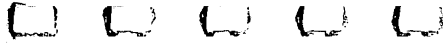
COMPUTE!'s FIRST BOOK OF AMIGA

Powerful games, tutorials, programming techniques, and more for the Commodore Amiga personal computer. Includes previously unpublished programs as well as the best from *COMPUTE!* magazine.

A **COMPUTE! Books** Publication
$16.95



COMPUTE!'s FIRST BOOK OF AMIGA

COMPUTE!™ Publications, Inc.
Part of ABC Consumer Magazines, Inc.
One of the ABC Publishing Companies
Greensboro, North Carolina

The following articles were originally published in *COMPUTE!* magazine, copyright 1986, COMPUTE! Publications, Inc.: "Switchbox" (March); "Requester Windows in Amiga BASIC" (March); "Tug-a-War" (April); "AmigaDOS Batch Files" (April); "Hickory, Dickory, Dock" (May); "Amiga Puzzle" (May); "Meet Ed, the AmigaDOS Editor" (June); "UFO Invasion" (June); "Printers for the Amiga" (June); "Programming in Modula-2" (July); "Hex War" (July); "Tightrope" (August); "Foolproof Input for Amiga BASIC" (August); "Beehive" (September); "Amiga BASIC Style" (September); "Pyramid Power for the Amiga" (October); "Getting Online" (October); "Cutting Telecommunications Costs" (November); "Biker Dave" (November); "Laser Strike" (December).

"Chain Reaction" was originally published in *COMPUTE!* magazine (January), copyright 1987, COMPUTE! Publications, Inc.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-87455-090-4

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is part of ABC Consumer Magazines, Inc., one of the ABC Publishing Companies, and is not associated with any manufacturer of personal computers. Amiga is a trademark of Commodore-Amiga.

Contents

Foreword

The powerful Amiga from Commodore has been supported by COMPUTE! Publications right from the beginning. And COMPUTE! Books continues this support with *COMPUTE!'s First Book of Amiga*, a collection of some of the best programs and articles from *COMPUTE!*, plus some previously unpublished programs and articles.

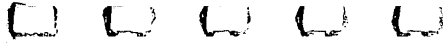
For game players we've included both strategy and fast-action games. "Hex War," "Switchbox," and "Chain Reaction" will challenge your mind. And for quick action there's "UFO Invasion" and "Biker Dave."

Use your computer for education with "Hickory, Dickory, Dock," an entertaining and easy way for preschoolers to learn to tell time (using both traditional and digital clocks). How well do you know your United States–geography facts? Find out playing "Pioneer," a game for one to four players which includes a detailed map of the U.S. as well as mouse control.

COMPUTE!'s First Book of Amiga includes plenty of helpful programming tips, too. See how to create requester windows in an Amiga BASIC program and how to write batch files which can customize the Amiga's startup procedure. Learn to use the Amiga's built-in editor program, ED, and explore programming in Modula-2.

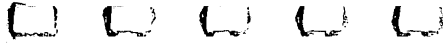
Telecommunications is one of the more popular uses for personal computers. *COMPUTE!'s First Book of Amiga*'s powerful terminal program, "AmigaTerm," operates at both 300 and 1200 bps. AmigaTerm includes the most asked-for features in terminal software, including macros and autodialing.

All the programs in *COMPUTE!'s First Book of Amiga* have been fully tested and are ready to type in and use. If you prefer not to type in the programs, there's a companion disk available for purchase which includes all the BASIC programs from the book ready to load and run. To order the *COMPUTE!'s First Book of Amiga* disk, call toll-free 1-800-346-6767 (in NY 212-887-8525) or use the coupon in the back of this book.



CHAPTER ONE

Recreation



Tug-o-War

Mark Tuttle
Translation by John Krause

Don't be fooled by the apparent simplicity of this two-player strategy game. It looks easy on the surface, but it's a stiff test of your concentration and ability to think ahead. A color monitor or TV and at least 512K RAM required.

Nearly everyone has played tug of war at one time or another. The traditional game pits two players or teams at opposite ends of a rope. At the middle of the rope is a flag, and each side tries to pull the flag into its territory. "Tug-o-War" is based on a similar concept. In this version, the flag is replaced with a round ball shape, and each player tries to maneuver the ball onto his or her side of the screen. Like many two-player games, the difficulty of Tug-o-War depends somewhat on the intelligence of your opponent. But even at the simplest level, you'll find that skill and foresight are essential to success.

Type in and save the program to disk. Before running the program set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. To run the program select *Start* from the *Run* menu.

Battle of the Colors

When you run Tug-o-War, two sets of colored boxes appear, one above the other. The lower, longer series of squares is the playing field. Near the middle of this area is a round ball; the outermost boxes at each end of the playing field represent each player's home position. The players alternate turns, each trying to move the ball in his or her own direction, until it reaches one of the home squares.

So far, so good—but how do you move the ball? It's not done by pulling a rope, but by changing the colors of boxes in the playing field. The color of the square under the ball determines which direction it moves and how far it travels. On any given turn, the ball can move either one or two squares to the

left, or one or two squares to the right. At the top of the screen are four boxes that show you which colors are linked to which directions. For instance, the leftmost box shows you which color makes the ball move one square to the left. The next box to the right shows you which color makes it move *two* squares to the left. The second pair of boxes show you which colors make the ball move in the opposite direction, to the right. By changing the color of the box where the ball is currently located, you can make it move toward your home square.

The playfield contains 11 boxes. When the game begins, each of these boxes is randomly given one of the four colors shown at the top of the screen. On each turn, you may change the color of one, several, or all of the boxes (however, you must always change at least one box). To select a square, simply move the mouse pointer to the desired box and press the left mouse button.

Where do the new colors come from? Every box cycles through the same series of four colors shown in the uppermost set of boxes, going from left to right. For example, if the colors shown there are yellow/purple/red/blue, then a yellow square always changes to purple; a purple square always changes to red; a blue square changes to yellow; and so on. In other words, the box's current color determines which color it gets after the next color change.

Though every turn involves at least one color change, the ball doesn't necessarily move on every turn. It only moves when you change all the boxes between your home position and the current position of the ball. For example, if the ball is three boxes away from your home square, then you must change the color of at least three boxes in order to move it at all.

Foresight Rewarded

As you can see, there's much more to this game than appears on the surface. At first you might be tempted to try to move the ball as often as possible. But that's usually a losing strategy. Remember, the *direction* the ball moves depends on the color of its square before you take the turn.

In many cases, you'll want to move the ball only if it's on a color that moves it toward your goal. But like other games of strategy, Tug-o-War rewards the player who looks beyond the current move and tries to set things up for future moves; sometimes it's wise to make a small, temporary sacrifice in or-

der to benefit later in the game. Because the boxes change colors in the same sequence, the effect of your own move is always completely predictable. However, since a single turn can change the color of many boxes, dramatic changes of fortune are also possible.

Tug-o-War

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
SAY TRANSLATE$("")←
SCREEN 2,320,200,3,1←
WINDOW 2," Tug-o-War ",,12,2←
FOR i=0 TO 7←
READ r,g,b←
PALETTE i,r,g,b←
NEXT←
RANDOMIZE TIMER←
DIM a(11)←
FOR i=1 TO 11←
a(i)=INT(RND(1)*4)+4←
NEXT←
row=3←
col=3:colr=4:GOSUB frame:GOSUB square←
col=4:colr=5:GOSUB frame:GOSUB square←
col=8:colr=6:GOSUB frame:GOSUB square←
col=9:colr=7:GOSUB frame:GOSUB square←
row=10←
LOCATE 5,11:COLOR 1,4:PRINT "2"←
LOCATE 5,14:COLOR 1,5:PRINT "1"←
LOCATE 5,26:COLOR 1,6:PRINT "1"←
LOCATE 5,29:COLOR 1,7:PRINT "2"←
LINE (64,36)-STEP(-16,0),1←
LINE -STEP(8,4),1←
LINE (48,36)-STEP(8,-4),1←
LINE (248,36)-STEP(16,0),1←
LINE -STEP(-8,4),1←
LINE (264,36)-STEP(-8,-4),1←
FOR col=0 TO 12←
GOSUB frame←
NEXT←
col=0:colr=3:GOSUB square←
col=12:colr=2:GOSUB square←
dot=6:GOSUB update←
SAY TRANSLATE$("welcome to tug o war.")←
main:←
LOCATE 17,15←
IF red THEN←
COLOR 2,0:PRINT "Red's turn "←
SAY TRANSLATE$("reds turn.")←
```

```
ELSE←
COLOR 3,Ø:PRINT "Blue's turn"←
SAY TRANSLATE$("blues turn.")←
END IF←
WHILE MOUSE(Ø)<>1 OR MOUSE(4)<8Ø OR MOUSE(4)>1Ø4 OR
MOUSE(3)<23 OR MOUSE(3)>276←
WEND←
click=INT(MOUSE(3)/24)←
IF (red AND click<=dot) OR (red=Ø AND click>=dot) T
HEN ←
temp=dot←
IF a(temp)=4 THEN dot=dot-2←
IF a(temp)=5 THEN dot=dot-1←
IF a(temp)=6 THEN dot=dot+1←
IF a(temp)=7 THEN dot=dot+2←
END IF←
IF red THEN←
FOR i=click TO 11←
a(i)=a(i)+1←
IF a(i)=8 THEN a(i)=4←
NEXT←
ELSE←
FOR i=1 TO click←
a(i)=a(i)+1←
IF a(i)=8 THEN a(i)=4←
NEXT←
END IF←
IF dot>11 THEN←
dot=12:GOSUB update←
LOCATE 17,15:COLOR 2,Ø:PRINT " Red wins! "←
SAY TRANSLATE$("red wins.")←
GOTO quit←
END IF←
IF dot<1 THEN←
dot=Ø:GOSUB update←
LOCATE 17,15:COLOR 3,Ø:PRINT "Blue wins! "←
SAY TRANSLATE$("blue wins.")←
GOTO quit←
END IF←
GOSUB update←
red=1-red←
GOTO main  ←
frame:←
x=24:IF 24*col>28Ø THEN x=23←
LINE (24*col,8*row)-STEP(x,24),1,b←
RETURN←
square:←
x=22:IF 24*col+1>28Ø THEN x=21←
LINE (24*col+1,8*row+1)-STEP(x,22),colr,bf←
RETURN←
update:←
```

