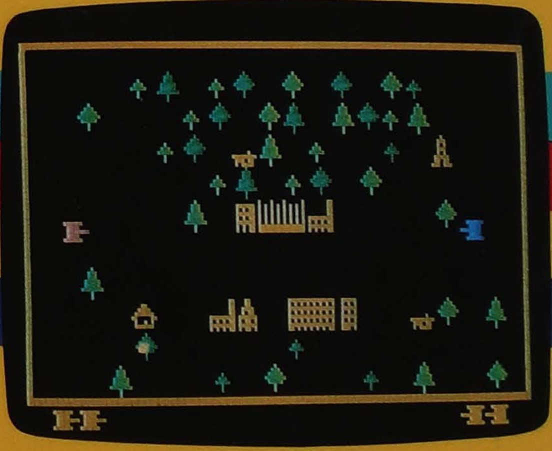
XL Compatible

COMPUTE!'s FIRST BOOK OF

ATARI GAMES

Fifteen games for Atari® computers, ready to type in and enjoy. Arcade games and learning games, including the best from **COMPUTE!** Magazine plus many never-before-published games and chapters on how to develop your own games.



A **COMPUTE! Books** Publication

$12.95

COMPUTE!'s FIRST BOOK OF

ATARI GAMES

COMPUTE!™ Publications, Inc. abc
A Subsidiary Of American Broadcasting Companies, Inc

The following articles were originally published in *COMPUTE!* magazine, copyright 1981, Small System Services, Inc.:
"Blockade" (August)
"Shoot" (September)

The following articles were originally published in *COMPUTE!* magazine, copyright 1982, Small System Services, Inc.:
"Word Hunt" (March)
"Programming Your First Game" (October)
"Tag" (October)
"MathMan" (October)
"Hidden Maze" (December)

The following articles were originally published in *COMPUTE!* magazine, copyright 1983, Small System Services, Inc.:
"Thunderbird" (January)
"Mastermaze" (February)
"Ski!" (February)
"Closeout" (March)

Printed in the United States of America

ISBN 0-942386-14-0

10 9 8 7 6 5 4 3 2

COMPUTE! Publications, Inc. Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC publishing companies, and is not associated with any manufacturer of personal computers. Atari is a trademark of Atari, Inc.

Contents

Foreword

When the first huge computers were built, games were not what the owners had in mind. Millions of dollars were invested in every machine. Computer time was valuable, and not to be wasted.

As computers shrank in size and increased in power, however, it was inevitable that weary programmers would begin exploring and programming, devising the forerunners of *Pac-Man* and *Donkey Kong*. Today, a vast number of the world's computers are built for one purpose only — to play games with whoever puts in a quarter.

Your Atari is not a dedicated game machine — it is much more versatile than that. But the Atari's designers knew that one of the most common uses of the machine would be play. Like the arcade machines, the Atari can give you experiences and entertainment that you could never find anywhere except in the worlds the computer can create.

This book serves a double purpose. First, it provides you with a variety of games which you merely type into the computer, save on tape or diskette, and then play again and again, as often as you like. Second, because the program is printed you can see exactly how the game's creator brought off the effects you like. It will be fairly easy for you to learn techniques that you can use in your own programs.

In fact, to make this book as useful as possible, many of the games are accompanied by explanations of how the program works. Chapters at the beginning and end of the book will also help you learn how to write your own games.

Much of the value of this book comes from its variety:

Besides being fun, MathMan and Word Hunt, for instance, are educational; Ski! can improve eye-hand coordination.

There are games that are simple and slow enough for small children. There are also games as fast and challenging as anything in the arcades.

No matter what level of programming skill you have reached, there will be programs for which you can learn techniques, ranging from fairly simple BASIC games to sophisticated all-machine language games like Shoot and Chiseler.

Even if you are a subscriber to *COMPUTE!* Magazine, there are things here you haven't seen before. One-third of the games in this book have never been published before and some of the others have been since refined and improved.

Some of the games here were originally programmed on other computers, and were "translated" for the Atari. Computer translation often requires as much creativity as the original program, since the requirements and features of computers can be very different.

Some games require more than simple translation: they require new coding in order to take advantage of the capabilities of the Atari. This is the case with three games in this book. Charles Brannon, of our editorial staff, has rewritten Tag, Ski! and Thunderbird for the Atari. In addition, E.H. Foerster has made important improvements to Brannon's version of Ski!.

Since the first printing of this book, Atari has introduced a new line of computers called the XL series. They've become very popular. We've ensured that all the games, save one, will run on any of the XL machines. That game, Ski, would have to be almost completely rewritten to run on an XL model computer. You can, however, play Ski on your 600 XL or 800 XL if you have the Atari BASIC cartridge. Simply plug it into the cartridge slot, type the program in, and play. Since the cartridge disables the built-in BASIC of the XL model computers, Ski will work.

All the rest of the games in this book can be typed in and played as is on an Atari XL computer.

Part One

The Atari As A Game Machine

Part One

Why the Atari Is a Great Game Machine

Orson Scott Card

When you push the cartridge into the slot, unbelievable things happen. Basketball players dribble and pass and shoot and steal. Spaceships go through complex maneuvers and blast asteroids out of the sky. Robots chase you through a maze. A little man climbs a ladder and puts out fires. Your computer has turned your TV screen into a universe of dazzling worlds, and it seems like whenever *you* aren't playing, your children or your parents are.

But you bought a computer, not just a game machine. And you bought an Atari, not just a computer. Which means that all the techniques that the wizard programmers used to create the games you like to play are all within your reach.

It probably won't surprise you that game designers like the Atari, too. That's because the computer was designed with many of the features that help programmers make their games run smoothly, with lots of graphic effects, and without a lot of extra, unnecessary programming steps.

The Atari Tool Kit

At their simplest level, computers consist of two basic parts, the central processing unit that performs operations, and the memory that holds all the instructions and data and results of those operations. The power of the computer is that it's very, very fast. The weakness is that it can still do only one thing at a time.

But there are ways of getting around that. The Atari computer saves a lot of time by actually having two processors: the 6502 handles the main business of whatever program is running, and the other, the ANTIC chip, handles what's going on on the TV screen. Once the main processor has told the ANTIC what to do, it can pretty well forget the screen and let the ANTIC go on telling the TV, every sixtieth of a second, what to put on the screen.

The Atari also has a sound system with four voices that plays through your TV speakers or your stereo system. There's a little speaker built into your console, too, that can be made to beep and click. And if you have a cassette or disk drive for permanent memory storage, the Atari has several ways of storing and reading information on both devices, some fast and complicated, some slow and simple.

That's the machinery, the hardware. Many of the most powerful features of your Atari as a game machine, however, are hidden in the computer's operating system and in its BASIC language. It isn't just the things that your computer can do — it's also the *way* you can get the computer to do them. It isn't quite as easy as wishing — this is the real world, after all. But with planning and a little study, you'll find that all those dazzling tricks the wizard programmers use aren't so miraculous after all. Or maybe that's the wrong way of looking at it. Maybe the tricks *are* miraculous — but the Atari makes it relatively easy for you to become a wizard.

Cutting Up the Memory Chain

To understand how these features work, it's important to understand how the Atari's memory is set up. Perhaps it's easiest to visualize computer memory as a long chain. The first link in the chain is memory location zero; the highest-numbered link your computer can read is location 65535.

Each link, or location, can hold one item of information. Your computer knows how to go from location to location and either read the item stored there or store a new item there. Whenever it stores something, it erases whatever was stored there before. However, when it reads something, the information at that location is unharmed — it can be read again and again without change.

What is stored at each location? The same thing every time: a number from 0 to 255. That's all — a long chain of memory locations, each holding a number from 0 to 255.

What matters, then, is what the computer *does* with that information.

It can interpret each number three ways:

Sometimes the computer reads those numbers as *instruc-*

tions, machine language commands that tell the 6502 processor what operations to perform.

Sometimes the computer reads the numbers as *numbers,* positive integers from 0 to 255 — or as positive and negative integers from -128 to +127.

And sometimes the computer reads the numbers as *addresses* — numbers which tell the computer where in memory *another* instruction or item of information is to be found.

How a number is interpreted depends on what you have told the computer to *do* with that information.

Screen Memory

Of great importance to arcade-type games is a long section of the memory chain that is used for screen memory. The Operating System (OS) tells ANTIC to look for screen memory starting at a certain address in memory. Then ANTIC interprets that screen memory according to certain predetermined patterns, called *graphic modes.*

Depending on the graphic mode your program is in, ANTIC will read screen memory as instructions to turn on or off little squares of color on the TV screen, or as instructions to display a certain character on the screen. When you type words on your computer, ANTIC doesn't care what you're saying. It only knows that at certain places in screen memory, the number stored in that location is the code for a certain letter.

So when alien invaders march back and forth across your screen, your 6502 is really just storing numbers in different memory locations, and ANTIC will make different patterns of color in different places on the screen.

Machine language is very fast when compared to BASIC. It can move large groups of numbers from one place to another in screen memory so quickly that it causes objects to move smoothly on the screen.

However, many computer users haven't yet learned how to use machine language. Instead, most of us rely on BASIC, a language a little closer to English, which, while a program is RUNning, translates our commands so the 6502 can understand them. The trouble is, this translation takes time. Our BASIC program often can't move numbers through screen memory quickly enough to make smooth movements on the screen.

Player-Missile Graphics

Here is where your Atari really starts to shine. ANTIC doesn't only look for screen memory — it can also look for another section of memory that holds up to five player shapes (or four player shapes and four narrower missile shapes), which can be moved across the screen, independent of what the rest of the screen memory is doing.

These player shapes can move much more quickly, and with a lot less calculation, than shapes in regular screen memory. They also are displayed on the screen without affecting the screen memory that is supposed to be displayed in the same place. This is why in "Tank Duel" you will see the tanks move right over trees — when the tank passes, the tree is still there, undisturbed.

You can also tell the computer which overlapping objects should have *priority* — does your airplane shape go in front of the cloud, or behind it, when they occupy the same place on the screen?

Your computer also notices what players or missiles are overlapping on the screen. That's how in *Asteroids* the game program notices whether you have crashed into a big rock or not, or whether another ship's missile has collided with your ship. If it doesn't matter, the program can ignore the collision; if it does matter, your program can go into its special effects routine and make an explosion.

The important thing to remember is that all of this activity is controlled by telling the 6502 what numbers to store in what locations. The Atari is designed so that you have almost complete control over all the numbers that matter. If you are working in machine language, you control all those numbers directly, but even in BASIC you can store numbers by using the POKE command, or read numbers by using the PEEK function. Some ways of moving the numbers around are faster than others. In these games, you'll see as many different techniques as there are programmers. But in the end, all the programmers are just putting numbers somewhere in the chain.

The Atari's power derives from the way it uses those numbers.

Part One

Changing Colors

Do you want to change the color of some object on the screen? You have 128 different combinations of color and brightness available to you — and to change a color, you merely have to POKE one number into one memory location. You can change the background, for instance, with the command POKE 712,66.

How many different colors can you display at once? Using the simplest graphics modes, you can show four colors at the same time. Add the five possible players, each with its own color, and there can be nine colors showing, each controlled by a single POKE. And the advanced graphics modes — 9, 10, and 11 — allow even more.

Sound

Your computer has four voices through the TV speaker. Four notes can sound at the same time — or two can be used together to make a combined sound. From BASIC, you can choose eight different distortion levels, 16 different loudness settings, and a wide range of pitches. Let's explore a bit.

Here's the sound of the sea:

```
10 FOR I=4 TO 14 STEP 2:SOUND 0,76-I
   ,8,I:FOR X=0 TO 150*RND(1):NEXT X
   :NEXT I
20 FOR I=14 TO 4 STEP -1:SOUND 0,60,
   8,I:FOR X=0 TO RND(1)*800:NEXT X:
   NEXT I:GOTO 10
30 FOR I=12 TO 4 STEP -1:SOUND 0,60,
   8,I:FOR X=0 TO RND(1)*800:NEXT X:
   NEXT I:GOTO 20
```

And here's the tick-tock of a clock:

```
5 FOR X=0 TO 1
10 SOUND 0,50+10*X,10,8:SOUND 0,0,0,
   0
20 FOR I=0 TO 150:NEXT I
30 NEXT X:GOTO 5
```

Program 1-1 is a simple program that will allow you to create sounds and modify the pitch and distortion while the sound is going on. If you have two pairs of paddles, you can work with two sounds at once.

Paddles One and Two control the first voice. Paddle One sets the pitch of the sound. Hold down the button, and the voice goes on and off in a staccato pattern.

Paddle Two controls the distortion level. Hold down the button, and the screen will report, over and over, what the number of the *pitch* is and what the number of the *distortion* is for the sound you're hearing.

If you find a sound you like, push the button, get the numbers, write them down, and then use them in a SOUND statement in a program. The SOUND statement looks like this:

SOUND 0,76,10,10

The first number is the voice number. The four voices are 0, 1, 2, and 3. The second number is the pitch number, ranging from 0 to 255. The third number is the distortion, which is always an even number from 0 to 14. The fourth number is the volume, any number from 0 (silent) to 15 (loud).

You can reproduce the sound you want by putting the pitch and distortion number shown on the screen in the right place in a SOUND statement in your own program.

Machine language programmers can even make their music or sound effects take place during *interrupt* time, when the TV screen isn't being actively displayed — but that's much too fast for BASIC to take advantage of.

Program 1-1. Sound

```
10 DIM PITCH(3),DISTORT(3),J(3):? "
   {CLEAR}"
20 ? "How many voices?  (1, 2, 3, or
    4)":OPEN #1,4,0,"K:"
30 GET #1,A:IF A<49 OR A>52 THEN 30
40 CLOSE #1:PRINT CHR$(A):A=A-49
50 ? "Paddles or Joysticks":OPEN #1,
   4,0,"K:"
60 GET #1,N:IF N<>74 AND N<>80 AND N
   <>106 AND N<>112 THEN 60
70 PRINT CHR$(N):CLOSE #1:ON N=74 OR
    N=106 GOTO 200
```

```
80 REM JOYSTICK-ONLY USERS MAY DELET
   E LINES 50 TO 190
100 FOR I=0 TO A:PITCH(I)=27+PEEK(62
    4+I*2):DISTORT(I)=2*INT(PEEK(625
    +I*2)/29):NEXT I
110 FOR I=0 TO A:SOUND I,PITCH(I),DI
    STORT(I),8:NEXT I
120 FOR I=0 TO A:ON  NOT PTRIG(I*2)
    GOSUB 150:ON  NOT PTRIG(I*2+1) G
    OSUB 140:NEXT I
130 GOTO 100
140 POSITION 1,I+1:? "Voice ";I,"pit
    ch ";PITCH(I);"{3 SPACES}distort
    ion ";DISTORT(I);"  ":RETURN
150 SOUND I,0,0,0:RETURN
190 REM PADDLE-ONLY USERS MAY DELETE
     LINE 50 TO 70 AND 200 TO 290
200 FOR I=0 TO A:PITCH(I)=100:DISTOR
    T(I)=10:SOUND I,PITCH(I),DISTORT
    (I),8:NEXT I
205 J(I)=J(I)-10:IF J(I)<0 OR J(I)>4
     THEN J(I)=0
210 FOR I=0 TO A:J(I)=STICK(I):IF J(
    I)=7 THEN J(I)=12
220 J(I)=J(I)-10:IF J(I)<0 OR J(I)>4
     THEN J(I)=0
230 ON J(I) GOSUB 240,250,260,270:ON
      NOT STRIG(I) GOSUB 290:NEXT I:
    ON PEEK(53279)=6 GOSUB 280:GOTO
    210
240 DISTORT(I)=DISTORT(I)-2:DISTORT(
    I)=DISTORT(I)+14*(DISTORT(I)<0):
    SOUND I,PITCH(I),DISTORT(I),8:RE
    TURN
250 DISTORT(I)=DISTORT(I)+2:DISTORT(
    I)=DISTORT(I)-14*(DISTORT(I)>14)
    :SOUND I,PITCH(I),DISTORT(I),8:R
    ETURN
260 PITCH(I)=PITCH(I)+1:PITCH(I)=PIT
    CH(I)-256*(PITCH(I)>255):SOUND I
    ,PITCH(I),DISTORT(I),8:RETURN
270 PITCH(I)=PITCH(I)-1:PITCH(I)=PIT
```

```
    CH(I)+256*(PITCH(I)<0):SOUND I,P
    ITCH(I),DISTORT(I),8:RETURN
280 POSITION 2,1:FOR I=0 TO A:? "Voi
    ce ";I;"{3 SPACES}Pitch ";PITCH(
    I);"{3 SPACES}Distortion ";DISTO
    RT(I);"   ":NEXT I:RETURN
290 SOUND I,PITCH(I),DISTORT(I),8:SO
    UND I,0,0,0:RETURN
```

Machine Language Subroutines

One of the tricks the Atari can play is changing the meaning of numbers in the middle of a program. For instance, several of the programs in this book will have statements that look like this:

A$ ="hhh*VLd"

And then, later, a statement that looks like this:

X =USR(ADR(A$))

What the first statement does is store a group of numbers as a *string* of characters. Ordinarily, the computer would interpret those numbers from then on as the letters or symbols shown.

But the second statement tells the computer to go to the address where those letters are stored — ADR(A$) — and interpret those numbers as machine language instructions — X =USR. What looked like a string of characters is really a subroutine for loading data from a disk file, very, very quickly.

There are other ways to get short machine language routines into game programs. What is important is that it allows programmers to write most of a program in BASIC, the language most programmers know best — and only write machine language in the sections of the program that really require speed.

Some of the programs in this book are entirely in machine language. You can tell which ones they are — they're the ones that consist almost entirely of lines like this:

DATA 15,233,0,55,99,5,23,120,120,0,0,0,40

Long, slow typing — but very fast games.

Most programs, though, are in BASIC, and even without a lot of training, you can follow most of what's going on. And because of those embedded machine language subroutines, most of those BASIC games play just as fast as you could ever want.

Additional Features

There are other features we've hardly touched on:

• Six different ways of displaying characters, which you can define into many different shapes and colors, ranging from the simple small white letters and numbers that the machine usually uses to the multicolored characters in games like "Tank Duel." As far as the computer knows, it's just displaying letters on the screen — but you see trees and buildings.

• Full joystick and paddle control, which you can use with one-word functions like STICK(0), PADDLE(5), or, for the buttons, STRIG(3) and PTRIG(7). The STICK functions give you one of eight directions; the PADDLE functions give you a number from 0 to 255. Notice how those are used in the sound demonstration program you used a minute ago.

• A sound track on the cassette, alongside the data track, that allows you to record music or speech and play it back, all controlled and timed from within a program. Several excellent games and educational programs use this feature. (Unfortunately, it's one of the few things we just can't do in a book of printed programs like this.)

• Three function buttons, OPTION, SELECT, and START, can be used however you wish in a program, simply by seeing what's in memory location 53279, like this:

```
FOR I=0 TO 10000:A=PEEK(53279):PRINT A:NEXT I
```

If you type in that line and then press those three buttons, one at a time or in combination, you will see what numbers the computer automatically stores, ready for you to read within a program.

• Scrolling, both up and down and side to side, so that the television screen can seem to be a window looking onto a much larger playfield. That's the technique used in "Ski!," which allows you to ski down a hill much larger than the TV screen could ever show all at once.

All of these features were carefully built into the Atari — this computer was designed to be a powerful game machine.

But much of its value comes from the simple fact that it is a computer. Playing word games, doing mathematical calculations, designing mazes, remembering intricate patterns — these are all things that computers do very well. They were originally developed to do vast amounts of tedious work very quickly.

It's just a bonus that they can be such a lot of fun.

Writing Your First Game

Richard Mansfield

Richard Mansfield, Senior Editor of COMPUTE! Publications, explains the details of a simple game. A beginning programmer can learn a great deal studying this short program.

If you are tempted to write your own games, go ahead. It's a good way to learn to program. Games are basically the same as any other kind of programming.

Computer games fall into two broad categories: imitations of old standards (checkers, Othello), and games which could not be played without a computer (*Space Invaders, Pac-Man*). This second category is more difficult to program for several reasons. For one thing, you've got to think up a new and entertaining concept and then adjust the action until it is just hard enough to be challenging but not so difficult that people want to give up.

This category (basically ''arcade'' games) is especially hard to program precisely because a good computer-only game exploits all of the computer's special attributes: speed, color, sound. To do this well, to make things look and respond just the way you imagine them, requires a good bit of programming experience. Usually, too, several things are happening *at once* in an arcade game. This often means that such a program must be written in machine language, which is far faster than BASIC.

High Card Slice

Old standards, on the other hand, can often be the best way to get started programming games. You already know the game concept, and cards or dice or game boards are fairly easily constructed and manipulated on your computer screen. To illustrate, let's take a look at a simple simulation of one of the oldest card games, High Card. The rules are simple: you place a bet, and then you draw a card from the deck. The computer, your opponent, draws a card too, and the highest card wins the money.

One simplification here is that there is no attempt to represent the cards on the screen. The entire game relies simply on words (''Ace of Spades,'' for example) when cards are drawn.

Like most computer programs, the program can be visualized as having four distinct zones: initialization, main loop, subroutines, data tables. We can go through the steps in programming this game by looking at each zone separately.

Initialization

From lines 10 through 80 we are ''teaching'' the computer some basics about this game. Initialization is the activity which must take place before any of the action can begin. Computers are so fast that they will zip up through these lines and start things off in the main loop at line 100 in a flash. However, as programmers, we are aware that several preliminary events took place inside before anything else.

In line 20, the computer discovers that there is a variable called ''dollars'' which is equal to 500. It sets aside a section (like a small box) in its memory which it labels ''dollars.'' When the game is running, it will add or subtract from this ''box'' (lines 230-240) to keep a running total of how much money you have left to bet. From time to time (line 110), it will check the box and report to the player how much he has. The box labelled ''dollars'' is called a *variable* because during the game the amount in it will vary.

Lines 30 through 60 are simple enough — they ask the player to give his or her name. The computer ''memorizes'' it in another ''box'' called ''name$'' and can now speak more personally to the player in lines 140 and 230. Also, the computer prints the rules of the game in line 60.

Line 70 ''reads'' four names (the face cards) from the data tables in lines 510 on. It also makes a ''mental note'' that it has already READ four items. So, when it's asked to READ again (line 80), it will start with the next unread item of data, which will be ''clubs.'' By now, the computer has ''memorized'' a variety of important facts: the player's name, the amount of his or her betting purse, the names of the face cards, and the suits of a standard deck. In less than a second, the computer has grasped and filed away the necessary facts to go on to the main loop where all the action takes place.

The Main Loop

After checking that the player has money to bet, the computer asks for the bet, checks again that the bet is possible, and then runs through one cycle of the game starting in line 160. At this point, a programmer might find it worthwhile to visualize the steps involved in the game:

1. Draw a card for the player
2. Draw for the computer
3. Decide who won
4. Adjust the player's purse

Since both draws are essentially identical actions (the only difference will be that we say ''Bob draws a...'' instead of ''The computer draws''), we don't need to program the draw twice. This is where subroutines come in handy.

The Subroutine

Twice in the main loop, we GOSUB 300. First the player draws, then the computer draws. Line 310 randomly picks two numbers, the card and the suit. If line 320 finds that this selection matches the one drawn just before by the player, it goes back for another draw. Line 330 makes the *name* of the card be the number if it wasn't a number higher than 10 (a face card).

Then line 340 announces the draw using three variables. The first variable (player$) is set up in either line 160 or 190 as appropriate. Then the card$ and suit$ variables are selected from the lists that were "memorized" back in the initialization phase (lines 70-80). The subroutine then RETURNs to the main loop.

Lines 210-240 decide and announce the winner of this round. First, if the variable ''card'' (the computer's card) is greater than (>) ''yourcard,'' the computer is declared the winner in line 240, the purse is adjusted, and the main loop is restarted (GOTO 100). If the cards are equal, nothing happens to the purse and the next round begins. Notice that we don't need to say ''IF YOURCARD > CARD'' at the start of line 230 to test if the player has won. It's the only possible thing if the computer has gotten this far.

Once you've solved a particular problem, you'll find you can use the solution in many future games. This subroutine which draws cards, for instance, would work just as well for poker, or blackjack, or dozens of other games. Subroutines are handy not only because they can be used repeatedly within a program, but because they can be saved and used repeatedly in future programs. So think up a simple, traditional game and teach it to your computer. There is probably no more pleasurable way to learn programming than to write a game.

Program 1-2. High Card Slice

```
10 REM *NECESSARY INITIAL INFORMATION
   *
20 DOLLARS=500:DIM NAME$(20),PLAYER$(
   20)
30 GRAPHICS 0:PRINT " WITH WHOM DO I
   HAVE THE PLEASURE"
40 PRINT " OF PLAYING HIGH CARD SLICE
   ?"
50 INPUT NAME$
60 PRINT " HIGH CARD WINS IN THIS GAM
   E!"
70 DIM CARD$(14*5),T$(10):FOR I=11 TO
    14:READ T$:CARD$(I*5-4,I*5)=T$:NE
   XT I
80 DIM SUIT$(8*4):FOR I=1 TO 4:READ T
   $:SUIT$(I*8-7,I*8)=T$:NEXT I
90 REM
100 REM **MAIN PROGRAM LOOP**
110 PRINT :PRINT " YOU HAVE $";DOLLAR
    S
120 IF DOLLARS<=0 THEN PRINT " THE GA
    ME IS OVER.{21 SPACES}YOU ARE OUT
     OF CASH.":END
130 PRINT " WHAT IS YOUR BET";:INPUT
    BET
140 IF DOLLARS<BET THEN PRINT " YOU O
```

```
    NLY HAVE $";DOLLARS;" TO BET",NAM
    E$:GOTO 130
150 YOURCARD=0:YURSUIT=0
160 PLAYER$=NAME$
170 GOSUB 300
180 YOURCARD=CARD:YURSUIT=SUIT
190 PLAYER$="THE COMPUTER"
200 GOSUB 300
210 IF CARD>YOURCARD THEN GOTO 240
220 IF CARD=YOURCARD THEN PRINT " A T
    IE!":GOTO 100
230 PRINT NAME$;" WINS":DOLLARS=DOLLA
    RS+BET:GOTO 100
240 PRINT " THE COMPUTER WINS":DOLLAR
    S=DOLLARS-BET:GOTO 100
290 REM
300 REM *SUBROUTINE TO DRAW THE CARDS
    *
310 CARD=INT(RND(0)*13)+2:SUIT=INT(RN
    D(0)*4)+1
320 IF CARD=YOURCARD AND SUIT=YURSUIT
     THEN 300:REM NO IDENTICAL DRAWS
330 IF CARD<11 THEN T$=STR$(CARD):GOT
    O 340
335 T$=CARD$(CARD*5-4,CARD*5)
340 PRINT " ";PLAYER$;" DRAWS THE ":P
    RINT " ";T$;" OF ";SUIT$(8*SUIT-7
    ,SUIT*8):PRINT
350 RETURN
490 REM
500 REM **DATA TABLE**
510 DATA JACK ,QUEEN,KING ,  ACE
520 DATA CLUBS{3 SPACES},DIAMONDS,HEA
    RTS  ,  SPADES
```

Part Two

Maze Games

Part Two

MASTERMAZE
Mazing in Three Dimensions

"Mastermaze," an extremely challenging game, uses page flipping and machine language to create up to 32 levels. This revised version of Mastermaze will create 27 mazes in less than one minute.

Almost everyone finds mazes an enjoyable challenge. If you are like me, however, you feel that mazes take only minutes to solve and can soon become monotonous. That is why I chose to use my personal computer to its fullest, having it perform functions impractical with paper and pencil. This three-dimensional maze game is the result.

One Level at a Time

First, let me explain how to use the program. Since it is impractical and nearly impossible to display an entire three-dimensional maze at one time, the program displays only the level that the player is on, which is really of no consequence to the user, but makes life a lot easier for the programmer. What we are doing is analogous to a book: instead of showing the entire book in one screen, we are displaying only one page at a time — the page that is being read.

After you have typed in the entire program, the first thing you must do is SAVE a copy to tape or disk. This program plays around with the display list, so typos could cause problems and possibly crash your computer.

Playing the Game

Once a copy has been SAVEd, type RUN and you will be prompted with the question "# OF LEVELS?" What the computer really wants to know is how deep you would like your maze to be. In other words, the computer wants you to tell it one more than the minimum number of down "tunnels" the user must pass through before he reaches the end. In terms of our book analogy, the computer is asking for the number of pages in the book.

For a first-time player, I suggest three or four levels at most. The minimum number of levels is one, and the maximum, for computers with 48K, is 32. The maximum number of levels will be less on computers with DOS present or less memory.

Once you have entered your desired number of levels, the program will ask "INVISIBLE (1) OR VISIBLE (2)?" If you try invisibility, beware. Although the screen appears to be blank, the walls to the maze are still there.

Now that the program has the necessary data, the computer begins to build the maze to your specifications. Before work actually begins, the screen informs you of the work to be done. After this short delay, the screen is turned off and the maze is constructed. The actual time needed to construct the mazes is relatively fast thanks to a machine language routine written by David Butler.

Once the computer has completed construction of the maze, the screen is turned back on, and you are asked to PRESS START TO BEGIN. Watch the word START closely. See how it is flashing on and off? This effect is produced by toggling CHACT (location 755 or hex 2F3) in rapid succession (alternately POKEing in one and two). You are asked to press START when you are ready because the program times you, and it would not be fair to start timing from the moment the maze was completed.

Therefore, when you are ready to begin, you press START, which tells the program that you are poised with joystick in hand; the top level is displayed and timing begins. You will see an S in the upper left corner of the screen, with the ball character (control-T) underneath. *You* are the ball character.

Threading the Tunnels

Just move the joystick in the direction you want to go. "Sure," you say, "but where *do* I want to go?" Simple enough. If you chose a one-level maze (chicken), you will see an F at the lower right corner of the maze. That's where you want to go. If you were gutsy, however, and chose any number of levels greater than one, you will see five graphics " + " characters at random points throughout the top level. These symbols represent tunnels, through which you must pass to reach the finish (which is always in the lower right of the bottom level of the maze). As you might have guessed, you always start at the upper left of the top level.

To pass through a tunnel, simply move onto the " +" symbol and press the fire button. Voilà! The new level is displayed instantly. Have you gone up or down? Well, if you were on the top level, the only place you could go is down. If you are in the middle of a maze of four or more levels, then I have absolutely no idea which direction you'll go; you may pass through the same level three or four times before you realize that you've gone nowhere.

In mazes of ten or more levels, be prepared to see the same level a few times before you make any progress. No matter how many levels you choose, however, the goal is still the same. You must try to go down to unexplored levels; if you end up on a level you have been on already, you have looped, and you must figure out whether you've gone up or down.

In any case, find the F on the lowest level, go to the space directly above it, and move down. If you do not push the joystick down, the timer will continue, and your record time will be lost. When the timer has stopped, you will hear five beeps.

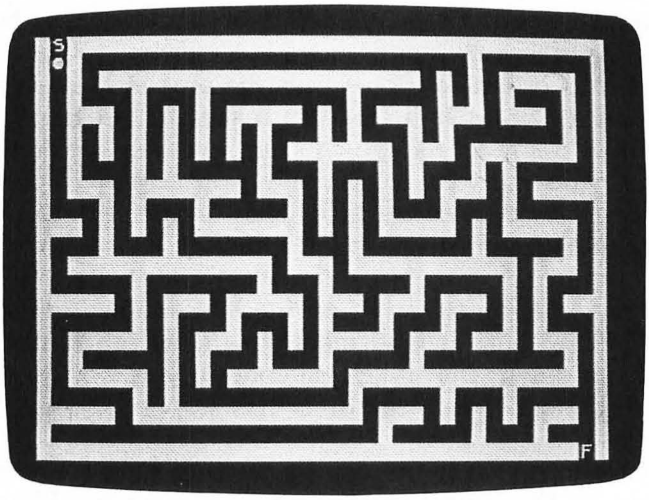
If you do not hear the five beeps, you have not stopped the timer, or the sound is gone on your machine. Either way, just remember to go down when you reach the finish — as you get better and better, times will get tougher and tougher to beat, and each second will become important.

That's all there is to it. After the five beeps have informed you that the timer has stopped, the screen will become visible (no change for visible mazers), and the time used to complete the maze will be displayed in hours:minutes:seconds format. The program will loop until you press the START button again, which will cause the program to re-RUN.

Possible Dead Ends

There are a few caveats, however. First, if you are attempting an invisible maze, some joystick directions may not work. There is nothing wrong with the program; if you cannot move in a certain direction, you have hit a wall (I told you they were still there). Second, don't even try to do deep invisible mazes without the consent of your psychologist. Third, each tunnel can be used only once, so make your moves wisely.

Last, and most important, don't *ever* remove lines 14 and 15. This program, as mentioned earlier, will cause the computer to do some strange things if you hit the BREAK key. Lines 14 and 15 turn off the BREAK key; the only way to get out of the program is to hit the SYSTEM RESET button.



An example of a one-level maze.

The Program

Now let's look at how the program accomplishes what it does. Line 8 is self-explanatory. Line 10 resets the screen and sets the variable TOP to the address of the LSB of the screen memory address. By POKEing different numbers into TOP and TOP+1, we can display any area of memory. Line 12 stores the value of SAVMSC (locations 88 and 89, 58 and 59 hex) into RL and RH, respectively. This step is necessary to reset the destination of PRINT statements after these locations have been modified by the maze generator routine.

Lines 20 and 23 obtain the required data from the user and determine the value of BOT, the page number of the lowest memory address to be used. Line 25 makes sure that we haven't used up all available memory, and informs the user of any memory conflict. Line 26 lets the user know that the delay which will follow is intentional, not something gone wrong with the program. Line 27 loads the machine language maze generator program from the DATA statements (lines 1000-1070) into free RAM at page six. The routine, written by David H. Butler, is a modification of Charles Bond's original maze generator in BASIC.

Line 28 turns off the screen and sets up the display for the start of the game. Line 29 clears memory using the CLEAR key. Line 30 establishes the top of maze memory and sets up a loop to construct each of the MAXLEV levels of the maze. Lines 60-80 set up the screen and call the maze generator routine.

Establishing Start and Finish

Line 120 restores the PRINT statement destination to its original value by POKEing RL and RH back into SAVMSC. Line 130 establishes the S in the upper left and the F in the lower right of the maze. Line 135 checks to see if any tunnels have to be built; in other words, if the maze is only one level, jump over the tunnel building routine (lines 140-170).

The tail end of line 170 restores the screen and sets up the console switches for reading. Line 172 executes a GOSUB to the routine that randomly sets the color of the background at the beginning and also each time the user passes through a tunnel. Line 173 loops indefinitely until the user presses the START button. This line is the one that toggles CHACT, as described earlier.

Line 174 makes the maze visible or invisible, based on your response to the second prompt at the beginning of the program. Line 175 resets the three-byte timer RTCLOK to zero. Line 180 determines the start position for the player and tells the display list where the first level of the maze is. Lines 185-321 are the main loop and should be self-explanatory.

A few notes, though: Line 190 reads the joystick and the trigger, lines 200-230 perform routine motion, line 235 checks for a win, line 240 checks for walls, and lines 300-321 change levels. Lines 400-415 stop the timer, sound the bell, and display the time used. Line 420 sets up the console switches for reading and POKEs a 124 into the attract mode flag ATRACT (location 77, hex 4D). The 124 in ATRACT gives the user approximately 16 seconds before the screen goes into attract mode.

Line 430 loops until the START button is pressed. Line 450 is the string A$ (we can't PRINT it because we've changed the screen memory locations). Don't forget to put the quotation marks towards the end of the line; doing that fools BASIC into reading trailing blanks to fill up A$. Finally, line 500 reads a random number from the random number generator RANDOM (location 53770, D20A hex), masks out the four low-order bits, and uses it to set the background color. If you're interested in the technical aspects of the game, read on. If not, RUN the program and have some fun.

Inner Secrets of Page Flipping

The programming tool behind the entire program is called page flipping. What this technique involves is changing the address that the ANTIC chip reads to determine the start of screen memory. This address is always in the display list, which is pointed to by SDLSTL and SDLSTH (locations 560 and 561, hex 230 and 231) in standard LSB, MSB order.

In the display list you will find all sorts of numbers; all have a meaning and should not be tampered with by the inexperienced programmer. In different graphics modes, the display list changes both in length and location.

In general, the display list follows two rules. First, all graphics modes accessible through BASIC have display lists that start with 112, 112, 112 in three successive bytes. These three bytes tell the ANTIC that there are to be 24 blank lines on the television screen.

Second, the fourth location of the display list contains a byte which has its sixth bit set. The rest of the byte varies depending on the graphics mode, but bit six is always set. Bit six, when set, tells the ANTIC chip that it is to begin direct memory access (DMA) at the location pointed to by the next two bytes. Therefore, any area in memory can be displayed by POKEing the address (LSB, MSB) into the location pointed to by SDLSTL and SDLSTH plus four.

This is the basis of this program. All screens are constructed before play begins, and, instead of drawing an entirely new screen, all the program does is change these addresses to point to the first byte of the new screen.

During the blank-out period at the start of the program, the entire maze is constructed, layer by layer, and the resulting mazes are stored in 1K decrements, starting with the last free kilobyte memory block before the display list. The maze generator routine does not even need to be modified for this purpose; all that was done was to change the PRINT destination pointer SAVMSC (location 88, hex 58, mentioned earlier). In other words, all I did was fool the maze generator routine into thinking that screen memory was located in middle area RAM (instead of the top), and since 960 bytes are needed for the standard GRAPHICS 0 screen, 1K blocks were very convenient.

The tunnels used this information both at construction time and at level-changing time. Random numbers were all that was necessary to build the tunnels; checks were required only to make sure that the tunnels would be within the maze and that they did not cut through maze walls. Since no other checks are made, it is possible to have many tunnels packed closely together.

The simple method of checking upward and downward movement causes tunnels to be disabled as they are used. When the player changes levels, a control-T character is left where the graphics plus symbol was previously. As a result, the checks for the graphics plus symbols will always fail on an already-used tunnel. This feature, added only to make the mazes more challenging, can easily be altered by changing the GOTO 185 in line 250 to GOTO 190.

This simple change makes the program think that you have just moved across or down (i.e., you have not changed levels). Therefore, the program replaces the previous space with the variable T, which contains the screen memory value of the space you were on before. When you move, the control-T is moved in the proper direction, and T is POKEd into the space you just moved from. It is confusing, but it works, and it works fast.

Tunnel Checking

The tunnels, when used, merely change the value of the sixth byte of the display list. Since 1K memory blocks are used, it is not necessary to change the fifth, LSB of the display list DMA address; it will always be zero. Either the sixth byte is added to four, or four is subtracted from it. The reason for this change should be evident — four pages constitute one kilobyte of memory.

Locating the mazes in this fashion greatly simplifies all checks. Instead of going through a series of different LSB, MSB checks to determine the location (two-dimensionally) of a space on two different levels, all that is required is a PEEK to the address plus 1024 (1K) and the address minus 1024. Again, this is how tunnel checking is done in lines 305 and 310.

Last, let's look at the timer. From the time the computer is powered up until the time it is powered down, the OS, as part of its stage one VBLANK (vertical blank) routine, increments the three-byte jiffy counter RTCLOK. RTCLOK is located in three consecutive bytes starting at address 18 decimal, 12 hex.

Unlike most of the system numbers, this clock is stored in MSB first, LSB last order. Since vertical blanks occur once every sixtieth of a second, this timer counts "jiffies" (sixtieths of a second). When the game start is pressed, zeros are POKEd into the clock addresses (line 175). As soon as the player has completed the maze, the locations are read and stored in the variable ET (for elapsed time). Simple mathematical manipulations derive the hours, minutes, and seconds and store them in the variables EH, EM, and ES, respectively.

That's all there is to it. Since we know that we started at zero, no other manipulations are needed. (Incidentally, it is possible to stop the clock, but doing so requires a shutdown of the entire system VBLANK routine, which can have disastrous effects on your program.)

Program 2-1. Mastermaze

```
1 REM ****MASTERMAZE****
2 REM **Mazing in Three*
3 REM ****Dimensions****
8 DIM A$(37):SW=0
10 GRAPHICS 0:TOP=PEEK(560)+256*PEEK(
   561)+4
12 RL=PEEK(88):RH=PEEK(89)
14 O=PEEK(16)-128:IF O<0 THEN O=O+128
15 POKE 16,O:POKE 53774,O
```

Part Two

```
20 ? :? "# OF LEVELS";:INPUT MAXLEV:M
   AXLEV=MAXLEV-1:? "INVISIBLE (1) OR
    VISIBLE (2)";:IF MAXLEV<Ø THEN MA
   XLEV=Ø
23 BOT=INT(TOP/256)-MAXLEV*4-4:INPUT
   INV
25 IF BOT*256<PEEK(144)+256*PEEK(145)
    THEN ? "***INSUFFICIENT MEMORY***
   ":GOTO 2Ø
26 POKE 752,1:? "{CLEAR}":POSITION 4,
   1Ø:? "CONSTRUCTING MAZE...WAIT FOR
    START"
27 RESTORE 1ØØØ:FOR I=1536 TO 169Ø:RE
   AD A:POKE I,A:NEXT I:POKE 755,1
28 POKE 559,Ø:? "{CLEAR}":POSITION 1Ø
   ,11:? "PRESS START TO BEGIN"
29 TM=PEEK(1Ø6):POKE 1Ø6,TM-6:POKE 88
   ,Ø:POKE 89,BOT:? "{CLEAR}":POKE 1Ø
   6,TM
3Ø R1=BOT+MAXLEV*4:FOR X=BOT TO R1 ST
   EP 4:POKE 77,Ø:POKE 88,Ø:POKE 89,X
6Ø SC=PEEK(88)+256*PEEK(89):A=SC+43
65 POSITION 2,Ø:POKE 752,1:FOR I=1 TO
    23:? "{37 SPACES}":NEXT I
7Ø POKE A,5
8Ø G=USR(1536,1675,A):GOSUB 5ØØ
12Ø NEXT X:POKE 88,RL:POKE 89,RH
13Ø POKE BOT*256+917,38:POKE R1*256+3
    ,51
135 IF MAXLEV=Ø THEN POKE 559,34:POKE
     53279,8:GOTO 172
14Ø FOR X=BOT TO R1-4 STEP 4:FOR Y=1
    TO 5
15Ø J=INT(RND(Ø)*876)+43
151 W=J-(INT(J/4Ø)*4Ø):IF W<3 OR W=39
     THEN 15Ø
155 IF PEEK(X*256+J)=Ø AND PEEK(X*256
    +1Ø24+J)=Ø THEN POKE X*256+J,83:P
    OKE X*256+1Ø24+J,83:GOTO 17Ø
```

```
160 GOTO 150
170 NEXT Y:NEXT X:POKE 559,34:POKE 53
    279,8
172 GOSUB 500
173 IF PEEK(53279)<>6 THEN POKE 755,-
    PEEK(755)+3:GOTO 173
174 POKE 755,INV
175 POKE 18,0:POKE 19,0:POKE 20,0
180 ST=R1*256+43:WIN=BOT*256+960:POKE
     TOP,0:POKE TOP+1,R1
185 S=PEEK(ST):T=ST:POKE ST,84
190 Q=STICK(0):R=STRIG(0):IF R=0 AND
    S=83 THEN 300
200 IF Q=7 THEN ST=ST+1
210 IF Q=11 THEN ST=ST-1
220 IF Q=14 THEN ST=ST-40
230 IF Q=13 THEN ST=ST+40
235 IF PEEK(ST)=38 THEN 400
240 IF PEEK(ST)=128 OR PEEK(ST)=51 TH
    EN ST=T
250 IF ST<>T THEN SW=0:POKE T,S:POKE
    77,0:GOTO 185
251 GOTO 190
300 IF SW=1 THEN 190
305 IF PEEK(ST+1024)=83 THEN R1=R1+4:
    ST=ST+1024:GOTO 320
310 IF PEEK(ST-1024)=83 THEN R1=R1-4:
    ST=ST-1024
320 IF R1<BOT OR R1>MAXLEV*4+BOT THEN
     330
321 POKE TOP+1,R1:SW=1:GOSUB 500:GOTO
     185
330 A=USR(58484)
400 ET=PEEK(18)*65536+PEEK(19)*256+PE
    EK(20):EH=INT(ET/216000):EM=INT((
    ET-EH*216000)/3600)
401 FOR X=1 TO 5:FOR Y=15 TO 0 STEP -
    0.2:SOUND 0,9,10,Y:NEXT Y:NEXT X:
    POKE 755,2
402 ES=INT((ET-EH*216000-EM*3600)/60)
```

```
403 ? "{CLEAR}":? :? "445 DATA ELAPSE
    D TIME: ";EH;":";EM;":";ES;"
    {19 SPACES}!"
404 ? "CONT":POSITION Ø,Ø:POKE 842,13
    :STOP
405 POKE 842,12
406 POSITION 2,15:RESTORE :FOR Y=Ø TO
     1
41Ø READ A$:FOR X=BOT*256+Y*4Ø TO BOT
    *256+Y*4Ø+LEN(A$)-1:POKE X+2,ASC(
    A$(X-BOT*256+1-Y*4Ø,X-BOT*256+1-Y
    *4Ø))-32
415 NEXT X:NEXT Y
42Ø POKE 53279,8:POKE 77,124
43Ø IF PEEK(53279)<>6 THEN 43Ø
44Ø RUN
45Ø DATA PRESS START FOR ANOTHER MAZE
    {1Ø SPACES}"
5ØØ AA=PEEK(5377Ø):AB=AA-(INT(AA/16)*
    16):SETCOLOR 2,AB,4:POKE 712,PEEK
    (71Ø):RETURN
1ØØØ DATA 1Ø4,1Ø4,133,2Ø8,1Ø4,133,2Ø7
     ,1Ø4,133,2Ø4,1Ø4,133,2Ø3,173,1Ø,
     21Ø,41,3,133,212
1Ø1Ø DATA 133,213,24,1Ø,168,165,2Ø3,1
     13,2Ø7,133,2Ø5,165,2Ø4,2ØØ,113,2
     Ø7,133,2Ø6,16Ø,Ø
1Ø2Ø DATA 177,2Ø5,2Ø1,128,2Ø8,4Ø,165,
     212,24,1Ø5,1,145,2Ø5,1Ø5,3,1Ø,16
     8,165,2Ø3,113
1Ø3Ø DATA 2Ø7,133,2Ø3,2ØØ,165,2Ø4,113
     ,2Ø7,133,2Ø4,169,Ø,168,145,2Ø3,1
     65,2Ø5,133,2Ø3,165
1Ø4Ø DATA 2Ø6,133,2Ø4,24,144,183,23Ø,
     212,165,212,41,3,133,212,197,213
     ,2Ø8,18Ø,16Ø,Ø
1Ø5Ø DATA 177,2Ø3,133,212,152,145,2Ø3
     ,169,251,24,1Ø1,212,176,24,198,2
     12,165,212,24,1Ø
```

```
1060 DATA 168,56,165,203,241,207,133,
     203,200,165,204,241,207,133,204,
     24,144,131,96,2
1070 DATA 0,176,255,254,255,80,0,1,0,
     216,255,255,255,40,0
```

Part Two

TAG

Ed Davis
Translated for the Atari by Charles Brannon

Presented here is a two-player game in graphics mode 1 — with a total of ten colors on the screen at once.

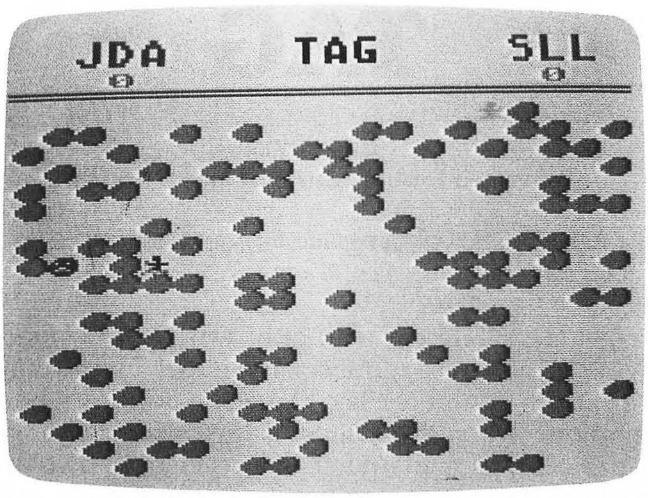
When playing real-life tag with only two players, nobody really wins because the number of tags per player remains constant. But in computer Tag, the clock decides who will be the champion. Every 15 seconds, if the person who is *It* cannot tag the other, the computer will reverse the It player. This feature allows a real fight for points. If you are not skilled in attacking, you can become skilled in evasive tactics and win the game.

