code	Length	
10 - 120	JT	275	
130 - 210	NW	521	
220 - 310	GO	602	
320 - 410	OY	509	
420 - 430	RE	76	

Ultra Sound

Imagine sitting in your easy chair in front of the color television set with a stereo speaker to your right and left. The **Star Raiders** cartridge is in the computer. After selecting your destination you press [H]. A slight rumble emanates from the speakers as the engines engage. From the forward view, you see the stars moving faster and faster towards you as the sound increases to a roar. You explode into hyperwarp and the sound from the speakers rattles your chair. RED ALERT!

You reach for the joystick to direct your photons but it's too late! You receive a direct hit from Zylon fire. The room echos from the impact, the vibration causes little nick-nacks to fall from the cabinet shelves. DAMAGE CONTROL! You can hear the cries of your injured crew reverberating through your star cruiser. No, it's your neighbors yelling for you to turn down your stereo. What excitement! Maybe next time you should use the head sets.

You can make a simple, inexpensive cable that will channel audio from your ATARI 800 to your stereo speakers. This article will show you how.

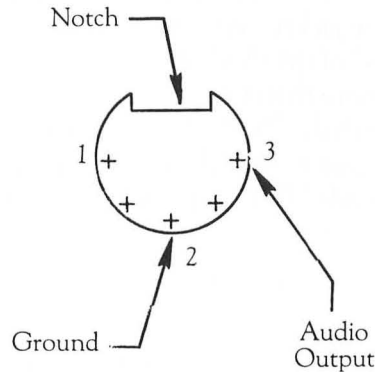
The cable will attach to most stereo systems or radios. Unfortunately, the other end will only attach to an ATARI 800 computer, where the monitor jack is external. The ATARI 400 would require disassembly, interior soldering and case modifications. There are three components that you need to buy. We have listed these items, their approximate cost, a possible distributor, and comments in Table 1.

Table 1. List of Components

Item	Distributor	Price	Comments
5-Pin Audio/Video Plug	Radio Shack (#274-003) APX (#90002;\$2.49)	\$1.49	Shielded
RCA Type Phone Plug	Radio Shack (#274-339)	\$1.39	Shielded
10 Ft. PVC Insulated Cable	Ask Local Electrician	\$4.61	0.25" O.D., shielded 2AWG 10-12 conductor
Total		\$7.49	

The 5-pin Audio/Video Plug is sometimes called a 5-Pin DIN plug. The outer jacket can be made of plastic (\$1.49) or metal (\$2.49). It contains five small pins mounted through an insulator panel and arranged in a 180-degree arc. There is a small notch at the top for alignment purposes (Figure 1.)

Figure 1.
5-Pin Audio/Video plug configuration
(outer facing side)



One side of the insulator panel usually has small numbers printed on the board. These numbers correspond with the numbers in Figure 1. For our purposes, it is important to know that the ATARI 800 uses pin 3 as the audio output and pin 2 as the ground. The RCA-type phono plug has an outer jacket of metal. The inner workings contain one large pin held in place by insulation. These units are usually sold in pairs since the typical use is for a two channel stereo input. The large pin is the audio input and the outer jacket is the ground.

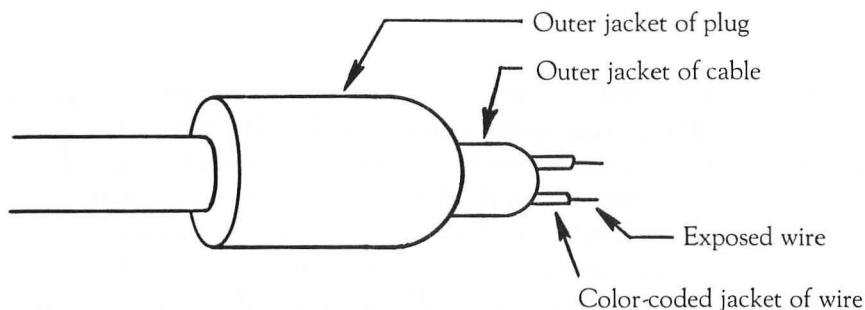
PVC insulated cable is sometimes called telephone cable. There are hundreds of different types of cables to choose from. We recommend a tinned copper, PVC insulated, conductor cable with 22-24 AWG stranded drain wire. Wire gauges much larger than 22 (i.e. 18, 16, 14 . . .) are very stiff and difficult to work with. Stranded wire should be color coded. The cable should be jacketed in a chrome PVC with an outer dimension (O.D.) of 0.25 inches to ensure a snug fit with our plugs. If you choose a smaller cable (e.g. speaker wire), you run the risk of pulling the wires out of the plugs or crimping the cable when you move the computer. If you have your computer in an area of severe electrical interference, we suggest that you purchase a cable with aluminum-

SOUND & MUSIC

polyester shielding. The minimum length for your cable should be 10 feet to allow for some flexibility in where you can place your components.

Next, gather the necessary tools for soldering. You will need a pencil-tip soldering iron with a heating element of 25 to 35 watts. The best solder for this application is an alloy of 40 percent tin and 60 percent lead with a resin flux core. This is sometimes referred to as television or electrical repair solder. In addition, you will need a razor, a needle-nose pliers, a wire cutter, a clampable heat sink and a clean, well-lighted work area. Remove the outer jacket from each plug and slide the jackets onto opposite ends of the cable. With a razor, carefully strip away $\frac{3}{4}$ " of the PVC cable cover from each of the cable ends. If your cable contains more than two color-coded wires, snip off the extra ones to make them flush with the PVC cable cover. Compare the ends of the cable side-by-side to make certain that the color codes are an exact match. Strip away $\frac{1}{4}$ " of the color-coded PVC from each wire (Figure 2). You are now ready to solder.

Figure 2. Cable Assembly.



Hold the inner workings of the plugs with a pliers and attach the heat sinks to the appropriate areas. Solder ground to ground and audio to audio. If the insulators begin to melt, discontinue soldering and attempt to re-straighten the pins. Once soldered, reassemble the plugs. Firmly insert the 5-pin plug into the ATARI 800 monitor jack and the RCA-type phone plug into the accessory or tape (in) jack on the back of your stereo. Boot something musical onto your 800, turn down the volume on your TV and switch your stereo to accessory or tape. If you have a stereo/mono switch, place the switch in mono position. Otherwise, the sound will only come through one speaker. Very slowly, turn up the volume. You should hear perfectly clean music. If you

hear a hum, you have a poor connection. Check that your solder has not bridged across the insulator.

For the adventurous experimenter, you could also build a frequency separator making this a pseudostereo rather than a monotone cable. Use a high/low frequency shunt and patch the high frequency to one channel and the low frequency to the other. I'll leave the design up to your imagination. In addition, the strength of the audio signal could be monitored and used to control some other devices. For example, you could place a fan on the top of your television and an inclinor platform beneath your chair. As you enter hyperwarp, the fan would blow faster and faster, and you would gently sink back into your seat. The seat would jolt whenever you were hit by enemy fire and it would pulse during engine damage. An affixed joystick on the arm of your chair would allow you to bank to the right or left, climb or dive, by shifting your weight. The ultimate in home aviation simulators!

by Thomas Krischan

done

'Tari Talkers

Voice Synthesizers for the ATARI 400 & 800

Confidently, I slipped into the Commander's chair. I pushed [START] and a vision of deep space, scattered with stars, flashed on the view-screen. My superior's deep voice washed through the room, "Welcome aboard, Commander. Your mission . . ." When he finished, I typed [G] for the Galaxy Map. Lt. Longri's tenor explained that a Zylon full battle patrol had entered sector A4. That fit my strategy! I punched the controls, and the ship leaped into hyperspace. Upon reentry, Captain Sumtra's dusky voice warned, "Zylon sector, sir." I punched for shields. "Shields," she replied.

The screen became a blur of ships, photon torpedoes, explosions. Lt.

SOUND & MUSIC

Longri calmly tracked our kills, while Captain Sumtra repeated every order smoothly. Suddenly, Damage Control's clipped, high-pitched voice screamed through the flight deck, "Shields lost!!" A Zylon fired at us. I punched hyperspace. The screen dissolved in a flash of white. Against a dark screen, the Federation's emblem appeared, the commander spoke quietly, "Posthumous . . . rank awarded . . . Garbage Scow Captain."

Now, two machines make it easy to add *voices* to your Atari programs. The *Type'n Talk* (TNT) from Votrax and *Echo-GP* from Street Electronic *synthesize*, or create speech, from written English almost as easily as characters are printed on your screen.

Applications far beyond obvious game enhancements abound. Imagine pronouncing dictionaries or spelling programs more flexible than *Speak-N-Spell*. Either system could be set up easily to speak for a speech-impaired person, or to voice, letter-for-letter, or word-for-word, all data entered by, or sent to a blind operator. My most successful program, so far, displays a four-color chart and explains it orally, with no text distracting from the visual. At least half the fun is watching a new user's face as the computer says, "Hello Mary!"

Both TNT and Echo are efficient, small, speak an unlimited vocabulary (anything you can print), take almost no memory, and cost less than \$500. Both speak with a distinct "computer voice" which the uninitiated can understand, with some concentration, but which quickly becomes "natural." It's a bit like getting used to that uncle with the funny accent.

Both units require an Atari 850 Interface and a cable. The cables are available from the manufacturers for an extra \$30, or can be made as follows. Order the 9-pin DB connector from Apex (APX-90006 \$5.50), and a 25-pin DB male connector from Radio Shack (\$3.50) or any electronics house. Buy a few feet of any 6-conductor (or more) conductor cable (Beldon #9421 is often used). Connect these according to the chart (Fig. 1), and you've saved \$20. The TNT requires an 8-ohm speaker (\$5-\$10) and a mini-phone jack. The Echo has a built-in speaker but you can add an external speaker for fidelity and volume. With an external speaker, the Echo puts out considerably more sound than the TNT.

Getting started is simple. Set the switch to 300 baud, plug the cable into serial port 1 or 2 of the 850, boot the system, and type the following statement [The "n"s represent the IOCB (see BASIC manual p. 26); the "x"s are the port number, 1 or 2]:

```
OPEN #n,8,0,"Rx":XIO34,#n,48,0,"Rx":XIO36,#n,12,0,"Rx"
```

After that, merely issue PRINT #n commands to make the units speak what you wish. A program to input a string from the keyboard and speak it takes no more than three lines. Both units include clear, usable manuals with lots of examples.

Although the units are similar, there are clear differences. The most important criterion to me was intelligibility. No speech synthesis device is worthwhile if you can't understand it. A frequent user will get accustomed to either of these. To check for immediate clarity, I took both units to the Lawrence Livermore Lab Science Fair and asked visitors to listen to a list of 20 words, spelled as recommended by both companies, spoken alternatively on one, then the other, unit. Since I have used the Votrax for six months and find it quite clear, I expected it to win this test. However, nearly all people listening to the two for the first time found the Echo clearly superior. The Echo seemed to excel with words beginning with "hard sounds" such as T, P, B.

Intelligibility aside, I examined reactions to the Echo's many unique features. Both units sound like computers, not people. But as one girl said, the Echo sounds like a "he," the TNT like an "it." The Echo software-switchable pitches (at normal speed) were a popular feature. The lower voices were easier for most people to understand and several suggested creating dialogues between different personalities, each with a different voice.

The Echo's "inflection" feature raises the tone of the last syllable before a question mark and lowers it before a period. Although only about half of the new listeners could describe this effect, it may have contributed to the Echo's superior intelligibility.

Spoken punctuation is another Echo plus. Normally, it speaks the punctuation commonly spoken (\$, #, =). But, at the drop of a software instruction, most punctuation (comma, period, semi-colon, parenthesis, etc.) or *all* (including spaces, returns, etc.) are spoken. This could be a real boon to the sight-impaired. Both units will spell capitalized acronyms. The Echo, however, has a letter mode which will spell out all words—very useful for a spelling program or a blind operator faced with an unintelligible word.

Both systems allow the user to create phoneme strings. This results in phrases with exceptional clarity. Frankly, since I get acceptable results with English, phoneme coding words seem like too much work. For instance, "catalogue" is coded "KA3DIL*1G"! If you decide to phoneme code, a TNT

SOUND & MUSIC

software option will send you a phoneme string as it translates from the English. You then polish it up for final phoneme codes.

The TNT's enclosure has some problems. The on/off switch is on the back panel, and worse, the unit has no "on" light. Many's the time the kids have left the TNT on all night! Echo has a light, and the switch is right up front.

So there's the balance. Both do a good job.

Intelligibility, features and price make the Echo distinctly superior.

by Ken Harms

ECHO-GP (Serial)

Street Electronics Corp.
1140 Mark Ave.
Carpinteria, CA 93013
(805) 684-4593
List Price—\$199.95

TYPE 'N TALK

Votrax
500 Stephenson Highway
Troy, MI 48084
800-521-1350
List Price—\$249 + speaker

Figure 1

WIRING CHART

Atari 9 pin DB Male	25 Pin DB Male	
	TNT	ECHO
1	—	—
2	20/8	20/8
3	3	3
4	2	2
5	7	7
6	20/8	20/8
7	4	5
8	5	4
9	—	—

Editor's Note:

Since this article was written, Votrax has released another voice synthesizer, the *Personal Speech System*. The new product lists for \$395 and offers several improvements over Votrax's early Type 'n Talk. Personal Speech System has

a 16K algorithm (versus 4K algorithm in TNT) which leads to 95 percent accuracy in pronunciation. It can produce music and sound effects and it has a real-time clock. In addition, the speech rate amplitude and inflection are user-programmable. And this time, the speaker is inside the unit.

Communications



done

Modems

Did you ever think about what a computer really is? Take the ATARI for example. With 48K bytes of memory it can store about the same amount of text as a 15-page document. A diskette can store about 40 more pages. You can think of your display screen as a “window” through which you can see this information, about one-quarter page at a time.

What’s the point? Well, the time is here when, for the price of a cheap suit, you can give your computer access to millions of pages of memory, instead of just 40 or so.

We are talking about the modem. Let’s de-mystify the modem, explore what it is, what it does, and then look at a few modems available for the ATARI.

Terminology

Here are some terms you will find in the world of modems:

- **MODEM** — The word derives from “modulate-demodulate.” A modem is a hardware device that translates an incoming sound signal (frequency) into a binary code that your computer will understand (computers do not understand sounds). The modem also works the other way around. It will translate an outgoing, computer-generated binary code into frequencies that can be transmitted over circuits used by the telephone company.
- **BAUD** — This term describes the rate at which data is transmitted. The telephone company has established 300 baud as a standard rate of data transmission for phone lines.

This equals 30 characters per second or approximately 350 words per minute. This is about as fast as most people can read. There is also a 1200 baud

Jon Loveless is a resident of the San Francisco Bay Area, and a Vice President of Marketing for Synapse Software. An ATARI hobbyist from the outset, he has a special interest in hardware and peripherals, especially printers and modems. He was one of the early editors of ANTIC, and a co-founder of ABACUS, but the demands of Synapse’s growth have limited his recent contributions to ANTIC.

COMMUNICATION

standard rate available on the phone system at a premium price. Watch for this price to fall over the next few years.

- **ACOUSTIC-COUPLED** — This describes the type of modem that transmits and receives directly through the standard telephone receiver. This kind of modem has two foam “cups” into which the earpiece and the mouthpiece of the receiver are placed. The cups channel the sound, audible as a high-pitched whine, to and from the phone system, and muffle extraneous noise.
- **DIRECT-CONNECT** — This is the newer breed of modem. It can connect directly to your telephone wall jack or plug into your telephone with a “Y” adapter. Outside sound interference and clumsy manipulation of the receiver are eliminated.
- **ANSWER-ORIGINATE** — These terms describe which modem is calling and which modem is answering. There must be a modem on each end, but they do not have to be the same brand. Either modem can do either job, but not at the same time.

Modems

By now, you may be interested in buying a modem, and wondering what features are important. Here are some things you should be aware of.

Acoustically-coupled modems, the “ear-muff” type, were the first on the market and are still the cheapest. They have definite drawbacks. Stray sounds in the vicinity of the modem can and do leak past the muffs and can affect data transmission. Also, using the acoustical modem is awkward, since the correct end of the phone receiver must be inserted in the correct end of the modem. This sounds minor, but the error is easily and frequently made.

Still, acoustic modems do work, are inexpensive and may meet your needs.

Prices for direct-connect modems seem to be dropping, and the higher degree of reliability for them makes it difficult to recommend anything else. If you think you would be even a semi-serious “on-liner,” you should think in terms of a direct-connect, plug-in modem. Your data will be cleaner, and the benefits of uploading and downloading data over networks, with the new information utilities, or with other individuals, will repay the extra investment.

Some modems have *status indicators*. When the modem is in use it is often important to know what the status of your connection is. Is the modem “ready?” With a direct-connect modem, is the simulated “receiver” on the

hook or off the hook? Has there been an accidental disconnect? Is the other end answering? The more information provided by the modem's status indicators, the better.

Some modems have *autodial* / *autoanswer*. You can dial a phone number from your ATARI keyboard! Admittedly, this is a luxury, but if you use a modem a lot, it is a nice feature to have. Autodial allows you to store telephone numbers in your software program, and have the modem do your dialing for you. This eliminates the need for a telephone near the computer, provided you have a phone cable long enough to reach your telephone jack.

Autoanswer is only needed for such serious data communications as operating a bulletin board service, or otherwise responding to the incoming call of another computer. Think of the possibilities, though! You can call your own computer from any remote terminal, or even from a phone booth with one of the miniature modem-terminals recently announced.

Other features to look for include:

- compatibility with the Bell 103 Standard;
- full-duplex and half-duplex (in case you only want to send or receive);
- 300 baud rate, 1200 baud optional;
- RS-232 plug compatibility for Atari 850 interface connection;
- proper connecting cables!

Cables

A word about cables is in order. Modems must be connected to your other equipment, and to the telephone line. You would think that an expensive item like a modem would come with the appropriate cables. Not always so, and the price difference between a more expensive unit with cables and a less expensive one without may be misleading (some cables cost \$50!). Also, some modems are designed to hook up more simply, eliminating some cable requirements. Before you buy, determine your complete system requirement, and compare the price for all pieces. You will want to include software costs, too.

All modems, once the proper connections have been made, will perform their primary function of data communications, so the bottom line in any decision should be: quality, price, and extra features. You will probably find your use of a modem will be greater than you now expect, so be open to the more capable units.

Any modem can work with the ATARI, if properly connected, but some

COMMUNICATION

have been built specifically with the ATARI in mind. We will discuss the principal ones here.

Atari 830 Modem (\$199.95)

The Atari Modem, sold by Atari, is a "Novation 'CAT'" modem in Atari dress. It is a standard acoustically-coupled modem with only very basic features. It is fine for a beginning user, or someone with limited needs. Since it is marketed as an Atari product, it comes with all required cables. It also needs the Atari 850 Interface, which some modems do not, so if you don't have the Interface you should seriously consider the Microconnection modem (see below), or others that bypass the Interface.

The Atari 830 is a plug-in-and-go product with good documentation. You will need software with this, as with all modems, and might well consider Atari's TeleLink cartridge (\$30) for a nice, modular system. Caution! TeleLink is a very limited program, and will not allow copying to disk. It will drive the Atari printer, but printing "on-line" is expensive. The major drawbacks with the Atari 830 are that it is acoustic, and has limited features.

An alternative buy would be the "Novation 'CAT'" if you can find cables. Two other "Novation" modems are compatible with the ATARI. One is the D-CAT, a basic direct-connect model, and the AUTO-CAT, that has the autodial feature mentioned earlier. Although not described in depth here, they are both good products that should be considered as in the running.

Microconnection-A (\$199 to \$328)

This direct-connect modem is made by the Microperipheral Corp. and comes in four versions all designed for the ATARI. This selection is very attractive to the prospective buyer.

For example, there is a bus-decoding version (\$249) that allows connection without using the Atari 850 Interface. This modem can be used with as small a system as the Atari 400 and the 410 Program recorder. This model has a DB-25 socket that allows connection of the Atari printer, again without Interface. This makes the Microconnection a good candidate for a small basic system. For \$30 more this model comes with autodial.

There is a plain version (\$199) that does require the Interface, and for an additional \$40 you can get the autodial and autoanswer features.

Caution! Microconnection's autodial uses pulse dialing (not touch tone) which cannot be used with the MCI or SPRINT long distance phone services, but you can manually dial SPRINT or MCI with this modem. If you are a heavy user of these long distance services this could be an important limitation.

Microperipheral has done a commendable job of supporting the ATARI, and their own software enhances the capability of their modem dramatically. The top of the line software, called TSMART (\$79.95) incorporates autodial as well as message preparation and storage features that reduce expensive "on-line" connection time. You will appreciate this after you see your first phone bill after buying a modem.

The Microconnection is relatively simple to connect and use. It comes with extensive, if dense, documentation which includes a listing of free bulletin board services, by area code (a nice touch!). Microperipheral Corp. maintains a user service accessible through CompuServe, over which you can get updates of their software. Now that's service!

Smartmodem (\$279)

This is a direct-connect modem by Hayes Microcomputer, Inc. Although it does not come as a model specifically for the ATARI, you can purchase a cable to connect it to the Atari 850 Interface (required). The fact that this modem does not come with a cable is a serious drawback in a product that costs so much. This is not a criticism of Hayes alone, as you will discover when you buy your first non-Atari printer, or other peripheral device.

Assuming you buy the "Hayes Stack," as it is also known, and are able to get or make a cable, you will have the most flexible modem in the price range. This is truly a "smart" modem. The heart of the device is a 280 microprocessor with a 2K byte control program built in. The only switch is an ON/OFF toggle! Everything else is program controlled, or preset by you, utilizing the configuration panel under the front cover.

Here are some of the features of the Smartmodem:

- either touch-tone or pulse dialing at any time;
- audio monitor allows you to hear what your phone line is doing (a real help when the receiving party is busy);
- storage of the last number dialed;
- automatic redial (helpful for disconnects, busy signals, etc.);
- seven LED status indicators on the front panel (impresses visitors);

COMMUNICATION

- complex dialing sequencing (e.g., dial number, wait for tone, send ID, dial another number, as required for MCI and SPRINT);
- programmable in any computer language and compatible with most data communication software.

The list goes on, but the point is made. The Hayes Smartmodem is very versatile, but suffers due to a lack of direct applicability to the ATARI. With the appropriate cable (I made my own) and almost any good terminal software, this modem is the most flexible.

There are other usable modems around, though not specifically for the ATARI. They will work fine with the proper cable, and some of the good software.

If you are not in the market for a modem now, I guarantee that you will be some day. It might be a good idea to wait, if you have no immediate urge to link up with the rest of the tribe. Prices keep coming down, and good gear gains reputation as satisfied users swap notes.

Keep your eyes open for new, low cost entrants to this field. For example, I noticed (but have not used) the Signalman MK-1 from Anchor Automation at an unbelievable price of \$99, including RS-232 connector cable. This direct-connect modem could be the forerunner of a price revolution.

Meanwhile, the modems we have discussed are definitely state-of-the-art products and can be expected to provide good service for a long while.

by Jon Loveless

ATARI BULLETIN BOARDS

State	Name	Phone Number	Type
CA	LAACE	213-988-8373	AMIS
CA	GFX	408-253-5216	AMIS
CA	IBBS	408-298-6930	AMIS
CA	ABACUS	415-587-8062	AMIS
CA	ACCESS	916-363-3304	AMIS
DC	WASHINGTON	202-276-8342	ARMU
GA	ROD R.	404-252-9438	ATAB
IL	WIZ-BANG	312-925-2929	AMIS
MA	MACRO EXCH.	617-667-7388	AMIS
MI	M.A.C.E. W.	313-274-3940	AMIS
MI	M.A.C.E.	313-544-0885	AMIS
MI	A.R.C.A.D.E.	313-978-8087	AMIS
MI	C.H.A.O.S.	517-373-6788	?
MI	G.R.A.S.S.	616-241-1971	AMIS
MO	A.U.R.A.	314-928-0598	AMIS

COMMUNICATIONS SOFTWARE

NY	SPIDER WEB	212-241-8965	AMIS
OH	FLAG CITY	419-423-0206	AMIS
OR	A.C.E.	503-343-4352	ARMU
TX	ARMADILLO	512-837-2003	AMIS
TX	ACUGD	817-498-1751	ARMU

These Bulletin Board numbers were verified as correct and available as of October, 1983

done Communications Software

After you purchase a modem and install it, you will soon be aware that there is one more important purchase you need to make—software. Without a good flexible program your modem will be useless. In this article we will introduce you to six different programs designed to be used with modems. These programs vary in ease of use and capability. We will show you the trade-offs and introduce some new vocabulary which will make our discussion more understandable.

Download—this refers to the physical reception of data. It can be in the form of a complete program that you are receiving from another computer or simply data that you are saving from CompuServe, or The SOURCE. The key word is “save.” So, download means to receive and save data or programs.

Upload—this is just the opposite of download. Upload refers to the act of sending a specific program or text to another computer via the trusty old modem.

Host Computer—this is the computer that your ATARI will talk to, assuming that you make the call. If you use the “auto-answer” capability of your modem, your computer becomes the host.

Translation—refers to the degree of character code incompatibility the specific software will compensate for. This inconsistency is often a problem with those characters where no real standard has been acknowledged, like special control characters. Translation also refers to a program’s ability to convert from one character encoding scheme to another. ASCII to EBCDIC for example.

Terminal emulator—refers to a program’s capacity to make your ATARI respond as if it were some other type of terminal. VT100 or ADM-3A

COMMUNICATION

come to mind as widely used terminals. This is usually accomplished by redefining key and control code functions.

Buffer—is often used to refer to a reserved portion of computer memory. This reserved area is used by terminal software to store programs which have been downloaded. These programs can be saved to disk later off-line. Programs which force you to save to disk on-line cost more for connect time because the disk is slower.

There are many other terms you will come across, but these few will give you a start. Now, let's see what you need in the way of software. It depends largely on your application. If you only want to "look" at the data available from some other computer system, your needs are simple. If you want to save the data, your needs are more complicated, and if you want to send and receive programs, communicate with a computer at your office, or perform other such sophisticated operations, you need a fancier program yet.

You will find a need for several different types of programs as you proceed, so let's sort out a few programs to see what they do, and then refer to the table on page 78 for a quick reference comparison.

TeleLink

This program is available on cartridge from Atari. It is an excellent beginning for the new modem user and it comes with a free subscription to CompuServe. This alone makes it worth the money. TeleLink's beauty is its simplicity. Plug it into the left slot of your ATARI and "log-on" as they say. The major drawbacks are its inability to save incoming data to disk or cassette or to upload and download programs. TeleLink can save data to your printer, but this can be costly in terms of connection costs. This is not a bad way to introduce yourself to telecomputing, but you'll end up wanting more features.

DataLink

Swiftly Software's program is probably the best all-around choice you can make as either a new or intermediate user of the modem. It is simple and friendly, yet very powerful. It will fulfill most of your needs including uploading/downloading, text capture, save to disk or printer, and screen review of data in memory. It allows you to prepare text before you make the phone connection, and save text after you hang up, both important features

when concerned about your phone costs. Above all else, DataLink is very easy to use. Documentation is pretty scant (six pages), which can be a handicap to the uninitiated, but is also a reflection of how easy this program is to use.

Download

This software by Computer Age is a great program, but has received little promotion or publicity for some reason. It is written in BASIC and machine language (where needed for speed) and offers a benefit in that it can be modified by the user. I particularly like this feature with the Hayes Smartmodem since it allows you to add a phone number menu and make full use of the power of auto-dial. In addition, it has two menus, one for parameters and another for memory management. The [OPTION] button accesses the main menu and that allows you to go to memory management as one of the options. It is not as easy to use as Datalink, but is more flexible.

T.H.E.

Binary Computer Software presents this recent addition to the communications market. It is possibly the most complex modem program available for the ATARI. As with any powerful program, this one requires study and practice to use effectively. The documentation is well done and is readily understood by the first time user. There are many system configurations possible using T.H.E. With all the bulletin boards being made available, each with different requirements, this flexibility is T.H.E.'s most important feature. This is the only package that will translate ASCII to EBCDIC. This feature would only be needed when communicating with an IBM system.

Chameleon

From APX (Atari Program Exchange) comes a powerful machine-language program that lets you tailor your ATARI to a wide variety of configurations that will satisfy almost any host computer requirement. The documentation is good, but the program must be used extensively in order to feel comfortable with the many commands and options. One of the unique features is the 80-column screen emulator. Using the ATARI scrolling capability you can make it think it is an 80-column computer rather than 40. I have found little practical use for this feature yet, but it sure looks nice. I

COMMUNICATION

NAME	TELNK	DATLK	DWNLD	CHAMN	TSMRT	T.H.E.
MANUFACTURER	(1)	(2)	(3)	(4)	(5)	(6)
MEDIA (c=cass/d=disk)	cartg	d	c/d	c/d	c/d	c/d
LEVEL OF FLEXIBILITY	low	mod	mod	high	high	high
DOCUMENTATION	good	fair	fair	excl	excl	good
MEMORY REQUIREMENT	cartg	24K	24K	24K	24K	24K

FEATURES

TRANSMISSION

.upload programs	no	yes	yes	yes	yes	yes
.download programs	no	yes	yes	yes	yes	yes
.download text	yes	yes	yes	yes	yes	yes
.full duplex	yes	yes	yes	yes	yes	yes
.half duplex	yes	yes	yes	yes	yes	yes
.terminal type	1	1	(7)	4	(7)	(7)
.BAUD rates	300	300	300	(8)	(9)	(10)
.translation choice	yes	yes	yes	yes	yes	yes

SEND DATA

.off-line prepare	no	yes	yes	yes	yes	yes
.store ID codes	no	no	no	yes	yes	yes
.preload programs	no	yes	yes	yes	yes	yes

RECEIVE DATA

.on-line save	yes	yes	yes	yes	yes	yes
.to printer	yes	yes	yes	yes	yes	yes
.to disk	no	yes	yes	yes	yes	yes
.to cassette	no	no	yes	yes	yes	yes
.off-line save	no	yes	yes	no	yes	yes
.to printer	no	yes	yes	no	yes	yes
.to disk	no	yes	yes	no	yes	yes
.to cassette	no	no	yes	no	yes	yes
.parity options	no	no	yes	yes	yes	yes
.memory toggle	no	no	yes	no	yes	yes
.memory management	no	yes	yes	yes	yes	yes

MISCELLANEOUS

.user modified	no	no	yes	(11)	yes	no
.phone # storage	no	no	no	no	yes	no
.format screen	mrgns	no	no	40/80	mrgns	mrgns
.redefine keys	no	no	lmted	yes	yes	yes

(1) TELELINK
ATARI, Inc.
1272 Borregas Ave.
Sunnyvale, CA 94086
\$29.95

(2) DATALINK
Swift Software
64 Broadhollow Road
Melville, NY 11747
\$39.95

(3) DOWNLOADER
Computer Age
Silver Spring, MD
\$24.95

(4) CHAMELEON
APX (Atari Program Exchange)
P.O. Box 427
155 Moffett Park Drive
Sunnyvale, CA 94086
\$17.95

(5) T-SMART
Microperipheral Inc.
2643A-151st Pl. N.E.
Redmond, WA 98052
\$79.95

- | | |
|---|---|
| <p>(6) T.H.E.
BiNARY Computers
3237 Woodward Ave.
Berkley, MI 48072
\$49.95</p> <p>(7) Terminal type may be defined largely through flexible parameter definition, if not by name.</p> <p>(8) Widest choice from 48 to 9600 BAUD.</p> | <p>(9) Choice of 300 or 600 BAUD.</p> <p>(10) Zero to 9600 BAUD.</p> <p>(11) Source code is provided for the adventurous assembly language programmer, but is sparsely commented.</p> |
|---|---|

wouldn't recommend this program to beginning users of modem software unless they are ready to roll up their sleeves and work with it. For the more sophisticated user this is a powerful tool. One caveat with this program is that it transfers files more slowly because it writes to disk rather than saving to a memory buffer.

T-Smart

Microperipheral Corporation offers a powerful and flexible program written expressly for their Microconnection modem. Its power rests partly in the fact that it was written with a particular modem in mind, and partly in the fact that it is reasonably simple to use for all the flexibility it has. It is completely menu driven, but a nice feature is the option to override the menu as you become familiar with the commands. It incorporates real autodial so that you can include your own list of phone numbers right in the program. Much of the program is written in BASIC allowing you to tailor it to your own needs. Finally, as with the Microconnection itself, it is well supported through a simple contact on CompuServe. I understand this even includes updates as they become available.

Take Your Pick

So you now have a bird's eye view of six pieces of software for your new modem. If you are like most users, you will find your needs satisfied by a simple program, occasionally needing more power or flexibility. For example, I still use TeleLink because of its simplicity. I check the electronic mail service (EMAIL) of CompuServe with TeleLink and nothing could be easier. I use Datalink often because it is simple yet quite powerful. T.H.E. is a newcomer

COMMUNICATION

and yet I already am finding some of its features and power attractive. Finally, if I owned a Microconnection, I would certainly use T-SMART because of the powerful design interaction between software and hardware, a well-planned pair.

For a first purchase I would be hard pressed not to recommend the Datalink program because of its nice blend of power and simplicity. It will satisfy the majority of your needs, and will allow access to most common services such as CompuServe, The SOURCE, and nearly all of the bulletin boards available. The greater parameter flexibility of some of the other programs is necessary for sophisticated communications between your ATARI and non-ATARI equipment, especially if you plan to do a fair amount of program exchange.

Our goal has been to shed light on the sometimes confusing topic of data communications. We would suggest that whatever hardware and software you decide to purchase, it be checked for compatibility. A good package will make your introduction to telecommunications easy and enjoyable. It really is a thrill when you successfully transfer your first program to a friend across town.

by Jon Loveless

done

Dialing For Data

Electronic information utilities are making a big splash on the American scene as more and more people buy computers. Most computers, including the ATARIs, can "communicate" with each other using these utilities. Communication between computers has brought about an entirely new kind of business.

What's an information utility? Essentially, it is an electronic network that

This information updated as of August, 1983.

Robert DeWitt is managing editor of ANTIC Magazine

sells computerized information and services to connected customers, just like the water utility sells water. At present this is done over telephone circuits, and soon it will also be done by TV cable.

Two such utilities are prominent now; CompuServe and The SOURCE. The American Telephone Company (Ma Bell) is expected to enter this field soon, and will certainly be a strong contender. There are other services around that connect computers but they are usually smaller, more specific, and more expensive. DIALOG, a scientific data-base, is an example.

General

CompuServe dates back to 1969 as a data-base service company for other big companies and government. It is owned by H&R Block, and is located in Columbus, Ohio. It uses DEC-10 mainframe computers and has about 63,000 subscribers. CompuServe publishes a monthly newsletter "Update," and a monthly magazine "Today." These are free to subscribers.

The SOURCE began in 1979 specifically as a consumer-oriented information utility, although it does serve businesses too. It was bought by Reader's Digest in 1980, and is located in McLean, Virginia. It uses six PRIME-750 mainframe computers and has 38,000 subscribers. The SOURCE publishes a bimonthly magazine "Sourceworld" that is free to subscribers.

Both utilities transmit at 300 baud or 1200 baud, and charge more for the higher rate. Since 300 baud is about 300 words per minute, it a comfortable rate for a human operator. This article refers to 300 baud service only.

Time Availability

Both utilities are available full time, but at higher cost during business hours (see below). The SOURCE officially closes daily from 4 A.M. to 6 A.M. EST for system work. This is 1 A.M. to 3 A.M. PST (western nightowls take note).

CompuServe claims to be up "99.4%" of the time. Both begin their evening rates at 6 P.M. (local time,), but The SOURCE initiates a still lower rate at midnight.

Access

To get connected with either of these utilities, the user calls a telephone number, gives an I.D. number and password, and is "logged on." Herein lies a

COMMUNICATION

significant difference. The user calls the telephone number at his own expense. If the closest access number is long distance, the user pays the charge. The SOURCE is clearly superior here, providing a local (no charge) number in about 350 major areas, including Alaska, Hawaii, Puerto Rico, and Canada.

CompuServe provides free local numbers in 200 cities, and a TYMNET or TeleNet number in about 200 more cities, for which the user pays an additional \$2.00 per hour. City size is no guarantee of having a local CompuServe number.

Cost

The SOURCE has a \$100 registration fee that dissuades many people. CompuServe charges \$20 for a "dumb" hookup, \$30 for a "smart" one that includes software, or \$40 for a smart one including five on-line hours. Most ATARI owners will want the dumb package and get their software elsewhere.

All time charges are figured to the nearest minute, local time. Regular time on The SOURCE is from 6 P.M. to 7 A.M. and all day on weekends and holidays. This is billed at \$7.75 per hour. CompuServe charges \$5.00 per hour from 6 P.M. to 5 A.M. weekdays and all day on weekends and holidays.

Rates during business hours for The SOURCE are \$18 per hour, and for CompuServe \$22.50 per hour. Anyone interested in CompuServe should add any long distance or TYMNET charges that could affect comparison.

The SOURCE has a few services that cost more, for the time they are used; commodity prices and stock analysis, Compu-U-Store ordering, and journal abstracts. These are designated as SOURCE*PLUS and cost \$15 per hour in regular time, or \$10 per hour after midnight. CompuServe has a few surcharges in the stock market service, and charges a flat fee for Comp-U-Store. CompuServe also adds \$2.00 to your monthly bill if you do *not* use MasterCard or VISA for payments.

News

Both utilities have news services. CompuServe is more extensive, offering Associated Press, Canadian Press, and two complete American newspapers (the Washington Post, and The St. Louis Post-Dispatch). The SOURCE offers

United Press International and selected N.Y. Times stories and features. Indexing by key word and key-word search of news is available with The SOURCE, but not with CompuServe.

Another difference is that CompuServe purges its news daily and has no historical news files. The SOURCE purges weekly (Friday A.M. maintenance) so it has a whole week's news available on Thursday night. This could be an important difference for researchers or people with special news interests.

Shopping

Both utilities offer shopping by Comp-U-Store. This allows on-line review of about 30,000 items, plus electronic ordering for delivery to the home. The SOURCE offers "ordering" mode at SOURCE*PLUS rates, and CompuServe charges an extra membership fee of \$18 per year to order. "Browsing" can be done on either utility at regular rates. Comp-U-Store itself is offered directly at \$25 a year plus 25 cents per minute, so getting it as a part of a broader utility service does represent a value.

The SOURCE offers a BARTER program for worldwide exchange of goods and services, and both utilities have bulletin boards in which users may advertise. CompuServe includes classified advertising from the newspapers it carries, but this is an expensive way to read classified ads.

On-Line Conversation

The most popular feature of either of these utilities is the on-line communication between and among users. CompuServe's version is called "CB Simulator," and it's a conversational free-for-all, with participants identified by fictitious "handles." The samples I've seen were bawdy and inane. If one perseveres, it is possible to find a party with mutual interests, and arrange a private talk. Groups can even conference on-line, and the exchange can be encrypted if all users have an encryption password.

The SOURCE offers CHAT, limited to two users who must be on-line and agree to the exchange, which is private. If you don't know anyone to chat with, you can query any user whose I.D. number shows up on the "online directory."

COMMUNICATION

***E*Mail**

E~~M~~ail is sure to become a new English word. It means electronic mail, and we will all be using it soon. Even now, users of these utilities enjoy the advantage of instantaneous message exchange, which can be printed or copied with the right equipment and software.

With either utility, messages can be E~~M~~ailed to any other user of that service. The user's I.D. is his address, and the message will wait for him until it is picked up.

The SOURCE allows for an unlimited number of letters to collect until read. With CompuServe, your mailbox is "full" with 10 letters, and no more can be received until the mailbox is relieved of at least one letter.

The SOURCE has an extra E~~M~~ail feature called Voicegram. It allows the member to call into the tollfree Customer Service number and dictate an E~~M~~ail letter to any user for a \$1.25 extra fee.

CompuServe allows its members to use its text editor program, FILGE, on E~~M~~ail.

***C*ustomer Service**

Both utilities maintain tollfree Customer Service numbers available 24 hours a day, and both were helpful and courteous when called. Both answer automatically, and put you on hold "airline fashion" if necessary. Waiting time was three minutes, at most.

The numbers are: The SOURCE (800) 336-3366; CompuServe (800) 848-8199.

***S*tock Market Information**

Both utilities provide stock market quotations, news, and analyses.

CompuServe calls its service MicroQuote, and charges five cents per quote. There is a \$1 minimum fee each time that data-base is used.

The SOURCE calls its stock quotation service UNISTOX, and offers it at no extra charge. Both services cover about 30,000 issues on the major exchanges. The SOURCE also covers trading in about 20 commodities. These quotations are charged at SOURCE*PLUS rates.

Bulletin Boards

Users can post their own notices on the bulletin boards of their respective utilities.

The SOURCE calls theirs POST. It is categorized by subject or interest. For example, there is an ATARI section in POST where I found about twelve notices.

The CompuServe board is called BULLET. There are three separate sections: Sale, Wanted, and Notices. Each section has a few hundred postings at a time. Each is key-worded and numbered. To find ATARI notices you must scan all three lists.

Programming Aids

Each of these utilities provides services for computer programmers. You can, in fact, program on-line and store data files with the utility.

CompuServe supports BASIC, Fortran, AOL, Pascal, BLISS 10, MACRO, SNOBOL and AID. They call this part of the the service the "programmers' area," and it is available at the regular rates. Each user of this area gets 128K bytes of free memory, if it is accessed at least monthly.

The SOURCE supports BASIC, COBOL, Fortran, RPG II, and assembly. It sells storage in blocks of 2,048 bytes. One to ten blocks cost fifty cents per month per block.

Both utilities allow word processing and text editing on-line. CompuServe calls their editor "FILGE." If you have only a terminal, these services make sense. If you have a computer, it is more economical to do these things off-line.

Games

Believe it or not, game playing on-line is a very popular part of these services, perhaps reflecting the high percentage of juvenile users. Each utility has its own main adventure game, and other games.

CompuServe is probably more game oriented than The SOURCE. It has "Adventure (in Colossal Cave)" and two other adventure-type games, including "Scott Adams Adventure." It has DecWars, and SpaceWars and

COMMUNICATION

MegaWars which are interactive with other users, and it sponsors periodic game contests among its subscribers.

The SOURCE's primary adventure game is Blackdragon, though it also has a selection. The SOURCE has more games than CompuServe, but generally they seem more trivial.

Special Interests

Each of these utilities has a vast number of special interest topics, and the variety increases all the time. It will be important to focus on your own two or three high priorities and compare specifically how these are handled by each service. CompuServe publishes a one-sheet Subject Index that you can review at any Radio Shack, and has an insert called Highlights in its magazine. The SOURCE has a pamphlet "SOURCE DIGEST" available at all Computerland stores.

Briefly, here are some special interest topics they provide about equally:

- film reviews
- airline schedules
- travel services
- electronic checkbook
- personal advisor
- legislation status
- sports information

Here are some specialties of The SOURCE:

- customized research (Information on Demand) extra fee
- Mobil Restaurant Guide
- some accredited college courses
- user publishing (royalty to user for material accessed by others)
- employment service (wanted and offered)
- personal appointment calendar

Here are some of the specialties of CompuServe:

- SOFTEX programs for sale and on-line delivery (downloading)
- Printer Art Gallery (downloading) extra fee
- Future File, by Nathan Muller
- Better Homes & Gardens food, decor
- World Book Encyclopedia

PRONTO, BANK ON YOUR ATARI

- limited home banking
- feedback to CompuServe (no charge)
- various contests
- general aviation information

Atari Support

CompuServe is going after the Atari market, and vice versa. Atari advertises on the back cover of each issue of CompuServe's magazine. There is also an official Atari department in CompuServe where users can "Talk to Atari."

The SOURCE, on the other hand, has no official Atari involvement at this time. But it does have an Atari section on the bulletin board.

There is no clear best choice for everyone but there could easily be a "wrong" choice for anyone. We hope this analysis will help you get with the one you need.

by Robert DeWitt

done

PRONTO

Bank On Your Atari

Soon you may be able to use the ATARI to do your banking without ever leaving home. A pilot electronic banking program called PRONTO was started last year by The Chemical Bank of New York for some of its customers who owned ATARIs, and is now being licensed to many more banks across the country. Crocker National Bank in San Francisco, Worthen Bank in Little Rock, and Florida National Bank in Jacksonville are just a few of the other financial institutions that have opted to use PRONTO for a test run in 1983.

The model program began in New York and served 200 customers of Chemical Bank who were willing to participate in this experiment. PRONTO is the latest among other electronic services offered by the bank that have

COMMUNICATION

included a corporate cash-management system and computer-automated tellers.

To begin using PRONTO, a customer needs to have an ATARI computer, a standard telephone line and a modem. Each home computer system serves as a "terminal" for the main program that runs on Tandem Computers at Chemical Bank headquarters. The user connects with the main system by dialing a local network number via phone and modem to begin transactions on a home video screen.

When the first test run began last November, PRONTO customers had to use an acoustic-coupled modem to transmit and receive data. This type of modem has two foam "cups" into which the earpiece and mouthpiece of a standard telephone are placed. Customers used the ATARI 830 (acoustic-coupled) Modem along with the ATARI 850 Interface device and a special cartridge to activate the program. The long-awaited 835 (direct-connect) Modem for the ATARI was not available at the time, but should be soon.

Direct-connect modems are more advanced and much easier to use, and will eventually replace all acoustic-coupled types. The ATARI 830 Modem connects directly with a telephone wall-jack or plugs into the telephone with a "Y" adapter. Most software communications systems that use a modem also require extra software such as TeleLink. The PRONTO system includes a communications-software cartridge, similar to TeleLink, that is supplied to the user at no extra cost.

The PRONTO software is a complete financial management system that allows you to get instant information about your bank account. It also provides screens with forms for household budgets. You may register checks, pay bills, send electronic mail to other PRONTO users and keep accurate tax records that include principal and interest categories. The budget screens allow you to list up to 50 items and five different personal budgets per household. Each family member can have a secret access code to insure privacy. You may monitor all your account activities and get an "electronic statement" along with your usual monthly printed statement.

Most people who were asked to participate in this project responded enthusiastically. In San Francisco, Crocker Bank announced to its employees and the general public that it was looking for participants to begin the PRONTO pilot in early 1983. The fifty openings for test users at Crocker were filled immediately. A total of 200 customers and employees are expected to be using the PRONTO pilot in San Francisco by July. Many users of the

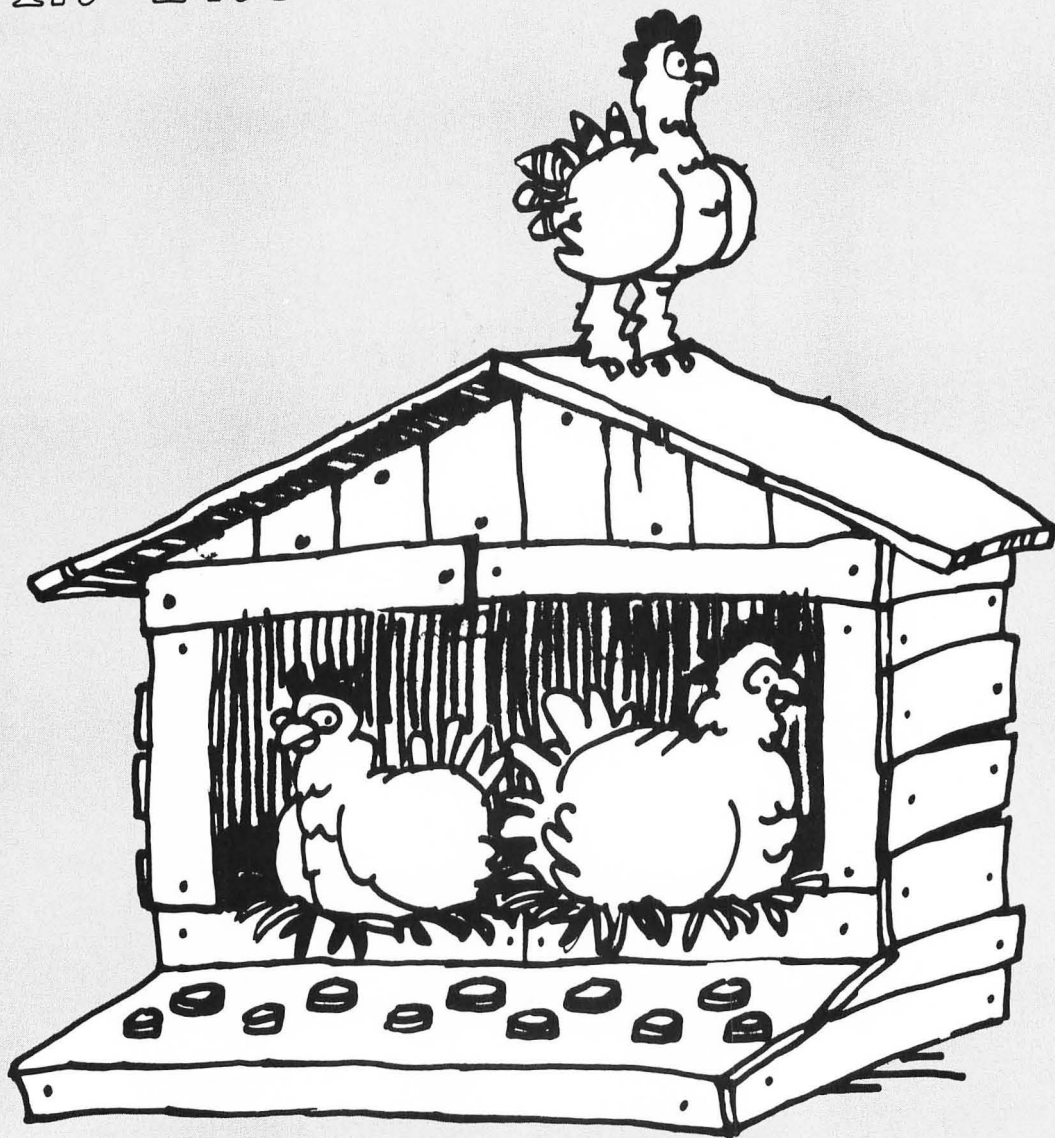
DIALING FOR DATA

Crocker system will have the option of using other hardware, such as the IBM PC or the Apple II.

The banks have not yet determined how much to charge for PRONTO, but when Chemical Bank queried its pilot customers, most agreed that they would be willing to pay about \$10 a month for this service. If you feel that you may be interested in this type of service, ask your own bank. Who knows? It may be offering electronic banking like PRONTO in the very near future.

by Deborah Burns

Games In The Public Domain



Chicken— A Great Game

Why did the chicken cross the road? To provide a premise for a computer game. Actually, our chicken is trying to score points by getting safely across this busy highway. Each time he succeeds adds to his score, but the cars go faster and faster. If he gets hit, the SPCA sends an ambulance and the cops slow the traffic down for a while.

This clever game can be yours for the copying, courtesy of Stan Ockers, who wrote it in BASIC and assembly language, and Mike Dunn, editor of A.C.E. Newsletter (Eugene, Oregon), who printed it first and gave us permission to pass it on to you.

System Requirements 16K RAM, joystick

by Stan Ockers

Stan Ockers, from Lockport, Illinois, has brightened the world of ATARI enthusiasts with his selfless support of user groups and contributions to "the public domain," that large and growing body of programs appearing in newsletters and in ANTIC for which no commercial software rights are claimed. **Chicken**, ANTIC's first public domain game, is representative of the high quality of Stan's programming.