```
FOR col=1 TO 11←
colr=a(col):GOSUB square←
NEXT←
CIRCLE (24*dot+11,91),5,1←
PAINT (24*dot+11,91),1←
RETURN  ←
quit:←
LOCATE 19,7:COLOR 1,Ø:PRINT "Click mouse to play ag
ain."←
SAY TRANSLATE$("click mouse to play again.")←
WHILE MOUSE(Ø)=Ø:WEND←
RUN←
DATA .5,.5,.5,Ø,Ø,Ø,1,Ø,Ø,Ø,Ø,1,Ø,1,Ø,1,1,Ø,1,Ø,1,Ø
,1,1←
←
```

UFO Invasion

John Robinson
Translation by Philip I. Nelson

A classic confrontation game, "UFO Invasion" pits you, the lone defender, against a horde of deadly machines from outer space.

You are at the controls of a surface-to-air missile launcher located somewhere near the Arctic Circle. Your radar outposts have detected a fleet of unidentified flying objects (UFOs) invading the North American continent. Forewarned about this possibility, you know that the waves of UFOs will attack your missile site before proceeding to the more populous areas lying southward. Are you up to the challenge?

Though it's not a particularly long program, "UFO Invasion" offers quite a test for your gaming skills, particularly at the higher levels.

Typing It In

Type and save the program to disk. To run the program select *Start* from the *Run* menu.

First Line of Defense

When UFO Invasion begins, you'll hear the sound of an alarm siren and see two warning messages scroll across the screen. The middle of the screen contains your control panel. The observatory window at the top gives you a direct view of the skyline in your defense sector. Within the window is the aiming crosshairs pointer for your missile launcher, and directly below is a radar screen.

When the saucer-shaped UFO appears, your job is to move the crosshairs onto the UFO (using the cursor keys) and launch a missile at it (by pressing the space bar). If your missile hits the UFO, the automated craft is vaporized immediately.

Before it can fire at your base, the UFO must locate your position. Once your position is located, the UFO is certain to

hit the mark. Your force shields are powerful enough to protect you against three hits by the UFO, but the fourth hit neutralizes your defenses and paves the way for a successful invasion (ending the game as well).

Control Panel

The control panel is equipped with six gauges to help you monitor events. On each side of the circular radar screen are two ladder gauges. The gauge at the far right shows you how many UFOs remain to be eliminated in the current level. There are eight levels in all; you must eliminate 29 UFOs at each level before advancing to the next.

The gauge directly to the right of the radar screen indicates how close the UFO is to locating your position. When this indicator reaches the top, the UFO scores a hit.

The gauge at the far left shows your points for the current level. You receive 100 points (shown as one bar on the ladder) for each UFO you destroy, with an additional bar for hitting the UFO before the timer is halfway to the top. If you score two bars for every UFO on the current level, you receive a bonus equal to 1000 points times the level number.

Directly underneath the radar screen are two additional indicators that show you how many levels have been completed, and how many hits your shields have sustained.

Press the cursor keys to move the aiming crosshairs left, right, up, or down. To fire a missile, press the space bar. You can quit the game at any time by pressing Q. In levels 1, 2, 4, 6, and 8 you can view the UFOs through the observatory window. In levels 3, 5, and 7 the sky is obscured by a thick cloud cover, forcing you to guide the missiles by radar alone. The radar screen shows the position of the UFO in relation to your aiming crosshairs. Aim with the cursor keys until the red dot is in the center of the radar panel; then fire.

UFO Invasion

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
' UFO Invasion for Amiga←
←
DEFINT A-Z←
RANDOMIZE TIMER←
SCREEN 1, 320, 200, 2, 1←
WINDOW 1,"UFO Invasion",(0,0)-(311,185),20,1←
```

```
PALETTE Ø,Ø,Ø,Ø←
PALETTE 1,1,1,1←
PALETTE 2,Ø,1,Ø←
PALETTE 3,1,Ø,Ø←
CLS←
DIM SH(2ØØØ),UFO(2ØØ),GD(2ØØ)←
DIM X(3Ø),Y(3Ø),Radar(5Ø)←
L=1: TL=8: Lives=Ø: Score=Ø←
RX=1ØØ/15: RY=5Ø/12←
LINE (111,51)-(211,159),3,B←
LINE (111,1Ø1)-(211,1Ø1),3←
LINE (121,1Ø1)-(121,159),3←
LINE (131,1Ø1)-(131,159),3←
LINE (191,1Ø1)-(191,159),3←
LINE (2Ø1,1Ø1)-(2Ø1,159),3←
FOR Y=157 TO 1Ø3 STEP -2←
LINE (111,Y)-(131,Y),3←
LINE (191,Y)-(211,Y),3←
NEXT←
CIRCLE (161,12Ø),16,1←
CIRCLE (161,12Ø),1Ø,1←
CIRCLE (161,12Ø),4,1←
LINE (146,12Ø)-(176,12Ø),1←
LINE (161,1Ø8)-(161,132),1←
FOR X=145 TO 173 STEP 4←
LINE (X,135)-(X+4,14Ø),3,B←
NEXT←
FOR X=154 TO 162 STEP 4←
LINE (X,147)-(X+4,151),3,B←
NEXT←
LINE (1,Ø)-(6,Ø),1←
LINE (Ø,1)-(7,1),1←
LINE (1,2)-(7,2),3←
LINE (Ø,3)-(7,3),1←
LINE (1,4)-(6,4),1←
GET (Ø,Ø)-(7,6),UFO←
LINE (Ø,Ø)-(7,7),Ø,bf←
LINE (Ø,3)-(4,3),2←
LINE (2,1)-(2,5),2←
GET (Ø,Ø)-(5,5),GD←
LINE (Ø,Ø)-(7,7),Ø,bf←
LINE (Ø,Ø)-(1,1),2,bf←
GET (Ø,Ø)-(1,1),Radar←
PUT (Ø,Ø),Radar←
GOSUB Siren←
B$=" Enemy Alert "←
GOSUB PrintMessage←
GOSUB Siren←
B$=" UFO Invasion "←
GOSUB PrintMessage←
GOTO NewLevel←
```

```
←
←
PrintMessage:←
FOR J=1 TO 39←
LOCATE 1,J←
SOUND 400+(J*10),.1←
PRINT LEFT$(B$,40-J)←
NEXT←
RETURN←
←
←
Siren:←
FOR J=1 TO 10←
FOR P=1000 TO 1900 STEP 55←
SOUND P,.2←
NEXT←
NEXT←
RETURN←
←
←
NewLevel:←
B=160: Pts=0: T=0←
FOR S=158 TO 102 STEP -2←
LINE (202,S)-(210,S),2←
NEXT←
IF L=3 OR L=5 OR L=7 THEN clr=0:GOTO Mip ELSE clr=1
←
FOR S=1 TO 60←
PSET(112+RND*98,52+RND*48),INT(RND*4)←
SOUND 1500+INT(RND(1)*1000),.1←
NEXT←
Mip:←
S=100: Xu=112+RND*90←
Yu=52+RND*40←
IF clr THEN PUT(Xu,Yu),UFO←
xg=160: yg=75←
PUT(xg,yg),GD←
LINE(142+L*4,136)-(144+L*4,139),1,bf←
←
←
MainLoop:←
X$=INKEY$: IF UCASE$(X$)="Q" THEN Quit←
IF X$="" OR X$<CHR$(28) OR X$>CHR$(32) THEN Skip←
ON ASC(X$)-27 GOTO Up, Down, Right, Left, Hit←
GOTO Skip←
Up:←
Cyg=Cyg-5:GOTO Skip←
Down:←
Cyg=Cyg+5:GOTO Skip←
Right:←
```

```
Cxg=Cxg+5:GOTO Skip←
Left:←
Cxg=Cxg-5:GOTO Skip←
Hit:←
Fired=1←
←
←
Skip:←
PUT (xg,yg),GD←
xg=xg+Cxg←
IF xg>200 THEN xg=200 ELSE IF xg<112 THEN xg=112←
yg=yg+Cyg←
IF yg>90 THEN yg=90 ELSE IF yg<52 THEN yg=52←
Cxg=0:Cyg=0←
PUT (xg,yg),GD←
IF RND<.1 THEN Cxu=RND*10-5:Cyu=RND*6-3←
IF clr THEN PUT (Xu,Yu),UFO←
Xu=Xu+Cxu←
IF Xu>200 THEN Xu=200 ELSE IF Xu<112 THEN Xu=112←
Yu=Yu+Cyu←
IF Yu>90 THEN Yu=90 ELSE IF Yu<52 THEN Yu=52←
IF clr THEN PUT (Xu,Yu),UFO←
IF NotFirst THEN PUT(Xr,Yr),Radar←
NotFirst=1←
Xr=161+(Xu-xg)/RX←
Yr=120+(Yu-yg)/RY←
PUT (Xr,Yr),Radar←
IF Fired THEN GOSUB Shoot:IF S=158 THEN AllGone←
T=T+1←
IF T>TL THEN B=B-2:T=0:LINE(192,B)-(200,B),2:IF B=1
02 THEN GOSUB TakeShot←
GOTO MainLoop←
←
←
Quit:←
B$=" Game stopped"←
Score=Score+Pts*100*L←
GOTO GameOver←
←
←
Shoot:←
IF clr THEN PUT (Xu,Yu),UFO←
LINE (160,100)-(xg+3,yg+6),2←
FOR J=50 TO 1000 STEP 200←
SOUND J,.1←
NEXT←
Fired=0←
LINE (160,100)-(xg+3,yg+6),0←
IF xg+3>Xu AND xg+3<Xu+9 AND yg+4>Yu AND yg+3<Yu+6
THEN HitUFO←
IF clr THEN PUT (Xu,Yu),UFO←
```

```
RETURN←
←
←
HitUFO:←
PUT (xg,yg),GD←
FOR e=1 TO 30←
X(e)=Xu+RND*6+1←
Y(e)=Yu+RND*6+1←
PSET (X(e),Y(e)),2←
SOUND 820,.1←
NEXT←
FOR e=1 TO 30←
PRESET (X(e),Y(e))←
NEXT←
IF clr THEN PSET (Xu+RND*6+1,Yu+RND*6+1),INT(RND*4)
←
Xu=112+RND*90←
Yu=52+RND*40←
IF clr THEN PUT(Xu,Yu),UFO←
PUT(xg,yg),GD←
AddScore:←
S=S+2←
LINE(202,S)-(210,S),0←
Pts=Pts+1←
GOSUB Here←
IF B>130 THEN Pts=Pts+1:GOSUB Here←
FOR X=B TO 160 STEP 2←
LINE(192,X)-(200,X),0←
NEXT←
B=160←
RETURN←
←
←
Here:←
Hx=112: Hy=160-Pts*2←
IF Pts>29 THEN Hx=122:Hy=160-(Pts-29)*2←
LINE (Hx,Hy)-(Hx+8,Hy),1←
RETURN←
←
←
AllGone:←
L=L+1: TL=TL-1←
FOR P=102 TO 158 STEP 2←
LINE(112,P)-(120,P),0←
LINE(122,P)-(130,P),0←
NEXT←
LINE(112,52)-(209,99),0,bf←
Score=Score+Pts*(L-1)*100←
IF Pts<58 THEN GOTO There←
FOR J=500 TO 2500 STEP 500←
SOUND J,2←
```

```
NEXT←
B$=" You passed Level "+STR$(L-1)←
GOSUB PrintMessage←
B$=STR$(1000*(L-1))+" Points bonus"←
GOSUB PrintMessage←
FOR J=1 TO 2000←
NEXT←
Score=Score+1000*(L-1)←
IF L=9 THEN PlayerWins←
There:←
IF L>8 THEN PlayerWins←
B$=" Level "+STR$(L)←
GOSUB PrintMessage←
FOR J=1 TO 1500←
NEXT←
GOTO NewLevel←
←
←
TakeShot:←
PUT (xg,yg),GD←
IF clr THEN PUT (Xu,Yu),UFO←
Xb=Xu+4: Yb=Yu+6←
LINE(Xb,Yb)-(112,100),2←
LINE (Xb,Yb)-(210,100),2←
LINE (Xb,Yu)-(112,52),2←
LINE (Xb,Yu)-(210,52),2←
LINE(Xb,Yb)-(112,100),0←
LINE (Xb,Yb)-(210,100),0←
LINE (Xb,Yu)-(112,52),0←
LINE (Xb,Yu)-(210,52),0←
IF clr THEN PUT (Xu,Yu),UFO←
PUT (xg,yg),GD←
PALETTE 0,1,0,0←
FOR K=400 TO 500←
SOUND K,.1←
NEXT←
PALETTE 0,0,0,0←
Pts=Pts-2←
IF Pts<0 THEN Pts=0←
IF Lives=3 THEN UFOgotcha←
FOR X=102 TO 158 STEP 2←
LINE(192,X)-(200,X),0←
NEXT←
B=160←
Lives=Lives+1←
LINE (151+4*Lives,148)-(153+4*Lives,150),2,bf←
RETURN←
←
←
UFOgotcha:←
PALETTE 0,1,0,0←
```

```
FOR J=400 TO 500 STEP 3€
SOUND J,.1€
NEXT€
FOR J=500 TO 400 STEP -3€
SOUND J,.1€
NEXT€
PALETTE 0,0,0,0€
B$=" Game Over"€
GOTO GameOver€
€
€
PlayerWins:€
B$=" You win!"€
GOSUB WinSound€
€
€
GameOver:€
GOSUB PrintMessage€
c$=" Score"+STR$(Score)€
FOR J=1 TO 15€
LOCATE 1,J€
PRINT LEFT$(c$,40-J)€
NEXT€
CleanBuffer:€
IF INKEY$<>"" THEN CleanBuffer€
LOCATE 23,9€
PRINT "Press Y to play again";€
X$=""€
WHILE X$=""€
X$=INKEY$€
WEND€
IF UCASE$(X$)="Y" THEN RUN€
CLS: END€
€
€
WinSound:€
FOR J=1 TO 2€
RESTORE MusicData€
SoundLoop:€
READ X€
IF X<>65535& THEN€
SOUND X,1€
GOTO SoundLoop€
END IF€
NEXT€
SOUND 550, 8€
RETURN€
€
€
MusicData:€
```

```
DATA 550, 500, 450, 400, 350←
DATA 550, 500, 450, 400, 350←
DATA 300, 350, 400, 450, 500←
DATA 65535←
←
```

Tightrope

Daniel Aven
Translation by Patrick Parrish

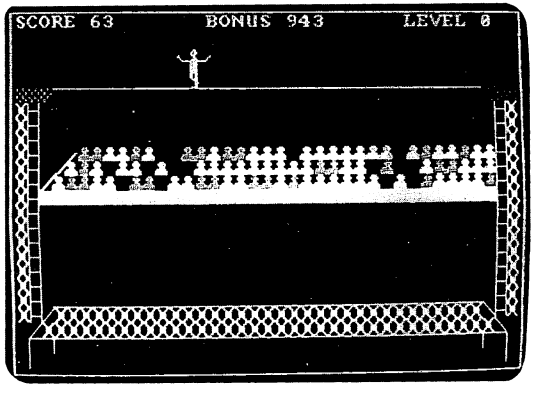
Stretched high above the circus arena, the tightrope beckons. Can you walk all the way across the rope without falling into the net? This interesting program is both an arcade game and a typing tutor. Requires 512K of memory.