Plug a joystick into jacks one and two, and get ready for some furious chasing and desperate dodging. After the game initializes, each player can type in his initials (three letters). You then select the final score (what you play to) from 1-10. Press OPTION to increase the final score, and SELECT when the desired number appears. The game will begin with player one in the upper left-hand corner, and player two in the opposite corner. Player one will be flashing, which indicates that he is *It*.

Play consists of It trying to catch the "victim" as fast as possible, while the "victim" tries to evade It for at least 15 seconds. Both players must maneuver about the screen, turning and twisting among a maze of pink rocks. But if you dally too long, the rocks will wake up, open their eyes, and further confound the conflict. Don't let one of the Living Rocks touch you.

Tag with a Twist

Tag uses character graphics in graphics mode one, but with a twist. Usually, if you want a redefined character set along with letters and numbers, you are limited to redefining punctuation and other special symbols and have to wait 10 to 15 seconds for a POKE loop that downloads the ROM character set to RAM.



Tag with a twist!

Tag, however, uses a Display List Interrupt (DLI) to "flip" the character set midway down the screen. This lets you use the upper portion of the display for normal text (using the entire character set), and the lower portion for as few or as many custom characters as desired. The DLI used in TAG also changes the screen colors, so you get five colors in each portion, for a total of ten simultaneous colors.

Flipping Out

Another interrupt-driven machine language routine in Tag uses Count-Down Timer 2 to "flip" the character set pointer every 16/60 of a second. In Tag, there are two character sets. The first character set, for example, displays one view of a running person. The other character set, at an offset of 512 bytes, displays another view.

When the CHBASE pointer is switched between the two views, the character appears to be running. Character set flipping can also be used to represent blinking, flashing, spinning, bouncing, or any other simple motion. And, since the flipping is controlled by machine language, the motion is fast and regular. It also simplifies the BASIC program.

Program 2-2. Tag

```
100 REM  TAG 
110 GOSUB 1170:REM INITIALIZE
120 PLR=1-PLR:IF PEEK(53279)=6 THEN R
    UN :REM ALLOW RESTART
130 IF PEEK(20)+256*PEEK(19)>900 THEN
     IT=1-IT:POKE 20,0:POKE 19,0:FOR
    W=15 TO 0 STEP -0.1:SOUND 0,10,12
    ,W:NEXT W
140 BLINK=BLINK-(BLINK>0):GOSUB 530
150 S=STICK(PLR):T=STRIG(PLR):POKE PO
    KEHERE+1,VV+IT
160 IF S=15 AND T=1 THEN S=S(PLR)
170 S(PLR)=S
180 SOUND PLR,S*5+100,10,4
190 TEST=POS(PLR)
200 TEST=TEST-20*(S=10 OR S=14 OR S=6
    )+20*(S=5 OR S=9 OR S=13)-(S>8 AN
    D S<12)+(S>4 AND S<8)
210 IF TEST<SCR+20 OR TEST>SCR+439 TH
    EN SOUND PLR,0,0,0:GOTO 120
220 CHR=(S>4 AND S<8)+3*(S>8 AND S<12
    )+2*(S=14 OR S=13)
230 SOUND PLR,0,0,0
240 P=PEEK(TEST):IF P=0 THEN POKE POS
    (PLR),0:POKE TEST,CHR+PLR*64:POS(
    PLR)=TEST:GOTO 120
250 Z=P-(PLR=0)*64:IF Z<1 OR Z>3 THEN
     280
260 IF PLR=IT THEN 310:REM GOTCHA
270 PLR=1-PLR:GOTO 310:REM WHOOPS!
280 IF P=196 THEN PLR=1-PLR:GOTO 310:
    REM "MONSTER" GOT PLAYER
290 GOTO 120
300 REM PLAYER CAUGHT ROUTINE
310 RESTORE 340:SOUND 3,0,0,0
320 POSITION 0,1:? #6;"  PLAYER ";(1-
    PLR)+1;" TAGGED! "
330 POKE POS(0),0:POKE POS(1),0
340 DATA 100,1,100,1,115,1,90,1,100,2
    ,120,3
```

Part Two

```
350 FOR I=1 TO 6:READ A,B
360 FOR W=15 TO Ø STEP -Ø.5/B:SOUND Ø
    ,A,1Ø,W:NEXT W
370 SOUND Ø,Ø,Ø,Ø:NEXT I
380 COLOR 32:PLOT Ø,1:DRAWTO 19,1
390 SCR(PLR)=SCR(PLR)+1
4ØØ POSITION 3,1:? #6;SCR(Ø):POSITION
     17,1:? #6;SCR(1)
41Ø IF MONSTERS THEN FOR I=1 TO MONST
    ERS:POKE MPOS(I),5+128:NEXT I
42Ø IF SCR(PLR)<ESCORE THEN IT=1-IT:G
    OSUB 151Ø:GOTO 12Ø
43Ø REM GAME OVER
44Ø FOR I=255 TO Ø STEP -5:POKE COLTA
    B+4,PEEK(5377Ø):SOUND Ø,I,12,4:SO
    UND 1,I,1Ø,4:NEXT I:SOUND Ø,Ø,Ø,Ø
45Ø POSITION Ø,1:? #6;"{3 SPACES}PLAY
    ER ";PLR+1;" wINS!{3 SPACES}"
46Ø FOR I=1 TO 5:FOR W=Ø TO 15:SOUND
    Ø,1Ø,Ø,W:NEXT W:FOR W=Ø TO 15:SOU
    ND Ø,12,Ø,15-W:NEXT W:NEXT I
47Ø POKE COLTAB+4,28:S=Ø:GOTO 49Ø
48Ø IF PEEK(2Ø)<25 THEN 51Ø
49Ø POKE 2Ø,Ø:POSITION 7,Ø:S=1-S:IF S
     THEN ? #6;"PRESS":GOTO 51Ø
5ØØ ? #6;"START":POKE 53279,Ø
51Ø IF PEEK(53279)<>6 THEN 48Ø
52Ø RUN
53Ø REM ...AND THE MONSTERS COME OUT
    TO PLAY
54Ø DURATION=DURATION-1:IF DURATION T
    HEN 59Ø:REM MAKE IT RARE
55Ø MONSTERS=MONSTERS+1:IF MONSTERS>8
     THEN MONSTERS=8:GOTO 59Ø
56Ø MPOS=SCR+2Ø+INT(42Ø*RND(Ø)):IF PE
    EK(MPOS)<>5+128 THEN 56Ø
57Ø MPOS(MONSTERS)=MPOS:MCUR(MONSTERS
    )=DIR(INT(8*RND(Ø))):MNERGY(MONST
    ERS)=2Ø-MONSTERS
58Ø BLINK=1Ø:POKE MPOS,6+128:DURATION
    =45:RETURN
59Ø IF MONSTERS=Ø OR BLINK THEN RETURN
```

```
6ØØ INDEX=INDEX+1:IF INDEX>MONSTERS T
    HEN INDEX=1
61Ø SOUND 3,INDEX*1Ø+2Ø,Ø,15
62Ø MPOS=MPOS(INDEX)+MCUR(INDEX):IF M
    POS<SCR+2Ø OR MPOS>SCR+419 THEN 6
    5Ø
63Ø P=PEEK(MPOS):IF P=Ø THEN POKE MPO
    S(INDEX),Ø:POKE MPOS,196:MPOS(IND
    EX)=MPOS:GOTO 67Ø
64Ø IF P<4 OR P>64 AND P<68 THEN PLR=
    1-(P>64):GOTO 31Ø:REM MONSTER BUM
    P PLAYER
65Ø MCUR(INDEX)=DIR(INT(8*RND(Ø)))
66Ø MNERGY(INDEX)=MNERGY(INDEX)-1
67Ø IF MNERGY(INDEX)>Ø THEN SOUND 3,Ø
    ,Ø,Ø:RETURN
68Ø REM TURN TO STONE
69Ø FOR I=1 TO 1Ø:SOUND 3,I*2+5Ø,Ø,8:
    NEXT I:SOUND 3,Ø,Ø,Ø
7ØØ MONSTERS=MONSTERS-1:POKE MPOS(IND
    EX),5+128:INDEX=INDEX-1
71Ø FOR I=INDEX+1 TO MONSTERS
72Ø MPOS(I)=MPOS(I+1):MCUR(I)=MCUR(I+
    1):MNERGY(I)=MNERGY(I+1)
73Ø NEXT I:SOUND 3,Ø,Ø,Ø
74Ø RETURN
75Ø END
76Ø CHSET=(PEEK(1Ø6)-8)*256:FOR I=Ø T
    O 7:POKE CHSET+I,Ø:POKE CHSET+512
    +I,Ø:NEXT I
77Ø RESTORE 81Ø:TP=Ø:IF PEEK(CHSET+8)
    =24 THEN 96Ø
78Ø READ A:IF A=-1 THEN 96Ø
79Ø FOR J=Ø TO 7:READ B:POKE CHSET+TP
    *512+A*8+J,B:SOUND Ø,B,1Ø,8:POKE
    712,B:NEXT J
8ØØ TP=1-TP:GOTO 78Ø:REM FOLLOWING DA
    TA STATEMENTS ARE CUSTOM CHARACTE
    RS
81Ø DATA 1,24,24,16,126,24,28,82,33
82Ø DATA 1,24,24,18,124,16,24,36,72
83Ø DATA 2,28,28,72,62,9,28,22,48
```

```
840 DATA 2,28,28,9,62,72,28,52,6
850 DATA 3,24,24,8,126,24,56,74,132
860 DATA 3,24,24,72,62,8,24,36,18
870 DATA 4,30,63,91,255,231,219,126,6
    0
880 DATA 4,30,63,91,255,231,195,126,6
    0
890 DATA 5,30,63,127,255,255,255,126,
    60
900 DATA 5,30,63,127,255,255,255,126,
    60
910 DATA 6,30,63,127,219,255,255,126,
    60
920 DATA 6,30,63,127,255,255,255,126,
    60
930 DATA 7,0,255,0,255,0,0,0,0
940 DATA 7,0,255,0,255,0,0,0,0
950 DATA -1
960 IF PEEK(1600)=173 THEN 980
970 FOR I=1536 TO 1629:READ A:POKE I,
    A:POKE 712,A:SOUND 0,A,10,8:NEXT
    I
980 SOUND 0,0,0,0:LET POKEHERE=1605:V
    V=22:COLTAB=1624
990 RETURN
1000 REM FOLLOWING IS MACHINE LANGUAG
     E CODE.  TYPE CAREFULLY.
1010 DATA 104,104,104,133,203,169
1020 DATA 36,141,0,2,169,6
1030 DATA 141,1,2,169,192,141
1040 DATA 14,212,169,76,141,40
1050 DATA 2,169,6,141,41,2
1060 DATA 169,16,141,26,2,96
1070 DATA 72,138,72,166,203,173
1080 DATA 92,6,141,10,212,141
1090 DATA 26,208,142,9,212,162
1100 DATA 4,189,87,6,157,21
1110 DATA 208,202,208,247,173,10
1120 DATA 210,9,6,141,22,208
1130 DATA 104,170,104,64,165,203
1140 DATA 73,2,133,203,169,16
1150 DATA 141,26,2,96,102,118
```

```
1160 DATA 72,216,28,0,0,0
1170 REM  INITIALIZATION CODE
1180 OPEN #1,4,0,"K:"
1190 GRAPHICS 2+16:POKE 538,0:POKE 54
     286,64
1200 POSITION 2,2:? #6;"THE":POSITION
      4,4:? #6;"GAME":POSITION 6,6:?
     #6;"of":POSITION 7,8:? #6;"t a g
     "
1210 FOR I=0 TO 3:SETCOLOR I,I,14-I*2
     :NEXT I
1220 FOR I=1 TO 50:POKE 53274,PEEK(53
     770):POKE 53279,0:POKE 712,PEEK(
     53770):NEXT I
1230 GOSUB 760:REM INITIALIZE CHSET A
     ND MACHINE LANGUAGE
1240 GRAPHICS 1+16:DL=PEEK(560)+256*P
     EEK(561)+4
1250 A=USR(1536,CHSET/256)
1260 SETCOLOR 4,0,14:SETCOLOR 3,15,8:
     SETCOLOR 0,2,10:SETCOLOR 2,9,6
1270 SCR=PEEK(DL)+256*PEEK(DL+1)+40
1280 POKE DL-1,7+64
1290 POKE DL+2,PEEK(DL+2)+128
1300 FOR I=1 TO 120
1310 P=SCR+30+INT(388*RND(0)):IF PEEK
     (P) THEN 1310
1320 POKE P,5+128:NEXT I
1330 FOR PLR=0 TO 1
1340 POSITION 6,0:? #6;"PLAYER ";PLR+
     1
1350 POSITION 1,1:? #6;"ENTER YOUR IN
     ITIALS":FOR I=1 TO 3
1360 GET #1,A:IF A<32 OR A>90 THEN 13
     60
1370 COLOR A+32*(A>64)+PLR*128:PLOT P
     LR*14+1+I,0:NEXT I
1380 COLOR 32:PLOT 5,0:DRAWTO 15,0:PL
     OT 0,1:DRAWTO 19,1:NEXT PLR:COLO
     R 48:PLOT 3,1:PLOT 17,1
1390 POSITION 7,0:? #6;"play to":ESCO
     RE=5
```

```
1400 IF PEEK(53279)=5 THEN 1460
1410 POSITION 8,1:? #6;"E ";ESCORE;"
     ";
1420 IF PEEK(53279)<>3 THEN 1400
1430 IF PEEK(53279)=3 THEN 1430
1440 ESCORE=ESCORE+1:IF ESCORE>10 THE
     N ESCORE=1
1450 GOTO 1400
1460 COLOR 32:PLOT 5,0:DRAWTO 15,0:PL
     OT 5,1:DRAWTO 15,1
1470 POSITION 9,0:? #6;"TAG":IT=0:PLR
     =IT
1480 POSITION 0,2:? #6;"{20 E}"
1490 DIM POS(1),S(1),SCR(1),MPOS(8),D
     IR(7),MCUR(8),MNERGY(8):SCR(0)=0
     :SCR(1)=0
1500 DIR(0)=20:DIR(1)=20:DIR(2)=19:DI
     R(3)=-19:DIR(4)=21:DIR(5)=-21:DI
     R(6)=1:DIR(7)=-1
1510 POKE 20,0:POKE 19,0:MONSTERS=0:D
     URATION=70
1520 POS(0)=SCR+20:POS(1)=SCR+419:S(0
     )=7:S(1)=11:Z=0
1530 RETURN
```

Hidden Maze

Gary Boden
Translated for the Atari by Charles Brannon

This game offers a different twist to maze puzzles: you can see only a very small section of the maze at a time.

Mazes present a challenge different from arcade-type "shootout" games, but the appeal of a maze can quickly fade once it has been solved. I have enhanced the challenge by hiding the complete maze from the player and showing only a realistically limited view from any position inside it. Although the view is from above rather than ground level, the player still gets a claustrophobic feeling similar to that of actually being inside the maze and groping along the corridors.

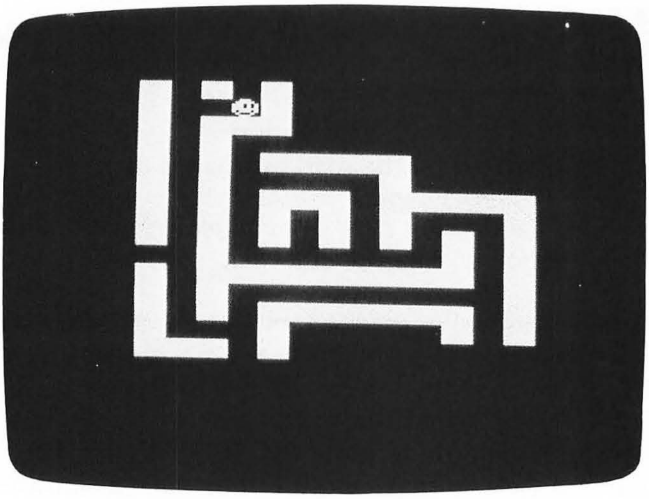
The objective is simply to find a way out of the maze in the least amount of time. You start at the center of the maze with only your player character visible. As you move through the passages, the walls around your player become visible, and the maze unfolds.

Playing Hidden Maze

Use the joystick to move your ebullient little character around the maze, your goal being the upper-left-hand corner of the screen. The challenge is in how long it takes you to get there. You can take a "cheat peek" of the entire maze by pressing the fire button. This will display the maze for about three seconds, then turn to black and delay your movement for another three seconds as a penalty. If you want a really good score, don't use it!

We can construct the maze directly on the screen (GRAPHICS 1 is used here, with custom characters for the walls and player). We make it invisible by setting its color equal to the background color (done here with SET-COLOR 2,0,0).

Then, to open up the maze, we just have to PEEK (into screen memory) the eight characters surrounding the player


The more success you have, the more of the maze you see.

character, and if the PEEKed character is an "invisible wall," replace it with a visible wall.

Scoring is provided with RTCLOCK, Atari's realtime clock, which is found at location 18, 19, and 20. These are used in the opposite of the normal LSB/MSB order. Chaining all three locations together will give the current "jiffy time" since the machine was turned on, measured in sixtieths of a second:

JIFFY=PEEK(20)+PEEK(19)*256+PEEK(18)*65536

Since location 18 only ticks every once in a long while, you can leave it out for most measurements. Dividing the jiffy time by 60 gives you the time in seconds:

SEC=(PEEK(20)+256*PEEK(19))/60

The ability to add timing to your programs will enhance them. Many games would lose their challenge without the time element. Try experimenting with locations 18, 19, and 20. Experimenting is an excellent way to learn more about your computer's capabilities.

Program 2-3. Hidden Maze

```
100 REM HIDDEN MAZE
110 GRAPHICS 17:GOSUB 360:GOSUB 480
```

Part Two

```
120 PPOS=SC+23Ø
130 POKE PPOS,5
140 DIM DIR(3)
150 DIR(Ø)=2Ø:DIR(1)=21:DIR(2)=19:DIR
    (3)=1
160 POKE 2Ø,Ø:POKE 19,Ø
170 FOR I=Ø TO 3
180 ZP=PPOS+DIR(I):PK=PEEK(ZP):POKE Z
    P,PK-64*(PK=129)
190 ZP=PPOS-DIR(I):PK=PEEK(ZP):POKE Z
    P,PK-64*(PK=129)
200 NEXT I
210 ST=STICK(Ø):TPOS=PPOS+2Ø*(ST=13)-
    2Ø*(ST=14)+(ST=7)-(ST=11)
220 CHR=3*(ST=11)+4*(ST=7)+5*(ST=14)+
    6*(ST=13)
230 IF STRIG(Ø)=Ø THEN SETCOLOR 2,Ø,1
    4:FOR W=1 TO 5ØØ:NEXT W:SETCOLOR
    2,Ø,Ø:FOR W=1 TO 5ØØ:NEXT W
240 IF STRIG(Ø)=Ø THEN 24Ø
250 IF PEEK(TPOS) THEN 27Ø
260 POKE PPOS,Ø:POKE TPOS,CHR:PPOS=TP
    OS:IF PPOS<>SC+21 THEN 17Ø
270 IF PPOS<>SC+21 THEN 17Ø
280 FOR I=1 TO 5Ø:FOR J=Ø TO 3:POKE 7
    Ø8+J,PEEK(5377Ø):NEXT J:NEXT I
290 GRAPHICS 18:? #6;"you did it{A}"
300 SEC=INT((PEEK(2Ø)+256*PEEK(19))/6
    Ø)
310 ? #6;"IN ";SEC;" SECONDS."
320 ? #6:? #6;"press FIRE to"
330 ? #6;"play again{N}"
340 IF STRIG(Ø) THEN POKE 711,PEEK(53
    77Ø):GOTO 34Ø
350 RUN
360 CHSET=(PEEK(1Ø6)-8)*256:FOR I=Ø T
    O 7:POKE CHSET+I,Ø:NEXT I
370 RESTORE 41Ø
380 READ A:IF A=-1 THEN RETURN
390 FOR J=Ø TO 7:READ B:POKE CHSET+A*
    8+J,B:NEXT J
400 GOTO 38Ø
```

```
410 DATA 3,56,124,174,174,254,186,68,
    56
420 DATA 4,56,124,234,234,254,186,68,
    56
430 DATA 5,56,84,214,254,254,186,68,5
    6
440 DATA 6,56,124,254,214,214,186,68,
    56
450 DATA 1,255,255,255,255,255,255,25
    5,255
460 DATA 127,16,24,28,3Ø,3Ø,28,24,16
470 DATA -1
48Ø GRAPHICS 17:POKE 756,CHSET/256
49Ø SC=PEEK(88)+256*PEEK(89):SETCOLOR
     2,Ø,Ø
5ØØ DIM A(3):A(Ø)=2:A(1)=-4Ø:A(2)=-2:
    A(3)=4Ø:WL=129:HL=Ø:TRAP 32767
51Ø A=SC+21
52Ø FOR I=1 TO 21:? #6;"!!!!!!!!!!!!!!
    !!!!!!":NEXT I:POKE A,5
53Ø J=INT(RND(1)*4):X=J
54Ø B=A+A(J)
55Ø IF PEEK(B)=WL THEN POKE B,J+1:POK
    E A+A(J)/2,HL:A=B:GOTO 53Ø
56Ø J=(J+1)*(J<3):IF J<>X THEN 54Ø
57Ø J=PEEK(A):POKE A,HL:IF J<5 THEN A
    =A-A(J-1):GOTO 53Ø
58Ø RETURN
```

Part Three

Two-Player Games

Blockade

Douglas Pinho

"Blockade" is an exciting two-player game that also demonstrates a simple joystick routine.

Surround (or Blockade) was a popular arcade game in the early days of video games. The format of the game is not complex, but it is still enjoyable and challenging. The object of the game is to build walls to trap the opposing player and force him to collide with: his own walls, the opposing player's walls, or the boundaries of the playfield. When this occurs, the player who did not crash receives a point. Upon every collision, the walls of the player who crashed will blink. The screen is then cleared and the game continues.

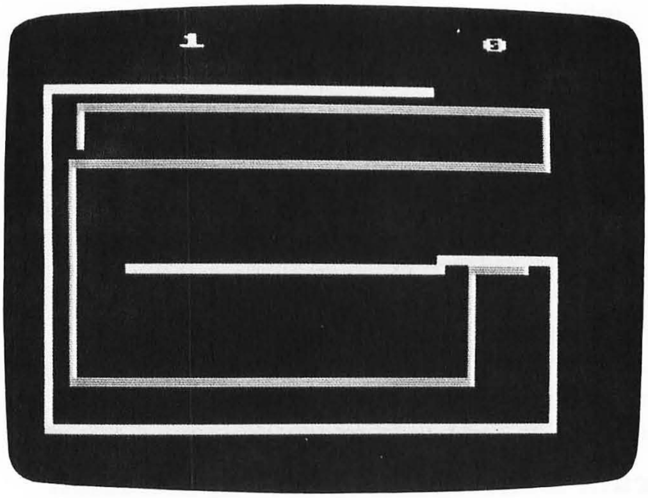
The first player to reach nine points is the winner. To start the next game, just press the fire button. To play, plug joysticks into the first two joystick ports (sticks 1 and 2).

Program Description and Explanation

Lines 1-2 set up the title display. Line 5 sets up a mixed graphics mode with one line of GR. 1 followed by one line of GR. 2 and 44 lines of GR. 5. START calculates the address of the display list in memory. This pointer is needed since the location of the display list depends upon the amount of memory installed in the Atari. The two POKEs then place instructions for the desired graphics modes at the appropriate memory locations. Line 10 initializes the variables X and Y, the starting location of player 1, and S and T which give the location of player 2. Variables X1 and Y1 and S1 and T1 are the increment or decrement values for plotting the walls on the screen. F is a flag to determine whether there was a simultaneous collision between the two players. H1 and B1 are used to keep score. Line 12 plots the boundaries of the playing field in blue. POKEing memory location 87 (current screen mode) with 5 directs the computer to plot in GR. mode 5. This is needed only in a mixed graphics mode. Line 14 goes to a subroutine at line 300 which prints the score in

GR. 2 characters. Line 15 checks for the end of the game.

Lines 20-120 contain the main game loop. Lines 25 to 43 check for joystick movement and assign the move variable (X1, Y1, S1, T1), and a value for P and L. One of the nice features of Atari BASIC is that you can use a variable as a GOSUB address. This feature is used in line 50 to branch to



Try to outwit an opponent in "Blockade."

different subroutines depending upon the value of P (player 1) and L (player 2). Note that in line 23 you must POKE 5 into memory location 87 again because it was changed during subroutine 300 (line 14). Lines 150 to 185 first check for a collision. If there is none, it plots the new block. A collision is found by locating the next position in front of the plotted block and finding its color. If the color is 0 (which is the background default color), it continues and plots the next block. If it is any other color, there is a collision. If the first player has collided, the program branches to line 201 to check for a simultaneous collision by the other player. Flag F is set if a simultaneous collision is found. Lines 210-220 update the score and blink the losing player's walls. Subroutine 300 prints the score at the top of the screen in GR. 2 characters. Subroutine 350 blinks the colors of the colliding player's walls.

Lines 400-410 check if you want to start a new game (prints in GR. 1 characters).

If you haven't played "Blockade" before, grab a friend and try it. It requires quick decisions and good strategy. You'll like it.

Program 3-1. Blockade

```
1 GRAPHICS 2+16:SETCOLOR 4,5,5:POSIT
  ION 6,5:PRINT #6;"BLOCKADE"
2 FOR D1=1 TO 6:FOR E1=Ø TO 89:SOUND
   1,E1,1Ø,1Ø:NEXT E1:NEXT D1:SOUND
  1,Ø,Ø,Ø
5 GRAPHICS 5+16:START=PEEK(56Ø)+PEEK
  (561)*256+4:POKE START-1,71:POKE S
  TART+2,6
1Ø X=13:Y=23:X1=1:Y1=1:S=66:T=23:S1=
   -1:T1=1:P=16Ø:L=17Ø:F=Ø
12 POKE 87,5:COLOR 3:PLOT Ø,3:DRAWTO
    Ø,46:DRAWTO 78,46:DRAWTO 78,3:DR
   AWTO Ø,3
14 GOSUB 3ØØ
15 IF H1=9 OR B1=9 THEN 4ØØ
2Ø B=STICK(Ø):H=STICK(1)
21 SOUND 3,2ØØ,1Ø,15
23 POKE 87,5
25 IF B=14 THEN Y1=-1:P=15Ø
27 IF H=14 THEN T1=-1:L=18Ø
3Ø IF B=13 THEN Y1=1:P=15Ø
32 IF H=13 THEN T1=1:L=18Ø
35 IF B=7 THEN X1=1:P=16Ø
37 IF H=7 THEN S1=1:L=17Ø
4Ø IF B=11 THEN X1=-1:P=16Ø
43 IF H=11 THEN S1=-1:L=17Ø
44 SOUND 3,15Ø,1Ø,15
5Ø GOSUB P:GOSUB L
12Ø GOTO 2Ø
15Ø Y=Y+Y1:COLOR 1:LOCATE X,Y,Z:IF Z
    <>Ø THEN GOTO 2Ø1
155 PLOT X,Y:RETURN
16Ø X=X+X1:COLOR 1:LOCATE X,Y,Z:IF Z
    <>Ø THEN GOTO 2Ø1
```

```
165 PLOT X,Y:RETURN
170 S=S+S1:COLOR 2:LOCATE S,T,U:IF U
    <>Ø THEN GOTO 22Ø
175 PLOT S,T:RETURN
18Ø T=T+T1:COLOR 2:LOCATE S,T,U:IF U
    <>Ø THEN GOTO 22Ø
185 PLOT S,T:RETURN
2Ø1 IF L=17Ø THEN S=S+S1:POSITION S,
    T:LOCATE S,T,U:IF U<>Ø THEN F=1
2Ø2 IF L=18Ø THEN T=T+T1:POSITION S,
    T:LOCATE S,T,U:IF U<>Ø THEN F=1
2Ø3 GOTO 21Ø
21Ø SOUND 3,Ø,Ø,Ø:SOUND 1,1ØØ,14,14:
    FOR W=1 TO 3ØØ:NEXT W:B1=B1+1:GO
    SUB 3ØØ:Q1=Ø:GOSUB 35Ø:SOUND 1,Ø
    ,Ø,Ø
211 IF F=1 THEN GOTO 22Ø
212 GOTO 5
22Ø SOUND 3,Ø,Ø,Ø:SOUND 2,15Ø,12,14:
    FOR W=1 TO 3ØØ:NEXT W:H1=H1+1:GO
    SUB 3ØØ:Q1=1:GOSUB 35Ø:SOUND 2,Ø
    ,Ø,Ø:GOTO 5
3ØØ POKE 87,2:POSITION 5,Ø:PRINT #6;
    H1:POSITION 15,Ø:PRINT #6;CHR$(B
    1+16):RETURN
35Ø FOR P1=1 TO 7:FOR U1=1 TO 4Ø:NEX
    T U1:SETCOLOR Q1,9,4:FOR G1=1 TO
     4Ø:NEXT G1:SETCOLOR Q1,4,6:NEXT
     P1:RETURN
4ØØ POKE 87,1:POSITION Ø,1:PRINT #6;
    "PRESS fire TO BEGIN!"
4Ø5 IF STRIG(Ø)=Ø OR STRIG(1)=Ø THEN
     H1=Ø:B1=Ø:GOTO 5
41Ø GOTO 4Ø5
```

Part Three

Tank Duel

Tom R. Halfhill

"Tank Duel" is a two-player action game requiring a pair of joysticks. Two program listings are included: standard Atari BASIC (16 RAM required), and BASIC A + (48K and BASIC A+ language required). Only the Atari BASIC version will run on an XL-model computer.

Most video and computer games pit you against the computer itself. They also tend to be tests of reaction reflexes. So it's no surprise that you, a mere human, are doomed to eventual failure. Computers have a way to go before they catch up with people in all aspects of intelligence, but when it comes to response time, silicon beats protoplasm any day.

That's why I've always enjoyed computer games that pit two *people* against each other. These games bring the full range of human characteristics into play, characteristics that are very difficult to program into a computer: real (as opposed to "artificial") intelligence; deception; skill that improves with experience; and emotions of excitement, frustration, and panic. The computer becomes the tool of interaction, the battlefield of conflict, like a chessboard. The unique capabilities of a computer also allow it to act as a referee, by creating a stage for the contest which precludes any actions not sanctioned by the program.

How to Play Tank Duel

"Tank Duel" transforms the screen into a large battlefield occupied by two tanks. Each player controls one of the tanks with a joystick. The tanks are fully maneuverable and armed with a cannon fired by pressing the joystick trigger button. In addition to the tanks on the screen, each player also has two extra tanks in reserve. When a player's on-screen tank is destroyed, it is automatically replaced by one of these reserve tanks. The object of the game is to survive: destroy all the enemy tanks before the enemy destroys yours.

First, type the program listing carefully, especially the DATA statements. Some of these DATA statements contain machine language subroutines, and a mistyped number could cause your computer to crash (the keyboard locks up and refuses to respond to your commands). If this happens, and the SYSTEM RESET key also locks up, the only way to recover is to turn the computer off and then on again. Of course, this means the program will be erased. So to be safe, save the program on disk or tape at least twice before running it for the first time. Then if the computer crashes, you can turn the computer off and on to clear it, and simply reload the program to start looking for the mistake.

Also make sure to type in the correct program listing. If you are using the standard Atari BASIC cartridge, type the Atari BASIC version. The other version is for people with BASIC A +, an advanced BASIC on disk that requires 48K (see "Technical Details," below).

When you're ready to play, type RUN and press RETURN. A title screen appears for 30 seconds while the program initializes (sets itself up). When the battlefield appears and you hear the low rumble of two idling tank engines, the game is ready to start.

Here's how the joystick controls work: to rotate a tank in place, move the stick right or left. Moving the stick right rotates the tank clockwise, and moving the stick left rotates the tank counterclockwise. Pushing the joystick forward drives the tank in whichever direction it's pointed. If these joystick controls sound familiar, it's because they are almost identical to the way you manipulate the spaceship in the Atari computer version of *Asteroids.* The only difference is that pulling backward on the joystick does not flip your tank into hyperspace!

The battlefield, surrounded by a screen border, is dotted with trees, houses, and other buildings of varying shapes and sizes. Note that you can drive your tank behind trees for concealment, but that you cannot drive through buildings (your tank just bounces off). You also cannot drive through the screen border. And if you try ramming the enemy tank, your own tank blows up, so forget about kamikaze charges.

To fire your tank's cannon, press the joystick trigger button. The gun's range is limited to about two-thirds the breadth or height of the screen.

When a player's tank is destroyed, action stops for a few seconds while it is replaced by one of the reserve tanks. The number of tanks each player has in reserve is indicated by the tank symbols in the lower left and right corners of the screen. Each time a player's tank is blown up, one of these tank symbols is replaced by a tiny cross.

Just to make the game a little more interesting, there is another hazard besides the enemy tanks — minefields. Every game, a minefield — with four mines — is planted at random somewhere on the battlefield. The minefield is about an inch square and is invisible. The only way you'll know that you found a minefield is when your tank blows up. (Hint: minefields are planted in otherwise empty space, never beneath trees, so you're safe in the forest.)

When all three of one player's tanks are destroyed, the game ends. The winner's tank flashes colors for a few seconds, and then the battlefield disappears. A screen message explains how to start a new game of Tank Duel by pressing either joystick trigger button.

Technical Details: BASIC A +

Tank Duel originally was written, not in Atari BASIC, but in BASIC A +. If you're not familiar with BASIC A +, it's an enhanced version of Atari BASIC produced by Optimized Systems Software, Inc., of Cupertino, California. At this writing, BASIC A + is available only on disk, although a cartridge version is being planned. The disk version requires at least 32K of memory (48K recommended). BASIC A + is *upward compatible* with Atari BASIC, which means any program written in Atari BASIC will run in BASIC A + (and will run faster, since BASIC A + is more streamlined). However, most programs written in BASIC A + will not run in Atari BASIC.

Why was BASIC A + my first choice? Because this program was my first attempt to use some of the Atari's special features such as player/missile graphics and redefined characters. This was easier done in BASIC A + than in Atari BASIC.

When standard Atari BASIC was created, it was decided to fit the whole language into a single cartridge within only 8K of memory. Although Atari BASIC has many powerful features (i.e., instant syntax checking, unlimited-length strings, built-in commands for graphics and sound, etc.), it lacks special commands for more advanced features such as player/missile graphics. P/M graphics, known on some other home computers as "sprites," allows you to create shapes and move them around the screen without disturbing other screen objects. It's also easy to determine if a P/M object is colliding with any other objects. This makes P/M graphics a powerful feature for games.

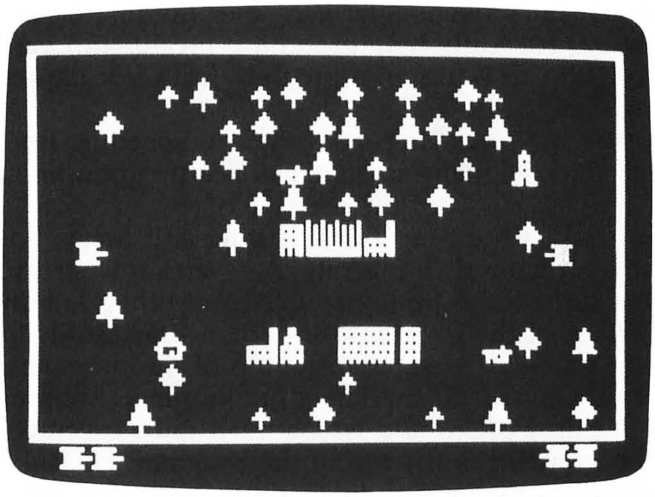
However, since Atari BASIC lacks built-in P/M commands, this feature is hard to learn and use. That's where BASIC A+ really shines. It has many more commands than Atari BASIC, including keywords such as PMGRAPHICS, PMCOLOR, and PMMOVE. I found it much easier to learn P/M graphics with these commands than to struggle with the many POKEs necessary in Atari BASIC. To see for yourself, compare the BASIC A+ program listing with the Atari BASIC listing.

After finishing Tank Duel in BASIC A+, though, I decided to translate it into standard Atari BASIC. Why? For one thing, I wanted to give copies of the game to a few friends who did not have BASIC A+. Second, the program could not have been published in this book unless it appealed to the widest possible range of readers, and only a minority of Atari users have BASIC A+. And finally, I wanted to learn how to use P/M graphics and other special features in my computer's standard language, Atari BASIC. However, writing Tank Duel first in BASIC A+ was a valuable experience. It introduced me to some of the Atari's advanced techniques in a more friendly way.

So if we have an Atari BASIC translation that anyone can use, why are we including the original BASIC A+ version here? For two reasons: Atari users unfamiliar with BASIC A+ can compare the two listings to see the differences, and people who already own BASIC A+ can type in that listing and take advantage of the language's superior features. The BASIC A+ version still runs a little faster and more smoothly than the Atari BASIC translation. Note that this is despite the fact that the Atari BASIC version includes two machine language subroutines to speed up the animation, while the original version is pure unadulterated BASIC.

Streamlined Programming

If you are interested in trying some game programming yourself, there are a few tricks you can learn by studying these program listings. (For the rest of this discussion, we'll


The beginning of the "Tank Duel."

assume you have some fundamental understanding of Atari BASIC.)

Tank Duel was programmed with two goals in mind: to make it run as fast as possible in BASIC, and to consume as little memory as possible. When writing any program that may be published or given to others, it is a good idea to make it workable on minimum systems so the widest possible range of people can use it. Tank Duel fits in 16K, which is the minimum Atari system that has been sold recently. Several techniques were used to make the program compact and fast. Note that not all of these are considered good programming techniques in programs which do not have to meet such requirements.

REM statements are kept to a minimum to save memory. To make up for this somewhat, most of the variable names are not meaningless single letters, but rather short words which mean what they do. (Luckily, Atari BASIC allows very long variable names.) For instance, the string variable SHAPE$ holds the player shape data; MISSILE$ holds the missile shape data; HORIZ is the player horizontal position value, and VERT is the vertical value; MHORIZ is the missile horizontal value; and so on. This doesn't consume as much memory as you might think, because Atari BASIC stores variables as short tokens internally after the first occurrence. What's more, it's really helpful to have meaningful variables when you stop working on a program for a few days and come back later.

Another way of saving memory, and speeding up execution, is to pack as many statements on a program line as will fit.

Also, look at the Atari BASIC listing and notice how the program is structured (the BASIC A+ version is structured somewhat differently). Many people group their subroutines at the bottom of the program. But when Atari BASIC encounters a GOSUB, it starts at the top of the program and searches downward for the target line number. Subroutines which are called often, and which need to be executed fast, should be grouped at the top of the program. In Tank Duel, the subroutines for moving tanks (lines 1000-1180), firing shots (lines 2000-2200), and checking for hits (lines 4000-4820) are placed above the "main loop" (lines 10040-10900). The subroutines for initialization (setting up the program) are tucked away beneath the main loop. Many of these lines are executed only once, when the game is first run.

A Little Machine Language

As mentioned, the BASIC A+ version of Tank Duel was fast enough to get along fine all by itself. But the Atari BASIC version needed a little help from machine language.

Since machine language, not BASIC, is actually the language which the computer uses internally, programs written directly in machine language do not have to be interpreted by the computer and always run much faster. Unfortunately, machine language is the hardest language for humans to master. Tank Duel needed machine language for two things: to rotate the tanks, and to move the tanks vertically and diagonally (the Atari has built-in provisions for easily moving P/M objects horizontally, but not for moving them in any other direction). When I wrote Tank Duel, I knew absolutely nothing about machine language. The solution was to use two machine language routines published in *COMPUTE!*. These are "canned" routines; that is, you can be totally ignorant about machine language and still use them. Since they are very short and very handy, let's review them briefly here.

The first one allows fairly fast vertical and diagonal movement. It originally appeared as "Adding High-Speed Vertical Positioning To P/M Graphics," by David H. Markley, in *COMPUTE!*, December 1981. It was reprinted in *COMPUTE!'s First Book of Atari Graphics*. To use this routine in your own programs, copy lines 11100-11160 from the Atari BASIC listing. The DATA statements are decimal equivalents of the machine language instructions; type them very carefully. Line 11160 POKEs the routine into page six, an area of free memory in the Atari.

This routine requires that the first number in the DATA statement which defines the shape of your player be equal to the height of the player. For example, look at line 12020. This DATA statement defines the shape of a tank facing "north" (up). There are nine numbers in the DATA statement. The first number is an eight because the player shape is eight bytes high. The following eight numbers are the eight bytes (the second and third numbers just happen to be eights by coincidence). If the player were ten bytes high, the DATA statement would contain 11 numbers — a ten followed by the ten bytes.

To call this routine, use this statement:

```
A =USR(1536,SHAPE,CURRENT LOCATION, NEW
LOCATION)
```

"A" is a dummy variable which doesn't mean anything, but is required by the syntax of the USR statement. There are four parameters in the parentheses of the USR statement. The first parameter is the address of the machine language routine itself; 1536 is the start of page six. The next parameter is the address of the player shape. Tank Duel stores the player shapes in SHAPE$ and uses the ADR function to find the address. The third parameter is the player's current location, the actual memory address of the vertical position. The last parameter is the new location, the memory address of the vertical position to which you want to move the object.

For an example, see the first statement in line 4200:

```
A =USR(1536,ADR(SHAPE$(64)),VERT, VERT +2)
```

The ADR function finds the address of the substring starting at SHAPE$(64). This is where the player shape that is being moved is stored. Remember that the first number in this shape is an eight, the height of the player. The third parameter, VERT, is a variable containing the memory address of the player's vertical position (it is an offset from PMBASE). The last parameter tells the routine to move the player to VERT +2, or two notches *down* the screen (shifting a player *up* in memory moves it *down* the screen, and vice versa).

P/M Metamorphosis

The second machine language routine in the Atari BASIC version of Tank Duel allows you to instantly change a player's shape. This is how the tanks rotate; the rotation is really an illusion. Actually there are eight different tank shapes stored in SHAPE$, one for each direction a tank can face. Pushing the joystick right or left calls this routine and draws the appropriate shape.

The routine originally appeared in the article "Extending Player/Missile Graphics," by Eric Stoltman, *COMPUTE!*, October 1981, and also was reprinted in *COMPUTE!'s First Book of Atari Graphics.* To use this routine in your own programs, copy lines 11180-11220. The DATA in line 11220 is the machine language, and line 11200 stores the routine in a string called MOVE$, since we already used page six for the other routine (although there was plenty of room left). The variable MOVE is the address of MOVE$, making it easier to call the routine later.

Like the previous routine, this one also needs to know the height of the player. It handles this differently, though. It stores the player's height in the 22nd number of the DATA statement. Notice that this is an eight. For a ten-byte player, you would change this number to a ten.

Call the routine with this statement:

```
A =USR(MOVE,CURRENT LOCATION,SHAPE)
```

"A" is the usual dummy variable. This routine uses only three parameters in the USR statement. The first is the address of the routine itself. Remember, we stored this routine in MOVE$ and set MOVE equal to the string address. You could replace MOVE with ADR(MOVE$) and get the same effect. The second parameter is the actual memory address of the player's vertical position, just as required before. And the last parameter, SHAPE, is where the player's shape is stored, also as before. See line 4700 for an example.

If you intend to tackle some game programming with player/missile graphics, these two routines will be invaluable additions to your routine library. They are short, easy to use, and relatively fast. For a fuller explanation, be sure to look up the original articles. Since they were first published, they've been the basis of several fine games in Atari BASIC.