Chicken will not work with the 1200XL computer.

```
4 REM ***** CHICKEN *****
5 REM BY STAN OCKERS REVISED BY GUY HU
  RT
9 C1=4:C2=0:C3=0:SCOREONE=500:RN3=36:R
N4=21:W=7:U=11:BSW=0:GOSUB 15000
10 OPEN #1,4,0,"K":DIM L$(20),S$(20),
C$(20),Z$(5):Z$(1)="/" :Z$(2)="@":Z$(3)
="%" :Z$(4)="$":Z$(5)=""
11 OPEN #2,4,0,"D:HI2.DAT":INPUT #2,HI
GH:CLOSE #2
12 REM DEAR OWMER:IF YOU OWN A DISK
13 REM CREATE A FILE CALLED HI2.DAT
14 REM AND STORE A ZERO IN IT
15 REM ELSE MAKE STMTS 11.725&742
16 REM REM'S UNTIL YOU OWN A DISK
19 POSITION 0,16:? #6;"INITIALIZING..."
:Q2=0
20 REM PAGE 6 ROUTINES AND DATA
```

```

40 FOR I=1536 TO 1587:READ A:POKE I,A:
NEXT I
41 REM VERTICAL BLANK RTNE.
42 DIM VB$(210):FOR I=1 TO 210:READ A:
VB$(I)=CHR$(A):NEXT I
43 REM LOAD PLAYER RTNE.
45 DIM LD$(73):FOR I=1 TO 73:READ A:LD
$(I)=CHR$(A):NEXT I
47 REM INSERT ADR. OF ROUT. IN PAGE 6
48 A=ADR(VB$):B=INT(A/256):POKE 1540,B
:POKE 1538,A-256*B
50 DATA 104,160,52,162,6,169,7,76,92,2
28,104,160,98,162,228,169,7,76,92,228
52 DATA 120,120,120,120,30,57,81,105,1
5,15,15,15,0,0,0,0,52,53,54,55,2,2,3,4
,12,0,0,0,15,11,11,11
54 REM LINE56-CHANGE 24T056TOSKIP ORTH
.
55 REM LINE56 CHANGE 28 TO 34 FOR CONT
. MOVEM.
56 DATA 72,138,72,152,72,162,0,189,120
,2,29,44,6,160,15,24,176,32,201,15,240
,28,201,14,208,2,160,13,201,13
57 DATA 208,2,160,14,201,11,208,2,160,
7,201,7,208,2,160,11,192,15,240,6,61,4
8,6,157,28,6,152,61,44,6,157,44,6
58 DATA 232,224,4,144,195
60 DATA 162,0,189,32,6,133,203,189,36,
6,133,204,189,40,6,133,209,198,209,16,
7,232,224,4
65 DATA 144,232,176,91,189,28,6,133,20
7
70 DATA 70,207,176,26,188,24,6,192,1,2
40,19,208,1,200,177
75 DATA 203,240,6,136,145,203,200,208,
245,136,145,203,222,24,6,70,207,176,29
,188,24,6,200,192,254,176,21
80 DATA 177,203,208,247,136,177,203,24
0,6,200,145,203,136,208,245,200,145,20
3,254,24,6,70,207,176,3,222,20
85 DATA 6,70,207,176,3,254,20,6,189,20
,6,157,0,208
90 DATA 24,144,154,162,4,189,11,208,24

```

```

0,5,169,0,157,39,6,202,208,243,104,168
,104,170,104,76,98,228
100 DATA 234,234,234,104,104,104,170,1
89,32,6,133,186,189,36,6,133,187,104,1
33,213,104,133,212
110 DATA 189,24,6,133,195,169,0,168,19
2,255,176,35,196,195,240,5,145,186,200
,208,243,162,0,161,212,240,11
120 DATA 145,186,230,212,200,192,255,1
76,11,208,241,169,0,145,186,200,192,25
5,144,249,96,234,234
150 REM CAR COLOR DATA
160 FOR I=1 TO 20:READ A:CS(I)=CHR$(A)
:NEXT I
170 DATA 24,60,218,68,90,186,70,150,54
,232,74,168,88,154,21,252,200,76,228,2
8
190 REM DEFINE PM AREA-SINGLE LINE RES
200 A=PEEK(106)-16:POKE 54279,A:PM=256
*A
205 REM PLAY MISSILE POINTERS
210 FOR I=4 TO 7:POKE 1568+I,A+I:NEXT
I
212 FOR I=1568 TO 1571:POKE I,0:NEXT I
218 REM DATA FOR PLAYER IMAGES
220 FOR I=PM TO PM+121:READ A:POKE I,A
:NEXT I
230 DATA 16,56,16,56,40,16,16,16,146,2
54,254,124,56,56,40,40,40,40,108,0
232 DATA 126,195,219,219,91,219,219,21
9,219,91,219,219,195,126,0
234 DATA 126,195,210,219,218,219,219,2
19,219,218,219,219,195,126,0
236 DATA 33,34,150,84,57,30,60,123,159
,30,52,86,151,36,194,193,0
238 DATA 16,56,16,56,40,16,16,56,124,2
54,186,56,56,40,40,40,44,32,96,0
240 DATA 16,56,16,56,40,16,16,146,214,
124,56,56,40,40,40,104,8,12,0
242 DATA 126,255,173,173,239,199,199,1
99,199,239,173,173,255,126,0
270 REM INIT HORIZ.& VERT. POSTN.
280 RESTORE 282:FOR I=1556 TO 1563:REA

```

```

D A:POKE I,A:NEXT I
282 DATA 120,120,120,120,30,57,81,105
288 REM INIT. COLORS
290 DIF=3:BONUS=300:POKE 704,40:CP=0:F
OR I=1 TO 3:POKE 704+I,ASC(C$(CP+I)):N
EXT I:CP=3:BPOS=1
295 REM DRAW ROAD SET PRIORITY
300 GRAPHICS 17:FOR I=1 TO 20:L$(I)="--
":NEXT I
305 FOR I=2 TO 20 STEP 2:S$(I)="--":S$(
I-1)="--":NEXT I
310 POSITION 0,2:? #6:L$:POSITION 0,11
:? #6:L$:POSITION 0,13:? #6:L$:POSITIO
N 0,22:? #6:L$
312 POSITION 0,5:? #6:S$:POSITION 0,8:
? #6:S$:POSITION 0,16:? #6:S$:POSITION
0,19:? #6:S$:POKE 710,90
340 REM INIT & PRINT INFO-RESET TIMER
350 SCORE=50:POSITION 0,1:? #6:"score
time":POSITION 0,23:? #6:"di
f high":
360 POSITION 0,0:? #6:SCORE:POSITION 1
5,22:? #6:HIG:POKE 19,0:POKE 20,0
365 REM INIT PM GR.-FLAGS
370 POKE 559,62:POKE 53277,3:I1=68:I2=
88:FL=I1
375 REM LOAD PLAYERS-SETCOLORS-PLAYERS
SIZES
380 LD=ADR(LD$):A=USR(LD,0,PM+RN5):A=U
SR(LD,1,PM+21):A=USR(LD,2,PM+21):A=USR
(LD,3,PM+21)
385 A=USR(1536):REM INSERT VBI RTNE.
390 POKE 53257,1:POKE 53258,1:POKE 532
59,1:POKE 623,1
391 IF BPOS=18 THEN POKE 53257,3:POKE
53258,3:POKE 53259,3
393 REM INIT SPEEDS
395 POKE 1576,2:FOR I=1577 TO 1579:POK
E I,RND(0)*DIF+1:NEXT I
398 POSITION 1,22:? #6:DIF
400 REM IF CARS OFF-SCREEN.CHANGE LANE
S
410 IF PEEK(1557)<15 AND PEEK(1561)=57

```

```

THEN POKE 1561,193:A=USR(LD,1,PM+RN3)
:POKE 1585,7:GOSUB 1000:POKE 705,C
420 IF PEEK(1557)>240 AND PEEK(1561)=1
93 THEN POKE 1561,57:A=USR(LD,1,PM+RN4)
):POKE 1585,11:GOSUB 1000:POKE 705,C
430 IF PEEK(1558)<15 AND PEEK(1562)=81
THEN POKE 1562,169:A=USR(LD,2,PM+RN3)
:POKE 1586,W:GOSUB 1000:POKE 706,C
440 IF PEEK(1558)>240 AND PEEK(1562)=1
69 THEN POKE 1562,81:A=USR(LD,2,PM+RN4)
):POKE 1586,U:GOSUB 1000:POKE 706,C
450 IF PEEK(1559)<15 AND PEEK(1563)=10
5 THEN POKE 1563,145:A=USR(LD,3,PM+RN3)
):POKE 1587,7:GOSUB 1000:POKE 707,C
460 IF PEEK(1559)>240 AND PEEK(1563)=1
45 THEN POKE 1563,105:A=USR(LD,3,PM+RN
4):POKE 1587,11:GOSUB 1000:POKE 707,C
465 REM PRINT TIME-CHECK FOR TIMEUP
470 TIME=15-PEEK(19):POSITION 16,0:? #
6:TIME;" ":IF TIME<=0 THEN 910
471 GOSUB 1200
472 REM RESET SOUND-HORN RTNE.
473 SOUND 0,0,0,0
475 IF RND(0)>0.5 THEN SOUND 1,0,0,0
480 IF RND(0)<0.05*RN2 THEN SOUND 1,7,
12,10
484 REM CHICKEN STOMP
485 P=PEEK(1564):IF P>15 OR P<35 THEN
500
490 IF P=15 THEN A=USR(LD,0,PM+RN5):GO
TO 500
492 IF FL=I1 THEN FL=I2:SOUND 0,16*RN,
6,8:GOTO 496
494 IF FL=I2 THEN FL=I1:SOUND 0,22*RN,
6,8
495 REM CHECK FOR REACHING BOTTOM
496 A=USR(LD,0,PM+FL+RN5)
500 IF PEEK(1560)>230 THEN 810
501 RN=INT(RND(0)*15+1):IF BPOS<>4 AND
BPOS<>19 THEN RN=1
505 REM CHECK FOR COLLISION
510 IF PEEK(53260)=0 THEN 410
515 REM SPLAT !

```



```

520 A=USR(LD,0,PM+51+RN5):FOR J=1 TO 3
:SOUND 0,RND(0)*255,8,8:SOUND 1,RND(0)
)*255,8,8
525 POKE 704,PEEK(704)+8:FOR I=1 TO 30
:NEXT I:NEXT J:SOUND 0,0,0,0:SOUND 1,0
,0,0:POKE 704,40
527 REM DECREASE SCORE-CHECK FOR 0
530 SCORE=SCORE-1:POSITION 0,0:? #6:SC
ORE;" " :IF SCORE<=0 THEN 702
533 REM MOVE UP, RELOAD BIRD-RESET COL
L
535 POKE 1560,PEEK(1560)-24:A=USR(LD,0
,PM+RN5):IF DIF>0 THEN DIF=DIF-1
550 IF DIF=0 THEN GOTO 700
590 POKE 53278,0:GOTO 395
700 REM SCORE=0
702 POSITION 3,2:? #6;"CHICKEN'S DEAD!
"
704 REM AMBULANCE
705 C=1
706 IF PEEK(1576+C)-0 THEN C=C+1:GOTO
706
707 IF C>3 THEN C=1
713 POKE 1576+C,1:POKE 1560+C,PEEK(156
0):POKE 1556+C,220:A=USR(LD,C,PM+107)
715 FOR J=1 TO 6:FOR P=60 TO 40 STEP -
2:SOUND 0,P,10,8:FOR I=1 TO 6:NEXT I
716 NEXT P:FOR P=40 TO 60 STEP 2:SOUND
0,P,10,8:FOR I=1 TO 6:NEXT I:NEXT P:N
EXT J:SOUND 0,0,0,0
718 REM NEW HIGH SCORE
720 A=USR(1546):IF SCORE<=HIGH THEN 73
0
721 HIGH=SCORE:FOR Q=1 TO 7:POSITION 1
1,23:? #6:HIGH;" " :POSITION 15,22:? #
6;"high":SOUND 0,40,10,15
722 FOR Q2=1 TO 50:NEXT Q2
723 POSITION 11,23:? #6;"high":POSITIO
N 15,22:? #6:HIGH;"-":SOUND 0,50,10,15
:FOR Q2=1 TO 50:NEXT Q2
724 NEXT Q:SOUND 0,0,0,0
725 TRAP 1300:OPEN #2,8,0,"D:HI2.DAT":
PRINT #2,HIGH:CLOSE #2

```

```

730 POSITION 2,6:? #6;"press FIRE butt
on ":POSITION 4,7:? #6;"to play again"
732 FOR I=53248 TO 53251:POKE I,0:NEXT
I:SOUND 0,0,0,0:SOUND 1,0,0,0
735 REM WAIT FOR BUTTON
740 IF STRIG(0)-1 THEN 740
741 C1-4:C2-0:C3-0:SCOREONE=500:BSW-0
742 OPEN #2,4,0,"D:HI2.DAT":INPUT #2,H
IGH:CLOSE #2
745 REM PM GRAPHICS OFF
750 POKE 53278,0:POKE 53277,0:A=USR(15
46):GOTO 280
800 REM BACK TO TOP-STOP BIRDMOVMT.
810 POKE 1560,30:A=USR(LD,0,PM+RN5):PO
KE 1576,0
815 REM SIGNAL AND INCREMENT SCORE
820 FOR I=1 TO 5:FOR J=10 TO 5 STEP -1
:SOUND 0,J,14,8:SOUND 1,J,2,8:NEXT J:S
OUND 0,0,0,0:SOUND 1,0,0,0
825 A=USR(LD,0,PM+68+RN5):FOR J=1 TO R
ND(0)*30:NEXT J:A=USR(LD,0,PM+88+RN5)
830 SCORE=SCORE+DIF*2:POSITION 0,0:? #
6:SCORE;" ":NEXT I
831 IF SCORE>=SCOREONE THEN 833
832 GOTO 840
833 FOR I=1 TO 3:POSITION 0,0:? #6;"wo
w!":GOSUB 835:POSITION 0,0:? #6:SCORE:
GOSUB 835:NEXT I
834 SCOREONE=SCOREONE+500:GOTO 840
835 FOR Q=20 TO 1 STEP -4:SOUND 1,Q*5,
10,15:NEXT Q:SOUND 1,0,0,0:RETURN
840 IF DIF<9 THEN DIF=DIF+1:IF DIF=9 T
HEN C1-4:C2=10:C3=0
841 IF DIF<>9 THEN C1-4:C2=0:C3=0
842 REM CHECK FOR BONUS
843 IF SCORE<BONUS THEN 850
844 SOUND 0,25,10,10:BONUS=BONUS+300:P
=PEEK(19):IF P<11 THEN POKE 19,0:GOTO
848
846 POKE 19,P-10
848 POSITION BPOS,12:? #6;"*":BPOS=BPO
S+1:BSW=BSW+1:IF BPOS>19 THEN BPOS=0
849 IF BPOS<>0 THEN W=7:U=11

```



```

850 IF BPOS=7 THEN RN2=17:GOTO 390
851 IF BPOS<>7 THEN RN2=1
852 IF BPOS=13 THEN RN3=107:RN4=107:GO
TO 390
853 IF BPOS<>13 THEN RN3=36:RN4=21
854 IF BPOS=0 THEN W=11:U=7
855 IF BPOS=16 THEN RN3=0:RN4=0:RN5=21
:GOTO 390
856 IF BPOS<>16 THEN RN3=36:RN4=21:RN5
=0
857 IF BPOS=19 THEN RN2=18:RN3=107:RN4
=0
858 IF BPOS<>19 THEN RN2=1:RN3=36:RN4=
21
859 IF BPOS=1 AND BSW>0 THEN GOSUB 140
0:GOTO 9000
869 GOTO 390
900 REM TIME'S UP RTNE.
910 POSITION 5,2:? #6;"TIME'S UP!"
920 GOTO 720
990 REM CHANGE CAR COLOR RTNE.
1000 IF BPOS=10 OR BPOS=19 THEN CP=CP-
1
1005 CP=CP+1:IF CP=20 THEN CP=1
1010 POKE 77,0:C=ASC(C$(CP)):RETURN
1200 IF TIME=10 THEN SETCOLOR 4,12,2:F
OR K=1 TO 10:SOUND 3,K,10,10:NEXT K:SO
UND 3,0,0,0
1210 IF TIME<>10 THEN SETCOLOR C1,C2,C
3
1220 RETURN
1300 GRAPHICS 0:SOUND 1,55,10,15:? "DI
SK PROBLEM.....HIGH SCORE WAS NOT SAVE
D...":FOR I=1 TO 500:NEXT I
1310 SOUND 0,0,0,0:? "TYPE RUN TO PLAY
AGAIN"
1320 TRAP 1300
1400 FOR I=1 TO 255:SOUND 0,I,12,12:SE
TCOLOR 4,I,10:NEXT I
1410 FOR BPO=1 TO 5:FOR BPOT=0 TO 19:S
OUND 0,BPOT*BPO,10,12
1420 POSITION BPOT,12:? #6:Z$(BPO):NEX
T BPOT:NEXT BPO

```

```

1425 SOUND 0,0,0,0
1426 POSITION 0,12:? #6:"&"
1427 POSITION 0,13:? #6:L$
1428 BSW=0
1430 RETURN
9000 FOR I=1 TO 150:NEXT I:POKE 19,99:
GOTO 470
15000 GRAPHICS 17:POSITION 0,0:? #6:"P
RESS OPTION FOR":POSITION 0,1:? #6:"OB
JECTIVE":POSITION 0,2
15010 ? #6:"PRESS TRIGGER":POSITION 0,
3:? #6:"BUTTON TO PLAY"
15020 IF PEEK(53279)-3 THEN 15050
15030 IF STRIG(0)-0 THEN 16000
15040 GOTO 15020
15050 GRAPHICS 17:POSITION 0,0:? #6:"Y
OUR OBJECTIVE?":POSITION 0,1:? #6:"SI
MPLE. JUST GET THE"
15055 POSITION 0,2:? #6:"chicken SAFEL
Y":POSITION 0,3:? #6:"ACROSS THE ROAD"
15060 POSITION 0,4:? #6:"without GETTI
NG HIT":POSITION 0,5:? #6:"BY CARS."
15070 POSITION 0,7:? #6:"YOU MUST LEAR
N THE":POSITION 0,8:? #6:"SCORING SYST
EM."
15080 POSITION 0,10:? #6:"all sorts of
neat":POSITION 0,11:? #6:"surprises a
re in":POSITION 0,12
15090 ? #6:"store for the":POSITION 0,
13:? #6:"advanced player."
15100 POSITION 0,15:? #6:"PLEASE PRESS
OPTION."
15160 IF PEEK(53279)-3 THEN 16000
15170 GOTO 15160
16000 GRAPHICS 17:POSITION 6,6:? #6:"C
HiCKeN":POSITION 0,7
16100 FOR I=1 TO 4:SOUND 0,16,6,8:FOR
J=1 TO 20:NEXT J:SOUND 0,22,6,8:FOR J=
1 TO 20:NEXT J
16150 NEXT I:SOUND 0,0,0,0
16200 ? #6:"written by":POSITION 4,8:
? #6:"STAN OCKERS":POSITION 0,9:Q2=0
16205 REM D&D BUGGED

```

```

16210 ? #6:"REVISED BY":FOR I=1 TO 200
:NEXT I:POSITION 4,10:? #6:"GUY A. HUR
T":MK=50
16220 FOR I=1 TO 250:NEXT I
16230 FOR I=1 TO 5:FOR J=10 TO 5 STEP
-1:SOUND 0,J,14,8:SOUND 1,J,2,8:NEXT J
:SOUND 0,0,0,0:SOUND 1,0,0,0
16240 FOR K=1 TO MK:NEXT K:MK=MK-10:NE
XT I
16250 RETURN

```

TYPO TABLE

Variable checksum = 1070383

Line	num	range	Code	Length
4	-	14	HB	523
15	-	50	NJ	532
52	-	65	JK	534
70	-	110	VV	598
120	-	230	OU	561
232	-	280	KD	509
290	-	312	KH	540
340	-	380	UC	504
385	-	420	JU	583
430	-	465	JI	509
470	-	494	VU	502
495	-	525	QN	518
527	-	707	AW	447
713	-	721	AD	507
722	-	735	KR	502
740	-	820	NR	552
825	-	840	II	512
841	-	854	IJ	530
855	-	1005	ZO	458
1010	-	1410	KZ	537
1420	-	15040	VF	495
15050	-	15080	LH	521
15090	-	16200	GN	529
16205	-	16250	RH	374

done

Attack on the Death Star

In order to protect your home base from the dreaded Death Star, you are launched in your X-wing fighter to attack the enemy. As the simulation begins, you are flying “down the trench,” the walls of the trench whipping past. The object: destroy the five radiation vents leading to the Death Star’s main reactor. If you succeed, the reactor will overheat and self-destruct, destroying the Death Star.

To hit the radiation vents, line up your cursor aiming system, using a joystick in Port One, and fire, using the red button. The vents are green oval openings in the bottom of the trench. The Death Star has a full complement of Tie fighters for its defense. The fighters attack one by one, firing furiously. If you are hit too many times, your fighter will explode and crash. To combat the Tie fighters, you’ll have to wait till the Tie fighter is in the center of the screen before you can hit it.

This game has been improved since it first appeared in ANTIC. It now has a scoring line and two new “endings.”

by David Plotkin

System Requirements: 32K RAM, joystick

*David Plotkin is a Chemical Engineer with Standard Oil of California, and a game programmer by avocation. His **Attack on the Death Star** has been improved since publication, and the new version is printed here in the book. Dave programs mostly in BASIC, but with some assembly language routines, and his games have appeared in several publications, including ANTIC.*

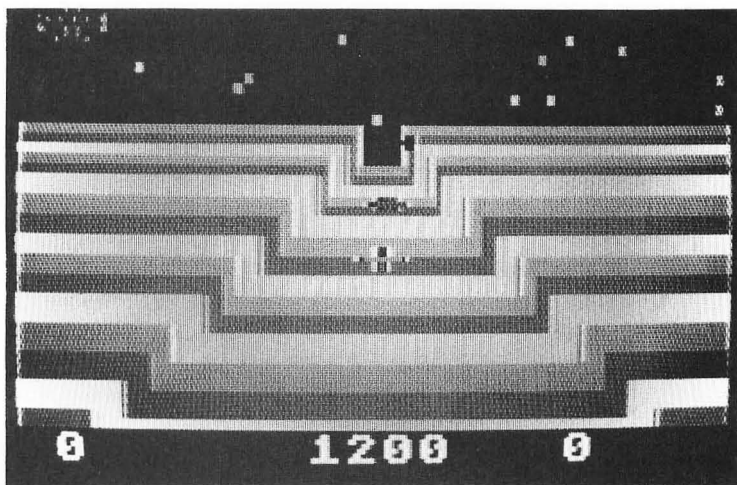
```
5 REM ***** DEATHSTAR *****
8 REM BY DAVE PLOTKIN REVISED BY D. PL
  OTKIN 1983
10 GOSUB 1500:SCORE=0:SD=1200:B=0:HISC
  ORE=0:F=1:F1=1:GOSUB 800
90 COL=PEEK(708):POKE 708,PEEK(709):PO
  KE 709,PEEK(710):POKE 710,COL
100 ST=PEEK(632):IF ((NUMH=RT AND ST=1
  3) OR (NUMH=RT+4 AND ST=14)) THEN ST=1
  5:GOTO 130
110 NUML=NUML+20*(ST=14)-20*(ST=13):NU
```

```

MH=NUMH+(NUML>255)-(NUML<0):NUML=NUML+
256*(NUML<0)-256*(NUML>255)
120 POKE DL4,NUML:POKE DL5,NUMH:NN=2*(
ST=13)-2*(ST=14):N=N+NN:Y3=Y3+NN
130 YTEMP=Y:IF SIZEL=0 THEN GOTO 175
140 IF SIZEL>1 THEN SIZEL=SIZEL-1:Y=70
*(SIZEL=2)+56*(SIZEL=1):SH=68*(SIZEL=2
)+76*(SIZEL=1):GOTO 180
150 IF ABS(Y-Y2-7)<5 AND ABS(X-X2)<4 T
HEN SOUND 1,40,8,8:POKE 656,0:POKE 657
,1:SCORE=SCORE+10:? SCORE:SIZEC=1.9
160 IF ABS(Y3-Y)<3 AND SIZEH<>0 THEN P
OKE 707,82:SOUND 2,100,8,10:B=B+1:POKE
656,0:POKE 657,15:? B:FOR W=1 TO 100:
NEXT W:IF B=5 THEN 700
170 SH=84:SOUND 2,0,0,0:SIZEL=0:Y=55
175 IF STRIG(0)=0 THEN SIZEL=3:Y=88:SO
UND 2,10,8,8:SH=60
180 IF (SIZEC=0 AND INT((30-5*B)*RND(0
))=0) THEN SIZEC=2:Y2=28:X2=120
190 IF SIZEC=0 THEN GOTO 280
200 IF SIZEC=2 THEN GOTO 210
202 SIZEC=SIZEC-0.1:Y2=Y2-2:X2=X2+5:PO
KE 53250,X2:F=F+1:IF F>4 THEN F=1
205 SH2=27*(F=1)+188*(F=2)+200*(F=3)+2
12*(F=4):SOUND 1,X2-120,10,10:IF SIZEC
<>0 AND SIZEC<>1.8 THEN GOTO 240
207 SH2=BLANK:SOUND 1,0,0,0:GOTO 240
210 XX=RND(0):Y2=Y2+2*(Y2<78)*(ST=13)-
2*(ST=14)*(Y2>20)+4*(XX>0.8)*(Y2<78)-4
*(XX<0.12)*(Y2>20)
220 FF=FF+2*(X2<112)-2*(X2>130):X2=X2+
FF:POKE 53250,X2:SH2=20*(Y2<65 AND Y2>
43)+40*(Y2<=43)
225 IF (SIZEC=1.8 OR SIZEC=0) THEN SH2
=BLANK:SOUND 1,40*(SIZEC=1.8),8,8*(SIZ
EC=1.8)
230 IF (SM=0 AND INT(RND(0)*(16-2*B))=
0) THEN Y1=Y2:X1=X2-4:POKE 53249,X1:SM
=1:SOUND 3,150,8,6
240 IF SM=0 THEN GOTO 280
250 Y1=Y1+2*(ST=13)*(Y1<78)-2*(ST=14):
SM=SM+0.25*LV

```

ATTACK ON THE DEATH STAR



```
255 SH1=BLANK*(SM=4)+128*(SM<4 AND SM>
=3)+148*(SM<3 AND SM>=2)+168*(SM<2)
260 IF SM=4 THEN SOUND 3,0,0,0:IF (Y1>
40-5*(LV=1) AND Y1<60+5*(LV=1)) THEN S
OUND 3,100,8,8:SETCOLOR 4,5,12:SD=SD-1
00*LV:FOR Q=1 TO 50:NEXT Q
265 IF SM=4 THEN SM=0:POKE 656,0:POKE
657,8:? SD:" ":SETCOLOR 4,0,0:SOUND 3
,0,0,0
270 IF SD<0 THEN SD=0:POKE 656,0:POKE
657,8:? SD:" ":GOTO 600
280 F1=F1+1:IF F1=4 THEN F1=1
282 SH4=224*(F1=1)+232*(F1=2)+240*(F1=
3)
285 IF SIZEH=0 THEN GOTO 320
290 IF SIZEH<1 THEN SIZEH=SIZEH+0.05:S
OUND 0,20*SIZEH,10,6:GOTO 400
300 IF SIZEH=1 THEN Y3=34+N:SH3=372:SI
ZEH=2:SOUND 0,10,8,3:X4=X4+4*LV:FOR Q=
0 TO 3:POKE 53252+Q,X4-4*Q:NEXT Q
305 IF Y3>85 THEN SH3=BLANK:SIZEH=0:GO
TO 400+200*(X4>=132)
310 Y3=Y3+3:SH3=92*(Y3-N>=75)+104*(Y3-
N>46 AND Y3-N<75)+116*(Y3-N<=46)
```



```

320 IF PEEK(19)>=10 THEN POKE 19,0:SIZEH=0.1:POKE 707,196
400 IF YTEMP<>Y THEN D=USR(ADR(E$),P0+YTEMP,PB+BLANK):D=USR(ADR(E$),P0+Y,PB+SH)
410 D=USR(ADR(E$)+50-25*(SIZEC<1.8),P2+Y2,PB+SH2):D=USR(ADR(E$)+50,P1+Y1,PB+SH1)
420 D=USR(ADR(E$)+25,P3+Y3,PB+SH3):D=USR(ADR(E$),P4,PB+SH4):GOTO 90
600 D=USR(ADR(E$),P0+YTEMP,PB+BLANK):D=USR(ADR(E$)+50,P2+Y2,PB+BLANK)
602 D=USR(ADR(E$)+50,P1+Y1,PB+BLANK):D=USR(ADR(E$)+25,P3+Y3,PB+BLANK)
605 IF SD>0 THEN GOTO 616
610 FOR W=1 TO 50:POKE 708,RND(0)*255:POKE 709,RND(0)*255:POKE 710,RND(0)*255
615 SOUND INT(RND(0)*4),RND(0)*255,8,8:NEXT W
616 NUML=PEEK(DL4):NUMH=PEEK(DL5)
620 IF NUMH-RT>=4 THEN GOTO 635
625 NUML=NUML+20:NUMH=NUMH+(NUML>255):NUML=NUML-256*(NUML>255)
630 POKE DL4,NUML:POKE DL5,NUMH:GOTO 620
635 FOR W=120 TO 1 STEP -1:SOUND 0,100,8,W/10:NEXT W
640 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0:SOUND 3,0,0,0
645 POKE 708,232:POKE 709,250:POKE 710,132:POKE 712,0:POKE 704,90
650 FOR XNEW=X4 TO 132:FOR Q=0 TO 3:POKE 53252+Q,XNEW-4*Q:NEXT Q:NEXT XNEW
655 FOR C=10 TO 100 STEP 10:FOR Y=30 TO 13 STEP -1:D=USR(ADR(E$),P0+Y,PB+13):SOUND 1,Y,6,8:NEXT Y
657 POKE P4+INT(RND(0)*8),PEEK(53770)
660 SOUND 0,120-C,8,8:POKE 711,C:FOR W=1 TO 75:NEXT W:NEXT C:SOUND 1,0,0,0
665 D=USR(ADR(E$),P0+13,PB+BLANK)
670 POKE 623,1:SETCOLOR 4,0,14:SOUND 0,140,8,8:FOR N=14 TO 0 STEP -1:SOUND 0

```

ATTACK ON THE DEATH STAR

```

7,140-N*10,8,8:SETCOLOR 4,0,N
673 FOR NN=1 TO 30:NEXT NN:XN=XN+8:POKE
53252,132+N:POKE 53255,120-XN:NEXT
N:XN=0
677 FOR N=14 TO 0 STEP -1:SOUND 0,140-
N*10,8,8:SETCOLOR 4,0,N:FOR NN=1 TO 30
:NEXT NN:XN=XN+8
680 POKE 53253,128+N:POKE 53254,124-X
N:NEXT N:XN=0:FOR W=1 TO 500:NEXT W:PO
KE 53277,0:GRAPHICS 17
683 SOUND 0,0,0,0:POSITION 1,8:? #6:"T
HE DEATH STAR HAS":? #6;"DESTROYED YOU
R HOME":? #6;"BASE.YOU lost!!!!!!"
684 IF SCORE>HISCORE THEN HISCORE=SCOR
E
685 ? #6:? #6:" FINAL SCORE ":SCORE:?
#6:" points":? #6:" TO PLAY AG
AIN":? #6:" PRESS FIRE BUTTON"
686 ? #6:" HIGH SCORE ";HISCORE
687 IF STRIG(0)=-1 THEN GOTO 687
690 B=0:SCORE=0:SD=1200:F=1:F1=1:GOSUB
836:GOTO 90
700 D=USR(ADR(E$),P0+YTEMP,PB+BLANK):D
=USR(ADR(E$)+50,P2+Y2,PB+BLANK)
701 D=USR(ADR(E$)+50,P1+Y1,PB+BLANK):D
=USR(ADR(E$)+25,P3+Y3,PB+BLANK)
703 SOUND 1,0,0,0:SOUND 2,0,0,0:SOUND
3,0,0,0
705 NUML=PEEK(DL4):NUMH=PEEK(DL5):FOR
W=1 TO 10:IND=40*(W/2=INT(W/2))-40*(W
/2<>INT(W/2))
710 NUML=NUML+IND:NUMH=NUMH+(NUML>255)
-(NUML<0):NUML=NUML-256*(NUML>255)+256
*(NUML<0)
715 SOUND 0,W*20,8,8:POKE DL4,NUML:POK
E DL5,NUMH
720 FOR M=1 TO 100:NEXT M:NEXT W:D=USR
(ADR(E$),PB+394,PB+BLANK):POKE 53277,0
:BYT=7:LIN=23
725 DS(1)="V%VV%VV%VV%j P%VV%VV%?%VV%VV%k Uz
@%VV%VV%j _%VV%VV%V?%d%VV%k Uz y%VV%m U ^y%VV%m5BW@
m5 * @6W u g 4 R "
726 DS(81)="-"!G%L6W u g % m5 - t e @ m5 BW @ %

```



```

mU^y k Uz y V ? % d = j _ k Uz @
? % j P

```

```

727 D$(161)="" : POKE DL4,0: POKE DL5,RT
+1: STRT=256*(RT+1)+10*20+6: D=USR(PICT
,STRT,ADR(D$),BYT,LIN,20)
728 FOR N=1 TO 150:NEXT N
730 FOR NN=1 TO 200:SOUND 0,NN,8,14:SO
UND 1,NN+10,8,14: POKE 712,14-NN/15:NEX
T NN:FOR NN=1 TO 36:COL=PEEK(708)
732 POKE 708,PEEK(709): POKE 709,PEEK(7
10): POKE 710,COL:SOUND 0,NN*6,8,14:SO
UND 1,NN*5,8,14
735 FOR MM=1 TO 15:NEXT MM:NEXT NN:POK
E 712,0:FOR Q=0 TO 2:FOR N=14 TO 0 STE
P -1: POKE 708+Q,N:SOUND 0,140-10*N,8,1
4
740 SOUND 1,152-10*N,8,14:FOR NN=1 TO
30:NEXT NN:NEXT N:FOR NN=1 TO 50:NEXT
NN:NEXT Q
745 FOR N=14 TO 0 STEP -1:SOUND 0,140,
8,N:SOUND 1,152,8,N:FOR NN=1 TO 30:NEX
T NN:NEXT N
775 GRAPHICS 17:POSITION 3,3:? #6:"CON
GRATULATIONS WARRIOR":? #6:" Y
OU HAVE DESTROYED":? #6:" THE"
780 ? #6:" death star":GOTO 684
800 DIM D$(650),E$(80):FOR A=1 TO 75:R
EAD I:E$(A,A)-CHR$(I):NEXT A
804 DATA 104,104,133,204,104,133,203,1
04,133,207,104,133,206,160,0,177,206,1
45,203,200,192,8,208,247,96
805 DATA 104,104,133,204,104,133,203,1
04,133,207,104,133,206,160,0,177,206,1
45,203,200,192,12,208,247,96
806 DATA 104,104,133,204,104,133,203,1
04,133,207,104,133,206,160,0,177,206,1
45,203,200,192,20,208,247,96
810 A=PEEK(106)-20:POKE 54279,A:PB=256
*A:FOR A=PB TO PB+247:READ I:POKE A,I:
NEXT A
815 DATA 0,0,0,0,0,0,129,129,153,231,2
31,153,129,129,0,0,0,0,0,0,0,0,0,0,0,0
,0,66,90,102,90,66,0,0,0,0,0,0,0,0

```

ATTACK ON THE DEATH STAR

```

816 DATA 0,0,0,0,0,0,0,0,36,60,36,0,0,
0,0,0,0,0,0,24,24,60,60,60,126,255,2
19,0,24,24,24,60,126,90,0
820 DATA 0,0,16,16,56,40,0,0,0,0,16,16
,124,16,16,0,0,0,0,0,24,60,126,255,1
26,60,24,0,0,0,0,0,24,60,126,60,24,0,0
821 DATA 0,0,0,0,0,24,60,24,0,0,0,0,0,
0,24,60,36,126,189,90,90,165,255,165,9
0,90,189,102,60,24,0,0
825 DATA 0,0,0,24,36,60,60,90,90,36,36
,36,90,90,60,36,24,0,0,0,0,0,0,0,0,
,24,36,24,24,24,36,24,0,0,0,0,0,0
826 DATA 16,32,96,184,42,60,8,16,0,0,0,
0,0,62,8,20,20,8,62,0,0,0,0,0,8,4,6,2
9,84,60,16,8,0,0,0,0
827 DATA 60,24,255,219,255,153,24,60,6
0,24,255,109,255,153,24,60,60,24,255,1
82,255,153,24,60
830 POKE 53256,1:POKE 53257,3:POKE 532
58,1:POKE 53259,1:POKE 53260,85
835 FOR A=PB+384 TO PB+1024:POKE A,0:N
EXT A:RT=PEEK(106)-8
836 POKE 623,17:X=121:X1=X:X2=X:POKE 5
3248,X:POKE 53250,X:POKE 53251,X
840 POKE 88,0:FOR N=0 TO 8:POKE 89,RT+
N:?"":NEXT N:POKE 106,RT:GRAPHICS 5:
POKE 559,0
850 PICT=ADR("hhLhKhNhMhhOhh*hhP
$01M-K@Pw%KeP-KfL%MeO-MfNJP")
")
865 POKE 704,98:POKE 705,36:POKE 706,5
0:POKE 707,196:POKE 708,232:POKE 709,2
50:POKE 710,132:POKE 711,24
870 D$(1)="p*****
**0*****UUUUUUUUUU0UUUUUUUUUU
UUUUUUUUUU"
871 D$(81)-"*****-2z*****UUUUUU
Um*yUUUUUUUUUUUUUUUUUUUUyUUUUUUUU
UUUUUUUUUU"
872 D$(161)-"j*)*****

```

08

ATTACK ON THE DEATH STAR

```

B+224)
960 POKE 53277,3:POKE 559,46:Y=56:D-US
R(ADR(ES),PB+512+Y,PB+84)
980 SIZEC=0:SM=0:SIZEL=0:N=0:Y3=38:SI
ZEH=0:FF=1:LV=1:POKE 656,1:POKE 657,0:P
RINT "BEGINNER"
990 IF PEEK(53279)=3 THEN LV=1:POKE 65
6,1:POKE 657,0:PRINT "BEGINNER"
1000 IF PEEK(53279)=5 THEN LV=2:POKE 6
56,1:POKE 657,0:PRINT "EXPERT"
1010 IF PEEK(53279)=6 THEN GOTO 1030
1020 GOTO 990
1030 POKE 656,1:POKE 657,0:PRINT "
1040 SOUND 0,10,8,3:P0=PB+512:P1=PB+64
0:P2=PB+768:P3=PB+896:P4=PB+394:BLANK=
524
1050 SH=84:SH1=BLANK:SH2=BLANK:SH3=BLA
NK:RETURN
1500 GRAPHICS 2+16:POSITION 3,4:PRINT
#6;"DOWN THE TRENCH"
1510 POSITION 0,5:PRINT #6;"attack on
the death":POSITION 7,6:PRINT #6;"star
":POSITION 2,7:PRINT #6;"BY DAVID PLOT
KIN"
1520 POSITION 0,8:PRINT #6;"*****
*****":POSITION 1,9:PRINT #6;"ple
ase wait 20 SEC."
1530 SOUND 2,55,12,8:FOR WW=1 TO 5
1540 FOR W=10 TO 100 STEP 2:SOUND 0,W,
10,8:SOUND 1,110-W,10,8:POKE 708,W:NEX
T W:NEXT WW
1550 SOUND 2,0,0,0:SOUND 1,0,0,0:SOUND
0,0,0,0
1560 FOR W=10 TO 255:POKE 710,W:SOUND
0,W,10,6:NEXT W:SOUND 0,0,0,0:RETURN

```

TYPO TABLE

Variable checksum = 2394018

Line num	range	Code	Length
5	- 120	PO	532
130	- 170	GA	554
175	- 207	YP	542
210	- 230	WT	503

GAMES

240	—	265	YD	548
270	—	305	DT	506
310	—	610	RS	603
615	—	645	ZM	504
650	—	670	KG	543
673	—	684	AW	509
685	—	705	PQ	608
710	—	727	VO	557
728	—	740	ME	569
745	—	805	VO	562
806	—	821	OP	594
825	—	836	JL	528
840	—	871	WW	525
872	—	877	LT	574
880	—	910	PV	520
920	—	960	JH	541
980	—	1040	UZ	538
1050	—	1540	NV	520
1550	—	1560	VM	209

Speed Demon

*A slick way to circumvent
Player/Missile programming.*

Speed Demon! is a one-player car-racing game using a joystick in Port One. You are given a high-performance stock car which leaks oil every now and then. At the start your car is warming up at the gate, waiting for the count-down. In the test window you see a prompt for the skill level you wish. There are two levels: pressing [1] starts you at the beginner's level; pressing [2] starts you at the pro level, in which your car has a severe oil leak. Once the count reaches zero, "Yer Off!" Your objective is to circle the entire course three times in as little time as possible.

Avoid hitting the bales of hay that line the course, and the oil slicks, left behind by your car. These cause your car to spin out. You can only resume driving when your car has regained traction.

To start the game, press [START] and get ready to burn rubber!!

by John Magdziarz

System Requirements: 16K RAM, joystick

```

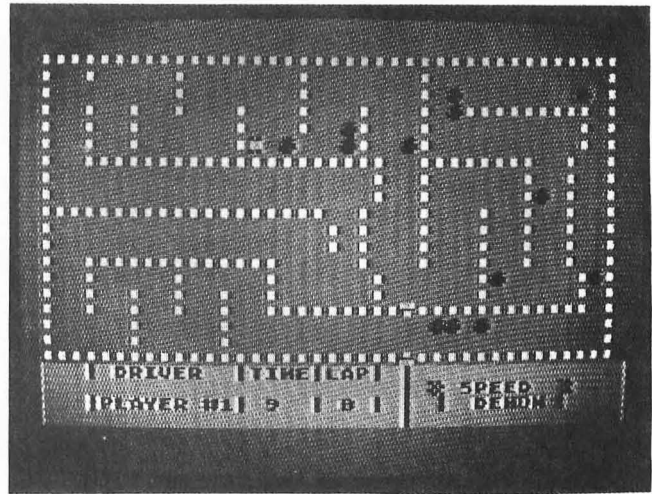
80 REM ***** SPEED DEMON *****
90 REM BY JOHN MAGDZIARZ 1982
100 GRAPHICS 1+16
110 POSITION 7,8:? #6:"ANTIC"
120 POSITION 6,10:? #6:"presents"
130 POSITION 4,12:? #6:"speed demon!"
140 FOR LOOP=1 TO 1000:NEXT LOOP
150 POSITION 3,16:? #6:"please wait..." : F0
R T=1 TO 600:NEXT T
160 GOTO 180
170 FOR LOOP=1 TO 10:NEXT LOOP:RETURN
180 GRAPHICS 0:POKE 559,0:REM SHUT OFF

```

```

SCREEN TEMPORARILY
190 OPEN #1,4,0,"K":REM OPEN KEYBOARD
FOR DIRECT
200 A=PEEK(106):POKE 106,A-5
210 CP=PEEK(106)+1:POKE 756,CP
220 CHAR=CP*256
230 FOR M=0 TO 1023
240 POKE CHAR+M,PEEK(57344+M):REM COPY
CHAR SET FROM ROM TO RAM
250 NEXT M
260 FOR NC=1 TO 9:READ OLD
270 DIF=(OLD+4+64)*8
280 FOR M=0 TO 7:READ LINE:REM POKE DA
TA FOR ALTERED CHARACTERS SET
290 POKE CHAR+DIF+M,LINE:NEXT M
300 NEXT NC:REM DATA FOR NEW CHARACTER
S
310 DATA 1,136,136,255,255,255,255,136
,136
320 DATA 2,60,60,190,190,60,60,190,190
330 DATA 3,34,34,191,191,191,191,34,34
340 DATA 4,190,190,60,60,190,190,60,60
350 DATA 9,0,0,20,20,20,20,0,0
360 DATA 10,204,204,51,51,204,204,51,5
1
370 DATA 11,255,85,255,170,60,20,60,0
380 DATA 12,0,60,20,60,170,255,85,255
390 DATA 13,8,42,168,42,170,170,168,40
400 GRAPHICS 0:POKE 756,CP:POKE 559,0
410 DL=PEEK(560)+PEEK(561)*256:REM ALT
ER DISPLAY LIST
420 POKE DL+3,68
430 FOR I=6 TO 23:POKE DL+I,4:NEXT I
440 POKE DL+28,4
450 FOR I=0 TO 4:READ COL:REM SET SCRE
EN COLORS, REGISTERS 0-4
460 POKE 708+I,COL:NEXT I
470 DATA 26,0,198,72,5
480 ? "☒":DIM HOR$(40),VERT$(30):DIM C
R1$(4)
490 POKE 559,0:? "☒":RESTORE 550:S1=7
500 FOR I2=1 TO 39:HOR$(I2,I2)=CHR$(13
):NEXT I2

```

```

510 FOR I=1 TO 30 STEP 3:VERT$(I,I)=CHR$(13):VERT$(I+1,I+1)=CHR$(29):VERT$(I+2,I+2)=CHR$(30):NEXT I
520 POKE 82,0
530 REM DRAW PLAYFIELD
540 FOR LOOP=1 TO 8:READ I,J,K,L:POSITION I,J: ? HOR$(K,L):NEXT LOOP
550 DATA 0,0,1,39,28,3,1,9,3,6,1,20,25,6,1,8,0,9,1,19,3,12,1,13,15,15,1,22,0,18,1,39
560 FOR LOOP=1 TO 27:READ I,J,K,L:POSITION I,J: ? VERT$(K,L):NEXT LOOP
570 REM DATA FOR PLAYFIELD
580 DATA 0,0,1,30,0,10,1,22,3,1,1,1,3,3,1,12,6,3,1,9,9,1,1,9
590 DATA 13,3,1,9,17,1,1,12,21,3,1,9,25,1,1,28,25,11,1,6,29,9,1,18
600 DATA 32,7,1,18,36,4,1,6,35,6,1,21,36,13,1,6,38,1,1,30
610 DATA 38,11,1,24,19,10,1,6,15,13,1,6,12,14,1,12,9,13,1,9
620 DATA 6,14,1,12,3,13,1,9,22,7,1,6,21,9,1,12,22,13,1,6
630 POSITION 3,19: ? " | DRIVER | TIME | L

```


GAMES

```
AP|"
640 POSITION 3,20:? " "
650 POSITION 3,21:? " |PLAYER #1| 0 |
0|"
660 POSITION 3,22:? " "
670 POSITION 24,19:? CHR$(124);CHR$(29
);CHR$(30);CHR$(124);CHR$(29);CHR$(30)
;CHR$(124);CHR$(29);CHR$(30);CHR$(124)
680 POSITION 26,20:? CHR$(14);" SPEED
";CHR$(14)
690 POSITION 26,21:? CHR$(2);" DEMON
";CHR$(22)
700 POSITION 24,15:? CHR$(15):POSITION
24,18:? CHR$(16)
710 FOR I=1 TO 4:CR1$(I,I)=CHR$(I+132)
:NEXT I:LAP1=-1:OK=0:AG=0:TIME1=0:LP=1
720 POSITION 23,16:? CR1$(1,1):X1=23:Y
1=16
730 POKE 559,34:POKE 752,1:POKE 53279,
0
740 SOUND 0,170,4,4
750 POSITION 26,19:? "WHAT LEVEL?";:GE
T #1,LEV
760 IF LEV=49 THEN SC=45:GOTO 790
770 SC=10
780 IF LEV<>1 AND LEV<>50 THEN 750
790 POSITION 26,19:? " "
800 REM
810 GOSUB 1280
820 FOR G=5 TO 1 STEP -1:POSITION 23,2
1:? G:SOUND 0,40,10,4:FOR T=1 TO 100:N
EXT T:SOUND 0,0,0,0:GOSUB 170:NEXT G
830 POSITION 23,21:? " "
840 SOUND 0,85,6,5
850 POKE 20,0:POKE 19,0:REM RESET REAL
TIME CLOCK
860 REM START OF GAME ROUTINE
870 POSITION 15,21:? PEEK(19):IF PEEK(
19)>98 THEN 480
880 IF PEEK(53279)=6 THEN 490
890 IF OK1=1 THEN 870
900 IF AG=1 THEN 1200
910 S=STICK(0):IF S=15 OR S=10 OR S=6
```

```

OR S-5 OR S-9 THEN S-S1
920 S1-S
930 REM READ STICK
940 ON S-4 GOTO 960,970,980,0,990,1000
,1010,0,1030,1040
950 GOTO 910
960 GOTO 870
970 GOTO 870
980 X1L-X1+1:Y1L-Y1:P1-1:GOTO 1070
990 GOTO 870
1000 GOTO 870
1010 X1L-X1-1:Y1L-Y1:P1-3:IF (X1-24 OR
X1-25) AND (Y1-16 OR Y1-17) THEN 870
1020 GOTO 1070
1030 X1L-X1:Y1L-Y1+1:P1-4:GOTO 1050
1040 X1L-X1:Y1L-Y1-1:P1-2
1050 IF (S-14 OR S-13) AND (X1-24) AND
(Y1-16 OR Y-17) THEN S-7:GOTO 940
1060 REM CHECK COLLISIONS
1070 LOCATE X1L,Y1L,Z1:POSITION X1L,Y1
L:PUT #6,Z1:LX-X1:LY-Y1
1080 IF Z1-17 THEN POSITION X1,Y1:? "
":X1-X1L:Y1-Y1L
1090 IF Z1<>32 THEN 1170
1100 REM MOVE CAR
1110 POSITION X1,Y1:? " ":POSITION X1L
,Y1L:? CR1$(P1,P1):IF P1<>LP THEN GOSU
B 1350
1120 L0-P1:X1-X1L:Y1-Y1L
1130 IF X1-24 AND (Y1-16 OR Y1-17) THE
N LAP1=LAP1+1:SOUND 2,50,12,10:GOSUB 1
70:POSITION 20,21:? LAP1
1140 SOUND 2,0,0,0
1150 IF LAP1-3 THEN POSITION X1,Y1:? "
":SOUND 0,0,0,0:GOSUB 1290:GOTO 640
1160 GOSUB 1230:GOTO 870
1170 K=INT(RND(0)*3+1)
1180 FOR I=14 TO 0 STEP -2:SOUND 1,100
,0,I:GOSUB 170:NEXT I
1190 FOR G=1 TO K:FOR I=1 TO 4:POSITIO
N X1,Y1:? CR1$(I,I):AG=1:GOTO 870
1200 NEXT I:NEXT G:AG=0:POSITION X1,Y1
:? CR1$(P1,P1)

```

```

1210 GOTO 910
1220 REM OIL SLICK PLACEMENT ROUTINE
1230 I=INT(RND(0)*SC+1)
1240 IF I=7 THEN GOSUB 1260
1250 RETURN
1260 LOCATE LX,LY,Z:IF Z=32 THEN POSIT
ION LX,LY:? CHR$(17):RETURN
1270 POSITION LX,LY:PUT #6,Z:RETURN
1280 RETURN
1290 POSITION 26,20:? "GAME OVER!":POS
ITION 25,21:? " "
1300 POKE 53279,0
1310 S2=PEEK(53279):IF S2<>6 THEN 1310
1320 GOTO 490
1330 RETURN
1340 REM CAR TURNING SOUND
1350 SOUND 2,5,0,8:SOUND 3,2,10,8:FOR
X=1 TO 10:NEXT X:SOUND 2,0,0,0:SOUND 3
,0,0,0
1360 RETURN

```

TYP0 TABLE

Variable checksum - 1065147

Line num	range	Code	Length
80	- 190	QP	480*
200	- 310	JG	394*
320	- 430	XV	436*
440	- 550	OY	568*
560	- 660	BX	514*
670	- 730	PS	543*
740	- 850	AF	516*
860	- 970	NF	403
980	- 1090	XZ	462*
1100	- 1190	QA	524*
1200	- 1310	FT	353*
1320	- 1360	GX	211*

Bats

The objects of **Bats** is to fly your bat through a cavern while avoiding the walls and eating insects. You score points for every insect eaten. Pressing the fire button causes your bat to fly higher, releasing it causes the bat to fall. Your bat always flies steadily forward. You start over after you either score 300 points or you lose your bats. You lose all points if you hit a stalactite. There are poison bugs, the color of your bat. Eat one of these and your bat dies, you lose all points, and 100 penalty points are deducted.