Arms outstretched, you venture cautiously onto the tightrope. The rope quivers for a moment, then steadies. Far below, in a packed circus tent, the crowd roars its encouragement. Don't worry; there's a safety net below. But you won't entertain the onlookers—or earn points in this game—by falling into the net. Your first few steps are hesitant, but with practice your progress becomes more sure. After what seems an eternity, you reach the other side. The crowd cheers its approval and cries out for a repeat performance.

"Tightrope" combines a novel game idea and realistic animation with an educational goal. You can play it either as an arcade game or as a typing tutor. In game mode, the object is to help the acrobat walk all the way across the tightrope without falling into the net. In tutorial mode, you must watch for a letter to appear next to the acrobat's head and type it on the keyboard before time runs out.

A Delicate Balance

If you choose the game of skill, your goal is simple: Move the animated acrobat all the way across the tightrope without falling into the net. As the acrobat walks along, he'll occasionally begin to fall to one side or the other. But there's always time to help him recover his balance by pressing the Z key to move him to his left or the slash (/) key to move him to his right. If you countermove just enough to enable the acrobat to regain his balance, all is well, and he begins to walk again. If you move too far in the opposite direction, the teetering starts all over again.



"Tightrope" requires 512K of memory and uses keyboard controls.

It's a delicate balancing act, and it grows more difficult each time the acrobat makes it across the rope. When you succeed in moving him to the opposite side, you advance to the next skill level. At each higher level, it becomes more and more difficult to keep him balanced.

If he loses his balance completely, the acrobat falls to the safety net and bounces a few times before coming to rest. At this point, you can try again at the same level or return to the main menu to choose a different game.

Your score is based on how far the acrobat gets before falling. Each successful step is worth a certain number of points, and this value increases at higher levels. In addition, bonus points are awarded for rapid progress—the faster you move the acrobat across the rope, the higher the bonus.

Typing Tutor

In the tutorial version of Tightrope, the object is the same—move the acrobat across the rope without falling—but different means are used to help him keep his balance.

When you see a character appear next to the acrobat's head, that's your cue to press the corresponding key on the keyboard. If you type the correct letter, the acrobat straightens up. If you press the wrong key, a buzzer sounds, and the acrobat teeters even more.

To remain on the rope, you must continue to type the same letters as those which appear on the screen. In other respects, the tutorial version of Tightrope is the same as the skill game.

Typing It In

Type in and save the program to disk. Before running the program, set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. To run the program, select *Start* from the *Run* menu.

Tightrope

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
Ø GOSUB setup:GOTO 7Ø←
1 PUT (X,Y),w3,PSET:RETURN←
2 PUT (X,Y),w2,PSET:RETURN←
3 PUT (X,Y),w1,PSET:RETURN←
4 PUT (X,Y),11,PSET:RETURN←
5 PUT (X,Y),12,PSET:RETURN←
6 PUT (X,Y),13,PSET:RETURN←
7 PUT (X,Y),r1,PSET:RETURN←
8 PUT (X,Y),r2,PSET:RETURN←
9 PUT (X,Y),r3,PSET:RETURN←
1Ø PUT (X,Y),w4,PSET:RETURN←
2Ø JV=Ø:a$=UCASE$(INKEY$)←
IF a$=CHR$(47) THEN ←
JV=1:AV=Ø:RETURN←
END IF←
IF a$=CHR$(9Ø) THEN←
JV=2:AV=Ø:RETURN ←
END IF←
21 AV=AV+1←
IF AV=b*4 THEN AV=Ø:RETURN←
22 GOTO 2Ø←
23 r1=INT(26*RND(1))+1←
X9=INT(X/8):r2=r1+64←
a=Ø:LOCATE 3,X9:PRINT CHR$(r2)←
24 a$=UCASE$(INKEY$):a=a+1←
IF a=b THEN←
M1=2←
GOSUB 177Ø:GOTO 3Ø←
END IF←
25 IF a$="" THEN 24←
26 IF a$=CHR$(r2) THEN←
M1=1←
GOSUB 174Ø:GOTO 3Ø←
END IF←
27 M1=2:GOSUB 177Ø←
3Ø LOCATE 3,X9:PRINT " "←
RETURN←
31 T3=3*INT(TIMER-T2)←
T=1ØØØ-T3:IF T<Ø THEN T=Ø←
```

```
32 RETURN€
7Ø CLS:FOR X=Ø TO 36 STEP 36€
FOR J=Ø TO 2€
PUT (X*8+J*8,4Ø),s14:NEXT J,X€
LINE (24,4Ø)-(287,4Ø),3€
75 FOR J=Ø TO 38 STEP 38€
FOR I=1 TO 15€
PUT (J*8,I*8+4Ø),s15€
PUT (J*8+8+(J=38)*16,I*8+4Ø),s16€
NEXT I,J€
8Ø FOR r=1 TO 3:FOR c=6-r TO 36€
X=RND(1):ROW=64+8*r:COL=c*8€
IF X<.3 THEN€
PUT(COL,ROW),s17:GOTO 1ØØ€
END IF€
85 IF X>=.3 AND X<.6 THEN€
PUT(COL,ROW),s18:GOTO 1ØØ€
END IF€
9Ø IF X>=.6 AND X<.9 THEN€
PUT(COL,ROW),s19:GOTO 1ØØ€
END IF€
95 LOCATE ROW/8,COL/8:PRINT " "€
1ØØ NEXT c,r€
1Ø5 LINE (39,75)-(17,96)€
LINE (38,75)-(16,96)€
FOR I=96 TO 96+7€
LINE (16,I)-(8*37-1,I):NEXT I€
1Ø7 LINE (23,2Ø*8-1)-(36*8,2Ø*8-1)€
LINE (7,22*8)-(38*8,22*8)€
LINE (23,2Ø*8)-(7,22*8)€
LINE (36*8,2Ø*8-1)-(38*8,22*8)€
1Ø8 FOR r=Ø TO 1€
FOR c=3-r TO 35+r€
PUT (c*8,(r+2Ø)*8),s15€
NEXT c,r€
1Ø9 LINE (7,22*8)-(7,24*8)€
LINE (38*8,22*8)-(38*8,24*8)€
LINE (23,22*8)-(23,23*8)€
LINE (36*8,22*8)-(36*8,23*8)€
16Ø Y=19:PX=7:PZ=1Ø:c=Ø€
44Ø T2=TIMER€
5ØØ IF w=2 AND d=8 THEN€
b=1ØØ€
END IF€
IF w=2 AND d=9 THEN €
b=65€
END IF€
€
51Ø P=Ø:GOSUB 3ØØØ€
FOR X=288 TO 4 STEP -3€
52Ø c=c+d+1:GOSUB 31€
```

```
550 LOCATE 1,1←
PRINT "SCORE"c"     "TAB(16)"BONUS"T" ";←
PRINT TAB(31)"LEVEL"d←
600 f=0:IF X<268 AND X>24 THEN GOSUB 780←
605 IF f=1 THEN 1080←
630 P=P+1:IF P=5 THEN P=1←
640 ON P GOSUB 1,2,3,2←
655 JV=0:a$=UCASE$(INKEY$):IF a$=" " THEN GOSUB 151
0 ←
IF a$=CHR$(47) THEN JV=1:GOSUB 810←
660 IF a$=CHR$(90) THEN JV=2:GOSUB 950←
665 IF f=1 THEN 1080←
670 NEXT X:X=X+3←
GOSUB 3000:GOSUB 10←
710 c=c+T:T=0←
720 IF d<9 THEN b=b-2:d=d+1←
750 IF w=2 AND d<9 THEN b=b-25←
760 IF d=9 THEN PX=6:PZ=9←
770 GOTO 440←
780 r=INT(9*RND(1))+1←
IF r>2 THEN RETURN←
800 IF r=1 THEN 950←
810 P=4:PUT (X,Y),11,PSET←
820 IF w=1 THEN GOSUB 20←
IF w=2 THEN GOSUB 23←
840 IF M1=1 THEN JV=2←
IF M1=2 THEN JV=1←
860 IF JV=2 THEN←
P=P-1←
ON P GOSUB ,,3,4,5,6←
IF P<4 THEN RETURN ELSE 820←
END IF←
880 P=P+1:ON P GOSUB ,,,4,5,6←
890 GOSUB 31←
910 LOCATE 1,21:PRINT T←
920 IF P>3 AND P<PX THEN 820←
930 f=1:RETURN←
950 P=7:PUT (X,Y),r1,PSET←
960 IF w=1 THEN GOSUB 20←
IF w=2 THEN GOSUB 23←
980 IF M1=1 THEN JV=1←
IF M1=2 THEN JV=2←
990 IF JV=1 THEN P=P-1←
1010 GOSUB 31←
1030 LOCATE 1,21:PRINT T←
1040 IF P<7 THEN P=3:RETURN←
1050 IF JV=1 THEN←
ON P GOSUB ,,,,,,7,8,9←
GOTO 960←
END IF←
```

```
1060 P=P+1€
ON P GOSUB ,,,,,,7,8,9€
1070 IF P>6 AND P<PZ THEN 960€
1075 f=1:RETURN€
1080 P=10€
PUT (X,Y),w4,PSET€
'FALLING MAN€
1090 Z2=23€
FOR Z=Z2 TO 150 STEP 6€
IF Z>Z2 THEN PUT (X,Z-6),s12€
1100 PUT (X,Z),s12€
SOUND (Z+15)*2,.08€
1110 NEXT Z€
PUT (X,Z-6),s12:Y=Y+50€
1120 FOR Z=150 TO Y STEP -6€
IF Z<150 THEN PUT (X,Z+6),s13€
1125 PUT (X,Z),s13€
SOUND (Z+15)*2,1:NEXT Z€
PUT (X,Z+6),s13€
1130 FOR Z=Y TO 150 STEP 6€
IF Z>Y THEN PUT (X,Z-6),s13€
1135 PUT (X,Z),s13€
SOUND (Z+15)*2,1:NEXT Z€
PUT (X,Z-6),s13:Y=Y+30€
IF Y<150 THEN 1120€
1160 PUT (X,Z-6),s13:LOCATE 3,6€
PRINT "Press <RETURN> to play again"€
LOCATE 4,7€
PRINT "Press <SPACE BAR> for menu"€
1170 a$=INKEY$€
IF a$<>" " AND a$<>CHR$(13) THEN 1170€
1180 c=0:M1=0:AV=0€
IF a$=CHR$(13) THEN 70€
GOSUB again:GOTO 70€
1510 T4=INT(TIMER)€
1520 a$=INKEY$€
IF a$="" THEN 1520€
1530 T5=INT(TIMER):T2=T2+T5-T4€
RETURN€
1740 SOUND 440,1:RETURN€
1770 SOUND 2300,1:RETURN€
3000 FOR DE=1 TO 400:NEXT:RETURN€
setup:€
DEFINT L,r,s,w€
SCREEN 1,320,200,2,1€
' OPEN WINDOW 3 WITH NO GADGETS OR€
' title BAR€
WINDOW 1,"",(0,0)-(311,25),16,1€
WINDOW 3,"",(0,0)-(311,185),16,1€
€
```