Program 3-2. Tank Duel — Atari BASIC

```
10 REM *** TANK DUEL ***
20 REM *** ATARI BASIC VERSION 2 ***
30 REM ****
50 GOSUB 11000
60 GOSUB 12000
70 GOSUB 14000
80 GOSUB 13000
90 GOTO 10000
1000 REM *** MOVE TANKS ***
1020 ON HEADING GOTO 1040,1060,1080,1
     100,1120,1140,1160,1180
1040 A=USR(1536,ADR(SHAPE$(1)),VERT,V
     ERT-2):VERT=VERT-2:RETURN
1060 A=USR(1536,ADR(SHAPE$(10)),VERT,
     VERT-2):POKE HREG,HORIZ+2:VERT=V
     ERT-2:HORIZ=HORIZ+2:RETURN
1080 POKE HREG,HORIZ+2:HORIZ=HORIZ+2:
     RETURN
1100 A=USR(1536,ADR(SHAPE$(28)),VERT,
     VERT+2):POKE HREG,HORIZ+2:VERT=V
     ERT+2:HORIZ=HORIZ+2:RETURN
1120 A=USR(1536,ADR(SHAPE$(37)),VERT,
     VERT+2):VERT=VERT+2:RETURN
1140 A=USR(1536,ADR(SHAPE$(46)),VERT,
     VERT+2):POKE HREG,HORIZ-2:VERT=V
     ERT+2:HORIZ=HORIZ-2:RETURN
1160 POKE HREG,HORIZ-2:HORIZ=HORIZ-2:
     RETURN
1180 A=USR(1536,ADR(SHAPE$(64)),VERT,
     VERT-2):POKE HREG,HORIZ-2:VERT=V
     ERT-2:HORIZ=HORIZ-2:RETURN
2000 REM *** FIRE MISSILES ***
```

```
2010 TRAP 2200:FOR I=15 TO 0 STEP -1:
     SOUND 2,90,4,I:NEXT I
2020 A=USR(1536,ADR(MISSILE$),PMBASE+
     384,MVERT):POKE 77,0:ON HEADING
     GOTO 2040,2060,2080,2100,2120,21
     40,2160,2180
2040 POKE MREG,MHORIZ+4:FOR I=1 TO 15
     :A=USR(1536,ADR(MISSILE$),MVERT,
     MVERT-4):MVERT=MVERT-4:NEXT I
2050 A=USR(1536,ADR(MISSILE$),MVERT,0
     ):RETURN
2060 POKE MREG,MHORIZ+4:MHORIZ=MHORIZ
     +4:FOR I=1 TO 15:A=USR(1536,ADR(
     MISSILE$),MVERT,MVERT-3)
2070 POKE MREG,MHORIZ+3:MVERT=MVERT-3
     :MHORIZ=MHORIZ+3:NEXT I:A=USR(15
     36,ADR(MISSILE$),MVERT,0):RETURN
2080 POKE MREG,MHORIZ+8:MHORIZ=MHORIZ
     +8:FOR I=1 TO 25:POKE MREG,MHORI
     Z+3:MHORIZ=MHORIZ+3:NEXT I:POKE
     MREG,0
2090 A=USR(1536,ADR(MISSILE$),MVERT,0
     ):RETURN
2100 POKE MREG,MHORIZ+4:MHORIZ=MHORIZ
     +4:FOR I=1 TO 15:A=USR(1536,ADR(
     MISSILE$),MVERT,MVERT+3)
2110 POKE MREG,MHORIZ+3:MVERT=MVERT+3
     :MHORIZ=MHORIZ+3:NEXT I:A=USR(15
     36,ADR(MISSILE$),MVERT,0):RETURN
2120 POKE MREG,MHORIZ+4:FOR I=1 TO 15
     :A=USR(1536,ADR(MISSILE$),MVERT,
     MVERT+4):MVERT=MVERT+4:NEXT I
2130 A=USR(1536,ADR(MISSILE$),MVERT,0
     ):RETURN
2140 POKE MREG,MHORIZ+3:MHORIZ=MHORIZ
     +3:FOR I=1 TO 15:A=USR(1536,ADR(
     MISSILE$),MVERT,MVERT+3)
2150 POKE MREG,MHORIZ-3:MVERT=MVERT+3
     :MHORIZ=MHORIZ-3:NEXT I:A=USR(15
     36,ADR(MISSILE$),MVERT,0):RETURN
2160 POKE MREG,MHORIZ:FOR I=1 TO 25:P
     OKE MREG,MHORIZ-3:MHORIZ=MHORIZ-
```

```
      3:NEXT I:POKE MREG,0
2170  A=USR(1536,ADR(MISSILE$),MVERT,0
      ):RETURN
2180  POKE MREG,MHORIZ+3:MHORIZ=MHORIZ
      +3:FOR I=1 TO 15:A=USR(1536,ADR(
      MISSILE$),MVERT,MVERT-3)
2190  POKE MREG,MHORIZ-3:MVERT=MVERT-3
      :MHORIZ=MHORIZ-3:NEXT I:A=USR(15
      36,ADR(MISSILE$),MVERT,0):RETURN
2200  A=USR(1536,ADR(MISSILE$),MVERT,0
      ):RETURN
4000  REM *** COLLISIONS ***
4020  REM *** BUMP PLAYFIELD ***
4040  POKE HITCLR,0:ON HEADING GOTO 40
      60,4080,4100,4120,4140,4160,4180
      ,4200
4060  A=USR(1536,ADR(SHAPE$(1)),VERT,V
      ERT+2):VERT=VERT+2:GOTO 4220
4080  A=USR(1536,ADR(SHAPE$(10)),VERT,
      VERT+2):POKE HREG,HORIZ-2:VERT=V
      ERT+2:HORIZ=HORIZ-2:GOTO 4220
4100  POKE HREG,HORIZ-2:HORIZ=HORIZ-2:
      GOTO 4220
4120  A=USR(1536,ADR(SHAPE$(28)),VERT,
      VERT-2):POKE HREG,HORIZ-2:VERT=V
      ERT-2:HORIZ=HORIZ-2:GOTO 4220
4140  A=USR(1536,ADR(SHAPE$(37)),VERT,
      VERT-2):VERT=VERT-2:GOTO 4220
4160  A=USR(1536,ADR(SHAPE$(46)),VERT,
      VERT-2):POKE HREG,HORIZ+2:VERT=V
      ERT-2:HORIZ=HORIZ+2:GOTO 4220
4180  POKE HREG,HORIZ+2:HORIZ=HORIZ+2:
      GOTO 4220
4200  A=USR(1536,ADR(SHAPE$(64)),VERT,
      VERT+2):POKE HREG,HORIZ+2:VERT=V
      ERT+2:HORIZ=HORIZ+2
4220  IF HREG=53248 THEN VERT0=VERT:HO
      RIZ0=HORIZ:RETURN
4240  VERT1=VERT:HORIZ1=HORIZ:RETURN
4500  REM *** EXPLOSIONS ***
4520  EXPLO=65:FOR VOL=14 TO 0 STEP -2
      :SOUND 0,90,0,VOL:SOUND 1,100,4,
      VOL:POKE PLYR,62
4540  A=USR(MOVE,VERT,ADR(SHAPE$(EXPLO
      +9))):EXPLO=EXPLO+9:FOR I=1 TO 1
      5:NEXT I:NEXT VOL
4560  IF PLYR=704 THEN TANKØ=TANKØ-1
4580  IF PLYR=705 THEN TANK1=TANK1-1:G
      OTO 472Ø
46ØØ  IF TANKØ=2 THEN POSITION 2,11:?
      #6;"X":GOTO 468Ø
462Ø  IF TANKØ=1 THEN POSITION 1,11:?
      #6;"X":GOTO 468Ø
464Ø  POSITION 3,11:? #6;"X":POP :GOTO
       15ØØØ
468Ø  FOR I=1 TO 5ØØ:NEXT I:VERTØ=PMBA
      SE+572:HORIZØ=6Ø:POKE PLYR,74:PO
      KE HITCLR,Ø:POKE 53248,HORIZØ
47ØØ  A=USR(MOVE,VERTØ,ADR(SHAPE$(2Ø))
      ):RETURN
472Ø  IF TANK1=2 THEN POSITION 17,11:?
       #6;"X":GOTO 48ØØ
474Ø  IF TANK1=1 THEN POSITION 18,11:?
       #6;"X":GOTO 48ØØ
476Ø  POSITION 16,11:? #6;"X":POP :GOT
      O 15ØØØ
48ØØ  FOR I=1 TO 5ØØ:NEXT I:VERT1=PMBA
      SE+7ØØ:HORIZ1=185:POKE PLYR,136:
      POKE HITCLR,Ø:POKE 53249,HORIZ1
482Ø  A=USR(MOVE,VERT1,ADR(SHAPE$(56))
      ):RETURN
1ØØØØ  A=USR(MOVE,VERTØ,ADR(SHAPE$(2Ø)
       )):A=USR(MOVE,VERT1,ADR(SHAPE$(
       56)))
1ØØ15  X=INT(18*RND(Ø)+1):Y=INT(1Ø*RND
       (Ø)+1):LOCATE X,Y,Z:IF Z<>32 TH
       EN 1ØØ15
1ØØ2Ø  POSITION X,Y:? #6;"w":TANKØ=3:T
       ANK1=3:IF PEEK(53252)=2 OR PEEK
       (53253)=2 THEN 1ØØ15
1ØØ3Ø  HEADINGØ=3:HEADING1=7:SOUND Ø,1
       8Ø,6,3
1ØØ4Ø  SØ=STICK(Ø):S1=STICK(1):IF SØ=1
       1 THEN HEADINGØ=HEADINGØ-1
```

```
10060 IF S1=11 THEN HEADING1=HEADING1
      -1
10080 IF S0=7 THEN HEADING0=HEADING0+
      1
10100 IF S1=7 THEN HEADING1=HEADING1+
      1
10120 IF HEADING0<1 THEN HEADING0=8
10140 IF HEADING1<1 THEN HEADING1=8
10160 IF HEADING0>8 THEN HEADING0=1
10180 IF HEADING1>8 THEN HEADING1=1
10200 A=USR(MOVE,VERT0,ADR(SHAPE$((HE
      ADING0-1)*8+HEADING0+1)))
10240 A=USR(MOVE,VERT1,ADR(SHAPE$((HE
      ADING1-1)*8+HEADING1+1)))
10260 VERT=VERT0:HORIZ=HORIZ0:IF STIC
      K(0)=15 AND STICK(1)=15 THEN SO
      UND 1,180,6,3
10280 IF STICK(0)=14 THEN SOUND 1,120
      ,6,6:HEADING=HEADING0:HREG=5324
      8:POKE HITCLR,0:GOSUB 1000
10290 VERT0=VERT:HORIZ0=HORIZ:IF PEEK
      (53252)=2 OR PEEK(53260)=2 THEN
       PLYR=704:GOSUB 4500
10300 COLL=PEEK(53252):IF COLL=1 OR C
      OLL=5 OR COLL=9 THEN HEADING=HE
      ADING0:HREG=53248:VERT=VERT0:HO
      RIZ=HORIZ0:GOSUB 4000
10310 VERT=VERT1:HORIZ=HORIZ1
10320 IF STICK(1)=14 THEN SOUND 1,120
      ,6,6:HEADING=HEADING1:HREG=5324
      9:POKE HITCLR,0:GOSUB 1000
10340 VERT1=VERT:HORIZ1=HORIZ:IF PEEK
      (53253)=2 OR PEEK(53261)=1 THEN
       PLYR=705:GOSUB 4500
10360 COLL=PEEK(53253):IF COLL=1 OR C
      OLL=5 OR COLL=9 THEN HEADING=HE
      ADING1:HREG=53249:VERT=VERT1:HO
      RIZ=HORIZ1:GOSUB 4000
10700 IF STRIG(0)=1 THEN 10780
10740 POKE HITCLR,0:POKE 53252,0:MVER
      T=VERT0-512+384+3:MHORIZ=HORIZ0
      -1:HEADING=HEADING0:MREG=53252:
       GOSUB 2000
10750 POKE 53252,0:TRAP 40000
10760 IF PEEK(53256)=2 OR PEEK(53256)
      =3 THEN PLYR=705:VERT=VERT1:GOS
      UB 4500
10780 IF STRIG(1)=1 THEN 10040
10820 POKE HITCLR,0:POKE 53253,0:MVER
      T=VERT1-640+384+3:MHORIZ=HORIZ1
      -1:HEADING=HEADING1:MREG=53253:
      GOSUB 2000
10860 POKE 53253,0:TRAP 40000
10880 IF PEEK(53257)=1 OR PEEK(53257)
      =3 THEN PLYR=704:VERT=VERT0:GOS
      UB 4500
10900 GOTO 10040
11000 REM *** SETUP PM & ML ROUTINES
      ***
11020 GRAPHICS 2+16:? #6;"{5 SPACES}t
      ank duel":? #6:? #6;"
      {4 SPACES}PLEASE WAIT":? #6;"
      {5 SPACES}30 SECONDS"
11040 PM=PEEK(106)-8:POKE 54279,PM:PM
      BASE=256*PM:POKE 559,46:POKE 53
      277,3:POKE 623,4:POKE 53260,0:H
      ITCLR=53278
11060 FOR I=PMBASE+384 TO PMBASE+768:
      POKE I,0:NEXT I:POKE 704,74:POK
      E 705,136:VERT0=PMBASE+572:HORI
      Z0=60
11080 VERT1=PMBASE+700:HORIZ1=185:MVE
      RT=PMBASE+384:POKE 53248,HORIZ0
      :POKE 53249,HORIZ1
11100 REM VERTICAL POSITIONING ROUTIN
      E
11120 DATA 104,162,5,104,149,220,202,
      16,250,198,220,198,222,160,0,17
      7,224,170
11140 DATA 168,165,223,240,9,169,0,14
      5,222,136,208,249,138,168,165,2
      21,240,7,177,224,145,220,136,20
      8,249,96
11160 FOR I=1536 TO 1579:READ A:POKE
```

```
       I,A:NEXT I
11180 REM FLIP PLAYER SHAPE ROUTINE
11200 DIM MOVE$(25):MOVE=ADR(MOVE$):F
      OR I=1 TO 25:READ A:MOVE$(I,I)=
      CHR$(A):NEXT I
11220 DATA 104,104,133,204,104,133,20
      3,104,133,207,104,133,206,160,0
      ,177,206,145,203,200,192,8,208,
      247,96
11240 RETURN
12000 REM *** PLAYER SHAPES (CLOCKWIS
      E N-NE-E-SE-S-SW-W-NW & EXPLOSI
      ON) ***
12020 DATA 8,8,8,42,62,62,62,62,34
12040 DATA 8,9,26,60,127,254,60,24,16
12060 DATA 8,0,252,120,127,120,252,0,
      0
12080 DATA 8,16,24,60,254,127,60,26,9
12100 DATA 8,34,62,62,62,62,42,8,8
12120 DATA 8,8,24,60,127,254,60,88,14
      4
12140 DATA 8,0,63,30,254,30,63,0,0
12160 DATA 8,144,88,60,254,127,60,24,
      8
12180 DATA 8,0,0,8,28,28,8,0,0
12200 DATA 8,0,8,34,92,20,34,8,0
12220 DATA 8,8,65,4,168,20,1,64,8
12240 DATA 8,148,1,20,160,1,20,1,136
12260 DATA 8,145,74,32,130,65,2,84,13
      7
12280 DATA 8,72,1,64,0,130,1,8,82
12300 DATA 8,129,0,0,0,0,128,1,66
12320 DATA 8,0,0,0,0,0,0,0,0
12340 DIM SHAPE$(144)
12360 FOR I=1 TO 144:READ A:SHAPE$(I,
      I)=CHR$(A):NEXT I
12380 DIM MISSILE$(2):MISSILE$(1,1)=C
      HR$(1):MISSILE$(2,2)=CHR$(5)
12400 RETURN
13000 REM *** PLAYFIELD SETUP ***
13010 POKE 756,CHSET/256:POSITION 0,0
      :? #6;CHR$(125):POSITION 0,0
```

```
13020 ? #6;"J":COLOR ASC("M"):PLOT 1,
      0:DRAWTO 19,0:? #6;"N":COLOR AS
      C("O"):PLOT 19,1:DRAWTO 19,11
13040 COLOR ASC("M"):PLOT 19,11:POSIT
      ION 0,11:? #6;"MYYMMMMMMMMMMMM
      MZZ":COLOR ASC("V"):PLOT 0,10:D
      RAWTO 0,1
13060 SETCOLOR 0,2,10:SETCOLOR 1,0,0:
      SETCOLOR 2,13,10:SETCOLOR 3,12,
      8
13080 POSITION 4,1:? #6;"Su St T t TS
      "
13100 POSITION 2,2:? #6;"T{3 SPACES}S
      t Tu UtsL"
13120 POSITION 5,3:? #6;"St EU S  s I
      "
13140 POSITION 7,4:? #6;"Su St T"
13180 POSITION 6,5:? #6;"U LJKF
      {4 SPACES}T"
13200 POSITION 2,7:? #6;"L"
13220 POSITION 4,8:? #6;"B  FD GHC  E
      T L"
13240 POSITION 4,9:? #6;"T{5 SPACES}S
      "
13260 POSITION 3,10:? #6;"L
      {3 SPACES}s T{3 SPACES}s U  T":
      RETURN
14000 REM *** REDEFINE CHARACTERS ***
14020 CHSET=(PEEK(106)-4)*256:FOR I=0
       TO 512:POKE CHSET+I,PEEK(57344
      +I):NEXT I
14040 RESTORE 14120
14060 READ A:IF A=-1 THEN RETURN
14080 FOR J=0 TO 7:READ B:POKE CHSET+
      A*8+J,B:NEXT J
14100 GOTO 14060
14120 DATA 33,0,24,60,126,255,90,126,
      0
14140 DATA 34,0,24,60,126,195,74,126,0
14160 DATA 35,62,42,62,42,62,42,62,58
14180 DATA 36,24,24,60,44,126,90,126,
      90
```

```
14200 DATA 37,0,0,0,0,4,255,118,94
14220 DATA 38,3,3,3,3,255,171,255,171
14240 DATA 39,127,85,127,85,127,85,12
      7,106
14260 DATA 40,255,85,255,85,255,85,25
      5,171
14280 DATA 41,16,56,40,56,40,124,108,
      238
14300 DATA 42,136,170,170,170,170,170
      ,255,255
14320 DATA 43,130,170,170,170,170,170
      ,254,254
14340 DATA 44,0,126,86,126,86,126,86,
      86
14360 DATA 45,255,255,0,0,0,0,0,0
14380 DATA 46,255,255,3,3,3,3,3,3
14400 DATA 47,3,3,3,3,3,3,3,3
14420 DATA 48,3,3,3,3,3,3,255,255
14440 DATA 49,0,0,0,0,0,0,255,255
14460 DATA 50,192,192,192,192,192,192
      ,255,255
14480 DATA 51,0,0,16,56,124,16,16,0
14500 DATA 52,8,28,62,127,62,8,8,0
14520 DATA 53,8,28,28,62,62,127,8,8
14540 DATA 54,192,192,192,192,192,192
      ,192,192
14560 DATA 55,1,64,0,0,0,0,2,128
14580 DATA 56,255,255,0,8,28,8,8,8
14600 DATA 57,255,255,0,252,120,127,1
      20,252
14620 DATA 58,255,255,0,63,30,254,30,
      63
14640 DATA 59,192,96,48,24,12,6,3,1
14660 DATA 61,255,255,192,192,192,192
      ,192,192
14680 DATA -1
15000 REM *** RESET GAME ***
15020 FOR I=255 TO 0 STEP -2:SOUND 0,
      I,10,6:POKE 704,I:POKE 705,I:NE
      XT I:SOUND 0,0,0,0:POKE 704,0:P
      OKE 705,0
15040 POSITION 0,0:? #6;CHR$(125):POK
```

```
      E 756,224:POSITION 2,3:? #6;"TO
       PLAY AGAIN":? #6;" PRESS FIRE
      BUTTON"
15Ø6Ø ? #6;"{4 SPACES}ON JOYSTICK"
15Ø8Ø IF STRIG(Ø)=1 AND STRIG(1)=1 TH
      EN 15Ø8Ø
151ØØ POKE HITCLR,Ø:A=USR(MOVE,VERTØ,
      ADR(SHAPE$(137))):A=USR(MOVE,VE
      RT1,ADR(SHAPE$(137)))
1512Ø POKE 7Ø4,74:POKE 7Ø5,136:VERTØ=
      PMBASE+572:VERT1=PMBASE+7ØØ:MVE
      RT=PMBASE+384:HORIZØ=6Ø:HORIZ1=
      185
1514Ø POKE 53248,HORIZØ:POKE 53249,HO
      RIZ1:GOTO 8Ø
```

Program 3-3. Tank Duel — BASIC A+

```
1Ø REM *** TANKDUEL ***
2Ø REM OSS BASIC A+ VERSION
3Ø REM
4Ø GRAPHICS 2+16:? #6;"{5 SPACES}tan
   k duel":? #6:? #6;"{4 SPACES}PLEA
   SE WAIT":? #6;"{5 SPACES}3Ø SECON
   DS"
9Ø GOSUB 13ØØØ
1ØØ GOSUB 9ØØØ
11Ø GOSUB 3ØØØ
12Ø GOTO 1ØØØØ
1ØØØ REM *** PLAYER Ø COLLISIONS ***
1Ø1Ø POKE 77,Ø
1Ø2Ø IF BUMP(Ø,1) OR BUMP(Ø,9) THEN
     1Ø6Ø
1Ø4Ø GOTO 122Ø
1Ø6Ø DAT=57:FOR VOL=14 TO Ø STEP -2:
     SOUND Ø,9Ø,Ø,VOL:SOUND 1,1ØØ,4,
     VOL:PMCOLOR Ø,3,14
1Ø8Ø{3 SPACES}MOVE ADR(SHAPE$(DAT+8)
     ),PMADR(Ø)+VERTØ,6:DAT=DAT+8:FO
     R I=1 TO 5Ø:NEXT I:NEXT VOL:TAN
     KØ=TANKØ-1
```

```
1120 IF TANKØ=2 THEN POSITION 2,11:?
      #6;"X":GOTO 1180
1140 IF TANKØ=1 THEN POSITION 1,11:?
      #6;"X":GOTO 1180
1160 IF TANKØ<1 THEN POSITION 3,11:?
      #6;"X":POP :GOTO 12000
1180 FOR I=1 TO 1000:NEXT I:VERTØ=50
     :HORIZØ=60:PMCOLOR Ø,4,6:PMMOVE
      Ø,HORIZØ
1200 MOVE ADR(SHAPE$(25)),PMADR(Ø)+V
     ERTØ,6:POKE 53278,Ø:RETURN
1220 IF BUMP(Ø,8) THEN POKE 53278,Ø:
     ON HEADINGØ GOTO 1260,1280,1300
     ,1320,1340,1360,1380,1400
1240 PMCLR 4:GOTO 2060
1260 PMMOVE Ø;-2:VERTØ=VERTØ+2:RETUR
     N
1280 PMMOVE Ø,HORIZØ-2;-2:HORIZØ=HOR
     IZØ-2:VERTØ=VERTØ+2:RETURN
1300 PMMOVE Ø,HORIZØ-2:HORIZØ=HORIZØ
     -2:RETURN
1320 PMMOVE Ø,HORIZØ-2;2:HORIZØ=HORI
     ZØ-2:VERTØ=VERTØ-2:RETURN
1340 PMMOVE Ø;2:VERTØ=VERTØ-2:RETURN

1360 PMMOVE Ø,HORIZØ+2;2:HORIZØ=HORI
     ZØ+2:VERTØ=VERTØ-2:RETURN
1380 PMMOVE Ø,HORIZØ+2:HORIZØ=HORIZØ
     +2:RETURN
1400 PMMOVE Ø,HORIZØ+2;-2:HORIZØ=HOR
     IZØ+2:VERTØ=VERTØ+2:RETURN
2000 REM *** PLAYER 1 COLLISIONS ***
2020 IF BUMP(1,Ø) OR BUMP(1,9) THEN
     2060
2040 GOTO 2220
2060 DAT=57:FOR VOL=14 TO Ø STEP -2:
     SOUND Ø,90,Ø,VOL:SOUND 1,100,4,
     VOL:PMCOLOR 1,3,14
2080{3 SPACES}MOVE ADR(SHAPE$(DAT+8)
     ),PMADR(1)+VERT1,6:DAT=DAT+8:FO
     R I=1 TO 50:NEXT I:NEXT VOL:TAN
     K1=TANK1-1
```

```
2120 IF TANK1=2 THEN POSITION 17,11:
     ? #6;"X":GOTO 2180
2140 IF TANK1=1 THEN POSITION 18,11:
     ? #6;"X":GOTO 2180
2160 IF TANK1<1 THEN POSITION 16,11:
     ? #6;"X":POP :GOTO 12000
2180 FOR I=1 TO 1000:NEXT I:VERT1=50
     :HORIZ1=185:PMCOLOR 1,7,8:PMMOV
     E 1,HORIZ1
2200 MOVE ADR(SHAPE$(57)),PMADR(1)+V
     ERT1,6:POKE 53278,0:RETURN
2220 IF BUMP(1,8) THEN POKE 53278,0:
     ON HEADING1 GOTO 2260,2280,2300
     ,2320,2340,2360,2380,2400
2240 PMCLR 5:GOTO 1060
2260 PMMOVE 1;-2:VERT1=VERT1+2:RETUR
     N
2280 PMMOVE 1,HORIZ1-2;-2:HORIZ1=HOR
     IZ1-2:VERT1=VERT1+2:RETURN
2300 PMMOVE 1,HORIZ1-2:HORIZ1=HORIZ1
     -2:RETURN
2320 PMMOVE 1,HORIZ1-2;2:HORIZ1=HORI
     Z1-2:VERT1=VERT1-2:RETURN
2340 PMMOVE 1;2:VERT1=VERT1-2:RETURN

2360 PMMOVE 1,HORIZ1+2;2:HORIZ1=HORI
     Z1+2:VERT1=VERT1-2:RETURN
2380 PMMOVE 1,HORIZ1+2:HORIZ1=HORIZ1
     +2:RETURN
2400 PMMOVE 1,HORIZ1+2;-2:HORIZ1=HOR
     IZ1+2:VERT1=VERT1+2:RETURN
3000 REM *** PLAYFIELD SETUP ***
3060 ? #6;CHR$(125):POSITION 0,0:POK
     E 756,CHSET/256
3080 ? #6;"J":COLOR ASC("M"):PLOT 1,
     0:DRAWTO 19,0:? #6;"N":COLOR AS
     C("O"):PLOT 19,1:DRAWTO 19,11
3100 COLOR ASC("M"):PLOT 19,11:POSIT
     ION 0,11:? #6;"MYYMMMMMMMMMMMMM
     MZZ":COLOR ASC("V"):PLOT 0,10:D
     RAWTO 0,1
3110 SETCOLOR 0,2,10:SETCOLOR 1,0,0:
     SETCOLOR 2,13,10:SETCOLOR 3,12,
     8
3120 POSITION 4,1:? #6;"Su St  T  t  Ts
     "
3140 POSITION 2,2:? #6;"T{3 SPACES}S
     t Tu  UtSL"
3160 POSITION 5,3:? #6;"St EU S    s I
     "
3180 POSITION 7,4:? #6;"Su St  T"
3200 POSITION 3,5:? #6;"A   t
     {3 SPACES}t S"
3220 POSITION 6,6:? #6;"U LJKF
     {4 SPACES}T"
3240 POSITION 2,7:? #6;"L"
3260 POSITION 2,8:? #6;"B"
3280 POSITION 6,9:? #6;"FD GHC   ET  
     A"
3300 POSITION 3,10:? #6;"Ut
     {10 SPACES}U sT"
3999 RETURN
9000 REM *** PLAYER SHAPES (CLOCKWIS
     E N-NE-E-SE-S-SW-W-NW & EXPLOSI
     ON) ***
9020 DATA 8,8,42,62,62,62,62,34
9040 DATA 9,26,60,127,254,60,24,16
9060 DATA 0,252,120,127,120,252,0,0
9080 DATA 16,24,60,254,127,60,26,9
9100 DATA 34,62,62,62,62,42,8,8
9120 DATA 8,24,60,127,254,60,88,144
9140 DATA 0,63,30,254,30,63,0,0
9160 DATA 144,88,60,254,127,60,24,8
9180 DATA 0,0,8,28,28,8,0,0
9200 DATA 0,8,34,92,20,34,8,0
9220 DATA 8,65,4,168,20,1,64,8
9240 DATA 148,1,20,160,1,20,1,136
9260 DATA 145,74,32,130,65,2,84,137
9280 DATA 72,1,64,0,130,1,8,82
9300 DATA 129,0,0,0,0,128,1,66
9320 DATA 0,0,0,0,0,0,0,0
9340 DIM SH$(1),SHAPE$(128)
9360 RESTORE :FOR I=1 TO 128:READ SH
```

```
     APE:SH$=CHR$(SHAPE):SHAPE$(I,I)
     =SH$:NEXT I
9380 RETURN
10000 PMGRAPHICS 2:PMCLR Ø:PMCLR 4:P
      MCLR 1:PMCLR 5:PMCOLOR Ø,4,6:P
      MCOLOR 1,7,8:POKE 623,4
10025 VERTØ=5Ø:VERT1=5Ø:HORIZØ=6Ø:HO
      RIZ1=185:TANKØ=3:TANK1=3:PMMOV
      E Ø,HORIZØ:PMMOVE 1,HORIZ1
10030 MOVE ADR(SHAPE$(9)),PMADR(Ø)+V
      ERTØ,6:MOVE ADR(SHAPE$(9)),PMA
      DR(1)+VERT1,6:MINES=Ø
10032 X=INT(18*RND(Ø)+1):Y=INT(1Ø*RN
      D(Ø)+1):LOCATE X,Y,Z:IF Z<>32
      THEN 1ØØ32
10034 POSITION X,Y:? #6;"w":IF X=1 O
      R X=2 OR X=17 OR X=18 AND Y=4
      OR Y=5 THEN POSITION X,Y:? #6;
      " ":GOTO 1ØØ32
10040 PMCLR 5:HØ=HSTICK(Ø):H1=HSTICK
      (1):HEADINGØ=HEADINGØ+HØ:HEADI
      NG1=HEADING1+H1:IF HEADINGØ<1
      THEN HEADINGØ=8
10060 IF HEADING1<1 THEN HEADING1=8
10080 IF HEADINGØ>8 THEN HEADINGØ=1
10100 IF HEADING1>8 THEN HEADING1=1
10105 POKE 53278,Ø
10110 ON HEADINGØ GOTO 1Ø12Ø,1Ø14Ø,1
      Ø16Ø,1Ø18Ø,1Ø2ØØ,1Ø22Ø,1Ø24Ø,1
      Ø26Ø
10120 MOVE ADR(SHAPE$(1)),PMADR(Ø)+V
      ERTØ,6:GOTO 1Ø28Ø
10140 MOVE ADR(SHAPE$(9)),PMADR(Ø)+V
      ERTØ,7:GOTO 1Ø28Ø
10160 MOVE ADR(SHAPE$(17)),PMADR(Ø)+
      VERTØ,6:GOTO 1Ø28Ø
10180 MOVE ADR(SHAPE$(25)),PMADR(Ø)+
      VERTØ,6:GOTO 1Ø28Ø
10200 MOVE ADR(SHAPE$(33)),PMADR(Ø)+
      VERTØ,6:GOTO 1Ø28Ø
10220 MOVE ADR(SHAPE$(41)),PMADR(Ø)+
      VERTØ,7:GOTO 1Ø28Ø
```

```
10240 MOVE ADR(SHAPE$(49)),PMADR(0)+
      VERTØ,6:GOTO 10280
10260 MOVE ADR(SHAPE$(57)),PMADR(0)+
      VERTØ,6
10280 IF BUMP(0,1) OR BUMP(0,9) THEN
       GOSUB 1000
10290 ON HEADING1 GOTO 10300,10320,1
      0340,10360,10380,10400,10420,1
      0430
10300 MOVE ADR(SHAPE$(1)),PMADR(1)+V
      ERT1,6:GOTO 10440
10320 MOVE ADR(SHAPE$(9)),PMADR(1)+V
      ERT1,7:GOTO 10440
10340 MOVE ADR(SHAPE$(17)),PMADR(1)+
      VERT1,6:GOTO 10440
10360 MOVE ADR(SHAPE$(25)),PMADR(1)+
      VERT1,6:GOTO 10440
10380 MOVE ADR(SHAPE$(33)),PMADR(1)+
      VERT1,6:GOTO 10440
10400 MOVE ADR(SHAPE$(41)),PMADR(1)+
      VERT1,7:GOTO 10440
10420 MOVE ADR(SHAPE$(49)),PMADR(1)+
      VERT1,6:GOTO 10440
10430 MOVE ADR(SHAPE$(57)),PMADR(1)+
      VERT1,6
10440 IF BUMP(1,0) OR BUMP(1,9) THEN
       GOSUB 2000
10445 IF VSTICK(0)=1 THEN SOUND 1,12
      0,6,6:ON HEADINGØ GOTO 10460,1
      0480,10500,10520,10540,10560,1
      0580,10600
10450 SOUND 1,180,6,3:GOTO 10620
10460 PMMOVE 0;2:VERTØ=VERTØ-2:GOTO
      10620
10480 PMMOVE 0,HORIZØ+2;2:HORIZØ=HOR
      IZØ+2:VERTØ=VERTØ-2:GOTO 10620
10500 PMMOVE 0,HORIZØ+2:HORIZØ=HORIZ
      Ø+2:GOTO 10620
10520 PMMOVE 0,HORIZØ+2;-2:HORIZØ=HO
      RIZØ+2:VERTØ=VERTØ+2:GOTO 10620
10540 PMMOVE 0;-2:VERTØ=VERTØ+2:GOTO
       10620
```

```
10560 PMMOVE 0,HORIZ0-2;-2:HORIZ0=HO
      RIZ0-2:VERT0=VERT0+2:GOTO 1062
      0
10580 PMMOVE 0,HORIZ0-2:HORIZ0=HORIZ
      0-2:GOTO 10620
10600 PMMOVE 0,HORIZ0-2;2:HORIZ0=HOR
      IZ0-2:VERT0=VERT0-2
10620 IF BUMP(0,1) OR BUMP(0,8) OR B
      UMP(0,9) THEN GOSUB 1000
10625 IF VSTICK(1)=1 THEN SOUND 0,12
      0,6,6:ON HEADING1 GOTO 10640,1
      0660,10680,10700,10720,10740,1
      0760,10780
10630 SOUND 0,180,6,3:GOTO 10800
10640 PMMOVE 1;2:VERT1=VERT1-2:GOTO
      10800
10660 PMMOVE 1,HORIZ1+2;2:HORIZ1=HOR
      IZ1+2:VERT1=VERT1-2:GOTO 10800
10680 PMMOVE 1,HORIZ1+2:HORIZ1=HORIZ
      1+2:GOTO 10800
10700 PMMOVE 1,HORIZ1+2;-2:HORIZ1=HO
      RIZ1+2:VERT1=VERT1+2:GOTO 1080
      0
10720 PMMOVE 1;-2:VERT1=VERT1+2:GOTO
       10800
10740 PMMOVE 1,HORIZ1-2;-2:HORIZ1=HO
      RIZ1-2:VERT1=VERT1+2:GOTO 1080
      0
10760 PMMOVE 1,HORIZ1-2:HORIZ1=HORIZ
      1-2:GOTO 10800
10780 PMMOVE 1,HORIZ1-2;2:HORIZ1=HOR
      IZ1-2:VERT1=VERT1-2
10800 IF BUMP(1,0) OR BUMP(1,8) OR B
      UMP(1,9) THEN GOSUB 2000
10805 IF STRIG(0)=0 THEN POKE 53278,
      0:MHORIZ=HORIZ0:FOR I=15 TO 0
      STEP -1:SOUND 2,90,0,I:NEXT I:
      GOTO 10820
10810 GOTO 11000
10820 ON HEADING0 GOTO 10840,10860,1
      0880,10900,10920,10940,10960,1
      0980
```

```
10840 MISSILE Ø,VERTØ,1:PMMOVE 4,MHO
      RIZ+3;5Ø:GOTO 11ØØØ
10860 MISSILE Ø,VERTØ,1:FOR I=2 TO 4
      Ø STEP 2:PMMOVE 4,MHORIZ+6+I;2
      :NEXT I:GOTO 11ØØØ
10880 MISSILE Ø,VERTØ+3,1:FOR I=2 TO
       5Ø STEP 2:PMMOVE 4,MHORIZ+I:N
      EXT I:GOTO 11ØØØ
10900 MISSILE Ø,VERTØ+4,1:FOR I=2 TO
       4Ø STEP 2:PMMOVE 4,MHORIZ+3+I
      ;-2:NEXT I:GOTO 11ØØØ
10920 MISSILE Ø,VERTØ+4,1:PMMOVE 4,M
      HORIZ+3;-5Ø:GOTO 11ØØØ
10940 MISSILE Ø,VERTØ+4,1:FOR I=2 TO
       4Ø STEP 2:PMMOVE 4,MHORIZ+2-I
      ;-2:NEXT I:GOTO 11ØØØ
10960 MISSILE Ø,VERTØ+3,1:FOR I=2 TO
       5Ø STEP 2:PMMOVE 4,MHORIZ-I:N
      EXT I:GOTO 11ØØØ
10980 MISSILE Ø,VERTØ+3,1:FOR I=2 TO
       4Ø STEP 2:PMMOVE 4,MHORIZ+3-I
      ;2:NEXT I
11000 IF BUMP(4,1) THEN GOSUB 1ØØØ
11010 PMCLR 4:IF STRIG(1)=Ø THEN POK
      E 53278,Ø:FOR I=15 TO Ø STEP -
      1:SOUND 2,9Ø,Ø,I:NEXT I:GOTO 1
      1Ø4Ø
11020 GOTO 1ØØ4Ø
11040 ON HEADING1 GOTO 11Ø6Ø,11Ø9Ø,1
      112Ø,1115Ø,1118Ø,1121Ø,1124Ø,1
      127Ø
11060 MISSILE 1,VERT1,1:PMMOVE 5,HOR
      IZ1+3;5Ø:GOTO 113ØØ
11090 MISSILE 1,VERT1,1:FOR I=2 TO 4
      Ø STEP 2:PMMOVE 5,HORIZ1+6+I;2
      :NEXT I:GOTO 113ØØ
11120 MISSILE 1,VERT1+3,1:FOR I=2 TO
       5Ø STEP 2:PMMOVE 5,HORIZ1+I:N
      EXT I:GOTO 113ØØ
11150 MISSILE 1,VERT1+4,1:FOR I=2 TO
       4Ø STEP 2:PMMOVE 5,HORIZ1+3+I
      ;-2:NEXT I:GOTO 113ØØ
```

```
11180 MISSILE 1,VERT1+4,1:PMMOVE 5,H
      ORIZ1+3;-50:GOTO 11300
11210 MISSILE 1,VERT1+4,1:FOR I=2 TO
       40 STEP 2:PMMOVE 5,HORIZ1+2-I
      ;-2:NEXT I:GOTO 11300
11240 MISSILE 1,VERT1+3,1:FOR I=2 TO
       50 STEP 2:PMMOVE 5,HORIZ1-I:N
      EXT I:GOTO 11300
11270 MISSILE 1,VERT1+3,1:FOR I=2 TO
       40 STEP 2:PMMOVE 5,HORIZ1+3-I
      ;2:NEXT I
11300 IF BUMP(5,0) THEN GOSUB 2000
11320 GOTO 10040
12000 REM *** RESET GAME ***
12020 FOR I=255 TO 0 STEP -2:SOUND 0
      ,I,10,6:POKE 704,I:POKE 705,I:
      NEXT I:SOUND 0,0,0,0:POKE 704,
      0:POKE 705,0
12040 POSITION 0,0:? #6;CHR$(125):PO
      KE 756,224:POSITION 2,3:? #6;"
       TO PLAY AGAIN":? #6;" PRESS F
      IRE BUTTON"
12060 ? #6;"{5 SPACES}ON JOYSTICK"
12080 IF STRIG(0)=1 AND STRIG(1)=1 T
      HEN 12080
12100 GOTO 110
13000 REM *** REDEFINE CHARACTERS **
      *
13020 CHSET=(PEEK(106)-8)*256:FOR I=
      0 TO 512:POKE CHSET+I,PEEK(573
      44+I):NEXT I
13021 RESTORE 13025
13022 READ A:IF A=-1 THEN RETURN
13023 FOR J=0 TO 7:READ B:POKE CHSET
      +A*8+J,B:NEXT J
13024 GOTO 13022
13025 DATA 33,0,24,60,126,255,90,126
      ,0
13026 DATA 34,0,24,60,126,195,74,126
      ,0
13027 DATA 35,62,42,62,42,62,42,62,5
      8
```

```
13028 DATA 36,24,24,60,44,126,90,126
      ,90
13029 DATA 37,0,0,0,0,4,255,118,94
13030 DATA 38,3,3,3,3,255,171,255,17
      1
13031 DATA 39,127,85,127,85,127,85,1
      27,106
13032 DATA 40,255,85,255,85,255,85,2
      55,171
13033 DATA 41,16,56,40,56,40,124,108
      ,238
13034 DATA 42,136,170,170,170,170,17
      0,255,255
13035 DATA 43,130,170,170,170,170,17
      0,254,254
13036 DATA 44,0,126,86,126,86,126,86
      ,86
13037 DATA 45,255,255,0,0,0,0,0,0
13038 DATA 46,255,255,3,3,3,3,3,3
13039 DATA 47,3,3,3,3,3,3,3,3
13040 DATA 48,3,3,3,3,3,3,255,255
13041 DATA 49,0,0,0,0,0,0,255,255
13042 DATA 50,192,192,192,192,192,19
      2,255,255
13043 DATA 51,0,0,16,56,124,16,16,0
13044 DATA 52,8,28,62,127,62,8,8,0
13045 DATA 53,8,28,28,62,62,127,8,8
13046 DATA 54,192,192,192,192,192,19
      2,192,192
13047 DATA 55,1,64,0,0,0,0,2,128
13048 DATA 56,255,255,0,8,28,8,8,8
13049 DATA 57,255,255,0,252,120,127,
      120,252
13050 DATA 58,255,255,0,63,30,254,30
      ,63
13051 DATA 59,192,96,48,24,12,6,3,1
13052 DATA 61,255,255,192,192,192,19
      2,192,192
13053 DATA -1
```

Part Three

Pick-up Sticks

Jason Lex Thomas

This game demonstrates that good games can be written completely in BASIC. "Pick-up Sticks" doesn't even use the statements PEEK or POKE.

"Pick-up Sticks" requires two players. Each player uses one joystick. The screen displays 15 sticks and two players. The players will appear at random, but always immediately beside each other. A stick will begin to flash, indicating that it is the next target. The first player reaching the stick gets the point. The player who has the most points, when all 15 sticks are gone, wins. A complication is that if both players reach the stick at the same time, no one gets the point. Since both players start out at virtually the same place on the screen, this can play a deciding role.

You'll notice that when a player moves, he leaves a trail behind him. This does not affect the game. But it does let you draw pretty pictures if you like.

No PEEKs or POKEs

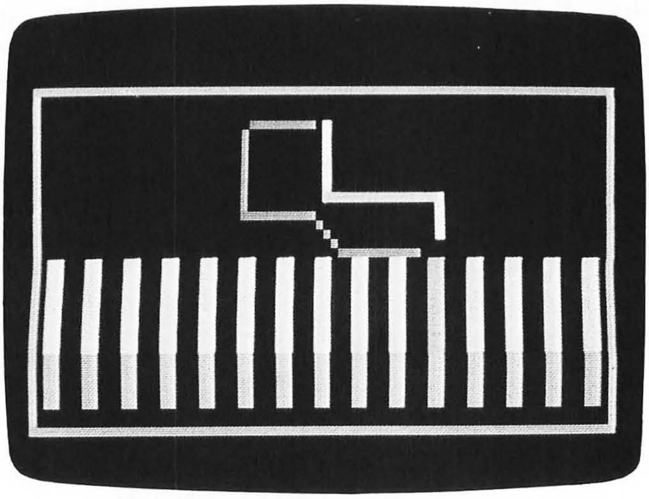
One of the first things I discovered about my computer was that Atari 8K BASIC is slow. It is so slow, in fact, that many games cannot be written without PEEKs or POKEs. And when it comes to moving images or players across a playfield, nothing comes in handier than a POKE or a USR command to call some machine language subroutine.

How do we get by with such a slow language? We could use PEEK and POKE or Assembler routines. But then there would be no reason for this article. The first criterion is to make the most of what you have. Don't get too fancy, either. It's been my experience that in BASIC, when you get fancy, you get slow. The second criterion that we must impose is that the more graphically oriented your program is, the more overhead involved by the computer. (Remember, it's not the amount of RAM used in a particular graphics mode that hurts you; it's the amount of coding required in your program to address it all.) Finally, when it comes to sound, experiment. Many good sound effects can be done by a simple SOUND command with various distortions. It may take awhile to achieve the results that you desire, but it can be done.

Here's proof: a program that I wrote called Pick-up Sticks. Nowhere in the entire program will you find one POKE or one USR statement. The game itself is quite simple and will run easily on any Atari.

Variable Usage Table

Variable	Usage
X$,Y$	Players' initials
A(15)	Array used to determine sticks' positions
XHOR,XVERT	Player X's horizontal and vertical position
YHOR,YVERT	Player Y's horizontal and vertical position
XDIR	Player X's current direction of travel
YDIR	Player Y's current direction of travel
NORTH,EAST,SOUTH,WEST	Playfield boundaries
CBLINK	Color cycle for blinking of sticks
SUBSC	Subscript for scanning through array A
MATCH	Used to determine which player picked up the stick
NUMMATCH	The number of sticks which have been picked up
XMAT	The number of sticks picked up by player X
YMAT	The number of sticks picked up by player Y
BLINK	Selected stick board position



Race to the sticks in "Pick-up Sticks."

Program 3-4. Pick-up Sticks

```
100 DIM X$(3),Y$(3),A(15):NORTH=5:EAS
    T=75:WEST=3:SOUTH=19:GOSUB 900
110 REM DRAW BORDER & INITIALIZE
120 GRAPHICS 21:COLOR 0:FOR J=1 TO 15
    :A(J)=J*5:NEXT J:XMAT=0:YMAT=XMAT
130 SETCOLOR 1,0,14
140 COLOR 1:PLOT 3,1:DRAWTO 78,1:DRAW
    TO 78,38:DRAWTO 3,38:DRAWTO 3,1
150 REM DRAW INITIAL BOARD
160 FOR J=5 TO 75 STEP 5:PLOT J,35:DR
    AWTO J,30:PLOT J+1,35:DRAWTO J+1,
    30:NEXT J
170 COLOR 2:FOR J=5 TO 75 STEP 5:PLOT
     J,29:DRAWTO J,20:PLOT J+1,29:DRA
    WTO J+1,20:NEXT J
180 XHOR=40:YHOR=XHOR+2:XVERT=5:YVERT
    =XVERT
190 SOUND 0,200,2,6
200 REM SELECT STICKS
210 GOSUB 530
220 REM PLAYERS
```

```
230 GOSUB 610
240 COLOR 2:PLOT XHOR,XVERT:COLOR 3:P
    LOT YHOR,YVERT
250 REM INPUT MOVE CHECK STICKS
260 XDIR=STICK(0):YDIR=STICK(1)
270 IF XDIR=14 THEN XVERT=XVERT-1
280 IF XDIR=6 THEN XVERT=XVERT-1:XHOR
    =XHOR+1
290 IF XDIR=13 THEN XVERT=XVERT+1
300 IF XDIR=5 THEN XVERT=XVERT+1:XHOR
    =XHOR+1
310 IF XDIR=11 THEN XHOR=XHOR-1
320 IF XDIR=9 THEN XHOR=XHOR-1:XVERT=
    XVERT+1
330 IF XDIR=7 THEN XHOR=XHOR+1
340 IF XDIR=10 THEN XHOR=XHOR-1:XVERT
    =XVERT-1
350 IF YDIR=14 THEN YVERT=YVERT-1
360 IF YDIR=6 THEN YVERT=YVERT-1:YHOR
    =YHOR+1
370 IF YDIR=13 THEN YVERT=YVERT+1
380 IF YDIR=5 THEN YVERT=YVERT+1:YHOR
    =YHOR+1
390 IF YDIR=11 THEN YHOR=YHOR-1
400 IF YDIR=9 THEN YHOR=YHOR-1:YVERT=
    YVERT+1
410 IF YDIR=7 THEN YHOR=YHOR+1
420 IF YDIR=10 THEN YHOR=YHOR-1:YVERT
    =YVERT-1
430 REM SET UP BOUNDARIES
440 IF XVERT<=WEST THEN XVERT=WEST
450 IF XVERT>=SOUTH THEN XVERT=SOUTH
460 IF YVERT<=WEST THEN YVERT=WEST
470 IF YVERT>=SOUTH THEN YVERT=SOUTH
480 IF XHOR<=NORTH THEN XHOR=NORTH
490 IF XHOR>=EAST THEN XHOR=EAST
500 IF YHOR<=NORTH THEN YHOR=NORTH
510 IF YHOR>=EAST THEN YHOR=EAST
520 GOTO 680
530 REM ROUTINE TO SELECT STICKS AND
    DETERMINE IF GAME OVER
```

```
540 CBLINK=1:SUBSC=INT(RND(0)*15)+1:C
    OUNT=0
550 FOR J=1 TO 15:IF A(J)<>0 THEN 570
560 NEXT J:GO TO 850
570 IF SUBSC>15 THEN SUBSC=1
580 BLINK=A(SUBSC):IF BLINK<5 THEN SU
    BSC=SUBSC+1:GOTO 570
590 A(SUBSC)=0
600 RETURN
610 REM BLINK STICKS
620 COLOR CBLINK
630 PLOT BLINK,29:DRAWTO BLINK,20
640 PLOT BLINK+1,29:DRAWTO BLINK+1,20
650 CBLINK=CBLINK+1
660 IF CBLINK>4 THEN CBLINK=1
670 RETURN
680 REM TEST TO SEE IF MATCH
690 MATCH=1
700 IF XVERT=19 AND (XHOR=BLINK OR XH
    OR=BLINK+1) THEN MATCH=MATCH+1
710 IF YVERT=19 AND (YHOR=BLINK OR YH
    OR=BLINK+1) THEN MATCH=MATCH+2
720 IF MATCH=1 THEN GOTO 220
730 REM BLANK ITEM
740 COLOR 0:PLOT BLINK,29:DRAWTO BLIN
    K,20
750 PLOT BLINK+1,29:DRAWTO BLINK+1,20
760 COLOR MATCH
770 PLOT BLINK,30:DRAWTO BLINK,35
780 PLOT BLINK+1,30:DRAWTO BLINK+1,35
790 IF MATCH=2 THEN XMAT=XMAT+1
800 IF MATCH=3 THEN YMAT=YMAT+1
810 FOR J=1 TO 250 STEP 5:SOUND 0,J,1
    0,10:NEXT J:SOUND 0,0,0,0:COLOR 0
820 FOR J=3 TO 19:PLOT 5,J:DRAWTO 75,
    J:NEXT J
830 NUMMATCH=NUMMATCH+1:IF NUMMATCH=1
    5 THEN 850
840 XVERT=INT(RND(0)*15)+3:YVERT=XVER
    T:XHOR=INT(RND(0)*50)+5:YHOR=XHOR
    +2:GOTO 190
850 REM GAME OVER
```

```
860 FOR J=1 TO 10:SOUND 0,20*J,10,10:
    FOR K=1 TO 10:NEXT K:SOUND 0,0,0,
    0:FOR K=1 TO 15:NEXT K:NEXT J
870 GRAPHICS 18:POSITION 5,1:? #6;"GA
    ME OVER":? #6
880 POSITION 2,3:? #6;X$;"  ";XMAT:POS
    ITION 11,3:? #6;Y$;"  ";YMAT
890 FOR J=1 TO 200:NEXT J:GOSUB 910:G
    OTO 110
900 GRAPHICS 18:POSITION 1,1:? #6;"PI
    CKUP STICKS"
910 POSITION 1,7:? #6;"PRESS FIRE TO
    PLAY":FOR WAIT=1 TO 30:NEXT WAIT
920 IF STRIG(0)<>STRIG(1) THEN 940
930 POSITION 1,7:? #6;"PRESS FIRE TO
    PLAY":FOR WAIT=1 TO 30:NEXT WAIT:
    GOTO 910
940 GRAPHICS 0:? "ENTER INITIALS OF F
    IRST PLAYER":INPUT X$
950 ? "ENTER INITIALS OF SECOND PLAYE
    R":INPUT Y$:RETURN
```

Poker Solitaire

Allen R. Breon

This strategy game is for one or two players, although more can play if your machine has more than 16K memory.