The cavern narrows as the game progresses. You get a bonus bat for every 1000 points, with four bonus bats maximum. The game ends when all bats are dead.

by Stan Ockers

System requirements: 16 RAM, joystick

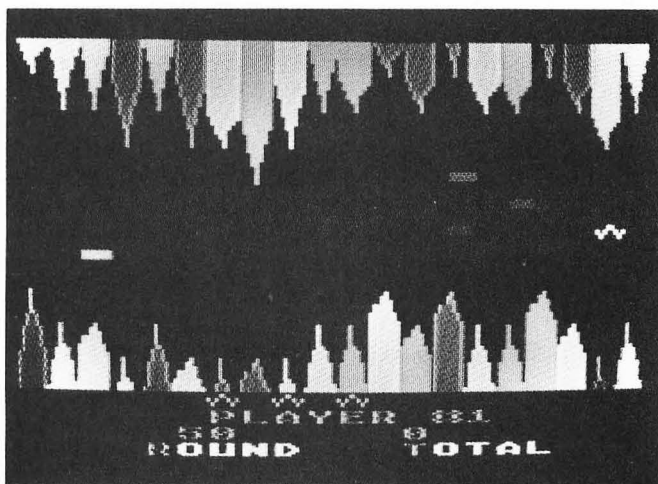
```

10 REM ***** BATS *****
20 REM BY STAN OCKERS 3-82
30 DIM ZZ$(32):FOR I=1 TO 32:READ A:ZZ
$(I)=CHR$(A):NEXT I:GOSUB 1240:CLR
40 DATA 104,104,133,204,104,133,203,10
4,133,206,104,133,205,162,4,160,0
50 DATA 177,203,145,205,136,208,249,23
0,204,230,206,202,208,240,96
60 TRAP 60:? "# PLAYERS "':POKE 764,25
5:INPUT NP
70 REM ** PM GRAPHICS **
80 DIM D$(1),F$((INT(ADR(D$)/1024)+1)*
1024-ADR(D$)-1),PM$(384),M$(128),P$(12
8),MM$(8)
90 RESTORE 100:FOR I=1 TO 8:READ A:MM$
(I)=CHR$(A):NEXT I
100 DATA 3,3,12,12,48,48,192,192
110 PM$=CHR$(0):PM$(384)=CHR$(0):PM$(2

```

GAMES

```
)=PM$:M$=PM$:P$=M$
120 REM ** MISSILE COLORS **
130 POKE 704,14:POKE 705,39:POKE 706,5
4:POKE 707,70
140 REM ** VBI ROUT. TO MOVE MISSILES
**
150 FOR I=1536 TO 1566:READ A:POKE I,A
:NEXT I
160 DATA 104,160,14,162,6,169,7,76,92,
228,90,120,150,180,162,3,222,10,6,189,
10,6,157,4,208,202,16,244,76,98,228
170 REM ** BAT IMAGES **
180 DIM BATDN$(5):BATDN$=P$:FOR I=2 TO
4:READ A:BATDN$(I,I)=CHR$(A):NEXT I
190 DATA 24,165,66
200 DIM BATUP$(5):BATUP$=P$:FOR I=2 TO
4:READ A:BATUP$(I,I)=CHR$(A):NEXT I
210 DATA 66,165,24
220 POKE 54279,ADR(PM$)/256:POKE 559,4
6:POKE 53277,3:POKE 623,4:A=USR(1536)
230 REM ** STALACTITES AND STALAGMITE
S **
240 DIM C$(42),U$(42):C$="*****$%*****
*&":U$="")("*****":FOR I=1 TO 14
:C$(I+14)=CHR$(ASC(C$(I))-32)
250 U$(I+14)=CHR$(ASC(U$(I))-32):C$(I+
28)=CHR$(ASC(C$(I))+128):U$(I+28)=CHR$
(ASC(U$(I))+128):NEXT I
260 DIM P(NP),SCORE(NP),TOTAL(NP),BN(N
P),BONUS(NP)
270 W=7:P=0:POKE 82,0
280 FOR I=1 TO NP:SCORE(I)=0:TOTAL(I)=
0:BN(I)=3:BONUS(I)=1000:NEXT I:NXTCV=3
00*NP
290 REM ** CHANGE WIDTH OF CAVERN **
300 IF W>3 THEN W=W-1
310 GOSUB 660
320 P=P+1:IF P>NP THEN P=1
330 IF BN(P)=0 THEN 320
340 M$=PM$:FOR I=0 TO 3:M$(YST+5*W+W*
(3-I))-MM$(2*I+1,2*I+2):NEXT I
350 REM ** MAIN LOOP **
360 POKE 656,1:POKE 657,22:? " Pull J
```



```

oystick"
370 IF STICK(0)<>13 THEN 370
380 REM ** SCORECARD **
390 ? CHR$(125):GOSUB 1080:POKE 656,0:
POKE 657,26:? "PLAYER #":P
400 POKE 656,1:POKE 657,24:? "Round
Total":GOSUB 1100
410 POKE 53248,30:YPOS=YST+20:POKE 532
78,0:T=0:DIS=12
420 FOR XPOS=47 TO 200:POKE 53248,XPOS
:IF STRIG(0)=0 THEN YPOS=YPOS-1:P$(YPO
S)=BATUP$
430 IF STRIG(0)=1 THEN YPOS=YPOS+1:P$(
YPOS)=BATDN$
440 IF PEEK(53256)>0 THEN POKE 1546,0:
TOTAL(P)=TOTAL(P)-100:GOTO 570
450 IF PEEK(53257)>0 THEN POKE 1547,0:
GOSUB 1130
460 IF PEEK(53258)>0 THEN POKE 1548,0:
GOSUB 1130
470 IF PEEK(53259)>0 THEN POKE 1549,0:
GOSUB 1130
480 IF PEEK(53252)>0 THEN 570
490 NEXT XPOS:P$=PM$
500 IF SCORE(P)<300 THEN 410

```

```

510 TOTAL(P)-TOTAL(P)+SCORE(P):SCORE(P)
  )-0:GOSUB 1110
520 IF TOTAL(P)>BONUS(P) AND BN(P)<4 T
HEN BONUS(P)-BONUS(P)+1000:BN(P)-BN(P)
+1:GOSUB 1080:DIS-10:T-30:GOSUB 790
530 FOR I=1 TO 30:GOSUB 1120:FOR J=1 T
O 30:NEXT J:GOSUB 1100:NEXT I
540 IF P=NP THEN 300
550 GOTO 320
560 REM ** LOSE A BAT **
570 DIS-10:T-9:GOSUB 790
580 YPOS=YPOS+1:P$(YPOS)-BATDN$:POKE 5
3278,0:SOUND 1,YPOS,10,10:IF PEEK(5325
2)-0 THEN 580
590 GOSUB 800:P$-PM$:SCORE(P)-0:BN(P)-
BN(P)-1:GOSUB 1080:IF BN(P)-0 THEN POK
E 656,0:POKE 657,6:?" ":GOSUB 830
600 GOSUB 1090:FOR I=1 TO NP:IF BN(I)>
0 THEN 530
610 NEXT I:GOSUB 1150:GRAPHICS 17:POSIT
ION 5,2:?" #6:"Game Over":FOR I=1 TO N
P:POSITION 3,2+2*I:?" #6:"Player #":I;
620 ? #6:?" - ":TOTAL(I):NEXT I:POSITIO
N 3,23:?" #6:"PRESS ANY KEY":
630 FOR I=1 TO 300:NEXT I:GOSUB 820:IF
FL=0 THEN 630
640 GOTO 270
650 REM ** DRAW CAVERN **
660 GOSUB 1150:GRAPHICS 2:GOSUB 1170:P
OKE 77,0
670 DL=INT(RND(0)*(8-W))+1:YST=8*(DL+
1)
680 FOR X=0 TO 19:GOSUB 770:Y=0:FOR I=
R+7-DL TO R+6:POSITION X,Y:?" #6:C$(I,I
):Y=Y+1:NEXT I
690 FOR I=1 TO W:POSITION X,Y:?" #6:?"
:Y=Y+1:NEXT I
700 IF DL+W>-10 THEN Y=Y-1:POSITION X,
Y:?" #6:?"':GOTO 720
710 GOSUB 770:FOR I=R TO R+9-DL-W:POSIT
ION X,Y:?" #6:U$(I,I):Y=Y+1:NEXT I
720 IF DL<-1 THEN DL=2:GOTO 750
730 IF DL>-10-W THEN DL=9-W:GOTO 750

```



```

740 DL=DL+INT(RND(0)*3)-1
750 NEXT X
760 RETURN
770 R=INT(RND(0)*6)*7+1:RETURN
780 REM ** SOUND SUBR'S **
790 FOR I=15 TO 0 STEP -1: SOUND 0,I,DI
S,I:FOR J=1 TO T:NEXT J:NEXT I:RETURN
800 FOR I=10 TO 2 STEP -2: SOUND 0,RND(
0)*255,8,I: SOUND 1,RND(0)*255,8,I:FOR
J=1 TO 30:NEXT J:NEXT I
810 SOUND 0,0,0,0: SOUND 1,0,0,0: RETURN

820 RESTORE 1050: LS=30: LL=5: GOSUB 840:
RETURN
830 RESTORE 1000: LS=20: LL=10
840 FL=0
850 READ I,J: IF I=3 THEN RETURN
860 IF I=0 THEN 890
870 IF PEEK(53775)<255 THEN FL=1: RETUR
N
880 SOUND 0,I,10,10: SOUND 1,I-2,10,6
890 FOR I=1 TO J: FOR K=1 TO LS: NEXT K:
NEXT I: SOUND 0,0,0,0: SOUND 1,0,0,0
900 FOR I=1 TO LL: NEXT I: GOTO 850
910 RESTORE 1010: LS=12: LL=12: GOSUB 840
920 IF FL=1 THEN RETURN
930 RESTORE 1030: GOSUB 840
940 IF FL=1 THEN RETURN
950 RESTORE 1010: GOSUB 840
960 IF FL=1 THEN RETURN
970 RESTORE 1040: GOSUB 840
980 IF FL=1 THEN RETURN
990 FOR I=1 TO 300: NEXT I: GOTO 910
1000 DATA 243,4,243,4,243,1,243,4,204,
4,217,1,217,4,243,1,243,4,255,1,243,6,
3,3
1010 DATA 243,1,217,1,204,1,182,1,162,
1,204,1,162,1,0,1,173,1,217,1,173,1,0,
1,182,1,230,1,182,1,0,1
1020 DATA 243,1,217,1,204,1,182,1,162,
1,204,1,162,1,121,1,3,3
1030 DATA 136,1,162,1,204,1,162,1,136,
4,3,3

```

GAMES

```
1040 DATA 162,1,204,1,162,1,121,1,243,
4,3,3
1050 DATA 81,4,85,2,102,1,108,1,121,6,
108,1,102,1,81,2,81,2,85,2,102,1,108,1
,121,8
1060 DATA 108,2,91,2,102,2,108,2,121,1
,128,1,121,1,108,1,102,2,121,2,81,4,10
2,4,121,8,3,3
1070 REM ** SUBR. TO INDICATE BATS LE
FT **
1080 POKE 656,0:POKE 657,6:? " ";
:POKE 657,6:FOR I=1 TO BN(P):? "+ ";N
EXT I:RETURN
1090 POKE 656,1:POKE 657,5:? SCORE(P);
" ":RETURN
1100 POKE 656,1:POKE 657,12:? TOTAL(P)
;" ":RETURN
1110 POKE 656,1:POKE 657,5:? " ":
RETURN
1120 POKE 656,1:POKE 657,12:? " "
:RETURN
1130 GOSUB 790:POKE 53278,0:SCORE(P)=S
CORE(P)+25:GOTO 1090
1140 REM ** SUBR. TO REMOVE PM GR. **
1150 POKE 53277,0:POKE 54272,0:FOR I=5
3261 TO 53264:POKE I,0:NEXT I:RETURN
1160 REM ** SUBR. TO INSERT PM GR. **
1170 POKE 53277,3:POKE 559,46:START=(P
EEK(106)+1):POKE 756,START
1180 REM ** ALTER DISPLAY LIST **
1190 A=PEEK(560)+256*PEEK(561)
1200 IF PEEK(A)<>66 THEN A=A+1:GOTO 12
00
1210 POKE A,70:POKE A+3,6:POKE A+4,6:P
OKE A+5,6
1220 RETURN
1230 REM ** CHANGE CHARACTER SET **
1240 POKE 106,PEEK(106)-5:GRAPHICS 0:S
TART=(PEEK(106)+1)*256:POKE 756,START
/256:POKE 752,1
1250 ? "INITIALIZING ....."
1260 A=USR(ADR(ZZ$),57344,START):RESTO
RE 1290
```

```

1270 READ X:IF X=-1 THEN RESTORE :RETU
RN
1280 FOR Y=0 TO 7:READ Z:POKE X+Y+STAR
T,Z:NEXT Y:GOTO 1270
1290 DATA 32,255,255,127,127,126,62,62
,60
1300 DATA 40,60,28,28,24,8,8,8,8
1310 DATA 48,255,127,126,60,56,24,8,8
1320 DATA 56,8,24,28,124,124,254,254,2
55
1330 DATA 64,60,126,126,126,126,126,12
7,255
1340 DATA 72,16,16,16,16,16,24,60,60
1350 DATA 80,255,255,255,255,255,255,2
55,255
1360 DATA 88,0,24,24,165,165,66,66,0
1370 DATA -1

```

TYPO TABLE

Variable checksum = 1034021

Line num	range	Code	Length
10	- 100	DF	522
110	- 200	EF	518
210	- 280	OX	551
290	- 400	UG	518
410	- 500	BE	502
510	- 590	KF	550
600	- 680	LG	582
690	- 800	FH	606
810	- 910	TW	514
920	- 1030	YJ	444
1040	- 1110	HK	509
1120	- 1210	MZ	520
1220	- 1330	GD	463
1340	- 1370	EY	112

BONUS GAMES

Tie-Fighter

Tie-Fighter is a game for the ATARI 400 or 800 computer requiring Atari BASIC 16K for cassette or 24K for disk system. The object of the game is to destroy the Tie-Fighters before they destroy you and your Rebel Base. You have a pilot's perspective into space, with cross-hairs in the middle of the screen. An enemy Tie-Fighter appears and approaches. By moving the joystick you can maneuver to bring the enemy into your sights. Push the button to fire at him. If you don't hit the Tie-Fighter directly it may take several shots to bring him down. If he manages to escape your pursuit (go off the screen) he will join other successful fighters attacking your base. If 10 Tie-Fighters get past you, your base is destroyed.

This game has four levels of difficulty, but as you get better the game speed will continue to increase even beyond level four. You start out the game with 40 units of energy. Each time you fire, the energy will go down one unit. When the energy runs out or when 10 Tie-Fighters have escaped, the game will be over. For every 10 points (or hits), your energy will be refueled to 40 units again and all misses will be cleared. There is a time limit for you to destroy the Tie-Fighters. When the timer runs out, the Tie-Fighter on the screen will be cleared and it counts as if you had let the Tie-Fighter escape. Pushing the space bar while you are playing will freeze the game. Pushing the space bar again will enable you to continue where you left off.

Notice the smooth animation of the Tie-Fighter. This is due to the fast string-handling ability of Atari BASIC. The PM\$ is defined to be the Player/Missile Base. When a player needs to be moved, all the program has to do is to assign the player's data(PO\$) to the PM\$.

Variables:

A — check collision flag.

C — sound counter.

E — the units of energy.

H — horizontal movement flag.

HARDERS — level of difficulty.

MI — misses (Tie-Fighters escaped).
 PO — location of player within Player/Missile Base.
 PO\$ — Player 0 data.
 PM\$ — Player/Missile Base.
 R — random number.
 S — STICK (0).
 SC — score or hits.
 T — units of time.
 V — vertical movement flag.
 X — horizontal coordinate of Player 0.
 Y — vertical coordinate of Player 0.

by Jimmy and Tommy Sa

System Requirements: 16K RAM, joystick

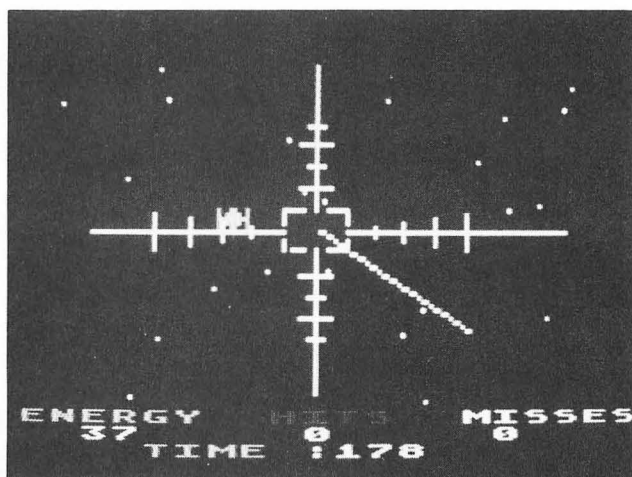
```

10 REM ***TIE-FIGHTER***
20 REM BY JIMMY AND TOMMY SA
30 GOSUB 1040:GOSUB 250:GOSUB 400:GOTO
  110
40 REM PLOT LASER ROUTINE
50 SOUND 0,1,4,6:FOR I=1 TO 4 STEP 3:C
OLOR I:PLOT 0,89:DRAWTO 75,37:PLOT 159
,89:DRAWTO 75,37:NEXT I:E=E-1
60 POKE 53278,15:POKE 53249,0
70 IF A>0 THEN SOUND 0,0,0,0:GOSUB 540
80 IF E=0 THEN POP:POKE 53248,0:GOTO
  990
90 ? "Hitenergy HITS MISSES": " "E:"
  "SC:" "MI:" "? " time :
  " "
100 REM MAIN LOOP
110 POKE 77,0:POKE 53248,X:PM$(P0+Y,P0
+Y+7)-P0$:SOUND 0,100,24,4*(S<>15):PO
KE 709,110-(R<0.1)*110
120 SOUND 1,20,8,1:C=C+1:T=T-1:R=RND(0
):Y=Y+V:X=X+H*HARDERS:S=STICK(0)
130 IF T<1 THEN GOSUB 750:GOTO 110
140 IF C>6-HARDERS THEN SOUND 1,230,10
,4:C=0

```

BONUS GAMES

```
150 ? "␣" time :";T;" "
160 IF X>204 OR X<47 OR Y>90 OR Y<14 T
HEN GOSUB 750
170 IF STRIG(0)=0 THEN POKE 53249,119:
A=PEEK(53260):GOSUB 50:SOUND 0,0,0,0
180 V--(S=13 OR S=5)+(S=14 OR S=10)
190 H--(S=7 OR S=6)+(S=9 OR S=11)
200 IF V=0 THEN V--(R>0.98)+(R<0.97)
210 IF H=0 THEN H--(R>0.98)+(R<0.97)
220 IF PEEK(764)<>255 THEN GOSUB 810
230 GOTO 110
240 REM DRAW PLAYFIELD
250 GRAPHICS 7:POKE 752,1:POKE 82,0
260 POKE 708,106:SETCOLOR 2,6,0:COLOR
1
270 A=PEEK(560)+PEEK(561)*256+4
280 IF PEEK(A)<>66 THEN A=A+1:GOTO 280
290 POKE A,70:POKE A+3,6:POKE A+4,6:PO
KE A+5,6
300 RESTORE
310 FOR I=1 TO 28:READ A,R:PLOT A,R:RE
AD A,R:DRAWTO A,R:NEXT I
320 DATA 75,0,75,33,75,42,75,76,78,42,
83,42,83,42,83,40,83,38,139,38,67,42,6
7,40,67,38,19,38,67,36,67,33,67,33
330 DATA 72,33,78,33,83,33,83,33,83,36
,67,42,72,42,113,34,113,42,105,41,105,
35,97,36,97,40,90,39,90,37,59,37
340 DATA 59,39,52,40,52,36,44,35,44,41
,35,42,35,34,71,48,79,48,77,53,73,53,7
1,58,79,58,73,63,77,63
350 DATA 77,14,73,14,71,18,79,18,73,23
,77,23,71,28,79,28,72,28
360 COLOR 2:FOR A=1 TO 10:PLOT RND(0)*
159,RND(0)*89:NEXT A
370 COLOR 1:FOR A=1 TO 20:PLOT RND(0)*
159,RND(0)*89:NEXT A
380 RETURN
390 REM SET UP PLAYER-MISSILE GRAPHICS
400 DIM PM$(2048),P0$(7):X=70:Y=20
410 A=ADR(PM$)
420 PMBASE=INT(A/1024)*1024
430 IF PMBASE<A THEN PMBASE=PMBASE+102
```



```

4
440 S=PMBASE-A
450 POKE 559,46:POKE 54279,PMBASE/256:
POKE 704,102:POKE 53277,3
460 PM$=CHR$(0):PM$(2048)=CHR$(0):PM$(
2)=PM$
470 RESTORE 490
480 FOR I=1 TO 7:READ A:P0$(I,I)=CHR$(
A):NEXT I
490 DATA 0,153,189,255,189,153,0
500 P0=S+512:PM$(P0+Y,P0+Y+7)=P0$:POKE
53248,X:PM$(P0+183,P0+183)=CHR$(28)
510 GOSUB 660
520 RETURN
530 REM EXPLOSION
540 SOUND 1,0,0,0
550 FOR I=1 TO 30:POKE 704,RND(0)*200+
14:NEXT I
560 FOR I=1 TO 30:POKE 704,RND(0)*200+
14:PM$(P0+Y+I/3,P0+Y+I/3)=CHR$(RND(0)*
20)
570 SOUND 0,200+I,8,16-I/2:SOUND 1,9*I
,16,16-I/2:SOUND 2,200+I,8,15:NEXT I
580 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND

```

BONUS GAMES

```
2,0,0,0
590 PM$(P0+Y,P0+Y+12)="-♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥":
REM 12 HEART-CHARACTERS
600 SC=SC+1
610 IF SC/10<>INT(SC/10) OR SC=0 THEN
640
620 HARDERS=HARDERS+1:E=40:MI=0
630 FOR I=1 TO 60:POKE 53279,RND(0)*2:
NEXT I
640 X=INT(RND(0)*100+70):Y=20:PM$(P0+Y
,P0+Y+7)=P0$:POKE 704,102:POKE 53248,0
650 REM WARNING ROUTINE
660 ? "⌨ RED ALERT"
670 FOR R=1 TO 2:FOR I=1 TO 2:SOUND 0,
116*I,10,8:FOR A=1 TO 20:NEXT A:SETCOL
OR 4,4,6
680 FOR A=1 TO I*28:NEXT A:NEXT I
690 SOUND R,0,0,0:FOR A=1 TO 50:NEXT A
:NEXT R
700 T=100+HARDERS*10+100*(HARDERS-1):V
-1:SETCOLOR 4,0,0:SOUND 0,0,0,0
710 ? "⌨energy HITS MISSES";" ";E;"
";SC;" ";MI:"? " time ":";"
..
720 FOR I=1 TO 100:NEXT I
730 SOUND 1,24,8,1:POKE 53248,X:RETURN

740 REM ESCAPE ROUTINE
750 SOUND 0,0,0,0:SOUND 1,0,0,0:MI=MI+
1
760 PM$(P0+Y,P0+Y+12)="-♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥":
REM 12 HEART-CHARACTER
770 FOR I=1 TO 10:POKE 53279,RND(0)*1:
NEXT I
780 IF MI=10 THEN POKE 53248,0:POP :GO
TO 920
790 GOSUB 640:SOUND 1,20,8,1:RETURN
800 REM STOP ROUTINE
810 SOUND 0,0,0,0:SOUND 1,0,0,0:POKE 7
64,255
820 IF PEEK(764)-255 THEN 820
830 SOUND 1,20,8,1:POKE 764,255:RETURN
```



```

840 REM GAME OVER ROUTINE
850 SOUND 0,0,0,0:POKE 53248,0:POKE 53
278,15
860 ? :? " score :";SC:?
870 ? " press trigger to play again!":
POP
880 POKE 77,128:X=USR(ADR(RAINBOW$),1)
890 IF STRIG(0)-1 THEN 880
900 RUN
910 REM REBEL BASE DESTROYED
920 ? " rebel base is under a
ttack!!!"
930 COLOR 4:FOR I=0 TO 80:PLOT 0,I:DRA
WTO 159,I:POKE 712,(RND(0)<0.5)*50:SO
UND 0,I*6,8,4:NEXT I
940 SOUND 0,0,0,0:FOR I=1 TO 3
950 FOR A=0 TO 30:SOUND 0,A*8,8,20-A/
2+I:SOUND 1,A*8,16,20-A/2+I:POKE 712,R
ND(0)*255:NEXT A
960 NEXT I:SETCOLOR 4,0,0
970 GRAPHICS 17:POKE 87,0:POKE 82,2
980 ? :? " rebel base is overran...
":? :? " all is lost!!":GOTO 850
990 SOUND 1,0,0,0:SOUND 0,0,0,0
1000 ? " you DON'T have ANY energy t
o fight!!!"
1010 FOR I=1 TO 200:NEXT I
1020 GOTO 920
1030 REM SELECT ROUTINE
1040 GRAPHICS 17:DIM RAINBOW$(32)
1050 HARDERS=1:E=40:SC=0:MI=0:V=1:H=1
1060 RESTORE 1080
1070 FOR I=1 TO 32:READ A:RAINBOW$(I,I)
)=CHR$(A):NEXT I
1080 DATA 104,104,104,72,162,57,160,0,
173,0,210,101,20,141,25,208,141,10,212
1090 DATA 136,208,242,202,208,237,104,
56,233,1,208,228,96
1100 POSITION 2,3:? #6:" level - "
1110 POSITION 2,15:? #6:"press":? #6
1120 ? #6:" select for level"
1130 ? #6:" start to play"
1140 SOUND 0,254,10,10:SOUND 1,255,10,

```


BONUS GAMES

```

10
1150 POSITION 4,10:? #6:"<tie-fighter>
"
1160 POSITION 11,3:? #6:HARDERS:A=PEEK
(53279)
1170 IF A=6 THEN T=100+HARDERS*10+100*
(HARDERS-1):SOUND 0,0,0,0:SOUND 1,0,0,
0:RETURN
1180 IF A=5 THEN HARDERS=HARDERS+1:IF
HARDERS>4 THEN HARDERS=1
1190 A=USR(ADR(RAINBOW$),1)
1200 GOTO 1160
1210 DATA 202,189,1,6,157,2,6
1220 DATA 224,0,208,245,173,61,6
1230 DATA 141,1,6,162,0,189,10,6,157,9
,6,232,224,7,208,245,173,62,6,141,16,6
,169,0
1240 DATA 141,60,6,238,60,6,104,170,10
4,76,98,228

```

TYPO TABLE

Variable checksum = 306808

Line num	range	Code	Length
10	- 90	FP	500
100	- 170	DY	547
180	- 290	CH	527
300	- 360	EL	500
370	- 480	HR	436
490	- 570	BE	502
580	- 670	YQ	607
680	- 760	TL	528
770	- 870	GZ	515
880	- 960	KO	530
970	- 1080	QI	530
1090	- 1170	MM	513
1180	- 1240	QQ	265

Tin Pan Alley Cats

After you've worked hard all day and programmed all night, the last thing you need is to have your few hours of sleep disturbed by alley cats howling on your back fence. There are three of the buggers—a green one, a white one and a pink one. Every now and then one jumps up where you can hit it with a tin can, if you're quick enough.

That's the scenario for **Tin Pan Alley Cats**, a one-player game requiring joystick and 16K RAM. You start out with 25 tin cans that you can kick toward the fence by pressing the fire button. But first you must move horizontally along the fence to lineup under the cats as they appear and disappear at random. The pink cat is the fastest, and hitting it scores the most points. The green cat is slowest, and hitting it yields the least. The cats will appear 35 times during a game, and the pace quickens as you use up your cans. When you hit a cat you will see it and hear it, and points will be added to your score.

If you score 2,000 points, you get five extra cans, and the cats appear five extra times. The bonus is repeated if you reach 3,000 and 4,000 points. The high score of your session is saved after each round. The difficulty of the game can be adjusted by changing the value of TUF in lines 280-310.

Thanks go to Stan Ockers for his ideas on vertical blank interrupts (ANTIC, June 1982). We modified the VBI into a fast joystick routine.

We also thank Jerry White for his ideas on sounds and the ATARI (ANTIC, October 1982).

by Rick Bloom and Rob Glassman

System Requirements: 16K RAM, joystick

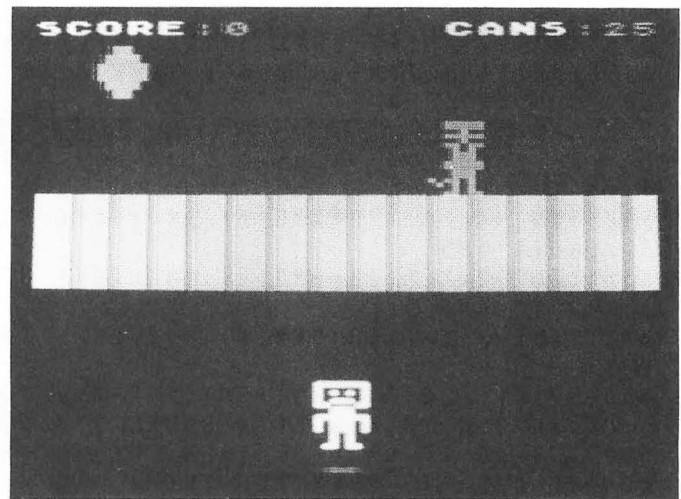
10 REM ***** TIN PAN ALLEY CATS **

20 REM BY RICK BLOOM AND ROB GLASSMAN
1983
50 ? "☐":POKE 752,1

BONUS GAMES

```
100 DIM P0$(1),P1$(1),P2$(1),P3$(1),MS
(1)
110 VTAB=PEEK(134)+256*PEEK(135):ATAB
-PEEK(140)+256*PEEK(141)
120 REM **** VBI JOYSTICK ROUTINE ***
130 FOR A=1536 TO 1582:READ ML:POKE A,
ML:NEXT A
140 DATA 104,160,10,162,6,169,7,76,92,
228,173,120,2,201,7,240,9,173,120,2,20
1,11,240,13,208
150 DATA 18,230,209,166,209,142,3,208,
144,9,176,7,198,209,166,209,142,3,208,
76,98,228
155 GOSUB 870:GOSUB 960
160 GRAPHICS 17:POSITION 4,10:? #6;"ge
t ready !!!"
170 FOR X=1 TO 600:NEXT X
180 BONUS=0:HSCOR=0:X=USR(1536):GOTO 1
250
190 REM CAT HIT SOUND
200 FOR X=40 TO 20 STEP -5
205 FOR J=704 TO 706:POKE J,X:NEXT J
210 SOUND 1,X,12,10
220 FOR Z=1 TO 10:NEXT Z
230 NEXT X
240 FOR Y=15 TO 55 STEP 5
245 FOR J=704 TO 706:POKE J,Y:NEXT J
250 SOUND 1,Y,12,10
260 FOR A=1 TO 8:NEXT A
270 NEXT Y
280 SOUND 1,0,0,0:RETURN
290 REM ***** SONG ROUTINE *****
300 IF X=-1 THEN SOUND 0,0,0,0:SOUND 1
,0,0,0:FOR W=1 TO 100:NEXT W:RETURN
310 READ X,Y,Z:IF X=0 THEN 340
320 IF Y=0 THEN SOUND 1,0,0,0
330 GOTO 380
340 IF Y=0 THEN SOUND 1,0,0,0
350 POKE 540,Z
360 IF PEEK(540)<>0 THEN 360
370 GOTO 300
380 POKE 540,Z
390 Z=PEEK(540):IF Z=0 THEN 410
```

TIN PAN ALLEY CATS



```
400 SOUND 0,X,10,12:SOUND 1,Y,10,8:GOT
0 390
410 SOUND 0,0,0,0:GOTO 300
420 DATA 81,162,6,0,162,3,85,162,6,0,1
62,3,96,217,6,0,217,3,102,217,6,0,217,
3,108,162,6,0,162,3,102,162,6,0,162,3
430 DATA 96,217,12,0,217,6,108,162,6,0
,162,3,108,162,3,102,162,6,0,162,3,96,
217,6,0,217,3,91,217,6,0,217,3
440 DATA 85,173,30,0,0,9,0,173,3,81,17
3,6,0,217,3,85,173,6,0,173,3,96,217,6,
0,217,3,102,217,6,0,173,3,108,173,6
450 DATA 0,173,3,102,173,6,0,173,3,96,
217,12,0,173,3,108,173,6,0,173,3,108,1
73,3,102,173,6,0,173,3,96,217,6
460 DATA 0,217,3,85,217,6,0,217,3,81,1
62,26,-1,-1,0
470 REM *** MISSILE MOVE ROUTINE ***
480 Z=USR(MOVE,M+YM3+DYM3,DM3,LM3,63):
Z=USR(MOVE,M+YM3+LM3+DYM3,B,-DYM3,63)
490 YM3-YM3+DYM3:XM3-XM3+DXM3:DXM3-0:D
YM3-0:RETURN
500 REM *** THE MAIN LOOP!!! ***
510 REM
```

BONUS GAMES

```
520 YM3-93:XM3-0:POKE 53278,1:POKE 532
51,PEEK(209):POKE 53255,0:TIME-0:SETCO
LOR 4,0,INT(CNS/5)
530 REM RANDOMLY POKE CAT TO SCREEN
540 A=(3*RND(0))+1:ON A GOTO 550,560,
570
550 TIME-1:X=(150*RND(0))+45:POKE 5324
8,X:GOTO 580
560 TIME-2:X=(150*RND(0))+45:POKE 5324
9,X:GOTO 580
570 TIME-3:X=(150*RND(0))+45:POKE 5325
0,X
580 N=INT((20*RND(0))+20):FOR X-1 TO 2
5:SOUND 3,N,10,8:NEXT X:SOUND 3,0,0,0
590 TUF-25:IF CNS<-20 THEN TUF=TUF-2
600 IF CNS<-15 THEN TUF=TUF-4
610 IF CNS<-10 THEN TUF=TUF-6
620 IF CNS<-5 THEN TUF=TUF-8
630 GOSUB 750
640 POKE 53248,0:POKE 53249,0:POKE 532
50,0
650 POKE 704,72:POKE 705,14:POKE 706,1
84
660 FOR X-1 TO 150:NEXT X:GOSUB 1070:IF
CNS=0 THEN 1100
670 KIT-KIT-1:IF KIT=0 THEN 1100
680 POKE 77,0:GOTO 510
689 REM *** COLLISION ROUTINE ***
690 REM
700 GOSUB 200:D3$=D02$:P3$(YP3,YP3+LP3
-1)=D3$:M$="":M$(128)=M$:M$(2)=M$
710 ON TIME GOTO 720,730,740
720 SCOR=SCOR+200:RETURN
730 SCOR=SCOR+100:RETURN
740 SCOR=SCOR+50:RETURN
749 REM *** FIRE MISSILE ***
750 FOR Z-1 TO TUF:RB=PEEK(209)+14
760 IF STRIG(0)=0 THEN D3$=D03$:P3$(YP
3,YP3+LP3-1)=D3$:SOUND 0,60,12,12:M$(Y
M3,YM3+LM3-1)=DM3$:GOTO 820
770 D3$=D02$:P3$(YP3,YP3+LP3-1)=D3$
780 ON TIME GOTO 790,800,810
790 FOR X-1 TO 5:NEXT X:NEXT Z:RETURN
```

```

800 FOR X=1 TO 10:NEXT X:NEXT Z:RETURN
810 FOR X=1 TO 20:NEXT X:NEXT Z:RETURN

820 MMR=1:CNS=CNS-1:SOUND 0,0,0,0
825 IF RB>-210 OR RB<-40 THEN RETURN
830 POKE 53255,RB:DYM3--6:GOSUB 470:IF
  PEEK(53259)<>0 THEN POKE 53255,0:GOTO
  690
840 MMR=MMR+1:IF MMR=20 THEN 770
850 GOTO 830
860 REM *** TITLE SCREEN ***
870 GRAPHICS 2+16:POSITION 1,4:? #6;"T
  IN CAN ALLEY CATS":POSITION 0,8:? #6;"
  by rick BLOOM"
880 POSITION 2,9:? #6;" AND rob glass
  man"
890 C1=8:C2=166:C3=86:C4=52:C5=0:CNT=1
900 POKE 708,C1:POKE 709,C2:POKE 710,C
  3:POKE 711,C4:POKE 712,C5
910 CNT=CNT+1:IF CNT=10 THEN 930
920 TEMP=C1:C1=C2:C2=C3:C3=C4:C4=TEMP:
  FOR X=1 TO 100:NEXT X:GOTO 900
930 RESTORE 420:GOSUB 300
940 RETURN
950 REM *** TITLE SCREEN #2 ***
960 GRAPHICS 18:POSITION 0,1:? #6;"sta
  rring...":POSITION 1,3:? #6;"GREENSLEE
  VES":POSITION 8,4:? #6;"...50 POINTS"
970 POKE 708,72:POKE 709,198:POKE 710,
  120:POKE 711,12:POKE 712,0
980 POSITION 1,5:? #6;"FRISKY WHITE":P
  OSITION 8,6:? #6;"...100 POINTS":POSITI
  ON 1,7:? #6;"PINK PANTHER"
990 POSITION 7,8:? #6;"...200 POINTS"
1000 RESTORE 420:GOSUB 310
1010 POSITION 0,10:? #6;"AND you AS...
  CAN CAN"
1020 RESTORE 420:GOSUB 310:RETURN
1030 REM *** DISPLAY LIST ***
1040 GRAPHICS 21
1050 ST=PEEK(560)+PEEK(561)*256+4:POK
  E ST-1,70:POKE ST+48,65:POKE ST+49,PEE

```

BONUS GAMES

```
K(560):POKE ST+50,PEEK(561)
1060 NON-PEEK(559):POKE 559,0:CNS-25
1070 POKE 87,1:POSITION 0,0:? #6:"score":SCORE
1075 IF SCORE>-2000 AND BONUS-0 THEN GO
SUB 2000
1076 IF SCORE>-3000 AND BONUS-1 THEN GO
SUB 2100
1077 IF SCORE>-4000 AND BONUS-2 THEN GO
SUB 2200
1080 IF CNS<10 THEN POSITION 13,0:? #6
:"cans:0":CNS:RETURN
1090 POSITION 13,0:? #6:"cans:":CNS:RE
TURN
1100 REM *** END OF GAME ***
1110 POKE 209,0:POKE 53251,0
1115 POKE 53277,0:POKE 559,2
1130 GRAPHICS 2+16:POSITION 6,2:? #6;"
meow !!!":POSITION 2,4:? #6:"FINAL SCO
RE ":SCORE
1140 NU=SCORE
1150 IF NU>HSCORE THEN HSCORE=NU:FOR X=1
TO 5:POSITION 0,6:? #6:"new high scor
e ":HSCORE:FOR W=1 TO 50:NEXT W:GOTO 11
70
1160 GOTO 1195
1170 POSITION 0,6:? #6:"NEW HIGH SCORE
":HSCORE:FOR W=1 TO 50:SOUND 2,W,10,10
:NEXT W
1180 POSITION 0,6:? #6:"new high score
":HSCORE:FOR W=1 TO 50:NEXT W:POSITION
0,6:? #6:"NEW HIGH SCORE ":HSCORE
1190 FOR W=1 TO 50:SOUND 2,51-W,10,10:
NEXT W:NEXT X:SOUND 2,0,0,0:GOTO 1200
1195 POSITION 2,6:? #6:"HIGH SCORE ":
HSCORE
1200 POSITION 1,8:? #6:"press trigger
for"
1205 POSITION 4,10:? #6:"another game"
1210 IF STRIG(0)=0 THEN SCORE=0:BONUS=0
:GOSUB 1040:POKE 53277,3:GOTO 1800
1220 GOTO 1210
1230 REM *** COUNTDOWN! ***
```



```

1240 GRAPHICS 18:POSITION 10,6:7 #6:N:
RETURN
1249 REM *** SET UP PMG STRINGS ***
1250 GRAPHICS 21
1260 PMBASE=PEEK(106)-12:POKE 54279,PM
BASE:PMBASE-PMBASE*256
1270 N=3:GOSUB 1240
1280 POKE 623,17:POKE 704,72:POKE 705,
14:POKE 706,184:POKE 707,138:POKE 5325
6,1:POKE 53257,1
1290 POKE 53258,1:POKE 53259,1:POKE 53
260,64:POKE 53248,0:POKE 53249,0:POKE
53250,0:POKE 209,120:POKE 53251,0
1300 FENCE=40:SIZE=17
1310 DIM D0$(SIZE)
1320 RESTORE 1330:FOR I=1 TO SIZE:READ
BYTE:D0$(I,I)-CHR$(BYTE):NEXT I
1330 DATA 248,248,32,248,32,216,80,126
,31,31,18,18,18,18,18,18,54
1340 DIM D1$(SIZE)
1350 RESTORE 1360:FOR I=1 TO SIZE:READ
BYTE:D1$(I,I)-CHR$(BYTE):NEXT I
1360 DATA 248,248,32,248,32,216,80,112
,248,249,115,118,124,120,80,80,80
1370 DIM D2$(SIZE)
1380 RESTORE 1390:FOR I=1 TO SIZE:READ
BYTE:D2$(I,I)-CHR$(BYTE):NEXT I
1390 DATA 62,62,8,62,8,54,20,28,28,62,
62,28,28,180,84,20,54
1400 YP3=101:LP3=16
1410 N=2:GOSUB 1240
1420 DIM D3$(LP3)
1430 RESTORE 1440:FOR I=1 TO LP3:READ
BYTE:D3$(I,I)-CHR$(BYTE):NEXT I
1440 DATA 254,130,170,130,186,198,56,5
6,254,186,186,186,186,40,40,108
1450 YM3=93:LM3=4:REM VERTICAL POSITIO
N AND LENGTH OF MISSILE3
1460 DIM DM3$(LM3)
1470 RESTORE 1480:FOR I=1 TO LM3:READ
BYTE:DM3$(I,I)-CHR$(BYTE):NEXT I:DM3=A
DR(DM3$)
1480 DATA 192,192,192,192

```


BONUS GAMES

```
1490 OFFSET=PMBASE+512-ATAB
1500 FOR I=0 TO 4
1510 V3=INT(OFFSET/256):V2=OFFSET-256*
V3
1520 POKE VTAB+2,V2:POKE VTAB+3,V3
1530 POKE VTAB+4,128:POKE VTAB+5,0
1540 POKE VTAB+6,128:POKE VTAB+7,0
1550 VTAB=VTAB+8:OFFSET=OFFSET+128
1560 IF I=3 THEN OFFSET=PMBASE+384-ATA
B
1570 NEXT I
1580 P0$(FENCE,FENCE+SIZE-1)=D0$
1590 P1$(FENCE,FENCE+SIZE-1)=D1$
1600 P2$(FENCE,FENCE+SIZE-1)=D2$
1610 P3$(YP3,YP3+LP3-1)=D3$
1620 M$(YM3,YM3+LM3-1)=DM3$
1630 FOR I=1 TO LM3:X=YM3+I-1:M$(X,X)=
CHR$(ASC(M$(X,X))+ASC(DM3$(I,I))):NEXT
I
1640 DIM B$(17):FOR I=1 TO 17:B$(I,I)=
CHR$(0):NEXT I:B=ADR(B$)
1650 N=1:GOSUB 1240
1659 REM MACHINE LANGUAGE MOVE ROUTINE
1660 DIM MOVES$(38):MOVE=ADR(MOVES$):M=A
DR(M$)-1
1670 RESTORE 1690
1680 FOR I=1 TO 37:READ BYTE:MOVES$(I,I
)=CHR$(BYTE):NEXT I
1690 DATA 104,104,133,204,104,133,203,
104,133,206,104,133,205,104,104,133,20
7,104,104,133,208
1700 DATA 160,0,177,203,37,208,113,205
,145,203,200,196,207,208,243,96
1710 GOSUB 1040
1720 DIM D01$(16),D02$(16),D03$(16)
1730 D01$=D3$
1740 RESTORE 1780
1750 FOR I=1 TO 16:READ BYTE:D02$(I,I)
=CHR$(BYTE):NEXT I
1760 FOR I=1 TO 16:READ BYTE:D03$(I,I)
=CHR$(BYTE):NEXT I
1770 D3$=D02$:P3$(YP3,YP3+LP3-1)=D3$
1780 DATA 254,130,170,130,186,198,56,5
```

```

6,254,186,186,186,40,40,40,108
1790 DATA 254,130,170,130,186,198,56,5
6,254,57,57,63,63,32,32,224
1800 REM *** SET UP SCREEN ***
1810 POKE 87,5:COLOR 3:FOR X=19 TO 29:
PLOT 0,X:DRAWTO 79,X:NEXT X
1820 COLOR 1:PLOT 10,2:DRAWTO 12,2:PLO
T 9,3:DRAWTO 13,3:PLOT 8,4:DRAWTO 14,4
:PLOT 8,5:DRAWTO 14,5:PLOT 9,6
1830 DRAWTO 13,6:PLOT 10,7:DRAWTO 12,7
1840 COLOR 2:FOR X=5 TO 75 STEP 5:PLOT
X,19:DRAWTO X,29:NEXT X:SETCOLOR 4,0,
4:SCOR=0:KIT=35
1850 POKE 711,86:POKE 53277,3:POKE 209
,120:POKE 559,46:GOTO 510
2000 CNS=CNS+5:KIT=KIT+5:BONUS=1
2005 FOR X=1 TO 5:SOUND 0,50,10,10:FOR
Y=1 TO 10:NEXT Y:SOUND 0,0,0,0:NEXT X
2010 FOR Z=1 TO 20:NEXT Z:RETURN
2100 CNS=CNS+5:KIT=KIT+5:BONUS=2
2105 FOR X=1 TO 5:SOUND 0,50,10,10:FOR
Y=1 TO 10:NEXT Y:SOUND 0,0,0,0:NEXT X
2110 FOR Z=1 TO 20:NEXT Z:RETURN
2200 CNS=CNS+5:KIT=KIT+5:BONUS=3
2205 FOR X=1 TO 5:SOUND 0,50,10,10:FOR
Y=1 TO 10:NEXT Y:SOUND 0,0,0,0:NEXT X
2210 FOR Z=1 TO 20:NEXT Z:RETURN

```

TYPO TABLE

Variable checksum = 2531540

Line num	range	Code	Length
10	- 155	RT	507
160	- 250	IO	381
260	- 370	VH	416
380	- 450	YZ	594
460	- 550	MN	520
560	- 650	OU	510
660	- 750	QA	397
760	- 850	EJ	507
860	- 960	KJ	641
970	- 1050	VY	552

BONUS GAMES

1060	-	1130	VM	527
1140	-	1195	AD	529
1200	-	1280	LC	520
1290	-	1390	VT	534
1400	-	1510	AV	413
1520	-	1630	QV	375
1640	-	1740	IC	456
1750	-	1820	RQ	526
1830	-	2105	JU	615
2110	-	2210	ZR	222

Drop

Catch the falling faces

You can never beat this game, just get better and better. It starts slowly as faces appear at the top of the playfield and fall towards the bottom. You move a dish laterally with the joystick to catch the faces before they reach the bottom. If you miss one, that ends your turn.

With every successful catch your score increases, and after a while bonus points accrue. However, the speed of the game increases too, and it is unlikely you will ever exceed 1000. A little "falling" sound accompanies the action.

by John Zakour

System Requirements: 16K RAM, joystick

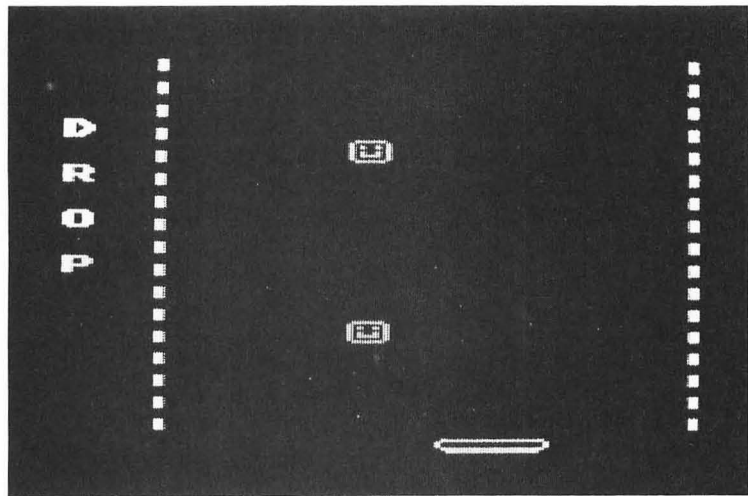
```

1 REM ***** DROP *****
2 REM BY JOHN ZAKOUR
5 GOTO 100
10 IF X=2 THEN X=3:RETURN
20 IF X=15 THEN X=14:RETURN
30 RETURN
100 DIM SA(15)
105 GOSUB 1000:C=1:HI=0
110 FOR N=1 TO 15:SA(X)=0:NEXT N:SA(7)
-1:SA(11)=-1
115 GRAPHICS 1+16:SETCOLOR 4,4,0
116 POSITION 2,5:PRINT #6;"d":POSITION
2,7:PRINT #6;"f":POSITION 2,9:PRINT #
6;"o":POSITION 2,11:PRINT #6;"h"
117 POKE 756,64
120 Y=19:X=10:DL=1
121 FOR PY=2 TO 18
122 POSITION 4,PY:PRINT #6;"&":POSITIO

```

BONUS GAMES

```
N 17,PY:PRINT #6;"8"
123 NEXT PY
124 DLA=10
125 DY=1:DX=INT(12*RND(1)+5)
127 DX2=INT(12*RND(1)+5):DY2=1
130 DX3=INT(9*RND(1)+5):DY3=1
160 POSITION X,Y:PRINT #6;" #5% "
200 S=SA(PEEK(632))
210 IF S<>0 THEN X=X+S:GOSUB 10:POSITI
ON X,Y:PRINT #6;" #5% "
220 POSITION DX,DY:PRINT #6;"1"
230 DL=DL+1
235 IF DL<DLA THEN 200
236 IF C=1 THEN POKE 756,64:C=2:GOTO 2
40
238 POKE 756,68:C=1
240 DL=1:POSITION DX,DY:PRINT #6;" ":D
Y=DY+1:POSITION DX,DY:PRINT #6;"1":POK
E 53762,DY+10:POKE 53763,163
245 IF DY>9 THEN POSITION DX2,DY2:PRIN
T #6;" ":DY2=DY2+1:POSITION DX2,DY2:PR
INT #6;"!"
247 IF DY>15 THEN POSITION DX3,DY3:PRI
NT #6;" ":DY3=DY3+1:POSITION DX3,DY3:P
RINT #6;"!"
250 IF DY<>19 THEN 200
255 IF DX=X+1 OR DX=X+2 OR DX=X+3 THEN
260
257 GOTO 266
260 SC=SC+INT(11-DLA):DLA=DLA-0.35:POS
ITION 1,1:PRINT #6:SC:DX-DX2:DY-DY2:DX
2-DX3:DY2-DY3:GOTO 130
265 DX-DX2:DY-DY2:GOTO 130
266 GOSUB 2000
267 SOUND 0,164,10,8:FOR S=1 TO 150:NE
XT S:SOUND 0,0,0,0:FOR S=1 TO 5:NEXT S
:SOUND 0,164,10,8:FOR S=1 TO 150:NEXT
S
268 SOUND 0,0,0,0:FOR S=1 TO 5:NEXT S:
SOUND 0,217,10,10:FOR S=1 TO 300:NEXT
S:SOUND 0,0,0,0
270 POSITION 6,8:PRINT #6;"game over":
SOUND 1,0,0,0:POKE 77,0
```



```

275 IF SC>HI THEN HI=SC
277 POSITION 5,9:PRINT #6;"hit button"
:POSITION 6,12:PRINT #6;"HIGH ";HI
279 IF STRIG(0)<>0 THEN 270
280 SC=0:GRAPHICS 1+16:SETCOLOR 4,4,0:
GOTO 120
400 GOTO 200
1000 GRAPHICS 2+16:POSITION 6,8:PRINT
#6;"PLEASE":POSITION 7,9:PRINT #6;"WAIT"
1002 POSITION 2,2:PRINT #6;"catch the
falling":POSITION 4,4:PRINT #6;"faces!
!!!!"
1005 FOR I=96 TO 720
1010 POKE 16384+I,PEEK(57344+I)
1020 POKE 17408+I,PEEK(57344+I)
1030 NEXT I
1100 FOR I=16392 TO 16439
1110 READ D
1120 POKE I,D
1130 NEXT I
1140 FOR I=17416 TO 17463
1150 READ D
1160 POKE I,D
1170 NEXT I

```

BONUS GAMES

```

1180 DATA 126,129,165,129,165,189,129,
126
1182 DATA 24,24,24,24,24,24,24,24
1184 DATA 0,0,0, 63,64,64,63,31
1186 DATA 0,0,0,255,0,0,255,255
1188 DATA 0,0,0,252,2,2,252,248
1189 DATA 0,0,0,0,24,24,24,24
1190 DATA 0,0,255,153,255,189,129,255
1192 DATA 0,0,0,0,24,24,24,24
1194 DATA 0,0,0, 63,64,64,63,31
1196 DATA 0,0,0,255,0,0,255,255
1198 DATA 0,0,0,252,2,2,252,248
1199 DATA 24,24,24,24,0,0,0,0
1200 POKE 756,64
1205 SOUND 0,31,10,8:FOR S=1 TO 50:NEX
T S:SOUND 0,0,0,0
1210 RETURN
2000 FOR LD=1 TO 15:SOUND 2,75,8,14:SO
UND 3,76,8,14:POKE 712,PEEK(53770)
2010 POSITION X,Y:PRINT #6;"#$%":FOR S
=1 TO 10:NEXT S:POSITION 4,Y:PRINT #6;
"$% "
2020 NEXT LD
2025 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND
2,0,0,0:SOUND 3,0,0,0:SETCOLOR 4,4,0
2030 RETURN
3000 REM THANKS:PAM C,JB,JB,ED,DAVID,
LARRY,& COREY

```

TYPO TABLE

Variable checksum = 286608

Line	num	range	Code	Length
1	-	117	IW	468
120	-	220	NP	419
230	-	260	QX	559
265	-	277	TM	565
279	-	1120	RD	428
1130	-	1190	IK	258
1192	-	2025	RP	627
2030	-	3000	HA	54

Fallout

A cycle of birth, labor, and goodbye

You are the man, the only player in this electrolife drama. You have three “lives” to live, and watch out, they go rather quickly! Once play begins you will find yourself confronted by a curtain of falling objects — babies (pink), diamonds (blue), and monsters (green). Your first task is to survive by intercepting babies (each confers an additional life), and then to catch the diamonds (they are worth points). Touching a monster takes one life away, and if you have no more, that’s the end of the game.

In the first wave the diamonds are worth one point each. In the second wave they are worth two points. This progresses up through 10 points at Level 10, after which they stay the same. The action, however, continues to change. The directions, angles and speed of the falling items changes randomly, usually for the worse.