```
WINDOW OUTPUT 3
PALETTE 0,0,0,0
PALETTE 1,.5,1,1
PALETTE 2,1,0,0
PALETTE 3,1,1,.1
WIDTH 40
CLS
DIM voice%(8),w4%(200)
GET (0,0)-(25,20),w4%
RESTORE VOICEDATA
FOR J=0 TO 8
READ voice%(J)
NEXT
' Speech will be synchronous
VOICEDATA:
DATA 110,0,170,0,22200,64,10,1,0

talk$="This is Tightrope"
LOCATE 12,11
PRINT talk$
GOSUB talk
L= 87 :DIM w1%(L)
FOR I=0 TO L:READ a$:w1%(I)=VAL("&h"+a$):NEXT
shapedata:
DATA 18,15,2,30,0,58,0,F8
DATA 0,4070,800,C030,1800,2078,2000,1FFF
DATA C000,F8,0,F8,0,F8,0,F8
DATA 0,F8,0,F8,0,D8,0,D8
DATA 0,D8,0,78,0,38,0,1C
DATA 0,18,0,78,0,30,0,78
DATA 0,F8,0,4070,800,C030,1800,2078
DATA 2000,1FFF,C000,F8,0,F8,0,F8
DATA 0,F8,0,F8,0,F8,0,D8
DATA 0,D8,0,D8,0,78,0,38
DATA 0,1C,0,18,0,78,0,0
L= 87 :DIM w2%(L)
FOR I=0 TO L:READ a$:w2%(I)=VAL("&h"+a$):NEXT
DATA 18,15,2,30,0,58,0,F8
DATA 0,4070,800,C030,1800,2078,2000,1FFF
DATA C000,F8,0,F8,0,F8,0,F8
DATA 0,F8,0,F8,0,D8,0,198
DATA 0,318,0,318,0,318,0,18C
DATA 0,186,0,30C,0,30,0,78
DATA 0,F8,0,4070,800,C030,1800,2078
DATA 2000,1FFF,C000,F8,0,F8,0,F8
DATA 0,F8,0,F8,0,F8,0,D8
DATA 0,198,0,318,0,318,0,318
DATA 0,18C,0,186,0,30C,0,0
L= 87 :DIM w3%(L)
FOR I=0 TO L:READ a$:w3%(I)=VAL("&h"+a$):NEXT
```

```
DATA 18,15,2,30,0,58,0,F8<
DATA 0,4070,800,C030,1800,2078,2000,1FFF<
DATA C000,F8,0,F8,0,F8,0,F8<
DATA 0,F8,0,F8,0,D8,0,18C<
DATA 0,306,0,603,0,601,8000,600<
DATA C000,600,C000,1E01,8000,30,0,78<
DATA 0,F8,0,4070,800,C030,1800,2078<
DATA 2000,1FFF,C000,F8,0,F8,0,F8<
DATA 0,F8,0,F8,0,F8,0,D8<
DATA 0,18C,0,306,0,603,0,601<
DATA 8000,600,C000,600,C000,1E01,8000,0<
L= 87 :DIM r1%(L)<
FOR I=0 TO L:READ a$:r1%(I)=VAL("&h"+a$):NEXT<
DATA 18,15,2,30,0,58,1000,F8<
DATA 1800,70,7000,31,C000,7F,0,1FC<
DATA 0,7F8,0,1CF8,0,F0F8,0,40F8<
DATA 0,F8,0,F8,0,D8,0,198<
DATA 0,318,0,30C,0,306,0,186<
DATA 0,18C,0,300,0,30,0,78<
DATA 1000,F8,1800,70,7000,31,C000,7F<
DATA 0,1FC,0,7F8,0,1CF8,0,F0F8<
DATA 0,40F8,0,F8,0,F8,0,D8<
DATA 0,198,0,318,0,30C,0,306<
DATA 0,186,0,18C,0,300,0,0<
L= 87 :DIM r2%(L)<
FOR I=0 TO L:READ a$:r2%(I)=VAL("&h"+a$):NEXT<
DATA 18,15,2,30,3800,58,6000,F8<
DATA C000,71,8000,33,0,7E,0,1FC<
DATA 0,3F8,0,6F8,0,CF8,0,18F8<
DATA 0,30F8,0,E0F8,0,D8,0,198<
DATA 0,318,0,30C,0,307,C000,180<
DATA C000,180,4000,300,0,30,3800,78<
DATA 6000,F8,C000,71,8000,33,0,7E<
DATA 0,1FC,0,3F8,0,6F8,0,CF8<
DATA 0,18F8,0,30F8,0,E0F8,0,D8<
DATA 0,198,0,318,0,30C,0,307<
DATA C000,180,C000,180,4000,300,0,0<
L= 87 :DIM r3%(L)<
FOR I=0 TO L:READ a$:r3%(I)=VAL("&h"+a$):NEXT<
DATA 15,15,2,1860,798,3330,330,6598<
DATA 47C2,CF98,478C,C718,C7B0,6331,C7C0,3FE3<
DATA 400,1FC6,400,F8C,400,F98,798,FB0<
DATA 330,FE0,7C2,FC0,78C,C00,7B0,C00<
DATA 7C0,C00,400,C00,400,C00,400,C00<
DATA 400,C00,400,3C00,0,1860,0,3330<
DATA 0,6798,4000,CF98,4000,C718,C000,6331<
DATA C000,3FE3,0,1FC6,0,F8C,0,F98<
DATA 0,FB0,0,FE0,0,FC0,0,C00<
DATA 0,C00,0,C00,0,C00,0,C00<
DATA 0,C00,0,C00,0,3C00,0,0<
L= 87 :DIM l1%(L)<
```

```
FOR I=Ø TO L:READ a$:11%(I)=VAL("&h"+a$):NEXT←
DATA 18,15,2,Ø,98,3Ø,3Ø,4Ø58←
DATA C2,EØF8,8C,387Ø,BØ,E3Ø,CØ,3F8←
DATA Ø,FC,Ø,FF,Ø,F9,C098,F8←
DATA 783Ø,F8,1ØC2,F8,8C,D8,BØ,198←
DATA CØ,318,Ø,318,Ø,318,Ø,3ØC←
DATA Ø,FØ4,Ø,C,Ø,Ø,Ø,3Ø←
DATA Ø,4Ø78,Ø,EØF8,Ø,387Ø,Ø,E3Ø←
DATA Ø,3F8,Ø,FC,Ø,FF,Ø,F9←
DATA CØØØ,F8,78ØØ,F8,1ØØØ,F8,Ø,D8←
DATA Ø,198,Ø,318,Ø,318,Ø,318←
DATA Ø,3ØC,Ø,FØ4,Ø,C,Ø,Ø←
L= 87 :DIM 12%(L)←
FOR I=Ø TO L:READ a$:12%(I)=VAL("&h"+a$):NEXT←
DATA 16,15,2,Ø,98,EØ6Ø,3Ø,3ØBØ←
DATA C2,19FØ,8C,CEØ,BØ,66Ø,CØ,3FØ←
DATA Ø,1F8,Ø,1FC,Ø,1F6,98,1F3←
DATA 3Ø,1F1,8ØC2,1FØ,EØ8C,1BØ,4ØBØ,33Ø←
DATA CØ,63Ø,Ø,63Ø,Ø,63Ø,Ø,618←
DATA Ø,1EØ8,Ø,18,Ø,Ø,Ø,EØ6Ø←
DATA Ø,3ØFØ,Ø,19FØ,Ø,CEØ,Ø,66Ø←
DATA Ø,3FØ,Ø,1F8,Ø,1FC,Ø,1F6←
DATA Ø,1F3,Ø,1F1,8ØØØ,1FØ,EØØØ,1BØ←
DATA 4ØØØ,33Ø,Ø,63Ø,Ø,63Ø,Ø,63Ø←
DATA Ø,618,Ø,1EØ8,Ø,18,Ø,Ø←
L= 87 :DIM 13%(L)←
FOR I=Ø TO L:READ a$:13%(I)=VAL("&h"+a$):NEXT←
DATA 14,15,2,C3,98,199,8Ø3Ø,832C←
DATA CØC2,867C,CØ8C,C638,CØBØ,E319,8ØCØ,31FF←
DATA Ø,18FE,Ø,C7C,Ø,67C,98,37C←
DATA 3Ø,1FC,C2,FC,8C,C,BØ,C←
DATA CØ,C,Ø,C,Ø,C,Ø,C←
DATA Ø,C,Ø,F,Ø,C3,Ø,199←
DATA 8ØØØ,833C,CØØØ,867C,CØØØ,C638,CØØØ,E319←
DATA 8ØØØ,31FF,Ø,18FE,Ø,C7C,Ø,67C←
DATA Ø,37C,Ø,1FC,Ø,FC,Ø,C←
DATA Ø,C,Ø,C,Ø,C,Ø,C←
DATA Ø,C,Ø,C,Ø,F,Ø,Ø←
L= 87 :DIM s12%(L)←
FOR I=Ø TO L:READ a$:s12%(I)=VAL("&h"+a$):NEXT←
DATA 13,15,2,C3C3,98,C243,3Ø,63C6←
DATA C2,324C,8C,1998,BØ,FFØ,CØ,7EØ←
DATA Ø,7EØ,Ø,7EØ,Ø,7EØ,98,7EØ←
DATA 3Ø,7EØ,C2,7EØ,8C,66Ø,BØ,C3Ø←
DATA CØ,1818,Ø,1818,Ø,1818,Ø,1818←
DATA Ø,1818,Ø,781E,Ø,C3C3,Ø,C3C3←
DATA Ø,63C6,Ø,324C,Ø,1998,Ø,FFØ←
DATA Ø,7EØ,Ø,7EØ,Ø,7EØ,Ø,7EØ←
DATA Ø,7EØ,Ø,7EØ,Ø,7EØ,Ø,66Ø←
DATA Ø,C3Ø,Ø,1818,Ø,1818,Ø,1818←
```

```
DATA Ø,1818,Ø,1818,Ø,781E,Ø,Ø←
L= 39 :DIM s13%(L)←
FOR I=Ø TO L:READ a$:s13%(I)=VAL("&h"+a$):NEXT←
DATA 1Ø,12,2,7Ø,78,78,38,Ø←
DATA 1C,183E,1C7F,6CF,18F,4ØØF,67ØF,2F8F←
DATA B9FF,9Ø7F,CØØF,FFFF,7FFE,7Ø,78,78←
DATA 38,Ø,1C,183E,1C7F,6CF,18F,4ØØF←
DATA 67ØF,2F8F,B9FF,9Ø7F,CØØF,FFFF,7FFE,1818←
L= 19 :DIM s18%(L)←
FOR I=Ø TO L:READ a$:s18%(I)=VAL("&h"+a$):NEXT←
DATA 8,8,2,1CØØ,3EØØ,3EØØ,1CØØ,7EØØ←
DATA FFØØ,FFØØ,FFØØ,Ø,Ø,Ø,Ø,Ø←
DATA 8Ø,8Ø,8Ø,Ø←
L= 19 :DIM s17%(L)←
FOR I=Ø TO L:READ a$:s17%(I)=VAL("&h"+a$):NEXT←
DATA 8,8,2,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,1CØØ,3EØØ,3EØØ,1CØØ,7EØØ←
DATA FF8Ø,FF8Ø,FF8Ø,Ø←
L= 19 :DIM s19%(L)←
FOR I=Ø TO L:READ a$:s19%(I)=VAL("&h"+a$):NEXT←
DATA 8,8,2,1CØØ,3EØØ,3EØØ,1CØØ,7EØØ←
DATA FFØØ,FFØØ,FFØØ,1CØØ,3EØØ,3EØØ,1CØØ,7EØØ←
DATA FF8Ø,FF8Ø,FF8Ø,Ø←
L= 19 :DIM s15%(L)←
FOR I=Ø TO L:READ a$:s15%(I)=VAL("&h"+a$):NEXT←
DATA 8,8,2,C3ØØ,66ØØ,3CØØ,3CØØ,3CØØ←
DATA 3CØØ,66ØØ,C3ØØ,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø←
L= 19 :DIM s16%(L)←
FOR I=Ø TO L:READ a$:s16%(I)=VAL("&h"+a$):NEXT←
DATA 8,8,2,81ØØ,81ØØ,81ØØ,81ØØ,FFØØ←
DATA 81ØØ,81ØØ,81ØØ,81ØØ,81ØØ,81ØØ,81ØØ,FFØØ←
DATA 81ØØ,81ØØ,81ØØ,Ø←
L= 19 :DIM s14%(L)←
FOR I=Ø TO L:READ a$:s14%(I)=VAL("&h"+a$):NEXT←
DATA 8,8,2,7Ø,78,78,38,Ø←
DATA 1C,3E,7F,CCCF,CC8F,33ØF,33ØF,CC8F←
DATA CCFF,337F,33ØF,FFFF←
again:←
RANDOMIZE TIMER←
←
CLS←
GOSUB title←
RETURN←
←
announce:←
talk$=c$←
←
talk:←
IF talkflag=Ø THEN SAY TRANSLATE$(talk$),voice%←
RETURN←
```