"Poker Solitaire" is a strategy game played with a standard deck of cards. The object is to create the best ten poker hands possible within a five-by-five matrix, using the first 25 cards in the deck. These are dealt one at a time and, once positioned in the matrix, cannot be moved. When 25 cards have been played, each hand of five cards (five across, five down) is scored; the final score is the sum of the ten individual hands.

The version presented here requires 13K of free RAM, without the REM statements, which are provided only to clarify the listing. One or two persons can play, although more players can easily be added if enough RAM is available.

Playing the Game

After typing RUN, you will see a title screen while initialization is taking place. When the game has been set up, the option/score screen will appear. The first time you see this screen, the top half will be blank. The bottom half will contain the options.

Only the option that is blinking can be changed. To change the value, execute the command, or view a screen, press the trigger of the joystick plugged into port one or press the OPTION console key. To move to another option, move the joystick in any direction or press the SELECT console key. You may start the game at any time by pressing START.

The options are fairly obvious, except for "DECKS." If there is more than one player, this option allows each player to have either a unique set of cards or the same sequence as the other players. In the former case, the cards received by one player have no bearing on the cards dealt to the other players — they are not being dealt from the same "deck." In the latter case, players will see who gets the highest score using the same cards.

Part Three

The Play Begins

After you have indicated you wish to START the game, there will be a delay of a few seconds, followed by the playing screen of the first player. This will consist of a five-by-five matrix, the notice PLAYER 1 vertically along the right-hand side, and the first card displayed in the lower right-hand corner.

The colored rectangle is a cursor that can be moved in any direction by the joystick. When you find a location you like, press the joystick trigger. The card will appear at that position and also remain at the bottom of the screen.

If you change your mind, simply move the joystick to reposition the card. Once you are satisfied with the location, press the trigger a second time to finalize your selection.

If there is more than one player, the next player's screen will immediately appear. Play continues until 24 rectangles have been filled for each player. It is not necessary to do anything to position the 25th card. It will be placed in the remaining location automatically. You can hold the trigger to view the screen; otherwise the next screen comes up automatically. Release the trigger to continue.

Scoring

After the final card for the last player has been positioned, the first player's screen will appear and be scored. The final score will appear in the lower right-hand corner. When the screen has been scored, either move the joystick in any direction or press the trigger to bring the next player's screen to be scored.

These are the scores for each possible hand:

HAND	SCORE	HAND	SCORE
One Pair	2	Two Pair	5
Three of a Kind	10	Straight	15
Flush	20	Full House	25
Four of a Kind	50	Straight Flush	75
Royal Flush	100		

After the scoring of the final player, moving the joystick or pressing the trigger will cause the option/score screen to appear. In addition to the options listed before, this screen will now contain each player's score for the game just completed, plus the highest score since RUN. By using the joystick or the console keys, you can review these screens. Press the BREAK key to end the game and return to BASIC.

Any time joystick or console input is required, you can instead press START to begin a new game, or OPTION or SELECT to return to the option/score screen.

The Program

The program is divided into five general areas: the game itself, the scoring routine, the options routine, initialization, and character set redefinition. The game and the scoring routines, since they are the most frequently used, are located near the beginning of the program to take advantage of the faster execution speed for statements near the beginning. Also near the beginning is a subroutine used by several of the routines for blinking displays while checking for input.

Extensive use is made of the capability to relocate the display and write screens. The location of the pointer for the display list is figured by calculating PEEK(560) + PEEK(561) *256 +4. The start of screen memory is the two-byte address beginning at location 88.

Changing the Screen

Each player in this game has his or her own playing screen. In addition, two screens are used to display the option/score data and to retain the high-score hands. Redrawing these screens each time they are needed would be time-consuming. Instead, each screen has its own memory location, which is stored in an array. When a screen is required, the appropriate numbers are POKEd into the pointers and the screen appears.

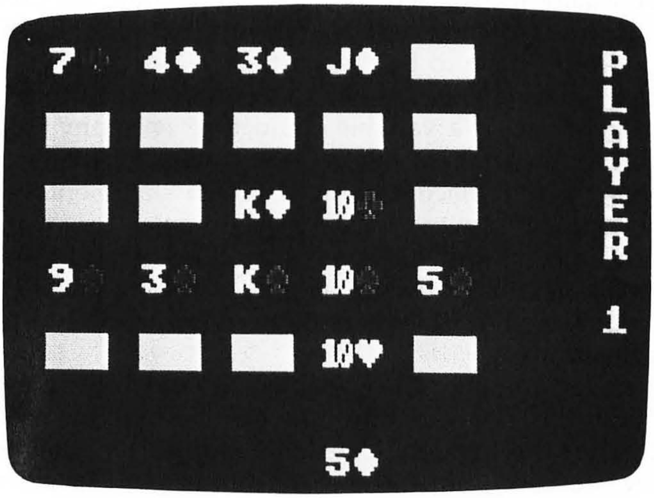
The screens are stored below RAMTOP (PEEK(106)*256). The first four pages below RAMTOP are used for the redefined character set. Each page holds 256 bytes of information. We need to set aside 240 bytes for each Graphics 2 screen we need. Each screen will begin on an even page. This way the least significant byte of the pointers will be zero, making housekeeping chores easier.

Before anything is stored in the area we have set aside, we must check to see if there is enough memory to hold everything. If the end of the program — PEEK(144) + PEEK(145)*256 — extends into the memory for the screens, the program will stop.

Part Three

Memory Requirements

Memory requirements change with the number of players that could play. With the REM statements removed, the program requires 13K RAM with two players. Each player beyond the first one uses about 900 additional bytes. Up to five people can play without affecting the game at all. Nine players will cause the option/score screen to display incorrectly, but will not affect the playing of the game. The variable MAXPL in line 4010 sets the maximum number of players. Set this to whatever your system can handle.



Where to put the five of diamonds??

The character set is redefined in lines 5000-5110. Graphics 2 displays can use only half the character set. We need uppercase and non-inverse characters most of the time. The machine language routine in line 5020 copies one page of memory. It is used to copy the first two pages of the ROM character set to the memory location set up in RAM. It is also used to copy a new high score display.

We redefine four unused characters to be the suit characters. These are copied directly from ROM. Also, we create a single character 10, designed using SuperFont (see

COMPUTE!'s First Book of Atari Graphics), and an inverse blank by turning on all bits.

Dealing the Cards

This program "deals" the cards by randomly choosing a number between one and 13 for the value and a second number between one and four for the suit. In order to make certain that no card is used more than once, these numbers are mapped into a 52-element array to represent each card. If the card has been used, a new value and suit are chosen. Note that while this is a good method for choosing cards for this game, it would be inadequate if all the cards in the deck were eventually to be used, because there could be problems finding an unused card near the end of the deck. In that case the deck would have to be "reshuffled."

Logical assignments are used to reduce the number of IF-THEN statements. If a variable could take on many values depending on several conditions, then memory requirements are decreased by replacing many IF-THEN statements with one line containing many logical assignments. This technique is used in lines 90 and 100, for instance, to determine the new cursor position caused by the motion of the joystick. These two lines replaced 12 statements otherwise needed to test all joystick positions.

Whenever possible, the variables N0 and N1 are used to replace 0 and 1. The variables take up one byte for each occurrence after the first; the constants require six bytes every time they are used. Since there are more than 250 occurrences, this is a considerable savings.

Program 3-5. Poker Solitaire

```
8 REM COMMENTS NOT NEEDED FOR PROGRAM
   EXECUTION
9 REM LEAST-USED ROUTINES AT END OF P
  ROGRAM
10 CLR :GOSUB 4000:GOTO 3000
15 REM BEGIN GAME
20 FOR J=N1 TO 25
25 REM RESET ATTRACT MODE
26 REM RANDOMLY CHOOSE VALUE AND SUIT
```

```
30 POKE 77,NØ:F=NØ:C=INT(RND(NØ)*13)+
   N1:S=INT(RND(NØ)*4)+N1:IF PLAY((S-
   N1)*13+C-1,DK-1)=N1 THEN 3Ø
35 REM RECORD THAT CARD HAS BEEN USED
36 REM IF SUIT=HEARTS OR DIAMONDS FLA
   G F WILL CAUSE SUIT TO BE PRINTED
   IN RED
4Ø PLAY((S-N1)*13+C-1,DK-1)=N1:IF S=N
   1 OR S=3 THEN F=32
5Ø POSITION 1Ø,11:? #6;CD$(C,C);CHR$(
   ASC(SUIT$(S,S))+F):POSITION X,Y:?
   #6;"@@";:POSITION X,Y
6Ø FOR I=N1 TO 25:NEXT I
7Ø STK=(PL-N1)*(STKS>N1)
8Ø A=STICK(STK):IF A=15 THEN 24Ø
85 REM DETERMINE NEW X-Y COORDINATES
9Ø X=X+3*((A>4) AND (A<8))-3*((A>8) A
   ND (A<12)):X=X*(X>=N1 AND X<=13)+1
   3*(X<N1)+(X>13)
1ØØ Y=Y+2*((A=5) OR (A=9) OR (A=13))-
    2*((A=6) OR (A=1Ø) OR (A=14)):Y=Y
    *(Y>=NØ AND Y<=8)+8*(Y<NØ)
1Ø5 REM CHECK IF NEW LOCATION IS FREE
1Ø6 REM IF NOT, SET PARAMETERS TO CAL
    L SUBROUTINE AT 1ØØØ
1Ø7 REM CARD CURRENTLY AT LOCATION WI
    LL BLINK ON AND OFF UNTIL JOYSTIC
    K IS MOVED
11Ø IF MATC(INT(X/3),INT(Y/2)+5*(PL-N
    1))=NØ THEN ON FLAG GOTO 18Ø,19Ø
12Ø K=NØ:IF FLAG=N1 THEN ? #6;BL32$;B
    L32$;:FLAG=2
13Ø XSV(PL)=INT(X/3):YSV(PL)=INT(Y/2)
    +5*(PL-N1):CD=MATC(XSV(PL),YSV(PL
    )):SUIT=MATS(XSV(PL),YSV(PL))
14Ø IF SUIT=N1 OR SUIT=3 THEN K=32
15Ø S$(N1)=CD$(CD):S$(2)=CHR$(ASC(SUI
    T$(SUIT))+K):A$="  ":XPOS=X:YPOS=
    Y
```

```
155 REM IF RETURN FROM SUBROUTINE WAS
     DUE TO TRIGGER BEING PUSHED THEN
     RETURN TO SUBROUTINE
160 GOSUB 1000:IF OP=7 AND L=15 THEN
    160
170 A=L:ON INT(OP/2)+(OP=7) GOTO 3010
    ,3010,3220,90
175 REM IF NEW LOCATION IS FREE, MOVE
     CURSOR
180 ? #6;BL32$;BL32$;
190 FLAG=N1:FOR I=N0 TO 2:SOUND N1,10
    0-I,10,15:SOUND 2,I*15,11,15:NEXT
     I:SOUND N1,0,N0,N0:SOUND 2,N0,N0
    ,N0
200 POSITION X,Y:? #6;"@@":POSITION X
    ,Y
205 REM CHECK FOR INPUT FROM JOYSTICK
    , TRIGGER, OR CONSOLE BUTTONS
210 IF STICK(STK)<>15 THEN 60
215 REM CONSOLE BUTTONS CHANGE LOCATI
    ON 53279
216 REM 3: OPTION
217 REM 5: SELECT
218 REM 6: START
219 REM 7: NO BUTTON PUSHED
220 OP=PEEK(53279):IF OP<7 THEN ON IN
    T(OP/2) GOTO 3010,3010,3220
230 IF J=25 THEN 250
240 ON STRIG(STK)+1 GOTO 270,210
250 FOR I=N1 TO 30:IF STICK(STK)<>15
    THEN POP :GOTO 60
260 NEXT I
265 REM TRIGGER HAS BEEN PUSHED
266 REM PRINT CARD AT LOCATION
270 FOR I=N0 TO 15:SOUND N1,100-S*5,1
    0,15-I:NEXT I
280 ? #6;CD$(C,C);CHR$(ASC(SUIT$(S,S)
    )+F):POSITION X,Y
290 FOR I=N0 TO 15:SOUND N0,100-C,10,
    15-I:NEXT I:FOR I=N1 TO 10:NEXT I
300 SOUND N0,N0,N0,N0:SOUND N1,N0,N0,
    N0
```

```
305 REM CHECK FOR INPUT
310 FOR I=N1 TO 25:OP=PEEK(53279):IF
    OP<7 THEN POP :ON INT(OP/2) GOTO
    3010,3010,3220
320 NEXT I
330 IF STICK(STK)<>15 THEN 60
340 IF J=25 THEN 360
350 ON STRIG(STK)+N1 GOTO 380,310
355 REM ALLOWS SCREEN TO BE HELD AFTE
    R AUTOMATIC PLACEMENT OF 25TH CAR
    D
360 FOR I=N1 TO 30:IF STICK(STK)<>15
    THEN POP :GOTO 60
370 NEXT I
375 REM TRIGGER WAS PUSHED FOR FINAL
    PLACEMENT OF CARD
380 FOR I=N0 TO 15:SOUND N0,190-C*2,1
    0,15-I:NEXT I
390 MATC(INT(X/3),INT(Y/2)+5*(PL-N1))
    =C:MATS(INT(X/3),INT(Y/2)+5*(PL-N
    1))=S
400 SOUND N0,N0,N0,N0:SOUND N1,N0,N0,
    N0
405 REM FIND TOPMOST/LEFTMOST FREE PO
    SITION FOR CURSOR
410 FOR Y=5*(PL-N1) TO 4+5*(PL-N1):FO
    R X=N0 TO 4:IF MATC(X,Y)=N0 THEN
    POP :GOTO 430
420 NEXT X:NEXT Y
430 X=X*3+N1:Y=(Y-5*(PL-N1))*2
440 IF J=25 AND STRIG(STK)=N0 THEN 44
    0
445 REM SAVE STATISTICS FOR THIS PLAY
    ER
450 XSV(PL)=X:YSV(PL)=Y:PL=PL+N1:IF P
    L>NP THEN PL=N1
460 IF DKS>N1 THEN DK=PL
465 REM RELOAD STATISTICS FOR NEXT PL
    AYER
470 X=XSV(PL):Y=YSV(PL):POKE 89,SC2(P
    L):POKE START+N1,SC2(PL)
480 POSITION 10,11:? #6;"  ":IF PL=N1
     THEN 510
```

```
490 IF DKS>N1 THEN 3Ø
5ØØ GOTO 5Ø
51Ø NEXT J:GOTO 2ØØØ
995 REM SUBROUTINE FOR BLINKING
996 REM VALUE IS IN S$;A$ CONTAINS EN
    OUGH BLANKS TO COVER S$
997 REM X-Y COORDINATES IN XPOS-YPOS
998 REM CHECKS FOR INPUT WHILE ALTERN
    ATELY PRINTING S$ AND A$
999 REM RETURNS TO CALLING SUBROUTINE
     IF INPUT RECEIVED, UNLESS START
    IS INDICATED, IN WHICH CASE GAME
    IS STARTED
1ØØØ POSITION XPOS,YPOS:? #6;A$:I=STR
     IG(STK):L=STICK(STK):OP=PEEK(532
     79)
1Ø1Ø IF I=N1 AND L=15 AND OP=7 THEN 1
     Ø3Ø
1Ø2Ø ON OP GOTO 1Ø3Ø,1Ø3Ø,1Ø8Ø,1Ø3Ø,1
     Ø8Ø,321Ø,1Ø8Ø
1Ø3Ø FOR K=N1 TO 1Ø:I=STRIG(STK):L=ST
     ICK(STK):OP=PEEK(53279)
1Ø4Ø IF I=NØ OR L<>15 OR OP<7 THEN PO
     SITION XPOS,YPOS:? #6;S$:POP :GO
     TO 1Ø2Ø
1Ø5Ø NEXT K:POSITION XPOS,YPOS:? #6;S
     $:FOR K=N1 TO 4Ø:I=STRIG(STK):L=
     STICK(STK):OP=PEEK(53279)
1Ø6Ø IF I=NØ OR L<>15 OR OP<7 THEN PO
     P :GOTO 1Ø2Ø
1Ø7Ø NEXT K:GOTO 1ØØØ
1Ø8Ø IF OP<>7 THEN I=(OP<>3):L=13+(OP
     <>5)*2
1Ø9Ø POSITION XPOS,YPOS:? #6;S$
11ØØ IF (S$="start" AND I=NØ) OR (LEN
     (S$)=2) THEN RETURN
111Ø FOR K=5 TO N1 STEP -1:SOUND NØ,I
     *(1ØØ-K)+L*5,14,15:NEXT K:SOUND
     NØ,NØ,NØ,NØ:RETURN
1995 REM SCORING ROUTINE
1996 REM CLEAR AND RESET OPTION/SCORE
      SCREEN
```

```
2ØØØ POKE 89,SC2(MAXPL+N1):POSITION N
     Ø,NØ:FOR I=NØ TO 7:? #6;BLANK$;:
     NEXT I
2Ø1Ø S$=STR$(HS):GOSUB 249Ø:POSITION
     3,NP+1:? #6;"high score";CHR$(AS
     C(":")+96);"  ";S$
2Ø2Ø FOR SCS=N1 TO NP:STK=(SCS-N1)*(S
     TKS>N1):POKE 89,SC2(SCS):POKE ST
     ART+N1,SC2(SCS):POSITION 1Ø,11:?
      #6;"   "
2Ø3Ø SC=NØ:FOR Y=5*(SCS-N1) TO 4+(5*(
     SCS-1)):FOR X=N1 TO 13:EC(X)=NØ:
     NEXT X:FOR X=N1 TO 4:ES(X)=NØ:NE
     XT X
2Ø35 REM PLACE INFORMATION FROM ONE H
     ORIZONTAL HAND INTO ARRAYS FOR S
     CORING
2Ø4Ø FOR X=NØ TO 4:C=MATC(X,Y):EC(C)=
     EC(C)+N1:S=MATS(X,Y):ES(S)=ES(S)
     +N1:NEXT X:XS=16:YS=(Y-5*(SCS-1)
     )*2
2Ø45 REM GO TO EVALUATION SUBROUTINE,
      THEN TO DISPLAY SUBROUTINE
2Ø5Ø GOSUB 211Ø:GOSUB 229Ø:NEXT Y
2Ø55 REM VERTICAL HANDS SCORED
2Ø6Ø FOR X=NØ TO 4:FOR Y=N1 TO 13:EC(
     Y)=NØ:NEXT Y:FOR Y=N1 TO 4:ES(Y)
     =NØ:NEXT Y
2Ø7Ø FOR Y=5*(SCS-N1) TO 4+5*(SCS-N1)
2Ø8Ø C=MATC(X,Y):EC(C)=EC(C)+N1:S=MAT
     S(X,Y):ES(S)=ES(S)+N1:NEXT Y:XS=
     X*3+N1:YS=1Ø+(INT(X/2)*2=X)
2Ø9Ø YS=1Ø
21ØØ GOSUB 211Ø:GOSUB 229Ø:NEXT X:GOT
     O 237Ø
21Ø5 REM EVALUATION SUBROUTINE
21Ø6 REM CHECK FOR FLUSH
211Ø F=NØ:FOR I=N1 TO 4:IF ES(I)=5 TH
     EN F=N1
212Ø NEXT I
2125 REM CHECK NUMBER OF OCCURRENCES
     FOR EACH VALUE
```

```
2130 S=N0:P=N0:T=N0:FOR I=N1 TO 13:ON
      EC(I)+N1 GOTO 2140,2150,2180,22
     00,2190
2140 S=N0:GOTO 2220
2150 S=S+N1:IF S=5 THEN POP :GOTO 225
     0
2160 IF S=4 AND I=4 AND EC(13)=N1 THE
     N POP :GOTO 2250
2170 GOTO 2220
2175 REM TWO OF A KIND
2180 P=P+N1:S=N0:GOTO 2210
2190 POP :GOTO 2240
2195 REM THREE OF A KIND
2200 T=T+N1:S=N0
2205 REM CHECK FOR FULL HOUSE
2210 IF P*2+T*3=5 THEN SCORE=SCORES(3
     ):POP :GOTO 2280
2220 NEXT I
2225 REM SCORES FLUSH, ONE PAIR, TWO
     PAIR, THREE OF A KIND
2230 SCORE=SCORES(4)*(F=N1)+SCORES(8)
     *(P=N1)+SCORES(7)*(P=2)+SCORES(6
     )*(T=N1):GOTO 2280
2235 REM FOUR OF A KIND
2240 SCORE=SCORES(2):GOTO 2280
2250 IF I=13 THEN GOTO 2270
2255 REM STRAIGHT FLUSH OR STRAIGHT
2260 SCORE=SCORES(N1)*(F=N1)+SCORES(5
     )*(F<>N1):GOTO 2280
2265 REM ROYAL FLUSH OR STRAIGHT
2270 SCORE=SCORES(N0)*(F=N1)+SCORES(5
     )*(F<>N1)
2280 RETURN
2285 REM DISPLAY SCORE
2286 REM ROYAL FLUSH HAS SEPARATE DIS
     PLAY ROUTINE
2290 IF SCORE=SCORES(N0) THEN 2330
2300 FOR I=6*SCORE TO 2.7*SCORE STEP
     -2:SOUND N0,I,10,5:SOUND N1,I/2,
     12,6:NEXT I
2305 REM SCORE OF 0 GETS A RASPBERRY
2310 IF SCORE=N0 THEN FOR I=N1 TO 15:
     SOUND N1,204,2,14:NEXT I
2320 POSITION XS+(SCORE<10),YS:S$=STR
     $(SCORE):SOUND N1,N0,N0,N0:SOUND
      N0,N0,N0,N0:GOSUB 2490:? #6;S$:
     GOTO 2350
2330 FOR I=13 TO N1 STEP -1:FOR J=N1
     TO 13:SOUND N0,I*J,10,8:SOUND N1
     ,J*15,14,10:NEXT J:NEXT I
2335 REM $ IS REDEFINED TO BE A SINGL
     E CHARACTER 10
2340 POSITION XS,YS:PUT #6,ASC("$")+1
     28:PUT #6,ASC("0")+128
2350 SOUND N0,N0,N0,N0:SOUND N1,N0,N0
     ,N0
2360 SC=SC+SCORE:RETURN
2365 REM ALL 10 HANDS HAVE BEEN SCORE
     D
2366 REM TOTAL SCORE IS DISPLAYED IN
     BOTTOM RIGHT-HAND CORNER
2370 FOR I=N1 TO 150:NEXT I
2380 FOR I=10 TO 60 STEP 10:FOR J=193
      TO 243 STEP 25:SOUND 0,I,10,15:
     SOUND 1,J,4,10:SOUND 2,243,1,15
2390 NEXT J:NEXT I:FOR I=N1 TO 25:NEX
     T I
2400 S$=STR$(SC):SOUND N0,N0,N0,N0:SO
     UND N1,N0,N0,N0:GOSUB 2490:A$="
     {3 SPACES}"
2410 XPOS=16+3-LEN(S$):YPOS=10:POSITI
     ON XPOS,YPOS:? #6;S$:FOR I=N1 TO
      25:NEXT I
2420 SOUND N0,N0,N0,N0:SOUND N1,N0,N0
     ,N0:SOUND 2,N0,N0,N0:FOR I=N1 TO
      25:NEXT I
2425 REM POKE ADDRESS OF OPTION/SCORE
      SCREEN INTO WRITE-SCREEN POINTE
     R
2426 REM IF PLAYER'S SCORE IS HIGHEST
      SO FAR, UPDATE HIGH SCORE LINE
2427 REM THEN REPLACE HIGH SCORE SCRE
     EN WITH PLAYER'S SCREEN
```

```
2430 POKE 89,SC2(MAXPL+N1):IF SC<=HS
     THEN 2450
2440 HS=SC:A=USR(1536,SC2(SCS)*256,SC
     2(MAXPL+2)*256):POSITION 15,NP+1
     :? #6;S$
2445 REM RECORD PLAYER'S SCORE ON OPT
     ION/SCORE SCREEN
2450 POSITION 3,(SCS-N1):? #6;"player
      ";
2460 PUT #6,SCS+272:? #6;CHR$(ASC(":"
     )+224);:POSITION 13+(SC<100)+(SC
     <10),PEEK(84):? #6;SC
2465 REM BLINK FINAL SCORE UNTIL PLAY
     ER INDICATES TO PROCEED
2470 POKE 89,SC2(SCS):GOSUB 1000:IF O
     P=3 OR OP=5 THEN POP :SCS=SCS+N1
     :GOTO 3000
2475 REM SCORE NEXT PLAYER'S SCREEN
2480 NEXT SCS:GOTO 3000
2485 REM SUBROUTINE TO CONVERT SCORE
     FROM TAN NUMERALS TO WHITE NUMER
     ALS
2490 FOR I=N1 TO LEN(S$):S$(I,I)=CHR$
     (ASC(S$(I,I))+224):NEXT I:RETURN
2995 REM OPTION/SCORE SCREEN ROUTINE
2996 REM VARIABLE SC IS USED TO CALCU
     LATE POSITION OF HIGH SCORE LINE
3000 SC=NP+N1
3005 REM ADDRESS OF OPTION/SCORE SCRE
     EN IS POKED TO WRITE AND DISPLAY
      SCREEN POINTERS
3010 POKE 89,SC2(MAXPL+N1):POKE START
     +N1,SC2(MAXPL+N1):POKE 559,34
3020 FOR I=N1 TO 45:NEXT I:STK=N0
3025 REM START OPTION
3030 XPOS=7:YPOS=7:S$="start":A$="
     {5 SPACES}":GOSUB 1000:IF I=N0 T
     HEN 3220
3035 REM NUMBER OF PLAYERS, SAME/DIFF
     ERENT DECKS, SAME/DIFFERENT JOYS
     TICKS OPTIONS
3040 FOR I=N1 TO 25:NEXT I
```

```
3Ø5Ø XPOS=15:FOR N=NØ TO 2:YPOS=9+N
3Ø55 REM S$ CONTAINS OPTION CURRENTLY
      BEING EXERCISED
3Ø6Ø S$=STR$(NP*(N=NØ)+DKS*(N=1)+STKS
     *(N=2)):A$=" ":GOSUB 1ØØØ:IF I=N
     1 THEN 311Ø
3Ø7Ø ON N+N1 GOSUB 3Ø8Ø,3Ø9Ø,31ØØ,31Ø
     Ø:GOTO 3Ø6Ø
3Ø75 REM UPDATE APPROPRIATE VALUE(S)
3Ø8Ø NP=NP*(NP<MAXPL)+N1:DKS=(NP-N1)*
     (DKS>N1)+N1:STKS=(NP<5)*(NP-N1)*
     (STKS>N1)+N1:RETURN
3Ø9Ø DKS=(NP-N1)*(DKS<NP)+N1:RETURN
31ØØ STKS=(NP<5)*(NP-N1)*(STKS<NP)+N1
     :RETURN
311Ø FOR I=N1 TO 25:NEXT I:NEXT N
3115 REM IF ANY SCORES ARE DISPLAYED,
      ALLOW THESE TO BE REVIEWED
312Ø IF SCS=NØ THEN 316Ø
313Ø FOR N=N1 TO SCS-N1
314Ø XPOS=1Ø:YPOS=(N-N1):S$=CHR$(ASC(
     STR$(N))+224):A$=" ":GOSUB 1ØØØ:
     IF I=NØ THEN GOSUB 319Ø
315Ø FOR I=N1 TO 25:NEXT I:NEXT N
316Ø IF HS=NØ AND SCS=NØ THEN 3Ø2Ø
3165 REM IF A HIGH SCORE EXISTS, ALLO
     W THAT SCREEN TO BE REVIEWED
317Ø N=MAXPL+2:XPOS=3:YPOS=SC:S$="hig
     h score":A$="{1Ø SPACES}":GOSUB 1
     ØØØ:IF I=NØ THEN GOSUB 319Ø
318Ø GOTO 3Ø2Ø
319Ø XPOS=NØ:YPOS=NØ:S$=" ":A$=" ":PO
     KE START+N1,SC2(N):FOR I=1 TO 5Ø
     :NEXT I
32ØØ GOSUB 1ØØØ:POKE START+N1,SC2(MAX
     PL+N1):RETURN
32Ø5 REM REINITIALIZE FOR NEW GAME
321Ø POSITION XPOS,YPOS:? #6;S$
322Ø FOR K=2 TO 1Ø:SOUND NØ,K*(13Ø-K)
     +L*5,14,15:NEXT K:SOUND NØ,NØ,NØ
     ,NØ
3225 REM BRANCHING PREMATURELY OUT OF
```

```
      FOR-NEXT LOOPS CAN CAUSE ERRORS
3226 REM POP's WILL CLEAR THE STACK O
     F UNNEEDED RETURN ADDRESSES
323Ø POP :POP :POP :POP :FOR I=NØ TO
     4:FOR J=NØ TO NP*5-1:MATC(I,J)=N
     Ø:MATS(I,J)=NØ:NEXT J:NEXT I:SCS
     =NØ
324Ø FOR I=NØ TO 51:FOR J=NØ TO DKS-1
     :PLAY(I,J)=NØ:NEXT J:NEXT I:POKE
      89,SC2(N1)
325Ø FOR PL=N1 TO NP:POKE 89,SC2(PL)
326Ø POSITION N1,NØ:FOR I=N1 TO 5:FOR
      J=N1 TO 5:? #6;BL32$;BL32$;" ";
     :NEXT J
327Ø ? #6;"  "
328Ø POSITION N1,(I)*2:NEXT I
329Ø POSITION N1,1Ø:? #6;BLANK$(N1,18
     ):X=N1:Y=NØ:POSITION N1,11:? #6;
     BLANK$(N1,18)
33ØØ XSV(PL)=X:YSV(PL)=Y
331Ø A$="PLAYER":FOR I=1 TO 6:POSITIO
     N 19,I-1:PUT #6,ASC(A$(I,I))+32:
     NEXT I:POSITION 19,7:PUT #6,272+
     PL:NEXT PL
332Ø POKE 89,SC2(N1):POKE START+N1,SC
     2(N1):PL=N1:DK=N1:FLAG=N1
333Ø FOR I=8 TO N1 STEP -1:FOR J=1 TO
      8:SOUND Ø,I,1Ø,8:SOUND 1,J,14,1
     Ø:NEXT J:NEXT I:SOUND Ø,Ø,Ø,Ø:SO
     UND 1,Ø,Ø,Ø
334Ø GOTO 2Ø
3995 REM INITIAL DIMENSIONS AND SET U
     P
4ØØØ GRAPHICS 2+16:POSITION 7,1:? #6;
     "POKER":POSITION 5,3:? #6;"SOLIT
     AIRE"
4ØØ5 REM MAXPL IS THE MAXIMUM NUMBER
     OF PLAYERS DURING A RUN
4Ø1Ø MAXPL=2:NØ=Ø:N1=1:HS=Ø
4Ø15 REM ARRAYS ARE DIMENSIONED FOR T
     HE MAXIMUM NUMBER OF PLAYERS
```

```
4020 DIM XSV(MAXPL),YSV(MAXPL),PLAY(5
     1,MAXPL-1),MATC(4,MAXPL*5-N1),MA
     TS(4,MAXPL*5-N1),SC2(MAXPL+2)
4030 DIM CD$(13),SUIT$(4),A$(12),S$(1
     2),EC(13),ES(4),SCORES(8),BLANK$
     (20),BL32$(N1)
4035 REM SCORE FOR EACH POSSIBLE POKE
     R HAND, IN DESCENDING ORDER
4040 RESTORE 4050:FOR I=NØ TO 8:READ
     J:SCORES(I)=J:NEXT I
4050 DATA 100,75,50,25,20,15,10,5,2,
4055 REM A DIFFERENT COLOR BLANK
4056 REM BLANK LINE (20 BLANKS)
4060 BL32$=CHR$(ASC("@")+32):BLANK$="
     {20 SPACES}"
4065 REM FIND 4 PAGES BELOW RAMTOP FO
     R REDEFINED CHARACTER SET
4070 CHSET=(PEEK(106)-4)*256
4075 REM FIND 1 PAGE FOR EACH OF THE
     DISPLAY SCREENS NEEDED
4076 REM (1 FOR EACH PLAYER, 1 FOR OP
     TION/DISPLAY SCREEN, 1 FOR HIGH
     SCORE SCREEN)
4080 FOR I=N1 TO MAXPL+2:NSC=(PEEK(1Ø
     6)-4-I)*256
4090 SC2(I)=INT(NSC/256):NEXT I:GOSUB
      5ØØØ
4095 REM BEGINNING OF DISPLAY SCREEN
4096 REM ALL SCREENS WILL BE ON EVEN
     PAGE, SO LEAST SIGNIFICANT BYTE
     WILL BE Ø
4100 START=PEEK(56Ø)+PEEK(561)*256+4:
     POKE 88,NØ
4105 REM CLEAR ALL SCREENS
4110 FOR PL=N1 TO MAXPL+N1:POKE 89,SC
     2(PL):POSITION NØ,NØ:FOR J=NØ TO
      11:? #6;BLANK$;:NEXT J:NEXT PL
4115 REM SET UP OPTION/SCORE SCREEN
4120 POSITION N1,9:? #6;"# OF PLAYERS
     : ";:NP=N1:? #6;NP:POSITION 8,1Ø
     :? #6;"DECKS : ";:DKS=N1:? #6;DKS
```

```
4130 POSITION 7,11:? #6;"STICKS: ";:S
     TKS=N1:? #6;STKS:POSITION 7,7:?
     #6;"start"
4135 REM CARD VALUES AND SUITS
4136 REM $ IS REDEFINED TO BE SINGLE
     CHARACTER 1Ø
4137 REM [,\,],^ IN INVERSE VIDEO ARE
      REDEFINED TO BE THE GRAPHICS CH
     ARACTERS FOR HEARTS,CLUBS,DIAMON
     DS,SPADES
4140 RESTORE 415Ø:FOR I=N1 TO 13:READ
      A$:CD$(I)=A$:NEXT I:FOR I=N1 TO
      4:READ A$:SUIT$(I)=A$:NEXT I
415Ø DATA 2,3,4,5,6,7,8,9,$,J,Q,K,A,[
     ,\,],^
4155 REM DISABLE DISPLAY SCREEN TO AV
     OID FLICKER WHILE NEW DISPLAY SC
     REEN ADDRESS IS POKED IN
416Ø POKE 559,NØ:POKE START,NØ
417Ø RETURN
4995 REM REDEFINE CHARACTER SET
4996 REM IF THE END OF THE PROGRAM (P
     EEK(144)+PEEK(145)*256) EXTENDS
     INTO THE MEMORY FOR DISPLAY SCRE
     ENS,
4997 REM THERE IS NOT ENOUGH MEMORY;
     SEE TEXT
5ØØØ IF SC2(MAXPL+2)*256<PEEK(144)+PE
     EK(145)*256 THEN GRAPHICS NØ:? "
     NOT ENOUGH MEMORY; DECREASE MAXP
     L":END
5ØØ5 REM MACHINE LANGUAGE ROUTINE TO
     COPY ONE PAGE OF MEMORY FROM ONE
      LOCATION TO ANOTHER
5Ø1Ø J=CHSET:RESTORE 5Ø2Ø:FOR I=NØ TO
      2Ø:READ A:POKE 1536+I,A:K=K+A:N
     EXT I
5Ø2Ø DATA 1Ø4,1Ø4,133,2Ø6,1Ø4,1Ø4,133
     ,2Ø4,1Ø4,16Ø,Ø,177,2Ø5,145,2Ø3,2
     ØØ,192,Ø,2Ø8,247,96
5Ø25 REM SUM CHECK HELPS DECREASE THE
      LIKELIHOOD OF SYSTEM CRASH DUE
     TO INCORRECT TYPING OF LINE 5Ø2Ø
```

```
5030 IF K<>3029 THEN GRAPHICS N0:? :?
      "CHECK LINE 5020 FOR ERROR":? :
     LIST 5020:END
5035 REM COPY FIRST HALF OF ROM CHARA
     CTER SET TO RAM
5040 A=USR(1536,57344,CHSET):A=USR(15
     36,57344+256,CHSET+256)
5045 REM REDEFINE SUIT CHARACTERS
5046 REM REPLACE [,◢,],^ WITH THE GRA
     PHICS CHARACTERS FOR THE SUITS
5050 S=J:J=J+472:A$(1,1)=CHR$(N0):A$(
     2,2)=CHR$(16):A$(3,3)=CHR$(96):A
     $(4,4)=CHR$(123)
5060 FOR I=N1 TO 4:FOR K=N0 TO 7:POKE
      J+((I-N1)*8+K),PEEK(57344+(ASC(
     A$(I,I))+64*(I<3))*8+K):NEXT K:N
     EXT I
5065 REM SINGLE CHARACTER 10
5070 RESTORE 5080:A=S+8*4:FOR I=N0 TO
      7:READ N:POKE A+I,N:NEXT I
5080 DATA 0,102,237,109,107,107,246,0
5085 REM REDEFINE @ TO BE INVERSE VID
     EO BLANK
5090 S=S+8*32:FOR I=0 TO 7:POKE S+I,2
     55:NEXT I
5095 REM ADDRESS OF NEW CHARACTER SET
5100 POKE 756,CHSET/256
5110 RETURN
```

Part Four

Brain Testers

Part Four

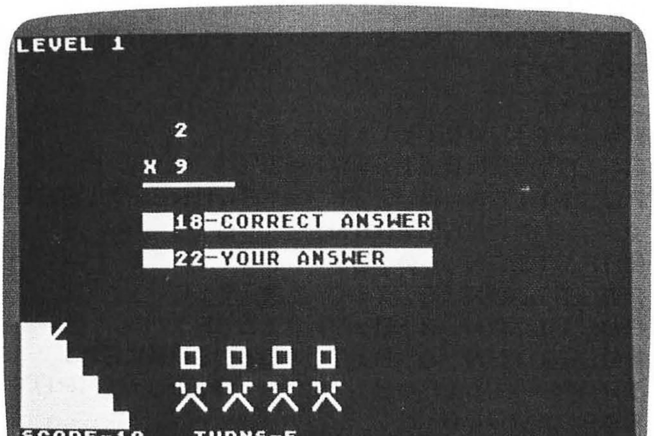
MathMan

Andy Hayes
Translated for the Atari by Charles Brannon

Entertaining animation makes this math practice game fun to play.

Here's a program which proves that computer-aided math practice need not be boring. In the guise of a game, "MathMan" teaches multiplication facts by presenting random problems. The player (or student) types in the answer and presses RETURN. If he is correct, his friends gathered below cheer, but if the player fails to guess correctly, one of his friends will run away in shame. If all six friends flee, the game is over.

A good player can advance to the next level by successfully completing ten problems. The problems get successively more difficult, so this single program will provide challenge for almost any elementary school child. (Adults may also enjoy the animation.)



```
LEVEL 1

        2
      X 9
      ___
      18-CORRECT ANSWER
      22-YOUR ANSWER

SCORE-10    TURNS-5
```

Multiplication practice can be fun with "Mathman."

Part Four

Program 4-1. MathMan

```
0 A=6
1 LV=1
10 GRAPHICS 0:POKE 82,0:POKE 752,1:?
   "{CLEAR}"
11 Y=INT(RND(0)*15):SETCOLOR 4,Y,6:SE
   TCOLOR 2,Y,4
20 POSITION 0,16
200 ? "█{F}"
220 ? "{3 SPACES}"
230 ? "{4 SPACES}"
240 ? "{5 SPACES}"
250 ? "{6 SPACES}"
260 ? "{7 SPACES}"
270 IF A=0 THEN 3000
275 FOR I=1 TO A
280 POSITION I*3+6,17
290 ? "{RIGHT}{Q}{E}{DOWN}{2 LEFT}
    {Z}{C}{DOWN}{2 LEFT}{E}{Q}{DOWN}
    {2 LEFT}{F}{G}"
295 NEXT I
298 SC=PEEK(88)+256*PEEK(89)
299 POSITION 0,0:? "LEVEL ";LV
322 IF O=10 THEN LV=LV+1:GOTO 2000
350 S=LV*2
355 O=O+1
360 B=INT(RND(1)*S)+1
370 C=INT(RND(1)*9)+1
375 POSITION 0,22:? "SCORE-";SCO;"
    {3 SPACES}TURNS-";O
380 POSITION 10,5:? B;"   "
390 POSITION 8,7:? "X  ";C
400 POSITION 8,8:? "{6 R}"
410 POSITION 8,9:? "{6 SPACES}"
415 TRAP 415:POSITION 8,10:INPUT AS:T
    RAP 40000
430 IF AS=B*C THEN 700
440 IF AS<>B*C THEN 1000
700 SCO=SCO+5*LV
711 POSITION 10,14:? "THANK YOU!!!"
715 X=X+1
```

```
730 E=INT(RND(1)*30)+210
742 FOR T=1 TO 10:POKE 710,PEEK(53770
    ):SOUND 0,T,10,8:NEXT T
743 SETCOLOR 2,9,4:SOUND 0,0,0,0
744 IF X=10 THEN X=0:GOTO 760
750 GOTO 715
760 FOR T=1 TO 500
770 COLOR 32:PLOT 0,23:DRAWTO 39,23
772 PLOT 0,10:DRAWTO 39,10
775 PLOT 0,16:DRAWTO 39,16
776 IF O=10 THEN 790
780 F=0:GOTO 10
790 LV=LV+1:GOTO 2000
1000 Q=SC+604
1005 POKE Q,0:Q=Q-39:POKE Q,10
1010 IF Q-SC<=409 THEN 1030
1020 GOTO 1005
1030 FOR I=1 TO 10:SOUND 0,I,0,10-I:S
     OUND 1,I*10+50,2,8
1040 POKE Q,128:POKE Q+1,128:POKE Q-1
     ,128:POKE Q+40,128:POKE Q-40,128
1050 POKE Q+40,0:POKE Q-40,0:POKE Q,0
     :POKE Q-1,0:POKE Q+1,0
1060 NEXT I:SOUND 1,0,0,0
1070 POSITION 8,10:? "{4 SPACES}":POS
     ITION 10,10:? B*C;"-CORRECT ANSW
     ER"
1071 POSITION 8,12:? "{4 SPACES}":POS
     ITION 10,12:? AS;"-YOUR ANSWER
     {3 SPACES}"
1072 FOR Z=1 TO 200:NEXT Z
1080 REM MAN RUNS AWAY
1090 REM
1100 FOR I=A*3+6 TO 35
1110 POSITION I,17:? " {Q}{E}{DOWN}
     {3 LEFT} {A}{C}{DOWN}{3 LEFT} |
     {F}{DOWN}{3 LEFT} {F}{G}"
1115 SOUND 0,100,0,8
1120 FOR W=1 TO 5:NEXT W
1125 SOUND 0,10,0,8
1130 POSITION I,17:? " {Q}{E}{DOWN}
     {3 LEFT} {A}{C}{DOWN}{3 LEFT} |
     {R}{DOWN}{3 LEFT} |{G}"
```

```
1140 FOR W=1 TO 5:NEXT W
1145 SOUND 0,0,0,0
1150 NEXT I
1160 A=A-1:GOTO 10
1199 END
1413 NEXT K
2000 PRINT "{CLEAR}{6 DOWN} YOU MADE
     IT THROUGH"
2005 PRINT "{6 SPACES}LEVEL ";LV-1
2010 PRINT "{2 DOWN}  YOU NOW ADVANCE
      TO"
2015 PRINT "{6 SPACES}LEVEL ";LV:O=0:
     FOR T=1 TO 500:NEXT T:GOTO 10
3000 REM
3010 PRINT "{CLEAR}{4 DOWN}SORRY BUT
     YOU LOST ALL";
3020 PRINT "{2 DOWN}{7 SPACES}YOUR ME
     N"
3030 PRINT "{4 DOWN}{4 SPACES}YOUR SC
     ORE WAS"
3040 PRINT "{2 DOWN}{7 SPACES}";SCO
```

Part Four

Word Hunt

Robert W. Baker
Translated for the Atari by Charles Brannon

Word search puzzles are interesting and enjoyable. This computerized word search game creates a puzzle using words entered by the players.

This game is designed to test your ability to find specific words or letter sequences hidden in a ten-by-ten letter matrix. Scoring is based on the time it takes to enter your correct answer within a given time period determined by the skill level selected. The program uses very little memory and will easily run in 8K.

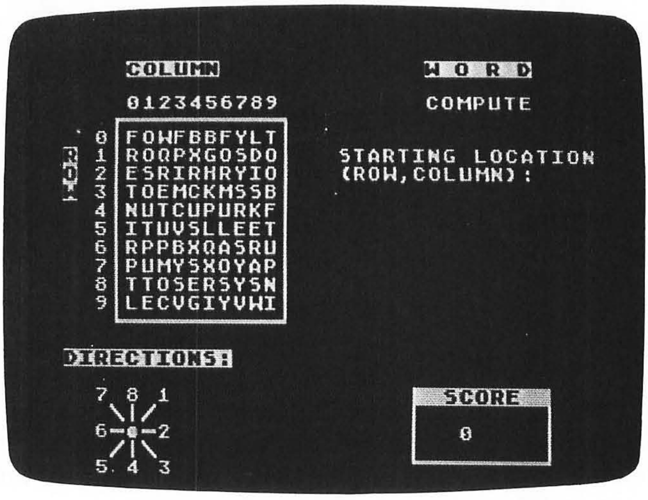
To play the game, first select the skill level you want to play at, between one and five. One is the easiest, allowing the maximum time of approximately 1.5 minutes to find each word. Skill level five, however, will allow only about 20 seconds to find each word.

Entering the Words

Next enter a list of ten words, each being three to eight characters long. Only the letters A to Z may be entered, but they really do not have to be words. You can even use the game to make learning foreign languages more fun. If two people are playing, let one player enter the words for the other to find. Try to mix the word lengths entering both long and short words for best results. If too many long words are entered, it may take awhile for the puzzle to be generated. If any word will not fit into the matrix, enter a new list of words when asked.

When the puzzle is ready, hit any key to start the game. Timing starts when the first word is shown.

Scoring for a correct answer is based on the amount of time it takes to respond, with 100 points maximum for each of ten words. If a correct answer is given in five seconds, you score 100 points. After that, your possible score decreases with time to a minimum of ten points for a correct answer. A wrong answer does not score any points, and you get only *one* try for each word.



```
     COLUMN                     W O R D
     0123456789                 COMPUTE
   0 FOWFBBFYLT
 R 1 ROQPXGOSDO           STARTING LOCATION
 O 2 ESRIRHRYIO           (ROW,COLUMN):
 W 3 TOEMCKMSSB
   4 NUTCUPURKF
   5 ITUVSLLEET
   6 RPPBXQASRU
   7 PUMYSXOYAP
   8 TTOSERSYSN
   9 LECVGIYVWI

DIRECTIONS:
   7 8 1
    \|/
   6-●-2
    /|\
   5 4 3                        SCORE
                                  0
```

How many words can you find in our puzzle?

To enter your answer, you give a row and column number of the first character of the word, followed by the direction code (see the diagram in the game). Any invalid entries are discarded, and you type only numbers; you do not type a comma or RETURN.

Program Variables

Before looking at how the program actually works, let's take a look at the major variables used in the program:

S — defines the size of the letter matrix to be created.
W — defines the number of words to be entered and used in the matrix.
M(S,S) — is the actual letter matrix; note that a floating point numeric matrix is used instead of a string matrix. More about this later.
W$(W) — contains the word list.
L(W,3) — remembers the starting location and direction of each word after it has been placed in the letter matrix. Each entry directly corresponds to the entry in the same position in the word matrix.
P(S,S) and F(8) — are working matrices used to create the actual letter matrix used in the game.