Your score at the end of each wave is augmented by multiplying the number of the level by the number of lives you have left at the time.

by Scott McKissock

System Requirements: 16K RAM, joystick

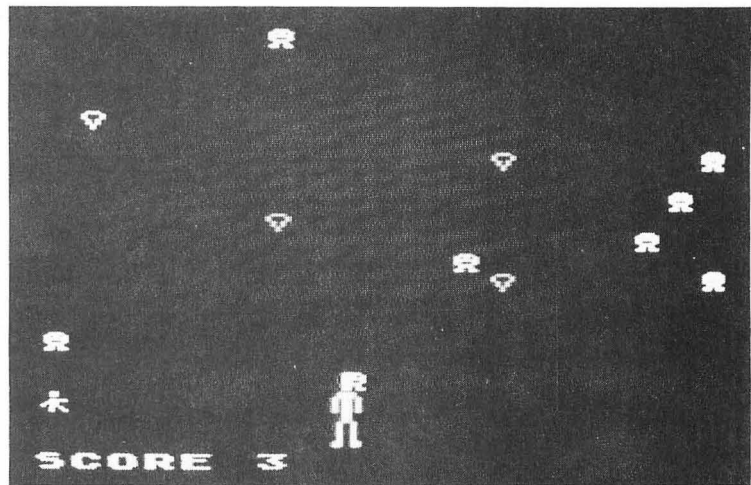
```

1 REM ***** FALLOUT *****
2 REM BY SCOTT MCKISSOCK
10 GOTO 400
49 REM MAIN LOOP
50 D=20+DR:DR=-DR:NS=INT(RND(0)*10)+SN
:FOR SC=1 TO NS:FOR T=1 TO FN:NEXT T
62 S=STICK(0):M=(S=7)*8-(S=11)*8:IF M=
0 THEN 65
63 X=X+M:POKE 53248,X:SOUND 0,255,10,8
:IF PEEK(53252) THEN COL=PEEK(53252):G
OTO 100
65 LB=LB-D:IF LB<0 THEN LB=LB+256:HB=H

```

BONUS GAMES

```
B-1:POKE 559,0:POKE DL,LB:POKE DL+1,HB
:POKE 559,46
70 SOUND 0,0,0,0:POKE DL,LB:IF HB=P106
-1 THEN POKE 559,0:POKE DL+1,HB-1:POKE
DL,248:POKE 559,46:GOTO 300
71 IF PEEK(53252) THEN COL=PEEK(53252)
:GOTO 100
72 OK=0:S=STICK(0):M=(S=7)*8-(S=11)*8:
X=X+M:IF M=0 THEN 77
74 IF X>=200 THEN X=56
75 IF X<=48 THEN X=192
76 POKE 53248,X
77 IF PEEK(53252) THEN COL=PEEK(53252)
:GOTO 100
80 NEXT SC:GOTO 50
100 SOUND 0,99,1,14:IF OK=1 THEN FOR T
=1 TO 30:NEXT T:POKE 53278,0:NEXT SC:G
OTO 50
110 OK=1:POKE 53278,0:REM CLEAR COLISI
ON REGISTER
115 GOTO 120+(20*COL):REM COL=VALUE IN
COLISION REGISTER 0
140 FOR T=15 TO 0 STEP -0.75:POKE 712,
T*16:SOUND 0,T*16,10,16-T:NEXT T:SOUND
0,0,0,0
145 LIVES=LIVES+1-(LIVES=6):POKE (P106
-4)*256+10+LIVES,1
150 NEXT SC:GOTO 50
160 FOR T=255 TO 5 STEP -5:POKE 704,T:
SOUND 0,T,2,15:NEXT T:SOUND 0,0,0,0:PO
KE 704,24
165 LIVES=LIVES-1:POKE (P106-4)*256+11
+LIVES,0
170 IF LIVES=0 THEN 250
175 NEXT SC:GOTO 50
180 GOTO 160
200 FOR T=-125 TO 125 STEP 10:POKE 710
,ABS(T):SOUND 0,ABS(T),10,15:NEXT T:SO
UND 0,0,0,0:POKE 710,148
205 AD=AD+1-(AD=10):SCORE=SCORE+AD:POS
ITION 7,0: ? #6:SCORE
210 POKE 53248,X:NEXT SC:GOTO 50
220 GOTO 140
```



```

240 GOTO 160
250 POKE 87,1:POKE 89,P106-1:POSITION
10,7:? #6;" game over "
255 POSITION 10,10:? #6;" PRESS S
TART "
257 POKE 559,0:POKE DL+1,P106-2:POKE D
L,248:POKE 559,46:REM DISPLAY "GAME OV
ER"
260 POKE DL+26,P106-5:FOR T=255 TO 252
STEP -1:POKE DL+25,T:FOR Y=1 TO 20:NE
XT Y:NEXT T:REM SCROLL SCORE
280 IF PEEK(53279)<>6 THEN 280:REM CHE
CK FOR START
285 SCORE=0:LIVES=3:LV=0:POKE DL+25,0:
POKE DL+26,P106-4:REM LINE UP SCORE
290 POKE 88,0:POKE 89,P106-4:POSITION
0,0:? #6;" score ";SCORE;" !! ":
GOTO 305
300 FOR T=1 TO 500:NEXT T:POKE 77,0
305 LV=LV+1:FN=60-(4*LV)-(LV=1)*60:DR=
1-(LV=1):SN=INT(12-LV*0.5):IF SN<0 THE
N SN=0
365 IF LV=1 THEN 385:REM NO BONUS IF G
AME JUST STARTED
370 FOR BNS=1 TO LIVES:SOUND 0,100-BNS

```

BONUS GAMES

```
*5,10,10:SCORE=SCORE+LV-1:POSITION 7,0
:? #6:SCORE:SOUND 1,90-BNS*5,10,10
375 FOR DLY=1 TO 50:NEXT DLY:SOUND 0,0
,0,0:SOUND 1,0,0,0
380 FOR DLY=1 TO 50:NEXT DLY:NEXT BNS
385 LB=216:HB=P106+15:POKE 559,0:POKE
DL,LB:POKE DL+1,HB:POKE 559,46
390 POKE 87,1:POKE 559,0:POKE 89,P106-
1:POSITION 14,7:? #6:"level ";LV:" com
plete":AD=0:POKE 559,46
395 POSITION 14,10:? #6:"NOW ADDING BO
NUS ":POKE 88,0:POKE 89,P106-4:OK=1:GO
TO 50
400 GRAPHICS 17:POSITION 6,8:? #6:"FaI
LoUT"
410 POSITION 4,12:? #6:"PLEASE WAIT"
420 POSITION 4,14:? #6:"55" SECONDS"
430 FOR R=1 TO 100
450 Z=PEEK(711):POKE 711,PEEK(709):POK
E 709,PEEK(708):POKE 708,Z:REM ROTATE
COLORS
460 N=PEEK(53770):FOR T=4 TO 10:SOUND
0,N,T,15:NEXT T:NEXT R:SOUND 0,0,0,0:R
EM RANDOM TONE
500 POKE 559,0:P106=PEEK(106)-16:FOR T
=(P106-2)*256 TO (P106+16)*256:POKE T,
0:NEXT T:REM CLEAR MEMORY
505 POKE 106,P106:REM MOVE RAMTOP DOWN
16 PAGES
510 CHSET=(PEEK(106)-8)*256:FOR I=0 TO
512:POKE CHSET+I,PEEK(57344+I):NEXT I
:REM MOVE CHARACTER SET INTO RAM
525 CSETP=CHSET/256:READ CHTR:IF CHTR=
-1 THEN GOTO 700
530 FOR I=CHTR*8 TO CHTR*8+7:READ A:PO
KE CHSET+I,A:NEXT I:GOTO 525:REM DRAW
CHARACTER SHAPES
600 DATA 1,12,12,0,15,28,44,10,9
610 DATA 2,14,21,31,17,14,10,27,0
620 DATA 3,14,17,17,10,10,4,4,0,-1
699 REM PUT SHAPES IN MEMORY
700 FOR I=(P106+16)*256 TO (P106*256)+
460 STEP -3:D=RND(0):IF D<0.9 THEN NEX
```

```

T I:GOTO 800
710 IF D>0.99 THEN POKE I,1:BB=BB+1:NEXT I:GOTO 800
715 IF D>0.96 THEN POKE I,131:NEXT I:GOTO 800
720 POKE I,66:NEXT I
799 REM SET UP P/M GRAPHICS
800 PBP=P106-16:POKE 54279,PBP:PMBASE=PBP*256:X=120:Y=92
810 POKE 53277,3:POKE 704,216
820 FOR I=PMBASE+512+Y TO PMBASE+526+Y:READ A:POKE I,A:NEXT I:POKE 53248,X
870 DATA 24,24,24,0,60,90,90,90,24,24,36,36,36,36,102
899 REM PUT IN DISPLAY LIST
900 GRAPHICS 17:POKE 559,46:POKE 756,CHSET/256:DL=PEEK(560)+256*PEEK(561)+4:POKE DL+24,70:POKE DL+25,0
910 POKE DL+26,P106-4:POKE DL+27,65:LB=PEEK(DL):HB=PEEK(DL+1):POKE 88,0:POKE 89,P106-4
920 POKE 88,0:POKE 89,P106-4:GOTO 285

```

TYPO TABLE

Variable checksum = 723906

Line num	range	Code	Length
1	- 70	JK	588
71	- 140	NY	622
145	- 205	SC	526
210	- 285	CJ	558
290	- 375	KS	586
380	- 420	JQ	513
430	- 525	FE	543
530	- 800	LA	506
810	- 920	LW	464

Skull Chase

Watch out for the trees.

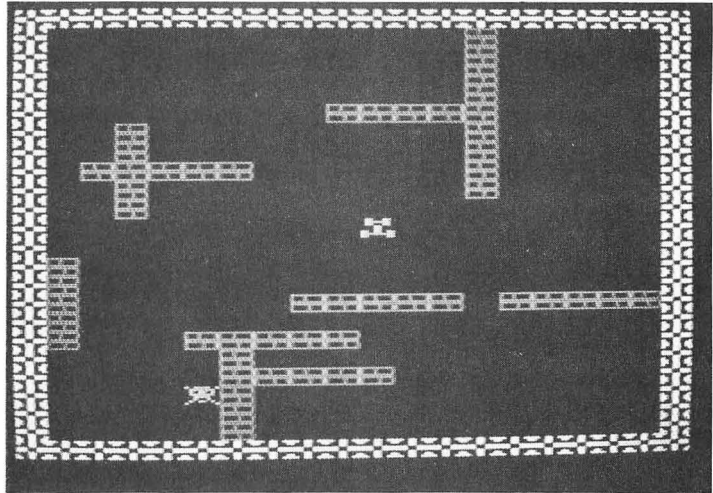
This game exploits the techniques of table lookup to get speed out of BASIC. As the player, you control the race car with a joystick plugged into Port 1. You are supposed to chase the skull, which moves randomly around the playfield to the extent permitted by the walls. When you catch the skull, action is frozen for an instant while you are credited with points, and a *tree* is planted somewhere in the playfield. Then you begin to chase the skull anew.

Obviously, that would be too easy. Now you must avoid the tree, or trees, while chasing the skull. There are two kinds of trees, pine and oak. If you touch a pine tree, you lose points; but if you hit an oak tree, you lose your car and the game is over for that round. Your score—and the high score for this session—will be displayed, and you will be prompted to play again if you wish.

by Dave Miller

System Requirements: 16K RAM, joystick

```
10 REM ***** SKULL CHASE *****
20 REM BY DAVE MILLER  APRIL 1983
100 GRAPHICS 17: DIM TITLES$(12): TITLES$=
"skull  CHASE": POSITION 4,8: FOR X=1 T
0 12
101 ? #6: TITLES$(X,X):: SOUND 0,20-(X*1.
5),8,14: FOR W=1 TO 250: NEXT W: NEXT X: S
OUND 0,0,0,0
110 DIM CHR(15),DX(15),DY(15): SCORE=0:
HIScore=0: XTRA=0
120 DX(14)=0: DX(13)=0: DX(9)=-1: DX(10)=
-1: DX(11)=-1: DX(5)=1: DX(6)=1: DX(7)=1
130 DY(11)=0: DY(7)=0: DY(6)=-1: DY(10)=-
1: DY(14)=-1: DY(5)=1: DY(9)=1: DY(13)=1
140 CHR(14)=92: CHR(13)=93: CHR(11)=94: C
HR(7)=95: CHR(10)=81: CHR(6)=69: CHR(9)=9
0: CHR(5)=67: SKULL=147-64: GOSUB 610
150 ? #6: "☒": POKE 708,196: POKE 709,22:
```



```

POKE 710,52:POKE 711,8
160 POKE 756,CHSET/256:SCREEN=PEEK(88)
+256*PEEK(89):GOSUB 660
170 PX=10:PY=11:POKE SCREEN+PX+20*PY,C
HR(14):DX=1:DY=1
180 BX=INT(16*RND(1)+3):BY=INT(20*RND(
1)+3)
190 REM ***** MAIN LOOP *****
200 ST=STICK(0):IF ST=15 THEN SOUND 2,
0,0,0:GOTO 310
210 SOUND 2,90,6,2
220 TX=PX+DX(ST):TY=PY+DY(ST)
230 CPOS=SCREEN+TX+20*TY
240 IF PEEK(CPOS)=SKULL THEN 420
250 IF PEEK(CPOS)=9 THEN 490
260 IF PEEK(CPOS)=10 THEN SCORE=SCORE-
XTRA:FOR W=1 TO 20:SOUND 3,100,10,14:P
OKE 712,58:NEXT W
270 SOUND 3,0,0,0:POKE 712,0
280 IF PEEK(CPOS) THEN 310
290 CHR=CHR(ST)
300 POKE SCREEN+PX+20*PY,0:POKE CPOS,C
HR:PX=TX:PY=TY
310 TEMPBX=BX+DX:TEMPBY=BY+DY
320 SPOS=SCREEN+TEMPBX+20*TEMPBY

```

BONUS GAMES

```
330 IF NOT PEEK(SPOS) THEN 380
340 SOUND 2,100,10,14:SOUND 2,0,0,0
350 IF RND(1)>0.5 THEN DX=-DX:GOTO 200
360 IF RND(1)>0.5 THEN DY=-DY:GOTO 200
370 DX=-DX:DY=-DY:GOTO 200
380 POKE SCREEN+BX+20*BY,0:POKE SPOS,S
KULL
390 BX=TEMPBX:BY=TEMPBY
400 GOTO 200
410 REM ***** SUCCESS *****
420 SCORE=SCORE+50+XTRA:XTRA=XTRA+25:P
OKE 77,0
430 FOR T=40 TO 10 STEP -10:SOUND 2,7,
10,12:SOUND 2,0,0,0:FOR I=1 TO 15:NEXT
I:NEXT T
440 A=PEEK(709):FOR I=255 TO 0 STEP -5
:POKE 709,I:NEXT I:POKE 709,A
450 S=INT(RND(0)*2)+9
460 R=SCREEN+INT(480*RND(0)):IF PEEK(R
) OR PEEK(R)=SKULL THEN GOTO 460
470 POKE R,S:POKE SCREEN+BX+20*BY,0:PO
KE SCREEN+PX+20*PY,0:GOTO 170
480 REM ***** FAILURE *****
490 SOUND 2,0,0,0
500 FOR I=100 TO 200 STEP 2
510 POKE SCREEN+PX+20*PY,INT(RND(0)*4
)+76:POKE 709,PEEK(53770)
520 SOUND 0,I,6,15-INT((I-100)/7)
530 NEXT I
540 POKE SCREEN+PX+20*PY,0
550 FOR I=150 TO 1 STEP -1:POKE 711,I:
POKE 709,I+16:POKE 708,I+32:POKE 710,I
+64:SOUND 0,I+50,10,8:NEXT I
560 SOUND 0,0,0,0:IF SCORE>=HISCORE TH
EN HISCORE=SCORE
570 GRAPHICS 1+16:POSITION 0,5:? #6:"y
our SCORE WAS ";SCORE:POSITION 0,8:? #6
:"HIGH SCORE IS ";HISCORE
580 POSITION 5,19:? #6:"press fire":IF
STRIG(0)=1 THEN 580
590 SCORE=0:GOTO 150
600 REM *** READ DATA FOR CHSET *****
610 CHSET=(PEEK(106)-8)*256
```



```

620 RESTORE 850:IF PEEK(CHSET+9*8)=56
THEN RETURN
630 READ A:A=A-64:IF A<0 THEN RETURN
640 FOR J=0 TO 7:READ B:POKE CHSET+A*8
+J,B:POKE 708+3*RND(0),PEEK(53770):NEX
T J
650 GOTO 630
655 REM ***** CREATE BRICK WALL*****
660 WALL=210:BWALL=188
670 FOR W=0 TO 5:P=INT(RND(0)*470)
680 FOR I=0 TO 4
690 POKE SCREEN+P+I,BWALL
700 NEXT I:NEXT W
710 FOR W=0 TO 5:P=INT(RND(0)*470)
720 FOR I=0 TO 4
730 POKE SCREEN+P+I*20,BWALL
740 NEXT I:NEXT W
750 REM ***** CREATE BORDER *****
760 FOR I=0 TO 19
770 POKE SCREEN+I,WALL
780 POKE SCREEN+460+I,WALL
790 NEXT I
800 FOR I=0 TO 23
810 POKE SCREEN+I*20,WALL
820 POKE SCREEN+19+I*20,WALL
830 NEXT I:RETURN
840 REM ***** DATA FOR CHSET *****
850 DATA 67,24,28,48,250,223,77,24,12
860 DATA 69,12,24,77,223,250,48,28,24
870 DATA 73,56,254,127,62,8,8,8,255
880 DATA 74,8,28,62,127,62,8,8,8
890 DATA 75,0,32,0,16,8,16,4,0
900 DATA 76,0,64,0,16,68,16,4,0
910 DATA 77,0,64,16,2,0,128,32,2
920 DATA 78,128,8,1,0,0,0,0,64
930 DATA 79,8,0,0,0,0,0,0,0
940 DATA 81,48,24,178,251,95,12,56,24
950 DATA 82,219,153,24,231,231,24,153,
219
960 DATA 83,189,126,90,126,60,36,90,12
9
970 DATA 90,24,56,12,95,251,178,24,48
980 DATA 92,102,126,102,24,24,219,255,

```


BONUS GAMES

195

990 DATA 93,195,255,219,24,24,102,126,
102

1000 DATA 94,15,239,226,94,94,226,239,
15

1010 DATA 95,240,247,71,122,122,71,247,
240

1020 DATA 124,255,145,145,255,255,137,
137,255

1030 DATA -1

TYPO TABLE

Variable checksum = 1142917

Line num	range	Code	Length
10	- 120	RP	528
130	- 160	RA	532
170	- 270	WH	544
280	- 390	BT	362
400	- 490	JT	524
500	- 570	TQ	500
580	- 680	OU	489
690	- 800	EN	246
810	- 920	VR	331
930	- 1030	RH	365

Crystal Caves

Making the most of a tight situation

Navigate your ship through treacherous caves in an attempt to get enough energy pellets to escape the cave you are in. The next cave is harder to escape from. **Crystal Caves** calls for both fast reflexes and strategy!

When you begin the game you will hear a beep. Your ship will appear in the center of the top part of your screen. You move right and left using the joystick. The fire button relocates you randomly at the top of the screen. If your ship touches any of the crystal walls, it will be destroyed. You only get one life.

If you absorb enough of the diamond-shaped energy pellets (by running into them), you will be transported out of the cave you are in. In the new cave will start fresh, but you will have to get more energy pellets to escape. Your score is based on how many caves you have gone through, and also how many extra energy pellets you have absorbed.

When your ship has been destroyed, the game will end and your score will be displayed. The game can be paused at any time by pressing [CTRL] and [I], and you can continue by pressing those keys again.

A good strategy for this game is to go into the largest "corridor" you can when faced with a choice of directions. You should use little jerks on the joystick when in cramped quarters. The fire button on your joystick should be used only if imminent death is certain.

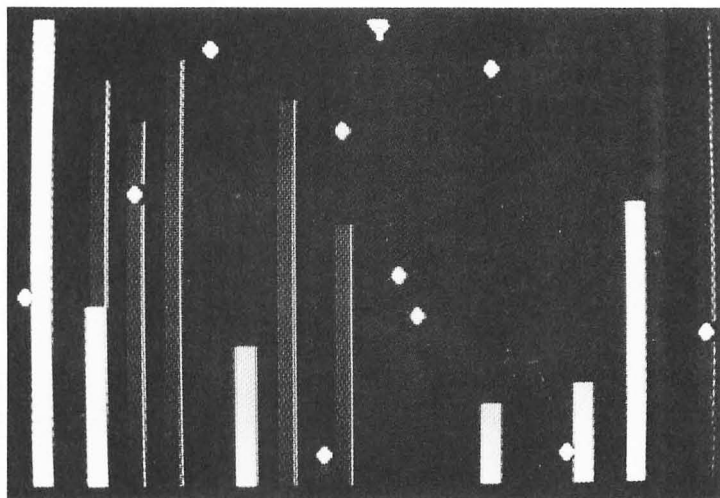
by Thomas Edwards

System Requirements: 16K RAM, joystick

```
5 REM ***** CRYSTAL CAVES *****
10 REM BY THOMAS EDWARDS
20 REM REVISION JUNE 10 1983
40 PRINT "☒":PR=0:ESCAPE=5:C1=0
50 GOSUB 380:DIM C$(4),D$(37):GOTO 320
```

BONUS GAMES

```
60 C$(1,1)-CHR$(33):C$(2,2)-CHR$(34):C
$(3,3)="-"
70 SC=PEEK(88)+256*PEEK(89)
80 RA=53770
90 PROB1=150:PROB2=150
100 REM
110 REM MAIN GAME BLOCK
120 ? :? :? :PRINT "
AVE " :C1+1
130 HW=0
140 P=SC+20:D$="-"
150 D$(1,1)=C$(1,1):FOR I=2 TO 36:D$(I
,I)="-" :NEXT I:D$(37,37)=C$(2,2)
160 FOR I=1 TO 30:PRINT D$:NEXT I
170 PRINT D$:IF PEEK(P)=0 THEN GOTO 22
0
180 IF PEEK(P)=96 THEN PR=PR+1:SOUND 0
,30,12,14:GOTO 200
190 POKE P,3:GOTO 280
200 SOUND 0,0,0,0:IF PR<ESCAPE THEN GO
TO 170
210 POKE P,3:FOR I=255 TO 0 STEP -1:SO
UND 1,I,10,10:NEXT I:SOUND 1,0,0,0:ESC
APE=ESCAPE+1:C1=C1+1:PR=0:GOTO 60
220 POKE P,3:S=PEEK(632):P=P+(S-7)-(S-
11):SOUND 1,(P-SC)*4.35+78,2,2:IF PEEK
(644)=0 THEN GOTO 750
230 IF PEEK(RA)>PROB1 THEN GOTO 260
240 IF PEEK(RA)<PROB2 THEN 170
250 T=PEEK(RA)/7.5+2:W=INT(PEEK(RA)/86
)+1:D$(T,T)=C$(W,W):GOTO 170
260 POKE SC+860+PEEK(RA)/6,96:GOTO 240
270 REM GAME OVER
280 FOR I=1 TO 12:FOR J=CA TO CA+7:POK
E J,PEEK(RA):POKE 710,PEEK(RA):SOUND 1
,PEEK(RA),8,10:NEXT J:NEXT I
290 POKE 710,0:SOUND 1,0,0,0
300 SOUND 0,0,0,0:RESTORE :CA=CA-24:FO
R I=1 TO 10:? :NEXT I:GOSUB 520
310 PRINT "CAVES FINISHED:" :C1=? "EXTR
A ENERGY PELLETS:" :PR
320 ? :? :? "PRESS START TO BE
GIN " :PR=0:ESCAPE=5:C1=0
```



```

330 ? " PRESS OPTION FOR INSTRUCTION
S "
340 IF PEEK(53279)-3 THEN GOSUB 780:GO
TO 320
350 IF PEEK(53279)<>6 THEN 340
360 FOR I9=70 TO 0 STEP -3:SOUND 0,15,
12,I9/5:NEXT I9:GOTO 60
370 REM REDEFINE CHARACTERS
380 X1=PEEK(106)
390 X2=X1-4
400 POKE 106,X2
410 POKE 709,13
420 GRAPHICS 0
430 POKE 710,0:POKE 752,1
440 CR=PEEK(756)*256
450 POKE 756,X2
460 CA=X2*256
470 GOSUB 620
480 FOR N=W TO 1023+W STEP 8
490 POKE CA+N,PEEK(CR+N)
500 NEXT N
510 W=W+1:IF W<8 THEN 480
520 FOR I=1 TO 3:CA=CA+8
530 DATA 170,170,170,170,170,170,170,1
70

```

BONUS GAMES

```

540 DATA 85,85,85,85,85,85,85,85
550 DATA 170,170,255,102,60,24,24,24
560 FOR ADDR=CA TO CA+7
570 READ DAT:POKE ADDR,DAT
580 NEXT ADDR
590 NEXT I
600 RETURN
610 REM TITLE
620 ? :? :?
630 ? "
640 ? "
650 ? "
660 ? "
670 ? :?
680 ? "
690 ? "
700 ? "
710 ? :? :? :?
720 ? " By Thomas Edwards"
730 RETURN
740 REM HYPERWARP ROUTINE
750 IF HW>C1+1 THEN GOTO 230
760 HW=HW+1:POKE P,0:P=SC+3+INT(35*RN
D(1)):GOTO 170
770 REM INSTRUCTIONS
780 ?
790 ? " CRYSTAL CAVES
"
800 ? :? " You have been cursed to
fly "
810 ? "through the Crystal Caves for t
he "
820 ? "rest of your natural life. The
"
830 ? "deeper you go in a cave, the mo
re "
840 ? "treacherous the cave becomes. I
f you"
850 ? "get enough energy pellets, you
can "
860 ? "start fresh on a new cave. Howe
ver,"

```

```

870 ? "each successive cave becomes har
der "
880 ? "to escape from. Use the joysti
ck to"
890 ? "move. The fire button will relo
cate"
900 ? "you randomly at the top of the
"
910 ? "screen. Good luck."
920 ? :? :? :? :RETURN

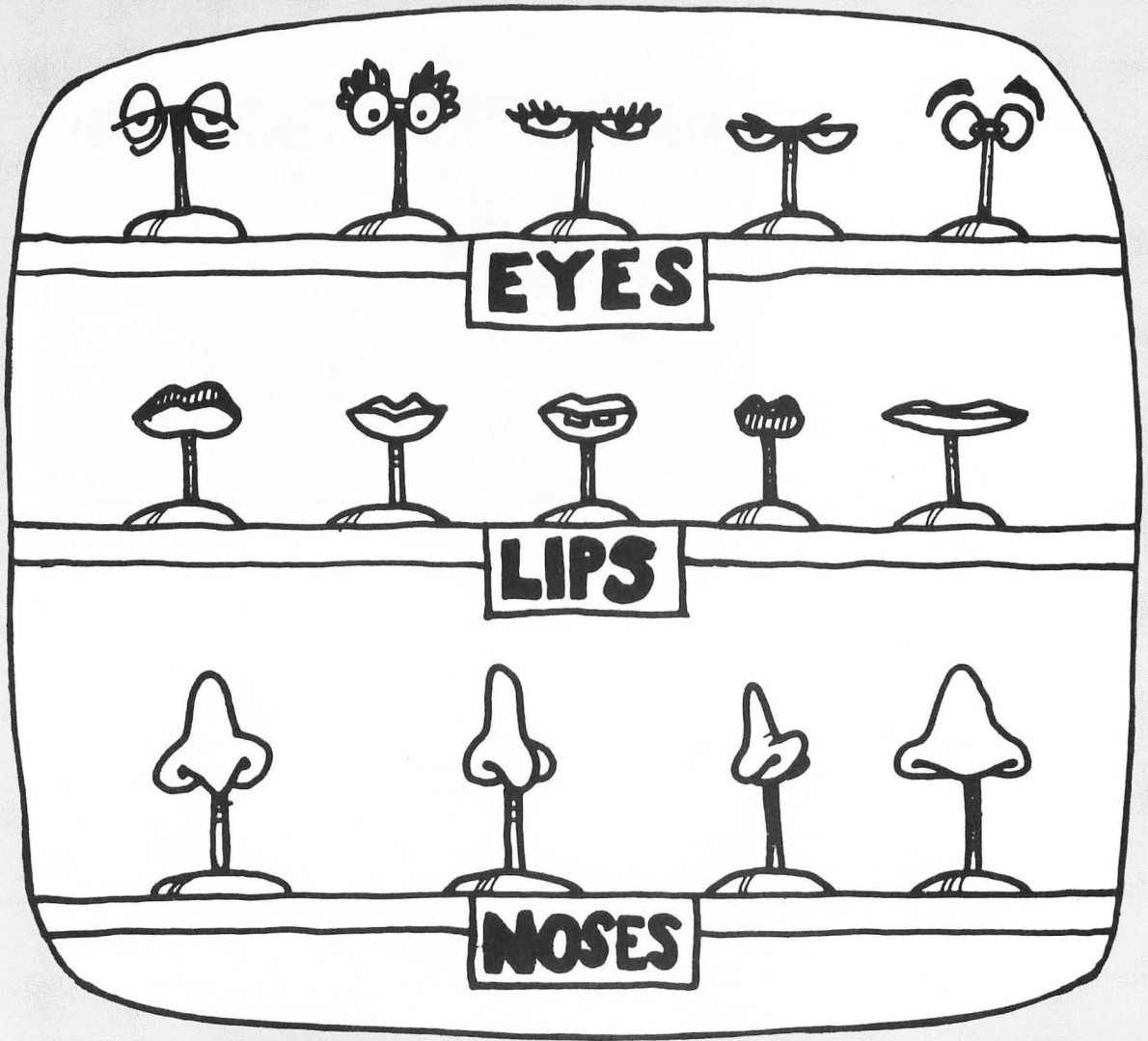
```

TYPO TABLE

Variable checksum - 440382

Line	num	range	Code	Length
5	-	120	ZI	426
130	-	210	YJ	536
220	-	300	SS	572
310	-	420	OA	442
430	-	540	UR	300
550	-	660	KV	237
670	-	780	QY	306
790	-	900	AK	507
910	-	920	PZ	44

Features



done

Translate

Those of you who use your ATARI computers for business applications someday might wish to print checks. It seems like a simple task to write a program that prints the date, amount, and payee, in specific locations on a check form. But who wants to enter the English translation of an amount like ONE THOUSAND FOUR HUNDRED SEVENTY EIGHT DOLLARS AND TWENTY THREE CENTS? If you've got to do that much typing, you might as well write your check by hand.

Your computer should be able to perform this task. Unfortunately, the translation of dollars and cents into English isn't as easy as just printing a number. I spent quite a while using the trial-and-error system to provide you with this program. I'm sure there must be a more efficient algorithm than the one I came up with, but this one does the job.

What I did was to store the English versions of the required numbers in the string N\$, and the starting and ending locations of each number in the two dimensional array, N. You enter the number in the normal numeric format, and the program does the required translation. The translation subroutine begins at line 130 and ends at line 320. B\$ is a string of 80 blanks, EA\$ holds the translated English amount, and AMOUNT\$ stores the numeric amount you enter through the keyboard.

The program will tell you what it wants and includes error-handling routines. The [BREAK] key and [SYSTEM RESET] have been left operational.

I haven't gone so far as to actually print your check, but I have taken care of the tricky part. Add your own inputs for date and payee, position your data according to the layout of your check form, and put your ATARI to work. When you actually print checks, please remember that Jerry starts with a "J."

by Jerry White

```
100 REM TRANSLATE 5/28/82 BY JERRY WHI
TE for ANTIC Magazine
110 DIM B$(80),N$(152),EA$(80),AMOUNT$
(10),N(27,27)
120 EA$="- " :EA$(80)="- " :EA$(2)-EA$:B$-
```


FEATURES

```
EAS:GOTO 330
130 EAS=B$:EAS=" ":SW=0:FOR ME=1 TO LA
140 IF AMOUNT$(ME,ME)="-." THEN EAS(LEN
(EAS)+1)="-AND ":? "AND ":GOTO 280
150 IF SW=1 THEN SW=0:GOTO 280
160 IF AMOUNT$(ME,ME)="-," THEN SW=2:GO
TO 270
170 TRAP 280:SW=0:N=VAL(AMOUNT$(ME,ME)
):J1=N(N,1):J2=N(N,2):TRAP 40000
180 IF N=0 THEN SW=2:GOTO 280
190 IF LA=ME OR ME=LA-3 OR ME=LA-5 OR
ME=LA-6 OR ME=LA-7 THEN 240
200 IF LA=8 AND ME=1 THEN 240
210 IF AMOUNT$(ME,ME)<>"1" THEN 230
220 SW=1:N=VAL(AMOUNT$(ME,ME+1)):J1=N(
N,1):J2=N(N,2):GOTO 240
230 N=N+18:J1=N(N,1):J2=N(N,2)
240 EAS(LEN(EAS)+1)-N$(J1,J2):? N$(J1,
J2);
250 IF ME=LA-5 AND N<>0 THEN EAS(LEN(E
AS)+1)="- HUNDRED":? " HUNDRED";
260 IF SW<>2 THEN EAS(LEN(EAS)+1)="- ":
? " ";
270 IF SW=2 THEN EAS(LEN(EAS)+1)="-THOU
SAND ":SW=0:? "THOUSAND "
280 NEXT ME
290 IF AMOUNT$(LA-1,LA)="-00" THEN EAS(
LEN(EAS)+1)="-NO ":? "NO ";
300 IF AMOUNT$(LA-1,LA)="-01" THEN EAS(
LEN(EAS)+1)="-CENT":? "CENT":GOTO 320
310 EAS(LEN(EAS)+1)="-CENTS":? "CENTS"
320 LEA=LEN(EAS):? :? EAS(2,LEA):? #2;
EAS(2,LEA):? #2:RETURN
330 N$="ONETWOTHREEFOURFIVESIXSEVENEIG
HTNINETELEVENTWELVETHIRTEENFOURTEENF
IFTEENSIXTEENSEVENTEEN"
340 N$(LEN(N$)+1)="-EIGHTEENNINETEENTWE
NTYTHIRTYFORTYFIFTYSIXTYSEVENTYEIGHTYN
INETY"
350 DATA 1,3,4,6,7,11,12,15,16,19,20,2
2,23,27,28,32,33,36,37,39,40,45,46,51,
52,59,60,67,68,74,75,81,82,90
360 DATA 91,98,99,106,107,112,113,118,
```

TRANSLATE: DOLLARS TO SENSE

```
119,123,124,128,129,133,134,140,141,146,147,152
370 GRAPHICS 0:POKE 82,2:POKE 83,39:POKE 710,160:POKE 752,1
380 ? :? " This program translates numeric"
390 ? :? "dollar and cent amounts into English."
400 ? :? "Input must be numeric so do not enter"
410 ? :? "dollar signs. Always include decimal"
420 ? :? "point between dollars and cents, and"
430 ? :? "a comma between the thousand and the"
440 ? :? "hundred columns when the amount is"
450 ? :? "greater than 999.99."
460 FOR J=1 TO 27:READ J1,J2:N(J,1)=J1:N(J,2)=J2:NEXT J
470 ? :? " Make sure your printer is ready,":? :? "then press START."
480 IF PEEK(53279)<>6 THEN 480
490 TRAP 560:CLOSE #2:OPEN #2,8,0,"P":TRAP 40000:POKE 752,0:CHR$(125)
500 ? :? "Enter numeric amount or just press":? :? "the RETURN key to quit";
510 INPUT AMOUNT$:LA=LEN(AMOUNT$):IF LA=0 THEN 570
520 TRAP 550:IF LA<4 OR LA>9 OR AMOUNT$(LA-2,LA-2)<>"." THEN 550
524 IF LA<7 THEN 530
525 IF LA>6 AND AMOUNT$(LA-6,LA-6)<>"," THEN ? CHR$(253):? "A , MUST SEPERATE THOUSANDS,HUNDREDS":GOTO 500
530 TRAP 40000:CHR$(125):? :? "CONVERTING $":AMOUNT$:? :? #2:"$":AMOUNT$
540 GOSUB 130:GOTO 500
550 ? CHR$(253):? "INVALID AMOUNT":GOTO 500
560 ? CHR$(253):? " READY PRINTER THEN PRESS START":GOTO 480
```

FEATURES

570 GRAPHICS 0:? :? "BASIC":? "IS";:END

TYPO TABLE

Variable checksum - 217201

Line	num	range	Code	Length
100	-	190	MD	547
200	-	300	QF	545
310	-	370	ZO	562
380	-	480	QE	529
490	-	550	CT	545
560	-	570	KF	101

Display Lists Simplified

An important step in understanding your ATARI's graphics capabilities is to create your own custom display lists. This article will show you step-by-step how to mix text and graphics on your TV screen. Our method uses BASIC commands to modify Graphics Modes 0 through 8. BASIC sacrifices some of the ATARI's flexibility; however, these techniques will help you eventually create display lists in assembly language.

The graphics capabilities of the ATARI are controlled by a microprocessor chip called ANTIC (Alpha-Numeric Television Interface Circuit). Any display list is a *program* for ANTIC.

There is a display list program provided automatically by each BASIC Graphics command, or you can define your own. The display list specifies where screen data is located, what display modes to use, and any special display options ANTIC is to implement. Since the display list describes the screen from top to bottom, any mix of graphics or text modes can be displayed on the screen.

To understand displays, you need to know a bit about television. In a TV, a beam of electrons is shot at the screen. The beam starts at the top left-hand corner and moves across the screen. When it reaches the right-hand side, the beam is turned off, returned to the left, and moved down slightly. It is then turned on again, and the process is repeated 262 times to form a completed screen image.

When the beam reaches the bottom right-hand corner of the screen, it is turned off and returned to the top left-hand corner to start over. These horizontal sweeps are called *scan lines* and are the basis of the display. The scan-line pattern actually starts above and ends below the physical boundaries of the TV screen. To assure that information is not displayed where you

Allan Moose is an associate professor (math/physics) at Southampton College, NY. Marian Lorenz is a special education teacher for handicapped children in Central Islip, NY. They have been frequent contributors to ANTIC.

FEATURES

can't see it, the ATARI display usually is restricted to 192 scan lines, positioned in the middle of the screen.

There are several other concepts you will need. These are:

ANTIC MODE NUMBER: ANTIC identifies modes with a set of numbers *different* from those used by BASIC. The ANTIC mode numbers corresponding to each BASIC Graphics Mode, 0 through 8, are listed in Table 2.

MODE LINE: A mode line is a grouping of scan lines into a fundamental unit for each Graphics Mode. For example, Graphics 8 uses one scan line per mode line; for Graphics 0 there are eight scan lines per mode line. Screen displays are made up of 192 scan lines grouped into mode lines (see Table 2).

LOAD MEMORY SCAN (LMS): The LMS number is the sum of the ANTIC mode number for the first mode line, plus 64. The LMS number has two functions. First, it tells ANTIC what mode will be used for the first mode line of the screen display. Second, LMS instructs ANTIC to take information from the screen memory area of RAM and display it. The next two bytes in the display list following the LMS number give ANTIC the starting address of the screen memory.

DISPLAY LIST POINTER: This is a variable that establishes the memory address for the first line of the display list. This address is found by the BASIC command: `PEEK (560)+PEEK(561)*256`.

JUMP WHILE VERTICAL BLANK (JVB): This signals ANTIC that the end of the display list has been reached and it must loop back to the beginning. The jump is located immediately following the last mode line of your display list and is indicated by the decimal number 65. The low byte of the return address is given by `PEEK (560)`. The high byte of the return address is given by `PEEK(561)`.

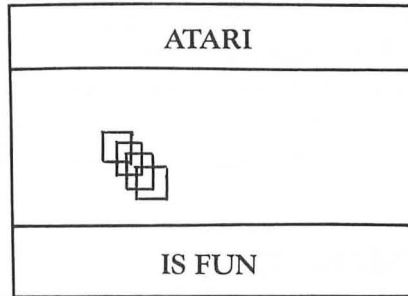
RAM REQUIREMENTS: The Graphics Modes differ in the number of bytes that must be set aside in memory for screen data (see Table 1).

RAM BYTES PER MODE LINE: Just as the Graphics Modes differ in their total RAM needs, they differ in the number of bytes required per mode line (see Table 2). This information is important for synchronizing the Operating System (OS) and ANTIC.

Developing a Custom Display List

Step 1

Make a rough sketch of what you want to appear on the screen. Our example appears as Figure 1.



Step 2

Select the Graphics Modes you want to use and the number of lines for each mode. Two requirements must be met. First, the total number of scan lines in all the mode lines should not exceed 192. If it does, the screen image may roll. However, the total can be less than 192 with no adverse effect. Second, when you insert new mode lines into an existing display list, the total number of bytes required for the inserted lines must be a whole multiple of the bytes required per mode line in the existing display list. To understand this more fully, refer to Figure 2. Diagrams such as this are invaluable in planning a display list.

Figure 2

RAM Bytes/Mode $2 \times 20 = 40$	<div>GRAPHICS MODE 2 (2 lines)</div>	Scanlines $2 \times 16 = 32$
$40 \times 128 = 5120$	<div>GRAPHICS MODE 8 (128 lines)</div>	$128 \times 1 = 128$
$4 \times 10 = 40$	<div>GRAPHICS MODE 1 (4 lines)</div>	$4 \times 8 = 32$
		TOTAL 192

FEATURES

Our example will modify a Graphics 8 display list. Each line of Graphics 8 requires 40 bytes of RAM. Therefore, at the top we must insert at least two lines of Mode 2 (two lines \times 20 bytes) to match the 40 bytes per line of Mode 8. At the bottom we will insert four lines of Mode 1, each requiring 10 bytes, for a total of 40 bytes.

Matching up the byte requirements between inserted lines and existing lines insures that the text and graphics will appear where we want them.

Step 3

After choosing the modes you want, determine from Table 1 which of them requires the most RAM. Use this mode as your base (existing) mode, onto which you make changes that create your custom display list. This insures that the OS has set aside sufficient memory to hold your screen data. We have chosen Modes 2, 8 and 1. Mode 8 requires the most RAM, so it will be our base mode, called in line 30, but first we'll write a line to clear the screen and turn off the cursor:

```
20 ? CHR$(125):POKE 752,1
```

Next we call the display list to be modified. Adding 16 to GR. 8 eliminates the GR. 0 window that is a normal part of GR. 8.

```
30 GRAPHICS 8+16
```

We recommend that you enter the program as we go along. It will help you understand the process.

Step 4

PEEK the display list pointer and assign it to a variable such as "DL".

```
40 DL=PEEK(560)+PEEK(561)*256+4
```

The number 4 is added to the display list pointer for insurance. Recall that the TV generates scan lines that do not appear on the screen. To allow for this, BASIC Graphics Modes generate 24 blank scan lines at the start of the display list. Adding 4 to the display list pointer will make sure that we don't inadvertently remove any of these lines.

Step 5

POKE the LMS instruction into DL minus 1. The value 71 derives from ANTIC mode number 7, plus 64. This instruction will establish the first mode line of the display list. If your first mode line belongs to your base mode, skip this step:

```
50 POKE DL-1,71
```

DISPLAY LISTS SIMPLIFIED

Step 6

Every mode line in your diagram requires a statement in your display list. Write these in the same order as they appear on the screen, and POKE the ANTIC mode numbers as appropriate. This is the second line of our Graphics Mode 2.

60 POKE DL + 2,7

From the diagram we can see that the next 128 lines are Graphics 8. Since this is our base mode, these lines already exist in the display list. The next mode lines to insert are the four Graphics 1 lines at the bottom.

70 POKE DL + 132,6

90 POKE DL + 134,6

80 POKE DL + 133,6

100 POKE DL + 135,6

Table 1

GRAPHICS MODE RAM REQUIREMENTS

MODE	BYTES	MODE	BYTES
8+16	8138	4+16	696
8	8112	4	694
7+16	4200	3+16	432
7	4190	3	434
6+16	2184	2+16	420
6	2174	2	424
5+16	1176	1+16	672
5	1174	1	674
		0	992

Table 2

BASIC MODE NUMBER	ANTIC NUMBER	TYPE	LMS BYTE	# OF MODE LINES	SCAN LINES/ MODE LINE	RAM BYTES/ MODE LINE
.... 0	2	... text	... 66 24 8 40
.... 1	6	... text	... 70 24 8 20
.... 2	7	... text	... 71 12 16 20
.... 3	8	... graphics	... 72 24 8 10
.... 4	9	... graphics	... 73 48 4 10
.... 5	10	... graphics	... 74 48 4 20
.... 6	11	... graphics	... 75 96 2 20
.... 7	13	... graphics	... 77 96 2 40
.... 8	15	... graphics	... 79 192 1 40

FEATURES

Step 7

End the display list with a JVB, followed by the low byte and high byte of the return address:

```
110 POKE DL+136,65
120 POKE DL+137,PEEK(560)
130 POKE DL+138,PEEK(561)
140 GOTO 140
```

Now RUN the program. You will see the top section (GR.2) black, the bottom section (GR.1) black, and the middle section (GR.8) blue. To make the middle section black, change line 30 to:

```
30 GRAPHICS 8+16:SETCOLOR 2,0,0
```

Table 3 shows the relevant portions of our display list and demonstrates another important point. Line 30 of our program has stored the LMS instruction in address 32825. Line 40 stores the value 7 in address 32828 to give us the second mode line of Graphics 2. Instructions for the Graphics 1 lines and JVB are stored in addresses 32958 through 32962.

Look at addresses 32921 through 32923. Note that here in the middle of the display list is another LMS instruction followed by a screen memory address! The reason is that ANTIC cannot address a block of memory longer than 4K bytes. Since Graphics 8 requires 8K bytes, the screen memory must be broken up into two blocks. ANTIC is sent to the first block of screen memory by the first LMS instruction in address 32825, and is sent to the second block of screen memory by the second LMS instruction in address 32921. “Jumping the 4K boundary” occurs only for Graphics 8.

You must be careful of two things when you modify a Graphics 8 display list. First, don’t clobber the second LMS instruction and the two following bytes by putting mode lines in their place. Second, you must calculate an offset if you change modes after the boundary jump. We did this in line 70, by adding two lines to the display list (DL+132 vs. DL+130).

At this point the actual display is written into screen memory. The next task will be to print “ATARI” in the Graphics 2 section. Line 10 established GR.8 and instructed the OS that data in screen memory is to be interpreted as graphics, not text. Consequently if we simply enter PRINT #6: “ATARI”, the OS will not carry out the command. The OS must be told how to interpret the data it finds in screen memory by POKEing the appropriate Graphics Mode number into memory address 87.

```
140 POKE 87,2
```

```
150 POSITION 8,0:PRINT #6; "ATARI"
```

The OS positions text or graphics on the screen by counting bytes from the start of the screen memory associated with the Graphics Mode value stored in location 87. Thus, it is possible for total screen memory to be considerably longer than the memory for the mode the OS is using. This disparity can cause "cursor out of range" error messages and trouble positioning material on the screen.

The cure for both problems is fairly simple. Before creating a display on the screen, change the start of the screen memory to coincide with the start of the mode section where you want the display to appear. For the Graphics Mode 8 section this will eliminate the trial-and-error method of placement. For the Graphics Mode 1 section this will prevent a "cursor out of range" message.

To write our display we start with:

```
160 POKE 87,8
```

to tell the OS what mode we're in. Then locate the current top of the screen address with:

```
170 TPSCRN=PEEK(88)+PEEK(89)*256
```

Next, offset the variable TPSCRN by the number of bytes in the Mode 2 lines + 1 (four Mode 2 lines × ten bytes per line = 40 bytes):

```
180 TPSCRN=TPSCRN+41
```

Finally, POKE this memory location back into 88 (low byte) and 89 (high byte):

```
190 POKE 88,TPSCRN-(INT(TPSCRN/256)*256)
```

```
200 POKE 89,INT(TPSCRN/256)
```

This procedure sets up the Graphics 8 section of our display so that the top left-hand corner corresponds to position 0,0. You can appreciate how much simpler it will be to place your display components.

```
210 COLOR 1:FOR I=1 TO 40 STEP 5
```

```
220 PLOT 60+I,40+I:DRAWTO100+I,40+I:
```

```
      DRAWTO 100+I,80+I:DRAWTO 60+I,80+I:
```

```
      DRAWTO 60+I,40+I
```

```
230 NEXT I
```

Finally, print "IS FUN" in the Mode 1 section at the bottom of the screen.

```
240 POKE 87,1
```

```
250 TPSCRN=TPSCRN+5121
```

FEATURES

Line 250 offsets TPSCRN to the beginning of the Mode 1 section. 5121 is obtained from $(128 \text{ lines of Gr. 8}) * (40 \text{ bytes per line}) = (5120 \text{ bytes}) + 1$.

260 POKE 88,TPSCRN-(INT(TPSCRN/256) * 256)

270 POKE 89,INT(TPSCRN/256)

280 POSITION 6,2:1#6; "IS FUN"

290 GOTO 290

Table 3

ADDRESS	OUR LABEL	VALUE	MEANS
32822	DL-4	112	Blank scan lines
23	DL-3	112	provide for "overscan"
24	DL-2	112	
25	DL-1	71	LMS — 64+7 sets ANTIC mode 7
26	DL	80	and one line of same
27	DL+1	129	gives address of start of screen
28	DL+2	7	memory
32829	DL+3	15	lo-byte+hi-byte*256=33104 sets ANTIC 7 for second mode line (equiv. of GR.2)
32921	DL+95	79	reverts to
22	DL+96	0	ANTIC mode 15
23	DL+97	144	for 128
32924	DL+98	15	mode lines
			(equivalent of GR.8)
			LMS and address for 4K boundary jump, includes one line of mode 15
32957	DL+131	15	sets ANTIC mode 6
32958	DL+132	6	for four lines
59	DL+133	6	(equivalent of GR.1)
60	DL+134	6	
61	DL+135	6	
62	DL+136	65	JVB to address given by next two
63	DL+137	54	bytes
32964	DL+138	128	lo-byte of return address 128*256=32768, hi-byte of return address
			+ 54
			32822 =return address

done

Tiny Text

Tiny Text is a small but clever cassette-based text editor written by Stan Ockers, originally in the A.C.E. Newsletter (3662 Vine Maple Dr., Eugene, OR 97405). Tiny Text was never intended to be an all-purpose word-processor, even though it does provide several of the important features found in larger programs. Tiny Text was written to facilitate submission of “machine readable” copy to the Eugene A.C.E. Newsletter. The real advantage of this program is that it is small, inexpensive, and very easy to use.

The program that follows is a slightly-enhanced version that includes:

- support for Atari 820 Printer.
- separate Print and Display modes.
- forms control for Print mode.
- top-of-page command for Print mode.
- save text on Cassette or Disk.
- error trap control.
- adapts to different RAM sizes.

Cassette tapes recorded by the original Tiny Text can still be used with this modified version. Finally, this version corrects a couple of minor formatting bugs and is about ten percent “tinier” than the original.

Using the Program

The [OPTION] key selects one of five options: LOAD, EDIT, PRINT, SAVE, and DISPLAY. The following paragraphs describe each of these options.

The LOAD option reloads text that was previously saved on cassette or disk. When LOAD is selected, you will be asked to enter the “file spec” of the text you want to load. If the text is on cassette, simply type a C. The computer will “beep” once to remind you to set up the recorder to play. Then press

Jim Carr has a B.S. in physics from Oregon State University, and is employed in the field of computer-controlled processing.

FEATURES

[RETURN] to begin loading the text. If the text is on disk, type the complete file name of the text file, for example, "D1:TTHelp.TXT".

The EDIT option lets you enter text or change text previously entered. When the Edit mode is requested, a blank area (text-entry window) appears in the center of the screen. Up to three lines of text can be typed into the window. Pressing [RETURN] causes text in the window to be added to previously entered text. You can use the standard screen-editing functions to edit text in the window. All trailing blanks in the window will be deleted, so it is good to end each entry at the end of a word and start each new entry with a space.

Such functions as tabulating and indentation are controlled by special formatting symbols. These symbols always cause the current line to end before the requested formatting function is executed:

CTRL E — End the current line and start a new line with no indentation.

CTRL I — Indent the next line.

CTRL S — Space before starting the next line.

CTRL T — Tab a specified number of spaces before the next line.

CTRL C — Center the next line.

CTRL P — Page. Advance the paper in the printer to the top of the next page before printing the next line.

When in the Edit mode, pressing the [SELECT] key will cause the line of text below the window to be moved up into the window. The normal screen-editing functions can then be used to fix the text in the window. Use the joystick to scroll the desired line to the position below the text window. Pressing [SELECT] twice (without making any changes) simply causes the text line to move up into the window and then back. To DELETE a line of text, move it below the text window and press [RETURN]. Press the joystick trigger to jump to the end of the text.

The PRINT option prints the formatted text on the printer. Before printing begins you may change the default settings for line length, tab stop, etc. Use the screen-edit functions to make any desired change, then press [RETURN]. The items that may be changed are:

Line — Line length (maximum number of characters per line).

Indent — The number of spaces to be indented (left margin).

Tab Stop — The number of spaces for the tab stop.

Paper Size — The total number of lines that can be printed on a fully-

covered page. For example, an 11-inch form with six lines per inch has 66 lines.

Forms Feed — The number of blank lines printed to separate the bottom of one page from the top of the next. For example, if three blank lines are required at the top and bottom of each page, then Forms Feed is set to six.

Save option lets you save text on either cassette or disk. The SAVE selection will ask for the "file name" to be used. If using a cassette, simply type C. The computer will beep twice for you to set up to record. After that, press [RETURN] to begin saving text. To save text on disk, enter the complete file name to be used. For example "D:TTHELP.TXT".

The DISPLAY option displays the text on the screen. It provides the same format-change options as the print option. Display is relatively slow. The program jumps to menu immediately after the last line.

Programming Notes

The default settings for the format control functions are defined at line 120.

If you make any changes to this program, you first make a change to line 14 which automatically expands the main data storage array T\$ to use all available memory. Try changing "SIZ=FRE(0)-50" to "SIZ=FRE(0)- 500". When you have finished making your changes you can restore the statement to its original form.