```
←
title:←
talk$="press 1 for game, 2 for typing"←
GOSUB talk←
WINDOW 4,"      Press 1 or 2",(65,70)-(250,110),16,
1←
PRINT:PRINT "        1- Game"←
PRINT:PRINT "        2- Typing"←
←
grabkey:←
a$=INKEY$:IF a$="" THEN grabkey←
w=VAL(a$)←
IF w<1 OR w>2 THEN grabkey←
talk$="Press 0 through 9 to choose difficulty level
."←
GOSUB talk←
WINDOW 4,"Press 0-9 for difficulty",(65,70)-(255,11
0),16,1←
PRINT:PRINT:PRINT "    Enter level (0-9)"←
←
grabkey1:←
a$=INKEY$:IF a$="" OR (a$<"0" OR a$>"9") THEN grabk
ey1←
b=VAL(a$)←
d=b:b=10-b:IF w=1 THEN b=b*2←
IF w=2 THEN b=b*40←
WINDOW CLOSE 4←
temp$="typing"←
IF w=1 THEN temp$="game" ←
talk$=temp$+"       "+"level"+STR$(d)←
GOSUB talk←
RETURN←
←
getout:←
WINDOW CLOSE 3←
SCREEN CLOSE 1←
WINDOW 1,"Tightrope",,31,-1←
WINDOW OUTPUT 1←
END←
←
```

Beehive

Steve Michel

To avoid getting stung in this delightful strategy game, you'll need to plan ahead. Requires 512K RAM.

"Beehive" is a two-player strategy game that requires you to concentrate fully and develop long-range planning skills. The game board consists of 121 hexagons arranged in a sloping 11 × 11 matrix. The name derives from the playing field's resemblance to the geometric precision of a honeycomb. The first player is assigned the left and right borders of the honeycomb, while the second player is assigned the top and bottom edges.

The object of the game is deceptively simple. Each player tries to connect a continuous line from one of his or her borders to the other. If you're player 1, for instance, you need to connect the left border with the right. The players alternate turns, filling in cells of the honeycomb one at a time. While attempting to complete your own course, you must also try to block your opponent's way, and this requires strategic thinking. The first player to connect both borders wins the game. As a reward, tiny bee faces appear along the line of connection, clearly marking the path to victory.

Starting the Game

Type in and save the program to disk. Before running the program set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. Once you have saved a copy of the game, select *Start* from the *Run* menu. Beehive begins by asking for the name of each player. After both players have entered their names, the beehive grid is drawn, and play begins. The computer determines randomly which player should take the first turn. Each player takes a turn by moving the mouse pointer to the desired cell and pressing the left mouse button once.

When you choose a cell, it is filled with a solid circle, and your turn ends. While connecting your own borders, you

should also be trying to prevent the other player from making a connection. Play continues until one player or the other completes a continuous line from one border to the other. At this point a victor is declared, and bee faces replace the circles along the entire winning route.

Beehive includes synthesized speech. Either player can toggle the speech effects on or off at any time. Press the left button once: A small box appears, indicating the current speech status. If speech was turned on, it is now turned off, and vice versa. Press the left button again to erase the speech box and resume the game.

Winning Strategies

Like most two-player games, Beehive adjusts itself to the skill of the players. The basic concept is simple enough that even small children can enjoy playing. But when two knowledgeable players are matched, play proceeds at a much higher level. The flexibility of the game allows many different strategies.

Here are some important points for beginners to keep in mind. First, your opening move does *not* have to occur in one of your border rows. In fact, you can often establish a better strategic position by starting somewhere near the middle of the playing field. In a typical game, you will have to swing back and forth between an expanding, offensive posture and a defensive, blocking posture. The middle areas accommodate both strategies well.

Second, it is not necessary that all of your cells be connected. That is, a new cell doesn't necessarily have to touch one of your existing cells. Any empty cell in the hive is fair game for either player, and it's often advantageous to space out your cells to allow multiple paths between borders. Starting multiple pathways makes it harder for an opponent to block your progress completely.

Finally, keep in mind that the hexagonal shape of each cell permits you to move in six different directions. Try not to get locked into a strict, straight-line strategy too often. Any pathway that connects both borders is legal, and in many cases the winning path will be quite roundabout.

Beehive

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
←
CLS←
talk$="": GOSUB talk←
GOSUB init←
GOSUB getnames ←
start:←
CLS: RANDOMIZE TIMER←
markers = 0: winner = 0: prev.player = 0←
player = INT(2*RND(1)+1)←
FOR j = 1 TO 11: FOR k=1 TO 31: hive%(j,k)=0: NEXT
k: NEXT j ←
FOR j = 1 TO 20: pathlen(j) = 0: NEXT j←
FOR j = 1 TO 65: path%(j) = 0: used%(j) = 0: node%(
j) = 0: NEXT j←
GOSUB drawscreen←
BREAK ON: ON BREAK GOSUB closeup←
←
main:←
IF prev.player <> player THEN←
COLOR 4←
LOCATE 1,2: PRINT "Player:                    "←
LOCATE 1,2: PRINT "Player: ";←
COLOR colr(player): PRINT LEFT$(player$(player),15)
←
talk$=player$(player): GOSUB talk←
prev.player = player←
END IF←
WHILE MOUSE(0) = 0←
x = MOUSE(0)←
a$=INKEY$:IF a$=" " THEN GOSUB readkey←
WEND←
GOSUB checkmouse←
IF used THEN main←
GOSUB checkline←
IF possible = 1 THEN GOSUB checkwinner←
LOCATE 3,2: PRINT "                "←
IF winner = 1 THEN drawpath←
IF player = 1 THEN ←
player = 2     ←
ELSE    ←
player = 1      ←
END IF←
GOTO main←
←
init:←
CLS: colr(1) = 2: colr(2) = 3←
DIM colcor%(11): FOR j = 1 TO 11: READ colcor%(j):
```

```
NEXT j€
DATA 5,4,4,3,3,2,2,1,1,Ø,Ø€
DIM row.inc%(6), col.inc%(6)€
FOR j = 1 TO 6: READ row.inc%(j), col.inc%(j): NEXT
j€
DATA -1,-1,Ø,1,1,1,1,Ø,Ø,-1,-1,-1€
DIM hive%(11,31)€
DIM used%(65), node%(65), path%(65), pathlen(2Ø)€
SCREEN 1,64Ø,2ØØ,3,2€
WINDOW 1,"BEE HIVE",,16,1€
GOSUB setcolor€
DIM hexa(1ØØ),ball1(1ØØ),ball2(1ØØ),eyes1(1ØØ),eyes
2(1ØØ)€
LINE (3Ø,1Ø)-(12,15),7: LINE - STEP (Ø,1Ø),7: LINE
- STEP (18,5),7€
LINE - STEP (18,-5),7: LINE - STEP (Ø,-1Ø),7: LINE
- STEP (-18,-5),7€
LINE (3Ø,11)-(13,15),6: LINE - STEP (Ø,9),6: LINE -
STEP (17,5),6€
LINE - STEP (16,-4),6: LINE - STEP (Ø,-1Ø),6: LINE
- STEP (-17,-4),6€
GET (12,1Ø)-(48,3Ø),hexa €
CLS: CIRCLE (3Ø,2Ø),11,colr(1): PAINT (3Ø,2Ø),colr(
1): GET (2Ø,9)-(4Ø,31),ball1€
GOSUB parts: GET (18,12)-(42,3Ø), eyes1 €
CLS: CIRCLE (3Ø,2Ø),11,colr(2): PAINT (3Ø,2Ø),colr(
2): GET (2Ø,9)-(4Ø,31),ball2€
GOSUB parts: GET (18,12)-(42,3Ø), eyes2: CLS€
RETURN€
€
parts:  €
CIRCLE (25,19),4,1: CIRCLE (35,19),4,1€
PAINT (25,19),1: PAINT (35,19),1€
PSET (29,17): LINE - STEP (-5,-5): LINE - STEP (-5,
3)€
PSET (31,17): LINE - STEP (5,-5): LINE - STEP (5,3)
€
CIRCLE (3Ø,24),2,1: PAINT (3Ø,24),1€
RETURN€
€
getnames:€
COLOR 4€
CLS: talk$="WELCOME TO BEEE HIVE": GOSUB talk€
a$ = " What is the name of player 1 ": PRINT €
PRINT a$;: talk$=a$: GOSUB talk: INPUT player$(1)€
a$ = " What is the name of player 2 ": PRINT €
PRINT a$;: talk$=a$: GOSUB talk: INPUT player$(2)€
talk$="Press space bar to turn speech off or on dur
ing game."€
LOCATE 15,14:PRINT talk$€
GOSUB talk:CLS: RETURN€
```

```
←
drawscreen:←
CLS: y = 7←
FOR r = 1 TO 11←
x = 180 - r * 18←
FOR c = 1 TO 11←
x = x + 36←
PUT (x,y),hexa,OR←
NEXT c ←
y = y + 15 ←
NEXT r    ←
PSET (595,12),2: GOSUB upndown: LINE -STEP (0,10),2
←
PSET (596,12),2: GOSUB upndown: LINE -STEP (0,10),2
←
PSET (597,12),2: GOSUB upndown: LINE -STEP (0,10),2
←
PSET (194,12),2: GOSUB upndown: LINE -STEP (0,10),2
←
PSET (195,12),2: GOSUB upndown: LINE -STEP (0,10),2
←
PSET (196,12),2: GOSUB upndown: LINE -STEP (0,10),2
←
y1=-5: y2=5: PSET (198,9),3: GOSUB across←
PSET (198,10),3: GOSUB across←
PSET (199,11),3: GOSUB across←
y1=5: y2=-5: PSET (19,173),3: GOSUB across←
PSET (19,174),3: GOSUB across←
PSET (19,175),3: GOSUB across←
RETURN←
←
upndown:←
FOR j = 1 TO 10←
LINE -STEP (0,10),colr(1)←
LINE -STEP (-18,5),colr(1)←
NEXT j←
RETURN←
←
across:←
FOR j = 1 TO 11←
LINE -STEP (18,y1),colr(2)←
LINE -STEP (18,y2),colr(2)←
NEXT j←
RETURN←
←
checkmouse:          ←
x = MOUSE(3): y = MOUSE(4)←
offset = 0: used = 0←
yr = INT (y/15+.5): row = yr: yr = yr * 15 ←
IF INT (yr/2) = yr/2 THEN offset = 18←
xr = INT ((x-offset)/36+.5): col = xr: xr = xr * 36
```

```
+ offset€
IF row < 1 OR row > 11 THEN€
used = 1€
RETURN€
END IF€
col = col - colcor%(row)€
IF  col < 1 OR col > 11 THEN€
used = 1€
RETURN€
END IF€
rowhive = row: colhive = 10+2*col-row€
IF hive%(row,colhive) <> 0 THEN €
used = 1€
RETURN€
END IF  €
markers = markers + 1€
hive%(row,colhive) = player€
IF player = 1 THEN €
PUT (xr-10,yr-9),ball1,OR €
ELSE   €
PUT (xr-10,yr-9),ball2,OR€
END IF€
RETURN€
€
checkline:€
possible=1€
IF player = 2 THEN €
FOR row = 1 TO 6: ff=0: fb=0€
FOR col = 1 TO 11: colhive=10+2*col-row€
IF hive%(row,colhive)=player THEN ff=1€
colhive = 10+2*(col)-(12-row)€
IF hive%(12-row,colhive)=player THEN fb=1€
NEXT col  €
IF ff=0 OR fb=0 THEN €
possible = 0€
row = 1E+09€
END IF€
NEXT row€
ELSE€
FOR col = 1 TO 6: ff=0: fb=0€
FOR row = 1 TO 11: colhive=10+2*col-row€
IF hive%(row,colhive)=player THEN ff=1€
colhive = 10+2*(12-col)-row€
IF hive%(row,colhive)=player THEN fb=1€
NEXT row €
IF ff=0 OR fb=0 THEN €
possible = 0€
col = 1E+09€
END IF€
NEXT col€
END IF€
```

```
RETURN€
€
checkwinner:€
LOCATE 3,2: COLOR 4: PRINT "Checking..."€
used.cntr = Ø: winner = Ø: node.cntr = Ø: node.tota
l = Ø: counter = Ø€
IF player = 1 THEN check1€
FOR col = 1 TO 11: row = 1€
IF hive%(row,1Ø+2*col-row) <> player THEN skip2€
noderow = row: nodecol = col: GOSUB usedlookup€
IF used.flag = 1 THEN skip2€
node.total = 1: path.total = 1:counter = 1€
path%(1) = 1ØØ * noderow + nodecol€
GOSUB checkpath€
IF winner = 1 THEN col = 1E+Ø9€
skip2:€
NEXT col€
RETURN€
€
check1:  €
FOR row = 1 TO 11: col = 1€
IF hive%(row,1Ø+2*col-row) <> player THEN skip1€
noderow = row: nodecol = col: GOSUB usedlookup€
IF used.flag = 1 THEN skip1€
node.total = 1: path.total = 1: counter = 1€
path%(1) = 1ØØ * noderow + nodecol€
GOSUB checkpath€
IF winner = 1 THEN row = 1E+Ø9€
skip1:€
NEXT row€
RETURN€
€
usedlookup:€
used.flag = Ø: search = 1ØØ * noderow + nodecol€
lk = Ø: IF used.cntr = Ø THEN skipsearch€
FOR lk = 1 TO used.cntr€
IF search = used%(lk) THEN €
used.flag = 1€
lk = 1E+Ø9€
END IF   €
NEXT lk€
skipsearch:€
IF used.flag = Ø THEN€
used.cntr = used.cntr + 1€
used%(used.cntr) = search€
END IF€
RETURN€
€
checkpath:€
node.cntr = Ø€
```

```
FOR nc = 1 TO 6←
noderow = noderow + row.inc%(nc): nodecol = nodecol
+ col.inc%(nc)←
IF noderow < 1 OR noderow > 11 OR nodecol < 1 OR no
decol > 11 THEN skipnode
←
IF hive%(noderow,10+2*nodecol-noderow) <> player TH
EN skipnode←
GOSUB usedlookup: IF used.flag = 1 THEN skipnode←
node.cntr = node.cntr + 1←
node.total = node.total + 1: node%(node.total) = 10
0 * noderow + nodecol←
IF (player = 2 AND noderow = 11) OR (player = 1 AND
nodecol = 11) THEN ←
winner = 1←
path.total = path.total + 1←
path%(path.total) = 100 * noderow + nodecol ←
nc = 1E+09←
END IF  ←
skipnode:←
NEXT nc←
IF winner = 1 THEN RETURN←
IF node.cntr = 0  AND node.total = 0 THEN RETURN←
IF node.cntr = 0 THEN ←
path.total = path.total - pathlen(counter)←
pathlen(counter) = 0←
counter = counter - 1←
END IF   ←
IF node.cntr > 1 THEN counter = counter + node.cntr
- 1←
noderow = INT(node%(node.total)/100)←
nodecol = node%(node.total) - 100 * noderow←
path.total = path.total + 1←
pathlen(counter) = pathlen(counter) + 1←
path%(path.total) = node%(node.total)←
node.total = node.total - 1 ←
GOTO checkpath  ←
←
drawpath:←
LOCATE 1,1: PRINT "                    ": COLOR 4←
LOCATE 1,1: PRINT "THE WINNER: ";:COLOR colr(player
): PRINT player$(player);←
a$ = "THE WINNER IS " + player$(player): talk$=a$:
GOSUB talk←
FOR j = 1 TO path.total: offset = 0←
row = INT(path%(j)/100): col = path%(j) - 100*row +
colcor%(row)←
IF row/2 = INT(row/2) THEN offset = 18 ←
xr = col * 36 + offset: yr = row * 15←
IF player = 1 THEN←
```

```
PUT (xr-10,yr-9), ball1, XOR ←
PUT (xr-12,yr-5), eyes1, OR←
ELSE←
PUT (xr-10,yr-9), ball2, XOR←
PUT (xr-12,yr-5), eyes2, OR←
END IF←
NEXT j  ←
←
goagain:←
LINE (419,139)-(625,186),7,b: LINE (420,140)-(624,1
85),7,b←
LINE (421,141)-(623,184),4,bf: COLOR 6←
LOCATE 19,55: a$ = " WANT TO PLAY AGAIN ? ": PRINT
a$; ←
LINE (431,162)-(487,180),7,bf: LOCATE 22,56: PRINT
" YES ";←
LINE (567,162)-(615,180),7,bf: LOCATE 22,73: PRINT
" NO "; ←
talk$=a$: GOSUB talk←
←
waiter:←
WHILE MOUSE(0) <> 1←
WEND←
x = MOUSE(3): y = MOUSE(4)←
IF y < 162 OR y > 180 THEN waiter←
IF x > 430 AND x < 488 THEN start←
IF x > 566 AND x < 616 THEN closeup←
GOTO waiter←
←
setcolor:←
PALETTE 0,.3,.3,.3    'grey←
PALETTE 1,0,0,0       'black←
PALETTE 2,0,1,0       'green←
PALETTE 3,0,0,1       'blue←
PALETTE 4,1,1,1       'white←
PALETTE 5,0,1,1       'aqua←
PALETTE 6,1,1,0       'yellow←
PALETTE 7,.8,.2,0     'red←
RETURN←
←
closeup:←
PALETTE 0,.1,.1,1     'blue←
PALETTE 1,1,1,1       'white←
PALETTE 2,0,0,0       'black←
PALETTE 3,.85,.2,0    'red←
WINDOW CLOSE 1←
SCREEN CLOSE 1←
STOP←
←
←
readkey:←
```

```
WINDOW 4,"Speech",(250,70)-(390,110),16,1€
IF TalkFlag=1 THEN€
talk$="Now I can talk."€
PRINT talk$€
TalkFlag=1-TalkFlag€
GOSUB talk€
GOTO clearmouse€
END IF€
IF TalkFlag=0 THEN€
talk$="OK, I'll be quiet."€
PRINT talk$€
GOSUB talk€
TalkFlag=1-TalkFlag€
END IF€
€
clearmouse:€
WHILE MOUSE(0)<>0:WEND€
PRINT "Press button once"€
PRINT "to continue..."€
WHILE MOUSE(0)<>1:WEND€
WHILE INKEY$<>"":WEND€
WINDOW CLOSE 4€
RETURN€
€
talk:€
IF TalkFlag=0 THEN SAY TRANSLATE$(talk$)€
RETURN€
€
```