Part Four

Program Description

Now let's take a look at how the program works. First the program gets the desired skill level (SL) as a number between one and five. The program sets a default value of three on the input line that the user can change before hitting the RETURN key. Lines 130-290 then get the list of words and check if each is a valid character string (A-Z). Each word is put into the word list in alphabetical order as it is entered by the user. This avoids the time-consuming process of sorting the entire word list at the end. In this way, there is a short delay as each word is entered. This short delay is not even noticeable by the user.

Line 340 initializes the letter matrix to all *'s (decimal value 42). Now each word in the word list is inserted randomly in the letter matrix in the following fashion:

1) The point matrix is cleared (line 360) so we can remember what points in the matrix have been tried for a particular word in the word list.
2) Lines 400-440 check that there is still at least one point in the letter matrix that has not been tried (entry in P is still 0). If all points have been tried, the user is asked to enter a new list of words since this list will not fit properly in the letter matrix.
3) A random starting point (that has not been tried) is chosen in line 450.
4) The starting point is flagged as having been tried (P value now 1), and then a check is made to see if the matrix position is open (still *) or matches the first letter of the word (lines 460-470).
5) Now the direction matrix (F) is cleared to remember what directions have been tried from this starting point (line 490).
6) A check is made that a least one direction still hasn't been tried from this point (lines 500-510).
7) A random direction (that has not been tried) is chosen in line 520.
8) Then the word is checked to see if it can physically fit in the matrix in the selected direction from the current starting point (lines 530-650). This insures that the word will not exceed the boundaries of the letter matrix from this point.

9) If the word can fit, then each character position in the selected direction is checked against the corresponding character of the word (lines 670-690). Each character in the matrix must match the corresponding character in the word or must be unused (still *).
10) If the word can be entered at this starting point and in this direction, each letter is inserted in the letter matrix (lines 710-720). Then the starting location and direction are saved for later use (line 740).
11) If the word will not fit, then the next direction is tried until all directions are exhausted from this point.

When all words have been put into the matrix, the remaining unused positions (still *) are filled in with random letters (lines 760-770).

Play Begins

Everything is now set to play the game, as soon as the player hits a key (lines 780-800). The letter matrix is displayed along with a direction code diagram and a score box (lines 820-960). A word is given to the player for him or her to find in the matrix, and the timer is restarted (lines 970-1000). Then the program prompts the player for the starting location and direction code (lines 1020-1170). The values entered are then checked to see if correct, first against the values saved when the word was put into the matrix (lines 1190-1210). If the value does not match, then the program checks to see if a "double" was created when the unused positions were filled with random letters. Thus the program checks the player's answer again to insure it is right or wrong (lines 1230-1280). If a bad answer is entered, it is indicated, and the correct answer is displayed with no score added (lines 1360-1430). A good answer is indicated and the appropriate score is displayed and added to the player's total. The score is based on the selected skill level and the time it takes to enter the answer.

That's all there is to it. I should explain that a numeric vector is used for the actual letter matrix since it is easier and faster to use. Most people who have tried this game have found it to be very interesting and fun to play. At times it can even be educational.

Program 4-2. Word Hunt

```
70 OPEN #1,4,0,"K:"
80 S=10:W=10:DIM M(S,S),W$(W*10),LN(W
   ),P(S,S),L(W,3),F(8),R$(10),T$(10)
85 T$="{10 SPACES}":FOR I=0 TO 9:W$(I*
   10+1,I*10+10)=T$:NEXT I
90 POKE 752,0:PRINT "{CLEAR}{DOWN}WHA
   T SKILL LEVEL"
100 ? :? "1 (EASY) - TO - 5 - (HARD)?
    3{2 LEFT}";
110 INPUT X:IF X<1 OR X>5 THEN 100
120 SL=6-X
130 ? "{2 DOWN}ENTER ";W;" WORDS,"
140 ? "Each 3 to 7 characters long
    {2 DOWN}"
150 REM *** GET WORDS & PUT IN ORDER
160 REM *** LONGEST TO SHORTEST
170 FOR X=1 TO W:L(X,1)=0:L(X,2)=0:L(
    X,3)=0
180 PRINT "WORD ";X;:INPUT R$
190 Q=LEN(R$)
200 IF Q<3 THEN ? "* TOO SHORT *":GOT
    O 180
210 IF Q>7 THEN ? "* TOO LONG  *":GOT
    O 180
220 X9=0:T$="*":T$(2)=R$:T$(LEN(T$)+1
    )="*":FOR Y=1 TO Q:A=ASC(T$(Y+1,Y
    +1))
230 IF A<65 OR A>90 THEN X9=1:Y=Q
240 NEXT Y:IF X9=1 THEN PRINT "* BAD
    WORD *":GOTO 180
250 IF X=1 THEN T$=R$:T$(Q+1)="*":W$(
    X*10-9,X*10)=T$:LN(X)=Q+1:GOTO 290
260 X9=0:FOR Y=1 TO X-1:IF Q<=LN(Y)-1
     THEN 280
270 FOR B=X TO Y+1 STEP -1:T$=W$((B-1
    )*10-9,(B-1)*10):W$(B*10-9,B*10)=
    T$:LN(B)=LN(B-1):NEXT B
```

```
275 T$=R$:T$(Q+1)="*":W$(Y*10-9,Y*10)
    =T$:LN(Y)=LEN(T$):Y=X-1
280 NEXT Y:IF X9=0 THEN T$=R$:T$(Q+1)
    ="*":W$(X*10-9,X*10)=T$:LN(X)=LEN
    (T$)
290 NEXT X
300 POKE 752,1:? "{CLEAR}{7 DOWN}That
    's enough words!"
310 PRINT "{6 DOWN}Please be patient.
    ...."
320 ? "{3 DOWN}{12 SPACES}I'm now maki
    ng the puzzle!"
330 REM *** INITIALIZE LETTER MATRIX
    ***
340 FOR X=1 TO S:FOR Y=1 TO S:M(Y,X)=
    42:NEXT Y:NEXT X:Q=0
350 REM *** INIT POINT MATRIX & GET N
    EXT WORD
360 FOR X=1 TO S:FOR Y=1 TO S:P(Y,X)=
    0:NEXT Y
370 NEXT X:Q=Q+1:IF Q>W THEN 760
380 G=LN(Q)-2
390 REM *** TRY ALL POINTS FOR EACH W
    ORD
400 X9=0:FOR X=1 TO S:FOR Y=1 TO S:IF
     P(Y,X)=0 THEN X9=1:X=S:Y=S
410 NEXT Y:NEXT X:IF X9=1 THEN 450
420 REM *** WORD WILL NOT FIT, TRY AG
    AIN
430 ? "{CLEAR}This list of words will
     not all fit."
440 ? :? "Please enter another list o
    f words!":GOTO 130
450 A=INT(S*RND(1)+1):B=INT(S*RND(1)+
    1):IF P(B,A)<>0 THEN 450
460 P(B,A)=1:IF M(B,A)=42 THEN 490
470 IF M(B,A)<>ASC(W$(Q*10-9)) THEN 4
    00
480 REM *** TRY ALL DIRECTIONS FROM T
```

```
    HIS POINT
490 FOR X=1 TO 8:F(X)=Ø:NEXT X
500 X9=Ø:FOR X=1 TO 8:IF F(X)=Ø THEN
    X9=1:X=8
510 NEXT X:IF X9=Ø THEN 4ØØ
520 D=INT(8*RND(1)+1):IF F(D)=1 THEN
    52Ø
530 F(D)=1:ON D GOTO 55Ø,59Ø,58Ø,62Ø,
    61Ø,65Ø,64Ø,56Ø
540 REM *** CHECK WORD WILL FIT
550 IF (A+G)>S THEN 5ØØ
560 IF (B-G)<1 THEN 5ØØ
570 GOTO 67Ø
580 IF (B+G)>S THEN 5ØØ
590 IF (A+G)>S THEN 5ØØ
600 GOTO 67Ø
610 IF (A-G)<1 THEN 5ØØ
620 IF (B+G)>S THEN 5ØØ
630 GOTO 67Ø
640 IF (B-G)<1 THEN 5ØØ
650 IF (A-G)<1 THEN 5ØØ
660 REM *** CHECK WORD MATCHES INTO M
    ATRIX
670 X=A:Y=B:X9=Ø:FOR N=2 TO G+1:GOSUB
     155Ø:IF M(Y,X)=42 THEN 69Ø
680 IF M(Y,X)<>ASC(W$((Q-1)*1Ø+N)) TH
    EN X9=1:N=G+1
690 NEXT N:X=A:Y=B:IF X9=1 THEN 5ØØ
700 REM *** ENTER WORD
710 FOR N=1 TO G+1:IF M(Y,X)=42 THEN
    M(Y,X)=ASC(W$((Q-1)*1Ø+N))
720 GOSUB 155Ø:NEXT N
730 REM *** SAVE START & DIRECTION IN
    FO
740 L(Q,1)=A-1:L(Q,2)=B-1:L(Q,3)=D:IF
     D<W THEN 36Ø
750 REM *** FILL IN SPACES
760 FOR Y=1 TO S:FOR X=1 TO S:IF M(Y,
    X)=42 THEN M(Y,X)=INT(25*RND(1)+6
    5)
770 NEXT X:NEXT Y:WF=Ø:TS=Ø
780 ? "{CLEAR}{10 DOWN}{15 SPACES}READY"
```

```
790 ? "{6 DOWN}Depress any key when r
    eady to play!"
800 IF PEEK(764)=255 THEN 800
805 POKE 764,255
810 REM *** SET UP DISPLAY
820 ? "{CLEAR}{DOWN}{4 SPACES}COLUMN"
    ;:POKE 85,26:? "W O R D"
830 REM *** PRINT 'ROW' DOWN LEFT COL
    UMN
840 REM *** START OUT DOWN 4
850 REM *** LATER DO 5 UP & 3 RIGHT
860 ? "{4 DOWN}R{DOWN}{LEFT}O{DOWN}
    {LEFT}W{5 UP}{3 RIGHT}";
870 FOR X=0 TO S-1:? X;:NEXT X:? :Y=1
    :GOSUB 1650
880 FOR Y=1 TO S:? "{2 RIGHT}";Y-1;"|
    ";
890 FOR X=1 TO S:? CHR$(M(Y,X));:NEXT
     X
900 ? "|":NEXT Y:Y=0:GOSUB 1650
910 ? :? "DIRECTIONS:":? "{DOWN}  7 8
     1"
920 ? "{3 SPACES}{G}|{F}":? "  6{R}
    {T}{R}2":? "{3 SPACES}{F}|{G}":?
    "  5 4 3"
930 G=16:GOSUB 1700:? "{5 SPACES}  SC
    ORE  ":POKE 85,25:? "{V}
    {7 SPACES}{B}"
940 POKE 85,25:? "{V}  0{4 SPACES}
    {B}"
950 POKE 85,25:? "{V}{7 SPACES}{B}"
955 POKE 85,25:? "{9 M}"
960 G=0:GOSUB 1700:? "{19 SPACES}":REM
     <-- 19 SPACES
970 WP=WP+1:IF WP>W THEN 1450
980 Q=LN(WP)-1
990 REM *** NEXT WORD
1000 GOSUB 1700:POKE 85,29-(Q/2):? W$
     ((WP-1)*10+1,(WP-1)*10+Q)
1005 POKE 20,0:POKE 19,0:REM KILL RTC
     LK
1010 REM *** GET START LOC
```

```
1020 G=3:GOSUB 1700:? "STARTING LOCAT
     ION":POKE 85,20:? "(ROW,COLUMN):
     "
1030 FOR G=6 TO 14:GOSUB 1700
1040 ? "{19 SPACES}":NEXT G:G=6:GOSUB
     1700:REM <-- 19 SPACES
1050 GET #1,B:IF B=155 THEN 1050
1070 PRINT CHR$(B);",";:IF B=48 THEN
     B=0:GOTO 1090
1080 B=B-48:IF B<1 OR B>9 THEN PRINT
     "{2 BACK S}";:GOTO 1050
1090 GET #1,A
1100 IF A=155 THEN 1090
1110 PRINT CHR$(A);:IF A=48 THEN A=0:
     GOTO 1140
1120 A=A-48:IF A<1 OR A>9 THEN 1030
1140 G=8:GOSUB 1700:PRINT "DIRECTION:
     ";? :POKE 85,20:? " {LEFT}";
1150 GET #1,D
1160 IF D=155 THEN 1150
1170 PRINT CHR$(D);:D=D-48:IF D<1 OR
     D>8 THEN 1140
1180 REM *** CHK IF GOOD INFO INPUT
1190 WT=PEEK(20)+256*PEEK(19):IF B<>L
     (WP,2) THEN 1230
1210 IF D=L(WP,3) THEN 1360
1220 REM *** CHK IF A DOUBLE MAY EXIS
     T
1230 X=A+1:Y=B+1:G=LN(WP)-1:IF M(Y,X)
     <>ASC(W$(WP*10-9)) THEN 1300
1240 X9=0:FOR N=2 TO G:GOSUB 1550:IF
     X<1 OR X>10 THEN 1270
1250 IF Y<1 OR Y>10 THEN 1270
1260 IF M(Y,X)=ASC(W$((WP-1)*10+N)) T
     HEN 1280
1270 X9=1:N=G
1280 NEXT N:IF X9=0 THEN 1360
1290 REM *** BAD START/DIR - NO SCORE
1300 G=6:GOSUB 1700:PRINT "
     {14 SPACES}";:B=L(WP,2):A=L(WP,1)
     :REM 14 SPACES
1310 ? B;",";A
```

```
132Ø G=1Ø:GOSUB 17ØØ:? "{13 SPACES}";L
     (WP,3):REM 13 SPACES
133Ø G=12:GOSUB 17ØØ:? "{ESC}{UP}
     {13 SPACES}{ESC}{UP}":REM 13 SPAC
     ES
134Ø G=13:GOSUB 17ØØ:? "{Z} NO, CORRE
     CT {C}":GOTO 142Ø
135Ø REM *** GOOD ANSWER - GET SCORE
136Ø IF WT<(SL*6Ø) THEN WS=1ØØ:GOTO 1
     39Ø:REM <-- MAX SCORE
137Ø IF WT>(SL*12ØØ) THEN WS=1Ø:GOTO
     139Ø:REM <-- MIN SCORE
138Ø WS=5+INT(((SL*12ØØ)-WT)/6Ø)
139Ø G=12:GOSUB 17ØØ:? "{ESC}{UP}"
14ØØ G=13:GOSUB 17ØØ:? "{Z} YES, ";WS
     ;" POINTS":TS=TS+WS
141Ø REM *** UPDATE TOTAL SCORE
142Ø G=18:GOSUB 17ØØ:? "{8 RIGHT}";TS
143Ø FOR X=1 TO 5ØØ:NEXT X:GOTO 96Ø
144Ø REM *** END GAME ***
145Ø POSITION 2,15
146Ø FOR X=1 TO 8:? "{12 SPACES}":NEXT
      X:REM <-- 12 SPACES
147Ø FOR G=-2 TO 14:GOSUB 17ØØ
148Ø PRINT "{17 SPACES}":NEXT G:REM 17
      SPACES
149Ø POSITION 2,15:? "PLAY AGAIN (Y O
     R N) ?"
15ØØ GET #1,R
151Ø IF R=ASC("Y") THEN 9Ø
152Ø IF R<>ASC("N") THEN 15ØØ
153Ø END
154Ø REM *** SUBR TO INC COORDINATES
     IN DIR
155Ø ON D GOTO 156Ø,157Ø,158Ø,159Ø,16
     ØØ,161Ø,162Ø,163Ø
156Ø Y=Y-1
157Ø X=X+1:RETURN
158Ø X=X+1
159Ø Y=Y+1:RETURN
16ØØ Y=Y+1
161Ø X=X-1:RETURN
```

```
1620 X=X-1
1630 Y=Y-1:RETURN
1640 REM *** SUBR FOR BOT TOP/BOTTOM
1650 PRINT "{3 RIGHT}";:IF Y=1 THEN ?
      "{Q}";:GOTO 1670
1660 PRINT "{Z}";
1670 FOR X=0 TO S-1:? "{R}";:NEXT X:I
     F Y=1 THEN PRINT "{E}":RETURN
1680 ? "{C}":RETURN
1690 REM *** SUBR TO POSITION
1700 POSITION 20,G+2:RETURN
```

Part Four

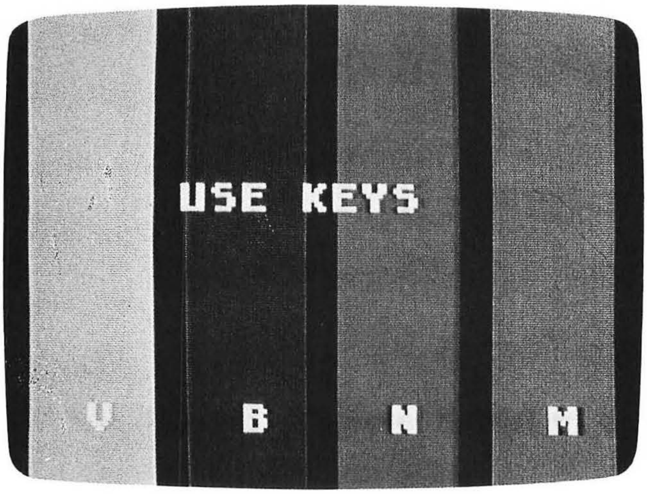
Total Recall

Tina Halcomb

"Total Recall" is a simple memory game. It's easy at first, but as the computer adds more and more to remember, the game becomes more difficult.

After playing a game or two of "Simon," it occurred to me that this would be a terrific game to write for the Atari. After all, the Atari has color graphics, sound — everything I could possibly need. However, I couldn't come up with an easy way to display more than four colors on the screen at one time without painting my television set.

Then along came the articles on Player/Missile Graphics. At last I have access to a multitude of colors. What a perfect solution to my problem:



"Total Recall."

Using Player/Missile Graphics I drew four vertical bars, giving each a separate color and sound. As the selection routine makes each decision, the luminescence of the bar chosen is changed and its tone sounded. Several counting routines are called upon to keep track of the events; and there are, of course, timing loops.

There are also times when the screen is needed for text. Instructions to the game might be needed. And since I have not discovered a way to make the Atari laugh, we need to be able to convey a message to the poor person who loses the game.

To clear the screen for the text display, you can position the bars to the far right where they are not visible. They can be returned to their playing positions at any time.

The keys **V B N M** are used to identify the bars. The keyboard is monitored to determine if your responses are correct. The Atari keeps score.

How good is your memory? Play a few games of "Total Recall" and you will soon find out.

Program 4-3. Total Recall

```
10 DIM A$(32)
15 REM ASSIGN TONE TO BAR
20 DIM TONE(4):TONE(1)=60:TONE(2)=72:
   TONE(3)=96:TONE(4)=144
1000 GRAPHICS 2+16:SETCOLOR 2,7,0:SET
     COLOR 4,7,0:SETCOLOR 0,0,14
1010 POSITION 3,5:PRINT #6;"TOTAL REC
     ALL"
1020 FOR W=1 TO 1000:NEXT W
1200 REM DRAW  BARS
1210 A=PEEK(106)-8:POKE 54279,A:PMBAS
     E=256*A:FOR X=512 TO 1023:POKE P
     MBASE+X,255:NEXT X
1215 REM ASSIGN COLOR TO BAR
1220 POKE 559,46:POKE 704,26:POKE 705
     ,116:POKE 706,68:POKE 707,196
1225 REM SET PLAYERS TO QUAD WIDTH AN
     D TURN ON P/M GRAPHICS
1230 POKE 53256,3:POKE 53257,3:POKE 5
     3258,3:POKE 53259,3:SETCOLOR 2,0
     ,0:SETCOLOR 4,0,0
```

```
1240 GOSUB 3000
1250 POKE 53277,3
1290 REM PLAY INTRODUCTION
1300 DELAY=100:RESTORE 1375
1310 READ SELECT
1320 IF SELECT=256 THEN 1500
1340 GOSUB 1400
1350 GOTO 1310
1375 DATA 2,1,0,1,2,1,0,1,3,2,1,0,1,2
     56
1400 REM FLASH BARS AND SOUND TONES
1410 A=704:P=PEEK(A+SELECT):C=INT(P/1
     6):L=P-C*16:POKE A+SELECT,C*16+1
     4
1420 SOUND 0,TONE(SELECT+1),10,10
1430 FOR Y=1 TO DELAY:NEXT Y:SOUND 0,
     0,0,0
1440 POKE A+SELECT,P
1450 RETURN
1500 REM SHORT INSTRUCTIONS
1510 GOSUB 4000
1520 GRAPHICS 2+16:SETCOLOR 2,7,0:SET
     COLOR 4,7,0:SETCOLOR 0,0,14:POKE
     559,46
1530 POSITION 3,3:PRINT #6;"PRESS RET
     URN"
1540 POSITION 4,6:PRINT #6;"TO START"
1550 POKE 764,255
1560 IF PEEK(764)=255 THEN 1560
2000 REM SHORT INSTRUCTIONS
2005 GRAPHICS 2+16:POKE 559,46:SETCOL
     OR 0,0,14:SETCOLOR 2,0,0:SETCOLO
     R 4,0,0
2008 POSITION 5,4:PRINT #6;"USE KEYS"
2010 POSITION 0,10:PRINT #6;"  V
     {4 SPACES}B{4 SPACES}N
     {4 SPACES}M "
2011 FOR W=1 TO 500:NEXT W
2014 REM PLAY GAME
2015 GOSUB 3000
2018 REM SET UP ENTIRE SEQUENCE OF 32
     IN A$
```

```
2020 FOR X=1 TO 32:A$(X,X)=STR$(INT(R
     ND(0)*4)):NEXT X
2030 CURRENT=1
2040 DELAY=50
2045 REM MAKE GAME GO FASTER AFTER I
     MOVES
2050 IF CURRENT>8 THEN DELAY=DELAY-1
2060 FOR X=1 TO CURRENT
2070 SELECT=VAL(A$(X,X)):GOSUB 1400:I
     F X<>CURRENT THEN FOR Y=1 TO DEL
     AY:NEXT Y:NEXT X
2075 REM CLEAR KEYBOARD AND WAIT FOR
     KEY TO BE PRESSED (DON'T WAIT TO
     O LONG)
2080 X=1:POKE 764,255
2090 WAIT=5*DELAY:SELECT=4
2100 WAIT=WAIT-1:IF WAIT=0 THEN 2300
2110 CH=PEEK(764):IF CH=255 THEN 2100
2120 POKE 764,255
2125 REM DECIDE WHICH KEY IS PRESSED
     IF NOT V B N M THEN IGNORE IT
2130 IF CH=16 THEN SELECT=0
2140 IF CH=21 THEN SELECT=1
2150 IF CH=35 THEN SELECT=2
2160 IF CH=37 THEN SELECT=3
2170 IF SELECT=4 THEN 2100
2180 IF SELECT<>VAL(A$(X,X)) THEN 221
     0
2190 GOSUB 1400:X=X+1:IF X<>CURRENT+1
      THEN 2090
2200 CURRENT=CURRENT+1:IF CURRENT<>32
      THEN FOR Y=1 TO 5*DELAY:NEXT Y:
     GOTO 2050
2210 REM
2300 REM SCORE ROUTINE
2310 GOSUB 4000
2320 GRAPHICS 2+16:SETCOLOR 2,7,0:SET
     COLOR 4,7,0:SETCOLOR 0,0,14
2330 IF CURRENT=32 THEN 2950
2340 POSITION 4,5:PRINT #6;"YOU LOSE"
2350 RESTORE 2400
2360 READ A
```

```
2370 IF A=256 THEN SOUND Ø,Ø,Ø,Ø:GOTO
      2600
2380 SOUND Ø,A,1Ø,1Ø
2390 FOR W=1 TO 5Ø:NEXT W
2395 GOTO 236Ø
2400 DATA 91,91,1Ø8,81,91,91,1Ø8,256
2600 GRAPHICS 2+16:POKE 559,46:SETCOL
     OR 2,7,Ø:SETCOLOR 4,7,Ø:SETCOLOR
      Ø,Ø,14
2610 POSITION 5,2:PRINT #6;"YOU GOT"
2620 POSITION 8,4:PRINT #6;CURRENT-1
2630 POSITION 5,6:PRINT #6;"CORRECT"
2640 FOR W=1 TO 1ØØØ:NEXT W
2650 REM
2700 IF CURRENT>6 THEN 275Ø
2710 GRAPHICS 2:POKE 559,46:SETCOLOR
     2,2,8:SETCOLOR 4,2,8:SETCOLOR Ø,
     3,Ø
2720 POSITION 4,2:PRINT #6;"NOW TRY I
     T "
2730 POSITION 4,4:PRINT #6;"WITH  YOU
     R"
2740 POSITION 4,6:PRINT #6;" T.V. ON!
     "
2745 FOR W=1 TO 2ØØ:NEXT W
2746 GOTO 15ØØ
2750 IF CURRENT>12 THEN 28ØØ
2760 GRAPHICS 2:POKE 559,46:SETCOLOR
     2,7,8:SETCOLOR 4,7,8:SETCOLOR Ø,
     5,Ø
2770 POSITION 3,2:PRINT #6;"THIS TIME
      USE"
2780 POSITION 6,4:PRINT #6;"*KEYS*"
2785 POSITION 5,6:PRINT #6;"V B N M"
2790 FOR W=1 TO 4ØØ:NEXT W
2795 GOTO 15ØØ
2800 IF CURRENT>24 THEN 285Ø
2810 GRAPHICS 2:POKE 559,46:SETCOLOR
     2,6,Ø:SETCOLOR 4,6,Ø:SETCOLOR Ø,
     Ø,14
2820 POSITION 2,2:PRINT #6;"YOU THOUG
     HT YOU"
```

```
2825 POSITION 4,4:PRINT #6;"WERE DOIN
     G"
2830 POSITION 2,6:PRINT #6;"PRETTY WE
     LL...."
2835 POSITION 3,8:PRINT #6;"DIDN'T YO
     U!!"
2840 FOR W=1 TO 400:NEXT W
2845 GOTO 1500
2850 IF CURRENT>28 THEN 2900
2860 GRAPHICS 2:POKE 559,46:SETCOLOR
     2,5,8:SETCOLOR 4,5,8:SETCOLOR 0,
     7,0
2865 POSITION 2,3:PRINT #6;"HOW MANY
     PEOPLE"
2870 POSITION 4,5:PRINT #6;"ARE PLAYI
     NG"
2880 POSITION 4,7:PRINT #6;"THIS GAME
     ??"
2890 FOR W=1 TO 400:NEXT W
2895 GOTO 1500
2900 GRAPHICS 2:POKE 559,46:SETCOLOR
     2,2,8:SETCOLOR 4,2,8:SETCOLOR 0,
     0,14
2910 POSITION 1,2:PRINT #6;"IF YOU'RE
      SO SMART"
2920 POSITION 3,4:PRINT #6;"WHY DIDN'
     T YOU"
2930 POSITION 5,6:PRINT #6;"WRITE THI
     S"
2935 POSITION 8,8:PRINT #6;"GAME "
2940 FOR W=1 TO 400:NEXT W
2945 GOTO 1500
2950 FOR T=1 TO 5
2955 GRAPHICS 2:SETCOLOR 2,7,0:SETCOL
     OR 4,7,0:SETCOLOR 0,5,8
2960 FOR W=1 TO 50:NEXT W
2965 POSITION 5,4:PRINT #6;"TILT!!!"
2970 FOR W=1 TO 200:NEXT W
2975 NEXT T
2990 GOTO 1500
3000 REM PUT BARS INTO PLAYING POSITI
     ON
```

```
3Ø1Ø POKE 53248,5Ø:POKE 53249,9Ø:POKE
      53250,13Ø:POKE 53251,17Ø:POKE 6
     23,4
3Ø2Ø RETURN
4ØØØ REM MOVE BARS OVER FOR TEXT DISP
     LAY
4Ø1Ø POKE 53248,25Ø:POKE 53249,25Ø:PO
     KE 5325Ø,25Ø:POKE 53251,25Ø
4Ø2Ø RETURN
```

Part Five

Fast Action

Chiseler

John Scarborough

Who writes arcade-quality games? John Scarborough is nineteen years old and learned machine language from a book. He wrote "Chiseler" on an Atari 400 with 16K, a cassette recorder, and an Assembler Editor cartridge. He didn't even have a full memory map. It doesn't take a college education or years of experience. It just takes patience, careful design, and love for what the computer can do. A little wizardry doesn't hurt, either.

It seems like an easy enough job. Four boards are laid out in front of you, and your task is to chisel them away. To help you, you have five sets of four chisels each. And not just ordinary tools — these chisels will hurl themselves at the boards and chip away strips of wood at a pretty reckless speed.

But it isn't as easy as it sounds. Your chisels also hurl themselves *away* from the boards each time they strike, and if you don't catch them and bounce them back, they'll get away for good. And you'll soon discover that every time you get rid of one board, another appears in its place.

The only way you can keep going on the job is either to be incredibly quick at catching chisels, or to get as many chisels as possible *behind* the new board when it appears. If you can trap them there, you can sit back and watch while they do the work for you. But don't doze off — you never know when they'll escape again, and you'll be back to work.

This is an arcade-speed game, but don't worry if you aren't an accomplished arcader. "Chiseler" begins at a slow enough speed that a beginner can pick it up pretty well. But it keeps its challenge, no matter how good you get.

You'll need a paddle controller. The game doesn't require any cartridges to play, but you will need either BASIC or an Assembler Editor to enter the program the first time.

Part Five

Entering Chiseler

There are two ways to enter and SAVE Chiseler. If you have an Assembler Editor, you can use it and the assembly listing, or you can use the "Machine Language Editor (MLX)" program found in Appendix C and the DATA listing. If you don't have an Assembler Editor, then you must use the MLX program to enter the DATA statements. It is important that those who use the MLX program read the article that accompanies it. When you use the MLX program, it may seem like extra work at first to have to type in two programs, but you'll save time later when you end up with a virtually error-free program.

Starting Up

Once you have entered the game and saved it on cassette or disk, you can start one of two ways.

If you entered the game from BASIC using MLX, then the game was saved as an autoboot program. If you have it on cassette, just remove all cartridges, put the cassette in the recorder, and turn on the computer while holding down the START button. You will hear one beep. Push PLAY on the recorder, then RETURN on the computer keyboard. The game will load in automatically. If Chiseler is on disk, turn on the disk drive, put in the disk, and turn on the computer.

If you entered the game with an Assembler Editor, then Chiseler must be loaded from the assembler cartridge or DOS.

From the assembler cartridge, use the LOAD command. Then the debug command G2800 will start the game.

If you created a binary file with MLX, then you must LOAD from the DOS menu. Choose option L, Binary Load. Enter the filename, then select M, RUN at $2800.

How to Play the Game

Chiseler comes up with two side walls and a back wall, with four colored boards stretching across from side to side. When you press the paddle trigger, the first of four chisels will come at you. You must move your paddle at the bottom of the screen to catch the chisel and hurl it back at the boards. It will bounce off the first board and come back toward the bottom of the screen.

Meanwhile, another chisel will start down toward you. If you catch it — and keep the first one going as well — two more chisels will come. As soon as you miss one chisel, new chisels will stop appearing, and you will never have more than four chisels at a time.

You will notice that as each chisel strikes the board, a thin strip of wood disappears. When all the wood is chipped away at one point, the chisels can get behind the first board, and start bouncing back and forth between the first and second boards. This can be repeated with every board.

When the last bit of a board has been destroyed, a new board will appear in its place. Try to have your chisel strike the last strip of wood from above, so the chisel will bounce back up on the screen and be trapped behind the new board. Then you can relax and watch the chisel do your work for you — until it breaks through again and starts coming back at you.

When the last chisel has got by you, that turn is over. To start the next turn, press the paddle trigger. Four new chisels will be launched. You get a total of five turns.

Scoring

Each strip of wood in the first board is worth one point; each in the second is worth two; in the third, three; and in the fourth, four.

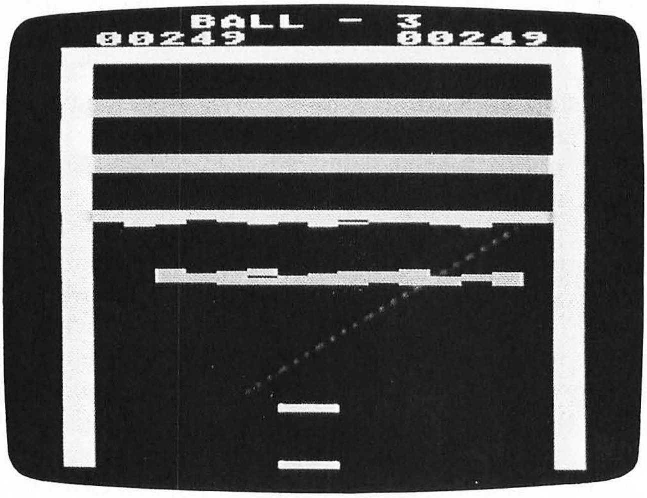
Speed

Every time a new board is put in place, the chisels on the screen speed up slightly. The more new boards, the faster the chisels. After a while, it gets going faster than human dexterity can handle. But try to handle it, because if you let the chisels get by you, they don't start slowly again on the next turn — they keep every bit of the speed they've built up.

Chisel Control

If the chisel strikes on the right half of your paddle, it will bounce off at an angle toward the right. If it strikes on the left half, it will bounce off toward the left. When you first start, you'll probably feel lucky to catch the chisel at all. But quite soon you'll start getting some finesse, and the chisels will go pretty much where you want them.

Part Five



This chiseler needs a lot of practice.

Entering Chiseler from BASIC with the Machine Language Editor: MLX

The last program in this book is the Machine Language Editor: MLX. It isn't a game — it's a utility to make sure that entering this long machine language program is easy and accurate.

MLX prompts you by asking for the program's starting address: enter 8192. Then MLX will prompt you again for the ending address: enter 11136. Finally it will ask for the RUN/initialization address: enter 10240. Start entering numbers exactly as they appear in this listing.

The seventh number on every line is a test number. When you enter that, MLX uses it to check to see if every number was entered correctly on that line. If it was, you will be prompted to enter the next line. If there was a mistake, you will hear a beep and you will be prompted to enter the same set of numbers again.

Entering Chiseler with the Assembler Editor

If you aren't a machine language programmer, don't worry. Just enter the program from BASIC with MLX. This listing is included for those who are learning machine language for their Atari's 6502 microprocessor. Instead of an inscrutable list of numbers, you can read the actual commands — with many remarks (which don't have to be entered), explaining exactly what is going on in every part of the program.

If you've ever felt the urge to get inside the arcade machine and see how *Pac-Man* or *Super Breakout* achieves those dazzling effects, this listing is for you. Once you have actually gone through the process of making the computer do tricks just by loading the right numbers into the Accumulator and the X and Y registers, performing the right operations on them, and then storing them into the right locations in memory, you'll realize that putting together a fantastic arcade game isn't such a mystery after all. You've got it right within your reach.

This program was written entirely in machine language, using the Assembler Editor cartridge. It can be assembled in a computer with only 16K of memory, then saved as a single program on cassette. To accomplish this, the program is divided into six subdivisions. This enables the 16K computer owner to assemble the programs without having to worry about running out of memory for the source code. Each unit must be typed in and assembled independently. This shouldn't be too monotonous if you read the remarks as you type. After all six have been assembled, the object code can be SAVED to cassette with this command:

```
SAVE#C:<2000,2B80
```

To retrieve the object code from cassette, enter

```
LOAD#C:<2000,2B80
```

These commands are utilized exactly like CLOAD and CSAVE.

Then the command DEBUG, followed by G2800, will start the game.

Program 5-1. Chiseler Using MLX

```
8192:162,000,169,000,157,000,232
8198:060,232,224,008,208,248,218
8204:169,255,157,000,060,232,117
8210:224,016,208,248,169,000,115
8216:133,176,169,062,133,177,106
8222:160,000,169,000,145,176,168
```

```
8228:200,208,251,166,177,232,246
8234:134,177,224,063,240,238,094
8240:169,034,141,006,062,169,117
8246:033,141,007,062,169,044,254
8252:141,008,062,141,009,062,227
8258:169,013,141,011,062,169,119
8264:016,141,013,062,141,023,212
8270:062,141,024,062,141,025,021
8276:062,141,026,062,141,027,031
8282:062,162,000,189,201,006,198
8288:157,033,062,232,224,005,041
8294:208,245,169,000,133,176,009
8300:169,062,133,177,160,042,083
8306:169,001,145,176,200,192,229
8312:059,208,249,160,062,169,003
8318:001,145,176,152,024,105,217
8324:016,168,169,001,145,176,039
8330:152,024,105,004,168,056,135
8336:201,241,144,233,166,177,026
8342:232,134,177,224,064,240,197
8348:005,160,006,076,125,032,048
8354:169,001,141,242,062,141,150
8360:002,063,169,000,133,176,199
8366:169,062,133,177,160,103,210
8372:162,068,138,145,176,232,077
8378:200,224,083,208,247,160,028
8384:163,162,147,138,145,176,099
8390:232,200,224,162,208,247,191
8396:160,223,162,098,138,145,106
8402:176,232,200,224,113,208,083
8408:247,169,063,133,177,160,141
8414:027,162,177,138,145,176,023
8420:232,200,224,192,208,247,251
8426:169,032,133,178,169,060,207
8432:133,179,160,000,152,145,241
8438:178,200,208,251,169,061,033
8444:133,179,169,000,145,178,032
8450:200,192,224,208,249,169,220
8456:032,133,178,169,060,133,201
8462:179,160,000,177,178,208,148
8468:008,200,192,120,208,247,227
8474:032,000,034,169,152,133,034
```

```
8480:178,160,000,177,178,208,165
8486:008,200,192,120,208,247,245
8492:032,049,034,169,016,133,221
8498:178,169,061,133,179,160,162
8504:000,177,178,208,008,200,059
8510:192,120,208,247,032,060,153
8516:034,169,136,133,178,160,110
8522:000,177,178,208,008,200,077
8528:192,120,208,247,032,071,182
8534:034,096,000,000,000,000,216
8540:000,000,000,000,000,000,092
8546:000,000,000,000,000,000,098
8552:000,000,000,000,000,000,104
8558:000,000,000,000,000,000,110
8564:000,000,000,000,000,000,116
8570:000,000,000,000,000,000,122
8576:000,000,000,000,000,000,128
8582:000,000,000,000,000,000,134
8588:000,000,000,000,000,000,140
8594:000,000,000,000,000,000,146
8600:000,000,000,000,000,000,152
8606:000,000,000,000,000,000,158
8612:000,000,000,000,000,000,164
8618:000,000,000,000,000,000,170
8624:000,000,000,000,000,000,176
8630:000,000,000,000,000,000,182
8636:000,000,000,000,000,000,188
8642:000,000,000,000,000,000,194
8648:000,000,000,000,000,000,200
8654:000,000,000,000,000,000,206
8660:000,000,000,000,000,000,212
8666:000,000,000,000,000,000,218
8672:000,000,000,000,000,000,224
8678:000,000,000,000,000,000,230
8684:000,000,000,000,000,000,236
8690:000,000,000,000,000,000,242
8696:000,000,000,000,000,000,248
8702:000,000,169,032,133,178,254
8708:169,060,133,179,160,000,193
8714:177,178,208,034,200,192,231
8720:120,208,247,160,000,169,152
8726:255,145,178,200,192,120,088
```

```
8732:208,249,169,041,141,210,022
8738:006,173,209,006,201,001,118
8744:240,006,056,233,001,141,205
8750:209,006,096,169,152,133,043
8756:178,169,060,133,179,076,079
8762:008,034,169,016,133,178,084
8768:169,061,133,179,076,008,178
8774:034,169,136,133,178,169,121
8780:061,133,179,076,008,034,055
8786:000,000,000,000,000,000,082
8792:000,000,000,000,000,000,088
8798:000,000,000,000,000,000,094
8804:000,000,000,000,000,000,100
8810:000,000,000,000,000,000,106
8816:000,000,000,000,000,000,112
8822:000,000,000,000,000,000,118
8828:000,000,000,000,000,000,124
8834:000,000,000,000,000,000,130
8840:000,000,000,000,000,000,136
8846:000,000,000,000,000,000,142
8852:000,000,000,000,000,000,148
8858:000,000,000,000,000,000,154
8864:000,000,000,000,000,000,160
8870:000,000,000,000,000,000,166
8876:000,000,000,000,000,000,172
8882:000,000,000,000,000,000,178
8888:000,000,000,000,000,000,184
8894:000,000,000,000,000,000,190
8900:000,000,000,000,000,000,196
8906:000,000,000,000,000,000,202
8912:000,000,000,000,000,000,208
8918:000,000,000,000,000,000,214
8924:000,000,000,000,000,000,220
8930:000,000,000,000,000,000,226
8936:000,000,000,000,000,000,232
8942:000,000,000,000,000,000,238
8948:000,000,000,000,000,000,244
8954:000,000,000,000,000,000,250
8960:169,062,141,047,002,169,078
8966:001,141,008,208,169,239,004
8972:141,192,002,169,015,141,160
8978:193,002,141,194,002,141,179
```

```
8984:195,002,141,199,002,169,220
8990:048,141,007,212,169,003,098
8996:141,029,208,169,000,133,204
9002:180,169,048,133,181,160,145
9008:000,152,145,180,200,208,165
9014:251,166,181,232,134,181,175
9020:224,056,208,242,169,255,190
9026:141,196,052,141,197,052,077
9032:141,198,052,169,255,141,004
9038:221,052,141,222,052,141,139
9044:223,052,169,000,141,196,097
9050:006,141,197,006,141,198,011
9056:006,141,199,006,169,001,106
9062:141,175,006,162,000,173,247
9068:010,210,056,201,075,144,036
9074:248,024,201,186,176,243,168
9080:157,176,006,232,224,004,151
9086:208,235,162,000,169,047,179
9092:157,180,006,169,112,157,145
9098:171,006,232,224,004,208,215
9104:241,162,000,173,010,210,172
9110:041,001,157,184,006,232,003
9116:224,004,208,243,162,000,229
9122:169,001,157,188,006,232,147
9128:224,004,208,246,162,000,244
9134:169,001,157,164,006,232,135
9140:224,004,208,246,173,112,123
9146:002,074,024,105,070,141,090
9152:160,006,169,255,056,237,051
9158:160,006,056,201,177,144,174
9164:002,169,177,141,000,208,133
9170:141,163,006,096,000,000,104
9176:000,000,000,000,000,000,216
9182:000,000,000,000,000,000,222
9188:000,000,000,000,000,000,228
9194:000,000,000,000,000,000,234
9200:000,000,000,000,000,000,240
9206:000,000,000,000,000,000,246
9212:000,000,000,000,174,162,076
9218:006,189,164,006,208,001,064
9224:096,189,184,006,208,012,191
9230:189,176,006,056,233,001,163
```

```
9236:157,176,006,076,035,036,250
9242:189,176,006,024,105,001,015
9248:157,176,006,189,188,006,242
9254:208,012,189,180,006,056,177
9260:233,001,157,180,006,076,185
9266:061,036,189,180,006,024,034
9272:105,001,157,180,006,189,182
9278:180,006,056,201,219,144,100
9284:044,208,009,173,175,006,171
9290:056,233,001,141,175,006,174
9296:169,001,141,199,006,189,017
9302:180,006,201,252,208,021,186
9308:169,000,224,003,208,006,190
9314:141,199,002,076,107,036,147
9320:157,193,002,169,000,157,014
9326:164,006,096,189,188,006,247
9332:208,003,076,228,036,189,088
9338:180,006,201,193,240,004,178
9344:201,218,208,096,189,176,192
9350:006,056,205,163,006,144,202
9356:087,056,233,016,024,205,249
9362:163,006,176,078,173,199,173
9368:006,208,029,162,001,189,235
9374:195,006,208,017,169,001,242
9380:157,195,006,173,175,006,108
9386:024,105,001,141,175,006,110
9392:076,184,036,232,224,004,164
9398:208,229,174,162,006,169,106
9404:000,157,188,006,169,017,213
9410:141,000,210,189,176,006,148
9416:056,233,005,141,160,006,033
9422:173,163,006,056,205,160,201
9428:006,144,008,169,000,157,184
9434:184,006,076,228,036,169,149
9440:001,157,184,006,189,176,169
9446:006,201,072,208,010,169,128
9452:001,157,184,006,169,029,014
9458:141,002,210,189,176,006,198
9464:201,190,208,010,169,000,002
9470:157,184,006,169,029,141,172
9476:002,210,189,180,006,201,024
9482:047,208,010,169,001,157,090
```

```
9488:188,006,169,029,141,002,039
9494:210,189,176,006,157,001,249
9500:208,189,180,006,133,182,158
9506:224,003,208,005,169,051,182
9512:076,047,037,138,024,105,211
9518:053,133,183,160,000,152,215
9524:145,182,200,224,003,208,246
9530:005,169,003,076,066,037,158
9536:169,192,145,182,200,145,073
9542:182,200,145,182,200,169,124
9548:000,145,182,096,000,000,243
9554:000,000,000,000,000,000,082
9560:000,000,000,000,000,000,088
9566:000,000,000,000,000,000,094
9572:000,000,000,000,000,000,100
9578:000,000,000,000,000,000,106
9584:000,000,000,000,000,000,112
9590:000,000,000,000,000,000,118
9596:000,000,000,000,000,000,124
9602:000,000,000,000,000,000,130
9608:000,000,000,000,000,000,136
9614:000,000,000,000,000,000,142
9620:000,000,000,000,000,000,148
9626:000,000,000,000,000,000,154
9632:000,000,000,000,000,000,160
9638:000,000,000,000,000,000,166
9644:000,000,000,000,000,000,172
9650:000,000,000,000,000,000,178
9656:000,000,000,000,000,000,184
9662:000,000,000,000,000,000,190
9668:000,000,000,000,000,000,196
9674:000,000,000,000,000,000,202
9680:000,000,000,000,000,000,208
9686:000,000,000,000,000,000,214
9692:000,000,000,000,000,000,220
9698:000,000,000,000,000,000,226
9704:000,000,000,000,000,000,232
9710:000,000,000,000,000,000,238
9716:000,000,000,000,000,000,244
9722:000,000,000,000,000,000,250
9728:174,162,006,189,171,006,196
9734:240,007,056,233,001,157,188
```

```
9740:171,006,096,189,180,006,148
9746:056,201,062,144,047,024,040
9752:201,070,176,003,076,071,109
9758:038,056,201,086,144,034,077
9764:024,201,094,176,003,076,098
9770:084,038,056,201,110,144,163
9776:021,024,201,118,176,003,079
9782:076,097,038,056,201,134,144
9788:144,008,024,201,142,176,243
9794:003,076,110,038,096,169,046
9800:070,141,170,006,169,004,120
9806:032,123,038,076,170,038,043
9812:169,094,141,170,006,169,065
9818:019,032,123,038,076,170,036
9824:038,169,118,141,170,006,226
9830:169,034,032,123,038,076,062
9836:170,038,169,142,141,170,170
9842:006,169,049,032,123,038,019
9848:076,170,038,141,169,006,208
9854:169,080,141,194,006,189,137
9860:176,006,024,205,194,006,231
9866:176,011,173,170,006,056,218
9872:253,180,006,141,168,006,130
9878:096,173,194,006,024,105,236
9884:008,141,194,006,172,169,078
9890:006,200,140,169,006,076,247
9896:131,038,173,169,006,024,197
9902:201,032,176,011,169,000,251
9908:133,178,169,060,133,179,008
9914:076,203,038,169,000,133,037
9920:178,169,061,133,179,173,061
9926:169,006,056,233,032,173,099
9932:169,006,010,010,010,072,225
9938:174,162,006,189,188,006,167
9944:240,011,104,024,105,008,196
9950:056,237,168,006,076,237,234
9956:038,104,024,105,008,056,051
9962:237,168,006,168,177,178,144
9968:208,001,096,169,000,145,091
9974:178,173,200,006,141,004,180
9980:210,032,007,033,032,000,054
9986:042,174,162,006,169,009,052
```

```
9992:157,171,006,189,188,006,213
9998:208,006,169,001,157,188,231
10004:006,096,169,000,157,188,124
10010:006,096,000,000,000,000,128
10016:000,000,000,000,000,000,032
10022:000,000,000,000,000,000,038
10028:000,000,000,000,000,000,044
10034:000,000,000,000,000,000,050
10040:000,000,000,000,000,000,056
10046:000,000,072,138,072,169,001
10052:008,162,056,141,010,212,145
10058:141,022,208,142,023,208,050
10064:169,224,141,009,212,169,236
10070:112,141,000,002,104,170,103
10076:104,064,000,000,000,000,004
10082:000,000,000,000,000,000,098
10088:000,000,000,000,000,000,104
10094:000,000,072,138,072,152,032
10100:072,169,152,162,120,160,183
10106:056,141,010,212,141,022,192
10112:208,142,023,208,140,024,105
10118:208,169,060,141,009,212,165
10124:169,224,141,000,002,104,012
10130:168,104,170,104,064,000,244
10136:000,000,000,000,000,000,152
10142:000,000,000,000,000,000,158
10148:000,000,000,000,000,000,164
10154:000,000,000,000,000,000,170
10160:000,000,000,000,000,000,176
10166:000,000,000,000,000,000,182
10172:000,000,000,000,000,000,188
10178:000,000,000,000,000,000,194
10184:000,000,000,000,000,000,200
10190:000,000,000,000,000,000,206
10196:000,000,000,000,000,000,212
10202:000,000,000,000,000,000,218
10208:072,138,072,169,088,162,157
10214:216,141,010,212,141,023,205
10220:208,142,024,208,169,064,027
10226:141,000,002,104,170,104,251
10232:064,000,000,000,000,000,056
10238:000,000,032,000,043,162,235
```

```
10244:000,169,080,157,201,006,105
10250:232,224,005,208,248,169,072
10256:000,141,008,210,169,001,033
10262:141,192,006,169,020,141,179
10268:209,006,032,000,032,032,083
10274:000,035,169,000,141,210,077
10280:006,169,166,141,001,210,221
10286:141,003,210,169,170,141,112
10292:005,210,141,007,210,169,026
10298:000,141,000,210,141,002,040
10304:210,141,004,210,141,006,008
10310:210,032,184,035,173,124,060
10316:002,208,248,169,001,141,077
10322:195,006,169,000,133,176,249
10328:169,062,133,177,173,192,226
10334:006,024,105,016,160,013,162
10340:145,176,174,210,006,240,027
10346:012,202,142,210,006,169,079
10352:060,141,006,210,076,124,217
10358:040,169,000,141,006,210,172
10364:169,000,133,077,141,000,132
10370:210,141,002,210,141,004,070
10376:210,169,000,141,162,006,056
10382:032,119,041,032,000,036,146
10388:032,000,038,162,001,142,011
10394:162,006,189,195,006,240,184
10400:009,032,119,041,032,000,137
10406:036,032,000,038,162,002,180
10412:142,162,006,189,195,006,104
10418:240,009,032,119,041,032,139
10424:000,036,032,000,038,162,196
10430:003,142,162,006,189,195,119
10436:006,240,009,032,119,041,131
10442:032,000,036,032,000,038,084
10448:173,199,006,240,126,162,090
10454:000,189,164,006,208,008,021
10460:232,224,004,208,246,076,186
10466:241,040,189,195,006,208,081
10472:106,232,224,004,208,233,215
10478:076,241,040,173,192,006,198
10484:201,005,208,063,169,006,128
10490:133,176,169,062,133,177,076
```

```
10496:160,000,169,039,145,176,177
10502:200,169,033,145,176,200,161
10508:169,045,145,176,200,169,148
10514:037,145,176,200,200,169,177
10520:047,145,176,200,169,054,047
10526:145,176,200,169,037,145,134
10532:176,200,169,050,145,176,184
10538:032,184,035,173,031,208,193
10544:201,006,208,246,076,015,032
10550:040,032,184,035,173,031,037
10556:208,201,006,208,003,076,250
10562:015,040,173,124,002,208,116
10568:238,174,192,006,232,142,032
10574:192,006,032,000,035,174,005
10580:209,006,142,211,006,032,178
10586:184,035,173,031,208,201,154
10592:006,208,003,076,015,040,188
10598:160,160,136,208,253,174,169
10604:211,006,202,142,211,006,118
10610:208,229,076,084,040,189,172
10616:180,006,024,201,080,176,019
10622:011,169,072,141,200,006,213
10628:169,005,141,213,006,096,250
10634:024,201,100,176,011,169,051
10640:108,141,200,006,169,004,004
10646:141,213,006,096,024,201,063
10652:128,176,011,169,144,141,157
10658:200,006,169,003,141,213,126
10664:006,096,169,217,141,200,229
10670:006,169,002,141,213,006,199
10676:096,000,000,000,000,000,020
10682:000,000,000,000,000,000,186
10688:000,000,000,000,000,000,192
10694:000,000,000,000,000,000,198
10700:000,000,000,000,000,000,204
10706:000,000,000,000,000,000,210
10712:000,000,000,000,000,000,216
10718:000,000,000,000,000,000,222
10724:000,000,000,000,000,000,228
10730:000,000,000,000,000,000,234
10736:000,000,000,000,000,000,240
10742:000,000,000,000,000,000,246
```

```
10748:000,000,000,000,169,000,165
10754:133,176,169,062,133,177,084
10760:174,213,006,202,142,213,190
10766:006,240,006,032,024,042,108
10772:076,008,042,096,174,175,079
10778:006,138,072,032,038,042,098
10784:104,170,202,208,246,096,034
10790:160,027,177,176,201,025,036
10796:208,009,169,016,145,176,255
10802:136,192,022,208,241,024,105
10808:105,001,145,176,169,033,173
10814:141,207,006,169,023,141,237
10820:208,006,172,207,006,177,076
10826:176,056,233,064,172,208,215
10832:006,056,209,176,144,023,182
10838:208,039,172,208,006,200,151
10844:140,208,006,172,207,006,063
10850:200,140,207,006,192,038,113
10856:208,220,076,127,042,162,171
10862:000,160,023,177,176,024,158
10868:105,064,157,201,006,232,113
10874:200,192,028,208,242,160,128
10880:033,162,000,189,201,006,207
10886:145,176,200,232,224,005,092
10892:208,245,096,000,000,000,177
10898:000,000,000,000,000,000,146
10904:000,000,000,000,000,000,152
10910:000,000,000,000,000,000,158
10916:000,000,000,000,000,000,164
10922:000,000,000,000,000,000,170
10928:000,000,000,000,000,000,176
10934:000,000,000,000,000,000,182
10940:000,000,000,000,000,000,188
10946:000,000,000,000,000,000,194
10952:000,000,000,000,000,000,200
10958:000,000,000,000,000,000,206
10964:000,000,000,000,000,000,212
10970:000,000,000,000,000,000,218
10976:000,000,000,000,000,000,224
10982:000,000,000,000,000,000,230
10988:000,000,000,000,000,000,236
10994:000,000,000,000,000,000,242
```

```
11000:000,000,000,000,000,000,248
11006:000,000,169,112,141,000,164
11012:006,141,001,006,169,198,013
11018:141,002,006,169,000,141,213
11024:003,006,169,062,141,004,145
11030:006,169,134,141,005,006,227
11036:169,006,141,006,006,141,241
11042:007,006,141,008,006,141,087
11048:009,006,141,010,006,169,125
11054:134,141,011,006,169,006,001
11060:162,000,157,012,006,232,109
11066:224,017,208,248,169,065,221
11072:141,029,006,169,000,141,038
11078:030,006,169,006,141,031,197
11084:006,169,000,141,047,002,185
11090:141,048,002,169,006,141,077
11096:049,002,169,064,141,000,001
11102:002,169,039,141,001,002,192
11108:169,192,141,014,212,169,229
11114:034,141,047,002,096,000,170
11120:000,000,000,000,000,000,112
11126:000,000,000,000,000,000,118
11132:000,000,000,000,000,000,124
```