If a system error occurs, it is trapped and printed out by the program. You are then prompted to press [RETURN] to make the program continue at the option selection menu. This will generally allow you to recover from errors without loss of data.

by Jim Carr

System Requirements: 16K RAM,

```

1 REM **** TINY TEXT ****
2 REM
3 REM Stan Ockers Sept-81
4 REM ACE Newsletter Nov-81
5 REM
6 REM Mod by Jim Carr 01-OCT-82
7 REM
12 DIM SP$(40):FOR I=1 TO 40:SP$(I,I)=
" ":NEXT I
14 DIM S$(45),I$(120),A$(128):SIZ=FRE(
0)-50:DIM T$(SIZ):FOR I=1 TO 45:READ A
:S$(I)=CHR$(A):NEXT I

```

FEATURES

```
20 DATA 104,104,133,204,104,133,203,10
4,133,206,104,133,205,104,104,168,162,
0,161,203,145,203,198,203,165
30 DATA 203,201,255,208,2,198,204,165,
203,197,205,208,236,165,204,197,206,20
8,230,96
40 FOR I=1536 TO 1643:READ A:POKE I,A:
NEXT I
50 DATA 104,104,133,204,104,133,203,10
4,133,206,104,133,205,162,0,169,240,32
,53,6,169,40,32,91,6
60 DATA 165,207,208,8,169,160,32,91,6,
24,144,10,169,40,32,53,6,169,120,32,91
,6,169,240,32,53,6,96
70 DATA 133,208,161,203,201,96,176,11,
201,32,176,5,24,105,64,208,2,233,32,12
9,205,230,203,208,2
80 DATA 230,204,230,205,208,2,230,206,
198,208,208,221,96,133,208,169,0,129,2
05,230,205,208,2
90 DATA 230,206,198,208,208,244,96
110 P=241:POKE 207,0:POKE 82,0:OPEN #2
,4,0,"E":TS(1)="":TS(480)="":TS(2)=
TS
120 SCR=PEEK(88)+256*PEEK(89)+120:LL=3
5:LM=1:IND=5:TAB=10:PS=66:FF=6:GOTO 50
0
290 ? "INSERT TEXT OR ... PRESS SELEC
T TO EDIT"
300 POSITION 0,0:? SIZ=LEN(T$):" FREE
":S=STICK(0):IF S=15 THEN 330
305 IF S=14 AND P<LEN(T$)-320 THEN P=P
+40
310 IF S=13 AND P>280 THEN P=P-40
315 IF S=11 AND P<LEN(T$)-280 THEN P=P
+1
320 IF S=7 AND P>241 THEN P=P-1
330 A=USR(1536,ADR(T$)+P-241,SCR)
335 K=0
340 POKE 53279,8:PK=PEEK(53279):IF PK=
5 THEN GOSUB 900
350 IF PK=3 THEN 500
360 IF PEEK(764)<255 THEN 400
```

```

365 K=K+1:IF K<10 THEN 340
370 IF STRIG(0)=0 THEN P=LEN(T$)-240:POKE 207,0
380 GOTO 300
400 POSITION 0,10:INPUT #2:IS$:PK=PEEK(207):IF PK=0 THEN AS$=""
405 LI=LEN(IS$):LT=LEN(T$):IF LI=0 THEN 460
407 IF LI+LT>SIZ THEN POSITION 0,1:? "OUT OF SPACE":GOTO 300
410 IF PK=1 THEN AS$=T$(P,P+39):IF T$(P+39,P+39)="" THEN IS$(LI+1)="" :LI=LI+1
420 LA=LEN(AS$):AD=ADR(T$):IF LI>LA THEN A=USR(ADR(SS),AD+LT-1,AD+P-2,LI-LA)
430 T$(P,P+LI-1)=IS$
440 IF LA>LI THEN T$(P+LI)=T$(P+LA)
450 P=P+LI:T$(LT+LI-LA+1)="" :POKE 207,0:GOTO 300
460 IF PEEK(207)=1 THEN 470
465 IF P<LEN(T$)-279 THEN T$(P)=T$(P+40)
470 POKE 764,255:GOTO 300
500 TRAP 950:ST=PEEK(560)+PEEK(561)*256+4:POKE ST-1,70:POKE ST+2,7:POKE ST+3,112:POKE ST+4,6:POKE ST+5,6
501 POKE ST+24,65
510 POKE ST+25,PEEK(560):POKE ST+26,PEEK(561)
515 OP=OP+1:IF OP=6 THEN OP=1
520 ? CHR$(125):POSITION 20,0:IF OP=1 THEN ? "LOAD"
522 IF OP=2 THEN ? "EDIT"
534 IF OP=3 THEN ? "PRINT"
536 IF OP=4 THEN ? "SAVE"
538 IF OP=5 THEN ? "DISPLAY"
540 POSITION 0,1:? "PRESS START TO BEGIN"
550 FOR D=1 TO 30:NEXT D
555 POKE 53279,8:IF PEEK(53279)=3 THEN 515
557 IF PEEK(53279)<>6 THEN 555
560 POKE 764,255:POSITION 20,1:? CHR$(

```


FEATURES

```
125):POSITION 0,1:ON OP GOTO 2000,290,
590,1500,590
590 FOR I=1 TO 6:? CHR$(127);CHR$(158)
::NEXT I:? :FOR I=1 TO 6:? " ";CHR$(15
9):" " ::NEXT I
594 POSITION 0,1:? "SET FORMAT CONTROL
S":POSITION 0,6:? "LINE LEFT IN- TAB
PAGE FORM"
595 ? "SIZE MARG DENT STOP SIZE FEED":
? CHR$(127);LL;" ";CHR$(127);LM;" ";CH
R$(127);IND;" ";CHR$(127);
596 ? TAB;" ";CHR$(127);PS;" ";CHR$(12
7);FF:POSITION 0,8
600 INPUT LL,LM,IND,TAB,PS,FF:P=240:P=
240:GOTO 710
610 P=240
670 GOTO 620
710 LINE=0:GRAPHICS 0:POSITION 0,3:FL=
0
715 RL=LL:TP=P:B=ASC(T$(TP,TP))
720 RL=RL-IND*(B=9)-TAB*(B=20)
725 IF B=19 AND OP=3 AND LINE<=(PS-FF)
THEN LPRINT " ":LINE=LINE+1
726 IF B=19 AND OP=5 THEN ?
727 IF B=16 AND OP=3 THEN FOR I=1 TO P
S-LINE:LPRINT " ":NEXT I:LINE=0
728 IF B=16 AND OP=5 THEN ? :? :? :LIN
E=0
735 C=0:K=0
740 K=K+1:TP=TP+1:IF K=RL+1 THEN 765
745 IF TP>LEN(T$)-241 THEN FL=1:GOTO 8
10
750 A=ASC(T$(TP,TP)):IF A<32 THEN C=0:
GOTO 780
755 IF A=32 THEN C=C+1
760 GOTO 740
765 IF C=0 THEN A$=T$(P+1,TP-1):TP=TP-
1:GOTO 810
767 IF T$(TP,TP)=" " THEN A$=T$(P+1,TP
-1):GOTO 810
768 IF T$(TP-1,TP-1)=" " THEN C=C-1
770 K=1
775 TP=TP-1:IF T$(TP,TP)<>" " THEN K=K
```

```

+1:GOTO 775
780 IF TP=P+1 THEN P=TP:GOTO 715
785 A$="":I=P+1
790 A$(LEN(A$)+1)=T$(I,I):IF T$(I,I)<>
" " THEN 805
795 IF C>1 THEN A=INT(K/C+RND(0)):IF A
>0 THEN FOR J=1 TO A:A$(LEN(A$)+1)=" "
:NEXT J:K=K-A
800 C=C-1
802 IF C=1 AND K>0 THEN FOR J=1 TO K:A
$(LEN(A$)+1)=" ":NEXT J
805 I=I+1:IF I<TP THEN 790
810 IF FL THEN A$=T$(P+1,TP-1)
815 IF OP=3 THEN LINE=LINE+1:IF LINE>(
PS-FF) THEN LINE=1:FOR I=1 TO FF:LPRIN
T " ":NEXT I
820 SP=LM+(B=9)*IND+(B=20)*TAB+(B=3)*(
LL-LEN(A$))/2:IF SP>40 THEN SP=40
830 IF OP=3 THEN LPRINT SP$(1,SP):A$
840 IF OP=5 THEN ? SP$(1,SP):A$
850 IF FL THEN 500
860 P=TP:GOTO 715
900 PK=PEEK(207):IF PK=1 THEN POKE 207
,0:GOTO 930
910 IF PK=0 AND P<LEN(T$)-279 THEN POK
E 207,1
930 A=USR(1536,ADR(T$)+P-241,SCR):FOR
D=1 TO 50:NEXT D:RETURN
950 ? "ERROR ":PEEK(195):" AT ":256*PE
EK(187)+PEEK(186):? "PRESS RETURN TO C
ONTINUE":INPUT I$:GOTO 500
1500 ? " ENTER FILE NAME":INPUT I$:OPE
N #3,8,0,I$:N=INT(LEN(T$)/128):? #3;N:
IF N=0 THEN ST=0:GOTO 1520
1510 FOR I=1 TO N:ST=128*I: ? #3:T$(ST-
127,ST):NEXT I
1520 ? #3:T$(ST+1,LEN(T$)):CLOSE #3:GO
TO 500
2000 ? " ENTER FILE NAME":INPUT I$:OPE
N #3,4,0,I$:INPUT #3,N:IF N=0 THEN BEG
=-127:GOTO 2020
2010 GRAPHICS 0:FOR I=1 TO N:BEG=128*I
-127:INPUT #3,A$: ? A$;:T$(BEG)=A$:NEXT

```

950 CLOSE #3:

FEATURES

I
2020 INPUT #3, A\$: T\$(BEG+128) = A\$: CLOSE
#3: POKE 1536, 104: GOTO 500

TYPO TABLE

Variable checksum = 636317

Line num range	Code	Length
1 - 40	RL	509•
50 - 110	QN	536
120 - 340	QW	525
350 - 420	MY	501
430 - 520	WU	536
522 - 594	CP	569
595 - 727	UF	505
728 - 775	NA	495•
780 - 820	WM	515•
830 - 1510	QJ	556•
1520 - 2020	CJ	310

done

Christmas Mailing Lister

Exchanging Christmas cards helps make this the holidays special, but digging through old slips of paper to find your addresses can take the fun out of it. Hand addressing all those outgoing envelopes is no thrill either. This year let your ATARI start handling this chore.

Christmas Mailing Lister is a cassette-based program that stores up to 140 addresses. You can create, change, or delete addresses at any time. You can print individual addresses, selected categories, or the whole file, sorted alphabetically by name or city. The printout can be done on labels, if you have the proper supplies and equipment, or in the form of an address book.

The unique feature that makes this nice for a Christmas list is that names are stored beginning with the letter entered in inverse video, rather than the first letter of the name field. This way your labels can read "John and Mary Smith," or "The John Smith Family," instead of "Smith, John and Sue," or "Smith Family, The John." Just type the capital "S" in inverse video. Unfortunately, this sort only works when running the whole list. An individual search for the Smith entry would still require hunting for "John and Mary Smith." "Smith" alone would not be enough.

You can also define up to six different categories for selected sub-sorts. Each name must belong to one category only, although this assignment may be changed at will. One possible use for the categories is to keep track of card exchange. For example, the categories could be defined as follows:

1. sent us a card in 1981
2. sent us a card in 1982
3. sent us a card in 1983
4. sent us a card in 1984
5. sent us a card in 1985
6. did not send card

This should keep you organized for a few years, by which time you'll probably have a disk drive and a store-bought program.

Bill Lukeroth is a heavy equipment claims adjuster, freelance writer and self-avowed "ATARI hacker."

FEATURES

This program requires a printer, a 410 Program Recorder, and at least 32K of RAM. The first step is to type the program into the computer. I recommend that you CSAVE to your permanent cassette and a backup before attempting to RUN the program. Note that "Merry Christmas!" in line 250 must be in upper-case inverse video.

When you RUN the program, first you'll see the title page, which changes to a menu after 20 seconds. You can shorten the wait by pressing [START]. The first four items on the menu require insertion of a data cassette, so the first time through you must select item #5 ("create a completely new address list"). Then you will define your six categories, each using 25 characters or less. You can bypass the category feature by pressing [RETURN] each time.

The next screen asks for a name, address, etc. Each of the first three fields can hold 28 characters. You can put in a nine-digit ZIP code (or shorter) and an area code with your phone numbers. Sorry, no numerical sorting with this program.

Enter a few addresses, then return to the main menu to experiment with the print, change and delete options. When you understand these, continue to enter addresses until you exhaust your list, or your computer's memory. Then return to the menu and select item #7 ("end"). You will be prompted to insert a blank cassette so you can record all your data onto tape. Do not use your program cassette for this. Also make a backup tape at this time, it's a lot of work to retype data! Now you can try the other program features without fear.

Tips and Hints

Every printer is different. The Atari 822, or other thermal printer (such as the Alphacom), does not have ready-made label paper. You can still cut and paste your labels though.

The Atari 825 Printer, and certain other 80-column printers (such as the Epson), can use fan-fold labels with adhesive backs. Typically these labels are spaced at one-inch (six lines) intervals. You may have to adjust lines 7220 and 7230 of the program to accomodate your labels. LE is the variable that determines the number of blank lines between labels. If you change the value of LE in 7220, you must change 7230 so that LE equals one less than it does in 7220.

7720 LE=2

7230 IF Q2\$ "Y" THEN ? #2;B4\$;NAME\$(105,119),
NAME\$(120,120):LE=1

CHRISTMAS MAILING LISTER

The Atari 820 Printer does not work well with fan-fold labels because these are too thick. Try Dennison's "file-folder labels," product number 36-471, which come in rolls of 250 labels.

When you are sorting the whole file, the screen should change color each time a sorting loop is completed. This reassures you that the sort is taking place.

Abort and return features include these: the [BREAK] key is disabled to prevent accidental crashes; YES or NO prompts require "Y," anything else returns to main menu; [OPTION] aborts to main menu, even while printing, except at a prompt. [OPTION] plus [RETURN] escapes a prompt. Atari screen editing is always available, but can destroy a screen if misused.

Load the data tape according to screen instructions and standard procedures. If there is a tape error, you must "end." The tape can take five to 10 minutes to load. A tone alerts you when it is finished.

Searching for a single entry requires you to enter the name line, exactly as entered, far enough to make the search unique. Remember, the inverse video character does not function in search mode. If you have "John and Mary Smith" and "John and Milly Doe" in your file, you will have to specify the search at least through the second letter of the woman's name to call the correct record.

If one of us has goofed terribly, the anguished program will go out in a blaze of glory, which should include the offending line number. Note this carefully and study the fault. To witness the death scene, type GOTO 9200 instead of RUN. Caution: this will erase any addresses not on tape.

May you have many pleasant holiday seasons.

by Bill LukerOTH

```
10 REM ** CHRISTMAS MAILING LIST **
20 REM BY BILL LUKEROTH
100 REM REVISION 0.3, WRITTEN 10/07/82
150 REM
160 REM MEMORY USED: 32K
180 REM DESCRIPTION: mailing list, print
  s labels or address books
190 DIM BK$(28): FOR L=1 TO 28: BK$(L,L)
  ="_": NEXT L: MSL=15400: REM allows for 1
  40 names
200 DIM MAIN$(MSL), NAMES(110), TEMPS(11
```

FEATURES

```
0),SEARCHNAME$(28),SEARCHCITY$(28),FIRM$(28),ADD$(28),CITY$(28)
210 DIM ZIP$(10),PHONES(14),Q2$(1),CAT$(1),C$(10),CAT1$(25),CAT2$(25),CAT3$(25),CAT4$(25),CAT5$(25),CAT6$(25)
220 DIM CIV$(1),CIV2$(1),NAME2$(110),B4$(6),B$(1):B4$="-" "":B$="-" "
230 FLAG1-0:C$="CATAGORY #":FLAG3-0:FLAG6-0:S-0
240 OOPS-9000:MENU-300:TRAP OOPS:DISBRK-9600:REBRK-9650
250 GRAPHICS 2+16:SETCOLOR 2,3,S:SETCOLOR 4,14,0:SETCOLOR 0,3,0:? #6:? #6;"
MERRY CHRISTMAS!"?:#6
260 ? #6;" MAILING LIST"?:#6:? #6:? #6:? #6
280 FOR TITLE=1 TO 30:IF PEEK(53279)=6 THEN POP:GOTO 300:REM check start button
285 FOR L0=1 TO 100:NEXT L0:IF S=0 THEN S=8:GOTO 288
287 S=0
288 SETCOLOR 2,3,S
290 NEXT TITLE
300 CLOSE #1:CLOSE #2:GRAPHICS 0
310 ? :? "CHOOSE ONE"?:?
320 ? " 1.SEARCH FOR A LISTING(IN ORDER TO PRINT A MAILING LABEL,OR CHANGE";
330 ? " OR DELETE A LISTING)."
340 ? " 2.ADD A LISTING."
350 ? " 3.PRINT A COMPLETE 'ADDRESS BOOK'."
360 ? " 4.PRINT MAILING LABELS FOR EVERYONE ON THE LIST."
370 ? " 5.CREATE A COMPLETELY NEW ADDRESS LIST(A NEW DATA BASE)."
380 ? " 6.CREATE A BACK-UP TAPE."
390 ? " 7.END."
400 ? :? "TYPE 1,2,3,4,5,6 OR 7":GOSUB DISBRK
410 INPUT Q1:GRAPHICS 0:IF Q1<1 OR Q1>7 THEN ? "ANSWER MUST BE BETWEEN 1 AND
```

CHRISTMAS MAILING LISTER

```
7." :? :GOTO 310
420 GOSUB DISBRK:ON Q1 GOTO 430,430,43
0,430,1100,2020,2000
430 FLAG6=FLAG6+1:IF FLAG6>1 THEN 500
440 ? "INSERT THE DATA CASSETTE,REWIND
TO START,PRESS 'PLAY' AND HIT 'RET
URN'"
445 OPEN #1,4,0,"C":REM get data from
cassette file
450 FOR L=1 TO 128:GET #1,DUMMY:NEXT L
:REM this loop does nothing but is req
uired by Atari BASIC
460 INPUT #1;CAT1$:INPUT #1;CAT2$:INPU
T #1;CAT3$:INPUT #1;CAT4$:INPUT #1;CAT
5$:INPUT #1;CAT6$
470 INPUT #1;TEMPS$:IF TEMPS$=CHR$(253)
THEN 490
480 MAIN$(LEN(MAIN$)+1)-TEMPS$:TEMPS$=""
:GOTO 470
490 SOUND 0,60,10,14:FOR L=1 TO 250:NE
XT L:SOUND 0,0,0,0:? :? "TURN RECORDER
OFF,THEN PRESS 'START' TO CONTINUE.
"
495 IF PEEK(53279)<>6 THEN 495
497 CLOSE #1:GRAPHICS 0:GOSUB DISBRK
500 ON Q1 GOTO 520,1210,1400,1870
520 ? "WHAT NAME ARE YOU LOOKING FOR?"
530 INPUT TEMPS$:MARK=28:GOSUB 8500
540 SEARCHNAME$=TEMPS$:TEMPS$=""
550 ? "WHAT CITY?(OPTIONAL.IF NOT NEED
ED TYPE 'N')":
560 INPUT TEMPS$:MARK=28:GOSUB 8500
570 SEARCHCITY$=TEMPS$:TEMPS$=""
575 NL=1
580 FOR L2=NL TO LEN(MAIN$)-109 STEP 1
10
585 GOSUB 7800:IF FLAG4=1 THEN POP :GO
TO 300
590 NAME$=MAIN$(L2,L2+109):FLAG2=0
595 REM line 600 compares name$ and se
arch$ character by character
600 FOR L3=1 TO LEN(SEARCHNAME$):CIV$=
NAME$(L3,L3):CIV2$=SEARCHNAME$(L3,L3):
```


FEATURES

```
XN=ASC(CIV$):XS=ASC(CIV2$)
605 IF XN<>XS AND XN<>XS+128 THEN FLAG
2=1
610 NEXT L3:IF FLAG2=1 THEN 630
620 NL=L2+110:POP :GOTO 680:REM names
match
630 NEXT L2
640 ? "NO RECORD FOUND.ARE YOU SURE TH
AT"
650 ? SEARCHNAMES?: "IS THE CORRECT SP
ELLING?":GOTO 310
680 IF SEARCHCITY$="N" THEN 750
690 CITY$=NAMES$(57,84):FLAG3=0
700 FOR L4=1 TO LEN(SEARCHCITY$)
705 GOSUB 7800:IF FLAG4=1 THEN 300
710 IF SEARCHCITY$(L4,L4)<>CITY$(L4,L4
) THEN FLAG3=1
720 NEXT L4:IF FLAG3=0 THEN 750
730 ? "FOUND ONE IN:":? CITY$:? "STILL
SEARCHING FOR THE RIGHT ONE.":? :GOTO
580
750 FIRMS$=NAMES$(1,28):ADD$=NAMES$(29,56
):CITY$=NAMES$(57,84):ZIP$=NAMES$(85,94)
:PHONES$=NAMES$(95,108)
755 CAT$=NAMES$(109,110)
770 GRAPHICS 0:SETCOLOR 2,5,2:GOSUB DI
SBRK: ? B$:FIRMS?: ? B$:ADD$: ? B$:CITY$: ?
B$:ZIP$: ? B$:PHONES?: ? B$:CAT$
780 POSITION 2,8: ? "DO YOU WANT TO:": ?
" 1.PRINT A LABEL": ? " 2.DELETE THIS
LISTING": ? " 3.CHANGE THIS LISTING"
790 ? " 4.RETURN TO MENU"
800 ? "CHOOSE 1,2,3 OR 4": :INPUT Q2
810 IF Q2<1 OR Q2>4 THEN 780
820 ON Q2 GOTO 840,900,950,300
835 REM label printing routine
840 GOSUB 7000
850 OPEN #2,8,0,"P":LABEL=0
860 GOSUB 7200
870 GOTO 300
895 REM file deletion routine
900 ? : ? "ARE YOU SURE THAT YOU WANT T
O DELETE THIS(ENTER Y OR N)": :INPUT Q
```

CHRISTMAS MAILING LISTER

```

2$
920 IF Q2$<>"Y" THEN 780
930 GOSUB 7500
940 GOTO 300
950 RESTORE :NAME$="":? "IF LINE IS O.
K. PRESS RETURN. IF NOT MAKE CHANGES
AND THEN PRESS RETURN"
960 ? "(HERE ARE YOUR CATAGORIES:)":GO
SUB 6200
970 POSITION 2,0
980 FOR L7=1 TO 6:INPUT TEMPS
990 GOSUB 7800:IF FLAG4=1 THEN 770
1000 READ CR,MARK
1010 IF LEN(TEMPS)>MARK THEN ? CHR$(25
3):RESTORE :POP :GOTO 770
1040 IF LEN(TEMPS)<MARK THEN TEMPS(LEN
(TEMPS)+1)=" ":GOTO 1040
1045 GOSUB 8500
1050 NAME$(LEN(NAME$)+1)=TEMPS
1060 NEXT L7
1070 MAIN$(NL-110,NL-1)=NAME$:GOTO 300
1090 REM new data base creation routin
0
1100 SETCOLOR 2,6,6:? "THIS IS GOING T
O ERASE ANY ADDRESSES NOW IN MEMORY. I
S THAT O.K.?"
1110 ? "(ENTER Y OR N)":INPUT Q2$
1120 IF Q2$<>"Y" THEN 300
1130 GOSUB DISBRK:MAIN$="":? "YOU'RE G
OING TO HAVE TO FURNISH THE NAMES FO
R 6 CATAGORIES. IF YOU DON'T"
1140 ? "WANT TO NAME A PARTICULAR CATA
GORY JUST PRESS 'RETURN'"
1150 ? :? C$:"1":INPUT CAT1$
1160 ? C$:"2":INPUT CAT2$
1170 ? C$:"3":INPUT CAT3$
1180 ? C$:"4":INPUT CAT4$
1190 ? C$:"5":INPUT CAT5$
1200 ? C$:"6":INPUT CAT6$
1205 ? :? "DOUBLE CHECK THE CATAGORIES
, IF THEY ARE O.K. ENTER 'Y', IF NOT
ENTER 'N'.":INPUT Q2$
1206 IF Q2$<>"Y" THEN GRAPHICS 0:? "LE

```

FEATURES

```
T'S TRY IT AGAIN:":GOTO 1130
1209 REM add a file routine
1210 GRAPHICS 0:SETCOLOR 2,6,2:FLAG1-1
:FLAG6-1:NAMES$-"":RESTORE :GOSUB DISBR
K
1220 IF LEN(MAINS$)-MSL THEN ? "ALL FIL
ES FULL":GOTO 310
1230 ? "YOU MAY NOW ADD UP TO ":(MSL-L
EN(MAINS$))/110:" ADDRESSES"
1240 ? "NAME:":BK$:? "STREET:":BK$:? "
CITY/ST:":BK$:? "ZIP CODE:":BK$(1,10):
? "PHONE #:":BK$(1,14)
1243 ? "CATAGORY:":BK$(1,1)
1245 ? :? :? :GOSUB 6200
1250 OPEN #1,4,0,"K:"
1260 FOR L9-1 TO 6
1265 GOSUB 7800:IF FLAG4-1 THEN RESTOR
E :GOTO 300
1270 READ CR,MARK:POSITION CR,L9:?"["
::REM move cursor to correct position
1280 GOSUB 5000
1290 NAMES$(LEN(NAMES$)+1)-TEMPS$
1300 NEXT L9
1305 CLOSE #1
1310 MAIN$(LEN(MAINS$)+1)-NAMES$:? :? "W
ANT TO ADD ANOTHER(ENTER Y OR N)":INP
UT Q2$
1320 RESTORE :IF Q2$="Y" THEN 1210
1330 GOTO 300
1390 REM address book routine
1400 SETCOLOR 2,13,2:?"DO YOU WANT TH
E BOOK SORTED ALPHA-          BETICALLY BY:"
1410 ? "1.LAST NAME":? "2.CITY":? "OR"
:?"3.UNSORTED"
1420 ? "(ENTER 1,2 OR 3)":INPUT Q5
1425 GOSUB 7800:IF FLAG4-1 THEN 300
1430 GRAPHICS 0:SETCOLOR 2,13,2:GOSUB
DISBRK:?"DO YOU WANT:":GOSUB 6200:GOS
UB 6210
1435 GRAPHICS 0:?" :? :? "          PLEA
SE STAND BY":GOSUB DISBRK
1440 FLAG5-1:STR-1:STR2-1:ENND-28:ON Q
5 GOTO 1460,1450,1800
```

CHRISTMAS MAILING LISTER

```
1450 STR=57:STR2=57:ENND=84:REM city$
1460 FOR L15=LEN(MAIN$)-219 TO 1 STEP
-110
1465 SETCOLOR 2,L15/110,L16
1470 IF FLAG5=0 THEN POP :GOTO 1800
1480 FLAG5=0
1490 FOR L16=1 TO L15 STEP 110
1500 NAME$=MAIN$(L16,L16+109):NAME2$=M
AIN$(L16+110,L16+219):IF Q5=2 THEN 151
0
1503 FOR L21=1 TO 28:CIV$=NAME$(L21,L2
1):IF ASC(CIV$)>159 THEN STR=L21
1504 NEXT L21
1505 FOR L22=1 TO 28:CIV$=NAME2$(L22,L
22):IF ASC(CIV$)>159 THEN STR2=L22
1506 NEXT L22
1510 IF NAME$(STR,ENND)<=NAME2$(STR2,E
NND) THEN 1530
1520 MAIN$(L16,L16+109)=NAME2$:MAIN$(L
16+110,L16+219)=NAME$:FLAG5=1
1530 NEXT L16
1540 NEXT L15
1550 REM sorting completed
1800 GRAPHICS 0:Q3=1:Q2$="Y":PAGE=-1:F
LAG4=0:OPEN #2,8,0,"P":GOSUB 5200:GOS
UB 6500
1810 ? :? "DO YOU WANT ANOTHER COPY?":
IF FLAG4=1 THEN 300
1820 ? "(ENTER Y OR N)":INPUT Q2$
1830 IF Q2$="Y" THEN 1800
1840 GOTO 300
1860 REM mass mailing routine
1870 SETCOLOR 2,4,4:? "DO YOU WANT MAI
LING LABELS FOR:"
1880 GOSUB 6200:GOSUB 6210
1890 GOSUB 7000
1900 PAGE=-1000:OPEN #2,8,0,"P":GOSUB
6500
1910 GOTO 300
2000 SETCOLOR 2,13,4:TEMP$="":IF FLAG1
=0 THEN 4999
2010 ? "SINCE YOU HAVE CHANGED SOME FI
LES(OR CREATED NEW ONES)YOU MUST NOW
```

FEATURES

```
SAVE      THE DATA ON TAPE."
2020 ? "INSERT THE DATA CASSETTE, REWIN
D TO      START, PRESS 'PLAY' AND 'RECOR
D' AND    HIT 'RETURN'."
2025 ? "MAKE SURE THAT YOU USE THE DAT
A TAPE, NOT THE PROGRAM TAPE.":GOSUB 5
500
2030 OPEN #1,8,0,"C:"
2040 FOR L=1 TO 128:PUT #1,0:NEXT L
2050 ? #1:CAT1$:? #1:CAT2$:? #1:CAT3$:
? #1:CAT4$:? #1:CAT5$:? #1:CAT6$
2055 IF INT(LEN(MAINS$)/110)<>LEN(MAINS
)/110 THEN MAINS=MAINS(1,LEN(MAINS)-1)
:GOTO 2055
2060 FOR L12=1 TO LEN(MAINS$)-109 STEP
110
2070 TEMP$=MAINS$(L12,L12+109):IF TEMP$
(1,1)="@ " THEN 2075
2073 ? #1:TEMP$
2075 NEXT L12
2080 ? #1:CHR$(253):CLOSE #1
2090 ? :? "DO YOU WANT TO MAKE A/ANOTH
ER BACK-UP TAPE(ENTER Y OR N)":INPUT
Q2$
2100 IF Q2$="Y" THEN 2020
2110 IF Q1=6 THEN 300
4999 GRAPHICS 0: ? :? "PROGRAM TERMINAT
ED.":END
5000 TEMP$="":LNL=1
5010 GET #1,KEY:IF KEY=155 THEN 5080:R
EM check return button
5020 IF KEY=126 AND LNL>1 THEN LNL=LNL
-1:TEMP$(LNL,LNL)="":? CHR$(KEY):REM
backspace
5030 IF KEY>96 AND KEY<123 THEN KEY=KE
Y-32:REM convert lower case to upper
5040 IF KEY<32 OR KEY>223 THEN 5010:RE
M mask out bad input
5050 IF KEY>122 AND KEY<160 THEN 5010:
REM ditto
5060 TEMP$(LNL,LNL)=CHR$(KEY):? CHR$(K
EY):LNL=LNL+1:IF LNL>MARK THEN 5080
5070 GOTO 5010
```

CHRISTMAS MAILING LISTER

```

5080 IF LEN(TEMP$)<MARK THEN TEMP$(LEN
(TEMP$)+1)="- " :GOTO 5080
5090 RETURN
5200 ? #2;" CATAGORY INDEX
":? #2
5210 ? #2;"1.";CAT1$:? #2;"2.";CAT2$:?
#2;"3.";CAT3$:? #2;"4.";CAT4$:? #2;"5
.";CAT5$:? #2;"6.";CAT6$
5220 FOR L18=1 TO 20: ? #2:NEXT L18:FOR
L19=1 TO 40: ? #2;"-";:NEXT L19:FOR L2
0=1 TO 5: ? #2:NEXT L20
5230 RETURN
5500 POKE 53775,35:POKE 53768,40:POKE
53764,0:POKE 53766,0:POKE 53773,225
5510 RETURN :REM per Atari this routin
e is necessary to help prevent tape er
rors
6000 FOR L10=1 TO CR: ? CHR$(31)::NEXT
L10:REM move cursor to right
6010 RETURN
6200 ? "1.";CAT1$:? "2.";CAT2$:? "3.";
CAT3$:? "4.";CAT4$:? "5.";CAT5$:? "6."
:CAT6$
6205 RETURN
6210 ? "7.ALL OF THE ABOVE.":? "(ENTER
1,2,3,4,5,6 OR 7)"::INPUT Q4
6220 RETURN
6500 LABEL=0:REM printing control rout
ine
6510 FOR L11=1 TO LEN(MAIN$)-109 STEP
110
6515 GOSUB 7800:IF FLAG4=1 THEN POP :R
ETURN
6520 NAMES=MAIN$(L11,L11+109)
6525 IF NAMES(1,1)="@ " THEN 6560
6530 IF Q4=7 THEN 6550
6540 IF VAL(NAMES(109,109))<>Q4 THEN 6
560
6550 PAGE=PAGE+1:IF PAGE=7 THEN PAGE=0
:FOR L14=1 TO 40: ? #2;"-";:NEXT L14:FO
R L15=1 TO 5: ? #2:NEXT L15
6553 FOR L19=1 TO LEN(NAMES):CIV$=NAME
$(L19,L19):IVC=ASC(CIV$):IF IVC>159 TH

```

FEATURES

```
EN NAMES$(L19,L19)-CHR$(IVC-128)
6554 NEXT L19:REM this changes inverse
characters back to normal,so we can p
rint them
6555 GOSUB 7200
6560 NEXT L11
6570 CLOSE #2:RETURN
7000 ? :? "DO YOU WANT THE PHONE NUMBE
R ON THE LABEL(ENTER Y OR N)";:INPUT
Q2$
7010 ? "HOW MANY COPIES";:INPUT Q3
7020 RETURN
7200 FOR L5=1 TO Q3
7205 GOSUB 7800:IF FLAG4=1 THEN POP :R
ETURN
7210 ? #2;B4$;NAMES$(1,28):? #2;B4$;NAM
E$(29,56):? #2;B4$;NAMES$(57,84):? #2;B
4$;NAMES$(85,94)
7220 LE=4
7230 IF Q2$="Y" THEN ? #2;B4$;NAMES$(95
,108),NAMES$(109,109):LE=3
7235 IF Q1=3 THEN LE=3
7240 FOR L6=1 TO LE: ? #2:NEXT L6
7250 NEXT L5:RETURN
7500 FLAG1=1:MAIN$(L2,L2)="-@" :REM dele
te file
7510 RETURN
7800 FLAG4=0:IF PEEK(53279)=3 THEN FLA
G4=1:REM check option button
7810 RETURN
8495 REM routine to convert lower case
letters to upper case
8500 FOR L1=1 TO LEN(TEMP$):T1=ASC(TEM
P$(L1,L1)):IF T1>96 THEN TEMP$(L1,L1)=
CHR$(T1-32):NEXT L1
8505 IF LEN(TEMP$)>MARK THEN TEMP$=TEM
P$(1,MARK)
8510 RETURN
9000 REM error trapping routine
9010 ERR=PEEK(195):REM error # stored
in location 195
9020 ERRLN=PEEK(187)*256+PEEK(186):VV
=0:REM error line # stored at these lo
```

CHRISTMAS MAILING LISTER

```

cations, low byte first
9030 SETCOLOR 2,3,4:? CHR$(253):TRAP 0
OPS:REM turn screen pink,sound buzzer,
reset trap
9040 IF ERR>8 AND ERR<138 THEN 9200
9050 IF ERR=141 THEN 9200
9060 IF ERR<>3 AND ERR<>8 THEN 9080
9070 ? "INPUT ERROR.EITHER THE VALUE W
AS      OUTSIDE THE EXPECTED RANGE OR
YOU"
9075 ? "INPUT A LETTER WHERE A NUMBER
WAS      CALLED FOR.":? :GOTO ERRLN-10
9080 IF ERR<>138 THEN 9110
9090 ? "PRINTER OR TAPE ERROR.MAKE SUR
E THAT THE DEVICE IS TURNED ON AND AL
L CABLE"
9100 ? "CONNECTIONS SECURE,AND THEN CH
OOSE:":GOTO 9130
9110 IF ERR<140 OR ERR>143 THEN 9200
9120 ? "TAPE ERROR.REWIND AND THEN CHO
OSE:"
9130 ? " 1.RETURN TO MAIN MENU"
9140 ? " 2.END"
9150 ? :? "(ENTER 1 OR 2)":TRAP 0OPS:I
NPUT ERRQ:REM reset trap before return
ing to main program
9160 ON ERRQ GOTO 9170,9190
9170 FLAG6=0:CLOSE #1:CLOSE #2:CLOSE #
3:GOTO MENU
9190 GRAPHICS 0:END
9200 GRAPHICS 0:SETCOLOR 2,3,0:POKE 75
2,1:FOR XX=1 TO 5:REM turn screen red,
turn cursor off;all hope is lost
9210 POSITION 14,10:? "FATAL ERROR":SO
UND 0,47,10,10:REM make warbler sound
9220 FOR YY=1 TO 25:NEXT YY
9230 POSITION 14,10:? "FATAL ERROR":SO
UND 0,64,10,10
9240 FOR YY=1 TO 25:NEXT YY
9250 NEXT XX
9260 ? :? "FATAL ERROR ";ERR;" AT LINE
";ERRLN:? "DEBUG AND RESTART":? :LIST
ERRLN:END
```


FEATURES

```
9600 REM routine to disable break key
9610 BB=PEEK(16):IF BB>127 THEN BB=BB-
128:POKE 16,BB:POKE 53774,BB
9620 RETURN
10000 REM supplies data for line 1270
10010 DATA 6,28,8,28,9,28,10,10,9,14,1
0,2
```

TYPO TABLE

Variable checksum - 6450064

Line num	range	Code	Length
10	- 220	EA	561
230	- 287	GY	502
288	- 390	TM	465
400	- 460	QD	551
470	- 570	PJ	510
575	- 650	BV	439
680	- 780	MM	584
790	- 920	ZO	359
930	- 1050	NL	373
1060	- 1180	LK	452
1190	- 1243	LL	524
1245	- 1330	CO	335
1390	- 1460	CM	520
1465	- 1530	KU	381
1540	- 1900	OB	440
1910	- 2050	HM	506
2055	- 5010	CJ	459
5020	- 5210	EW	517
5220	- 6500	PM	520
6510	- 6570	BG	489
7000	- 7500	DK	486
7510	- 9030	XO	519
9040	- 9130	VV	505
9140	- 9230	DW	503
9240	- 10010	WW	282

Save the Pieces

Whenever you spend time and effort entering program code, word-processing text, or other voluminous data into your computer, be sure to save your work periodically to disk or tape. You should do this as often as every fifteen minutes or so. You won't regret it.

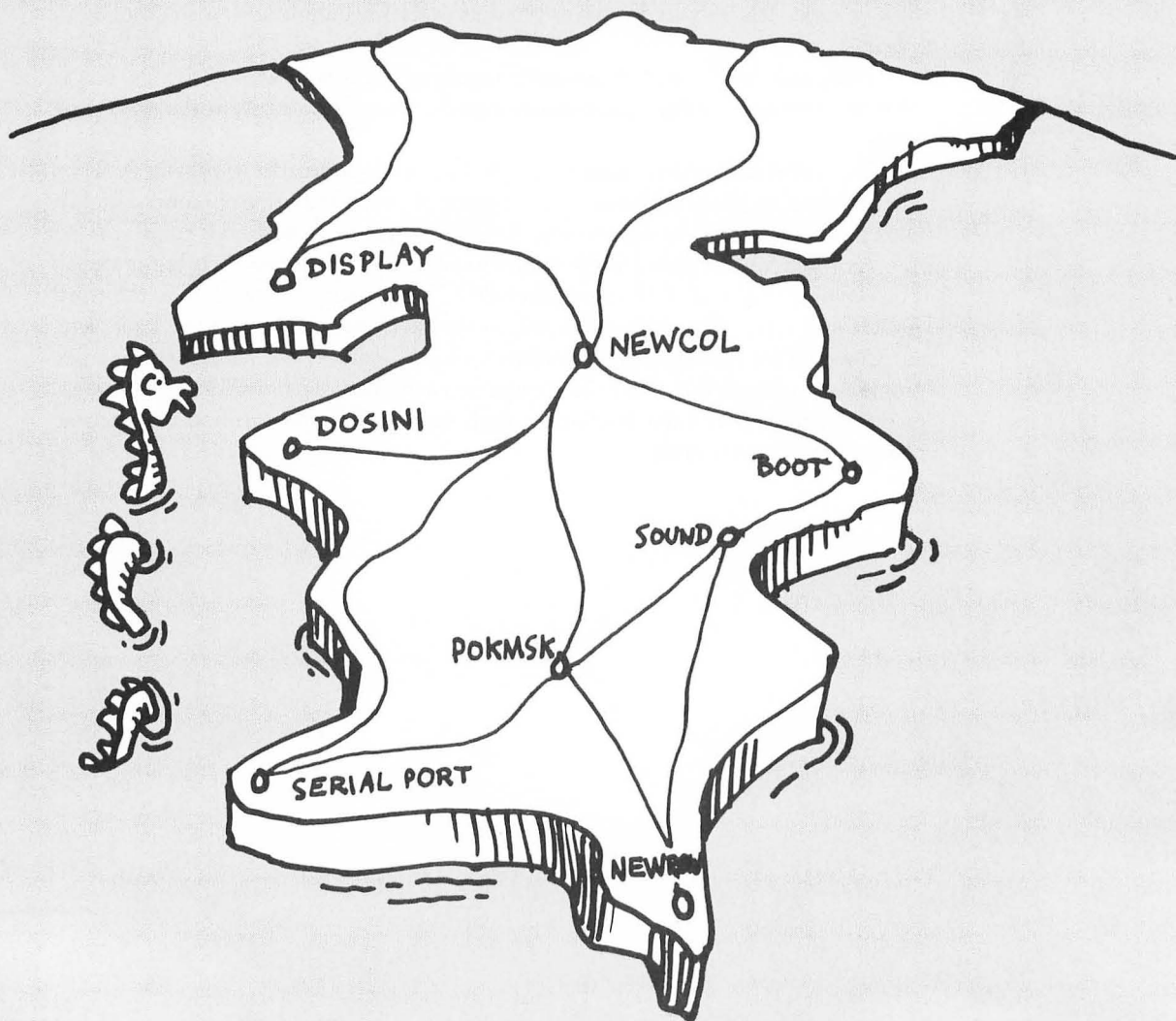
This protects you against loss of the major portion of your work if you lose power or suffer computer lockup. These conditions do occur, and usually at the very worst times.

Good intentions don't count here. You have to do it in order to benefit. A cheap kitchen timer or photo lab timer should be part of your computing paraphernalia. Just start it ticking when you start typing. You'll be surprised how soon it rings.

As you save, alternate the file names so you don't write over your last material. For example, call your first saved piece DOC1, the second save DOC2, the third DOC1 again, etc. This makes sure you always have protection against a "bad" save.

Cassettes for the Atari are notorious for loading problems. When backing up a program on cassette it is wise to save twice on each side (four times in all). Be sure to record the footage counter reading for each save so that you can find the starting places of the various saves.

Systems Guide



Memory Map

What follows in this section is a list of important locations in the silicon memory of your ATARI 400 or 800. This sequential list of memory locations is called a *memory map*. The memory inside any ATARI is divided into sections called *pages*. Each page contains 256 bytes (locations). In a 64K machine there are 256 pages of memory. Memory is further divided into RAM and ROM locations. The ROM cannot be changed by the program. It is created at the factory and contains those values and programs always available on any ATARI. The RAM memory addresses 0 - 1012 can be, and are altered by running programs. It is these low memory addresses that are described here. To adequately identify these RAM locations, our format gives the decimal value of the location, the equivalent hexadecimal (base 16) value, the number of contiguous locations serving the specified function, the name used in the Operating System listing published by Atari, and a description of the function. For example:

783 \$30F 1 CASFLG Cassette mode indicator.

This means that location 783 (30F in hexadecimal) uses one byte for the function called CASFLG, and is the cassette mode indicator.

NOTE: Hexidecimal numbers are preceeded by a dollar sign (\$). This notation is arbitrary but is the consensus method of indicating base 16 numbers.

The low memory locations that follow are used by the Operating System for housekeeping functions. Information such as location of Player/Missile data, screen colors, timer values, interrupt vectors, display list pointers and almost any other important information needed by the system to operate is stored here.

Figure 1 shows the gross memory map. This should give you some idea of where BASIC resides in memory, what part of memory is used and unused, etc.

James Capparell, a native of Rochester, New York, and graduate of the University of Rochester, is the founder and publisher of ANTIC—The ATARI Resource. His interest in ATARI computers began while working as a programmer at NASA's Ames Research Center in Mountain View, California. He also did programming for Ford Aerospace in Palo Alto. He originally obtained an ATARI to augment his NASA projects, but soon became involved with it as a hobbyist and as founder of ABACUS, the Atari Bay Area Computer Users' Society.

SYSTEMS GUIDE

Not all of the 65,536 possible memory locations are described, nor need to be. Most of memory is left “free” for the user. If you have 16K memory, your ATARI has RAM locations 0 through 16,383 available. Then accessible memory jumps to the locations reserved for cartridges, such as BASIC, from 40960 through 49151, and then to high memory where the Operating System’s ROM locations are found (See Figure 1).

Many low memory RAM locations are initialized by the Operating System in ROM when the computer is turned on. These values govern execution of user programs and are important for programmers to know. Additional information about memory is found in *De Re Atari*, the *Atari Technical User Notes*, (available from Atari Program Exchange), and in *Mapping the ATARI*, from COMPUTE! Books.

FIGURE 1

With no DOS	Address in Hexidecimal	With DOS 2.0S
Operating System RAM	0000	Operating System RAM
	1000	
	2000	DOS 2.0S
	3000	
	4000	
Free	5000	Free
RAM	6000	RAM
Space	9000	Space
	A000	
BASIC or other	B000	BASIC or other
8K cartridge	C000	8K cartridge
unallocated	D000	unallocated
hardware I/O	E000	hardware I/O
Operating System	F000	Operating System
ROM	FFFF	ROM

Page Zero

<i>1 \$0 2 LINZBS</i>	May be used to store VBLANK timer.
<i>2 \$2 2 CASINI</i>	If cassette booted successfully during powerup then JSR thru here.
<i>4 \$4 2 RAMLO</i>	Ram pointer for memory test used on powerup.
<i>6 \$6 1 TRAMSZ</i>	Temporary register for RAM size.
<i>7 \$7 1 TSTDAT</i>	RAM test data register.
<i>8 \$8 1 WARMST</i>	Warmstart flag set equal to 1 when S/RESET pushed. When set equal to 0 then powerup retry.
<i>9 \$9 1 BOOT</i>	Boot flag success indicator. When equal to 1 then successful disk boot. When equal to 2 then successful cassette boot.
<i>10 \$A 2 DOSVEC</i>	Disk software start vector.
<i>12 \$C 2 DOSINI</i>	Used to store address of initialization of application upon DOS boot. JSR indirect thru here to initialize application.
<i>14 \$E 2 APPMHI</i>	Contains highest address of RAM needed by user. Screen handler opens S: only if no RAM needed below this address.
<i>16 \$10 1 POKMSK</i>	IRQ service uses and alters POKMSK. These are POKEY interrupts. Shadow for IRQEN[\$D20E]. bit 7=1 Break key interrupt enable. bit 6=1 Other key interrupt enable. bit 5=1 Serial input data ready interrupt enable. bit 4=1 Serial output data needed interrupt enable. bit 3=1 Serial out finished interrupt enable. bit 2=1 Timer 4 interrupt enable. bit 1=1 Timer 1 interrupt enable.
<i>17 \$11 1 BRKKEY</i>	This is initialized to 1 by OS (1=no break key pressed). Monitored by keyboard, also screen editor. Break during I/O returns status of \$80. This is set to 0 when break key is pressed.
<i>18 \$12 3 RTCLOK</i>	Updated every Vblank interrupt (1/60 Sec.) Called frame counter, initialized to 0 and overflows to 0. The least significant byte of counter is \$12 and it uses 16 msec units.
<i>21 \$15 2 BUFADR</i>	Indirect buffer address register. Used as a temporary Page Zero pointer to current disk buffer.
<i>23 \$17 1 ICCOMT</i>	Command for CIO vector. Used to find correct vector to the handler routine.

SYSTEMS GUIDE

24 \$18 2 DSKFMS	Disk file manager pointer. Used as vector to FMS.
26 \$1A 2 DSKUTL	Disk utilities pointer. Points to a buffer for utilities package.
28 \$1C 1 PTIMOT	Printer timeout every printer status request. Typical timeout for the 825 is 5 seconds. Initialized to 30 sec.
29 \$1D 1 PBPNT	Print buffer pointer, index into printer buffer ranges from 0 to value of PBUFSZ.
30 \$1E 1 PBUFSZ	Print buffer size of printer record for current mode. normal = 40 bytes double width = 20 bytes sideways = 29 bytes (Atari 820 printer) status = 4
31 \$1F 1 PTEMP	Printer handler uses this temp register to save value of character to output to printer.
32 \$20 1 ZIOCB	Handler index number into the device name table for currently opened file. Set to 255 if no file opened.
33 \$21 1 ICDNOZ	Device # (DRIVE #). Initialized to 1.
34 \$22 1 ICCOMZ	Command code.
35 \$23 1 ICSTAZ	Status of last IOCB action.
36 \$24 2 ICBALZ	Buffer address for data transfer.
38 \$26 2 ICPTLZ	Put byte routine (address -1) set by OS.
40 \$28 2 ICBLIZ	Buffer length byte count used by PUT and GET commands.
42 \$2A 2 ICAZIZ	Auxiliary information first byte used in OPEN to specify type of file access.
43 \$2B 1 ICAX2Z	Auxiliary information second byte. CIO working variables.
44 \$2C 2 ICSPRZ	Spare bytes local CIO use.
46 \$2E 1 ICSPRZ	IOCB Number multiplied by 16.
47 \$2F 1 CIOCHR	Character byte for current operation.
48 \$30 1 STATUS	Internal status storage.
49 \$31 1 CHKSUM	Single byte sum with carry to least significant bit.
50 \$32 2 BUFRLO	Pointer to data buffer transmitted during I/O operation.
52 \$34 2 BFENLO	Next byte past end of data buffer.
54 \$36 1 CRETRY	Number of command frame retries. Default is 13.
55 \$37 1 DRETRY	Number of device retries. Default is one.
56 \$38 1 BUFRFL	Buffer full flag. (255 indicates full).
57 \$39 1 RECVDN	Receive done flag. (255 indicates done).
58 \$3A 1 XMTDON	Transmission done flag. (255 indicates done).

59 \$3B 1 CHKSNT	Checksum sent flag. (255 indicates done).
60 \$3C 1 NOCKSM	No checksum follows data flag. Zero indicates checksum follows transmission.
61 \$3D 1 BPTR	Cassette record data index into data portion of record being read or written. Values range 0 to current value BLIMI [\$28A]. When BPTR=BLIM then buffer CASBOFF [\$3FD] is empty if reading or full if writing.
62 \$3E 1 FTYPE	Interecord gap type. Copy of ICAX2Z from open command. (0 indicates continuous gaps; non-zero indicates normal gaps.)
63 \$3F 1 FEOF	Cassette end of file flag used by cassette handler to indicate end of file.
64 \$40 1 FREQ	Beep count. Retain and count number of beeps requested of beep routine by cassette handler during open processing; one beep for play, two for record.
65 \$41 1 SOUNDR	Noisy I/O flag, when I/O is done buzzer sounds. POKE 0 and it won't buzz.
66 \$42 1 CRITIC	Defines critical section (if non-zero) checked on NMI process after stage 1 processed.
67 \$43 7 FMSZPO	Disk file manager zero page.
74 \$4A 1 CKEY	Cassette boot request flag on powerup (coldstart). Start key checked, if pressed then CKEY is set.
75 \$4B 1 CASSBT	Cassette boot flag.
76 \$4C 1 DSTAT	Display status used by display handler.
77 \$4D 1 ATTRACT	Attract flag set to 0 by IRQ whenever a key is pressed. Incremented every 4 seconds by stage 1 Vblank. When value is < 127 then value is set to \$FE until attract mode is terminated.
78 \$4E 1 DRKMSK	Dark attract mask=\$FE when attract mode inactive.
79 \$4F 1 COLRSH	Attract color shifter XOR'd with playfield colors. At stage 2 Vblank color registers are XOR'd with COLRSH and DRKMSK then sent to hardware color registers. When attract inactive COLRSH=0 and DRKMSK=\$F6 reducing luminance 50% and COLRSH=RTCLOCK+1 affecting color change every $256/60=4.1$ sec.
80 \$50 1 TEMP	Used by display handler in moving data to and from screen.
81 \$51 1 HOLD1	Same as [\$50]. When BASIC in use these 2 locations

SYSTEMS GUIDE

	called LOMEM and point to 256 byte buffer at end of OS. RAM used to tokenize one line of BASIC.
82 \$52 1 LMARGN	Column of left margin of text screen, initialized to 2.
83 \$53 1 RMARGN	Column of right margin of text screen initialized to 39. Margins are user alterable. Ignored in every mode but 0.
84 \$54 1 ROWCRS	Display row number used in graphics screen and mode 0. Range 0 - 191. This location and COLCRS define the cursor location for the next data element to be read/written to main screen segment.
85 \$55 2 COLCRS	Display column number used in graphics and mode 0 (lobyte). Range 0 - 319 (hibyte). Home position is 0,0 for both graphics and text.
87 \$57 1 DINDEX	Display mode current screen mode obtained from low order 4 bits of most recent open AUX1 byte.
88 \$58 2 SAVMSC	Lowest address of display memory this location corresponds to the upper left corner of screen.
90 \$5A 1 OLDROW	These next 3 locations are updated from ROWCRS and COLCRS before every operation. This location is used by DRAWTO and XIO to determine starting row.
91 \$5B 2 OLDCOL	These variables used only in draw and fill commands.
93 \$5D 1 OLDCHR	Retains value of character under visible text cursor. Used to restore character after cursor moves.
94 \$5E 2 OLDADR	Retains memory address of current visible text cursor location. Used in conjunction with OLDCHR to restore character value after cursor moves.
96 \$60 1 NEWROW	Indicates row location that the DRAWTO and XIO fill routine will use.
97 \$61 2 NEWCOL	Indicates column DRAWTO and XIO will go.
99 \$63 1 LOGCOL	Points at cursor position in logical line. A logical line can contain up to 3 physical lines. This variable is used by display handler and ranges from 0 to 119.
100 \$64 2 ADRESS	Temporary storage holds contents of SAVMSC [\$58].
102 \$66 2 MLTTMP	OPNTMP first byte used in open as temporary storage.
104 \$68 2 SAVADR	Temporary storage.
106 \$6A 1 RAMTOP	RAM size defined by power on logic. This value is

MEMORY MAP

<i>107 \$6B 1 BUFCNT</i>	given in pages (page = 256 bytes of memory).
<i>108 \$6C 2 BUFSTR</i>	Screen editor current logical line size.
<i>110 \$6E 1 BITMSK</i>	Editor low byte.
<i>111 \$6F 1 SHFAMT</i>	Used in bit mapping routines by OS display handler.
<i>112 \$70 2 ROWAC</i>	Pixel justification.
<i>114 \$72 2 COLAC</i>	Control for row and column point plotting.
<i>116 \$74 2 ENDPT</i>	Controls column point plotting.
	Contains larger of DELTAR and DELTAC used in conjunction with ROWAC/COLAC to control plotting of line points.
<i>118 \$76 1 DELTAR</i>	Contains absolute value of NEWROW minus ROWCRS.
<i>119 \$77 2 DELTAC</i>	Contains absolute value NEWCOL minus COLCRS. These values and ROWINC and COLINC define slope of line to be drawn.
<i>121 \$79 1 ROWINC</i>	Row increment +1 or -1.
<i>122 \$7A 1 COLINC</i>	Column increment +1 or -1.
<i>123 \$7B 1 SWPFLG</i>	Split screen cursor control.
<i>124 \$7C 1 HOLDCH</i>	Character moved here before control and shift logic processed.
<i>125 \$7D 1 INSDAT</i>	Temporary storage used by display handler.
<i>126 \$7E 2 COUNTR</i>	Initially contains larger of DELTAR and DELTAC which is number of iterations to generate a line. This value decremented after every point is plotted. When =0 then line is finished.
<i>128 \$80 2 LOMEM</i>	This points to a 256 byte buffer used to tokenize one line of BASIC. This buffer is located at the end of the O.S.RAM.
<i>130 \$82 2 VNTP</i>	Points to list of all variable names used in a program. Each name is stored in the order entered. Maximum of 128 names.
<i>132 \$84 2 VNTD</i>	Points to end of variable name table. Points to a zero byte when all 128 names not used.
<i>134 \$86 2 VVTP</i>	Points to variable value table. Eight bytes allocated for each variable in name table.
<i>136 \$88 2 STMTAB</i>	Points to statement table which contains the tokenized BASIC statements. Also the immediate mode lines.
<i>138 \$8A 2 STMCUR</i>	The BASIC interpreter uses this pointer to access the tokens within a line of the statement table.