Pyramid Power

Mike Lightstone

This colorful action game, originally written for the IBM PC/PCjr, runs on any Amiga computer with 512K memory. A joystick is required.

The object of "Pyramid Power" is to fill in all the cubes that make up the pyramid by jumping onto each one—while evading some hazardous pursuers. The pyramid is 6 cubes wide by 6 cubes high. If you succeed in filling all 21 cubes, you advance to a new level.

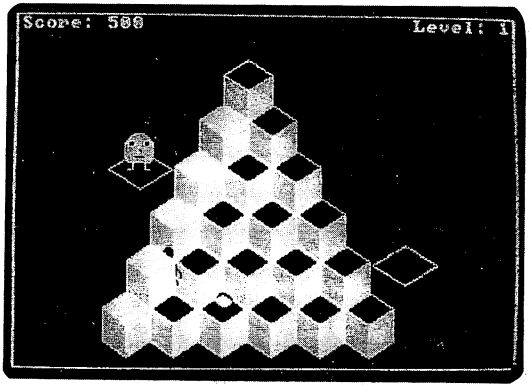
Your pursuers consist of a bouncing rock and a pesky buglike creature. The rock comes bouncing down randomly from the top of the screen, starting over again every time it reaches the bottom of the pyramid. The creature is a little smarter. It constantly follows your every move as you jump from cube to cube. If your player collides with either one, the game ends.

You can also lose the game by jumping in the wrong direction and falling off the edge of the pyramid. This happens frequently when you're fleeing in panic from the tumbling rock or nasty creature.

Type in the program and save a copy before you run it. The small ← character indicates where each program line ends. Don't try to type this character—we deliberately chose one that's not on the Amiga keyboard. The ← character merely shows where you should press RETURN (or move the cursor off the line) to enter one program line and start another. Before running the program set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. The joystick controls your movement. Plug the joystick into the port next to the mouse port (do not unplug the mouse).

Two Escape Routes

To make things a little easier, there are two special ways you can avoid your pursuers. A pair of elevators flanking the base of the pyramid stand ready to transport you at any time to the



"Pyramid Power" for the 512K Amiga features colorful action on a three-dimensional playing field.

apex. To get on the elevator, you have to jump upward from the cubes at the bottom corners of the pyramid. Just get on the elevator and ride to the top. You can use an elevator as often as you like. But be careful not to miss when you jump, or you'll fall off the edge and lose the game.

The scoring system is pretty simple. Jumping on an empty cube is worth 100 points times the number of the level you're on, and elevator rides subtract 100 points times your level number. In other words, cubes are worth 100 points on level 1, 200 points on level 2, and so on. Elevator rides subtract 100 points on level 1, 200 points on level 2, and so on. Advancing to a new level earns a bonus of 1000 points. The program keeps track of your current score and high score, but the high score may reflect the points you gained before your last elevator ride. Finally, Pyramid Power gets harder at the third level and again at the eighth.

Pyramid Power

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
←
setup:←
CLEAR ,25000←
CLEAR ,65536&←
SCREEN 1,320,200,2,1←
WINDOW 1,"",(0,0)-(311,25),16,1←
WINDOW 2,"",(0,0)-(311,185),16,1←
WINDOW OUTPUT 2←
CLS←
```

```
PALETTE Ø,Ø,Ø,Ø←
PALETTE 3,1,1,1←
PALETTE 2,.8,Ø,.93←
PALETTE 1,Ø,.93,.87←
←
DIM b(12,7),c(8Ø),f(8Ø)←
sp=.25:lev=1:hs=Ø:RANDOMIZE TIMER←
CLS:LOCATE 4,8:COLOR 2,Ø←
PRINT "P Y R A M I D - P O W E R"←
COLOR 3,Ø:LOCATE 1Ø,1:GOSUB player←
PRINT "The object of the game is to change the"←
PRINT "color of all the cubes while avoiding"←
PRINT "the bouncing rocks and creatures. Use"←
PRINT "joystick #2 to move. For a fast trip"←
PRINT "up, take the elevators. Be careful not"←
PRINT "to fall off the edges."←
GOSUB creatureshape←
GOSUB button←
←
restart:←
CLS:LOCATE 4,1Ø:PRINT"Player:":PUT(15Ø,2Ø),a←
LOCATE 8,1Ø:PRINT"Creature:":PUT (155,5Ø),q←
LOCATE 12,1Ø:PRINT"Rock:":CIRCLE (158,92),5,3←
PAINT (158,92),3,3:LOCATE 16,1Ø←
PRINT"Elevator:":LINE(165,123)-(185,112),3←
LINE-(2Ø5,123),3:LINE-(185,134),3:LINE-(165,123),3←
GET(151,87)-(164,97),c:GET(164,111)-(2Ø6,135),f←
GOSUB button←
←
readdata:←
RESTORE:FOR z=1 TO 7:FOR z1=Ø TO 12←
READ b(z1,z):NEXT z1,z←
←
start:←
CLS:z1=Ø:FOR z=19Ø TO 4Ø STEP -26←
FOR z3=7Ø+z1*15 TO 22Ø-z1*15 STEP 3Ø←
LINE (z3,z)-(z3,z-18),3:LINE-(z3+15,z-27),3←
LINE-(z3,z-36),3:LINE-(z3-15,z-27),3:LINE-(z3,z-18)
,3←
LINE-(z3,z),3:LINE-(z3+15,z-9),3:LINE-(z3+15,z-27),
3←
LINE(z3,z)-(z3-15,z-9),3:LINE-(z3-15,z-27),3←
PAINT(z3+7,z-9),1,3:PAINT(z3-7,z-9),2,3←
NEXT:z1=z1+1:NEXT←
←
x=6:y=1←
GOSUB playerxy←
j=7:k=2:j1=.5:k1=-.5:k2=1.5←
PUT(49+j*15,23+(k-1)*26),c←
g=6:h=5:g1=Ø:h1=Ø←
PUT(5Ø+g*15,13+(h-1)*26),q←
```