Program 5-2a. Chiseler Using Assembler Editor — Program Control

```
10 ;
20 ;PROGRAM CONTROL
30 ;
40  *=$2800
50 PTRGST=$6C2
60 BALOFS=$6A2
70 BALLNB=$6C0
80 PDLCNT=$23B8
90 LAUNCH=$6C3
0100 STIMER=53279
0110 BRKFRQ=$6C8
0120 PTRIG0=$27C
0130 ATRACT=$4D
0140 BALLS=$6AF
0150 NOMORE=$6C7
0160 BALIFE=$6A4
```

```
Ø17Ø SCRCNT=$2AØØ
Ø18Ø SCRNPL=$BØ
Ø19Ø SCRNPH=$B1
Ø2ØØ BALSPD=$6D1
Ø21Ø HIGH1=$6C9
Ø22Ø RSTFRQ=$6D2
Ø23Ø TMPDLY=$6D3
Ø24Ø BL1VPS=$6B4
Ø25Ø AUDCTL=$D2Ø8
Ø26Ø BRKVAL=$6D5
Ø27Ø STDLST=$2BØØ
Ø28Ø SETPLF=$2ØØØ
Ø29Ø SETPMG=$23ØØ
Ø3ØØ BALLCT=$24ØØ
Ø31Ø BRKCNT=$26ØØ
Ø32Ø ;
Ø33Ø ;SET UP DISPLAY LIST
Ø34Ø  JSR STDLST
Ø35Ø ;
Ø36Ø ;SET HIGH SCORE EQUAL TO ØØØØØ
Ø37Ø  LDX #Ø
Ø38Ø  LDA #$5Ø
Ø39Ø CLRHGH STA HIGH1,X
Ø4ØØ  INX
Ø41Ø  CPX #5
Ø42Ø  BNE CLRHGH
Ø43Ø ;
Ø44Ø ;THIS IS THE INTRODUCTION TO
Ø45Ø ;THE MAIN LOOP
Ø46Ø ;
Ø47Ø ;SET THE AUDIO CONTROL
Ø48Ø START LDA #Ø
Ø49Ø  STA AUDCTL
Ø5ØØ ;
Ø51Ø ;SET BALL NUMBER TO 1
Ø52Ø  LDA #1
Ø53Ø  STA BALLNB
Ø54Ø ;
Ø55Ø ;SET BALL SPEED
Ø56Ø  LDA #2Ø
Ø57Ø  STA BALSPD
Ø58Ø ;
```

```
0590 ;SET PLAYFIELD
0600  JSR SETPLF
0610 ;
0620 ;SET P/M GRAPHICS
0630  JSR SETPMG
0640 ;
0650 ;CLEAR BRICK RESET FREQUENCY
0660  LDA #0
0670  STA RSTFRQ
0680 ;
0690 ;SET AUDIO CONTROL
0700  LDA #$A6
0710  STA $D201
0720  STA $D203
0730  LDA #$AA
0740  STA $D205
0750  STA $D207
0760 ;
0770 ;MAKE SURE FREQUENCY IS OFF
0780  LDA #0
0790  STA $D200
0800  STA $D202
0810  STA $D204
0820  STA $D206
0830 ;
0840 ;WAIT FOR PADDLE TRIGGER
0850 TRGCNT JSR PDLCNT
0860  LDA PTRIG0
0870  BNE TRGCNT
0880 ;
0890 ;SHOW THAT BALL 1
0900 ;HAS BEEN LAUNCHED
0910  LDA #1
0920  STA LAUNCH
0930 ;
0940 ;THE MAIN LOOP
0950 ;
0960 ;SET SCREEN POINTER TO $3E00
0970 MAINLP LDA #$0
0980  STA SCRNPL
0990  LDA #$3E
1000  STA SCRNPH
```

```
1010 ;
1020 ;DISPLAY BALL NUMBER
1030  LDA BALLNB
1040  CLC
1050  ADC #$10
1060  LDY #$D          ;DISPLAY
1070  STA (SCRNPL),Y ;BALL NUMBER
1080 ;
1090 ;IF A BLOCK HAS RECENTLY BEEN
1100 ;RESET, THEN PRODUCE SOUND
1110  LDX RSTFRQ
1120  BEQ NOSND
1130  DEX
1140  STX RSTFRQ
1150  LDA #$3C
1160  STA $D206
1170  JMP FIXATC
1180 NOSND LDA #0
1190  STA $D206
1200 ;
1210 ;INHIBIT ATTRACT MODE
1220 FIXATC LDA #0
1230  STA ATRACT
1240 ;
1250 ;TURN OFF FREQUENCY
1260  STA $D200
1270  STA $D202
1280  STA $D204
1290 ;
1300 ;ENABLE BALL 1 CONTROL
1310 BALCNT LDA #0
1320  STA BALOFS
1330 ;
1340 ;UPDATE BALL 1'S STATUS
1350  JSR GETFRQ
1360  JSR BALLCT
1370  JSR BRKCNT
1380 ;
1390 ;ENABLE BALL 2 CONTROL
1400  LDX #1
1410  STX BALOFS
1420 ;
```

```
1430 ;HAS BALL 2 BEEN LAUNCHED?
1440  LDA LAUNCH,X
1450  BEQ CHKLN2
1460 ;
1470 ;IF SO, THEN UPDATE ITS STATUS
1480  JSR GETFRQ
1490  JSR BALLCT
1500  JSR BRKCNT
1510 ;
1520 ;FOLLOW THE SAME PROCEDURE FOR
1530 ;BALLS THREE AND FOUR
1540 CHKLN2 LDX #2
1550  STX BALOFS
1560  LDA LAUNCH,X
1570  BEQ CHKLN3
1580  JSR GETFRQ
1590  JSR BALLCT
1600  JSR BRKCNT
1610 CHKLN3 LDX #3
1620  STX BALOFS
1630  LDA LAUNCH,X
1640  BEQ CHKEND
1650  JSR GETFRQ
1660  JSR BALLCT
1670  JSR BRKCNT
1680 ;
1690 ;CHECK IF ANY BALLS
1700 ;ARE STILL ALIVE
1710 CHKEND LDA NOMORE
1720  BEQ DELAY
1730  LDX #0
1740 LFECHK LDA BALIFE,X
1750  BNE CHKVOD
1760  INX
1770  CPX #4
1780  BNE LFECHK
1790  JMP DEAD
1800 CHKVOD LDA LAUNCH,X
1810  BNE DELAY
1820  INX
1830  CPX #4
1840  BNE LFECHK
```

```
1850  JMP DEAD
1860 DEAD LDA BALLNB
1870  CMP #5
1880  BNE GTPTRG
1890 ;
1900 ;ALL BALLS ARE DEAD
1910 ;DISPLAY "GAME OVER"
1920  LDA #$6
1930  STA SCRNPL
1940  LDA #$3E
1950  STA SCRNPH
1960  LDY #0
1970 GMEOVR LDA #$27 ;G
1980  STA (SCRNPL),Y
1990  INY
2000  LDA #$21 ;A
2010  STA (SCRNPL),Y
2020  INY
2030  LDA #$2D ;M
2040  STA (SCRNPL),Y
2050  INY
2060  LDA #$25 ;E
2070  STA (SCRNPL),Y
2080  INY
2090  INY
2100  LDA #$2F ;O
2110  STA (SCRNPL),Y
2120  INY
2130  LDA #$36 ;V
2140  STA (SCRNPL),Y
2150  INY
2160  LDA #$25 ;E
2170  STA (SCRNPL),Y
2180  INY
2190  LDA #$32 ;R
2200  STA (SCRNPL),Y
2210 ;
2220 ;WAIT FOR USER TO PRESS START
2230 CHKSTM JSR PDLCNT
2240  LDA STIMER
2250  CMP #6
2260  BNE CHKSTM
```

```
2270 ;
2280 ;START OVER
2290  JMP START
2300 ;
2310 ;ALL BALLS ARE DEAD, BUT NOT
2320 ;ALL BALL SETS
2330 ;WAIT FOR TRIGGER OR START KEY
2340 GTPTRG JSR PDLCNT
2350  LDA STIMER
2360  CMP #6
2370  BNE TRIGER
2380 ;
2390 ;START OVER
2400  JMP START
2410 TRIGER LDA PTRIGØ
2420  BNE GTPTRG
2430 ;
2440 ;INCREMENT BALL NUMBER
2450  LDX BALLNB
2460  INX
2470  STX BALLNB
2480 ;
2490 ;RESET
2500  JSR SETPMG
2510 ;
2520 ;DELAY PROGRAM EXECUTION
2530 DELAY LDX BALSPD
2540  STX TMPDLY
2550 DLY1
2560  JSR PDLCNT
2570  LDA STIMER
2580  CMP #6
2590  BNE CNTDLY
2600  JMP START
2610 CNTDLY LDY #$AØ
2620 DLY2 DEY
2630  BNE DLY2
2640  LDX TMPDLY
2650  DEX
2660  STX TMPDLY
2670  BNE DLY1
2680 ;
```

```
2690 ;EXECUTE MAIN LOOP AGAIN
2700  JMP MAINLP
2710 ;
2720 ;SET BRICK FREQUENCY
2730 ;AND BRICK VALUE
2740 GETFRQ LDA BL1VPS,X
2750  CLC
2760  CMP #$50
2770  BCS FRQ2
2780  LDA #$48
2790  STA BRKFRQ
2800  LDA #5 ;ADD 1 TO COMPENSATE
2810  STA BRKVAL ;FOR OFFSET
2820  RTS
2830 FRQ2 CLC
2840  CMP #$64
2850  BCS FRQ3
2860  LDA #$6C
2870  STA BRKFRQ
2880  LDA #4
2890  STA BRKVAL
2900  RTS
2910 FRQ3 CLC
2920  CMP #$80
2930  BCS FRQ4
2940  LDA #$90
2950  STA BRKFRQ
2960  LDA #3
2970  STA BRKVAL
2980  RTS
2990 FRQ4 LDA #$D9
3000  STA BRKFRQ
3010  LDA #2
3020  STA BRKVAL
3030  RTS
```

Program 5-2b. Chiseler Using Assembler Editor — Display Setup

```
10 ;
20 ;SET UP THE DISPLAY LIST
30 ;
```

Part Five

```
40   *=$2B00
50  ;
60  ;BLANK THE TOP 16 SCAN LINES
70   LDA #$70
80   STA $600
90   STA $601
0100  ;
0110  ;SET UP 1 LINE OF IR MODE 6 (BA
     SIC
0120  ;MODE 1), ENABLE DISPLAY LIST
0130  ;INTERRUPT AND LOAD THE MEMORY
0140  ;SCAN COUNTER WITH $3E00
0150   LDA #$C6
0160   STA $602
0170   LDA #$00
0180   STA $603
0190   LDA #$3E
0200   STA $604
0210  ;
0220  ;SET UP 1 LINE OF IR MODE 1 AND
0230  ;ENABLE 2ND DLI
0240   LDA #$86
0250   STA $605
0260  ;
0270  ;SET UP 5 LINES OF IR MODE 1
0280   LDA #$6
0290   STA $606
0300   STA $607
0310   STA $608
0320   STA $609
0330   STA $60A
0340  ;
0350  ;SET UP 1 LINE OF IR MODE 1 AND
0360  ;ENABLE 3RD DLI
0370   LDA #$86
0380   STA $60B
0390  ;
0400  ;SET UP 17 LINES OF IR MODE 1
0410   LDA #$6
0420   LDX #0
0430 MODE17 STA $60C,X
0440   INX
```

```
0450  CPX #17
0460  BNE MODE17
0470 ;
0480 ;WAIT FOR VERTICAL BLANK, THEN
0490 ;EXECUTE DISPLAY LIST AGAIN
0500  LDA #$41
0510  STA $61D
0520  LDA #$00
0530  STA $61E
0540  LDA #$6
0550  STA $61F
0560 DLISTL=$230
0570 DLISTH=$231
0580 DMACTL=$22F
0590 NMIEN=$D40E
0600 DLILOW=$200
0610 DLIHGH=$201
0620 ;DISABLE DMA
0630  LDA #0
0640  STA DMACTL
0650 ;
0660 ;INFORM ANTIC OF DISPLAY LIST
0670 ;LOCATION
0680  STA DLISTL
0690  LDA #$6
0700  STA DLISTH
0710 ;
0720 ;INFORM ANTIC OF DISPLAY LIST
0730 ;INTERRUPT LOCATION
0740  LDA #$40
0750  STA DLILOW
0760  LDA #$27
0770  STA DLIHGH
0780 ;
0790 ;ENABLE DISPLAY LIST INTERUPTS
0800  LDA #$C0
0810  STA NMIEN
0820 ;
0830 ;ENABLE DMA
0840  LDA #$22
0850  STA DMACTL
0860  RTS
```

Program 5-2c. Chiseler Using Assembler Editor — Display List Interrupts

```
10 ;
20 ;DISPLAY LIST INTERRUPTS
30 ;
40 DLILOW=$200
50 CHBASE=$D409
60 WSYNC=$D40A
70 COLPFØ=$DØ16
80 COLPF1=$DØ17
90 COLPF2=$DØ18
Ø1ØØ ;
Ø11Ø ;THIS DLI WILL AFFECT
Ø12Ø ;THE SECOND MODE LINE
Ø13Ø  *=$274Ø
Ø14Ø ;
Ø15Ø ;SAVE REGISTERS
Ø16Ø  PHA
Ø17Ø  TXA
Ø18Ø  PHA
Ø19Ø ;
Ø2ØØ ;SET COLPFØ (SCORE) TO GREY
Ø21Ø ;AND COLPF1 (HIGH) TO RED
Ø22Ø  LDA #$Ø8
Ø23Ø  LDX #$38
Ø24Ø  STA WSYNC
Ø25Ø  STA COLPFØ
Ø26Ø  STX COLPF1
Ø27Ø ;
Ø28Ø ;ENABLE STANDARD CHBASE
Ø29Ø  LDA #$EØ
Ø3ØØ  STA CHBASE
Ø31Ø ;
Ø32Ø ;ENABLE 2ND DLI
Ø33Ø  LDA #$7Ø
Ø34Ø  STA DLILOW
Ø35Ø ;
Ø36Ø ;RESTORE REGISTERS
Ø37Ø  PLA
Ø38Ø  TAX
Ø39Ø  PLA
Ø4ØØ ;
```

```
0410 ;RETURN FROM INTERRUPT
0420  RTI
0430 ;
0440 ;THIS DLI WILL AFFECT
0450 ;MODE LINES 2-6
0460  *=$2770
0470  PHA
0480  TXA
0490  PHA
0500  TYA
0510  PHA
0520 ;
0530 ;SET COLPF0 (WALLS) TO LIGHT BL
   UE
0540 ;SET COLPF1 (BLOCK 1) TO BLUE
0550 ;SET COLPF2 (BLOCK 2) TO RED
0560  LDA #$98
0570  LDX #$78
0580  LDY #$38
0590  STA WSYNC
0600  STA COLPF0
0610  STX COLPF1
0620  STY COLPF2
0630 ;
0640 ;CHANGE CHARACTER BASE TO $3CE0
0650  LDA #$3C
0660  STA CHBASE
0670 ;
0680 ;ENABLE 3RD DLI
0690  LDA #$E0
0700  STA DLILOW
0710  PLA
0720  TAY
0730  PLA
0740  TAX
0750  PLA
0760  RTI
0770 ;
0780 ;THIS DLI WILL AFFECT
0790 ;MODE LINES 8-24
0800  *=$27E0
0810  PHA
```

```
0820  TXA
0830  PHA
0840 ;
0850 ;SET COLPF0 (BLOCK 3) TO VIOLET
0860 ;AND COLPF1 (BLOCK 4) TO YELLOW
    -
0870 ;GREEN
0880  LDA #$58
0890  LDX #$D8
0900  STA WSYNC
0910  STA COLPF1
0920  STX COLPF2
0930 ;
0940 ;ENABLE 1ST DLI
0950  LDA #$40
0960  STA DLILOW
0970  PLA
0980  TAX
0990  PLA
1000  RTI
```

Program 5-2d. Chiseler Using Assembler Editor — Playfield Setup

```
10 ;
20 ;SET PLAYFIELD
30 ;
40  *=$2000
50 NEWCHB=$3C00
60 SCRNPL=$B0
70 SCRNPH=$B1
80 CHBSPL=$B2
90 CHBSPH=$B3
0100 FILRW1=$2200
0110 FILRW2=$2231
0120 FILRW3=$223C
0130 FILRW4=$2247
0140 HIGH1=$6C9
0150 ;
0160 ;SET CHAR 0 TO A BLANK SPACE
0170  LDX #0
0180  LDA #$0
```

```
Ø19Ø SBLANK STA NEWCHB,X
Ø2ØØ  INX
Ø21Ø  CPX #8
Ø22Ø  BNE SBLANK
Ø23Ø ;
Ø24Ø ;SET CHAR 1 TO A SOLID SPACE
Ø25Ø  LDA #$FF
Ø26Ø SSOLID STA NEWCHB,X
Ø27Ø  INX
Ø28Ø  CPX #16
Ø29Ø  BNE SSOLID
Ø3ØØ ;
Ø31Ø ;SCRNPL POINTS TO THE
Ø32Ø ;SCREEN MEMORY REGION
Ø33Ø  LDA #$ØØ
Ø34Ø  STA SCRNPL
Ø35Ø  LDA #$3E
Ø36Ø  STA SCRNPH
Ø37Ø ;
Ø38Ø ;CLEAR SCREEN MEMORY REGION
Ø39Ø CLRSMR LDY #Ø
Ø4ØØ  LDA #Ø
Ø41Ø STRSMR STA (SCRNPL),Y
Ø42Ø  INY
Ø43Ø  BNE STRSMR
Ø44Ø  LDX SCRNPH
Ø45Ø  INX
Ø46Ø  STX SCRNPH
Ø47Ø  CPX #$3F ;SCREEN MEMORY
Ø48Ø  BEQ CLRSMR
Ø49Ø ;
Ø5ØØ ;PRINT "BALL - Ø" ON TOP LINE
Ø51Ø  LDA #$22   ;B
Ø52Ø  STA $3EØ6
Ø53Ø  LDA #$21   ;A
Ø54Ø  STA $3EØ7
Ø55Ø  LDA #$2C   ;L
Ø56Ø  STA $3EØ8 ;L
Ø57Ø  STA $3EØ9
Ø58Ø  LDA #$D    ;-
Ø59Ø  STA $3EØB
Ø6ØØ  LDA #$1Ø   ;Ø
```

```
0610 ;
0620 ;DISPLAY SCORE (00000)
0630  STA $3E0D
0640  STA $3E17
0650  STA $3E18
0660  STA $3E19
0670  STA $3E1A
0680  STA $3E1B
0690 ;
0700 ;DISPLAY HIGH (XXXXX)
0710  LDX #0
0720 DSPHGH LDA HIGH1,X
0730  STA $3E21,X
0740  INX
0750  CPX #5
0760  BNE DSPHGH
0770 ;
0780 ;RESET SCREEN MEMORY POINTER
0790  LDA #$0
0800  STA SCRNPL
0810  LDA #$3E
0820  STA SCRNPH
0830 ;
0840 ;DRAW TOP WALL
0850  LDY #$2A
0860  LDA #1
0870 DRWTOP STA (SCRNPL),Y
0880  INY
0890  CPY #$3B
0900  BNE DRWTOP
0910 ;
0920 ;DRAW LEFT AND RIGHT WALLS
0930 DRWALS LDY #$3E
0940 DRWLFT LDA #1
0950  STA (SCRNPL),Y
0960  TYA
0970  CLC
0980  ADC #16
0990  TAY
1000  LDA #1
1010 DRWRGT STA (SCRNPL),Y
1020  TYA
```

```
1030  CLC
1040  ADC #4
1050  TAY
1060  SEC
1070  CMP #$F1
1080  BCC DRWLFT
1090  LDX SCRNPH
1100  INX
1110  STX SCRNPH
1120  CPX #$40
1130  BEQ TWOMOR
1140  LDY #6
1150  JMP DRWLFT
1160 ;
1170 ;PERFECT WALL
1180 TWOMOR LDA #$1
1190  STA $3EF2
1200  STA $3F02
1210  LDA #$0
1220  STA SCRNPL
1230  LDA #$3E
1240  STA SCRNPH
1250 ;
1260 ;LOAD EACH BRICK ROW WITH
1270 ;15 DISTINCT CHARACTERS
1280 ;
1290 ;ROW 1 GETS CHARS 4-12
1300 ;(#$40 IS ADDED FOR COLPF1)
1310  LDY #$67
1320  LDX #$44
1330 FIXRW1 TXA
1340  STA (SCRNPL),Y
1350  INX
1360  INY
1370  CPX #$53
1380  BNE FIXRW1
1390 ;
1400 ;ROW 2 GETS CHARS 13-21
1410 ;(#$80 IS ADDED FOR COLPF2)
1420  LDY #$A3
1430  LDX #$93
1440 FIXRW2 TXA
```

```
1450  STA (SCRNPL),Y
1460  INX
1470  INY
1480  CPX #$A2
1490  BNE FIXRW2
1500 ;
1510 ;ROW 3 USES CHARS 22-30
1520 ;(ADD #$40 FOR COLPF1)
1530  LDY #$DF
1540  LDX #$62
1550 FIXRW3 TXA
1560  STA (SCRNPL),Y
1570  INX
1580  INY
1590  CPX #$71
1600  BNE FIXRW3
1610  LDA #$3F
1620  STA SCRNPH
1630 ;
1640 ;ROW 4 USES CHARS 31-3F
1650 ;(ADD #$80 FOR COLPF2)
1660  LDY #$1B
1670  LDX #$B1
1680 FIXRW4 TXA
1690  STA (SCRNPL),Y
1700  INX
1710  INY
1720  CPX #$C0
1730  BNE FIXRW4
1740 ;
1750 ;CLEAR CHARACTER BASE
1760 ;FROM $3C20-3DFF
1770  LDA #$20
1780  STA CHBSPL
1790  LDA #$3C
1800  STA CHBSPH
1810  LDY #0
1820  TYA
1830 CLRCHB STA (CHBSPL),Y
1840  INY
1850  BNE CLRCHB
1860  LDA #$3D
```

```
1870  STA CHBSPH
1880  LDA #0
1890 CHBTWO STA (CHBSPL),Y
1900  INY
1910  CPY #$E0
1920  BNE CHBTWO
1930 ;
1940 ;CHECK IF ANY ROWS HAVE BEEN
1950 ;CLEARED.  IF THEY HAVE, THEN
1960 ;RESET THEM
1970 CHKRW1 LDA #$20
1980  STA CHBSPL
1990  LDA #$3C
2000  STA CHBSPH
2010  LDY #0
2020 GETRW1 LDA (CHBSPL),Y
2030  BNE CHKRW2
2040  INY
2050  CPY #120 ;CHECK 120 BRICKS
2060  BNE GETRW1
2070  JSR FILRW1
2080 CHKRW2 LDA #$98
2090  STA CHBSPL
2100  LDY #0
2110 GETRW2 LDA (CHBSPL),Y
2120  BNE CHKRW3
2130  INY
2140  CPY #120
2150  BNE GETRW2
2160  JSR FILRW2
2170 CHKRW3 LDA #$10
2180  STA CHBSPL
2190  LDA #$3D
2200  STA CHBSPH
2210  LDY #0
2220 GETRW3 LDA (CHBSPL),Y
2230  BNE CHKRW4
2240  INY
2250  CPY #120
2260  BNE GETRW3
2270  JSR FILRW3
2280 CHKRW4 LDA #$88
```

```
2290  STA CHBSPL
2300  LDY #0
2310 GETRW4 LDA (CHBSPL),Y
2320  BNE FINISH
2330  INY
2340  CPY #120
2350  BNE GETRW4
2360  JSR FILRW4
2370 FINISH RTS
```

Program 5-2e. Chiseler Using Assembler Editor — Reset Brick Rows

```
10 ;
20 ;SHOULD BRICK ROWS BE RESET?
30 ;
40  *=$2200
50 CHBSPL=$B2
60 CHBSPH=$B3
70 BALSPD=$6D1
80 RSTFRQ=$6D2
90 ;
0100 ;MAKE CHARACTER BASE POINTER
0110 ;POINT TO BRICK ROW 1
0120 BRKROW LDA #$20
0130  STA CHBSPL
0140  LDA #$3C
0150  STA CHBSPH
0160 ;
0170 ;IS BRICK ROW CLEAR?
0180 SETY LDY #$0
0190 CHKROW LDA (CHBSPL),Y
0200 ;
0210 ;IF NOT THEN RETURN
0220  BNE NOTCLR
0230  INY
0240  CPY #$78
0250  BNE CHKROW
0260 ;
0270 ;IF SO, THEN REFILL BRICK ROW
0280  LDY #$0
0290  LDA #$FF
```

```
0300 FILROW STA (CHBSPL),Y
0310  INY
0320  CPY #$78
0330  BNE FILROW
0340 ;
0350 ;PRODUCE SOUND TO CORRESPOND
0360 ;WITH RESET
0370  LDA #$29
0380  STA RSTFRQ
0390 ;
0400 ;SPEED UP BALLS
0410  LDA BALSPD
0420  CMP #1
0430  BEQ NOTCLR
0440  SEC
0450  SBC #1
0460  STA BALSPD
0470 NOTCLR RTS
0480 ;
0490 ;MAKE CHARACTER BASE POINTER
0500 ;POINT TO BRICK ROW 2
0510 BRKRW2 LDA #$98
0520  STA CHBSPL
0530  LDA #$3C
0540  STA CHBSPH
0550  JMP SETY
0560 ;
0570 ;MAKE CHARACTER BASE POINTER
0580 ;POINT TO BRICK ROW 3
0590 BRKRW3 LDA #$10
0600  STA CHBSPL
0610  LDA #$3D
0620  STA CHBSPH
0630  JMP SETY
0640 ;
0650 ;MAKE CHARACTER BASE POINTER
0660 ;POINT TO BRICK ROW 4
0670 BRKRW4 LDA #$88
0680  STA CHBSPL
0690  LDA #$3D
0700  STA CHBSPH
0710  JMP SETY
```

Program 5-2f. Chiseler Using Assembler Editor — Player/Missile Graphics

```
10  ;
20  ;SET PLAYER/MISSILE GRAPHICS
30  ;
40   *=$2300
50  PMBASE=$D4Ø7
60  PMBSPL=$B4
70  PMBSPH=$B5
80  GRACTL=$DØ1D
90  DMACTL=$22F
Ø1ØØ  SIZEPØ=$DØØ8
Ø11Ø  COLPMØ=$2CØ
Ø12Ø  COLPM1=$2C1
Ø13Ø  COLPM2=$2C2
Ø14Ø  COLPM3=$2C3
Ø15Ø  COLPF3=$2C7
Ø16Ø  NMIEN=$D4ØE
Ø17Ø  PADDLØ=$27Ø
Ø18Ø  HPOSPØ=$DØØØ
Ø19Ø  RANDOM=$D2ØA
Ø2ØØ  TEMP=$6AØ
Ø21Ø  PDLHPS=$6A3
Ø22Ø  BALLS=$6AF
Ø23Ø  NOMORE=$6C7
Ø24Ø  LAUNCH=$6C3
Ø25Ø  AVOID=$6AB
Ø26Ø  BALIFE=$6A4
Ø27Ø  BL1HPS=$6BØ
Ø28Ø  BL1VPS=$6B4
Ø29Ø  BL1HDR=$6B8
Ø3ØØ  BL1VDR=$6BC
Ø31Ø  ;
Ø32Ø  ;ENABLE PLAYER/MISSILE DMA
Ø33Ø   LDA #62
Ø34Ø   STA DMACTL
Ø35Ø  ;
Ø36Ø  ;SET PLAYER Ø (PADDLE)
Ø37Ø  ;TO DOUBLE WIDTH
Ø38Ø   LDA #1
Ø39Ø   STA SIZEPØ
```

```
0400 ;
0410 ;COLOR PADDLE ORANGE-GREEN
0420  LDA #$EF
0430  STA COLPM0
0440 ;
0450 ;COLOR BALLS WHITE
0460  LDA #$F
0470  STA COLPM1
0480  STA COLPM2
0490  STA COLPM3
0500  STA COLPF3
0510 ;
0520 ;SET P/M BASE TO $3000
0530  LDA #$30
0540  STA PMBASE
0550 ;
0560 ;ENABLE PLAYER/MISSILE GRAPHICS
0570  LDA #3
0580  STA GRACTL
0590 ;
0600 ;CLEAR P/M GRAPHICS TABLE
0610  LDA #$0
0620  STA PMBSPL
0630  LDA #$30
0640  STA PMBSPH
0650  LDY #0
0660  TYA
0670 CLRPMB STA (PMBSPL),Y
0680  INY
0690  BNE CLRPMB
0700  LDX PMBSPH
0710  INX
0720  STX PMBSPH
0730  CPX #$38
0740  BNE CLRPMB
0750 ;
0760 ;DRAW PADDLE
0770  LDA #$FF
0780  STA $34C4
0790  STA $34C5
0800  STA $34C6
0810  LDA #$FF
```

```
0820  STA $34DD
0830  STA $34DE
0840  STA $34DF
0850 ;
0860 ;MAKE SURE BALLS DON'T LAUNCH
0870 ;BEFORE THEIR TIME IS DUE
0880  LDA #0
0890  STA $6C4 ;LAUNCH+1
0900  STA $6C5
0910  STA $6C6
0920 ;
0930 ;WHEN NOMORE=1, NO MORE BALLS
0940 ;WILL BE LAUNCHED.  NOMORE WILL
0950 ;BE SET TO 1 WHEN USER MISSES
0960 ;A BALL
0970  STA NOMORE
0980 ;
0990 ;BALLS HOLDS THE NUMBER OF BALL
    S
1000 ;ON THE FIELD AT ONE TIME
1010 ;THIS WILL BE INITIALIZED TO 1
1020  LDA #1
1030  STA BALLS
1040 ;
1050 ;SET EACH BALL TO A RANDOM
1060 ;HORIZONTAL POSITION
1070 ;BETWEEN #$4B AND #$BA
1080  LDX #0
1090 RNDHPS LDA RANDOM
1100  SEC
1110  CMP #$4B
1120  BCC RNDHPS
1130  CLC
1140  CMP #$BA
1150  BCS RNDHPS
1160  STA BL1HPS,X
1170  INX
1180  CPX #4
1190  BNE RNDHPS
1200 ;
1210 ;SET EACH BALL'S VERTICAL
1220 ;POSITION TO TOP OF FIELD
```

```
1230  LDX #0
1240 FIXVPS LDA #$2F
1250  STA BL1VPS,X
1260 ;
1270 ;BALLS MUST AVOID BRICKS
1280 ;UNTIL THEY HIT THE PADDLE
1290  LDA #$70
1300  STA AVOID,X
1310  INX
1320  CPX #4
1330  BNE FIXVPS
1340 ;
1350 ;SET EACH BALL TO A RANDOM
1360 ;HORIZONTAL DIRECTION
1370  LDX #0
1380 RNDHDR LDA RANDOM
1390  AND #1
1400  STA BL1HDR,X
1410  INX
1420  CPX #4
1430  BNE RNDHDR
1440 ;
1450 ;ALL BALLS MUST MOVE DOWN
1460 ;WHEN THEY ARE LAUNCHED
1470  LDX #0
1480 FIXVDR LDA #1
1490  STA BL1VDR,X
1500  INX
1510  CPX #4
1520  BNE FIXVDR
1530 ;
1540 ;GIVE EACH BALL LIFE
1550  LDX #0
1560 ACTVTE LDA #1
1570  STA BALIFE,X
1580  INX
1590  CPX #4
1600  BNE ACTVTE
1610 ;HORIZONTAL PADDLE POSITION
1620 ;MUST CORRESPOND TO PADDLE 0
1630 PDLCNT LDA PADDL0
1640  LSR A
```

```
1650  CLC
1660  ADC #$46
1670  STA TEMP
1680  LDA #$FF
1690  SEC
1700  SBC TEMP
1710  SEC
1720  CMP #$B1
1730  BCC STRHPS
1740  LDA #$B1
1750 STRHPS STA HPOSPØ
1760 ;
1770 ;THE POSITION OF THE PADDLE
1780 ;WILL BE NEEDED LATER
1790  STA PDLHPS
1800  RTS
```

Closeout

L.L. Beh

"Closeout" is a fast action game which has been enjoyed by both children and adults.

This program just fits into a 16K Atari. Almost all lines contain multiple statements, so make no alterations unless you have a bigger machine.

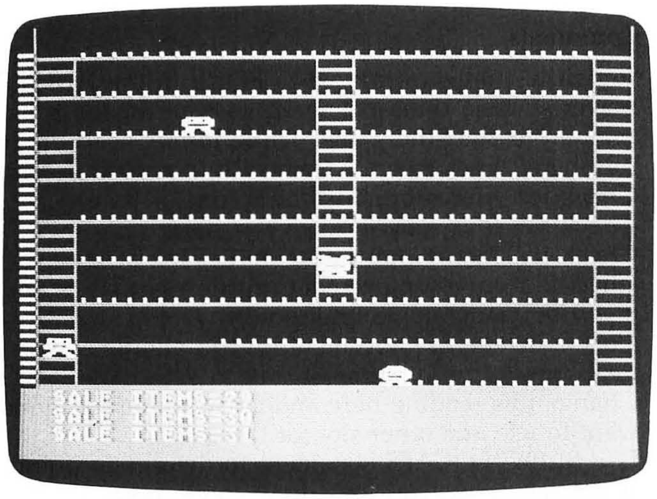
Scrambling for Bargains

There's a huge sale going on at a local department store. You arrive at the multistory building hungry for bargains. Boldly, you enter the store and look around — and see bargains galore. A real sale! You start gathering up sale items, but then become aware of a strange group of shoppers. Wherever you go, they follow you around. Soon you learn their true intentions — they are out to stop you at all costs, so they can have the store to themselves. What's worse, they're armed with ray guns from the Toy Department which are modified to work.

The object of "Closeout" is to snatch up as many sale items as possible while evading the hostile bargain hunters. Don't let them get too close, because they'll either capture you or shoot you. Some of them can shoot farther than others. You can shoot back with the slingshot you bought in the Sporting Goods Department (50 percent off), but since slingshots require two hands to shoot, you must drop 25 sale items each time you use it.

You have only one chance and about three minutes of play. Extra time is awarded for higher scores. The remaining time is indicated on the left, and colors change as time runs out (there can be up to nine different colors on the screen at a time). When your score surpasses 25 points, you can shoot your slingshot, but remember, it costs you 25 points. You can only shoot horizontally, by aiming the joystick and pressing the fire button. The best strategy is to shoot only when cornered.

Part Five



```
SALE ITEMS=29
SALE ITEMS=30
SALE ITEMS=31
```

How many bargains can you find in "Closeout"?

After you have typed in the program, save it twice on a disk or tape, then type RUN. There will be a short initialization pause, and the screen will clear to GRAPHICS 7. The four players will appear, and the floors and stairways will be drawn. Short instructions will appear. Plug your joystick into port one. Press the fire button to start.

Your shopper is on the ground floor in the bottom left corner. Use the joystick to move left and right, or up and down stairs. You must be directly under the stairs to use them, and to exit onto a floor you must be standing exactly on it.

The program might run faster on U.S. Ataris than on my New Zealand model. American televisions use the NTSC standard, which allows 60 vertical blank interrupts per second, while New Zealand's PAL standard allows only 50. On the other hand, the 6502 Central Processing Unit chip in our Ataris is clocked at three megahertz, as opposed to 1.8 MHz in U.S. Ataris, so the two factors might cancel each other out.

Good luck! I'd like to know who can better my score of 1200 points.

Since I omitted REM statements from the program to save space, here is a short explanation:

Part Five

Line No.	Comments
10	Initialize. Jump to line 209 to POKE in the P/M utility and player shapes, then to line 112 to wait for the trigger to be pressed to start the game. These jumps keep the lower numbered lines free for frequently accessed statements.
20-23	Check for your moves up the stairs.
30-33	Check for your moves down the stairs.
37-44	Check for enemy movement on the left stairs.
55-59	Check for enemy movement on the right stairs and also give the enemy some brainpower.
60-65	Check for enemy movement on the middle stairs.
73-76	Read joystick zero (leftmost slot) to determine your moves. Change this reading here and at two other locations if you want to use any other slot for the game.
78	Use BASIC LOCATE statement to determine points scored.
79	If the sale items run low, draw some more.
80-82	Determine movement of Player 2 — Enemy No. 1.
84-86	Determine movement of Player 3 — Enemy No. 2.
87-89	Determine movement of Player 4 — Enemy No. 3.
90	Check if you can shoot.
93-95	Check to see if you are in the enemy shooting range. Note: Some enemies can shoot farther than others.
96	Game timekeeper.
97	Go back to start of loop.
100-101	Draw dots — sale items.
105-108	Draw time bar graph and erase the portion of time that has run out; also award extra time for high scores.
109-114	You got shot! So these lines get you off the visible side of the screen, and wait for the trigger to be pressed to start a new game.
209	Only simple constants and variables are used in the program. These constants and variables are used all over the place to conserve RAM; they are also used for the line numbers. These make the program look very untidy, but this is the only way I can get this program to run on my 16K Atari.
210-215	POKE the P/M utility and players' shapes and colors into RAM.
216-225	Constants and variables for the stairs and building levels; draw it out in Graphics 7.
230	Data for P/M graphics utility.
236-237	Last eight DATAs in line 236 and all of line 237 are for the P/M shapes. Change these if you like different shapes.