SYSTEMS GUIDE

140 \$8C 2 STARP	Points to the block containing all the string and array data. Memory is reserved and enlarged whenever a dimension statement is encountered. Strings are stored one byte (ATASCII) per character. Arrays are stored as six byte BCD (Binary Coded Decimal) per element.
142 \$8E 2 RUNSTK	Points to the software run time stack. The stack maintains GOSUB and FOR/NEXT entries. The POP statement affects this stack.
144 \$90 2 MEMTOP	Points to the end of the user program. The FRE function returns the value calculated by subtracting the contents of this location from the contents of HIMEM at \$2E5 and \$2E6. Don't confuse this MEMTOP with the O.S. variable of the same name at \$2E5.

The remainder of Page Zero is used by BASIC cartridge, floating point routines and assembler cartridges.

Page One is the stack area and is not available for use by programmers.

Page Two

512 \$200 2 VDSLST	Initialized to [\$E7B3] if NMI interrupt occurred and it was caused by a DLI then JMP thru here. Since the OS does not use DLIs this is initialized to point to an RTI.
514 \$202 2 VPRCED	Initialized to [\$E7B2] if IRQ interrupt occurred due to serial I/O bus proceed line then JMP thru here.
516 \$204 2 VINTER	Initialized to [\$E7B2] if IRQ interrupt due to serial I/O bus interrupt then JMP thru here.
518 \$206 2 VBREAK	Initialized to [\$E7B2] if IRQ interrupt due to 6502 BRK instruction execution then JMP thru here.
520 \$208 2 VKEYBD	Initialized to [\$FFBE] if IRQ interrupt due to keypress then JMP thru here to keyboard handler.
522 \$20A 2 VSERIN	Initialized to [\$EB11] if IRQ interrupt due to I/O bus input ready then JMP thru here.
524 \$20C 2 VSEROR	Initialized to [\$EA90] if IRQ interrupt due to I/O bus output ready then JMP thru here.
526 \$20E 2 VSEROC	Initialized to [\$EAD1] if IRQ interrupt due to I/O bus output complete then JMP thru here.
528 \$210 2 VTIMR1	POKEY timer 1 interrupt vector.

MEMORY MAP

530 \$212 2 VTIMR2	POKEY timer 2 interrupt vector.
532 \$214 2 VTIMR4	POKEY timer 4 interrupt vector.
534 \$216 2 VIMIRQ	Initialized to [\$E6F6] if IRQ interrupt occurs then JMP thru here to determine cause.
536 \$218 2 CDTMV1	SIO timeout decremented at every VBLANK stage 1 when this location counts down to 0 then JSR thru CDTMA1 [\$226].
538 \$21A 2 CDTMV2	Timer decremented at almost every VBLANK subject to critical section test (stage 2 process).
540 \$21C 2 CDTMV3	Timer decremented at almost every VBLANK subject to critical section test (stage 2 process).
542 \$21E 2 CDTMV4	Timer same as 2 & 3.
544 \$220 2 CDTMV5	Timer same as 2,3 & 4. 3,4,5 set flags CDTMF3=\$22A CDTMF4=\$22C and CDTMV5=\$22E when they equal zero.
546 \$222 2 VVBLKI	Initialized to [\$E701] stage 1 vertical blank vector NMI interrupt.
548 \$224 2 VVBLKD	Initialized to [\$E93E] system return from interrupt.
550 \$226 2 CDTMA1	SIO timeout vector. When CDTMV1 [\$218] times out it vectors through here.
552 \$228 2 CDTMA2	NO SYSTEM FUNCTION. Available to user enter address of routine to be executed at timer count down to 0.
554 \$22A 1 CDTMF3	Byte flag set when CDTMV3 [\$21C,21D] counts down to 0.
555 \$22B 1 SRTIMR	Software repeat timer, controlled by IRQ device routine, establishes initial ½ second delay before key will repeat. Stage 2 Vblank establishes 1/10 second repeat rate. Decrements timer, implements auto repeat logic.
556 \$22C 1 CDTMF4	Byte flag set when CDTMV4 [\$21E] counts down to 0.
557 \$22D 1 INTEMP	Used by SETVBL routine.
558 \$22E 1 CDTMF5	Byte flag set when CDTMV5 [\$220] counts to 0.
559 \$22F 1 SMDCTL	Shadow for DMACTL [\$D400] default value \$22. bit 5=1 enable Display List instruction fetch DMA. bit 4=1 enable 1 line P/M resolution. =0 enable 2 line P/M resolution. bit 3=1 enable Player DMA. bit 2=1 enable Missile DMA.

SYSTEMS GUIDE

	bit 1,0=0 0 no Playfield DMA. =0 1 narrow Playfield DMA 128 color clocks. =1 0 standard Playfield DMA 160 clocks. =1 1 wide Playfield DMA 192 clocks.
560 \$230 2 <i>SDLSTL</i>	Shadow for DLISTL [\$D402]. This location initialized to Start of Display List.
562 \$232 1 <i>SSKCTL</i>	Shadow for SKCTL [\$D20F]. bit 7=1 force break serial output to 0. bit 6,4=serial port mode control. bit 3=1 serial output transmitted as 2-tone instead of logic true/false. bit 2=1 pot counter completes within 2 scan lines instead of 1 frame time. bit 1=1 enable keyboard scanning circuit.
563 \$233 1 <i>SPARE</i>	NO OPERATING SYSTEM FUNCTION.
564 \$234 1 <i>LPENH</i>	Light pen horizontal value shadow for [\$D40C]. Value range 0-227 wrap to 0 at right edge of standard width screen.
565 \$235 1 <i>LPENV</i>	Light pen vertical value shadow for [\$D40D]. Value same as VCOUNT 2 line resolution. Both pen values modified if any joystick trigger lines pulled low.
566 \$236 4 <i>SPARE</i>	NO OPERATING SYSTEM FUNCTION.
570 \$23A 1 <i>CDEVIC</i>	SIO bus I.D. number.
571 \$23B 1 <i>CCOMND</i>	SIO bus command code.
572 \$23C 1 <i>CAUXI</i>	SIO auxiliary byte loaded from location 778.
573 \$23D 1 <i>CAUX2</i>	SIO command auxiliary byte loaded from location 779.
574 \$23E 1 <i>TEMP</i>	Receives one-byte responses from serial bus controllers.
575 \$23F 1 <i>ERRFLG</i>	SIO error flag. Indicates any device error except timeout errors.
576 \$240 1 <i>DFLAGS</i>	Disk flags from sector 1, contains value of first byte of boot file.
577 \$241 1 <i>DBSECT</i>	Number of disk boot sectors.
578 \$242 2 <i>BOOTAD</i>	Address where disk boot loader will be put.
580 \$244 1 <i>COLDST</i>	Coldstart flag when = 1 then powerup in progress. When = 0 then S/RESET in progress. If set = 1 during normal program execution then S/RESET will act like powerup giving some protection.
581 \$245 1 <i>SPARE</i>	NO OPERATING SYSTEM FUNCTION.

MEMORY MAP

582 \$246 1 DSKTIM	Disk timeout register.
583 \$247 40 LINBUF	Forty byte character line buffer used to temporarily buffer one physical line of text when screen editor is moving screen data.
623 \$26F 1 GPRIOR	Global priority shadow for PRIOR [\$D01B] controls priority of player/missile/playfield.
624 \$270 8 PADDL0- PADDL7	These locations store values returned when paddles are used. Values are between 0 and 228.
632 \$278 4 STICK0- STICK3	These locations store values returned when a joystick is used. There are 9 possible values. These locations are shadows (duplicates) of ROM locations 53760-53767.
636 \$27C 8 PTRIG0- PTRIG7	These locations store values of trigger on paddles.
644 \$284 4 STRIG0	Joystick trigger 0 - 3.
648 \$288 1 CSTAT	Cassette status register.
649 \$289 1 WMODE	Used by cassette handler as read/write mode flag. Zero = read; \$80 = write.
650 \$28A 1 BLIM	Cassette record data size count of number of data bytes being read. Range 1-128 depends on record control byte.
651 \$28B 4 SPARE	NO OPERATING SYSTEM FUNCTION. Use of these bytes in your program may conflict with later OS upgrades.
656 \$290 1 TXTROW	Text window row cursor range 0-3. Specifies where next read/write will occur.
657 \$291 2 TXTCOL	Text window column cursor range 0-39 used in split screen. These two variables give cursor position.
659 \$293 1 TINDEX	Split screen text window graphics mode. Index always = 0. When SWPFLG [\$7B]=0. This is split screen equivalent of DINDEX.
660 \$294 2 TXTMSC	Split screen text window version SAVMSC [\$58].
662 \$296 6 TXTOLD	Old row and old column for text and then some split screen cursor data.
668 \$29C 1 TMPXI	Temporary storage.
669 \$29D 1 HOLD3	Used by the display handler to hold scroll loop counter.
670 \$29E 1 SUBTMP	Temporary storage. Unknown function.
671 \$29F 1 HOLD2	Temporary storage. Unknown function.
672 \$2A0 1 DMASK	Pixel location mask.

SYSTEMS GUIDE

673 \$2A1 1 <i>TMPLBT</i>	Temporary storage for bit map.
674 \$2A2 1 <i>ESCFLG</i>	Used by screen editor. Flag set to \$80 when ESC[\$1B] character detected. Reset to 0 following output of next character. Causes character following ESC to be displayed, only exception is EOL [\$9B].
675 \$2A3 15 <i>TABMAP</i>	Indicates where tab stops are set. There are 120 possible tab stops in one logical line.
690 \$2B2 4 <i>LOGMAP</i>	Logical line bitmap. When a bit is set then a logical line starts at the corresponding physical row number. All bits set to 1 when text screen is opened or cleared.
694 \$2B6 1 <i>INVFLG</i>	Inverse video flag toggled by ATARI logo key sets bit 7 = 1
695 \$2B7 1 <i>FILFLG</i>	Indicates to display handler whether current operation is Fill (not 0) or Draw (0)
696 \$2B8 1 <i>TMPROW</i>	Temporary storage used by ROWCRS [\$54].
697 \$2B9 2 <i>TMPCOL</i>	Temporary storage used by COLCRS [\$55].
699 \$2BB 1 <i>SCRFLG</i>	Scroll flag set to number of physical lines minus 1 that were deleted from top of screen. Since logical lines range from 1 - 3 physical lines then this variable ranges from 0 - 2.
700 \$2BC 1 <i>HOLD4</i>	Used to save and restore value in ATACHR[\$2FB] during fill process when ATACHR is temporarily set to value in FILDAT[\$2FD].
701 \$2BD 1 <i>HOLD5</i>	Similar function to HOLD4.
702 \$2BE 1 <i>SHFLOK</i>	Shift/control lock flag initialized to \$40 at powerup \$00=normal mode lower case alpha \$61-\$7A \$40=caps lock upper case \$41-\$5A \$80=control lock \$01-\$1A
703 \$2BF 1 <i>BOTSCR</i>	Bottom of screen. If = 4, then mixed graphics mode. If = 24 then normal text mode.
704 \$2C0 4 <i>PCOLR0 -</i> <i>PCOLR3</i>	Player color registers and shadows = COLPM0[\$D012] = COLPM1[\$D013] = COLPM2[\$D014] = COLPM3[\$D015]
<i>PCOLR0</i>	
<i>PCOLR1</i>	
<i>PCOLR2</i>	
<i>PCOLR3</i>	
708 \$2C4 5 <i>COLOR0 -</i> <i>COLOR4</i>	Playfield color registers and shadows. = COLPF0[\$D106]
<i>COLOR0</i>	

MEMORY MAP

<i>COLOR1</i>	= COLPF1[\$D017]
<i>COLOR2</i>	= COLPF2[\$D018]
<i>COLOR3</i>	= COLPF3[\$D019]
<i>COLOR4</i>	= COLBK[\$D01A]
<i>713 \$2C9 23 SPARE</i>	**
<i>736 \$2E0 2 GLBABS</i>	Contains entry address of code for auto-boot/run.
<i>738 \$2E1 2 SPARE</i>	**
<i>740 \$2E4 1 RAMSIZ</i>	Size in pages (page = 256 bytes) of available RAM permanently retains RAM top address contained in TRAMSZ[\$6]. With BASIC and 48K installed this equals \$160=40960 bytes.
<i>741 \$2E5 2 MEMTOP</i>	Top of available user memory RAMSIZ less display list and display memory (first nonuseable program address). This value established by powerup logic and reset. Re-established when screen display is opened.
<i>743 \$2E7 2 MEMLO</i>	Bottom of available user memory established at powerup and reset, not altered after that.
<i>745 \$2E9 1 SPARE</i>	**
<i>746 \$2EA 4 DVSTAT</i>	Device status buffer Get status command puts information in these bytes.
<i>750 \$2EE 2 CBAUDL</i>	Cassette baud rate low byte.
<i>752 \$2F0 1 CRSINH</i>	Cursor inhibit flag. When equal to 0 then cursor turned on. If not equal 0 then no visible cursor.
<i>753 \$2F1 1 KEYDEL</i>	Key delay set to 3 whenever key code accepted. Decrement every 1/60 sec by stage 2 VBLANK process until it reaches 0.
<i>754 \$2F2 1 CH1</i>	Prior key code read and accepted. Current key pressed compared with contents of CH1 if same then debounce time checked if OK then accepted. If current key not the same as CH1 then accepted. When code is accepted then stored in CH[\$2FC].
<i>755 \$2F3 1 CHACT</i>	Shadow for CHACTL[\$D401] character control register.
<i>bit 2 = 1</i>	Causes current character line to invert, sampled at every char. line.
<i>bit 1 = 2</i>	In 40 char. mode if bit 7 of current char. code = 1 then char. is blue on white.
<i>bit 0 = 1</i>	In 40 char. mode if bit 7 of current char. code = 1 then char. will be blank. Blinking char. produced by

SYSTEMS GUIDE

	setting bit 7 of char. and periodically changing bit 0 here.
756 \$2F4 1 CHBAS	Vector to page address of character set initialized to \$E0 (upper case and punctuation). The character set in ROM is located at \$E000-\$E3FF, shadow for CHBASE[\$D409].
757 \$2F5 5 SPARE	**
762 \$2FA 1 CHAR	Contains internal code corresponding to what is in ATACHR[\$2FB]. This will be converted to ATASCII code.
763 \$2FB 1 ATACHR	ATASCII value for most recent character read or written or value of the graphics point. This value also determines color of line in draw and fill commands.
764 \$2FC 1 CH	Holds keyboard code for a character (not ATASCII). Keyboard handler gets all data from here when it gets a character it writes \$FF here to indicate code read. This location loaded when a key is pressed causing an IRQ interrupt which vectors at \$208. This interrupt service routine loads the code into \$2FC for processing at VBLANK stage 2.
765 \$2FD 1 FILDAT	Color to be used by XIO FILL Command.
766 \$2FE 1 DSPFLG	Display flag will allow control codes other than EOL [\$98B] to be displayed if flag not equal to 0. If flag = 0 then control codes processed normally.
767 \$2FF 1 SSFLAG	Start/stop flag toggled by control-1 keys cleared by break key, reset key, or powerup

Page Three

DCB DEVICE CONTROL BLOCK

Used for handler/SIO communication and between user and disk handler.

768 \$300 1 DDEVIC	Pheripheral unit bus I.D. number.
769 \$301 1 DUNIT	Unit number.
770 \$302 1 DCOMND	Bus command.
771 \$303 1 DSTATS	Command type status return.
772 \$304 2 DBUFLO	Data buffer pointer. Set by handler to indicate source or destination data buffer.
774 \$306 1 DTIMLO	Device timeout in 64/80 second units. Set by handler.
775 \$307 1 DUNUSE	Unused in DCB.

MEMORY MAP

776 \$308 2 DBYTLO	Number of bytes to be transferred into or out of data buffer. Not required if no data transfer.
778 \$30A 1 DAUX1	Command auxiliary byte 1. Device specific information set by handler.
779 \$30B 1 DAUX2	Command auxiliary byte 2.
780 \$30C 2 TIMER1	Initial timer value.
782 \$30E 1 ADDCOR	Used for interpolation adjustment of baud rate.
783 \$30F 1 CASFLG	Cassette mode when set.
784 \$310 2 TIMER2	Final timer value used with TIMER1 to compute interval for baud rate.
786 \$312 2 TEMP1	Temporary storage.
788 \$314 1 TEMP2	**
789 \$315 1 TEMP3	**
790 \$316 1 SAVIO	Save serial data port.
791 \$317 1 TIMFLG	Time out flag for baud rate correction.
792 \$318 1 STACKP	SIO stack pointer save cell.
793 \$319 1 TSTAT	Temporary status holder.
794 \$31A 38 HTABS	Start of handler address table.

IOCB START OF I/O CONTROL BLOCKS

Used to communicate information between user program and CIO.

Space reserved for 8 IOCBs.

832 \$340 1 ICHID	Handler index number (\$FF = IOCB unused).
833 \$341 1 ICDNO	Device number (drive number).
834 \$342 1 ICCOM	Command code.
835 \$343 1 ICSTA	Status of last IOCB action.
836 \$344 2 ICBAL	Buffer address.
838 \$346 2 ICPTL	Put byte routine address minus 1.
840 \$348 2 ICBLL	Buffer length.
842 \$34A 1 ICAX1	Auxiliary information first byte.
843 \$34B 1 ICAX2	Auxiliary information second byte.
844 \$34C 3 ICSPR	*** Spare bytes ***
860 \$350 16 IOCB # 1	
876 \$360 16 IOCB # 2	
892 \$370 16 IOCB # 3	
908 \$380 16 IOCB # 4	
924 \$390 16 IOCB # 5	
940 \$3A0 16 IOCB # 6	
956 \$3B0 16 IOCB # 7	
972 \$3C0 40 PRNBUF	Printer buffer.
1012 \$3E8 21 SPARE	*** Spare bytes ***

Scrolling

The strongest features of the ATARI relative to game programming are the 12 Graphics Modes (high resolution 320 x 192); two direct memory access (DMA) video channels (sort of a simplified multiprocessor system); display-list controlled, memory-mapped graphics; redefinable character sets; hooks for vertical blank interrupts and scan line interrupts; and, of course, four channels of sound.

The ATARI maps its memory to video *via* an LSI chip called ANTIC. This chip is a dedicated processor with its own instruction set. These instructions make up what is called a *display list*. The display list controls the Graphics Mode which will be displayed on the screen. Recall that there are 12 modes, each specifying memory use, resolution and color. The display list tells ANTIC what part of the 6502 memory space to display, what mode to display, whether an interrupt should be generated, and whether horizontal and/or vertical scrolling should be enabled. It is this last feature which will be demonstrated here.

There are two methods which can be used to scroll the image. The first is direct and easy to comprehend. The display list has, as part of its instructions, a feature called Load Memory Scan (LMS). This operator is three bytes long. The last two bytes are the address (lo-byte/hi-byte, 6502 style) of the start of display memory. As a result, the entire address space is available for display under program control. This gives the observer a "window" into memory. Scrolling windows are created by simply changing the two address bytes of the LMS. In other words, it is not data being moved through memory, but a window moving across the data residing in memory which causes the image to scroll.

Listing 1 should give a good idea of "coarse" vertical scrolling. I call it coarse because the image moves a full character width at a time. Lines 170 and 180 are really doing all the dirty work. The new display address is being inserted into the display list at this point after appropriate incrementing or

"Scrolling," by James Caparell, is reprinted with permission from MICRO Magazine, Issue No. 42, P.O. Box 6502, Amherst, NH 03031.

decrementing of the address bytes. I've chosen to vertically scroll the entire image but it is an easy matter to set up a scrolling window within a background display. In fact, Listing 2 does just that, only in the horizontal direction.

I've also mixed two modes on the screen. The only complication here is the need to have more than one LMS instruction. The second LMS restores the pointer to memory prior to the horizontal intrusion. There is nothing to stop you from placing an LMS instruction on every mode line; each could be scrolling in independent directions.

Listing 3 is meant to demonstrate the second scrolling method, smooth or fine scrolling. This is accomplished with the help of hardware scrolling registers, one for horizontal and another for vertical direction. When the appropriate bits are set in a display list instruction, the values in each of these registers control the number of scan lines vertically or color clocks horizontally that each line will be displaced. The limitation here is the amount of fine scrolling allowed. A line can be moved eight full color-clocks horizontally and 16 scan lines vertically. When this amount is scrolled, the LMS address bytes must be incremented or decremented and the whole process must be started again. in this way smooth scrolling can be maintained.

by James Capparell

```
10 REM COARSE VERTICAL SCROLLING DEMO
15 REM PRESS UP/DOWN ARROWS TO MOVE DI
   SPLAY THRU MEMORY
20 DLIST=PEEK(560)+PEEK(561)*256:REM G
   ET START OF DISPLAY LIST
30 LMSL=DLIST+4:REM POINTER TO DISPLAY
   MEMORY
40 LMSH=DLIST+5
50 DISPLAYL=0:REM INITIALIZE ADDRESS O
   F DISPLAY MEMORY
55 REM READ KEYBOARD
60 IF PEEK(764)=255 THEN GOTO 60:REM W
   AIT FOR KEY
70 IF PEEK(764)=14 THEN POKE 764,255:G
   OTO 110:REM UP ARROW /
80 IF PEEK(764)=15 THEN POKE 764,255:G
   OTO 140:REM DOWN ARROW ?
90 GOTO 60
```

SYSTEMS GUIDE

```
100 REM MOVE DISPLAY WINDOW INTO LOWER
    MEMORY
110 DISPLAYL=DISPLAYL-40
120 IF DISPLAYL>=0 THEN GOTO 170:REM C
    AN'T DISPLAY NEGATIVE MEMORY
122 DISPLAYH=DISPLAYH-1:DISPLAYL=0
124 IF DISPLAYH<0 THEN DISPLAYH=0
126 GOTO 170
130 REM MOVE DISPLAY WINDOW INTO HIGHE
    R MEMORY
140 DISPLAYL=DISPLAYL+40
150 IF DISPLAYL>240 THEN DISPLAYH=DISP
    LAYH+1:DISPLAYL=0
160 REM CHANGE DISPLAY MEMORY POINTER
170 POKE LMSL,DISPLAYL:REM PUT NEW DIP
    LAY ADDR IN DISPLAY LIST
180 POKE LMSH,DISPLAYH
200 GOTO 60:REM GO WAIT FOR KEYBOARD E
    NTRY
```

```
10 REM COARSE HORIZONTAL SCROLLING DEM
    O
20 REM USE LEFT AND RIGHT POINTING ARR
    OWS TO CONTROL SCROLL DIRECTION
25 LIST
30 DLST=PEEK(561)*256+PEEK(560)
35 DMEM=PEEK(DLST+4)+PEEK(DLST+5)*256
40 SKIPH=INT((DMEM+280)/256):SKIPL=DME
    M+280-SKIPH*256
45 POKE DLST+15,66:POKE DLST+16,SKIPL:
    POKE DLST+17,SKIPH
50 ADDR=DLST+13:ADDRH=DLST+14:VALL=0:
    VALH=3
55 POKE DLST+12,71:POKE ADDR,VALL:POK
    E ADDRH,VALH
60 IF PEEK(764)=255 THEN GOTO 60:REM S
    CAN KE
65 IF PEEK(764)=7 THEN POKE 764,255:GO
```

```

TO 100:REM RIGHT ARROW ?
70 IF PEEK(764)=6 THEN POKE 764,255:GO
TO 140:REM LEFT ARROW ?
80 GOTO 50:REM ONLY ARROWS ARE LEGAL R
ESPONSE
90 REM SCROLL RIGHT
100 VALL=PEEK(DLST+13)+1:REM MOVE DISP
LAY TO LEFT
110 IF VALL>255 THEN VALL=0:VALH=VALH+
1
120 GOTO 55
130 REM SCROLL LEFT
140 VALL=PEEK(DLST+13)-1:REM MOVE DISP
LAY TO RIGHT
150 IF VALL<0 THEN VALL=0:VALH=VALH-1
160 IF VALH<0 THEN VALH=0
170 GOTO 55

```

```

10 REM FINE SCROLLING HORIZONTALLY AND
VERICALLY
20 DLST=PEEK(560)+256*PEEK(561)
25 DMEM=PEEK(DLIST+4)+PEEK(DLST+5)*256
30 SKIPH=INT((DMEM+280)/256):SKIPL=DME
M+280-SKIPH*256
35 VALL=0:VALH=2
40 POKE DLST+12,119:POKE DLST+13,VALL:
POKE DLST+14,VALH
45 POKE DLST+15,66:POKE DLST+16,SKIPL:
POKE DLST+17,SKIPH
50 IF PEEK(764)=255 THEN GOTO 50:REM S
CAN KEYBOARD
55 IF PEEK(764)=14 THEN POKE 764,255:G
OTO 200:REM UP ARROW ?
60 IF PEEK(764)=15 THEN POKE 764,255:G
OTO 250:REM DOWN ARROW ?
65 IF PEEK(764)=6 THEN GOTO 300:REM LE
FT ARROW ?

```

SYSTEMS GUIDE

```
70 IF PEEK(764)=7 THEN GOTO 350:REM RI
GHT ARROW ?
75 GOTO 50:REM IGNORE OTHER RESPONSES
200 Y=Y+1:IF Y<16 THEN GOTO 500
210 Y=0
215 VALL=VALL+40
220 IF VALL>240 THEN VALL=0:VALH=VALH+
1
230 GOTO 450
250 Y=Y-1
255 IF Y>-1 THEN GOTO 500
260 Y=15
265 VALL=VALL-40
280 GOTO 445
300 X=X-1:IF X>-1 THEN GOTO 505
305 X=15
310 VALL=PEEK(DLST+13)+2
325 GOTO 445
350 X=X+1:IF X<16 THEN GOTO 505
355 X=0
360 VALL=PEEK(DLST+13)-2
440 IF VALL<0 THEN VALL=0:VALH=VALH-1
445 IF VALH<0 THEN VALH=0
450 POKE DLST+12,119:POKE DLST+13,VALL
:POKE DLST+14,VALH
500 POKE 54277,Y:REM VERTICAL SCROLL R
EGISTER
505 POKE 54276,X:REM HORIZONTAL SCROLL
REGISTER
510 GOTO 50
```

ANTIC Disassembler and Raster Scan Graphics

Recall that the display list is the set of instructions used to control an LSI chip called ANTIC. ANTIC, a dumb microprocessor, functions as a graphics controller. Its principal duties are to specify the location in memory to be displayed, the mode of display (12 Graphics Modes with differing resolutions to choose from), horizontal/vertical scroll enable and display list instruction interrupt enable.

Included here is an ANTIC disassembler. This program requires you to enter a BASIC Graphics Mode numbered 0-11, and will then locate the associated display list and decode the instructions. Note that this program prints the ANTIC display modes numbered 2 - 15. Use the program and the ANTIC/BASIC correspondences will become apparent. (See Listing 1.) Also described are basic raster scan graphics (ATARI style), and then a quick lesson in the use of display list interrupts.

The normal NTSC raster television is made up of 625 interlaced scan lines. These scan lines are the horizontal lines appearing in the picture tube phosphor when energized by the electron beam as it sweeps left to right, top to bottom, across your screen. Interlacing occurs in normal television to eliminate flicker. It simply means that all even scan line rows are "painted" in one frame, and all odd lines in the next. The frame refresh rate is 60 Hz.

Each ATARI frame image contains 262 scan lines with no interlacing. Every frame is the duplicate of the prior one unless there is program intervention. The image is repainted 60 times per second, and the electron beam is turned off at the end of every scan line. At that time it is returned to the left edge of the screen to start the next line trace. This is called horizontal blank time.

The beam is also turned off after every frame so that it may return to the

"ANTIC Disassembler," by James Caparelli, is reprinted with permission from MICRO Magazine, Issue No. 43, P.O. Box 6502, Amherst, NH 03031.

SYSTEMS GUIDE

top left corner of the screen. This is called vertical blank time. These two time periods are very important to the would-be animator. It is crucial to understand how much time is available and how to enter code so that it will be executed at the appropriate moment.

The 6502 microchip in the ATARI cycles at 1.79 megahertz, almost twice as fast as the normal 6502. This cycle rate was chosen so that two color-clock widths on a scan line equal one machine cycle. There are 228 color-clocks on every scan line, and the maximum displayable width of any scan line is 176 color-clocks, called “wide playfield” in the ATARI literature. The maximum resolution is $\frac{1}{2}$ color-clock, and therefore ATARI can display up to 352 picture elements (pixels) horizontally. The maximum vertical resolution, in scan line units is 240. Effectively, ATARI has a high-resolution mode of 352×240 .

It’s important to realize that there are physical limitations to this size display. Depending on your television’s adjustment, some of the displayed image may appear on the curved edge of the picture tube. This overlap is called overscan. While overscan is not important in normal television viewing, it is crucial if what your word processor is printing you can’t see.

Atari, in its Operating System (OS), used a more conservative screen size of 320 (160 clocks) horizontally by 192 scan lines vertically. This width screen is called “normal playfield” in the documentation. In this way Atari defeated normal overscan and assured us of seeing an entire image. There is a narrow playfield width as well, 256 pixels (128 clocks wide). These dimensions and timing are important since what is not used at display time is left over and available at interrupt time. (See Table 1 for timing.)

It is relatively simple to change between screen widths. Location \$22F controls playfield width. Called SDMCTL in the documentation, it is initialized to 34. Writing a 35 will change the screen dimension to wide, and writing 33 will reduce the screen to narrow. SDMCTL is the OS shadow for a hardware register in the ANTIC chip at \$D400, called DMACTL.

Since many of these hardware locations are write only, the OS keeps copies, called shadows, in RAM. Shadow registers update the associated hardware at vertical-blank-interrupt time. Remember to use the shadows to effect a permanent change to the entire frame. The exception occurs when using a display list interrupt. These interrupts can occur, under programmer control, on any scan line of every frame. To effect an immediate change at scan line interrupt, you must write directly to the hardware register.

To use the display list interrupt (DLI), a number of things must be accom-

plished. First, write the DLI service routine. The important thing to remember here is to save and restore any registers needed by the routine. Then find a free place in memory for this routine. (As you know, ATARI has reserved Page Six, decimal 1536-1791, just for users.) Next, update the vector at \$200 and \$201 to point to start of the routine. Now change the appropriate display list instruction to cause an interrupt (accomplished by turning on bit 7 of the instruction). Finally, enable DLIs by setting bit 7 of hardware register \$D40E, called NMIEN (Non-Maskable Interrupt Enable). See Listing 2 for a simple example.

Also remember to set the interrupt in the mode line prior to the location where you would have the changes occur. Then write to a location call WSYNC \$D40A. This will cause any changes to be delayed to the start of the next scan line and, therefore, allow a smoothly synchronized transition.

DLIs can be used for everything from putting many colors on the screen, to changing among a number of character sets, to moving Player/Missiles around. To get the most from your ATARI, experiment with this concept.

by James Capparell

Table 1: Timing

1.79 MHZ machine cycle
262 scan lines per frame
228 color clocks per scan line
60 frames per second refresh rate
 $1.79/60 = 29868$ machine cycles per frame
 $29868/262 = 114$ machine cycles per scan line
 $228/114 = 2$ color clocks per machine cycle

Vertical Blank Time

$262 \text{ scan lines} - 192 \text{ displayed scan line} = 70$
 $70 \times 114 \text{ cycles/line} = 7980 \text{ cycles available*}$

Horizontal Blank Time

Wide Playfield
 $228 \text{ clocks} - 192 \text{ clocks} = 36 \text{ clocks}$
 $36/2 = 18 \text{ machine cycles}$
Normal Playfield
 $228 \text{ clocks} - 160 \text{ machine cycles} = 68 \text{ clocks}$
 $68/2 = 34 \text{ machine cycles}$
Narrow Playfield
 $228 \text{ clocks} - 128 \text{ clocks} = 100 \text{ clocks}$
 $100/2 = 50 \text{ machine cycles}$

*All graphics are cycle-stealing direct Memory Access (DMA). Depending on graphics mode and memory refresh, this value will be less.

SYSTEMS GUIDE

```
10 REM *** PROG1 ***
20 REM MEMORY AND DISPLAY LIST VARIES
  WITH GRAPHICS MODE
30 REM DUMP AND DISASSEMBLE DISPLAY LI
  ST
40 REM
100 ? " INPUT GRAPHICS MODE ";;INPUT M
  ODE
110 LST=PEEK(560)+PEEK(561)*256:REM FI
  ND START OF DISPLAY LIST
120 MEMRY=PEEK(LST+4)+PEEK(LST+5)*256:
  REM FIND START OF DISPLAY MEM.
130 RAMTOP=PEEK(106)*256:REM NUMBER OF
  PAGES IN MEM DEFINED AT POWER ON
140 REM LIST
150 LPRINT " OS GRAPHICS MODE ";MODE
160 LPRINT " RAM AVAILABLE AT POWER ON
  ";RAMTOP
170 LPRINT " START OF DISPLAY LIST ";L
  ST
180 LPRINT " START OF DISPLAY MEMORY "
  ;MEMRY
190 REM DUMP DISPLAY LIST WITH DISASSE
  MBLY OF INSTRUCTIONS
195 LMS=64:INT=128:HSCR=16:VSCRL=32:JV
  B=65:JMP=1
200 FOR I=LST TO MEMRY-1
205 LPRINT I;" ";PEEK(I);
210 INST=PEEK(I):REM DISPLAY LIST VALU
  E
215 IF INST>-128 THEN GOSUB 1100:GOTO
  400
220 GOSUB 1140
400 NEXT I
410 STOP
1100 INST=INST-INT:REM GET RID OF INTE
  RRUPT BIT
1105 LPRINT " INSTRUCTION INTERRUPT EN
  ABLE "
1140 GOSUB 2000:REM FIND JUMPS AND BLA
  NKS
1150 IF INST=0 THEN RETURN
1160 GOSUB 1400:REM GO FIND LMS
```

```

1170 GOSUB 1500:REM GO FIND VSCROL
1180 GOSUB 1600:REM GO FIND HORIZONTAL
    SCROLL
1190 GOSUB 1700:REM TRANSLATE ANTIC MO
DE TO OS GRAPHICS MODE
1200 RETURN
1400 IF INST<66 THEN RETURN :REM NO LM
S
1405 LPRINT " LOAD MEM SCAN FROM ";PEE
K(I+1)+PEEK(I+2)*256
1410 INST=INST-LMS:REM GET RID OF LMS
BIT
1420 I=I+2:REM INCREMENT LOOP AROUND A
DDRESS BYTES
1430 RETURN
1500 IF INST<34 THEN RETURN :REM NO VS
CROL ENABLE
1510 INST=INST-VSCRL:REM GET RID OF VS
CROLL BIT
1520 LPRINT "VERTICAL SCROLL ENABLED"
1530 RETURN
1600 IF INST<18 THEN RETURN :REM NO HS
CROLL ENABLE
1610 INST=INST-HSCRL:REM GET RID OF HO
RIZONTAL SCROLL BIT
1620 LPRINT " HORIZONTAL SCROLL ENABL
ED "
1630 RETURN
1700 LPRINT " ANTIC DISPLAY MODE ";INS
T
1750 RETURN
2000 IF INST=0 OR INST=16 OR INST=32 O
R INST=48 OR INST=64 OR INST=80 OR INS
T=96 OR INST=112 THEN GOSUB 2100
2010 IF INST=1 THEN GOSUB 2200
2020 IF INST=65 THEN GOSUB 2300
2030 RETURN
2100 LPRINT " BLANK ";INT(INST/16)+1;"
    LINES "
2110 INST=0:RETURN
2120 REM
2200 LPRINT " JUMP INSTRUCTION TO ";PE
EK(I+1)+PEEK(I+2)*256

```

```

2210 I=I+2:REM INCREMENT AROUND ADDRES
S BYTES
2215 INST=INST-JMP:RETURN
2220 REM
2300 LPRINT " JUMP & WAIT FOR VERTICAL
BLANK TO ";PEEK(I+1)+PEEK(I+2)*256
2310 I=I+2:REM INCREMENT AROUND ADDRES
S BYTES
2315 INST=INST-JVB:RETURN

```

TYP0 TABLE

Variable checksum = 351876

Line	num	range	Code	Length
10	-	170	QI	468
180	-	1105	PT	372
1140	-	1430	SU	397
1500	-	2010	MI	393
2020	-	2315	IP	376

```

10 REM *** PROGRAM 2 ***
20 REM THIS WILL CREATE A DISPLAY LIST
WITH DLI ENABLED
30 REM THE SCREEN WIDTH IS NARROWED AT
DLI TIME AS WELL
40 REM
45 GRAPHICS 0:SETCOLOR 4,4,9:REM SET B
ORDER COLOR
50 DLST=PEEK(560)+PEEK(561)*256:REM FI
ND START OF DISPLAY LIST
60 POKE DLST+14,PEEK(DLST+14)+128:REM
TURN ON INTERRUPT BIT 7
70 FOR L=0 TO 29:REM POKE DLI SERVICE
ROUTINE INTO PAGE 6
80 READ INSTRUCT:POKE 1536+L,INSTRUCT
90 NEXT L
100 DATA 72,138,72,169,40,162,48,141,1
0,212,141,23,208

```

```

110 DATA 142,24,208,169,33,141,0,212,1
62,140,142,26,208,104,170,104,64
120 POKE 512,0:POKE 513,6:REM POINT TO
DLI INTERRUPT SERVICE ROUTINE
130 POKE 54286,192:REM ENABLE DMI
140 LIST
150 REM *** DLI SERVICE ROUTINE ***
152 REM PHA SAVE REGISTERS
154 REM TXA
156 REM PHA
158 REM LDA #$28 CHARACTER LUMINENCE
160 REM LDX #$30 BACKGROUND COLOR
162 REM STA $D40A WAIT FOR HORIZONTAL
SYNCH
164 REM STA $D017 PLAYFIELD 1
166 REM STX $D018 PLAYFIELD 2
168 REM LDA #21 NARROW PLAYFIELD
170 REM STA $D400 DMACTL ENABLE NAROW
WIDTH
172 REM LDX #$8C BORDER COLOR
174 REM STX $D01A COLBK
176 REM PLA RESTORE REGISTERS
178 REM TAX
180 REM PLA
182 REM RTI RETURN FROM INTERRUPT

```

TYPO TABLE

Variable checksum = 99022

Line	num	range	Code	Length
10	-	110	TN	531
120	-	166	OH	357
168	-	182	LQ	198

Interrupts

An important feature to understand is the vertical blank (Vblank) interrupt. I will give you a working definition of what an interrupt is, then discuss how Vblank fits into the overall interrupt structure, what is accomplished in this time period, and how programmers may access this interrupt for their own use. I will also provide a simple program to illustrate the use of Vblank vectors and how to insert code at VVBLKD.

Recall from my discussion of raster scan graphics, that the term vertical blank is given to that time period when the electron beam is turned off and returned to the upper-left corner of the video screen, ready to start tracing a new frame. The number of machine cycles available at Vblank is some fraction of 29868 machine cycles that are needed to trace one entire television frame. In the normal Graphics Mode O (text screen), approximately 7980 machine cycles are left over at Vblank to be shared by the Operating System Vblank interrupt service routine (ISR) and any programmer supplied code. The term interrupt applies to any signal, originating from hardware or software, which serves to suspend normal mainline program flow.

When an interrupting event occurs, the program counter (PC) and processor status registers are automatically saved on the system stack. The processor then executes special code referred to as an interrupt service routine (ISR). The address of the ISR is found in a memory location reserved for this purpose, called an interrupt vector. When the ISR is finished, the values of the PC and status registers are retrieved from the stack and processing of the suspended program is resumed as if nothing had intervened. This all happens at machine speed—in hundreds of microseconds.

The vertical blank interrupt is an essential part of the ATARI Operating System and appears as a non-maskable interrupt (NMI) to the system. The NMI is one of three possible interrupts that the ATARI can process. These three—chip reset, NMI, and IRQ—are analyzed further by interrupt service software. Whenever an NMI or an IRQ signal occurs, the appropriate service

"Interrupts," by James Caparell, is reprinted with permission from MICRO Magazine, Issue No. 43, P.O. Box 6502, Amherst, NH 03031.

routine is executed. These service routines interrogate a status register to isolate the interrupting source. See Table 1 for a breakdown of vectors and contents for each type of interrupt.

It's apparent from Table 1 that all NMI interrupts are vectored through location \$FFFA to the NMI interrupt service routine starting at address \$E7B4. Since there are three possible causes of an NMI, the ISR must determine the source of the interrupt by interrogating an NMI status register at address \$D40F. This location, called NMIST in the documentation, has bit 7 set when a DLI occurs, bit 6 set when [SYSTEM RESET] has been pressed. If neither a DLI nor a [SYSTEM RESET] caused the NMI, then a Vblank interrupt is assumed by the ISR and the processor jumps to the address contained in the vector at \$0222. There are actually two vectors used by Vblank through which a programmer may introduce additional or replacement code. One vector, referred to as vertical blank immediate vector VVBLKI, is at address \$0222. This vector normally contains the address \$E7D1, the start of the

Table 1.

INTERRUPT	VECTOR	ISR LOCATION
CHIP RESET	FFFC	E477
NMI	FFFA	E784
	Display list	Jump through 0200
	Vertical Blank	0222 and 0224
	S/Reset key	E474
IRQ	FFFE	E6F3
	Serial bus output ready jump through	020C
	Serial bus output complete	020A
	Serial bus input ready	020E
	*Serial bus proceed line	0202
	*Serial bus interrupt line	0204
	*Pokey timer 1	0210
	*Pokey timer 2	0212
	*Pokey timer 4(Bug in O.S. timer 4)	0214
	Keyboard key scan	0208
	Break key	???
	*6502 break instruction	0206

* These vectors are unused by the O.S. and are initialized to point to an RTI instruction.

system Stage 1 Vblank ISR. Should it be necessary to either replace system functions or simply perform operations prior to the system code, then you would use this vector. The other vector location, called vertical blank deferred VVBLKD, is at address \$0224. This vector normally contains the address \$E93E, which is the start of code for the system return from interrupt. The contents of \$0224 would be changed to point to new code when your operation was needed after system housekeeping was accomplished.

The Vblank process is actually divided into two stages. Whenever a Vblank NMI occurs, the following events always happen:

1. Processor registers A, X, and Y are pushed on stack.
2. Interrupt request is cleared by writing zero to \$D40F.
3. Jump through VVBLKI normally pointing to Stage 1 Vblank.

When Stage 1 processing is executed, it increments the three-byte counter called RTCLOCK at addresses \$12, \$13, and \$14. Location at \$12 is the most significant byte. This counter wraps to zero after approximately 77 hours and then continues counting. The attract mode is also processed at Stage 1; that is the process which causes the colors on your screen to start shifting if no key has been pressed on the keyboard in the previous nine minutes.

Additionally, system timer one at locations \$218 and \$219 is decremented if it is non-zero. When the counter goes to zero, an indirect JSR is performed *via* a vector at addresses \$226 and \$227. Note that an indirect JSR is performed by copying the address from the vector to the stack and executing an RTS instruction.

At this point a test is made to determine if a time-critical section of code was interrupted. If either the I bit in the processor status register or the critical flag at address \$42 are set, then the interrupted code is assumed to be time-critical. When this occurs, the registers are restored and an RTI instruction is executed.

The critical flag can be set by a Serial I/O in progress. If no time constraints are present, then Stage 2 processing is begun. It is in this section of code that IRQ interrupts are enabled, keyboard auto-repeat logic is processed, keyboard debounce is performed, and system timers, 2, 3, 4, and 5 are processed. In addition, the color data for playfield and Player/Missiles are updated. This color data and other RAM locations, called shadow registers, are

INTERRUPTS

copied into their associated hardware locations. Stage 2 also reads the game controller data from jacks 1, 2, 3, and 4 into RAM memory.

To insert code either at VVBLKI or VVBLKD, the address where the new code resides must be placed into the appropriate vector. A system routine insures that both bytes of the vector will be updated while Vblank is enabled. A vertical blank can be processed during a call to this routine. The routine is called SETVBV in the documentation and the calling sequence is:

Register A (update indicator)

- = 1-5 then update timers 1-5
- = 6 for immediate Vblank vector VVBLKI
- = 7 for deferred Vblank vector VVBLKD

Register X most significant byte of vector address (hi-byte)

Register Y = least-significant byte of vector address (lo-byte)

JSR SETVBV Jump to subroutine

The A, X, and Y registers may be altered.

The display list interrupt will always be enabled on return.

A knowledge of processing interrupts and inserting code at interrupt vectors is essential to get the most from the ATARI. With this example you should have enough information to experiment with the Vblank vectors. Interrupt-driven sound control, page flipping, animation techniques, greater color control, and many other procedures are possible.

James Capparell

```
00 ; ** PROGRAM EXAMPLE 1 **
20 ; PROGRAM SETS UP A VVBLKD ISR
30 ;
40 ; SET UP NEW VECTOR WITH A BASIC US
R CALL A-USR(1536)
50 ; NEED TO DO THIS WHENEVER SYSTEM I
S RESET
60 *= $600          PUT IN PAGE 6 DECIMAL
1536
70 PLA             NULL VALUE FROM BASIC
80 LDA #7          INDICATOR FOR VVBLKD
90 LDX #06         HIGH BYTE FOR VECTOR
```

SYSTEMS GUIDE

```
ADDR
0100 LDY #$40      LOW BYTE FOR VECTOR A
DDR
0110 JSR $E45C      SET UP DEFER
0120 RTS           RETURN TO BASIC
0130 ; ** ** ** **
0140 ; ROUTINE AT DECIMAL 1600 IS DESI
    ; GINED TO WASTE TIME.
0150 ; PUT A NUMBER FROM 1 - 5 IN DECI
    ; MAL 1568.
0160 ; USE POKE 1568,N
0170 ; THIS IS THE ISR WHICH SIMPLY WA
    ; STES TIME.
0180 *- $640
0190 LDX 0          INIT COUNTERS
0200 LDY 0
0210 LOOP1 INX      INCR COUNT
0220 CPX $620       DELAY VALUE
0230 BEQ LOOP2
0240 CLC            FORCE BRANCH
0250 BCC LOOP1
0260 LOOP2 INY
0270 CPY $620       DELAY VALUE
0280 BEQ EXIT       DONE ?
0290 CLC            NO-FORCE BRANCH
0300 BCC LOOP1
0310 EXIT JMP $E93E TAKE NORMAL VBLANK
EXIT
```

Handling Media

Diskettes and cassettes with computer data on them are easily damaged, especially diskettes. Never touch the surface of the magnetic medium with your fingers. Oil from your skin will interfere with the readability of data underneath.

Protect tapes and disks from magnetic sources such as televisions, telephones and magnetized tools. Prolonged exposure to sunlight and heat can be damaging too, so store media safely away when not in use.

Dust and ash from cigarettes can accumulate on exposed disks, so always keep disks in their envelopes when not in use. Vertical storage in protective boxes helps. Liquid spilled on disks or tapes is almost always fatal. It is best not to eat or drink in your computing area.

Disks must be perfectly flat and free to move in their protective sheaths. Never bend or fold a diskette, nor write on it with a pen or pencil that requires pressure. Do not use paperclips on disks as these may crimp the sheath. Accidental creasing or crushing of disks in briefcases is a common tragedy.

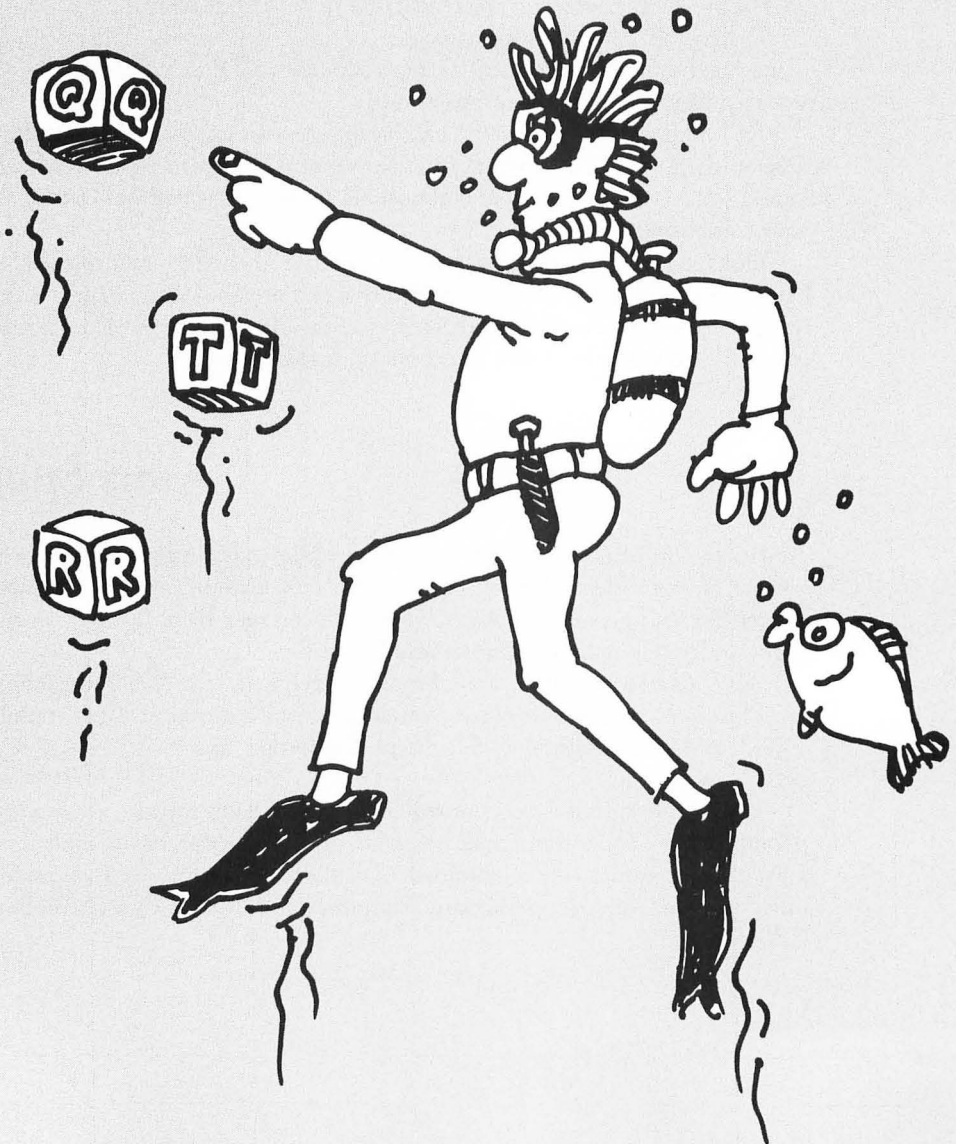
Atari Support

Atari is the only microcomputer company with an extensive program of help for the owners of its products. There are more the 1700 authorized service centers in the United States, plus others abroad, where you can seek help. Just look in the yellow pages under "Computers - Service and Repair."

Also, Atari maintains a staff of trained Agents and Product Specialists available at toll-free phone lines to answer questions from customers. These numbers are: 800-538-8543 (continental U.S., except California) and 800-672-1404 (California only.)

Users groups, that is, local or regional clubs of Atari owners, are located in many populated areas and are especially helpful to beginners. Atari has an office from which a list of these groups can be obtained. Call 408-743-4196 for user group information only, or write User Group Support, Atari, Inc., 1399 Moffet Park Drive, Sunnyvale, CA 94086.

Assembly Language



Move-It

Move-It provides the ATARI programmer with the ability to move one byte of data into a range of memory locations. This assembly language routine is position independent. It is loaded into a string from data statements 250 and 260. The routine is useful for clearing sections of screen memory, Player/Missile memory, erasing a player, and clearing memory used in page flipping.

The parameters which control this routine are passed in a USR statement. The start location and byte total to be moved are passed. There are no limitations on the total bytes which can be moved.