```
f1=11:f2=5€
PUT(f1*15+56,5*26-3),f€
PUT(27,5*26-3),f€
checksquares:€
IF x<>INT(x) OR y<>INT(y) THEN€
GOSUB move€
IF k1=1 THEN gameover€
END IF€
IF sq=21 THEN finished€
IF x<>INT(x) OR y<>INT(y) THEN rock€
LOCATE 1,1:PRINT "Score:"score€
LOCATE 1,32:PRINT "Level:"lev€
IF STICK(2)<>0 AND STICK(3)<>0 THEN€
GOSUB move€
IF k1=1 THEN gameover €
END IF€
€
rock:€
PUT(49+j*15,23+(k-1)*26),c€
IF k=INT(k) AND k1=1.5 AND j=INT(j) THEN€
j1=INT(3*RND(1))-1:j1=j1/2€
k1=-.5:k2=k-.5:SOUND 126,2€
END IF€
IF j1=0 THEN j1=-.5€
IF j=x AND k=y THEN€
GOSUB creaturerock€
IF k1=1 THEN gameover€
END IF€
j=j+j1:k=k+k1:IF k=k2 THEN k1=1.5€
IF k=8 THEN k=1:j=6:k2=.5€
PUT(49+j*15,23+(k-1)*26),c€
€
creature:€
PUT(50+g*15,13+(h-1)*26),q€
IF g<>INT(g) OR h<>INT(h) THEN creaturecont€
IF g<x THEN g1=sp€
IF g>x THEN g1=-sp€
IF h>y THEN h1=-sp€
IF h<y THEN h1=sp€
IF h=y OR g=x THEN g1=0:h1=0€
IF g=x AND h<y THEN€
h1=sp:g1=(INT(3*RND(1))-1)*sp€
IF g1=0 THEN g1=sp€
END IF€
IF g=x AND h>y THEN€
h1=-sp:g1=(INT(3*RND(1))-1)*sp€
IF g1=0 THEN g1=-sp€
END IF€
IF h=y AND g<x THEN€
g1=sp:h1=(INT(3*RND(1))-1)*sp€
```

```
IF h1=0 OR h+h1>6 THEN h1=-sp
END IF
IF h=y AND g>x THEN
g1=-sp:h1=(INT(3*RND(1))-1)*sp
IF h1=0 OR h+h1>6 THEN h1=-sp
END IF
creaturecont:
g=g+g1:h=h+h1
PUT(50+g*15,13+(h-1)*26),q
IF x=g AND y=h THEN
GOSUB creaturerock
IF k1=1 THEN gameover
END IF
GOTO checksquares
```

```
move:
GOSUB playerxy
IF x<>INT(x) OR y<>INT(y) THEN movecont
IF STICK(2)=1 AND STICK(3)=1 THEN x1=.5:y1=.5
IF STICK(2)=-1 AND STICK(3)=1 THEN x1=-.5:y1=.5
IF STICK(2)=1 AND STICK(3)=-1 THEN x1=.5:y1=-.5
IF STICK(2)=-1 AND STICK(3)=-1 THEN x1=-.5:y1=-.5
movecont:
x=x+x1:y=y+y1
IF x=INT(x) OR y=INT(y) THEN x1=0:y1=0
IF x=INT(x) THEN SOUND 880,1 ELSE SOUND 440,2
IF x=INT(x) AND b(x,y)=1 THEN
GOSUB rocky:PAINT(47+x*15,30+(y-1)*27),3,3
sq=sq+1:b(x,y)=0:GOSUB rocky
nn=1:GOSUB scorecalc
END IF
IF sq=21 THEN RETURN
IF x=INT(x) AND y=INT(y) AND b(x,y)=4 THEN
GOSUB rightelevator:nn=-1:GOSUB scorecalc
END IF
IF x=INT(x) AND y=INT(y) AND b(x,y)=5 THEN
GOSUB leftelevator:nn=-1:GOSUB scorecalc
END IF
IF x=INT(x) AND y=INT(y) AND b(x,y)=3 THEN
GOSUB edge:IF k1=1 THEN RETURN
END IF
IF (j=x AND k=y) OR (g=x AND h=y) THEN
GOSUB creaturerock:IF k1=1 THEN RETURN
END IF
IF y<1 THEN y=1:x=6:x1=0:y1=0
GOSUB playerxy
RETURN
```

```
rocky:
PUT(49+j*15,23+(k-1)*26),c
```

```
PUT(50+g*15,13+(h-1)*26),q<
RETURN<
<
rightelevator:<
PUT(f1*15+56,5*26-3),f<
z1=5:FOR z=11 TO 7 STEP-.25<
GOSUB playerzz1<
PUT(z*15+56,z1*26-3),f<
z3=6-z1<
SOUND z3*200,1<
GOSUB playerzz1<
PUT(z*15+56,z1*26-3),f<
z1=z1-.25:NEXT<
PUT(f1*15+56,5*26-3),f<
x=6:y=1:RETURN<
<
leftelevator:<
PUT (27,5*26-3),f<
z1=5:FOR z=0 TO 4 STEP .25<
PUT(40+z*14,15+(z1-1)*26),a<
PUT(z*15+27,z1*26-3),f<
z3=6-z1<
SOUND z3*200,1<
PUT(40+z*14,15+(z1-1)*26),a<
PUT(z*15+27,z1*26-3),f<
z1=z1-.25:NEXT<
PUT(27,5*26-3),f<
x=6:y=1:RETURN<
<
finished:<
CLS:FOR Z2=3 TO 0 STEP -1<
z=13:z1=10<
FOR z3=1 TO 11<
LINE(155-z,100-z1)-(155+z,100+z1),Z2,b<
z=z+13:z1=z1+8<
SOUND z*10,2<
NEXT:NEXT<
score=score+lev*1000:lev=lev+1<
IF lev>2 THEN sp=.5<
IF lev>7 THEN sp=1<
sq=0:COLOR 3,0:ts=ts+21:GOTO readdata<
<
creaturerock:<
GOSUB playerxy:FOR z1=1 TO 20<
x=x+SIN(z1)/5:GOSUB playerxy<
SOUND 255,1<
GOSUB playerxy:x=x-SIN(z1)/5<
NEXT:k1=1:RETURN<
<
edge:<
z=y+.4:y1=-.2:IF x<6 THEN x1=-.12 ELSE x1=.12<
```

```
edgecont:←
IF z>6 THEN z=6←
z=z+y1:x=x+x1:y1=y1+.Ø3←
PUT(52+x*14,11+(z-1)*26),a←
SOUND z*18Ø,1←
PUT(52+x*14,11+(z-1)*26),a←
IF z>6 THEN k1=1:RETURN←
GOTO edgecont←
←
gameover:←
CLS:IF score>hs THEN hs=score←
ts=ts+sq:LOCATE 6,9:PRINT"High Score:"hs←
LOCATE 1Ø,9:PRINT"You scored"score"points."←
LOCATE 12,9:PRINT"You filled"ts"squares."←
LOCATE 14,9:PRINT"You were on level "MID$(STR$(lev)
,2)"."←
LOCATE 2Ø,4:PRINT"Do you wish to play again (Y/N)?"
←
key3:←
z$=UCASE$(INKEY$)←
IF z$="" OR (z$<>"Y" AND z$<>"N") THEN key3←
IF z$="Y" THEN←
score=Ø:lev=1:sq=Ø:ts=Ø:sp=.25:k1=Ø:GOTO readdata←
END IF←
GOTO quit←
←
playerxy:←
PUT(52+x*14,11+(y-1)*26),a:RETURN←
←
playerzz1:←
PUT(64+(z+1)*14,18+(z1-1)*26),a:RETURN←
←
scorecalc:←
score=score+nn*1ØØ*lev:RETURN←
←
griddata:←
DATA 3,3,3,3,3,3,1,3,3,3,3,3,3←
DATA 3,3,3,3,3,1,Ø,1,3,3,3,3,3←
DATA 3,3,3,3,1,Ø,1,Ø,1,3,3,3,3←
DATA 3,3,3,1,Ø,1,Ø,1,Ø,1,3,3,3←
DATA 5,3,1,Ø,1,Ø,1,Ø,1,Ø,1,3,4←
DATA 3,1,Ø,1,Ø,1,Ø,1,Ø,1,Ø,1,3←
DATA 3,3,3,3,3,3,3,3,3,3,3,3,3←
←
quit:←
WINDOW CLOSE 2←
SCREEN CLOSE 1←
WINDOW 1,"Pyramid Power",,31,-1←
CLEAR ,25ØØØ←
END  ←
←
```