Program 5-3. Closeout

```
10 GRAPHICS 7:SETCOLOR 2,Ø,Ø:POKE 75
   2,1:? "{14 SPACES}CLOSEOUT!":GOSUB
    2Ø9:GOSUB 112:Y1=181:GOTO C
2Ø IF X1=J THEN IF Y1>A AND Y1<=B OR
    Y1>C AND Y1<=D OR Y1>E AND Y1<=I
    THEN Y1=Y1-3:RETURN
21 IF X1=L THEN IF Y1>A AND Y1<=E OR
    Y1>F AND Y1<=I THEN Y1=Y1-3:RETU
   RN
22 IF X1=K THEN IF Y1>A AND Y1<=G TH
   EN Y1=Y1-3:RETURN
23 RETURN
3Ø IF X1=J THEN IF Y1>=A AND Y1<B OR
    Y1>=C AND Y1<D OR Y1>=E AND Y1<I
    THEN Y1=Y1+3:RETURN
31 IF X1=K THEN IF Y1>=A AND Y1<G TH
   EN Y1=Y1+3:RETURN
32 IF X1=L THEN IF Y1>=A AND Y1<E OR
    Y1>=F AND Y1<I THEN Y1=Y1+3
33 RETURN
37 IF Y5=I OR Y5=H OR Y5=G OR Y5=F O
   R Y5=E OR Y5=D OR Y5=C OR Y5=B OR
    Y5=A THEN GOSUB 42:RETURN
41 RETURN
42 Z=INT(RND(Ø)*3):IF Z=Ø THEN IF Y5
   =I OR Y5=H OR Y5=G OR Y5=F OR Y5=
   D OR Y5=B THEN X9=Ø:Y9=-3:RETURN
43 IF Z=1 THEN IF Y5=A OR Y5=C OR Y5
   =E OR Y5=F OR Y5=G OR Y5=H THEN X
   9=Ø:Y9=3:RETURN
44 X9=4:Y9=Ø:RETURN
55 IF Y5=I OR Y5=H OR Y5=G OR Y5=F O
   R Y5=E OR Y5=D OR Y5=C OR Y5=B OR
    Y5=A THEN GOSUB 57
56 RETURN
57 IF Y5>Y1 THEN IF Y5=B OR Y5=C OR
   Y5=D OR Y5=E OR Y5=G OR Y5=H OR Y
   5=I THEN X9=Ø:Y9=-3:RETURN
58 IF Y5<Y1 THEN IF Y5=A OR Y5=B OR
   Y5=C OR Y5=D OR Y5=F OR Y5=G OR Y
   5=H THEN X9=Ø:Y9=3:RETURN
```

```
59 X9=-4:Y9=Ø:RETURN
6Ø IF Y5=G OR Y5=F OR Y5=E OR Y5=D O
   R Y5=C OR Y5=B OR Y5=A THEN GOSUB
    62
61 RETURN
62 Z=INT(RND(Ø)*4):IF Z=Ø THEN IF Y5
   =E OR Y5=F OR Y5=D OR Y5=C OR Y5=
   B OR Y5=A THEN X9=Ø:Y9=3:RETURN
63 IF Z=1 THEN IF Y5=B OR Y5=C OR Y5
   =D OR Y5=E OR Y5=G OR Y5=F THEN X
   9=Ø:Y9=-3:RETURN
64 IF Z=2 THEN X9=-4:Y9=Ø:RETURN
65 X9=4:Y9=Ø:RETURN
73 S=STICK(Ø):IF S=14 OR S=1Ø OR S=6
    THEN GOSUB 2Ø
74 IF S=13 OR S=9 OR S=5 THEN GOSUB
   3Ø
75 IF S=7 AND X1<L THEN IF Y1=A OR Y
   1=B OR Y1=C OR Y1=D OR Y1=E OR Y1
   =F OR Y1=G OR Y1=H OR Y1=I THEN X
   1=X1+4
76 IF S=11 AND X1>J THEN IF Y1=A OR
   Y1=B OR Y1=C OR Y1=D OR Y1=E OR Y
   1=F OR Y1=G OR Y1=H OR Y1=I THEN
   X1=X1-4
78 Q=X1-42:R=(Y1-25)/2:LOCATE Q,R,Z:
   IF Z=1 THEN COLOR 4:PLOT Q,R:T=T+
   1:T1=T1+1:? "SALE ITEMS=";:? T
79 IF T1>T2 THEN GOSUB 1ØØ
8Ø POKE 77,Ø:POKE M,X1:POKE N,Y1:SOU
   ND Ø,Y1,1Ø,7:IF X2=J THEN Y5=Y2:X
   9=X7:Y9=Y7:GOSUB A:X7=X9:Y7=Y9
81 IF X2=K THEN Y5=Y2:X9=X7:Y9=Y7:GO
   SUB 6Ø:X7=X9:Y7=Y9
82 IF X2=L THEN Y5=Y2:X9=X7:Y9=Y7:GO
   SUB B:X7=X9:Y7=Y9
84 X2=X2+X7:Y2=Y2+Y7:POKE M+1,X2:POK
   E N+1,Y2:IF X3=J THEN Y5=Y3:X9=X8
   :Y9=Y8:GOSUB A:X8=X9:Y8=Y9
85 IF X3=K THEN Y5=Y3:X9=X8:Y9=Y8:GO
   SUB 6Ø:X8=X9:Y8=Y9
```

```
86 IF X3=L THEN Y5=Y3:X9=X8:Y9=Y8:GO
   SUB B:X8=X9:Y8=Y9
87 X3=X3+X8:Y3=Y3+Y8:POKE M+2,X3:POK
   E N+2,Y3:IF X4=J THEN Y5=Y4:X9=X6
   :Y9=Y6:GOSUB A:X6=X9:Y6=Y9
88 SOUND Ø,Ø,Ø,Ø:IF X4=K THEN Y5=Y4:
   X9=X6:Y9=Y6:GOSUB 6Ø:X6=X9:Y6=Y9
89 IF X4=L THEN Y5=Y4:X9=X6:Y9=Y6:GO
   SUB B:X6=X9:Y6=Y9
9Ø X4=X4+X6:Y4=Y4+Y6:POKE M+3,X4:POK
   E N+3,Y4:IF STRIG(Ø)=Ø AND T>25 T
   HEN IF Y1=Y2 OR Y1=Y3 OR Y1=Y4 TH
   EN GOSUB H
93 IF Y1=Y2 AND 7Ø>ABS(X1-X2) THEN U
   =X2-44:V=(Y2-31)/2:GOSUB E
94 IF Y1=Y3 AND 6Ø>ABS(X1-X3) THEN U
   =X3-44:V=(Y3-31)/2:GOSUB E
95 IF Y1=Y4 AND 55>ABS(X1-X4) THEN U
   =X4-44:V=(Y4-31)/2:GOSUB E
96 IF PEEK(19)>A1 THEN GOSUB 1Ø5
97 GOTO C
1ØØ SOUND Ø,Ø,Ø,Ø:COLOR 1:SETCOLOR Ø
    ,C1,9
1Ø1 FOR Y=6 TO 79 STEP 9:FOR W=16 TO
     148 STEP 4:PLOT W,Y:NEXT W:NEXT
     Y:T1=Ø:RETURN
1Ø5 C1=INT(RND(Ø)*15):SETCOLOR 1,C1,
    8:COLOR 4:PLOT Ø,V1:DRAWTO 3,V1:
    A1=A1+1:V1=V1+2:IF A1<>41 THEN R
    ETURN
1Ø6 IF T<T3 THEN GOSUB 112:RETURN
1Ø7 ? "EXTRA 3.5 MIN":T3=T3+T4:? "NE
    XT BONUS AT ";:? T3:T4=T4+1ØØ
1Ø8 COLOR 2:FOR Y=Ø TO 78 STEP 2:PLO
    T Ø,Y:DRAWTO 3,Y:NEXT Y:A1=1:V1=
    Ø:POKE 19,Ø:POKE 2Ø,Ø:RETURN
1Ø9 SOUND Ø,2,6,15:COLOR 2:PLOT U,V:
    DRAWTO Q,R:POKE M,1:POKE N,247:C
    OLOR 4:PLOT U,V:DRAWTO Q,R
112 SOUND Ø,Ø,Ø,Ø:GOSUB 216:GOSUB 1Ø
    Ø:? :? "GAME OVER, SALES FOUND="
    ;:? T:T=Ø:? "To play-Press FIRE"
```

```
113 IF STRIG(Ø)<>Ø THEN 113
114 ? "Extra TIME at 2ØØ":GOSUB 1Ø8:
    T=Ø:T3=2ØØ:T4=3ØØ:RETURN
163 W=Ø:T=T-25:? "SALE ITEMS=";:? T:
    IF Y1=Y2 THEN U=X2-44:V=(Y2-31)/
    2:I1=M+1:J1=N+1:GOSUB I:Y2=Z:X2=
    L:RETURN
164 IF Y1=Y3 THEN U=X3-44:V=(Y3-31)/
    2:I1=M+2:J1=N+2:GOSUB I:Y3=Z:X3=
    L:RETURN
165 IF Y1=Y4 THEN U=X4-44:V=(Y4-31)/
    2:I1=M+3:J1=N+3:GOSUB I:Y4=Z:X4=
    L:RETURN
181 SOUND Ø,1,6,15:COLOR 1:PLOT Q,R:
    DRAWTO U,V:COLOR 4:PLOT Q,R:DRAW
    TO U,V:SOUND Ø,Ø,Ø,Ø:POKE I1,1:P
    OKE J1,247
189 Z=V*2+67:IF Z=A OR Z=B OR Z=C OR
     Z=D OR Z=E OR Z=F OR Z=I OR Z=G
     THEN GOSUB T2:RETURN
19Ø V=3:GOTO 189
2Ø9 A=37:B=55:C=73:D=91:E=1Ø9:F=127:
    G=145:H=163:I=181:J=54:K=126:L=1
    98:M=53248:N=178Ø:O=1784:P=7Ø4:T
    2=22Ø
21Ø ? "Please wait":FOR Y=1536 TO 17
    Ø6:READ Z:POKE Y,Z:NEXT Y:FOR Y=
    1774 TO 1787:POKE Y,Ø:NEXT Y:PM=
    PEEK(1Ø6)-32
211 PMBASE=256*PM:FOR Y=PMBASE+1Ø23
    TO PMBASE+2Ø47:POKE Y,Ø:NEXT Y:F
    OR Y=PMBASE+1Ø25 TO PMBASE+1Ø32:
    READ Z
212 POKE Y,Z:NEXT Y:FOR Y=PMBASE+128
    1 TO PMBASE+1288:READ Z:POKE Y,Z
    :NEXT Y:FOR Y=PMBASE+1537 TO PMB
    ASE+1544
213 READ Z:POKE Y,Z:NEXT Y:FOR Y=PMB
    ASE+1793 TO PMBASE+18ØØ:READ Z:P
    OKE Y,Z:NEXT Y:POKE P+2,76:POKE
    P+3,2Ø4
214 POKE P,252:POKE P+1,14Ø:POKE 559
```

```
    ,62:POKE 623,1:POKE 1788,PM+4:PO
    KE 53277,3:POKE 54279,PM:X=USR(1
    696)
215 POKE O,8:POKE O+1,8:POKE O+2,8:P
    OKE O+3,8:RETURN
216 X1=J:Y1=I:Y2=B:X2=62:Y3=E:X3=154
    :Y4=G:X4=122:X6=4:Y6=Ø:X7=-4:Y7=
    Ø:X8=-4:Y8=Ø
22Ø COLOR 3:SETCOLOR 2,15-C1,5:FOR Y
    =7 TO 79 STEP 9:PLOT 4,Y:DRAWTO
    159,Y:NEXT Y
221 PLOT 4,Ø:DRAWTO 4,79:PLOT 159,Ø:
    DRAWTO 159,79:PLOT 149,79:DRAWTO
     149,52:FOR Y=52 TO 79 STEP 3:PL
    OT 149,Y
222 DRAWTO 159,Y:NEXT Y:PLOT 14,79:D
    RAWTO 14,43:FOR Y=43 TO 79 STEP
    3:PLOT 4,Y:DRAWTO 14,Y:NEXT Y:PL
    OT 149,43
223 DRAWTO 149,7:FOR Y=7 TO 43 STEP
    3:PLOT 149,Y:DRAWTO 159,Y:NEXT Y
    :PLOT 14,7:DRAWTO 14,16:FOR Y=7
    TO 16 STEP 3
224 PLOT 4,Y:DRAWTO 14,Y:NEXT Y:PLOT
     77,7:DRAWTO 77,61:PLOT 87,7:DRA
    WTO 87,61:FOR Y=7 TO 61 STEP 3
225 PLOT 77,Y:DRAWTO 87,Y:NEXT Y:PLO
    T 14,25:DRAWTO 14,34:FOR Y=25 TO
     34 STEP 3:PLOT 4,Y:DRAWTO 14,Y:
    NEXT Y:RETURN
23Ø DATA 162,3,189,244,6,24Ø,89,56,2
    21,24Ø,6,24Ø,83,141,254,6,1Ø6,14
    1,255,6,142,253,6,24,169,Ø,1Ø9,2
    53,6,24,1Ø9
231 DATA 252,6,133,2Ø4,133,2Ø6,189,2
    4Ø,6,133,2Ø3,173,254,6,133,2Ø5,1
    89,248,6,17Ø,232,46,255,6,144,16
    ,168,177,2Ø3
232 DATA 145,2Ø5,169,Ø,145,2Ø3,136,2
    Ø2,2Ø8,244,76,87,6,16Ø,Ø,177,2Ø3
    ,145,2Ø5,169,Ø,145,2Ø3,2ØØ,2Ø2,2
    Ø8,244,174
```

```
234 DATA 253,6,173,254,6,157,240,6,1
    89,236,6,240,48,133,203,24,138,1
    41,253,6,109,235,6,133,204,24,17
    3,253,6,109
235 DATA 252,6,133,206,189,240,6,133
    ,205,189,248,6,170,160,0,177,203
    ,145,205,200,202,208,248,174,253
    ,6,169,0,157
236 DATA 236,6,202,48,3,76,2,6,76,98
    ,228,0,0,104,169,7,162,6,160,0,3
    2,92,228,96,60,126,219,255,195,1
    26,60,231
237 DATA 126,219,255,129,126,102,195
    ,129,195,126,90,255,129,255,60,1
    02,126,90,126,195,255,60,102,195
```

Part Five

SKI!

Charles Brannon and E.H. Foerster

"Ski!" is a fine-scrolling arcade-style game that lets you test your skill at electronic winter sports. It will run in 16K if you remove all REM statements. (Remove only the text; *leave in the line number and REM.) Unfortunately, Ski! will* not *work on an XL-model computer, unless you plug in an Atari BASIC cartridge.*

Ski down Pine Mountain and never leave the warmth of your home. That's exactly what "Ski!" allows you to do.

The object of Ski! is to ski down the slalom course without running into any rocks, trees, or other obstacles while trying to go between the flags. Using a joystick plugged into the first port to control the skier, you "gobble up" bonus points planted in the snow. You can move the joystick left or right to turn. You can also position your player up or down to change difficulty, points, and maneuverability.

The higher you go, the faster the scene scrolls, and the more points you win. The higher speeds necessitate fast response. The novice will want to position himself a little below midway up the screen. That way, you have room to pull back if you need to duck. If you hit a rock, tree, or flag, you crash, and start over at the bottom of the screen. You lose fifty points for every crash.

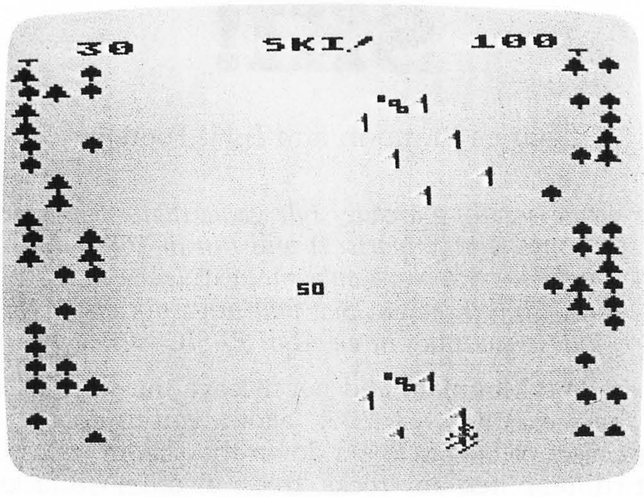
Up the Hill

Every time you play the game, a random ski course is generated. If you wish, you can see the screen scroll in reverse as the course is being laid out. Your computer will buzz when the game is ready to play. Press FIRE to begin.

Fine Scrolling

Fine scrolling couples coarse scrolling (which moves the pointers to screen memory around) with a special feature of the ANTIC chip.

To fine scroll, you set a special bit in every line of the display list you wish to scroll. You then scroll one scan line at a time by storing numbers from 0-15 in VSCROL. When you



A tricky maneuver between the flagpoles in "Ski!"

reach the limit of ANTIC's fine-scrolling resolution (8 scan lines in GRAPHICS 1), you reset VSCROL and then coarsely scroll a full eight scan lines. Coarse scrolling is described in *COMPUTE!'s Second Book of Atari.* Machine language is required for fine scrolling, since you must reset VSCROL and perform the coarse scroll almost simultaneously, or else you get a jumpy, unpleasant display.

Interfacing to BASIC

The fine-scrolling routine could be written as a USR statement, but BASIC would have to call it every time a scroll was needed, and this would be too slow. We need to periodically update the screen in a way that's not dependent on BASIC.

The Vertical Blank Interrupt (VBI) is perfect for this task. Every 1/60th of a second, the scroll routine is called to update the screen. BASIC can control the speed with memory location zero. POKEing a number from 1-255 controls the speed from one (fastest) to 255. A zero will stop the scrolling, although the vertical blank routine will still be "hooked up." BASIC sets up the VBLANK scrolling routine by passing the address of the Load Memory Scan counter to change in the display list (which can be found on a normal screen with LMS = PEEK(560) + 256*PEEK(561) + 4) and the number of lines to scroll. BASIC can PEEK location 1 to see how many full lines still need to be scrolled.

The VBLANK routine will stop scrolling when it runs out of lines, and memory location 1 will hold a zero. You could use the machine language routine in your own programs, but since it is not general-purpose, you will be limited to unidirectional scrolling in GRAPHICS 1. Be sure to use the "disable routine" (A =USR(1638)) to remove the VBI routine from the system.

An ANTIC Anomaly

It's not mentioned anywhere as far as I know, but the address of the start of your screen memory for fine scrolling should start on a 4K boundary. ANTIC apparently cannot cross a 4K boundary, so if your screen buffer (that holds the rocks, trees, etc.) is too long, ANTIC can get confused and start displaying nonsense. Another thing to watch for: when using a vertical blank routine, be sure to include a CLD (Clear Decimal) at the start of the program. If you don't, your arithmetic will be foiled every time BASIC calls the floating point routines (which use BCD math).

Strings are used extensively in the BASIC program, to prevent memory conflicts. A string is used to hold the display list, the screen memory area, the player/missile memory, and the shapes for the player. The screen memory area and the player/missile address are insured to be on proper page boundaries by modification of the Variable Value Table. Because of this, line 100 must be typed first, in order for the program to work properly.

Typing in Ski!

It is extremely important that you follow these typing instructions carefully. (It is a good idea to read all the directions first.)

1. Type in Program 5-4.

   Do not run program at this time.

2. LIST this program to disk (LIST "D:LOADER") or cassette (LIST "C:"). You might want to make two copies.
3. Turn off your computer; then turn it back on.

4. Type in Program 5-5.

Do not run program at this time.

5. LIST this program to disk (LIST "D:SKI.LST) or cassette (LIST "C:"). Again you may wish to make two copies.

6. Type NEW and ENTER the loader (ENTER "D:LOADER or ENTER "C:").

7. RUN the loader. If it runs correctly then go to step 8; if it doesn't run correctly, or the computer crashes, turn off your computer and then turn it back on and reENTER the loader and *check* your typing of the program. Once corrections have been made, complete step 2 *before* running the program.

8. Delete all remaining program lines. (Do not turn off your computer.)

9. ENTER Program 5-5 (ENTER "D:SKI.LST or ENTER "C:").

10. RUN the program. If it runs correctly, move on to step 11. If it does not run correctly, or the system crashes, check your typing. You may have to reENTER the program from disk or tape. Make corrections and go back to step 5.

11. Once you are certain the game RUNs correctly, SAVE Ski! (SAVE "D:SKI" OR CSAVE).

12. The next time you wish to use this game, just load the SAVEd version of Ski!.

Program 5-4. Loader for Ski!

Ski! will not work on an XL-model unless an Atari BASIC cartridge is plugged into the slot.

```
Ø REM LOADER FOR 'SKI' BASIC RAM
1Ø ? "JUST A MOMENT":DIM A$(746):A=1
   :B=Ø:C=2Ø:FOR D=Ø TO 36:GOSUB 7Ø:
   NEXT D:C=6:GOSUB 7Ø
2Ø IF B<>73882 THEN ? "CHECK ALL DAT
   A LINES":END
3Ø VNTD=PEEK(132)+256*PEEK(133)
4Ø A=USR(ADR(A$),746)
5Ø A=USR(ADR(A$)+22,VNTD+1,ADR(A$),7
   46)
6Ø GOTO 1ØØØ
7Ø E=Ø:FOR F=1 TO C:READ G:E=E+G:B=B
```

```
    +G:A$(A,A)=CHR$(G):A=A+1:NEXT F
80 READ F:IF F<>E THEN ? "CHECK DATA
     STATEMENTS AT LINE ";100+D*10:END
90 RETURN
100 DATA 104,104,170,104,168,138,162
    ,134,76,129,168,104,104,170,104,
    168,138,162,134,76,2617
110 DATA 253,168,104,162,3,104,149,1
    53,202,16,250,56,165,155,229,153
    ,165,156,229,154,3026
120 DATA 104,170,144,16,24,101,154,1
    33,154,138,101,156,133,156,232,1
    04,168,76,227,168,2659
130 DATA 232,104,168,101,153,133,153
    ,176,2,198,154,152,24,101,155,13
    3,155,176,2,198,2670
140 DATA 156,152,73,255,168,200,76,7
    6,169,104,104,104,160,4,200,177,
    138,201,60,208,2785
150 DATA 249,200,200,200,177,138,32,
    40,172,160,7,104,145,157,136,192
    ,2,208,248,56,2823
160 DATA 170,104,229,140,145,157,200
    ,138,229,141,145,157,96,112,112,
    112,70,155,34,102,2748
170 DATA 20,144,38,38,38,38,38,38,38
    ,38,38,38,38,38,38,38,38,38,38,3
    8,848
180 DATA 38,38,6,65,130,9,0,0,0,21,0
    ,0,0,0,0,0,0,0,0,0,307
190 DATA 0,0,6,14,28,24,32,0,128,0,0
    ,0,0,0,0,0,0,0,0,0,232
200 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ,0,0,0,0,0,0,0
210 DATA 0,192,192,220,20,28,7,5,7,0
    ,0,24,52,44,60,24,0,16,56,56,100
    3
220 DATA 124,124,254,16,16,8,28,62,6
    2,62,8,8,0,0,56,94,106,94,116,56
    ,1294
230 DATA 0,0,119,69,117,21,119,0,0,8
    ,24,56,120,8,8,8,8,0,0,0,685
```

```
240 DATA 48,88,56,16,186,254,89,24,1
    56,82,33,16,8,0,0,0,12,26,28,8,1
    130
250 DATA 93,127,154,24,57,74,132,8,1
    6,0,0,24,60,60,24,24,60,186,89,2
    4,1236
260 DATA 154,170,198,65,65,1,18,36,7
    4,161,18,156,77,10,24,24,0,0,0,0
    ,1251
270 DATA 0,169,0,133,0,169,1,141,99,
    6,169,8,141,98,6,104,104,133,7,1
    04,1592
280 DATA 133,6,104,104,133,1,162,10,
    160,117,169,7,32,92,228,96,216,1
    73,4,208,2155
290 DATA 240,4,169,0,133,0,165,0,240
    ,85,165,1,240,81,206,99,6,173,99
    ,6,2112
300 DATA 208,73,198,203,208,26,169,2
    ,133,203,173,124,2,208,3,206,100
    ,6,173,125,2543
310 DATA 2,208,3,238,100,6,173,100,6
    ,141,0,208,165,0,141,99,6,206,98
    ,6,1906
320 DATA 174,98,6,142,5,212,208,27,1
    60,0,56,177,6,233,20,145,6,160,1
    ,177,2013
330 DATA 6,233,0,145,6,169,7,141,98,
    6,141,5,212,198,1,76,98,228,0,0,
    1770
340 DATA 0,0,104,162,228,160,98,169,
    7,32,92,228,96,173,10,210,41,3,2
    01,3,2017
350 DATA 176,247,96,170,169,72,224,1
    ,240,6,169,73,144,2,169,0,145,20
    3,96,165,2567
360 DATA 206,133,205,173,10,210,41,7
    ,24,105,6,168,169,134,145,203,17
    3,10,210,48,2380
370 DATA 14,169,23,133,207,169,18,13
    3,208,169,22,133,209,208,12,169,
    17,133,207,169,2522
380 DATA 22,133,208,169,18,133,209,5
    6,152,229,209,168,162,3,152,24,1
    01,207,168,169,2692
390 DATA 204,145,203,152,24,101,208,
    168,169,204,145,203,202,208,235,
    96,104,104,133,204,3212
400 DATA 104,133,203,169,0,133,205,1
    69,24,133,206,32,113,6,168,32,11
    3,6,32,123,2104
410 DATA 6,136,16,247,160,17,32,113,
    6,201,1,240,4,200,144,1,200,32,1
    13,6,1875
420 DATA 32,123,6,200,192,20,208,245
    ,173,10,210,201,13,176,18,24,165
    ,205,105,10,2336
430 DATA 197,206,176,9,32,139,6,240,
    52,208,2,208,194,173,10,210,201,
    25,176,4,2468
440 DATA 169,7,208,20,173,10,210,201
    ,25,176,4,169,10,208,9,173,10,21
    0,201,25,2218
450 DATA 176,19,169,139,170,173,10,2
    10,41,15,201,12,176,247,24,105,3
    ,168,138,145,2341
460 DATA 203,230,206,165,206,201,175
    ,208,1,96,24,169,20,101,203,133,
    203,165,204,105,3018
470 DATA 0,133,204,208,182,0,727
1000 A=0:FOR B=1 TO 5
1010 GRAPHICS 0:POSITION 2,4
1020 FOR C=1 TO 10:? A*10:A=A+1:NEXT
      C
1030 ? "CONT":POSITION 2,0:POKE 842,
     13:STOP
1040 POKE 842,12
1050 NEXT B
1060 ? "BASIC RAM IS LOADED.  DELETE
      REMAINING LINES AND ENTER LIST
     ING 2."
```

Program 5-5. Ski!

```
100 DIM SCREEN$(1),PM$(1)
101 DIM LEFT$(20),CENTER$(20),RIGHT$
    (20),CURR$(20),CRASH$(20),ERASE$
    (20),DIR(8),SCR(4),DLIST$(1)
102 DIM T$(20),TOPLINE$(20):GOTO 130
110 REM * SKI * LINE 100 MUST BE TYP
    EDIN FIRST!!!
120 HI=INT(A/256):LO=A-HI*256:RETURN

125 POKE 66,1:FOR W=1 TO 10:POKE 532
    79,0:POKE 53279,8:NEXT W:POKE 66
    ,0:RETURN
130 GOSUB 790:REM Initialization rou
    tines
140 REM PLAYER ROUTINE
150 POKE 559,62:POKE 54279,PMBASE
160 POKE 53277,3:POKE 704,2*16+6
170 PO=1024:YP=180:XP=128
180 PM$(PO)=CHR$(0):PM$(PO+254)=CHR$
    (0):PM$(PO+1)=PM$(PO)
195 SCR(0)=0:SCR(1)=10:SCR(2)=4:SCR(
    3)=2:SCR(4)=1
200 ERASE$=CHR$(0):ERASE$(20)=CHR$(0
    ):ERASE$(2)=ERASE$
210 LEFT$=ERASE$:CENTER$=ERASE$:RIGH
    T$=ERASE$:CRASH$=ERASE$
220 FOR I=0 TO 15
230 LEFT$(I+2,I+2)=CHR$(PEEK(CHSET+2
    08+I))
240 CENTER$(I+2,I+2)=CHR$(PEEK(CHSET
    +224+I))
250 RIGHT$(I+2,I+2)=CHR$(PEEK(CHSET+
    104+I))
260 CRASH$(I+2,I+2)=CHR$(PEEK(CHSET+
    240+I))
270 NEXT I
280 DIR(0)=0:DIR(1)=20:DIR(2)=19:DIR
    (3)=21:DIR(4)=1:FOR I=0 TO 3:DIR
    (I+5)=-DIR(I):NEXT I:DIR(5)=-1
290 CURR$=CENTER$
```

```
295 POKE 1636,XP:POKE 2Ø3,2:SCR=Ø
300 PM$(PO+YP,PO+YP+2Ø)=CURR$
310 SCR=SCR+SCR(PEEK(Ø)):POKE 77,Ø
320 POSITION 2,Ø:? #6;SCR;"  ";:POSIT
    ION 15,Ø:IF PEEK(Ø)<>Ø THEN ? #6
    ;(5-PEEK(Ø))*1ØØ
330 IF PEEK(1)<3 THEN POKE Ø,Ø:GOTO
    74Ø
34Ø ST=STICK(Ø)
35Ø LEFT= NOT PTRIG(1):RIGHT= NOT PT
    RIG(Ø):LR=LEFT+2*RIGHT
36Ø CURR$=CENTER$:XP=PEEK(1636)
37Ø IF LEFT THEN CURR$=LEFT$:IF LR<>
    OLR THEN SV=2:TI=5
38Ø IF RIGHT THEN CURR$=RIGHT$:IF LR
    <>OLR THEN SV=4:TI=5
39Ø IF TI>O THEN TI=TI-1:SOUND Ø,SV,
    Ø,TI
4ØØ IF LR=Ø THEN SOUND Ø,Ø,Ø,Ø:TI=Ø
41Ø OLR=L
42Ø UP=(ST=14 OR ST=1Ø OR ST=6):DOWN
    =(ST=5 OR ST=9 OR ST=13)
43Ø YP=YP-2*UP+2*DOWN:IF YP>2ØØ THEN
     YP=2ØØ
44Ø IF YP<4Ø THEN YP=4Ø
45Ø POKE Ø,1+(YP>13Ø)+(YP>16Ø)+(YP>1
    85)
46Ø IF PEEK(PØPF)=Ø THEN 3ØØ
47Ø WHICH=INT(LOG(PEEK(PØPF))/LOG(2)
    +Ø.1):TEMP=PEEK(Ø):POKE Ø,Ø
48Ø PM$(PO+YP,PO+YP+2Ø)=ERASE$
49Ø POKE HITCLR,1:IF WHICH<>2 THEN 6
    2Ø
5ØØ REM POINTS
51Ø PTR=ASC(DLIST$(8))+256*ASC(DLIST
    $(9))
52Ø LINE=INT((YP-39)/8)+1
53Ø COL=INT((XP-49)/8)+1
54Ø LOC=PTR+LINE*2Ø+COL:SOUND Ø,Ø,Ø,
    Ø
55Ø FOR I=Ø TO 8:P=PEEK(LOC+DIR(I))
56Ø IF P<128 OR P>192 THEN 59Ø
```

```
570 POKE LOC+DIR(I),0
580 SCR=SCR+(P=139)*5Ø+(P=134)*1ØØ*(
    5-TEMP):I=11:NEXT I:GOTO 6ØØ
59Ø NEXT I:GOTO 61Ø
6ØØ FOR W=15 TO Ø STEP -1:SOUND Ø,2Ø
    ,1Ø,W:NEXT W
61Ø POKE Ø,TEMP:POKE HITCLR,1:GOTO 3
    ØØ
62Ø REM  CRASH!
63Ø SOUND Ø,Ø,Ø,Ø
64Ø PM$(PO+YP,PO+YP+2Ø)=CRASH$
65Ø FOR W=1ØØ TO 15Ø STEP 2:SOUND Ø,
    W,12,1Ø:NEXT W
66Ø PM$(PO+YP,PO+YP+2Ø)=ERASE$
67Ø YP=2ØØ
68Ø PM$(PO+YP,PO+YP+2Ø)=CURR$
69Ø POKE O,1:SOUND Ø,Ø,Ø,Ø
7ØØ XP=INT(72+9Ø*RND(Ø)):POKE 53248,
    XP:POKE 1636,XP
71Ø IF PEEK(PØPF)<>Ø THEN POKE HITCL
    R,1:GOTO 7ØØ
72Ø POKE HITCLR,Ø:SCR=SCR-5Ø:IF SCR<
    Ø THEN SCR=Ø
73Ø GOTO 3ØØ
74Ø IF SCR>HSCR THEN HSCR=SCR
745 POSITION 8,Ø:? #6;"  HIGH ";HSCR
75Ø SOUND Ø,Ø,Ø,Ø
76Ø SCREEN$(326,336)="press{,}start"
77Ø IF STRIG(Ø) THEN 77Ø
78Ø GOTO 13Ø
79Ø REM INITIALIZATION
8ØØ GRAPHICS 17:HILO=12Ø:POKE 53248,
    Ø:POKE Ø,Ø
81Ø SETCOLOR 4,Ø,12:SETCOLOR 1,12,8:
    SETCOLOR 2,9,6:SETCOLOR Ø,15,4
82Ø PØPF=53252:HITCLR=53278:POKE HIT
    CLR,Ø
83Ø SCRBASE=PEEK(1Ø6)-16:REM 4K BOUN
    DARY
84Ø PMBASE=SCRBASE-8:REM 2K BOUNDARY
    ,DOUBLE-LINE RES
85Ø CHBASE=PMBASE:REM FILL UP OFFSET
```

```
     WITH CHARACTERS
870 VNTD=PEEK(132)+256*PEEK(133)
880 A=USR(VNTD+90,ADR(SCREEN$),4097,
    4097,SCRBASE*256)
890 A=USR(VNTD+90,ADR(PM$),2049,2049
    ,PMBASE*256)
900 A=USR(VNTD+90,ADR(DLIST$),40,40,
    VNTD+134)
910 CHSET=CHBASE*256
920 A=USR(VNTD+23,CHSET,VNTD+174,120
    ):A=USR(VNTD+23,CHSET+128,57472,
    344):A=USR(VNTD+23,CHSET+208,VNT
    D+294,48)
930 A=VNTD+377:GOSUB 120:POKE VNTD+3
    68,HI:POKE VNTD+370,LO
940 A=USR(VNTD+23,1649,VNTD+494,103)
950 Z=USR(VNTD+483):REM DISABLE VBLA
    NK
960 POKE 756,CHBASE:RESTORE 990
980 A=ADR(DLIST$):GOSUB HILO:POKE 56
    1,HI:POKE 560,LO
1020 DLIST$(32)=CHR$(PEEK(560)):DLIS
     T$(33)=CHR$(PEEK(561))
1030 SCREEN$(1)=CHR$(0):SCREEN$(4095
     )=CHR$(0):SCREEN$(2)=SCREEN$
1040 TOPLINE$=SCREEN$
1050 A=ADR(TOPLINE$):GOSUB HILO
1060 DLIST$(5,5)=CHR$(LO):DLIST$(6,6
     )=CHR$(HI)
1070 POKE 88,LO:POKE 89,HI
1080 POSITION 8,0:? #6;"SKI!";
1082 SCREEN$(121,139)="press{,}butto
     n{,}to{,}ski"
1083 SCREEN$(163,178)="pull{█}joysti
     ck{█}to":SCREEN$(185,195)="view
     {,}course"

1085 SCREEN$(403,419)="G{2 █}A{2 █}n
     {2 █}A{2 █}S{2 █}r"
1090 A=USR(VNTD+597,ADR(SCREEN$)+480
     )
1410 A=SCRBASE*256
```

```
1420 A=A-20*(STICK(0)=14)+20*(STICK(
     0)=13)
1430 IF A>SCRBASE*256+3480 THEN A=A-
     20
1440 IF A<SCRBASE*256 THEN A=A+20
1450 GOSUB HILO:T$=CHR$(LO):T$(2)=CH
     R$(HI):DLIST$(8,9)=T$
1460 IF STRIG(0)=1 THEN 1420
1470 A=SCRBASE*256+3480:GOSUB HILO
1480 T$=CHR$(LO):T$(2)=CHR$(HI):DLIS
     T$(8,9)=T$
1481 GOSUB 125:IF STRIG(0) THEN 1481
1490 A=USR(VNTD+342,ADR(DLIST$(8)),1
     76)
1495 SCREEN$(121,195)=SCREEN$(120)
1500 RETURN
2000 VNTD=PEEK(132)+256*PEEK(133)
2010 POKE VNTD+342+26,78
2020 POKE VNTD+342+28,67
2030 POKE VNTD+342+39,85
2040 POKE VNTD+342+43,81
2050 POKE VNTD+342+51,73
2060 A=USR(VNTD+23,20000,VNTD+342,36
     5)
2070 A=USR(VNTD+23,20000+52+30,20000
     +52,365)
2075 A=USR(VNTD+23,20000+36+9,20000+
     36,400)
2080 RESTORE 2000
2085 FOR A=0 TO 8:READ B:POKE 20036+
     A,B:NEXT A
2086 DATA 173,4,208,240,4,169,0,133,
     0
2090 FOR A=0 TO 29:READ B:POKE 20000
     +61+A,B:NEXT A
2095 DATA 198,203,208,26,169,2,133,2
     03
2100 DATA 173,124,2,208,3,206,100,6
2110 DATA 173,125,2,208,3,238,100,6
2112 DATA 173,100,6,141,0,208
2200 FOR A=0 TO 103:? A,PEEK(VNTD+36
     0+A),PEEK(20000+A):NEXT A
```

Thunderbird

Dave Sanders
Translated for the Atari by Charles Brannon

"Thunderbird" offers a challenge for the experienced game player as well as the novice.

"Thunderbird" will demand your undivided attention and 16K of memory. The object of Thunderbird is to score as high as possible. Using a joystick you move the bird left and right, using it to bounce a ball into a wall of bricks. The object of the game is to clear out all the bricks, without letting the ball escape past you. A 1,000 point bonus is awarded when you break out the bottom of the wall (a "breakthrough"); and if you're really good, you get 10,000 points for clearing out all the bricks (no mean feat).

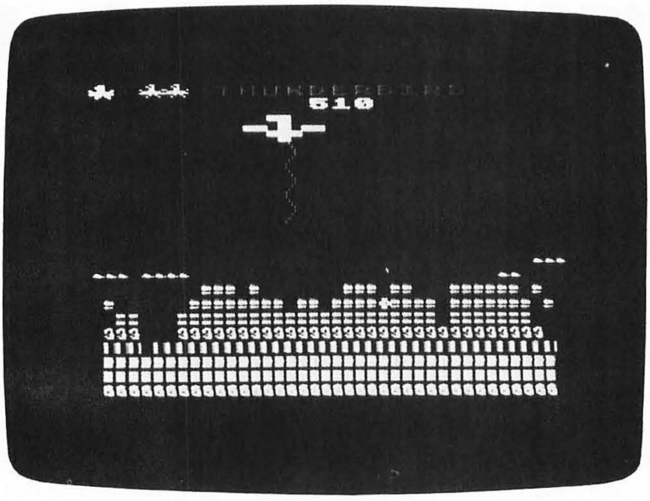
Shades of Zeus

The Thunderbird can unleash the most awesome power of nature — lightning — at the touch of a button (the fire button). Thunderbird will "beam down" several luminous "tiles" that serve to deflect the ball downward when hit. You can lay down tiles like a cap over a hole the ball has created, to force it to widen the hole. Every time the ball hits a tile, it swoops downward, but 25 points are subtracted from your score. That should discourage overuse of this miraculous feature.

Vertical Blank and IRG 4

Here's a bit of information about the programming. The playing field is a mixed-mode display consisting of two rows of GRAPHICS 1 text and 21 rows of a multicolored character mode, IRG 4. This lets us have multicolored bricks.

Player/missile graphics are used to represent the bird, which can be any of three sizes, depending on the skill level. The bird is moved left and right by a small machine language routine that is executed every 1/60 second during the TV's vertical blank (when the electron beam is traveling from the lower right-hand corner to the upper left-hand corner of the screen).



Unleashing a lightning bolt in "Thunderbird."

IRG mode 4, the multicolor mode, is quite interesting. A single character can be any of three colors. To design these colored characters, divide the character horizontally into four two-bit zones. Each two-bit block controls one pixel of color within the character (a multicolor character's resolution is 4x8). No color would be 00, color one is 01, two 10, and three 11 (simple two-bit binary). For example, one of the bricks consists of several colored bands:

```
1110
2220
3330
1110
2220
3330
1110
0000
```

The numbers correspond to a "COLOR" statement. One side and the bottom row are left blank, so the blocks won't touch. The pattern, when expanded into binary, would look like:

```
01010100
10101000
```

```
11111100
01010100
10101000
11111100
01010100
00000000
```

Such a "custom character" would look strange on a normal screen (although you would see some semblance of multicolors, due to artifacting). But when displayed on either an IRG 4 or IRG 5 mode screen, each character is like a tiny 4x8 block of GRAPHICS 7 pixels. Also, any character printed in inverse (with the Atari logo key) will look different. The COLOR 3 pixels in such a character will be displayed as COLOR 4 (normally available only in GRAPHICS 1 or 2).

To create an IRG 4 screen, you must replace the bytes for GRAPHICS 0 by modifying the display list. Luckily, the resolution of IRG 4 is identical to GRAPHICS 0, 40x24.

```
DL=PEEK(560)+256*PEEK(561)+4
POKE DL-1,4+64
FOR I=2 TO 24:POKE DL+I,4:NEXT I
```

See lines 160-180 of Thunderbird. You can also try out IRG 5, which displays these characters in double height (40x12).

Program 5-6. Thunderbird

```
100 REM  THUNDERBIRD 
110 REM Atari Version
120 GRAPHICS 0:BASE=(PEEK(106)-16)*25
    6:GOSUB 1560:REM remove old playe
    rs from screen
130 DIM A$(40),BALL$(4):POKE 82,0:BAL
    L$="*▩{J}{▉}":BALLS=4
140 CHSET=BASE:IF PEEK(CHSET+9)<>252
    THEN GOSUB 1200:REM If not initia
    lized
150 GRAPHICS 0:POKE 752,1:POKE 559,0:
    REM Turn off cursor, screen
160 DLIST=PEEK(560)+256*PEEK(561)+4:R
    EM location of display list
170 FOR I=3 TO 24:POKE DLIST+I,4:NEXT
```

```
     I:REM Change mode zero lines to
    IRG 4 (mulicolor character)
180 POKE DLIST-1,6+64:POKE DLIST+2,6:
    REM top two lines GRAPHICS 1
190 POKE 756,CHSET/256:REM turn on ch
    aracter set
200 SETCOLOR 0,0,12:SETCOLOR 1,3,6:RE
    M white and red
210 RESTORE 240:REM draw brick area
220 POSITION 0,0:? BALL$(1,BALLS):REM
     display # of balls (birds) left
230 REM Pattern of wall:
240 DATA 1,1,2,3,14,129,130,131
250 SCR=PEEK(88)+256*PEEK(89):REM loc
    ate screen memory
260 REM put bricks on screen
270 FOR I=SCR+520 TO SCR+800 STEP 40:
    READ A:FOR J=I+1 TO I+38:POKE J,A
    :NEXT J:NEXT I
280 POSITION 5,0:? #6;"THUNDERBIRD"
290 P0=BASE+1024:PADR=P0+48:REM playe
    r zero.
300 POKE 704,28+176*(DIFF=1)+80*(DIFF
    =2):REM Gold, green, or violet
310 POKE 54279,BASE/256:REM single-li
    ne res.
320 POKE 53277,3:POKE 53256,3-2*(DIFF
    =1)-3*(DIFF=2):REM Start P/M DMA,
     select width according to diffic
    ulty
330 RESTORE 370
340 FOR I=0 TO 21:POKE PADR+I,0:NEXT
    I:REM clear out player
350 FOR I=0 TO 7*(3-DIFF) STEP 3-DIFF
    :READ A:FOR J=0 TO 3-DIFF:POKE PA
    DR+I+J,A:NEXT J:NEXT I
360 REM bird pattern
370 DATA 0,24,8,107,28,8,0,0
380 IF PEEK(547)<>6 THEN A=USR(1536):
    REM turn on VBLANK if necessary
390 POKE 559,62:GOSUB 750:REM turn on
     screen (single-line res. P/M), w
    ait for START
400 DY=1:DX=0.5:IF RND(1)>0.5 THEN DX
    =-0.5:REM Set up ball direction
410 BX=INT(40*RND(0)):BY=INT(7*RND(0)
    +3):REM select random starting po
    sition
420 REM  Main Loop 
430 IF STRIG(0)=0 THEN GOSUB 800:REM
    allow "thunder"
440 IF STICK(0)<>15 THEN POKE 77,0
450 TX=BX+DX:TY=BY+DY:REM update ball
460 IF TY<1 THEN GOSUB 600:GOTO 430:R
    EM check for miss
470 IF TY>20 THEN DY=-DY:IF TX>0 AND
    TX<39 THEN GOSUB 920:GOTO 430:REM
     check for breakthrough
480 IF TX<0 THEN TX=0:DX=-DX
485 IF TX>39 THEN TX=39:DX=-DX
490 TPOS=SCR+TX+40*TY:REM check for o
    bstacles
500 IF PEEK(TPOS)=0 THEN POKE TPOS,5:
    POKE SCR+BX+40*BY,0:BX=TX:BY=TY:G
    OTO 430
510 REM Rebound tiles (lasered down)
520 IF PEEK(TPOS)=4 THEN GOSUB 890:SC
    ORE=SCORE-20:DY=ABS(DY):GOTO 560
530 DY=-ABS(DY):IF RND(0)>0.5 THEN DX
    =-DX
540 FOR W=14 TO 0 STEP -2:SOUND 0,W*5
    ,10,W:NEXT W
550 SCORE=SCORE+(BY-11)*5:BLOCKS=BLOC
    KS+1:REM score according to row
560 POKE TPOS,0:POSITION 29-LEN(STR$(
    SCORE))/2,0:? " ";SCORE;" ";
570 IF BLOCKS=304 THEN 1000:REM BREAK
    -OUT!
580 IF SCORE<0 THEN 720
590 GOTO 500
600 REM Hit bird?
610 IF PEEK(53252) THEN DY=-DY:Z=1:GO
    TO 630
620 GOTO 660
```

```
630 FOR W=14 TO 0 STEP -2:SOUND 0,W+1
    0,10,W:NEXT W
640 POKE SCR+BX+40*BY,0:BX=BX+DX:BY=B
    Y+DY
650 POKE 53278,255:RETURN
660 REM Ball out of bounds (past bird
    )
670 POKE SCR+BX+40*BY,0
680 FOR W=100 TO 0 STEP -5:SOUND 0,W,
    12,8:NEXT W:FOR W=W=0 TO 100 STEP
     5:SOUND 0,W,12,8:NEXT W:SOUND 0,
    0,0,0
690 POKE 53278,255
700 BALLS=BALLS-1:POSITION BALLS,0:?
    " ";
710 IF BALLS>0 THEN 400
720 REM  GAME OVER 
730 POSITION 5,0:? " GAmE OVeR "
740 GOSUB 750:RUN
750 IF PEEK(53279)=6 THEN POSITION 20
    ,0:? "{5 SPACES}":RETURN
760 IF PEEK(20)>20 THEN POSITION 20,0
    :? "PRESS"
770 IF PEEK(20)>40 THEN POSITION 20,0
    :? "start":POKE 20,0
780 GOTO 750
790 REM LASER DOWN
800 XPOS=(PEEK(1664)-48)/4+4-2*(DIFF=
    1)-3*(DIFF=2):FLIP=0:REM equate p
    layer pos. to screen pos.
810 FOR I=3 TO 12:WHERE=SCR+XPOS+40*I
820 P=PEEK(WHERE):POKE WHERE,6+FLIP:F
    LIP=1-FLIP:REM zig-zag line
830 SOUND 0,I*10,0,15-I:POKE 710,PEEK
    (53770):REM scintillate color
840 NEXT I
850 FOR I=3 TO 12:POKE SCR+XPOS+40*I,
    0:NEXT I:REM erase lightning
860 WHERE=SCR+12*40+XPOS:SOUND 0,0,0,
    0:POKE WHERE-1,4:POKE WHERE+1,4:P
    OKE WHERE,4:REM lay down tiles
870 SETCOLOR 2,9,4:RETURN
```

```
880 REM sound effect:
890 FOR W=Ø TO 24Ø STEP 3Ø:SOUND Ø,W,
    12,15-W/17:SOUND 1,W+1Ø,1Ø,15-W/1
    7:NEXT W:SOUND Ø,Ø,Ø,Ø:SOUND 1,Ø,
    Ø,Ø
9ØØ RETURN
91Ø REM break-through
92Ø IF DONE THEN RETURN
93Ø FOR I=1 TO 1ØØ:POKE 53274,PEEK(53
    77Ø):SOUND Ø,I,Ø,15-I/1Ø:NEXT I
94Ø SOUND Ø,Ø,Ø,Ø:POSITION 4,Ø:? "bre
    akthrough":POSITION 22,Ø:? "1ØØØ
    point BONUS"
95Ø FOR I=1 TO 1Ø:POSITION 22,Ø:? "1Ø
    ØØ":FOR W=1 TO 2Ø:NEXT W:POSITION
     22,Ø:? "{4 SPACES}":FOR W=1 TO 2
    Ø:NEXT W:NEXT I
96Ø POSITION 4,Ø:? " THUNDERBIRD ":PO
    SITION 22,Ø:? "{17 SPACES}"
97Ø FOR I=1 TO 1Ø:FOR J=Ø TO 15 STEP
    5:SOUND Ø,5Ø+1Ø-I,Ø,15-J:NEXT J:S
    CORE=SCORE+1ØØ
98Ø POSITION 29-LEN(STR$(SCORE))/2,Ø:
    ? " ";SCORE;" ";
99Ø NEXT I:DONE=1:RETURN
1ØØØ REM  All bricks cleared
1Ø1Ø GOSUB 11ØØ:REM do "BLAST"
1Ø2Ø FOR I=1 TO 5Ø:FOR J=Ø TO 3:POKE
     7Ø8+J,PEEK(5377Ø):NEXT J:Z=Z*(Z<
     5)+1
1Ø3Ø SOUND Ø,I+Z,1Ø,I/1Ø:SOUND 1,I+Z+
     1Ø,1Ø,I/1Ø:NEXT I
1Ø4Ø SOUND Ø,Ø,Ø,Ø:SOUND 1,Ø,Ø,Ø:GOSU
     B 156Ø
1Ø5Ø GRAPHICS 18:POSITION Ø,6:? #6;"
     {Q}{P}{L}{3 P} point bonus{F}"
1Ø6Ø FOR W=1 TO 1ØØ:SOUND Ø,PEEK(5377
     Ø),Ø,15-W/1Ø:POKE 712,(3-FLIP*2)
     *16+FLIP*4+4:FLIP=1-FLIP:NEXT W
1Ø7Ø SCORE=SCORE+1ØØØØ:SOUND Ø,Ø,Ø,Ø
1Ø8Ø DIFF=DIFF+1:IF DIFF>2 THEN DIFF=
     2
```

```
1090 GOTO 150
1100 POKE 82,5:POSITION 5,10
1110 ? "!!!!  #{6 SPACES}m{4 SPACES}.
     ..  !!!!!"
1120 ? "!{3 SPACES}! #{5 SPACES}m m
     .{3 SPACES}.{3 SPACES}!"
1130 ? "!{3 SPACES}! #{4 SPACES}m
     {3 SPACES}m .{7 SPACES}!"
1140 ? "!!!!  #{4 SPACES}m{3 SPACES}m
       ...{4 SPACES}!"
1150 ? "!{3 SPACES}! #{4 SPACES}mmmmmmm
     {5 SPACES}.{3 SPACES}!"
1160 ? "!{3 SPACES}! #{4 SPACES}m
     {3 SPACES}m .{3 SPACES}.
     {3 SPACES}!"
1170 ? "!!!!  #### m{3 SPACES}m  ...
     {4 SPACES}!"
1180 POKE 82,0:RETURN
1190 END
1200 REM  Initialization stuff
1210 POKE 88,0:POKE 89,BASE/256:? "
     {CLEAR}":GRAPHICS 2+16:REM CLEAR
     S OUT P/M AND CHARACTER MEMORY
1220 POSITION 5,0:? #6;"thunderbird":
     POSITION 6,4:? #6;"patience":POS
     ITION 5,8:? #6;"READING ML"
1230 RESTORE 1260
1240 FOR I=1536 TO 1611:READ A:SOUND
     0,A,10,8:POKE 712,A:POKE I,A:NEX
     T I
1250 A=USR(1536):GOTO 1400
1260 DATA 104,173,34,2,141,74
1270 DATA 6,173,35,2,141,75
1280 DATA 6,169,6,162,6,160
1290 DATA 23,32,92,228,96,24
1300 DATA 173,128,6,141,0,208
1310 DATA 173,124,2,208,6,206
1320 DATA 128,6,206,128,6,173
1330 DATA 125,2,208,6,238,128
1340 DATA 6,238,128,6,173,128
1350 DATA 6,201,1,176,5,169
1360 DATA 200,141,128,6,201,250
```

```
1370 DATA 144,5,169,32,141,128
1380 DATA 6,76,73,6
1390 REM
1400 POSITION 3,8:? #6;"LOADING  CHSE
     T"
1410 FOR I=128 TO 510:POKE CHSET+I,PE
     EK(57344+I):SOUND 0,I/2,12,8:POK
     E 712,I/2:NEXT I
1420 RESTORE 1460
1430 READ A:IF A=-1 THEN SOUND 0,0,0,
     0:SOUND 1,0,0,0:RETURN
1440 FOR J=0 TO 7:READ B:SOUND 0,B,10
     ,8:SOUND 1,B+10,10,8:POKE 712,B:
     POKE CHSET+A*8+J,B:NEXT J
1450 GOTO 1430
1460 DATA 1,0,252,168,84,252,168,252,
     0
1470 DATA 2,0,168,168,252,252,168,168
     ,0
1480 DATA 3,0,216,120,184,228,180,212
     ,0
1490 DATA 4,0,0,0,219,150,0,0,0
1500 DATA 5,0,40,40,169,169,40,40,0
1510 DATA 6,192,192,48,48,12,12,3,3
1520 DATA 7,3,3,12,12,48,48,192,192
1530 DATA 10,24,40,24,153,126,255,20,
     34
1540 DATA 14,0,126,126,126,126,126,12
     6,0
1550 DATA -1
1560 REM KILL P/M GRAPHICS
1570 POKE 53277,0:FOR I=0 TO 3:POKE 5
     3261+I,0:NEXT I
1580 RETURN
```

Part Five

Shoot

John H. Palevich

"Shoot" is a machine language arcade-style game that must be initialized on a 16K or greater Atari with or without DOS, but will run on any Atari, even an Atari with 8K of RAM.