Interesting sounds are also generated by the BASIC routine. The soundless version moves bytes at the rate of almost a quarter of a million per second (256×960).

by Jerry White

```
10 ; This is a position independent su  
broutine  
20 ; found in DATA statements line num  
bered 250 and 260  
30 ; Calling Sequence from BASIC is:  
40 ; A = USR(ADR($STR),Start Addr,Count)  
50 ;  
60          *- $600 ;can go anywhere  
70          PLA      ;ignore argument c  
ount  
80          PLA      ;save lo-byte of d  
est addr  
90          STA $CC  
0100        PLA ;save hi-byte of dest  
addr  
0110        STA $CB  
0120        PLA      ;save total to be
```

ASSEMBLY LANGUAGE

```
moved
0130      STA  $CE ;*
0140  PLA      ;save total to be moved
0150      STA  $CD ; *
0160      LDX  $CE ; count of bytes
      to move
0170      LDY  #0  ; init index
0180      LDA  #0  ; init character
      to be moved
0190  MOV      STA  ($CB),Y ; move data
0200      DEY      ; decrement index
0210      BNE  MOV ; go move next character
0220      INC  $CC ; incr dest address
      lo-byte
0230      DEX      ; decr lo-byte count to move
0240      BMI  EXIT
0250      BNE  MOV ; go move next character
0260      LDY  $CD ; hi-byte of count to move
0270      BNE  MOV ; go move next character
0280  EXIT      DEC  $CC ; decr lo-byte dest addr
0290      LDY  #0
0300      STA  ($CB),Y
0310      RTS      ; return to BASIC
0320      .END
```

```
5 GOTO 35:REM MOVEIT UTILITY/DEMO BY JERRY WHITE 3/31/82 ANTIC MAGAZINE
15 REM THATS INCREDIBLE SUBROUTINE
20 FOR M=0 TO 255:Z$(19,19)=CHR$(M):Z=USR(ADR(Z$),SM,960):NEXT M:RETURN
```



```

30 REM SM SCREEN MEMORY C SPEAKER
35 GRAPHICS 0:POKE 82,0:DIM Z$(42):POKE 752,1:S-PEEK(560)+PEEK(561)*256+4:S M-PEEK(S)+PEEK(S+1)*256:C-53279
45 REM Z$ ASSEMBLER ROUTINE STRING (POSITION 19 CHARACTER TO MOVE)
50 REM Z USRCADR(Z$).START ADR.HOW MANY)
60 FOR X=1 TO 42:READ IT:Z$(X,X)-CHR$(IT):NEXT X:POKE 82,0:" "":POKE 83,39:SOUND 0,0,0,0
65 Z$(19,19)-CHR$(128):Z-USR(ADR(Z$),SM,960):POKE 710,113:POSITION 39,0:"CHR$(160):
75 ? " This DEMO demonstrates an assembler ":" MOVE routine called from BASIC.
80 ? " Possible uses would be to move ":" blanks or special characters to an
85 ? " area of screen memory, or to clear ":" RAM used for player missiles or
90 ? " page flipping etc.

95 ? " SELECT OPTION NUMBER:
   ":" (1) FAST WITH SOUND
   ":"GOSUB 230
100 ? " (2) VERY FAST WITH SOUND
   ":"GOSUB 230:" (3) THAT'S INCREDIBLE (SILENT) ":"GOSUB 230
110 POKE 764,255:CLOSE #1:OPEN #1,4,0,"K":GET #1,K:CLOSE #1
115 REM ACCEPT ONLY A 1 2 OR 3
120 IF K<49 OR K>51 THEN FOR ME=15 TO 0 STEP -.5:SOUND 0,102,12,ME:NEXT ME:GOTO 110
130 REM I LOVE SOUND ROUTINES
135 FOR J=1 TO 7:POKE 710,J*16:FOR X=2 TO 0 STEP -1:FOR ME=14 TO 0 STEP -2:SOUND 0,X+J,2,ME:NEXT ME:NEXT J
145 REM EXECUTE SELECTED MOVEIT ROUTINE
E

```


ASSEMBLY LANGUAGE

```
150 IF K-51 THEN GOSUB 20:GOTO 175
155 IF K-50 THEN GOSUB 190:GOTO 175
160 GOSUB 210:GOTO 175
170 REM DING...ALL DONE...START OVER
175 FOR ME-15 TO 0 STEP -0.2:SOUND 0,0
,2,ME:NEXT ME:RUN
185 REM VERY FAST SUBROUTINE WITH SOUND
D
190 FOR M=0 TO 255:Z$(19,19)=CHR$(M):P
OKE 53761,168:POKE 53763,168:POKE 5376
0,255-M:POKE 53768,13:POKE 712,M
195 POKE 53762,M:POKE 53762,M/8:POKE 5
3768,2:Z=USR(ADR(Z$),SM,960):POKE 5376
1,0:POKE 53763,0:NEXT M:RETURN
205 REM FAST SUBROUTINE WITH SOUND
210 FOR M=255 TO 0 STEP -1:Z$(19,19)=C
HR$(M):POKE 53760,M:FOR V=175 TO 160 S
TEP -1:POKE 53761,V
215 POKE 53768,V-160:NEXT V:POKE 712,M
:Z=USR(ADR(Z$),SM,960):NEXT M:RETURN
225 REM BLINK 6 BUZZ SUBROUTING
230 FOR JW=0 TO 8:POKE 755,1:POKE C,0:
POKE C,8:NEXT JW:FOR JW=0 TO 8:POKE 75
5,2:NEXT JW:RETURN
240 REM DATA TO CREATE Z$ ASSEMBLER SU
BROUTINE
250 DATA 104,104,133,204,104,133,203,1
04,133,206,104,133,205,166,206,160,0,1
69,0,145,203,136
260 DATA 208,251,230,204,202,48,6,208,
244,164,205,208,240,198,204,160,0,145,
203,96
```

TYPO TABLE

Variable checksum = 139086

Line	num	range	Code	Length
5	-	60	CB	588
65	-	95	AQ	537
100	-	145	JJ	522
150	-	205	GJ	516
210	-	260	UT	534

done

Bubble Sort

This is a handy Sort Utility intended to be called from BASIC and allows you to sort almost anything that can fit in your computer's memory. The flexibility of the sort should cover many applications. Records may be any size up to 256 bytes. The sort fields may be any size up to the length of the record. You can sort on as many different fields as you need, and each field can be independently sorted in ascending or descending sequence.

The sorting technique is the traditional Bubble Sort which works by looking through a file of records in memory, and comparing the sort field of each record to the one following it. If any two adjacent records are not in sequence, the sort will exchange the positions of those two records. The sort continues to scan the file until there are no more records to exchange. In this way, records with the higher sort fields get pushed towards the end of the file, and records with the lower sort fields get pushed towards the beginning of the file. All of this takes place in memory so that it appears that the records bubble into place.

The sort only requires 182 bytes and the machine language is relocatable, therefore you can load and execute this sort anywhere in memory. Although you can put the sort in any program you like, your file size is going to be limited by available memory. For large files, it is best to write a small BASIC program that contains only this sort, a string large enough to hold your file, and whatever BASIC statements it takes to load a file, call the sort and write out the new sequenced file.

Although the sort works very fast, its speed can be improved by about 30 percent by turning ANTIC off. Just before calling the sort, save the value at PEEK(559) then POKE in a zero. All this does is shut down the screen display, but in so doing, it makes about 30 percent more CPU cycles available to the sort. After the sort, POKE the saved value back into 559 and the screen display will turn back on.

All sort parameters are passed to the sort in the BASIC USR call in the following sequence: 1. Address of the string containing the file; 2. Length of the records; 3. Number of records to be sorted. The next parameters specify

ASSEMBLY LANGUAGE

the fields to be sorted by: 4.1. Position of the first byte of the field; 4.2. Length of the field; 4.3. '0' for ascending sequence, or '1' for descending sequence. Sort fields are specified in Major to Minor order. That is, if you want to sort on state, and zip code within state, then state is the Major order and should be the first set of sort field parameters. The only limitation on the number of sort fields is the number of parameters that fit in the BASIC statement calling the sort.

The program in Listing 2 loads the machine language code for the sort in Lines 1 to 9. The rest of the program demonstrates one of many techniques that can be used to read an unsequenced file, sort and rewrite a sequenced file. Type and run the program and at the prompt, enter the first and last names of about nine friends. The first names will be sorted ascending, the last names will be sorted descending and then displayed on the screen.

by Adrian Dery

```
0 REM ***** SORT UTILITY DEMONSTRAT
ION *****
1 DATA 216,104,56,233,3,133,217,104,13
3,204,104,133,203,104,133,215,104,133,
214,104,133,210,104,133,209,162,0
2 DATA 104,104,157,0,1,232,228,217,208
,246,56,165,209,233,2,133,209,165,210,
233,0,133,210,48,108,165,209,133,211
3 DATA 165,210,133,212,165,204,133,206
,133,208,165,203,133,205,24,101,214,13
3,207,165,208,101,215,133,208,160
4 DATA 0,185,0,1,190,2,1,134,218,190,1
,1,200,200,200,132,216,168,136,177,205
,209,207,240,12,165,218,208,4,144
5 DATA 16,176,46,144,44,176,10,200,202
,208,234,164,216,196,217,208,210,198,2
11,169,255,197,211,208,6,166,212,240
6 DATA 11,198,212,165,208,133,206,165,
207,24,144,172,165,213,240,4,134,213,2
08,148,96,134,213,160,0,177,205,170
7 DATA 177,207,145,205,138,145,207,200
,196,214,208,241,240,203
8 DIM SORT$(182):FOR I=1 TO 182
9 READ A: SORT$(I,I)=CHR$(A):NEXT I
```

```

100 REM -----
105 REM INPUT A FILE TO BE SORTED
110 DIM FILE$(270),NAME$(15)
115 FILE$=" ":FILE$(270)=FILE$
120 FILE$(2)=FILE$
125 GRAPHICS 0
130 ? "ENTER THE NAMES OF 9 FRIENDS"
135 FOR I=0 TO 8:LE=I*30+1
140 ? I+1;" FIRST NAME "":INPUT NAME$
145 FILE$(LE,LE+14)=NAME$
150 ? I+1;" LAST NAME "":INPUT NAME$
155 FILE$(LE+15,LE+29)=NAME$
160 NEXT I
200 REM -----
205 REM PRINT UNSORTED FILE
210 GRAPHICS 0:? "UNSORTED NAME LIST"
215 FOR I=0 TO 8:LE=I*30+1
220 ? FILE$(LE,LE+29)
225 NEXT I
300 REM -----
305 REM SORT AND PRINT THE FILE
310 ANTIC=PEEK(559):POKE 559,0
315 X=USR(ADR(SORT$),ADR(FILE$),30,9,1
6,15,1,1,15,0)
320 POKE 559,ANTIC
325 ? :? "SORTED NAME LIST"
330 FOR I=0 TO 8:LE=I*30+1
335 ? FILE$(LE,LE+29)
340 NEXT I
345 END

```

TYPO TABLE

Variable checksum = 170377

Line	num	range	Code	Length
0	-	5	OH	596
6	-	135	GC	463
140	-	300	UQ	331
305	-	345	IC	269

ASSEMBLY LANGUAGE

```
0100 :UTILITY SORT - CALLED FROM BASIC
0105 ;
0110 :ENTRY PARAMETERS:
0115 ;
0120 ; 1. FILE ADDRESS
0125 ; 2. RECORD LENGTH <=256 BYTES
0130 ; 3. NUMBER OF RECORDS TO SORT
0135 ; 4. ANY NUMBER OF FIELDS TO SORT IN
0140 ; MAJOR TO MINOR ORDER
0145 ; 4.1 FIELD POSITION
0150 ; 4.2 FIELD LENGTH
0155 ; 4.3 0=ASCENDING 1=DESCENDING
0160 ;
0165 ORG $0600
0170 FILE = 203 ;FILE START ADDRESS
0175 PNTR1 = 205 ;POINTERS TO TWO
0180 PNTR2 = 207 ;ADJACENT RECORDS.
0185 RECNBR = 209 ;NUMBER OF RECORDS
0190 SCOUNT = 211 ;RECORD COUNTER
0195 BUBLE = 213 ;OUT OF SEQUENCE
0200 RECSIZ = 214 ;SIZE OF RECORD
0205 FLDNDX = 216 ;SORT FIELD COUNTER
0210 FLDCNT = 217 ;NUMBER OF SORT FIELDS
0215 SORTAD = 218 ;ASCENDING/DESCENDING
0220 STACK = 256 ;SAVE SORT FIELDS HERE
0225 ;
0230 ;DETERMINE HOW MANY FIELDS TO SORT
0235 CLD
0240 PLA ;ALL BUT TH
```

```

E FIRST
0245      SEC                      ;THREE PARA
METERS
0250      SBC #3                   ;ARE FIELDS
TO
0255      STA FLDCNT               ;SORT
0260      ;
0265      ;PICK UP SORT PARAMETERS
0270      PLA                      ;FILE START
0275      STA FILE+1               ;ADDRESS
0280      PLA                      ;
0285      STA FILE                 ;
0290      PLA                      ;RECORD LEN
GTH
0295      STA RECSIZ+1             ;
0300      PLA                      ;
0305      STA RECSIZ               ;
0310      PLA                      ;NUMBER OF
RECORDS
0315      STA RECNR+1              ;
0320      PLA                      ;
0325      STA RECNR               ;
0330      ;
0335      ;PICK UP FIELDS TO SORT
0340      LDX #0
0345      PICKFIELDS
0350      PLA                      ;GET ALL TH
E SORT
0355      PLA                      ;FIELD PARA
METERS FOR
0360      STA STACK,X              ;POSITION,
LENGTH
0365      INX                      ;AND DIRECT
ION.
0370      CPX FLDCNT               ;ANY MORE
0375      BNE PICKFIELDS           ;GO GET THE
M
0380      ;
0385      ;SET UP NUMBER OF RECORDS TO SORT
0390      SEC
0395      LDA RECNR                ;MUST BE AT
LEAST
0400      SBC #2                   ;TWO RECORD

```

ASSEMBLY LANGUAGE

```

S TO
0405      STA RECNR      ; SORT
0410      LDA RECNR+1    ;
0415      SBC #0         ;
0420      STA RECNR+1    ;
0425      BMI ENDSORT    ; ELSE GET 0
UT
0430 ;
0435 ; MAIN LINE SORT LOOP
0440 ;
0445 SORT   LDA RECNR      ; RESET NUMB
ER OF
0450      STA SCOUNT      ; RECORDS TO
      SORT
0455      LDA RECNR+1     ;
0460      STA SCOUNT+1    ;
0465      LDA FILE+1      ; SET UP POI
NTERS
0470      STA PNTR1+1     ; FOR THE FI
RST
0475      STA PNTR2+1     ; AND
0480      LDA FILE        ; SECOND REC
ORDS.
0485 BUMP RECORD
0490      STA PNTR1       ; PUT PNTR2
0495      CLC             ; AHEAD
0500      ADC RECSIZ       ; OF
0505      STA PNTR2       ; PNTR1
0510      LDA PNTR2+1     ; BY
0515      ADC RECSIZ+1    ; ONE
0520      STA PNTR2+1     ; RECORD.
0525 ;
0530 ; SEQUENCE CHECK RECORDS
0535 ;
0540      LDY #0          ; RESET STAC
K INDEX
0545 NEXTFIELD
0550      LDA STACK,Y      ; FIELD POSI
TION.
0555      LDX STACK+2,Y    ; SORT DIREC
TION
0560      STX SORTAD       ; SAVE IT.
0565      LDX STACK+1,Y   ; FIELD LENG

```


BUBBLE SORT

```

TH.
0570      INY      ; BUMP
0575      INY      ; STACK
0580      INY      ; INDEX
0585      STY FLDNDX ; AND SAVE I
T.
0590      TAY      ; FIELD POSI
TION TO Y
0595      DEY      ; MAKE RELAT
IVE TO ZERO
0600 SEQCHECK
0605      LDA (PNTR1),Y ; COMPARE AD
JACENT
0610      CMP (PNTR2),Y ; RECORDS
0615      BEQ SEQNDX ; = KEEP ON
LOOKING
0620      LDA SORTAD ; GET SORT D
IRECTION
0625      BNE DSNDG ; GO TO DESC
ENDING
0630 ;
0635 ; SORT IN ASCENDING SEQUENCE
0640 ;
0645      BCC BUMPINDEX ; < BUMP NEX
T RECORD
0650      BCS SWAP ; > SWAP POS
ITIONS
0655 ;
0660 ; SORT IN DESCENDING SEQUENCE
0665 ;
0670 DSNDG BCC SWAP ; < SWAP POS
ITIONS
0675      BCS BUMPINDEX ; > BUMP NEX
T RECORD
0680 ;
0685 SEQNDX INY ; CHECK THE
LENGTH OF
0690      DEX      ; THE SORT F
IELD AND
0695      BNE SEQCHECK ; KEEP SEQUE
NCE CHECKING.
0700      LDY FLDNDX ; ANY MORE F
IELDS

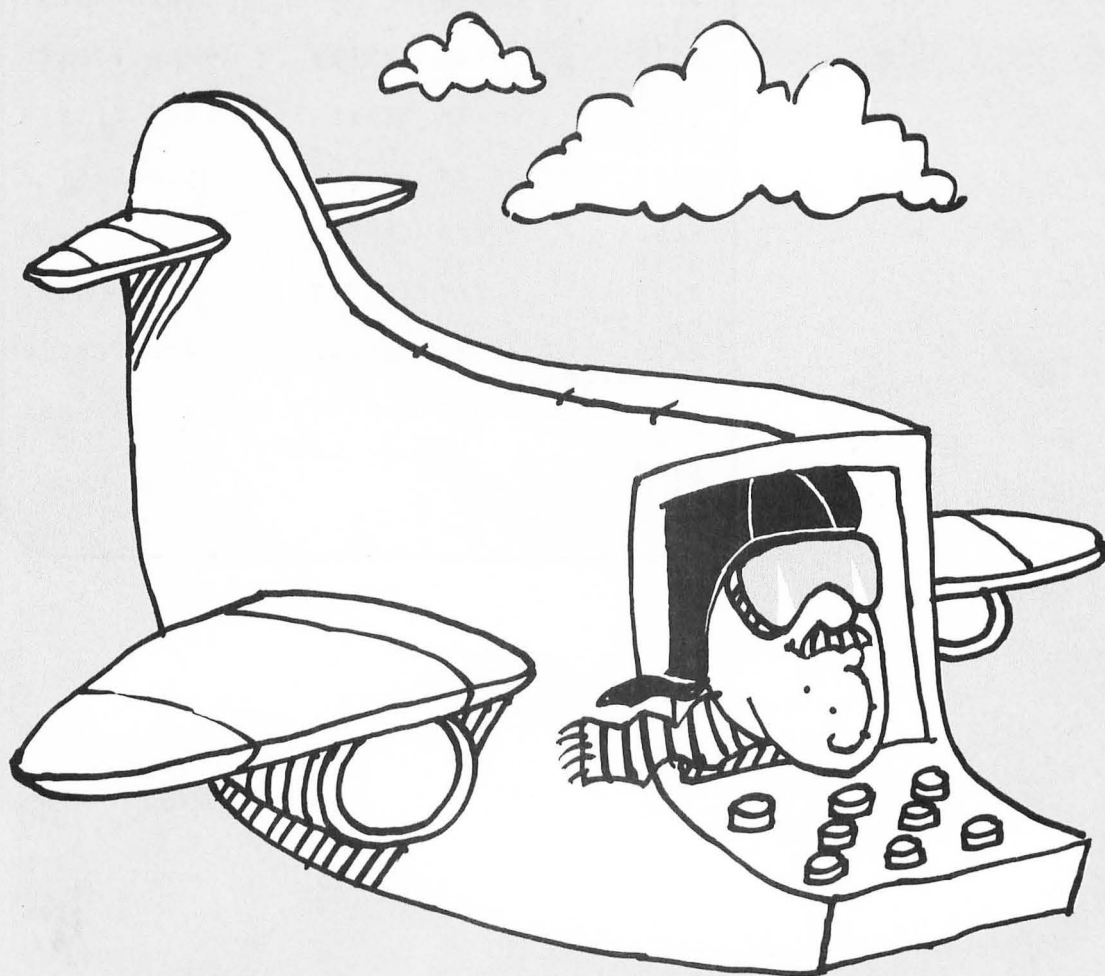
```

ASSEMBLY LANGUAGE

```
0705          CPY FLDCNT      ; TO SORT
0710          BNE NEXTFIELD    ; YES, GO TO
IT
0715          ;
0720          ; INDEX THROUGH THE SORT FILE
0725          ;
0730  BUMPINDEX
0735          DEC SCOUNT      ; COUNT DOWN
RECORDS
0740          LDA #255         ; AND CHECK
FOR
0745          CMP SCOUNT      ; END OF FIL
E.
0750          BNE NOTEOF      ;
0755          LDX SCOUNT+1     ;
0760          BEQ CKSWAP      ;
0765          DEC SCOUNT+1     ;
0770  NOTEOF  LDA PNTR2+1     ; BUMP PNTR2
AND
0775          STA PNTR1+1     ; PNTR1 TO T
HE
0780          LDA PNTR2       ; NEXT RECOR
DS.
0785          CLC
0790          BCC BUMPRECORD
0795          ;
0800          ; AT END OF FILE SEE IF A SWAP WAS
MADE
0805          ;
0810  CKSWAP  LDA BUBLE       ; IF NO RECO
RDS SWAPPED
0815          BEQ ENDSORT     ; THEN IS EN
D OF SORT,
0820          STX BUBLE       ; ELSE SEQUE
NCE CHECK
0825          BNE SORT        ; THE FILE A
GAIN.
0830  ENDSORT
0835          RTS             ; BACK TO BA
SIC
0840          ;
0845          ; SWAP RECORDS IF OUT OF SEQUENCE
0850          ;
```

```
0855 SWAP      STX BUBLE      ; STILL OUT  
OF SEQUENCE  
0860          LDY #0  
0865 SWAPLOP  
0870          LDA (PNTR1),Y    ; THIS ROUTI  
NE  
0875          TAX              ; EXCHANGES  
THE  
0880          LDA (PNTR2),Y    ; POSITIONS  
OF TWO  
0885          STA (PNTR1),Y    ; OUT OF SEQ  
UENCE  
0890          TXA              ; ADJACENT R  
CORDS  
0895          STA (PNTR2),Y    ;  
0900          INY  
0905          CPY RECSIZ      ; KEEP LOOPI  
NG FOR  
0910          BNE SWAPLOP      ; THE LENGTH  
OF RECORD.  
0915          BEQ BUMPINDEX    ; GO GET NEX  
T RECORD  
0920          .END
```

Pilot Your Atari



done
rest of Book done

Pilot Your Atari

PILOT is not just another computer language designed to meet some of the needs of new programmers, educators, and children. PILOT grew out of work by John Starkweather at the University of California at San Francisco back in 1972. He wanted a language that would make it easy to write tutorial programs for students, programs capable of recognizing responses other than the typical "1, 2, 3" choices prevalent in many current teaching programs. With PILOT, it is as easy to ask, "Who was the first president of the United States?" and record and score answers such as "President Washington," "I believe it was G. Washington," "George Washington," "GEORGE WASHINGTON," "Washington." PILOT needs only three statements to accomplish this type of user interaction.

Dean Brown at Stanford Research Institute proved that teachers could understand PILOT, and students loved it. Since PILOT is word-oriented, as contrasted to BASIC's number orientation, it naturally fits the "riddle" and "tell-a-story" type of program which youngsters like. At the same time, Seymour Papert at MIT developed a new way to conceptualize and teach geometry and shapes. This development was called "turtle graphics" and proved ideal for use on home computers. Atari wisely included a turtle graphics command language with the PILOT module.

The old "Cartesian coordinate" system required commands like this:

Start at position $X=20$ and $Y=10$. Draw a line to $X=40$ and $Y=10$; draw a line to $X=40$ and $Y=30$; draw a line to $X=20$ and $Y=30$; finally, draw a line to $X=20$ and $Y=10$.

Can you guess what figure this is? How big is it? Using turtle graphics the same picture can be drawn like this:

Ken Harms is a resident of the San Francisco Bay Area, and is Vice President of Administration for the California division of the American Cancer Society. He is especially interested in PILOT and Logo, and in computing as a tool to enhance the education of his two daughters. He is one of the earliest and most dedicated of Atari PILOT programmers whose articles in ANTIC regularly expand the usefulness of that language.

PILOT YOUR ATARI

Do this 4 times: draw a line 20 spaces long, turn Right 90 degrees.

The box shape is more apparent and the commands are more readily understood. A small collection of 14 and 15 commands represent the core of PILOT. All are only one or two characters long and easily remembered—a “J” is the “jump to” command. Anyone who is not a good typist will appreciate the wisdom of short commands. Short, easy to remember commands and turtle graphics combined with Atari’s wonderful screen editor will make almost anyone’s introduction to computing more pleasurable and rewarding. Finally, PILOT programs become naturally organized around modules. This encourages a well-structured programming style.

PILOT is available in two packages; one is just the language cartridge and users’ guide (about \$90), the other is a well-documented, comprehensive package that I recommend (about \$130). This package includes:

PILOT CARTRIDGE—(love those cartridges; little fingers can’t destroy them).

STUDENT PILOT—a cleverly illustrated learner’s manual for the new programmer.

PILOT PRIMER—an instruction manual for the experienced programmer.

DEMONSTRATION TAPES—two cassettes showing language, color, graphics, and sound.

POCKET CHART—presents all commands in an easy-to-use format.

I like Atari’s version of PILOT. There are still a few rough spots: not all syntax errors are caught, the manuals do not include indices, several commands are not explained in the manual, and a few typographical errors remain to confuse you. In spite of these few “start-up” problems, Atari PILOT meets its “primary design goals”: it is “consistent and easy to learn . . . it allows reasonable access to the Atari system capabilities, but not at the user’s expense.”

We intend to help you get the most from PILOT. Watch for programming tips, warnings, and more help.

by Ken Harms

done

Large Text

This series of articles will show you how to do what Atari left out of the PILOT manuals—fancy tricks such as large letters and changing colors, useful features like breaking strings into words, and using the mysterious commands in the demonstration programs.

When you run your PILOT program, three sets of instructions work together to give you the result you need. The Operating System in the 400/800 provides the instructions for reading the keyboard, and for writing characters to the TV screen and I/O devices, such as disk drives and printers. Additionally, the PILOT cartridge contains the translation system which actually interprets your PILOT program for the Atari hardware. These two systems working together allow the ATARI to perform the instructions you provide with the third type of instructions, the PILOT application program.

PILOT programs operate on data stored in the computer's memory or RAM (random access memory). PILOT stores each variable, constant, or instruction as a value in a unique location or address. These are like P.O. boxes. You can put messages into them and read data from them. Some addresses are used by the Operating System to hold information such as the color used on the screen and what size text characters to print, large or small. PILOT lets you change the contents of these addresses to give greater graphics control.

The Operating System supports fourteen different ways to display data on the screen. Those of you familiar with BASIC know eight of these modes. PILOT normally uses only two modes, Graphics 0, and Graphics 7; the first is a text mode, the second is a graphics mode. But you can turn on at least two of the extra modes to display large letters as eyecatching program titles.

To enable large text, we need to change values in two special addresses, 1373 and 1374, by using a special form of the Compute command:

C:@B1373=16

C:@B1374=1

This command might read as: "Compute the 'byte' at address 1373 equals 16". "Byte," in this context, means a value in memory. The first command puts a 16 in address 1373 to tell the ATARI that you want a graphic screen with

PILOT YOUR ATARI

regular letters at the bottom. The value 1 at address 1374 tells the ATARI that you want it to print medium-large letters. These Mode 1 letters are so large that only 20 fit on a line. Listing 1, lines 20 and 30, demonstrates these commands.

The next command you'll need is WRITE. It tells the ATARI to write data to a specific "device." These devices are identified by letters such as "D" for disk, "P" for printer, "C" for cassette and "S" for screen. Line 40 tells the ATARI to write anything you want. So, with those three simple commands, you have a dramatic opening for a program.

Change the contents of location 1374 to determine the size and number of characters per line.

1374=0	regular letters, 40 per line
1374=1	20 rows of medium letters, 20 characters per line
1374=2	10 rows of large letters 20 characters per line

The *TEST 2 module demonstrates Mode 2 large letters. In both modes, try using upper, lower and inverse characters. You'll find that each prints in a different color for interesting effects.

Address 1373 is the "sub-mode" address.

1373 =0	a full screen (no "text window")
1373 =16	split screen (text "text window")
1373Z=32	full screen opens without erasing prior data

Listing 2 uses the 32 sub-mode to erase the text window. If you're in sub-mode 0 or 32, any text (even the READY at the end of a program) clears the screen; use a PA: command to keep the screen up. To change any mode or sub-mode, you must CLOSE:S between modes and issue both 1373 and 1374 commands in the next mode. After entering a new mode, always issue a WRITE command before a type command (T:).

Next time, we'll look at changing colors and breaking strings into letters or words.

by Ken W. Harms

```
10 *TEST1 [MEDIUM LETTERS MODE 1
20 C:@B1373-16 [SPLIT SCREEN
30 C:@B1374-1 [SET MODE 1
40 WRITE:S, MODE 1 LETTERS
50 PA:240 [PAUSE TO WATCH SCREEN
```

```
60 CLOSE:S [REQUIRED TO CHANGE MODES
70 J:*TEST2
80 *TEST2 [LARGE LETTERS MODE 2
90 C:@B1373-16 [SPLIT SCREEN
100 C:@B1374-2 [SET MODE 2
110 WRITE:S, THIS IS MODE 2
120 T: "I"YPED TEXT APPEARS BELOW SCRE
EN
130 PA:240
140 CLOSE:S
150 J:*TEST0
160 *TEST0
170 C:@B1373-0
180 C:@B1374-0
190 WRITE:S, THIS IS WRITE IN MODE 0
200 PA:100
```


done Colors For Your Pilot

This time I will show you how to use all 128 colors of the ATARI and how you can rapidly change these colors in your displays. To display data on the TV screen, PILOT first gets data (character or graphics information) from your program and then looks at special memory locations to determine the color to use. You can use a maximum of four colors at one time on your screen. Each color is selected by the PEN:(color) instruction. This instruction calls these locations by the names "Red," "Blue," "Yellow," and "Erase." Once PILOT knows what name (location) a line belongs to, it uses the color value found there for all lines drawn by that PEN:(color) instruction.

When PILOT looks at the "Blue" location it will find a color value there. This value will cause the ATARI to draw blue lines when you first turn it on. Fortunately, you can put any color value into these locations. So, even though PILOT calls these locations by color names (for convenience) any color may be found there. You can change these colors using a special form of the C:ompute command. Turn your machine on and type this in direct mode:

C:@B710=86

C:@B712=5*16+6

The first instruction might be spoken "Compute byte 710 equals 86." In this case, the 710 is the special address PILOT calls its "Blue" location. The 86 is a color value for a red color. In effect we put "red paint into a can labeled blue."

In the second instruction, the 712 is PILOT's "Erase" register. The "5" is a hue (color) number and the "6" is a luminance number (more on them later).

In the graphics mode, PILOT uses four locations, or registers. Their names, addresses and uses are listed in Table 1.

You change the color of any register (paint can) by placing a different color value in any of the addresses. Color values are made up of two numbers, a "hue number" and a "luminance" or brightness number. Table 2 gives these values and what they usually look like on my TV.

COLORS FOR YOUR PILOT

TABLE 1

Name	Register	Value	Used for	Address
Red	0	70	Graphics	708
Yellow	1	26	Graphics	709
Blue	2	148	Text Window & Graphics	710
None	3	148	Not Used	711
Erase	4	0	Background & Border	712

TABLE 2

Hue	Luminance
0=gray	0—lowest possible luminance (black)
1=green brown	2—
2=yellow/orange	4—
3=orange	6—
4=red/orange	8—
5=pink	10—
6=bluish purple	12—
7=purple	14—maximum luminance (white)
8=blue	
9=bright blue	
10=turquoise	
11=greenish blue	
12=green	
13=yellowish green	
14=orangish green	
15=light orange	

The color value needed in each register is calculated as follows:

Hue number *16+luminance number.

A color value for the red we used above is 86 or "16*5+6." Changing a register can be done at any time in your program.

The listing draws two horses in different color registers and then changes the colors rapidly to illustrate the power of this technique.

Let me leave you with an experiment: Use Mode 1 or Mode 2 letters (see previous article) and determine which color registers are used for upper-case and lower-case letters.

PILOT YOUR ATARI

You may be interested in a new learning club for PILOT/Logo users. It has a good newsletter, simple programs and an educational orientation. It is free to people under 18. Write to:

Young People's Logo Association
1208 Hillsdale Drive
Richardson, Texas 75081

by Ken Harms

```
10 R:HOUSES
20 R:----- Draws houses and shifts
30 R:----- all four color registers
40 R:----- ANTIC Issue 3
50 *COLOR
60 GR: CLEAR
70 GR: GOTO -20,10
80 U: *HOUSE
90 GR: GOTO 20,10
100 U: *HOUSE
110 U: *REGISTER0
120 PA: 240
130 U: *REGISTER1
140 PA: 240
150 U: *REGISTER2
160 PA: 240
170 U: *REGISTER4
180 E:
190 *HOUSE
200 GR: PEN YELLOW
210 GR: TURNT00
220 GR: TURN135; DRAW 14
230 GR: TURN 45; PEN BLUE; DRAW 15 [REG 2
240 GR: TURN 90; DRAW 5
250 GR: TURN 90; FILL 8
260 GR: TURN -90; DRAW 10
270 GR: TURN -90; PEN RED; FILL 8 [REG 0
280 GR: TURN 90; PEN BLUE; DRAW 5
290 GR: TURN 90; FILL 14
300 GR: TURN 45; PEN YELLOW; FILL 14 [REG
1
```

```

310 E:
320 *REGISTER0
330 C:#A-192 [HUE 12 LUM 0
340 *INCREMENT0
350 C:@B708-#A
360 T:708 - #A
370 PA:30
380 C:#A-#A+2
390 J(#A<202):*INCREMENT0
400 E:
410 *REGISTER1
420 C:#A-224 [HUE 14 LUM 0
430 *INCREMENT1
440 C:@B709-#A
450 T:709 - #A
460 PA:30
470 C:#A-#A+2
480 J(#A<228):*INCREMENT1
490 E:
500 *REGISTER2
510 C:#A-80 [HUE 5 LUM 0
520 *INCREMENT2
530 C:@B710-#A
540 T:710 - #A
550 PA:30
560 C:#A-#A+2
570 J(#A<88):*INCREMENT2
580 E:
590 *REGISTER4
600 C:#A-144 [HUE 9 LUM 0
610 *INCREMENT4
620 C:@B712-#A
630 T:712 - #A
640 PA:30
650 C:#A-#A+2
660 J(#A<152):*INCREMENT4
670 E:

```

The Musical Pilot

This article will open the door to string parsing, a powerful way to analyze PILOT strings. Along the way, we'll read and write on the disk (or cassette), do some Boolean algebra, change data types and reveal a beautiful PILOT bug. And, oh yes, we'll play four-voice music.

As always, we'll be way "beyond the book." Since it will be getting pretty deep, I'll give page references to Atari's *PILOT Primer*.

A string is a combination of letters, numbers, symbols, words, etc., "strung together." In PILOT, a "string variable" is made by giving it a name (always beginning with "\$") in an A:cept or C:ompute instruction (pp. 69-76). The book tells how to concatenate ("grow") strings. We'll discuss how to parse ("cut") strings so you can analyze each part of a string. This could be useful for analyzing sentences, riddles, or in this case, for storing data for a program's use (PILOT lacks a "Data" statement).

String parsing relies on the Match String command which produces three pre-named variables, \$Left, \$Match, and \$Right (pp. 41-44, 81-82). Parsing programs work as follows (refer to the Pilot Player listing):

1. Place the string into the "accept buffer" (line 1270).
2. Match on the "separator." In this case, I used the blank as a separator. In line 1280, we skip over the initial blank, which the A:cept instruction inserts in each string, and M:atch on the second blank. (Note the right arrow in the instruction which doesn't print in front of the "_").
3. Check for the end of string (the JN: in line 1290).
4. Store the remainder of the string (found in \$Right) in a safe place (line 1300).
5. Use \$LEFT as the parsed word, letter, etc. (lines 1310-1370).
6. Jump back to step 1.

Although this may seem complicated, it's conceptually as easy as BASIC.

To play a C,D,E,F chord for a sixteenth, the Pilot Composer produces a string looking like this: "1 3 5 6 16 !". The first four values are the usual notes (pp. 106-107) for each of the ATARI's four voices. The "16" is the inverse duration of the note (1/16 of a note). The "!" is a "terminator" to tell us that we're out of notes. Our problem: parse it and play it. The *Loop2 routine

THE MUSICAL PILOT

(lines 1250–1390) cuts the string and sets up variables for each voice and for the PAUSE command. After each Match String, the variables look this way (the underlines represent blanks):

	\$PLAYVALUES				
PASS	BEFORE MATCH	\$LEFT	\$MATCH	\$RIGHT	\$LEFT USED FOR
0	<u>_1_3_5_6_16_</u>	NULL	NULL	NULL	
1	<u>_1_3_5_6_16_</u>	<u>_1_</u>	<u>_____</u>	<u>_3_5_6_16_</u>	#A
2	<u>_3_5_6_16_</u>	<u>_3_</u>	<u>_____</u>	<u>_5_6_16_</u>	#B
3	<u>_5_6_16_</u>	<u>_5_</u>	<u>_____</u>	<u>_6_16_</u>	#C
4	<u>_6_16_</u>	<u>_6_</u>	<u>_____</u>	<u>_16_</u>	#D
5	<u>_16_</u>	<u>_16_</u>	<u>_____</u>	<u>_____</u>	#L
6	<u>_____</u>	<u>_16_</u>	<u>_____</u>	<u>_____</u>	NO MATCH

Simply put, each value marches to the left into the \$LEFT bucket and then gets used. Notice that the “no match” in pass six did not change any of the special string variables.

The PILOT Composer parses strings in a similar fashion but on each letter. In this case, the match parsing instruction (line 1200) skips two spaces (the leading blank and the first letter) and M:atches on the next character to put *all* remaining characters in \$MATCH (the comma does that). Once the string is split, a simple \$LEFT inspection finds the character and then restores the balance of the string. The *TRANSLATE module (lines 1400–1690) performs a similar M:atch to find good notes and durations in \$GOODNOTES and \$GOODDURATION, and then to translate them into note and duration values. The translation lookup in \$NOTEABLE is “fail safe” — it first M:atches on the note followed by “/” and then M:atches on the subsequent “.” This forces the value (a 5, say) into \$LEFT. This was required, since at M:atch for 1 or 8 without the “.” would have found the value of notes C and G. Of course, I could have designed the string in reverse order — that’s an improvement for you to make.

Let’s digress to the music before going on with the programming. The PILOT Music “System” now has two simple programs. PILOT Composer accepts four-note chords composed of the eight basic notes (no sharps or flats), followed by a duration (a whole note, half note, etc.). It checks these data, catches most errors, and rings a “bell” when it’s ready for another chord. It won’t find short chords, so make sure you enter four notes and a duration, or change the *TRANSLATE module between lines 1670 and 1680. Chords are written to the disk or cassette every 10 chords. This is required since the maximum length of an accept buffer is 254 characters.

PILOT YOUR ATARI

The PILOT Player asks for a tempo (how fast to play) and a file of music. It then opens that file and plays the notes stored there.

Back to the PILOT Composer program. Under PILOT (p.73), strings are concatenated by naming two strings in a C:ompute (or A:ccept) instruction (e.g.:C:\$ONE=\$ONE\$TWO). If, however, one of the strings is “undefined,” because it has never been used before, it has the value of a text literal rather than the value of a string. In the example, if \$TWO had the value JOHN but \$ONE was undefined, the new value of \$ONE would be \$ONEJOHN — hardly what we wanted! I avoid this by initializing strings used in this way (see lines 130 and 140).

PILOT input and output (I/O) is handled with READ:, WRITE: and CLOSE: instructions. Each instruction requires a “device name” (a “C:” for cassette or a “D:” for disk) and, for disk, a file name. These are separated from following data by a comma. The data can be text literals, numeric or string variables. In a single file, READ: must be separated from WRITE: by a CLOSE:. You can try this in immediate mode or in a program:

DISK

```
WRITE:D:TEST,ABCD  
CLOSE:D:TEST  
READ:D:TEST,$STRING  
T:$STRING
```

CASSETTE

```
WRITE:C,ABCD  
CLOSE:C:  
READ:C:,$STRING  
T:$STRING
```

We’ll have more on I/O in a future article to discuss a hidden glitch. For now, just do as line 430 does and put all device specifications in a single string.

Keeping a clean screen in a program often requires erasing a line on the screen. It’s not so simple in PILOT since the “blank line” string automatically defaults to one character. Lines 750 and 1230 show an easy way; just print a series of blanks followed by a non-printing character such as an arrow. Line 750, for instance, prints the #A followed by a blank and a left arrow. When the line is printed, the right-most character is blanked out, and the left arrow holds the space, but doesn’t show. You can type an arrow by keying [ESC] then holding down the [CTRL] key while typing the desired arrow key. Repeat all three strokes for each arrow.

Although the Primer tells us that variables come in two flavors — strings (pp. 69–81) and numerics (pp. 85–92), we never find out how to change one into the other. It’s simple but tricky. String variables can be made from numeric variables by C:omputing or A:ccepting them:

```
C:$ONE=#A
A:$ONE=#A
```

A string variable can be turned into a numeric variable ONLY by A:cepting it:

```
A:#A=$ONE
```

After this instruction, #A will have the numeric value from \$ONE; non-numeric data will be disregarded (see the Player program, lines 1310-1350).

Line 1140 in the Player program presents a powerful way to combine “relational operators” to make “conditional statements” (pp. 89-90). Linking conditions with “+” signs creates “logical ors.” For instance, line 1140 would be read, “if #T=256 OR if #T=128 OR if #T=64 then J:ump . . .” In other words, if #T equaled any one of the three numbers, the program would find a “true” and J:ump. Neat! But, you can’t do it the other way, with a JN: instruction to execute on a “false,” because the “N” looks at the M:atch register, not at the conditionals.

You can get “logical ands” by multiplying the conditionals:

```
T(#T=100)*(#U=200)*(#V=50):ALL THREE
```

This statement would be read: “if #T=100 AND if #U=200 AND if #V=50 then T:ype ALL THREE.”

At last, the BUG. (A friend says that micros are too small to have bugs. She claims that they have fleas!) Right there on page 31 the Primer tells us that the computer “ignores” remarks. Although that may be accurate in the linguistic sense, it’s not so in the operative sense. In line 1150 in the Composer program the remark set off by a “[” MUST be typed without spaces. It seems that the [turns any intervening spaces into significant space and, therefore, part of the accept buffer. Ditto for other commands. I don’t know if it’s a bug or a flea — I know it’s a bear to figure out! (Atari’s internal manuals even have it wrong!) Be safe, don’t use brackets when in doubt.

by Ken Harms

```
50 R:    PILOT COMPOSER
60 R:    ANTIC, VOL. 1, NO. 4
70 R:    K. W. HARMS
80 R:
100 R:    INIT
```


PILOT YOUR ATARI

```
110 *INIT
120 C:#A=0
130 C:$NOTEVALUES=
140 C:$PLAYVALUES=
150 C:$END=!
160 C:$GOODNOTES=C D E F G A B 0
170 C:$GOODDURATION=1 2 4 8 S 0
180 C:$NOTETABLE=C. 1/ D. 3/ E. 5/ F.
6/ G. 8/ A. 10/ B. 12/ 0. 0/ 1. 1/ 2.
2/ 4. 4/ 8. 8/ S. 16/
300 R:      FILE
310 *FILE
320 R:
330 T:ENTER DEVICE TO SAVE MUSIC ON
340 T:D=DISK, C=CASSETTE
350 A:$D
360 R:NEXT, CHECK TO SEE IF CASSETTE
370 M: C
380 CY:$FILESPEC=C:
390 JY:*FILEDONE [IF CASS JUMP OUT
400 M: D
410 TY:ENTER FILE NAME
420 AY:$FILE [GET FILE NAME
430 CY:$FILESPEC=$D:$FILE
440 TN:I DON'T KNOW THAT DEVICE
450 JN:*FILE
460 *FILEDONE
470 T: [ESC-CTRL-CLEAR .. CLEARS SCREEN
500 R:      INSTRUCTIONS
510 *INSTRUCTIONS
520 R:
530 T:
540 T:NOTES ARE: C D E F G A B
550 T:      AND 0 FOR OFF
560 T:
570 T:DURATIONS ARE:
580 T:      1=WHOLE      2=HALF
590 T:      4=QUARTER    8=EIGHTH
600 T:      S=SIXTEENTH  0=NONE
610 T:
620 T:ENTER & TO QUIT
630 T:
```

```

700 R:          ENTER
710 *ENTER
720 R:
730 C:#A=#A+1
740 POS:1,12
750 T:ENTER 4 NOTES + DURATION FOR CHO
RD #A [SPACE, ESC-CTRL-LEFT
760 POS:17,15
770 A:$NOTES
780 M:&
790 JY:*ENDER
800 EY:
810 U:*CHECKNOTES
820 SO:20 [BEEP ON COMPLETION
830 PA:7
840 SO:0
850 WRITE(#A=10):$FILESPEC,$PLAYVALUES
860 C(#A=10):#A=0
870 J:*ENTER
900 R:          ENDER
910 *ENDER
920 R:
930 C:$PLAYVALUES-$PLAYVALUES!
940 WRITE:$FILESPEC,$PLAYVALUES
950 CLOSE:$FILESPEC
960 T:
970 T:          SAVED IN FILE $FILESPEC
980 T:
990 T:          SESSION ENDED
1000 E:
1100 R:          CHECKNOTES
1110 *CHECKNOTES
1120 R:
1130 A:=$NOTES [MOVE $N. TO ACCEPT
1140 MS:, [MATCH ON 1ST BLANK
1150 A:=$RIGHT!/[ADD/,MOVE TO ACCEPT
1160 C:#C=0 [SETS NOTE COUNTER TO 0
1170 C:$NOTEVALUES-
1180 C:#G=0
1190 *LOOP
1200 MS:[ ] , [SKIPS 2 SPACES
1210 CN(#G=0):$PLAYVALUES=$PLAYVALUES$
NOTEVALUES

```

PILOT YOUR ATARI

```
1220 POSN(#G-0):2,22
1230 TN(#G-0):
      [ESC-CTRL-UP
1240 EN:
1250 MS:$RIGHT[MATCH W/O 1ST LETTER
1260 C:$SAVE-$MATCH [SAVE ALL
1270 A:-$LEFT [$L. HAS BLANK+LETTER
1280 MS:[ ] [SKIP BLANK & LETTER
1290 R:$LEFT HAS THE LETTER WE NEED
1300 C:$NOTE-$LEFT
1310 U:*TRANSLATE
1320 A:-$SAVE [PUT ALL IN BUFFER
1330 J:*LOOP
1400 R:      TRANSLATE
1410 *TRANSLATE
1420 R:
1430 C:#C-#C+1
1440 E(#C-7):
1450 A(#C<5):-$GOODNOTES
1460 A(#C<5):-$GOODDURATION
1470 M:$NOTE
1480 POSN:2,22
1490 TN:ERROR IN THIS VALUE: $NOTE
1500 R:SET G FLAG FOR BAD NOTE
1510 CN:#G-1
1520 EN:
1530 A(#C-6):-$NOTE
1540 M(#C-6):!
1550 EY(#C-6):
1560 POSN(#C-6):2,22
1570 TN(#C-6):TOO MANY VALUES:$NOTE
1580 CN(#C-6):#G-1
1590 EN(#C-6):
1600 POS(#C>6):2,22
1610 T(#C>6):TOO MANY VALUES: $NOTE
1620 C(#C>6):#G-1
1630 E(#C>6):
1640 A:-$NOTETABLE
1650 MS:$NOTE.
1660 A:-$RIGHT
1670 MS:/
1680 C:$NOTEVALUES-$NOTEVALUES$LEFT
1690 E:
```

```

50 R:          PILOT PLAYER
60 R:          ANTIC, VOL. 1, NO. 4
70 R:          K. W. HARMS
80 R:
300 R:         FILE
310 *FILE
320 R:
330 T:ENTER DEVICE TO PLAY MUSIC FROM
340 T:D-DISK, C-CASSETTE
350 A:$D
360 R:NEXT, CHECK TO SEE IF CASSETTE
370 M: C
380 CY:$FILESPEC-C:
390 JY:*FILEDONE [IF CASS JUMP OUT
400 M: D
410 TY:ENTER FILE NAME
420 AY:$FILE      [GET FILE NAME
430 CY:$FILESPEC-$D:$FILE
440 TN:I DON'T KNOW THAT DEVICE
450 JN:*FILE
460 *FILEDONE
470 T: [ESC-CTRL-CLEAR .. CLEARS SCREEN
1000 R:         TEMPO & PLAY
1010 R:
1020 R:         TEMPO
1030 *TEMPO
1040 T: [ESC-CTRL-CLEAR CLEARS SCREEN
1050 POS:9,5
1060 T:PLEASE ENTER A TEMPO
1070 T:
1080 T:          256 - Adagio
1090 T:          128 - Andante
1100 T:          64 - Allegro
1110 POS:17,11
1120 *RESTART
1130 A:#T
1140 J(#T-256)+(#T-128)+(#T-64):*READ
1150 T:PLEASE ENTER NUMBER AGAIN
1160 J:*RESTART
1170 R:          READ
1180 *READ
1190 T:

```

```
1200 T:          PLAYING FILE $FILESPEC
1210 READ:$FILESPEC,$PLAYVALUES
1220 R:THIS DEMOS WORD PARSING
1230 *LOOP1
1240 C:#N-0
1250 *LOOP2
1260 C:#N-#N+1
1270 A:-$PLAYVALUES
1280 MS:☐_
1290 JN:*READ
1300 C:$PLAYVALUES-$RIGHT
1310 A(#N-1):#A-$LEFT
1320 A(#N-2):#B-$LEFT
1330 A(#N-3):#C-$LEFT
1340 A(#N-4):#D-$LEFT
1350 A(#N-5):#L-$LEFT
1360 A:-$LEFT
1370 M:I
1380 EY:
1390 J(#N<5):*LOOP2
1400 SO:#A#B#C#D
1410 PA:#T/#L
1420 J:*LOOP1
```

Holiday Trees

Add to your holiday pleasure by decking out these cybernetic trees using this PILOT program. It comes complete with colored lights, a scrolling message, and “Jingle Bells” in one-part harmony. To do this we will use some innovative techniques that will expand your understanding of PILOT programming.

Let’s wander through the listing. After the title lines, we find a J:ump command at line 50. As you’ll see, we U:se *PARSE, *COLORS, and *LLOOP over and over as the program operates. Each time PILOT hits a U:se or J:ump command, it goes to the first instruction (in this case, line 1) and reads every line until it finds the required module name. Putting often-used modules near the front of the listing makes the program run faster. PILOT is fast. Even putting the modules at the end of the 225 lines of this program did not noticeably slow down the song, but this programming concept makes it run even faster.

Now J:ump to *DRAWTREES (lines 1000–1540). This module uses a mirror-image concept to draw two trees for nearly the price of one. Notice that the first tree is drawn at X=–40, Y=32 (lines 1050–1070) and the second at X=40, Y=32 (lines 1080 and 1090). This means that the Y positions in both trees are the same while the X positions differ by only the sign. As a result, we can draw in the same location in both trees by using positive and negative values of the same number for the X position.

We use this concept to draw the stars and balls with a single position and *MIRRORSTAR and *MIRRORBALL modules (lines 2100–2160 and 2400–2460). The C:ompute instruction in line 2140 changes the sign of #X by multiplying it by –1. Simple and neat!

Back to the *TREE module. PILOT graphics uses only four colors. Although it calls these RED, BLUE, YELLOW and ERASE, PILOT really looks at a memory location each time it draws in a PEN color to see what color should be used. Normally, of course, it finds a number in BLUE which means blue. In line 1650, we force a different number into location 708 to tell PILOT that we want it to draw in green whenever it hits a BLUE command. Line 1760 sets the RED pen to brown. Location 709 controls YELLOW and

PILOT YOUR ATARI

711 the ERASE commands. You might want to experiment to see how these “registers” work.

After we finish drawing and decorating the trees, we end up at line 1530, which C:omputes a string into the \$MESSAGE variable. I had to double-space the message because the A:ccept command, used later in the *PARSE module, automatically inserts blanks at the start and end of each string. At present, there doesn't seem to be a good way around this restriction, but we end up with a nice message anyway. Although the printer doesn't show it, an Escape character is placed between each word to preserve word spacing. This is necessary because A:ccept also condenses all multiple spaces to single spaces. The Escape character will not print the message: you enter it by pressing the [ESC] key twice.

You'll probably want to enter your own message. Just type [space] [ESC] [space] between each word and two [ESC]'s at the end. Also, keep the message less than 255 letters long.

When finished drawing the trees, we J:ump to *MAINLOOP (lines 600–699). This module is the workhorse, it plays the song, calls for the message and color changes. It's rather long but really simple to type in. All the *LLOOP commands are on multiples of three — just type it once and use ATARI's wonderful screen editor to change the line number. Ditto for the SO:ound and PA:use commands.

*MAINLINE does one other important thing. Since the program doesn't use any keystrokes, the ATARI would soon begin changing screen colors. The C:ompute in line 688 puts a 0 in location 77 to tell the computer that a key has been pressed even when none was. This delays the “attract” mode each time through the loop.

The next module, *LLOOP, simply calls *PARSE and *COLORS. The *PARSE module breaks strings into individual characters (“parsing”). As you type it, remember the two right arrows in line 150 and 37 in line 180. The arrows tell the MS: command to skip a character for each arrow before looking for a M:atch.

After skipping 37 characters in line 180, the MS:\$RIGHT in line 190 forces the first 37 letters into the \$LEFT string which we T:ype in line 210. That's the billboard section of the message. By repeatedly stripping off the first character and adding it to the end of the message, we make the words march across the text window at the bottom of the graphic screen. Oh yes, C:@B656? That's a memory location which tells PILOT to T:ype the message

HOLIDAY TREES

on the second line of the text window. Without that, each message would Type on a different line and would scroll off the top. (Just for fun, the lines are numbered 0 through 3.)

Although *PARSE is busy, *COLORS (lines 300–400) is a speedy devil too. By C:omputing different values for location 709, *COLORS changes the color in the YELLOW pen. This flashes red, blue, brown, and yellow in the stars and balls.

To close, let me answer two questions. How do I get PILOT to number the modules in different series? Simple. As I build a program, each module is stored in a different disk file. After all modules are debugged, each is LOAded into memory and RENumbered in a number series which doesn't overlap with any other module. It's then SAVED, memory NEWed and the next module loaded. After all are RENumbered, all are LOAded into a complete program and SAVED in a different file.

Last, how do I get those big letters in the R:emarks? Just enter a control N (a bar symbol) right after the colon.

Best wishes for a happy holiday season watching your cybernetic trees!

by Ken Harms

```

1 R:EEQ&
10 R:CHRISTMAS TREES
20 R:
30 R:      ANTIC, VOLUME 1, NO. 5
40 R:
50 J:*DRAWTREES
100 R:
110 R:      PARSE
120 R:
130 *PARSE
140 A:-$MESSAGE
150 MS:  ,
160 MS:$RIGHT
170 A:$MESSAGE-$MATCH$LEFT
180 MS:
,
185 R: LINE 180 IS 37 RIGHT ARROWS AND
      COMMA

```

PILOT YOUR ATARI

```
190 MS:$RIGHT
200 C:@B656-1
210 T:$LEFT
220 E:
300 R:
310 R:▢ COLORS
320 R:
330 *COLORS
340 C:#B-#B+1
350 C(#B-1):@B709-146
360 C(#B-2):@B709-66
370 C(#B-3):@B709-26
380 C(#B-4):@B709-18
390 C(#B-4):#B-0
400 E:
500 R:
510 R:▢ LLOOP
520 R:
530 *LLOOP
540 U:*COLORS
550 U:*PARSE
560 SO:0
570 E:
600 R:
601 R:▢ MAINLOOP
602 R:
603 *MAINLOOP
604 U:*PARSE
605 R: 1ST PARSE TO GET TEXT
606 R: NOTE NUMBER SEQUENCE
607 SO:22
608 PA:16
609 U:*LLOOP
610 SO:22
611 PA:16
612 U:*LLOOP
613 SO:22
614 PA:32
615 U:*LLOOP
616 SO:22
617 PA:16
618 U:*LLOOP
619 SO:22
```

620 PA:16
621 U:*LLOOP
622 SO:22
623 PA:32
624 U:*LLOOP
625 SO:22
626 PA:16
627 U:*LLOOP
628 SO:25
629 PA:16
630 U:*LLOOP
631 SO:18
632 PA:24
633 U:*LLOOP
634 SO:20
635 PA:8
636 U:*LLOOP
637 SO:22
638 PA:48
639 U:*LLOOP
640 SO:0
641 PA:16
642 U:*LLOOP
643 SO:23
644 PA:16
645 U:*LLOOP
646 SO:23
647 PA:16
648 U:*LLOOP
649 SO:23
650 PA:24
651 U:*LLOOP
652 SO:23
653 PA:8
654 U:*LLOOP
655 SO:23
656 PA:16
657 U:*LLOOP
658 SO:22
659 PA:16
660 U:*LLOOP
661 SO:22
662 PA:16

PILOT YOUR ATARI

```
663 U:*LLOOP
664 SO:22
665 PA:8
666 U:*LLOOP
667 SO:22
668 PA:8
669 U:*LLOOP
670 SO:25
671 PA:16
672 U:*LLOOP
673 SO:25
674 PA:16
675 U:*LLOOP
676 SO:23
677 PA:16
678 U:*LLOOP
679 SO:20
680 PA:16
681 U:*LLOOP
682 SO:18
683 PA:48
684 U:*LLOOP
685 SO:0
686 SO:0
687 PA:64
688 C:@B77-0
689 J:*MAINLOOP
1000 R:
1010 R:▣ DRAWTREES
1020 R:
1030 *DRAWTREES
1040 GR:CLEAR
1050 C:#X--40
1060 C:#Y--28
1070 U:*TREE
1080 C:#X-40
1090 U:*TREE
1100 R: NOW PUT SOME STARS ON THEM
1110 C:#X--40
1120 C:#Y-32
1130 U:*STAR
1140 C:#X-40
1150 U:*STAR
```




HOLIDAY TREES

```
1160 R: OK THAT DID THE TOPS, NOW FOR
A FEW MORE
1170 C: #X--48
1180 C: #Y-16
1190 U: *STAR
1200 U: *MIRRORSTAR
1210 C: #X--32
1220 U: *STAR
1230 U: *MIRRORSTAR
1240 C: #X--56
1250 C: #Y-0
1260 U: *STAR
1270 U: *MIRRORSTAR
1280 C: #X--24
1290 U: *STAR
1300 U: *MIRRORSTAR
1310 C: #X--65
1320 C: #Y--20
1330 U: *STAR
1340 U: *MIRRORSTAR
1350 C: #X--13
1360 U: *STAR
1370 U: *MIRRORSTAR
1380 R: HOW BOUT A FEW BALLS?
1390 C: #X--43
1400 C: #Y-8
1410 U: *BALL
1420 U: *MIRRORBALL
1430 C: #X--50
1440 C: #Y--10
1450 U: *BALL
1460 U: *MIRRORBALL
1470 C: #X--33
1480 C: #Y--12
1490 U: *BALL
1500 U: *MIRRORBALL
1510 R: TREES DRAWN, SET UP TYPING, CO
LOUR AND MUSIC LOOP
1520 R: SPACE BETWEEN EACH CHARACTER, H
IT SPACE,ESC,ESC,SPACE BETWEEN EACH WO
RD AND SPACE,ESC,ESC,SPACE,ESC,ESC AT
E
1530 C: $MESSAGE-H A V E A H A P P
```

PILOT YOUR ATARI

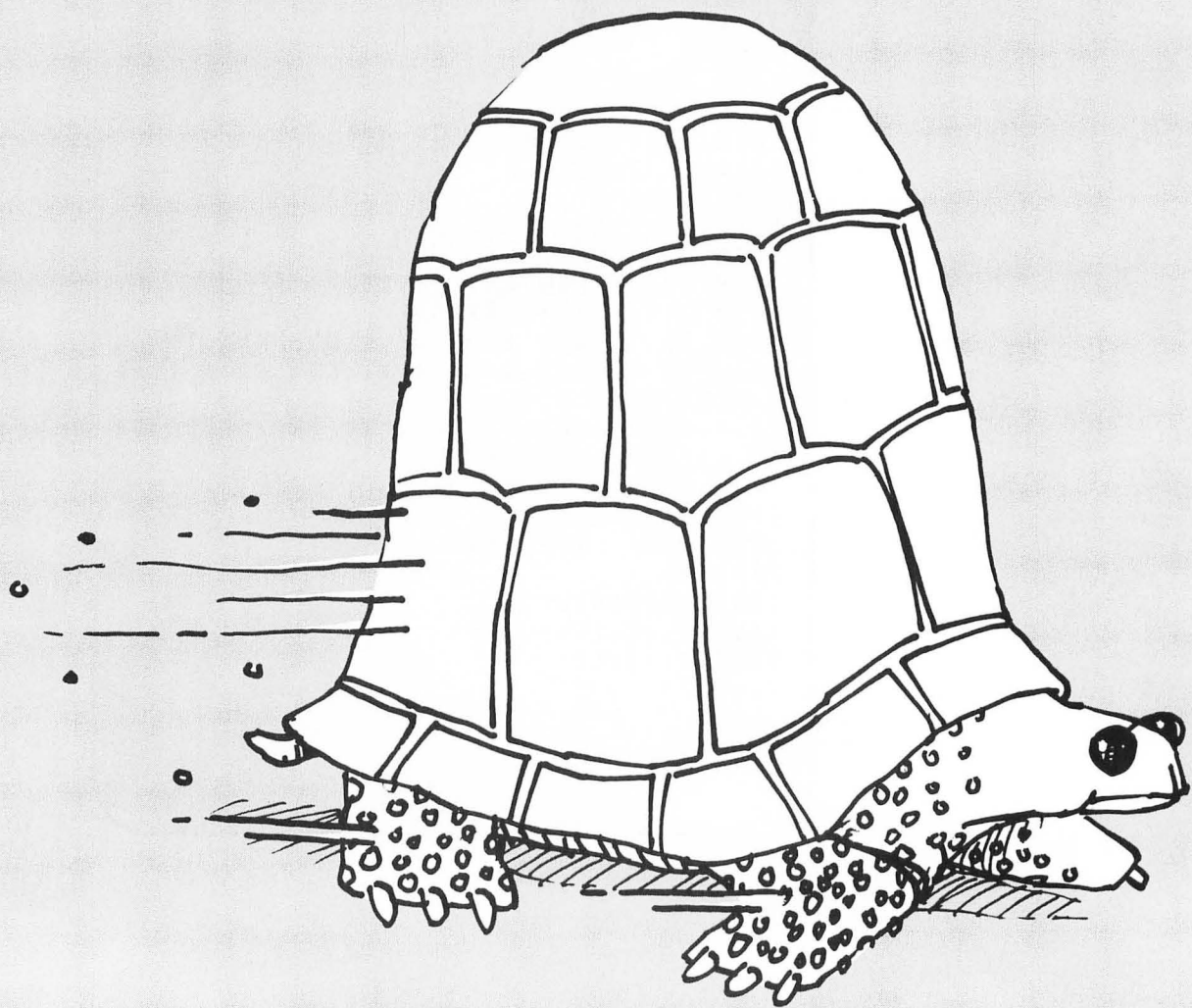
```
Y H O L I D A Y !
1540 J: *MAINLOOP
1600 R:
1610 R: TREE
1620 R:
1630 *TREE
1640 R: NEXT LINE SETS "BLUE" PEN TO
GREEN
1650 C:@B710=(12*16)+6
1660 GR:PEN BLUE
1670 GR:GOTO #X+28,#Y+5
1680 GR:TURNTO 0
1690 GR:TURN -26
1700 GR:DRAW 63
1710 GR:TURN 232
1720 GR:DRAW 2
1730 GR:FILL 61
1740 R: DRAW THE TRUNK
1750 R: NEXT LINE SETS "RED" PEN TO BR
OWN
1760 C:@B708=(14*16)+(4)
1770 GR:PEN RED
1780 GR:GOTO #X+4,#Y
1790 GR:TURNTO 0
1800 GR:DRAW 4
1810 GR:PEN ERASE
1820 GR:GOTO #X-4,#Y-1
1830 GR:PEN RED
1840 GR:FILL 5
1850 E:
1900 R:
1910 R: STAR
1920 R:
1930 *STAR
1940 GR:PEN YELLOW
1950 GR:GOTO #X,#Y
1960 GR:TURNTO 0
1970 GR:DRAW 4
1980 GR:TURN 180
1990 GR:DRAW 2
2000 GR:TURN 90
2010 GR:DRAW 2
2020 GR:TURN 180
```

```

2030 GR: DRAW 4
2040 E:
2100 R:
2110 R:  MIRRORSTAR
2120 R:
2130 *MIRRORSTAR
2140 C: #X = (#X * -1) + 1
2150 U: *STAR
2160 E:
2200 R:
2210 R:  BALL
2220 R:
2230 *BALL
2240 GR: PEN YELLOW
2250 GR: GOTO #X, #Y
2260 GR: TURNT0 0
2270 C: #A = 0
2280 *STARTBALL
2290 C: #A = #A + 1
2300 GR: 4 (DRAW #A; TURN90)
2310 J (#A < 3): *STARTBALL
2320 GR: TURNT0 270; PEN BLUE; DRAW 1
2330 GR: 1 (TURN 90; PEN YELLOW; DRAW 2; PE
N BLUE; DRAW 2)
2340 GR: 3 (TURN 90; DRAW 1; PEN YELLOW; DR
AW 2; PEN BLUE; DRAW 2)
2350 E:
2400 R:
2410 R:  MIRRORBALL
2420 R:
2430 *MIRRORBALL
2440 C: #X = (#X * -1) + 1
2450 U: *BALL
2460 E:

```

Forth Factory



Turtle Graphics

This chapter first appeared in *ANTIC* as two articles implementing a turtle graphics system in Forth.

Let me make two quick points about Forth:

- Doing this project in any other computer language would have been so involved that I would never have done it, and so lengthy that this magazine would never have published it.
- Doing it in Forth was so easy it took me considerably longer to write the English for this article than the Forth code!

Those of you who have Pink Noise Studios' *pns-Forth* (I use version 1.4) can edit the screens accompanying these articles "as is" and start turtle-ing. If you have another implementation of Forth for the ATARI, some revisions are inevitable. I have used words like *PLOT* and *DRAWTO* that *pns-Forth* provides for making graphics calls to the ATARI's Operating System. Your system may already have similar words. Later, I'll discuss the functions of any non-fig-Forth words that I've used.

Turtle Graphics Versus Coordinate Graphics

"Turtle graphics" is a simple but powerful approach to creating graphic designs with a computer. It was originally developed in the 1960's at MIT — primarily by computer scientist, child psychologist and educator, Seymour Papert — as part of the Logo system.

Let me give you a very simple example of how it works. Suppose we want to draw a square on the screen, 10 units on a side. The sequence of commands

10 DRAW	10 DRAW
90 TURN	90 DRAW
10 DRAW	10 DRAW
90 TURN	90 TURN

Gordon Smith is a graduate student in physiology at Stanford University. He is interested in graphics and music, as well as Forth programming.

FORTH FACTORY

or, in a shorter form,

```
4 ( 10 DRAW 90 TURN )
```

requests an imaginary “turtle” on the screen to crawl 10 units forward, draw a line as it goes, turn 90 degrees clockwise, and repeat four times. The turtle will leave behind a square.

By typing

```
DEFINE SQUARE AS 4 ( 10 MOVE 90 TURN ) END
```

we can add the new command `SQUARE` to our turtle’s graphics repertoire. Then typing the single command

```
SQUARE
```

will have the same effect as our previous sequence of commands. For example, to draw a square tilted by thirty degrees, we need only to type

```
30 TURN
```

```
SQUARE.
```

The conventional approach to graphics, in which one must specify fixed screen coordinates and the endpoints of each line, is much more complicated.

The principle advantage of turtle graphics is that it describes shapes in an intrinsic way, without referring to where they are or how they’re oriented. The numbers used in turtle graphics represent easily visualized things, like lengths of lines or angles.

A further important aspect of a turtle graphics system is the nature of the programming it encourages: structured, modular, and hierarchical. The `DEFINE . . . AS . . . END` construct shown above is the key to this. Basic sub-designs can be made into new turtle commands which are then as much a part of the turtle’s language as the predefined system commands. These higher-level commands can then be used to define still higher ones, and so on.

For example, a simple picture of a house like that in Figure 1 could be drawn with a long sequence of `DRAW`s and `TURN`s (along with another command for the turtle to move without drawing). But the structure of the design cries out for the programmer to instead first enrich the turtle’s vocabulary by defining commands such as, perhaps, `RECTANGLE`, `WINDOW`, `DOOR`, `FRONT` and `ROOF`, before using these higher-level commands to define one called `HOUSE`.

The Forth Advantage

Forth is so ideally suited to turtle graphics that, in a sense, implementing it is a trivial exercise.

The most complicated aspect of turtle graphics is the problem of providing a programming environment in which turtle commands can be executed. Such a capability is already intrinsic to Forth, while it is quite foreign to conventional languages like BASIC.

The point here is that the turtle's language can be just an extension of the Forth language — turtle commands are simply Forth words. There is no need to write an extensible command language processor. That's what a Forth system already is!

What the Screens Contain

The ten screens of Forth listed in this article lay the necessary foundations for us to build a turtle graphics system. The words here are not specifically turtle-oriented. Rather, they extend Forth's capabilities in directions particularly useful to the application.

Screens 1, 2, and 3 add some trigonometric capability to Forth. If the turtle is to move 10 units forward at 30 degrees from the vertical, we need to compute how far up and how far over she goes. For this we use a lookup-table approach. Scaling the values by 10,000 enables us to store them as single-precision integers. The words SIN* and COS* are the result of this.

For example,
10 30 SIN*

leaves 5, or 10 times the sine of 30 degrees, on the stack; and this is how far over the turtle would move.

Screen 4a makes available a defining word, VALUE, for a new data type. An alternative to CONSTANT and VARIABLE, VALUE words tend to make Forth code more readable. They are best explained by the following example:

```
VALUE A VALUE B VALUE C
ok
2 TO A 3 TO B
ok
A . B .
2 3 ok
A B + TO C C .
5 ok
```

VALUE words return their value when executed, except when they are preceded by TO, in which case they store the top of the stack into

FORTH FACTORY

themselves. (This idea has been discussed in the “Forth Dimensions” newsletter of the Forth Interest Group.)

In screen 4a the words `TO` and `VALUE` are defined in assembly language, rather than Forth, so that they will execute as fast as `CONSTANT`s and `VARIABLE`s. If you don’t have an assembler, use the alternate Forth code on screen 4b.

Screens 5 through 8, culminating in the word `CLIP`, implement a line-clipping algorithm. We want the turtle to be able to cross the edge of the screen, so that if we execute `SQUARE` when she is near the top we’ll get something like Figure 2. But the Operating System will refuse to draw a line whose endpoints aren’t both within the screen boundaries. Therefore, we must be able to calculate the endpoints of the portion of the line which lies on the screen. If we give `CLIP` the coordinates of two points, it first determines whether any part of the line between them lies within a “clipping rectangle” whose extent we can specify by setting the values of `LEFT`, `RIGHT`, `TOP`, and `BOTTOM`. (Note that these words are in the vocabulary `CLIPPING`.) If so, it returns the coordinates of the endpoints of the portion within the clipping rectangle, and a true flag. If not, it returns only a false flag.

For example, suppose we set the clipping rectangle to be the size of the Graphics Mode-7 screen with

```
CLIPPING
0 TO LEFT
159 TO RIGHT
0 TO TOP
79 TO BOTTOM
Then
30 30 50 50 CLIP
```

leaves `30 30 50 50 1` on the stack because the line between (30,30) and (50,50) is completely within the clipping rectangle. But

```
80 100 200 40 CLIP
```

leaves `122 79 159 61 1` because only the portion between (122,79) and (159,61) of the specified line lies inside the clipping rectangle. And

```
200 200 300 300 CLIP
```

leaves `0` because no part of the line lies inside. The Cohen-Sutherland algorithm that `CLIP` uses is described in detail in Chapter 5, “Clipping and Windowing,” of Newman and Sproull’s *Principles of Interactive Computer Graphics*.

The last screen, number 9, defines the word GRAPHICS for opening the screen in the graphics mode specified by the top of the stack, and LINE, which takes the coordinates of two endpoints and draws the clipped part of it on the screen.

If you want to see the clipping in action, before the rest of the code is given, try the following: Define the words BORDER, RANDOM__LINE, and RANDOM__LINES as

```
: BORDER
    CLIPPING
    1 COLOR
    LEFT BOTTOM PLOT
    LEFT TOP DRAWTO
    RIGHT TOP DRAWTO
    RIGHT BOTTOM DRAWTO
    LEFT BOTTOM DRAWTO ;

: RANDOM__LINE
    4 0 DO CRANDOM LOOP LINE ;

: RANDOM__LINES
    0 DO RANDOM__LINE LOOP ;
```

and then type

```
CLIPPING
20 TO LEFT
140 TO RIGHT
20 TO TOP
60 TO BOTTOM
7 GRAPHICS
BORDER
100 RANDOM__LINES
```

The Inhabitants and Language of Turtleland

Four independent turtles live in Turtleland. Multiple turtles open up interesting possibilities, like having turtles chase each other. With four turtles, each can draw in a different color (there are only four colors possible at one time). If you want a different number, you can change the value of the constant #TURTLES on screen 2 before loading. One turtle at a time

FORTH FACTORY

can be designated the “active turtle” with the SET ACTIVE command. She is the one who will respond when we type a command like “10 DRAW.”

Each turtle carries a pen. The active turtle’s pen can be lowered with the PENDOWN command, leaving a trail when she moves, or raised with the PENUP command. The more general SET PEN command can be used to do either.

The SET INK command fills the active turtle’s pen with various colors of ink, depending on the graphics mode used. (Modes 3 through 8 can be selected with the SET MODE command.) In all modes, ink of type 0 is erasing ink. It is black, the same color as the background, except in Mode 8 when it is light blue. The command, ERASING, is the same as 0 SET INK. Both choose erasing ink. In Modes 3, 5, and 7, there is also ink of type 1 (gold), type 2 (light green), and type 3 (dark blue). In Modes 4, 6, and 8, types 2 and 3 are not available. The number of ink types is determined by the color video capabilities of the CTIA or GTIA chip. The colors are established by the Operating System when it opens the screen. You can use pns-Forth SET-COLOR word to change them.

Each turtle has a position and a heading. The heading is the number of degrees clockwise from the vertical that she is facing. The active turtle’s heading can be changed directly to any value with SET HEADING, also known as TURNTO, or it can be changed incrementally by the commands RIGHT (or TURN) and LEFT.

The system keeps track of each turtle’s position with X and Y coordinates. These are *not* the same as the screen column and row numbers. The SET MODE command arranges these coordinates so that the turtle’s home at X=0 and Y=0 is the center of the screen, and so that there are one hundred X or Y units per pixel. This means that if a turtle is at X=1000 and Y=500 she will appear ten pixels to the right and five pixels up from the center. You can arrange the coordinates differently if you wish.

The active turtle’s coordinates can be individually or jointly set with the commands SET X, SET Y, or SET POSITION (also known as GOTO). They cause the turtle to leave a track only if her pen is down. MOVETO can be used to temporarily raise the pen, or DRAWTO to lower it, before changing position. The pen is restored to its original state after the change.

The most interesting way to move the active turtle is with FORWARD, BACKWARD, DRAW, and MOVE commands. These move her a specified number of steps in whatever direction she is currently heading. FORWARD

and BACKWARD draw a line only if the pen is down; DRAW always draws; MOVE never does. Each step normally moves the turtle one pixel, a distance of 100 units in XY coordinates, unless you use the SET SIZE command to alter the step size. By changing the step size you can use the same word to draw the same shape in different sizes.

A turtle's heading and her XY coordinates are always integers. The maximum range for X and Y is from -32768 to 32767. If you drive a turtle beyond this range you may see unwanted tracks as she "jumps" to the other edge of Turtleland.

Usually you can't see all of Turtleland on the screen. For example: in Mode 7 the screen displays only the part of Turtleland from X=15900 to X=15800 and from Y=-7900 to Y=7800. You can select your own "window" into Turtleland with SET WINDOW command. Any tracks beyond the edges of the window won't be visible. Changing the window will affect the number of X or Y units per pixel. An alternate way to set the window (and the step size) is with the PER-PIXEL command.

The reason that the system defaults to 100 units per pixel is to let the turtle sit "between" pixels. If we used a coordinate system as coarse as the screen pixels, then every time we moved a turtle at some angle, her new position would get "rounded" to the nearest pixel. We wouldn't be able to do a series of moves without errors accumulating. Using 100 XY units per pixel gives us increased precision.

The SET MODE command establishes the whole screen as the "viewport." This means that the view of Turtleland visible through the window will be projected onto all of the screen. You can select any rectangular piece of the screen to be the viewport with the SET VIEWPORT command. When you experiment with this, use the FRAME or NEW commands to draw a frame around the new viewport so you can see where it is.

So far, four commands — MODE, SIZE, WINDOW, and VIEWPORT — relate to Turtleland as a whole, and seven of them — ACTIVE, PEN, INK, HEADING, X, Y, and POSITION — relate to the turtles. It is also possible for you to determine the current value of any of these parameters, by leaving out the word SET or by changing it to SHOW. For example, the command X by itself (i.e., not preceded by SET) leaves the active turtle's current X coordinate on the stack, where it can be used by any word for any purpose. So, the command SHOW X will display some message like "Turtle #1 is at X=300."

FORTH FACTORY

The system also has miscellaneous commands like CLEAR for clearing the screen, FRAME for drawing a frame around your picture, and HOME, START, and NEW for starting over. The command BYE leaves Turtleland and returns to pns-Forth.

Of course, all the usual Forth words are still available while you're in Turtleland, in case you need to do arithmetic, comparisons, branching, looping, or whatever. You can use the more compact loop syntax (. . .) and (. . . +) in place of the structures 0 DO . . . LOOP and 0 DO . . . +LOOP.

The important command DEFINE . . . AS . . . END allows you to add new words to the turtle's vocabulary. This makes it very easy to change any of my command names that you don't like.

As an interesting example, you might want to

```
DEFINE HILDA
  AS 1 SET ACTIVE END
DEFINE GILDA
  AS 2 SET ACTIVE END
DEFINE MATILDA
  AS 3 SET ACTIVE END
```

so that you can talk to a turtle simply by invoking her name.

Using The System

To start turtle-ing, just use the SET MODE command. If you want to have Turtleland displayed in Graphics Mode 7, for example, type 7 SET MODE. After this you can immediately move the turtles around with 10 DRAW, 45 TURN, etc. SET MODE initializes the system as follows:

- All four turtles are home at X=0 and Y=0, with heading 0 degrees.
- They all have their pens down.
- Their pens are filled with various ink types as described under the START command in the glossary.
- Turtle #1 is active.
- The window is such that X=0, Y=0 is in the center of the screen and there are 100 X or Y units per pixel.
- The viewport is the whole screen.

After you get acquainted with the various commands, you'll want to start extending the system by defining your own. Here is an example of a new command:

```
VALUE STEPS
VALUE INCREMENT
VALUE ANGLE
DEFINE POLYSPI AS
  TO ANGLE
  TO INCREMENT
  0 TO STEPS
  BEGIN
    STEPS INCREMENT +TO STEPS
    STEPS FORWARD
    ANGLE TURN
  AGAIN
END
```

POLYSPI can make all sorts of interesting polygonal spirals. It expects to find two numbers on the stack. It stores the top one in `ANGLE`; this will be how many degrees the turtle will turn between each move. The one below gets stored in `INCREMENT`; this will be how many more steps the turtle will take each time compared to the previous time. Next `STEPS` is initialized to 0 and we enter a Forth `BEGIN . . . AGAIN` loop. The words between `BEGIN` and `AGAIN` will be executed indefinitely. (You must press a yellow console button to stop `POLYSPI`.) Each time through the loop, `STEPS` is incremented by `INCREMENT`, and the turtle takes the number of steps in `STEPS` and turns the number of degrees in `ANGLE`. Thus `POLYSPI` is just an automated sequence of `FORWARD`s and `TURN`s. For example, `2 90 POLYSPI` is really the same as

```
2 FORWARD 90 TURN
4 FORWARD 90 TURN
6 FORWARD 90 TURN
```

and so on.

The three `VALUE` words `POLYSPI` uses make it easy to see what's going on. However, another definition of `POLYSPI` is possible which uses no variables at all:

```
DEFINE POLYSPI AS
  0
  BEGIN
    3 PICK +
```

FORTH FACTORY

```
DUP FORWARD
OVER TURN
AGAIN
END
```

This version keeps everything on the stack, using the Forth words PICK, DUP, and OVER for stack manipulation. You can make a variety of patterns with this one command by changing its two parameters.

Pressing a yellow console button will break out of an indefinite loop of turtle moves. In fact, every time a turtle changes position, the system checks the console buttons and returns to command level if one is depressed. This makes it easy to regain control.

As mentioned earlier, ten of the words used in my screens are pns-Forth words which won't be available (at least not with the same meanings) in other Forth systems. Two of these, 1- and TABLE, are common Forth extensions whose high-level definitions are

```
: 1- 1 - ;
and
: TABLE  BUILDS DOES  OVER
  + + @ ;
```

The others are highly system-specific. Four of them — SETUP S, CLOSE S, SPLIT-SCREEN, and GR. — were used in the word GRAPHICS. Their definitions are quite complex, as these words are part of pns-Forth's interface to the CIO routines in the Operating System. Their joint effect in the word GRAPHICS, however, is quite simple. Any Forth system sold for the ATARI will probably have words for opening the screen for graphics. Simply use whatever your system provides to define your own GRAPHICS, which takes one number from the stack and opens the screen in that mode, with a text window at the bottom.

The last four words specific to pns-Forth are CL#, COLOR, PLOT, and DRAWTO. These are used by LINE, FRAME, and POSITION. The first two are simple to define; just use

```
0 VARIABLE CL#
and
: COLOR  DUP CL# ! PAD C! ;
```

CL# is a variable which is used to keep track of the color data used to plot a pixel. COLOR takes a number from the stack and stores it both in

CL# and at PAD, for later use by PLOT and DRAWTO. The definitions of PLOT and DRAWTO are complicated because these words result in calls to CIO. Again, however, their functions are simple and your system probably provides similar words. Define a PLOT which takes a column and a row number from the stack, moves the screen cursor to that position, and plots a pixel there using whatever byte is at PAD as the color data. Similarly, define a DRAWTO which takes a column and a row number from the stack, and draws a line from the current position of the screen cursor to this specified position, using the byte at PAD as color data.

I believe that all the other words I've used in this system are either standard fig-Forth words or new words that I've defined.

Glossary of Turtle Commands

MODE Commands

SET MODE [mode ---] Opens the screen in the Graphics Mode specified by mode , which should be 3–8. Sets up a default viewport, window, and step size by executing WHOLE-SCREEN SET VIEWPORT and 100 PER-PIXEL. Draws a frame around the viewport with ink of type 1. Initializes the turtles by executing START.

MODE [--- mode] Leaves the number of the current Graphics Mode on the stack.

SHOW MODE [---] Displays a message indicating the current Graphics Mode.

ACTIVE Commands

SET ACTIVE [turtle# ---] Makes the turtle whose number is turtle# the active turtle. Future commands will be directed to her.

ACTIVE [--- turtle#] Leaves the number of the active turtle on the stack.

SHOW ACTIVE [---] Displays a message indicating the currently active turtle.

FORTH FACTORY

PEN Commands

SET PEN [state ---] Lowers the active turtle's pen if state is nonzero and raises it if state is zero.

PEN [--- state] Leaves 1 on the stack if the active turtle's pen is down and 0 if it is up.

SHOW PEN [---] Displays a message indicating whether the active turtle's pen is up or down.

INK Commands

SET INK [ink#] Fills the active turtle's pen with ink of type ink# . Type 0 ink is erasing ink. Types 1, 2, and 3 are colored. Types 2 and 3 are not available in modes 4, 6, or 8.

INK [--- ink#] Leaves on the stack the type of ink in the active turtle's pen.

SHOW INK [---] Displays a message indicating the type of ink in the active turtle's pen.

HEADING Commands

SET HEADING [degrees ---] Makes the active turtle head in the direction specified by degrees . Directions are measured clockwise from the vertical.

HEADING

[--- degrees] Leaves the active turtle's heading on the stack.

SHOW HEADING [---] Displays a message indicating the active turtle's heading.

X Commands

SET X [x ---] Changes the active turtle's X coordinate to x . Draws a line if her pen is down.

X [--- x] Leaves the active turtle's X coordinate on the stack.

SHOW X [---] Displays a message indicating the active turtle's X coordinate.

Y Commands

Similar to X Commands.

POSITION Commands

SET POSITION [*x* *y* *---*] Changes the active turtle's coordinates to $X = x$ and $Y = y$. Draws a line if her pen is down.

POSITION [*---* *x* *y*] Leaves the active turtle's X and Y coordinates on the stack.

SHOW POSITION [*---*] Displays a message indicating the active turtle's X and Y coordinates.

SIZE Commands

SET SIZE [*distance* *steps* *---*] Sets the step size so that the number of steps given by *steps* will cover a distance in XY coordinates given by *distance*.

SIZE [*---* *distance* *steps*] Leaves the current size parameters on the stack.

SHOW SIZE [*---*] Displays a message indicating the current step size.

WINDOW Commands

SET WINDOW [*xmin* *xmax* *ymin* *ymax* *---*] Sets the window to be the region from $X = xmin$ to $X = xmax$ and from $Y = ymin$ to $Y = ymax$.

WINDOW [*---* *xmin* *xmax* *ymin* *ymax*] Leaves the current window parameters on the stack.

SHOW WINDOW [*---*] Displays a message indicating the current window.

VIEWPORT Commands

SET VIEWPORT [*left* *right* *top* *bottom* *---*] Sets the viewport to extend from screen column *left* to screen column *right* and from screen row *top* to screen row *bottom*.

FORTH FACTORY

WHOLE-SCREEN SET VIEWPORT [---] Sets the viewport to extend from column 1 to the next to the last column and from row 1 to the next to the last row.

VIEWPORT [--- left right top bottom] Leaves the current viewport parameters on the stack.

SHOW VIEWPORT [---] Displays a message indicating the current viewport.

Other Commands

CLEAR [---] Clears the graphics screen without affecting the turtles.

FRAME [ink# ---] Draw a frame around the viewport, using ink of type ink# .

HOME [---] Moves the active turtle to X=0 and Y=0 with heading 0, without drawing a line, and then lowers her pen.

START [---] HOMEs all the turtles first. Then fills their pens with ink. (In Mode 3, 5, or 7, the Nth turtle's pen is filled with ink of type N. In Mode 2, 4, or 6, turtle 0's pen is filled with type 0 ink while the pens of turtles 1, 2, and 3 are filled with type 1 ink, the only colored ink available in these modes.) Finally, makes turtle 1 the active turtle.

NEW [---] Clears the screen, draws a frame with type 1 ink, and initializes the turtles by executing START.

PER-PIXEL [distance ---] Sets the window so that the point X=0, Y=0 is the center of the viewport, and so that the distance in XY coordinates given by distance will be the size of one pixel. Also, sets the step size so that each step is distance units long.

FORWARD [steps ---] Moves the active turtle forward the number of steps specified by steps . The movement is in the direction she is currently heading if steps is positive and in the opposite direction if steps is negative. The turtle's heading is unaffected. A line is drawn if her pen is down.

BACKWARD [steps ---] Like FORWARD except in the opposite direction.

DRAW [steps ---] Lowers the active turtle's pen so that a line will definitely be drawn as she moves forward the number of steps given by steps . Then her pen is returned to its previous state.

MOVE [steps ---] Raises the active turtle's pen so that a line will definitely not be drawn as she moves forward the number of steps given by steps . Then her pen is returned to its previous state.

RIGHT [degrees ---] Turns the active turtle the specified number of degrees, to the right if degrees is positive and to the left if negative.

LEFT [degrees ---] Like RIGHT except in the opposite direction.

TURN [degrees ---] The same as RIGHT.

GOTO [x y ---] The same as SET POSITION.

DRAWTO [x y ---] Lowers the active turtle's pen so that a line will definitely be drawn as she moves to $X = x$ and $Y = y$. Then her pen is returned to its previous state.

MOVETO [x y ---] Raises the active turtle's pen so that a line will definitely not be drawn as she moves to $X = x$ and $Y = y$. Then her pen is returned to its previous state.

TURNTO [degrees ---] The same as SET HEADING.

PENDOWN [---] Lowers the active turtle's pen: This is the same as 1 SET PENSTATE.

PENUP [---] Raises the active turtle's pen. This is the same as 0 SET PENSTATE.

PENDOWN? [--- flag] Leaves a 1 on the stack if the active turtle's pen is down and a 0 if it is up. This is the same as PEN.

PENUP? [--- flag] Leaves a 1 on the stack if the active turtle's pen is up and a 0 if it is down. This is the opposite of PEN.

ERASING [---] Fills the active turtle's pen with type 0 ink (the erasing type). This is the same as 0 SET INK.

(. . .) [#loops ---] Executes the words between the left parenthesis and the right parenthesis the number of times given by #loops .

DEFINE . . . AS . . . END Defines the word between DEFINE and AS to be a new turtle command which will execute the words between AS and END.

BYE [---] Leaves Turtleland and returns to pns-Forth.

by Gordon Smith

```
(      Turtle Graphics I, screen 1      )
DECIMAL
```

```
TABLE SINES
```

```
0000 , 0175 , 0349 , 0523 , 0698 ,
0872 , 1045 , 1219 , 1392 , 1564 ,
1736 , 1908 , 2079 , 2250 , 2419 ,
2588 , 2756 , 2924 , 3090 , 3256 ,
3420 , 3584 , 3746 , 3907 , 4067 ,
4226 , 4384 , 4540 , 4695 , 4848 ,
5000 , 5150 , 5299 , 5446 , 5592 ,
5736 , 5878 , 6018 , 6157 , 6293 ,
6428 , 6561 , 6691 , 6820 , 6947 ,
7071 , 7193 , 7314 , 7431 , 7547 ,
7660 , 7771 , 7880 , 7986 , 8090 ,
8192 , 8290 , 8387 , 8480 , 8572 ,
8660 , 8746 , 8829 , 8910 , 8988 ,
9063 , 9135 , 9205 , 9272 , 9336 ,
9397 , 9455 , 9511 , 9563 , 9613 ,
9659 , 9703 , 9744 , 9781 , 9816 ,
9848 , 9877 , 9903 , 9925 , 9945 ,
9962 , 9976 , 9986 , 9994 , 9998 ,
10000 , -->
```

```
(      Turtle Graphics I, screen 2      )
```

```
: (SIN)      ( n1 --- n2 )
      DUP 90 > IF
      180 SWAP - THEN
      SINES ;
: SIN      ( n1 --- n2 )
      ( Returns 10000 times the sine )
      ( of n1 degrees. )
      360 MOD
      DUP 0 < IF
      360 + THEN
      DUP 180 > IF
      180 - (SIN) MINUS ELSE
      (SIN) THEN ;
: COS      ( n1 --- n2 )
      ( Returns 10000 times the cosine )
      ( of n1 degrees. )
      360 MOD 90 + SIN ;
```

```
-->
```

```
( 32 Turtle Graphics I, screen 3      )
```

```

)
: SIN* ( n1 n2 --- n3 )
  ( Returns n1 times the sine of )
  ( n2 degrees. )
  SIN 100000 */ ;
: COS* ( n1 n2 --- n3 )
  ( Returns n1 times the cosine of )
  ( n2 degrees. )
  COS 100000 */ ;
-->
( 33 Turtle Graphics I, screen 4a )
0 VARIABLE TO-FLAG
CODE TO ( --- )
  1 # LDA, TO-FLAG STA,
    NEXT JMP, END-CODE
: VALUE
                                0 CONSTANT
                                ;CODE
                                TO-FLAG LDA, 0- IF,
  2 # LDY, W )Y LDA, PHA,
  INY, W )Y LDA, PUSH JMP, ELSE,
    0 # LDA, TO-FLAG STA,
  BOT LDA, 2 # LDY, W )Y STA,
  BOT 1+ LDA, INY, W )Y STA,
                                POP JMP, THEN,
                                END-CODE
-->
( 34 Turtle Graphics I, screen 4b )
0 VARIABLE TO-FLAG
: TO
  1 TO-FLAG ! ;
: VALUE
  <BUILDS 0 ,
DOES> TO-FLAG @ IF
  0 TO-FLAG !
    ! ELSE
    @ THEN ;
-->
( Turtle Graphics I, screen 5 )
VOCABULARY CLIPPING IMMEDIATE
CLIPPING DEFINITIONS
VALUE LEFT VALUE TOP

```



```

VALUE RIGHT          VALUE BOTTOM
2 BASE !
: CODE      ( p --- n )
              0
              OVER TOP < IF
1000 + SWAP DROP ELSE
              SWAP BOTTOM > IF
                0100 + THEN
                  THEN
                OVER LEFT < IF
0001 + SWAP DROP ELSE
                SWAP RIGHT > IF
                  0010 + THEN
                    THEN ;

-->
( Turtle Graphics I, screen 6 )
VALUE X1
VALUE Y1
VALUE C1
VALUE X2
VALUE Y2
VALUE C2
VALUE C
: CLIP_X      ( n1 --- n2 )
              Y1 -
                X2 X1 -
                  Y2 Y1 -
                    */ X1 + ;
: CLIP_Y      ( n1 --- n2 )
              X1 -
                Y2 Y1 -
                  X2 X1 -
                    */ Y1 + ;

-->
( 37 Turtle Graphics I, screen 7 )
2 BASE !
: WHERE?      ( --- p )
              C 0001 AND IF
                LEFT LEFT CLIP_Y ELSE
                  C 0010 AND IF
                    RIGHT RIGHT CLIP_Y ELSE
                      C 0100 AND IF
                        BOTTOM CLIP_X BOTTOM ELSE

```

```

                                C 1000 AND IF
                                TOP CLIP_X TOP THEN
                                THEN
                                THEN
                                THEN ;

DECIMAL
: HERE! ( p --- )

                                C C1 = IF
                                TO Y1 TO X1 X1 Y1 CODE TO C1 ELSE
                                TO Y2 TO X2 X2 Y2 CODE TO C2 THEN
:
-->
( 38 Turtle Graphics I, screen 8 )
FORTH DEFINITIONS
: CLIP ( p1 p2 --- p1' p2' t )
      ( or )
      ( p1 p2 --- f )

                                CLIPPING
                                TO Y2 TO X2 X2 Y2 CODE TO C2
                                TO Y1 TO X1 X1 Y1 CODE TO C1
                                BEGIN
                                C1 C2 OR WHILE
                                C1 C2 AND IF
                                Ø ;S THEN
                                C1 IF
                                C1 TO C ELSE
                                C2 TO C THEN
                                WHERE? HERE! REPEAT
                                X1 Y1 X2 Y2 1 ;
-->
( 39 Turtle Graphics I, screen 9 )

: GRAPHICS ( n --- )
  ( Clears the screen and sets it up )

  ( for graphics mode n with a text )
  ( window. )
  >R SETUP S CLOSE S
  SPLIT-SCREEN R> BR. ;
: LINE ( p1 p2 --- )
  ( Displays whatever piece of the )
  ( line from p1 to p2 is within )
  ( the clipping window. )

```

```

        CL# @ COLOR
        CLIP IF
        PLOT DRAWTO THEN ;
;S

( Turtle Graphics II, screen 1
)
DECIMAL
: VALUES
    <BUILDS 0 DO
        0 , LOOP
    DOES> OVER + +
        TO-FLAG @ IF
    0 TO-FLAG ! ! ELSE
        @ THEN ;
VALUE PREFIX
: SET ( --- ) 2 TO PREFIX ;
: SHOW ( --- ) 4 TO PREFIX ;
: ROOT: ( --- )
    <BUILDS SMUDGE ]
    DOES> PREFIX + @ EXECUTE
        0 TO PREFIX ;
-->
( Turtle Graphics II, screen 2
)
4 CONSTANT #TURTLES
VALUE WHICH
( The number of the active turtle )
: ACTIVE! ( n --- ) TO WHICH ;
: .WHICH ( --- )
    ." Turtle #" WHICH . ;
: ACTIVE? ( --- )
    .WHICH ." is active " CR ;
ROOT: ACTIVE WHICH ACTIVE! ACTIVE? ;
-->
( Turtle Graphics II, screen 3
)
: MODE@ ( --- n ) 87 C@ ;
: MODE? ( --- )
    ." This is graphics mode "
        MODE@ . CR ;
TABLE MAX_COL# ( n1 --- n2 )
    39 , 19 , 19 , 39 , 79 , 79 ,

```

```

159 , 159 , 319 ,
TABLE MAX_ROW# ( n1 --- n2 )
19 , 19 , 9 , 19 , 39 , 39 ,
79 , 79 , 159 ,
: WHOLE-SCREEN ( --- n1 n2 n3 n4 )
1 MODE@ MAX_COL# 1-
1 MODE@ MAX_ROW# 1-
-->
( 44 Turtle Graphics II, screen 4
)
: VIEWPORT@ ( --- n1 n2 n3 n4 )
CLIPPING LEFT RIGHT TOP BOTTOM ;
: VIEWPORT? ( --- ) CLIPPING
." The viewport is from column "
LEFT . ." to " CR ." column "
RIGHT . ." and from row " TOP .
." to row " BOTTOM . CR ;
VALUE XMIN VALUE YMIN
VALUE XMAX VALUE YMAX
: WINDOW@ ( --- n1 n2 n3 n4 )
XMIN XMAX YMIN YMAX ;
: WINDOW? ( --- )
." The window is from X=" XMIN .
." to X=" XMAX . CR ." and from Y="
YMIN . ." to Y=" YMAX . CR
:
-->
( Turtle Graphics II, screen 5
)
VALUE ØCOL VALUE ØROW
: ORIGIN! ( --- ) CLIPPING
XMIN MINUS RIGHT LEFT -
XMAX XMIN - */ LEFT + TO ØCOL
YMAX MINUS TOP BOTTOM -
YMAX YMIN - */ TOP + TO ØROW ;
: VIEWPORT! ( n1 n2 n3 n4 --- )
CLIPPING
MODE@ MAX_ROW# MIN TO BOTTOM
Ø MAX TO TOP
MODE@ MAX_COL# MIN TO RIGHT
Ø MAX TO LEFT
ORIGIN! ;
: WINDOW! ( n1 n2 n3 n4 --- )

```

FORTH FACTORY

```
TO YMAX TO YMIN TO XMAX TO XMIN
ORIGIN! ;

-->
( Turtle Graphics II, screen 6
)
ROOT: VIEWPORT
VIEWPORT@ VIEWPORT! VIEWPORT? ;
ROOT: WINDOW
WINDOW@ WINDOW! WINDOW? ;
: LEFT- ( --- n )
CLIPPING LEFT 1- 0 MAX ;
: TOP- ( --- n )
CLIPPING TOP 1- 0 MAX ;
: RIGHT+ ( --- n ) CLIPPING
RIGHT 1+ MODE@ MAX_COL# MIN ;
: BOTTOM+ ( --- n ) CLIPPING
BOTTOM 1+ MODE@ MAX_ROW# MIN ;
: FRAME ( n --- ) COLOR
LEFT- TOP- PLOT
RIGHT+ TOP- DRAWTO
RIGHT+ BOTTOM+ DRAWTO
LEFT- BOTTOM+ DRAWTO
LEFT- TOP- DRAWTO ;

-->
( Turtle Graphics II, screen 7
)
#TURTLES VALUES PEN()
: PEN@ ( --- flag ) WHICH PEN() ;
: PENDOWN? ( --- flag ) PEN@ ;
: PENUP? ( --- flag ) PEN@ 0- ;
: PEN! ( flag --- )
0- 0- WHICH TO PEN() ;
: PENDOWN ( --- ) 1 PEN! ;
: PENUP ( --- ) 0 PEN! ;
: PEN? ( --- ) .WHICH
." has her pen " PEN@ IF
." down " ELSE
." up " THEN
CR ;
ROOT: PEN PEN@ PEN! PEN? ;

-->
( Turtle Graphics II, screen 8
)
```

TURTLE GRAPHICS

```
#TURTLES VALUES INK()
: INK@ ( --- n ) WHICH INK() :
: INK! ( n --- ) WHICH TO INK<()
;
: ERASING ( --- ) Ø INK! ;
: INK? ( --- )
  .WHICH ." is using ink #" INK@ . CR
;
ROOT: INK INK@ INK! INK? ;
-->
( Turtle Graphics II, screen 9
)

#TURTLES VALUES HEADING()
: HEADING@ ( --- n )
  WHICH HEADING() ;
: HEADING? ( --- ) .WHICH
  ." has heading " HEADING@ . CR ;
: HEADING' ( n --- )
  360 MOD WHICH TO HEADING() ;
: TURNTO ( n --- ) HEADING! ;
ROOT: HEADING
  HEADING@ HEADING! HEADING? ;
: TURN ( n --- )
  HEADING@ + HEADING! ;
: RIGHT ( n --- ) TURN ;
: LEFT ( n --- ) MINUS TURN ;
-->
( Turtle Graphics II, screen 10
)

#TURTLES VALUES X()
#TURTLES VALUES Y()
: X@ ( --- n ) WHICH X() ;
: Y@ ( --- n ) WHICH Y() ;
: X? ( --- )
  .WHICH ." is at X=" X@ . CR ;
: Y? ( --- )
  .WHICH ." is at Y=" Y@ . CR ;
: POSITION@ ( --- n1 n2 ) X@ Y@
: POSITION? ( --- ) .WHICH
  ." is at X=" X@ . ." and Y=" Y@ . CR
;
-->
( Turtle Graphics II, screen 11
```



```

)
: X->COL ( n1 --- n2 ) CLIPPING
  RIGHT LEFT - XMAX XMIN - */ ØCOL + ;
: Y->ROW ( n1 --- n2 ) CLIPPING
  TOP BOTTOM - YMAX YMIN - */ ØROW + ;

: SCALE ( n1 n2 --- n3 n4 )
  SWAP X->COL SWAP Y->ROW ;
: ?CONSOLE ( --- flag )
  53279 C@ 7 = NOT ;
: POSITION! ( n1 n2 --- )
  ?CONSOLE IF
    SP! CR ." ok" QUIT THEN
    PEN@ IF
      INK@ COLOR
    OVER OVER SCALE POSITION@ SCALE
      LINE THEN
    WHICH TO Y() WHICH TO X() ;
-->
( Turtle Graphics II, screen 12
)
: GOTO ( n1 n2 --- ) POSITION! ;
ROOT: POSITION
  POSITION@ POSITION! POSITION? ;
: X! ( n --- ) Y@ POSITION! ;
: Y! ( n --- ) X@ SWAP POSITION! ;
ROOT: X X@ X! X? ;
ROOT: Y Y@ Y! Y? ;
: MOVETO ( n1 n2 --- )
  PEN@ ROT ROT PENUP POSITION! PEN! ;
: DRAWTO ( n1 n2 --- )
  PEN@ ROT ROT PENDOWN POSITION! PEN!
;
-->
( Turtle Graphics II, screen 13
)
VALUE SIZE_N VALUE SIZE_D
: SIZE@ ( --- n1 n2 )
  SIZE_N SIZE_D ;
: SIZE* ( n1 --- n2 ) SIZE@ */ ;
: SIZE! ( n1 n2 --- )
  TO SIZE_D TO SIZE_N ;
: SIZE? ( --- )

```

```

        SIZE_D DUP . 1 = IF
        ." step is " ELSE
        ." steps are " THEN
        ." a distance of " SIZE_N . CR ;
ROOT: SIZE    SIZE@ SIZE! SIZE? ;
-->
(      Turtle Graphics II, screen 14
)
: VECTOR      ( n --- n1 n2 )
    DUP HEADING@ SIN*  X@ +
    SWAP HEADING@ COS*  Y@ + ;
: FORWARD      ( n --- )
    SIZE* VECTOR POSITION! ;
: BACKWARD      ( n --- ) MINUS FORWARD
;
: MOVE      ( n --- )
    PEN@ SWAP  PENUP FORWARD  PEN! ;
: DRAW      ( n --- )
    PEN@ SWAP  PENDOWN FORWARD  PEN! ;
-->
(      Turtle Graphics II, screen 15
)
: PER-PIXEL      ( n --- )
    CLIPPING >R
    RIGHT LEFT - 2 /
    DUP MINUS R *  SWAP 1+ R *
    BOTTOM TOP - 2 /
    DUP MINUS R *  SWAP 1+ R *
    SET WINDOW  R> 1 SET SIZE ;
( Make SURE you typed the >R and R> )
( in this correctly. )
: SCREEN-DEFAULTS      ( --- )
    WHOLE-SCREEN SET VIEWPORT
    100 PER-PIXEL ;
TABLE GR.BYTES      ( n1 --- n2 )
    960 , 400 , 200 , 200 , 400 ,
    800 , 1600 , 3200 , 6400 ,
: CLEAR      ( --- )
    88 @ MODE@ GR.BYTES  ERASE ;
-->
(      Turtle Graphics II, screen 16
)
: HOME      ( --- )

```



```

Ø Ø MOVETO Ø TURNTO PENDOWN ;
: START ( --- )
  *TURTLES Ø DO
    I SET ACTIVE HOME
    MODE@ 2 MOD IF
      I ELSE I Ø- Ø- THEN
        SET INK LOOP
        1 SET ACTIVE ;
: MODE! ( n --- )
  GRAPHICS SCREEN-DEFAULTS
    1 FRAME START ;
ROOT: MODE MODE@ MODE! MODE? ;
: NEW ( --- ) CLEAR 1 FRAME START
;
: BYE ( --- ) Ø GRAPHICS
  Ø 710 C! 68 712 C! ;
-->
( Turtle Graphics II, screen 17
)
: DEFINE [COMPILE] : ; IMMEDIATE
: AS ; IMMEDIATE
: END [COMPILE] ; ; IMMEDIATE
: \ ( ignores rest of line )
  IN @ C/L / 1+ C/L * IN ! ; IMMEDIATE
: (
  COMPILE Ø [COMPILE] DO ; IMMEDIATE
: ) [COMPILE] LOOP ; IMMEDIATE
: +) [COMPILE] +LOOP ; IMMEDIATE
;S

```


WHY BUY THIS BOOK?

If you have an ATARI computer, you need two main things:
INFORMATION about using it, and **PROGRAMS** to learn from and enjoy.

That's what ANTIC magazine gives you — strictly ATARI information and programs written by other ATARI owners, like yourself.

This book selects and reprints the **BEST** material from Volume One of ANTIC—The ATARI Resource, plus six new Bonus Games and several new articles.

Remember—
If you own an ATARI,
You should be reading ANTIC.

\$12.95