This game must be entered using the "Machine Language Editor: MLX" program found in Appendix C. Please refer to Appendix C before typing in this program.

Loading Shoot

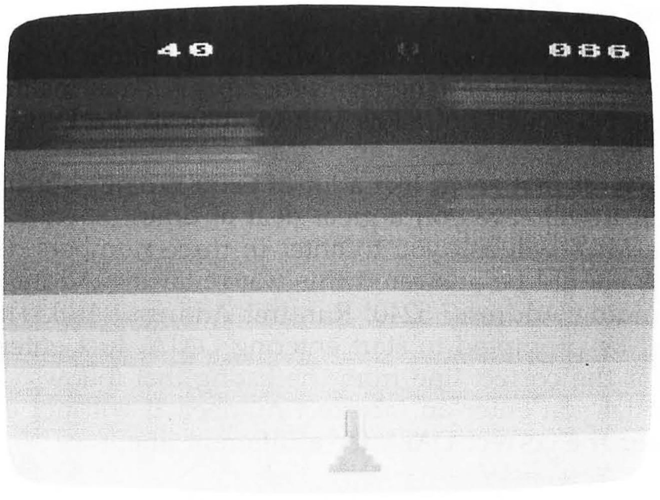
Once you have typed in and SAVEd "Shoot," LOADing the program is simple. With the MLX you have created a boot tape or a boot disk. What is a boot tape or disk? It is the name of a tape or disk that has a machine language program on it, along with information to tell the Atari how to load it into memory and where to jump to begin execution. You can think of a boot tape as a do-it-yourself ROM pack, since you need not have BASIC (or any other cartridge) installed in your Atari at the time you boot up the system.

If you have a boot disk, simply turn on your disk drive. When the busy light goes out, insert the boot disk of Shoot and turn on the Atari. A few seconds later the title screen will appear.

If you have a boot tape, simply place it in the program recorder and rewind to the beginning. Press the Play button on the recorder. Open the lid and remove the BASIC cartridge. Turn off all the peripherals (especially all 815s, 810s and 850s) except for the cassette recorder. Turn off the Atari, press down on the START button, and turn it back on. It should beep once, which is your signal to press the RETURN key and wait. The boot tape will load into the RAM of your Atari. Once there, the cassette will stop and the game will begin.

Playing Shoot

First you will see a copyright message which will last for about 8 to 12 seconds. Then the message will disappear and



You have to be accurate and fast to hit the moving targets in "Shoot."

three zeros will appear. The left (green) one is your score. The middle (red) one is your high score. The right (yellow) one is time remaining. Plug a joystick into controller jack 1 (far left) and press the START button.

Shazam! Eight rows of assorted sizes and colors of airplanes, helicopters, and saucers will start flying hither and yon across the screen. Push the joystick left and right to aim the gun, press the button to fire the missile, then use the joystick to guide the missile into one of the planes. If you miss, try again. If you hit the plane, it will explode and you will score some points: Helicopter — 5 points, Plane — 10 points, Saucer — 25 points. Clearing a rack of planes within 30 seconds gives you a bonus of 50 points. If you take more than 30 seconds to clear a rack of planes, the game will give you another full rack of planes immediately. For every 15 points you score you get an additional second of play time. When the timer goes to zero, your game ends, the high score is adjusted, and the program waits for you to press on the console buttons: press START to restart the game.

Well, that's Shoot in a nutshell. Enjoy the game.

Typing in Shoot

The Machine Language Editor (MLX) was written to help you type in long machine language programs *without* making mistakes. MLX will not allow you to enter in the DATA incorrectly.

It may at first seem like a lot of extra typing, but in the long run it will save you a great deal of time.

The MLX will ask you to enter in three numbers; the prompts should be answered this way: Starting Address? 4096; Ending Address? 5240; Run/Init Address? 4118. Then you will be prompted to start entering DATA. Just enter the DATA for the correct line from the listing that follows. MLX will not let you enter an incorrect number. It will not even let you enter the correct DATA for a different line.

Program 5-7. Shoot Using MLX (see Appendix C)

```
4096:000,009,000,016,008,016,049
4102:024,096,169,060,141,002,242
4108:211,169,022,133,010,169,214
4114:016,133,011,096,076,078,172
4120:018,112,112,112,070,000,192
4126:024,240,112,240,112,240,230
4132:112,240,112,240,112,240,068
4138:112,240,112,240,112,240,074
4144:112,240,112,240,112,065,161
4150:025,016,040,067,041,049,036
4156:057,056,049,032,074,032,104
4162:072,032,080,065,076,069,204
4168:086,073,067,072,128,144,130
4174:130,146,132,148,134,150,150
4180:136,152,200,008,024,040,132
4186:056,072,088,104,120,128,146
4192:026,026,000,001,002,003,154
4198:004,005,006,007,008,124,000
4204:124,001,002,003,002,001,241
4210:255,254,253,254,000,000,106
4216:001,002,003,004,005,006,141
4222:007,008,009,010,011,000,171
4228:001,000,001,000,001,000,135
```

```
4234:001,000,001,001,000,001,142
4240:255,000,000,003,006,012,164
4246:028,060,126,255,000,192,043
4252:096,048,056,060,126,255,029
4258:000,024,024,024,024,060,062
4264:126,255,000,000,248,032,061
4270:242,158,144,240,000,000,190
4276:031,004,079,121,009,015,183
4282:000,000,001,013,063,127,134
4288:024,000,000,000,128,176,008
4294:252,254,024,000,000,024,240
4300:036,126,129,126,000,000,109
4306:001,005,000,008,255,005,228
4312:000,000,002,010,001,024,253
4318:254,010,001,016,003,025,019
4324:000,032,253,025,000,032,058
4330:072,138,072,166,176,232,066
4336:189,076,016,141,010,212,116
4342:141,026,208,166,176,173,112
4348:008,208,041,001,240,019,001
4354:169,000,157,098,016,157,087
4360:109,016,189,120,016,024,226
4366:101,177,133,177,141,030,005
4372:208,232,134,176,189,098,033
4378:016,024,125,109,016,157,217
4384:098,016,141,000,208,189,172
4390:087,016,141,018,208,189,185
4396:131,016,141,008,208,104,140
4402:170,104,064,165,177,208,170
4408:008,169,128,141,003,210,203
4414:076,144,017,056,233,001,077
4420:133,177,169,138,141,003,061
4426:210,162,005,189,000,024,152
4432:024,105,001,009,016,157,136
4438:000,024,201,026,208,009,042
4444:169,016,157,000,024,202,148
4450:076,077,017,165,183,208,056
4456:039,166,181,232,134,181,013
4462:224,015,208,030,162,000,237
4468:134,181,162,005,189,014,033
4474:024,024,105,001,009,144,173
4480:157,014,024,201,154,144,054
```

```
4486:009,169,144,157,014,024,139
4492:202,076,120,017,166,182,135
4498:232,134,182,224,060,208,162
4504:034,162,000,134,182,165,061
4510:183,208,046,162,005,189,183
4516:014,024,056,233,001,009,245
4522:144,157,014,024,201,159,101
4528:208,009,169,153,157,014,118
4534:024,202,076,163,017,169,065
4540:000,162,006,029,013,024,166
4546:202,208,250,041,015,201,087
4552:000,208,004,169,001,133,203
4558:183,169,000,133,077,173,173
4564:120,002,074,074,170,189,073
4570:142,016,133,179,202,138,004
4576:010,010,010,170,160,000,072
4582:189,146,016,153,096,026,088
4588:200,153,096,026,232,200,119
4594:192,016,208,240,165,178,217
4600:024,101,179,133,178,141,236
4606:004,208,165,180,240,038,065
4612:170,169,000,157,128,025,141
4618:202,240,017,165,177,208,251
4624:018,134,180,169,255,157,161
4630:128,025,142,000,210,076,091
4636:042,018,134,180,076,042,008
4642:018,162,000,142,000,210,054
4648:134,180,165,183,208,022,164
4654:173,132,002,208,017,165,231
4660:180,208,013,169,098,133,085
4666:180,165,179,010,010,024,114
4672:105,132,133,178,169,255,012
4678:133,176,141,030,208,076,066
4684:095,228,169,168,141,001,110
4690:210,169,128,141,003,210,175
4696:169,000,141,000,210,169,009
4702:048,141,002,210,162,128,017
4708:169,000,157,255,025,157,095
4714:127,025,202,208,247,169,060
4720:000,162,008,157,255,207,133
4726:202,208,250,169,046,141,110
4732:047,002,169,024,141,007,002
```

```
4738:212,169,003,141,029,208,124
4744:169,016,013,111,002,141,076
4750:111,002,141,027,208,169,032
4756:000,133,180,169,001,133,252
4762:183,169,064,141,014,212,169
4768:169,016,141,049,002,169,194
4774:025,141,048,002,169,016,055
4780:141,001,002,169,234,141,092
4786:000,002,162,017,160,053,060
4792:169,006,032,092,228,169,112
4798:192,141,014,212,169,198,092
4804:141,196,002,169,054,141,131
4810:197,002,169,024,141,198,165
4816:002,169,010,141,199,002,219
4822:162,020,189,055,016,032,176
4828:014,020,009,192,157,255,099
4834:023,202,208,242,165,019,061
4840:024,105,003,197,019,208,020
4846:252,162,020,169,000,157,230
4852:255,023,202,208,250,169,071
4858:016,141,005,024,169,080,173
4864:141,012,024,169,144,141,119
4870:019,024,169,001,133,183,023
4876:169,008,141,031,208,173,230
4882:031,208,201,001,208,006,161
4888:032,042,020,076,078,018,034
4894:201,006,208,239,169,000,085
4900:162,006,157,013,024,157,043
4906:255,023,202,208,247,169,122
4912:145,141,017,024,169,146,178
4918:141,018,024,169,144,141,179
4924:019,024,169,016,141,005,178
4930:024,169,000,133,183,133,196
4936:177,133,182,133,181,162,016
4942:024,160,000,032,197,019,254
4948:200,192,008,208,248,162,078
4954:007,160,208,169,003,141,010
4960:042,002,032,092,228,169,149
4966:192,141,014,212,173,042,108
4972:002,208,003,076,077,019,237
4978:160,008,169,000,025,108,072
4984:016,136,208,250,201,000,163
```

```
4990:208,010,169,050,024,101,176
4996:177,133,177,076,077,019,023
5002:165,183,240,220,165,019,106
5008:024,105,002,197,019,208,187
5014:252,162,000,189,007,024,016
5020:041,031,221,000,024,240,201
5026:005,176,008,076,176,019,110
5032:232,224,006,208,236,076,126
5038:008,019,162,006,189,255,045
5044:023,041,031,009,064,157,249
5050:006,024,202,208,243,076,177
5056:008,019,000,000,000,142,105
5062:194,019,140,195,019,173,170
5068:010,210,041,007,201,006,167
5074:176,247,010,010,170,189,244
5080:210,016,153,109,016,189,141
5086:211,016,153,120,016,189,159
5092:212,016,153,131,016,169,157
5098:000,153,098,016,189,213,135
5104:016,170,172,194,019,169,212
5110:008,141,196,019,189,170,201
5116:016,153,000,026,232,200,111
5122:206,196,019,208,243,152,002
5128:170,172,195,019,096,000,148
5134:140,013,020,168,138,072,053
5140:152,042,042,042,042,041,125
5146:003,170,152,041,159,029,068
5152:246,254,168,104,170,152,102
5158:172,013,020,096,162,032,021
5164:169,012,157,066,003,032,227
5170:086,228,169,020,157,069,011
5176:003,169,117,157,068,003,061
5182:169,003,157,066,003,169,117
5188:008,157,074,003,169,128,095
5194:157,075,003,032,086,228,143
5200:169,000,157,068,003,169,134
5206:016,157,069,003,169,120,108
5212:157,072,003,169,004,157,142
5218:073,003,169,011,157,066,065
5224:003,032,086,228,169,012,122
5230:157,066,003,032,086,228,170
5236:096,067,058,155,000,224,204
```

Appendix A

Writing Your Own Games

Appendix A

Writing Your Own Games: Where to Get More Information

Tom R. Halfhill

By now you've probably typed in, played, and admired some of the games in this book. In time, perhaps some of them will be counted among your favorites. If so, then *COMPUTE!'s First Book of Atari Games* has succeeded. Our main goal was simply to provide more than a dozen fun games for about a third of the cost of one commercial computer game. You can stop right here and have your money's worth.

But this book could be something more. Once you realize that these games were written not by professional programmers, but rather by ordinary hobbyists who probably had never touched a computer in their lives until they bought one, it's not so hard to picture yourself writing games, too.

Unfortunately, too many people dismiss this idea, besieged by self-doubts. "I'll never be able to program like that," they complain. "I don't know anything about computers. And math was my worst subject in school."

Almost always you will hear this kind of statement from adults. Meanwhile, grade school children and teenagers are developing into crack programmers. Two years ago most of them knew nothing about computers either, but they learned. And some of them are flunking math, too. It should be common knowledge by now that mathematics and technical genius have very little to do with computer programming.

Instead, good programmers tend to be people who are creative, have a willingness to learn new things in an exploratory way, and can think logically.

We're not saying that everyone can be a good computer programmer. But many more can than you might suspect. Don't be afraid to see if this includes you.

Appendix A

One reason people are reluctant to attempt game programming — admittedly one of the most difficult types of programming — is their fear of machine language. Virtually all commercial games these days are programmed in machine language because BASIC is just too slow. But the fact is, many good games have been written in BASIC. There are clever ways to get around machine language if you want.

The games in this book are perfect examples. They cover a very wide range of styles and techniques. A few, such as "Blockade," are written entirely in straightforward BASIC — easily within reach of the beginning-to-intermediate home programmer. Blockade has been in my personal program library since I first typed it in from *COMPUTE!* more than two years ago. It's been played as many hours as some commercial games for which I paid $35.

On the other hand, there are games such as "Chiseler" and "Shoot" which are written completely in machine language. They are the work of advanced programmers, and are as fast and as fun as any games on the commercial software market.

Between these two extremes are games such as "Ski!," "Thunderbird," and "Closeout," hybrids of BASIC and machine language. Closeout is a particularly good example, because the author was not a machine language programmer. He used a machine language routine published in *COMPUTE!*, a routine easily used by BASIC programmers who know little or nothing about machine language. If you decide to try your hand at game programming, one of your first goals should be to acquire a "subroutine library" stocked with routines of this type. You'll also need to start collecting magazines and books with important information about your computer.

A couple of years ago the information cupboard for Atari computers was very bare. The machines were new, the idea of home computers was new, and hardware prices were very high. This meant the Atari market was very small. At first it was difficult to find out even the simplest facts about the computers. Some of the Atari's most powerful features — such as player/missile graphics and programmable characters — were not even mentioned in the standard manuals (and still aren't). Home programmers were wandering in the dark.

Today, though, the market abounds with good software,

Appendix A

books, and magazines. If you really want to learn, almost all the information is out there, somewhere. Not all of it is explained as clearly as it could be, but usually it's decipherable. At least, for the most part, it's available.

But now aspiring programmers are faced with a new problem — with *so much* information available, it's hard to choose. Which books are clearly written, and which are just plain confusing? Which material is suitable for beginners, or intermediates, or advanced programmers? Which books should be read first? Which of the additional manuals sold by Atari are really worth buying?

We can't tackle all of these questions here, but we can recommend sources for further reading and experimenting if you want to start programming your own games. This list includes material which covers a range of skills, from beginning to advanced. Naturally, some of these items are from COMPUTE! Publications. But some are sold by our competitors, too. This shouldn't be construed as an endorsement, but rather as a list of reliable sources, a starting point for your own explorations.

Further Reading

*Atari, Inc. *Atari Personal Computer Operating System.* Sunnyvale, CA: Atari, Inc., 1980.

*——. *Atari Personal Computer System Hardware Manual.* Sunnyvale, CA: Atari, Inc., 1980.

Carris, Bill. *Inside Atari BASIC.* Reston, VA: Reston Publishing Co., Inc., 1983.

Chadwick, Ian. *Mapping the Atari.* Greensboro, NC: COMPUTE! Books, 1983.

*Chen, Amy and others. *De Re Atari.* Sunnyvale, CA: Atari, Inc., 1981.

Editors of *COMPUTE!*. *COMPUTE!'s First Book of Atari.* Greensboro, NC: COMPUTE! Books, 1981.

*——. *COMPUTE!'s First Book of Atari Graphics.* Greensboro, NC: COMPUTE! Books, 1982

*——. *COMPUTE!'s Second Book of Atari.* Greensboro, NC: COMPUTE! Books, 1982.

Inman, Don and Kurt Inman. *The Atari Assembler.* Reston, VA: Reston Publishing Co., Inc., 1981.

Appendix A

*Leventhal, Lance A. *6502 Assembly Language Programming.* Berkeley, CA: Osborne/McGraw-Hill, 1979.

Mansfield, Richard. *Machine Language for Beginners.* Greensboro, NC: COMPUTE! Books, 1983.

Moore, Herb, Judy Lower, and Bob Albrecht. *Atari Sound and Graphics, A Self-teaching Guide.* New York: John Wiley & Sons, Inc., 1982.

Poole, Lon, Martin McNiff, and Steven Cook. *Your Atari Computer.* Berkeley, CA: Osborne/McGraw-Hill, 1982.

Sherer, Robin. *Tricky Tutorial #1: Display Lists.* Soquel, CA: Educational Software, Inc., 1981.

*_____. *Tricky Tutorial #2: Horizontal and Vertical Scrolling.* Soquel, CA: Educational Software, Inc., 1981.

Sherer, Robin Alan, Bill Bryner. *Tricky Tutorial #5: Player Missile Graphics.* Soquel, CA: Educational Software, Inc., 1982.

*Wilkinson, Bill. *The Atari BASIC Sourcebook.* Greensboro, NC: COMPUTE! Books, 1983.

*Zaks, Rodnay. *Programming the 6502.* Berkeley, CA: Sybex, Inc., 1980.

* For the more advanced readers

Appendix B

Beginner's Guide to Typing in Programs

Appendix B

A Beginner's Guide to Typing in Programs

What Is a Program?

A computer cannot perform any task by itself. Like a car without gas, a computer has *potential,* but without a program, it isn't going anywhere. Most of the programs published in *COMPUTE!'s First Book of Atari Games* are written in a computer language called BASIC. Atari 8K BASIC is easy to learn.

BASIC Programs

Computers can be picky. Unlike the English language, which is full of ambiguities, BASIC usually has only one "right way" of stating something. Every letter, character, or number is significant. A common mistake is substituting a letter such as "O" for the numeral "0", a lowercase "l" for the numeral "1", or an uppercase "B" for the numeral "8". Also, you must enter all punctuation such as colons and commas just as they appear in the book. Spacing can be important. To be safe, type in the listings *exactly* as they appear.

Braces and Special Characters

The exception to this typing rule is when you see the braces, such as "{DOWN}". Anything within a set of braces is a special character or characters that cannot easily be listed on a printer. When you come across such a special statement, refer to the section of this book entitled "Listing Conventions."

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program. Some DATA statements contain actual programs (called machine language); others contain graphics codes. These lines are expecially sensitive to errors.

If a single number in any one DATA statement is mistyped, your machine could "lock up," or "crash." The keyboard, break key, and RESET keys may all seem "dead," and the screen may go blank. Don't panic — no damage is done. To regain control, you have to turn off your computer, then turn it back on. This will erase whatever program was in memory, so always SAVE a copy of your program before you RUN it. If your computer crashes, you can LOAD the program and look for your mistake.

Sometimes a mistyped DATA statement will cause an error message when the program is RUN. The error message may refer to the program line that READs the data. *This error is still in the DATA statements, though.*

Get to Know Your Machine

You should familiarize yourself with your computer before attempting to type in a program. Learn the statements you use to store and retrieve programs from tape or disk. You'll want to save a copy of your program, so that you won't have to type it in every time you want to use it. Learn to use the machine's editing functions. How do you change a line if you made a mistake? You can always retype the line, but you at least need to know how to backspace. Do you know how to enter inverse video, lowercase, and control characters? It's all explained in your computer's manuals.

A Quick Review

1. Type in the program a line at a time, in order. Press RETURN at the end of each line. Use backspace or the back arrow to correct mistakes.
2. Check the line you've typed against the line in the listing. You can check the entire program again if you get an error when you RUN the program.
3. Make sure you've entered statements in braces as the appropriate control key (see "Listing Conventions" elsewhere in this book).
4. Be sure to SAVE the program on tape or disk *before* RUNning the program.

Appendix C

Using the Machine Language Editor: MLX

Appendix C

Using the Machine Language Editor:MLX

Charles Brannon

Remember the last time you typed in a long machine language program? You typed in hundreds of DATA statements, numbers, and commas. Even then, you couldn't be sure if you'd typed it in right. So you went back, proofread, tried to run the program, crashed, went back and proofread again, corrected a few typing errors, ran again, crashed, rechecked your typing .... Frustrating, wasn't it?

Until now, though, that has been the best way to enter machine language into your machine. Unless you happen to own the Assembler Editor cartridge and are willing to wrangle with machine language on the assembly level, it is much easier to enter a BASIC program that reads the DATA statements and POKEs the numbers into memory.

Some of these "BASIC loaders" will use a *checksum* to see if you've typed the numbers correctly. The simplest checksum is just the sum of all the numbers in the DATA statements. If you make an error, your checksum will not match up. Some programmers have made your task easier by creating checksums every ten lines, so you can zero in on your errors.

There is a problem with BASIC loaders, however. Sometimes a program should reside in low memory, which is where BASIC stores its BASIC programs (including the loader). If the loader was RUN, it would destroy itself as it POKEd the machine language into memory. Sometimes a cassette user will create a program that resides in the same area of memory ($0700-$1EFF, approximately) as the Disk Operating System on disk-based machines.

Appendix C

To get around the low memory problem, some BASIC loaders will directly create the loadable object file (binary file). You can then go to DOS and load the file with menu selection L, or name the file AUTORUN.SYS and have it boot up with the DOS. But this excludes the cassette-based Atari users.

A Thorny Problem

Both of the high-quality machine language programs in this book have this problem. "Shoot" got around it by creating a "boot tape" that cassette users could load as conveniently as CLOAD. Unfortunately, many disk owners don't have a tape drive. "Chiseler" was meant to be entered by assembling it with the Assembler Editor cartridge and run from that environment. For the novice, this is too difficult, and since Chiseler resides in low memory, your average BASIC loader can't cope with it.

MLX was designed to solve these problems. It is a great way to enter all those long machine language programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255. It will prevent you from entering the wrong numbers on the wrong line. In short, MLX will make proofreading obsolete!

Boot Disks

In addition, MLX will generate a ready-to-use boot tape or boot disk. It also has an option to create binary files for DOS users. A boot disk is like those commercial games you buy. You just insert the disk, remove any cartridges, and turn on your computer. The game will then automatically load.

Boot Tapes

Using a boot tape is almost as simple. Just insert it into your player, rewind, press PLAY. Hold down the START key while turning on your computer until you hear a beep (like the one you hear with CLOAD). Then press a key on the keyboard, and the program will automatically load and run.

Incidentally, the binary file option is more useful for utilities than games. Binary files are loaded from the DOS menu (selection L) or automatically if the file is named "AUTORUN.SYS". If you can't stand the thought of putting only one game on each disk (as with boot disks), you can place several binary file machine language games on one disk. *This option is not workable with Shoot, since the program would overwrite DOS.*

Using MLX

Type in and SAVE MLX (you'll want to use it in the future). When you're ready to key in the ML program, RUN it. The program will ask you for three addresses: the start address, the ending address, and the run address. These numbers should be: 4096, 5240, 4118, respectively, for Shoot, and 8192, 11136, 10240 for Chiseler. If you get stuck, refer to the screen dumps below.

Shoot

```
Starting Address?4096
  Ending Address?5240
Run/Init Address?4118

         Tape or Disk:█
```

Chiseler

```
Starting Address?8192
  Ending Address?11136
Run/Init Address?10240

         Tape or Disk:█
```

After you enter the addresses, you'll be asked to press either T for boot tape, or D for disk. If you press D, you'll be asked if you want to generate a boot disk (press D) or a binary file (press F).

You'll then get the prompt:

4096: (for Shoot) or
8192: (for Chiseler)

The prompt is the current line you are entering from the listing. Each line is six numbers plus a checksum. If you enter any of the six numbers wrong, or enter the checksum wrong, the Atari will ring the buzzer and prompt you to re-enter the line. If you enter it correctly, a pleasant bell tone will sound and you can enter the next line.

A Special Editor

You are not using the normal Atari editor with MLX. For example, it will only accept numbers as input. If you need to make a correction, press the < DELETE/BACK S > key; the entire number is deleted. You can press it as many times as necessary back to the start of the line. If you enter three-digit numbers as listed, the computer will automatically print the comma and go on to accept the next number. If you enter less than three digits, you can press either the comma, SPACE bar, or RETURN key to advance to the next number. When you get to the checksum value, the Atari will emit a low drone to remind you to be careful. The checksum will automatically appear in inverse video; don't worry, and don't press the logo key to un-reverse it. It's highlighted for emphasis.

When testing MLX, I've found it to be extremely easy to enter long listings. With the audio cues provided, you don't even have to look at the screen if you're a touch-typist. And if you have the Atari CX85 Numerical Keypad, you're really on easy street!

Done at Last!

When you get through typing, assuming you type it all in one session, you can then save the completed and bug-free program to tape or disk. Follow the screen instructions. With a boot disk, the program will offer to format the disk. If you press Y (yes), be sure you have a blank disk in drive one — not your program disk! After the file is written, the program will end, and you can proceed to boot up your tape or disk (you may need to remove the BASIC cartridge). Now if you get any errors while writing, you probably have a bad tape or disk, or the disk was full (binary file), or you've made a typo somewhere in the MLX itself.

Command Control

What if you don't want to enter the whole program in one sitting? MLX lets you enter as much as you want, save the whole schmeer, and then reload the boot tape, boot disk, or binary file when you want to continue. MLX recognizes these few commands:

CTRL-S: Save
CTRL-L: Load
CTRL-N: New Address
CTRL-D: Display

Hold down CTRL while your press the appropriate key. You will jump out of the line you've been typing, so I recommend you do it at a new prompt. Use the Save command to save what you've been working on. MLX will write the boot tape or disk file as if you've finished, but the boot tape or disk won't run, of course, until you finish the typing. Remember what address you stop on. The next time you RUN MLX, answer all the prompts as you did before, then insert the disk or tape. When you get to the entry prompt (4096: or 8192:), press CTRL-L to reload the file into memory. You'll then use the New Address command to resume typing.

New Address and Display

After you press CTRL-N, enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksum won't match up. You can use the Display command to display a section of your typing. After you press CTRL-D, enter two addresses within the line number range of the listing. You can abort the listing by pressing any key.

Sample Display

```
4096:
Display :From?4096
             To?4118

4096:000,009,000,016,008,016
4102:024,096,169,060,141,002
4108:211,169,022,133,010,169
4114:016,133,011,096,076,078

4096:000,009,000,016,008,016,052
```

```
Incorrect
4096:000,009,000,016,008,016,049
4102:
4102:
4102:
New Address?4114

4114:016,133,011,096,█
```

Tricky Stuff

The special commands may seem a little confusing, but as you work with MLX, they will become valuable. For example, what if you forgot where you stopped typing? Use the Display command to scan memory from the beginning to the end of the program. When you see a bunch of zeros, stop the listing (press a key) and continue typing where the zeros start. Chiseler contains many sections of zeros. To avoid typing them, you can use the New Address command to skip over the blocks of zeros. Be careful, though; you don't want to skip over anything you *should* type.

Making Copies of Your Boot Tapes or Disks

You can use the SAVE and LOAD commands to make copies of the completed game. Use the LOAD command to reLOAD the boot tape or disk, then insert a new tape or disk and use the Save command to create a new copy.

Programmers will find MLX interesting. It contains many useful input/output subroutines such as high-speed save/recall of a huge string (BUFFER$), a sector input/output subroutine, and a sector control routine. Be careful, though; you could accidentally wipe out a disk with the sector routines if you don't use them correctly.

I hope you will find MLX to be a true labor-saving program. Since it has been tested by entering actual programs, you can count on it as an aid for generating bug-free machine language. Be sure to save MLX; it will be used for future applications in both *COMPUTE!* Magazine and COMPUTE! Books.

Program C-1. Machine Language Editor: MLX

```
100 GRAPHICS 0:DL=PEEK(560)+256*PEEK
    (561)+4:POKE DL-1,71:POKE DL+2,6
```

Appendix C

```
110 POSITION 8,Ø:? "MLX":POSITION 23
    ,Ø:? "failsafe entry":POKE 71Ø,Ø:?
120 ? "Starting Address";:INPUT BEG:
    ? "   Ending Address";:INPUT FIN:
    ? "Run/Init Address";:INPUT STAR
    TADR
130 DIM A(6),BUFFER$(FIN-BEG+127),T$
    (2Ø),F$(2Ø),CIO$(7),SECTOR$(128)
    ,DSKINV$(6)
140 OPEN #1,4,Ø,"K:":? :? ,"Tape or
    Disk:";
150 BUFFER$=CHR$(Ø):BUFFER$(FIN-BEG+
    3Ø)=BUFFER$:BUFFER$(2)=BUFFER$:S
    ECTOR$=BUFFER$
160 ADDR=BEG:CIO$="hhh":CIO$(4)=CHR$
    (17Ø):CIO$(5)="LV":CIO$(7)=CHR$(
    228)
170 GET #1,MEDIA:IF MEDIA<>84 AND ME
    DIA<>68 THEN 17Ø
180 ? CHR$(MEDIA):? :IF MEDIA<>ASC("
    T") THEN BUFFER$="":GOTO 25Ø
190 BEG=BEG-24:BUFFER$=CHR$(Ø):BUFFE
    R$(2)=CHR$((FIN-BEG+127)/128)
200 H=INT(BEG/256):L=BEG-H*256:BUFFE
    R$(3)=CHR$(L):BUFFER$(4)=CHR$(H)
210 PINIT=BEG+8:H=INT(PINIT/256):L=P
    INIT-H*256:BUFFER$(5)=CHR$(L):BU
    FFER$(6)=CHR$(H)
220 FOR I=7 TO 24:READ A:BUFFER$(I)=
    CHR$(A):NEXT I:DATA 24,96,169,6Ø
    ,141,2,211,169,Ø,133,1Ø,169,Ø,13
    3,11,76,Ø,Ø
230 H=INT(STARTADR/256):L=STARTADR-H
    *256:BUFFER$(15)=CHR$(L):BUFFER$
    (19)=CHR$(H)
240 BUFFER$(23)=CHR$(L):BUFFER$(24)=
    CHR$(H)
250 IF MEDIA<>ASC("D") THEN 36Ø
260 ? :? "Boot Disk or Binary File:";
270 GET #1,DTYPE:IF DTYPE<>68 AND DT
    YPE<>7Ø THEN 27Ø
280 ? CHR$(DTYPE):IF DTYPE=7Ø THEN 36Ø
```

```
290 BEG=BEG-30:BUFFER$=CHR$(0):BUFFE
    R$(2)=CHR$((FIN-BEG+127)/128)
300 H=INT(BEG/256):L=BEG-H*256:BUFFE
    R$(3)=CHR$(L):BUFFER$(4)=CHR$(H)
310 PINIT=STARTADR:H=INT(PINIT/256):
    L=PINIT-H*256:BUFFER$(5)=CHR$(L)
    :BUFFER$(6)=CHR$(H)
320 RESTORE 330:FOR I=7 TO 30:READ A
    :BUFFER$(I)=CHR$(A):NEXT I
330 DATA 169,0,141,231,2,133,14,169,
    0,141,232,2,133,15,169,0,133,10,
    169,0,133,11,24,96
340 H=INT(BEG/256):L=BEG-H*256:BUFFE
    R$(8)=CHR$(L):BUFFER$(15)=CHR$(H
    )
350 H=INT(STARTADR/256):L=STARTADR-H
    *256:BUFFER$(22)=CHR$(L):BUFFER$
    (26)=CHR$(H)
360 GRAPHICS 0:POKE 712,10:POKE 710,
    10:POKE 709,2
370 ? ADDR;":";:FOR J=1 TO 6
380 GOSUB 570:IF N=-1 THEN J=J-1:GOT
    O 380
390 IF N=-19 THEN 720
400 IF N=-12 THEN LET READ=1:GOTO 72
    0
410 TRAP 410:IF N=-14 THEN ? :? "New
     Address";:INPUT ADDR:? :GOTO 37
    0
420 TRAP 32767:IF N<>-4 THEN 480
430 TRAP 430:? :? "Display:From";:IN
    PUT F:? ,"To";:INPUT T:TRAP 3276
    7
440 IF F<BEG OR F>FIN OR T<BEG OR T>
    FIN OR T<F THEN ? CHR$(253);"At
    least ";BEG;", Not More Than ";F
    IN:GOTO 430
450 FOR I=F TO T STEP 6:? :? I;":";:
    FOR K=0 TO 5:N=PEEK(ADR(BUFFER$)
    +I+K-BEG):T$="000":T$(4-LEN(STR$
    (N)))=STR$(N)
460 IF PEEK(764)<255 THEN GET #1,A:P
```

```
     OP :POP :? :GOTO 370
470  ? T$;",";:NEXT K:? CHR$(126);:NE
     XT I:? :? :GOTO 370
480  IF N<0 THEN ? :GOTO 370
490  A(J)=N:NEXT J
500  CKSUM=ADDR-INT(ADDR/256)*256:FOR
      I=1 TO 6:CKSUM=CKSUM+A(I):CKSUM
     =CKSUM-256*(CKSUM>255):NEXT I
510  RF=128:SOUND 0,200,12,8:GOSUB 57
     0:SOUND 0,0,0,0:RF=0:? CHR$(126)
520  IF N<>CKSUM THEN ? :? "Incorrect
     ";CHR$(253);:? :GOTO 370
530  FOR W=15 TO 0 STEP -1:SOUND 0,50
     ,10,W:NEXT W
540  FOR I=1 TO 6:POKE ADR(BUFFER$)+A
     DDR-BEG+I-1,A(I):NEXT I
550  ADDR=ADDR+6:IF ADDR<=FIN THEN 37
     0
560  GOTO 710
570  N=0:Z=0
580  GET #1,A:IF A=155 OR A=44 OR A=3
     2 THEN 670
590  IF A<32 THEN N=-A:RETURN
600  IF A<>126 THEN 630
610  GOSUB 690:IF I=1 AND T=44 THEN N
     =-1:? CHR$(126);:GOTO 690
620  GOTO 570
630  IF A<48 OR A>57 THEN 580
640  ? CHR$(A+RF);:N=N*10+A-48
650  IF N>255 THEN ? CHR$(253);:A=126
     :GOTO 600
660  Z=Z+1:IF Z<3 THEN 580
670  IF Z=0 THEN ? CHR$(253);:GOTO 57
     0
680  ? ",";:RETURN
690  POKE 752,1:FOR I=1 TO 3:? CHR$(3
     0);:GET #6,T:IF T<>44 AND T<>58
     THEN ? CHR$(A);:NEXT I
700  POKE 752,0:? " ";CHR$(126);:RETU
     RN
710  GRAPHICS 0:POKE 710,26:POKE 712,
     26:POKE 709,2
```

```
720 IF MEDIA=ASC("T") THEN 890
730 REM  DISK 
740 IF READ THEN ? :? "Load File":?
750 IF DTYPE<>ASC("F") THEN 1040
760 ? :? "Enter AUTORUN.SYS for auto
    matic use":? :? "Enter filename"
    :INPUT T$
770 F$=T$:IF LEN(T$)>2 THEN IF T$(1,
    2)<>"D:" THEN F$="D:":F$(3)=T$
780 TRAP 870:CLOSE #2:OPEN #2,8-4*RE
    AD,0,F$:? :? "Working..."
790 IF READ THEN FOR I=1 TO 6:GET #2
    ,A:NEXT I:GOTO 820
800 PUT #2,255:PUT #2,255
810 H=INT(BEG/256):L=BEG-H*256:PUT #
    2,L:PUT #2,H:H=INT(FIN/256):L=FI
    N-H*256:PUT #2,L:PUT #2,H
820 GOSUB 970:IF PEEK(195)>1 THEN 87
    0
830 IF STARTADR=0 OR READ THEN 850
840 PUT #2,224:PUT #2,2:PUT #2,225:P
    UT #2,2:H=INT(STARTADR/256):L=ST
    ARTADR-H*256:PUT #2,L:PUT #2,H
850 TRAP 32767:CLOSE #2:? "Finished.
    ":IF READ THEN ? :? :LET READ=0:
    GOTO 360
860 END
870 ? "Error ";PEEK(195);" trying to
     access":? F$:CLOSE #2:? :GOTO 7
    60
880 REM  BOOT TAPE 
890 IF READ THEN ? :? "Read Tape"
900 ? :? :? "Insert, Rewind Tape.":?
     "Press PLAY ";:IF  NOT READ THE
    N ? "& RECORD"
910 ? :? "Press RETURN when ready:";
920 TRAP 960:CLOSE #2:OPEN #2,8-4*RE
    AD,128,"C:":? :? "Working..."
930 GOSUB 970:IF PEEK(195)>1 THEN 960
940 CLOSE #2:TRAP 32767:? "Finished.
    ":? :? :IF READ THEN LET READ=0:
    GOTO 360
```

```
950 END
960 ? :? "Error ";PEEK(195);" when r
    eading/writing boot tape":? :CLO
    SE #2:GOTO 890
970 REM CIO Load/Save File#2 opened
     READ=0 for write, READ=1 for re
    ad
980 X=32:REM File#2,$20
990 ICCOM=834:ICBADR=836:ICBLEN=840:
    ICSTAT=835
1000 H=INT(ADR(BUFFER$)/256):L=ADR(B
     UFFER$)-H*256:POKE ICBADR+X,L:P
     OKE ICBADR+X+1,H
1010 L=FIN-BEG+1:H=INT(L/256):L=L-H*
     256:POKE ICBLEN+X,L:POKE ICBLEN
     +X+1,H
1020 POKE ICCOM+X,11-4*READ:A=USR(AD
     R(CIO$),X)
1030 POKE 195,PEEK(ICSTAT):RETURN
1040 REM  SECTOR I/O 
1050 IF READ THEN 1100
1060 ? :? "Format Disk In Drive 1? (
     Y/N):";
1070 GET #1,A:IF A<>78 AND A<>89 THE
     N 1070
1080 ? CHR$(A):IF A=78 THEN 1100
1090 ? :? "Formatting...":XIO 254,#2
     ,0,0,"D:":? "Format Complete":?

1100 NR=INT((FIN-BEG+127)/128):BUFFE
     R$(FIN-BEG+2)=CHR$(0):IF READ T
     HEN ? "Reading...":GOTO 1120
1110 ? "Writing..."
1120 FOR I=1 TO NR:S=I
1130 IF READ THEN GOSUB 1220:BUFFER$
     (I*128-127)=SECTOR$:GOTO 1160
1140 SECTOR$=BUFFER$(I*128-127)
1150 GOSUB 1220
1160 IF PEEK(DSTATS)<>1 THEN 1200
1170 NEXT I
1180 IF  NOT READ THEN END
1190 ? :? :LET READ=0:GOTO 360
```

```
1200 ? "Error on disk access.":? "Ma
     y need formatting.":GOTO 1040
1210 REM
1220 REM  SECTOR ACCESS SUBROUTINE
1230 REM Drive ONE
1240 REM Pass buffer in SECTOR$
1250 REM sector # in variable S
1260 REM READ=1 for read,
1270 REM READ=0 for write
1280 BASE=3*256
1290 DUNIT=BASE+1:DCOMND=BASE+2:DSTA
     TS=BASE+3
1300 DBUFLO=BASE+4:DBUFHI=BASE+5
1310 DBYTLO=BASE+8:DBYTHI=BASE+9
1320 DAUX1=BASE+10:DAUX2=BASE+11
1330 REM DIM DSKINV$(4)
1340 DSKINV$="hLS":DSKINV$(4)=CHR$(2
     28)
1350 POKE DUNIT,1:A=ADR(SECTOR$):H=I
     NT(A/256):L=A-256*H
1360 POKE DBUFHI,H
1370 POKE DBUFLO,L
1380 POKE DCOMND,87-5*READ
1390 POKE DAUX2,INT(S/256):POKE DAUX
     1,S-PEEK(DAUX2)*256
1400 A=USR(ADR(DSKINV$))
1410 RETURN
```

Listing Conventions

In order to make special characters, inverse video, and cursor characters easy to type in, *COMPUTE!* Magazine's Atari listing conventions are used in all the program listings in this book.

Please refer to the following tables and explanations if you come across an unusual symbol in a program listing.

Atari Conventions

Characters in inverse video will appear like: INVERSE VIDEO
Enter these characters with the Atari logo key, {⅄}.

When you see	Type	See	
{CLEAR}	ESC SHIFT <	↖	Clear Screen
{UP}	ESC CTRL -	↑	Cursor Up
{DOWN}	ESC CTRL =	↓	Cursor Down
{LEFT}	ESC CTRL +	←	Cursor Left
{RIGHT}	ESC CTRL *	→	Cursor Right
{BACK S}	ESC DELETE	◀	Backspace
{DELETE}	ESC CTRL DELETE	◧	Delete Character
{INSERT}	ESC CTRL INSERT	◨	Insert Character
{DEL LINE}	ESC SHIFT DELETE	⬒	Delete Line
{INS LINE}	ESC SHIFT INSERT	⬓	Insert Line
{TAB}	ESC TAB	▶	TAB key
{CLR TAB}	ESC CTRL TAB	◧	Clear TAB
{SET TAB}	ESC SHIFT TAB	◨	Set TAB stop
{BELL}	ESC CTRL 2	◳	Ring Buzzer
{ESC}	ESC ESC	↰	ESCape key

Graphics characters, such as CTRL-T, the ball character ● will appear as the "normal" letter enclosed in braces, e.g., {T}.

A series of identical control characters, such as 10 spaces, three cursor-lefts, or 20 CTRL-R's, will appear as {10 SPACES}, {3 LEFT}, {20 R}, etc. If the character in braces is in inverse video, that character or characters should be entered with the Atari logo key. For example, {♥} means to enter a reverse-field heart with CTRL-comma, {5▣} means to enter five inverse-video CTRL-U's.

Index

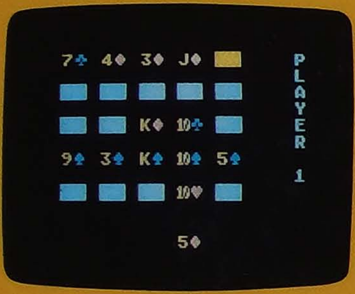
COMPUTE!'s
First Book of Atari Games

Fifteen games for the Atari computer, complete and ready to type in, so no programming knowledge is necessary.

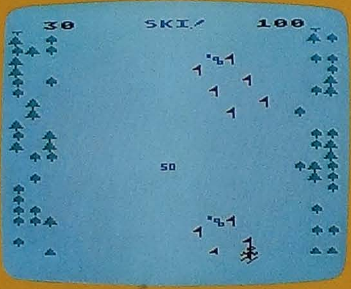
Here are ten of the best games from *COMPUTE!* Magazine, including the fast-action, all-machine-language "Shoot." Several of these games have been updated and improved since their original appearance in the magazine.

There are also five new never-before-published games, including the fast-action, arcade-style "Chiseler."

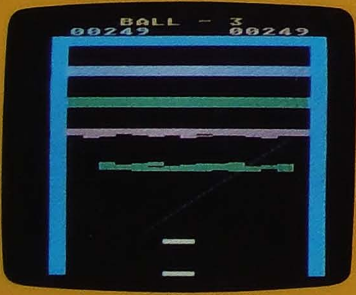
Since you can follow the complete BASIC listing, you can see for yourself how programmers create games. Also, several chapters are devoted to showing you just how to program your own games.



Poker Solitaire



Ski!



Chiseler

Thunderbird

ISBN 0-942386-14-0

$12.95