```
player:←
DEFINT a,q:l=87:DIM a(l):RESTORE player←
FOR i=Ø TO l:READ a$:a(i)=VAL("&h"+a$):NEXT:RETURN←
DATA 13,15,2,3F8,Ø,FFE,Ø,1FFF←
DATA Ø,3FFF,8ØØØ,7FFF,CØØØ,E3F8,EØØØ,E3F8←
DATA EØØØ,FFFF,EØØØ,FFFF,EØØØ,FFBF,EØØØ,FF1F←
DATA EØØØ,FFFF,EØØØ,FFFF,EØØØ,FCØ7,EØØØ,FFFF←
DATA EØØØ,7FFF,CØØØ,3FFF,8ØØØ,4Ø4,Ø,4Ø4←
DATA Ø,4Ø4,Ø,3CØ7,8ØØØ,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
←
creatureshape:←
l=87:DIM q(l):RESTORE creatureshape←
FOR i=Ø TO l:READ a$:q(i)=VAL("&h"+a$):NEXT:RETURN←
DATA 11,15,2,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA 38Ø,Ø,FEØ,Ø,FEØ,Ø,47C4,Ø←
DATA 1FFØ,Ø,3FF8,Ø,3FF8,Ø,3FFA,Ø←
DATA 3FFA,Ø,1FEØ,Ø,2Ø1Ø,1CØ,Ø,3EØ←
DATA Ø,7FØ,Ø,7FØ,Ø,7FØ,Ø,3EØ←
DATA Ø,23E2,Ø,57F5,Ø,8FF8,8ØØØ,1FFC←
DATA Ø,1FFC,Ø,1FFC,Ø,3FFE,Ø,5FFD←
DATA Ø,9FFC,8ØØØ,9FFC,8ØØØ,8FF8,8ØØØ,FF8←
DATA Ø,13E4,Ø,2ØØ2,Ø,2ØØ2,Ø,Ø←
←
button: ←
LOCATE 22,6:PRINT "Hit the fire button to play."←
WHILE STRIG(3)=Ø:WEND←
RETURN←
```

Biker Dave

David Schwener
Translation by Tim Midkiff

Here's a game guaranteed to bring out the daredevil in any computer owner.

As the ramp nears, you focus your mind, tighten your grip on the handlebars, and accelerate the motorcycle for the final approach. The deep, throaty cry of your machine's powerful engine drowns the spectators' cheers, and the onrushing wind pushes against your body like a gigantic hand. If your speed and timing aren't exactly right, you may overshoot the ramp and lose control or fall short into the line of cars.

Will you earn fame by surviving the jump or tumble into anonymity with a cartwheeling crash? As your speed mounts and the sidelines fade into a blur, there's no more time to wonder and no chance to turn back. Only the utmost in coordination and skill will bring you safely to earth on the other side.

Over the Ramp

In the upper left corner of the screen is the garage where you begin the ride. The rest of the screen contains the racetrack, with a couple of tunnels along the way, and a formidable obstacle which consists of several autos flanked by launching and landing ramps. Pressing the left mouse button controls your speed. (Avoid the right button; pressing it may crash the program.) Your goal is to ride down the track, through the tunnels, and toward the final obstacle, gaining just enough speed to jump over the cars without crashing. The game ends when you manage to jump nine cars at once or crash your last bike.

That may sound easy, but it's not as simple as you might think. For one thing, your bike is a specially built stunt machine with no brakes. Should you reach too high a speed, there's no way to slow down again. And if you accelerate too fast, the bike rises up into a wheelie. That's not bad in itself, but if you accelerate too hard from a wheelie position, the bike tips backwards and crashes.

As you approach the launching ramp, you need to go just fast enough to clear the parked cars, but not so fast that you lose control and miss the landing ramp on the other side. A successful jump requires precise timing and sure control of the throttle. The score you earn depends on the number of cars jumped and the number of attempts you made at that level.

Each time you jump over the cars, the racetrack crew moves the launching ramp and adds another car to the lineup. Unfortunately, the crew is somewhat unreliable and has been known to change the launching ramp's angle slightly when moving it. Thus, even though you may have jumped three cars with a speed of 100 miles per hour, there's no guarantee that the same speed will work every time.

Typing It In

Type in and save the program to disk. Before running the program, set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. To run the program, select *Start* from the *Run* menu.

Biker Dave

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
GOSUB Initialize←
←
Setup:←
COLOR 3,0:CLS:RANDOMIZE TIMER←
GOSUB IntScreen←
sp=5:b$=" ":s1=70:s2=110:jp=100:bi=5:t=0:tr=1:s=0:t
c=3:so=10:f=1000←
FOR i=0 TO 255:IF i<128 THEN pw(i)=127 ELSE pw(i)=-
128:NEXT:WAVE 3,pw ←
GET(283,1)-(307,17),t←
←
StartRun:←
FOR i=1 TO 500:NEXT:x=7:y=4:PUT (x,y),d0←
FOR i=1 TO 3:LOCATE 6,15:PRINT"GET READY":SOUND 176
0,2←
FOR j=1 TO 250:NEXT:LOCATE 6,15:PRINT"          "←
FOR j=1 TO 250:NEXT:NEXT:FOR i=1 TO 150:NEXT←
WHILE x<280 AND sp<s2←
x=x+INT(sp*.04)+1:IF MOUSE(0)<>0 THEN sp=sp+5←
LOCATE 21,19:PRINT sp:FOR i=1 TO 20-(sp MOD 25):NEX
T←
SOUND sp*so+f,.04,,3←
```

```
IF sp>s1 THEN PUT(x,y),d1,PSET ELSE PUT(x,y),dØ,PSE
T€
SOUND sp*so+f,.Ø4,,3€
IF sp>s1 THEN PUT(x,y),d1,PSET ELSE PUT(x,y),dØ,PSE
T€
PUT(283,1),t,PSET€
WEND€
IF sp>=s2 THEN CrashRight1€
€
JumpLeft:€
y=68:c1=114+c*8€
WHILE x>c1€
x=x-INT(sp*.Ø4)-1:IF MOUSE(Ø)<>Ø THEN sp=sp+5€
LOCATE 21,19:PRINT sp:FOR i=1 TO 2Ø-(sp MOD 25):NEX
T€
SOUND sp*so+f,.Ø2,,3€
PUT(x,y),d2,PSET:PUT(283,65),t,PSET€
WEND€
dx=-3:dy=INT(dx*.35):k=Ø:mp=INT(((sp-jp)*.5+jp)*.1Ø
5)€
PUT(x,y),d2:GOSUB Ramp1:PUT(x,y),d3:c1=c1-9€
1Ø x1=x:x=x+dx:y1=y:y=y+dy:k=k+1:FOR i=1 TO 2Ø-(sp
MOD 25):NEXT€
SOUND sp*so+f,.Ø2,,3€
IF x<93 THEN IF x>88 THEN IF y<48 OR y>56 THEN Cras
hLeft€
IF x<88 THEN IF y>68 THEN PUT(x1,y1),d3:y=68:dy=Ø€
IF k=mp THEN dy=-dy*.3€
IF dy=Ø THEN PUT(x,y),d2,PSET ELSE PUT(x1,y1),d3:PU
T(x,y),d3€
PUT(3,65),t,PSET:IF x<11 THEN PUT(92,5Ø),hp,PSET:GO
TO JumpRight€
GOTO 1Ø€
€
JumpRight:€
c1=2Ø9-c*16:y=132€
WHILE x<c1€
x=x+INT(sp*.Ø4)+1:IF MOUSE(Ø)<>Ø THEN sp=sp+5€
LOCATE 21,19:PRINT sp:FOR i=1 TO 2Ø-(sp MOD 5):NEXT
€
SOUND sp*so+f,.Ø2,,3€
PUT(x,y),dØ,PSET:PUT(3,129),t,PSET€
WEND€
dx=4:dy=-INT(dx*.5):k=Ø€
PUT(x,y),dØ:GOSUB Ramp2:PUT(x,y),d1€
mp=INT(((sp-jp+c)*.5+jp+c)*.1Ø5)€
2Ø x1=x:x=x+dx:y1=y:y=y+dy:k=k+1:FOR i=1 TO 2Ø-(sp
MOD 25):NEXT€
SOUND sp*so+f,.Ø2,,3€
IF x>291 THEN PUT(x1,y1),d1:GOTO ResetGame€
IF y>132 THEN y=132:dy=Ø:IF x<28Ø OR x>289 THEN Cra
```

```
shRight2 ELSE GoodJump€
IF k=mp THEN dy=-INT(dy*.3)€
PUT(x1,y1),d1:PUT(x,y),d1:GOTO 2Ø€
€
CrashLeft:€
PUT(x1,y1),d3:y=68:k=Ø€
WHILE x>11€
x=x+dx:FOR i=1 TO 5Ø-(sp MOD 25):NEXT:SOUND RND*25Ø
*so+f,.Ø2,,3€
IF k=Ø THEN PUT(x,y),d3,PSET:k=1 ELSE PUT(x,y),d2,P
SET:k=Ø€
PUT(3,65),t,PSET:PUT(92,5Ø),hp,PSET€
WEND€
GOTO ResetGame€
€
CrashRight1:€
k=Ø:dx=3€
WHILE x<28Ø€
x=x+dx:FOR i=1 TO 5Ø-(sp MOD 25):NEXT:SOUND RND*25Ø
*so+f,.Ø2,,3€
IF k=Ø THEN PUT(x,y),dØ,PSET:k=1 ELSE PUT(x,y),d1,P
SET:k=Ø€
PUT(283,1),t,PSET€
WEND€
GOTO ResetGame€
€
CrashRight2:€
k=Ø:dx=3€
WHILE x<291€
IF k=Ø THEN PUT(x1,y),d1:PUT(x,y),dØ:k=1 ELSE PUT(x
1,y),dØ:PUT(x,y),d1:k=Ø€
x1=x:x=x+dx:FOR i=1 TO 5Ø-(sp MOD 25):NEXT:SOUND RN
D*25Ø*sØ+f,.Ø2,,3€
WEND€
IF k=Ø THEN PUT(x1,y),d1 ELSE PUT(x1,y),dØ€
GOTO ResetGame€
€
GoodJump:€
PUT(x1,y1),d1:PUT(x1,y),dØ€
WHILE x<291€
PUT(x1,y),dØ:PUT(x,y),dØ€
x1=x:x=x+dx:FOR i=1 TO 2Ø-(sp MOD 5):NEXT€
WEND€
PUT(x1,y),dØ€
LOCATE 22,14:PRINT tc:c=c+1:tc=tc+c:s=2*(c*1Ø-tr*5)
:tr=2€
s1=RND*1Ø+36+8*c:s2=RND*15+81+8*c:jp=RND*2Ø+9Ø€
FOR i=t TO t+s STEP 5€
LOCATE 23,19:PRINT i:SOUND 15ØØ,.ØØ1,,3:FOR j=1 TO
5Ø:NEXT€
NEXT:t=t+s:IF c>9 THEN d=1:EndGame€
```

```
c1=99+(c-1)*8:PUT(c1,71),r1:GOSUB Ramp1←
c1=231-(c-1)*16:PUT(c1,135),r2:GOSUB Ramp2:GOSUB Ad
dCar←
GOTO 30←
←
ResetGame:←
bi=bi-1:j=3←
FOR i=1 TO 16:FOR k=1 TO 255:NEXT←
SOUND 200,.5,RND*255←
SCROLL (0,0)-(311,153),j,i:j=-j:NEXT←
30 IF bi>0 THEN tr=tr+1:LOCATE 22,35:PRINT bi ELSE
EndGame←
sp=5←
LOCATE 21,19:PRINT" 0  ":GOTO StartRun←
←
IntScreen:←
LINE(3,17)-(282,17),2←
LINE(282,81)-(28,81),2←
LINE(28,145)-(319,145),2←
COLOR 2←
AREA(287,144):AREA(253,135):AREA(253,144):AREAFILL←
PUT(92,50),hp ←
AREA(140,80):AREA(123,71):AREA(123,80):AREAFILL←
GET(123,71)-(140,80),r1←
AREA(183,144):AREA(200,135):AREA(200,144):AREAFILL←
GET(183,135)-(200,144),r2←
FOR c=1 TO 3:GOSUB AddCar:NEXT:c=3←
COLOR 1←
AREA(283,17):AREA(283,1):AREA(308,1):AREA(308,81):A
REA(283,81)←
AREA(283,65):AREA(291,65):AREA(291,17):AREA(283,17)
:AREAFILL←
AREA(27,81):AREA(27,65):AREA(3,65):AREA(3,145):AREA
(27,145)←
AREA(27,129):AREA(19,129):AREA(19,81):AREA(27,81):A
REAFILL←
LOCATE 21,13:PRINT"SPEED: 0   MPH"←
PRINT" CARS JUMPED: 0         BIKES LEFT: 5"←
LOCATE 23,13:PRINT"SCORE: 0"←
RETURN←
←
Ramp1:c1=99+c*8:PUT(c1,71),r1,PSET:RETURN←
Ramp2:c1=231-c*16:PUT(c1,135),r2,PSET:RETURN←
AddCar:c1=253-c*16:PUT(c1,137),a0,PSET:RETURN←
←
EndGame:←
CLS:COLOR 3←
IF d THEN←
LOCATE 5,5:PRINT"CONSIDER YOURSELF KING BIKER!!"←
ELSE←
```

```
LOCATE 7,4:PRINT"YOU HAVE WRECKED YOUR LAST BIKE!"←
END IF←
COLOR 1:LOCATE 9,5:PRINT"YOU SUCCESSFULLY JUMPED";t
c-c;"CARS"←
LOCATE 11,11:PRINT"FINAL SCORE IS";t:COLOR 3←
LOCATE 14,8:PRINT"PRESS BUTTON TO PLAY AGAIN":i=MOU
SE(Ø)←
WHILE MOUSE(Ø)=Ø:WEND←
GOTO Setup←
←
Initialize:←
DEFINT a-z:DEFSNG r,g,b:DIM t(1ØØ),r1(49),r2(49),pw
(255),nw(255)←
SCREEN 1,32Ø,2ØØ,2,1←
WINDOW 3,"",(Ø,Ø)-(311,185),16,1←
WINDOW OUTPUT 3←
WIDTH 4Ø:RESTORE PaletteData:FOR i=Ø TO 3:READ r,g,
b:PALETTE i,r,g,b:NEXT←
PaletteData:←
DATA .13,Ø,.73,.9,.9,.9,Ø,Ø,Ø,.8,.2,Ø←
←
DaveRight:←
DIM dØ(55):RESTORE DaveRight ←
FOR i=Ø TO 55:READ a$:dØ(i)=VAL("&H"+a$):NEXT←
DATA 19,D,2,Ø,4Ø,Ø,6ØØØ,Ø←
DATA 6Ø74,1,8Ø7F,3,8ØØØ,6,44ØØ,C←
DATA 3AØØ,1F,1ØØ,3F,BF94,4Ø,EØFE,Ø←
DATA CØ63,Ø,CØ3E,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA 4ØØ,Ø,AØØ,18,1ØØ,3E,3FØØ,7F←
DATA 7F8Ø,63,718Ø,63,318Ø,3E,1FØØ,Ø←
←
DaveRightWheel:←
DIM d1(55):RESTORE DaveRightWheel  ←
FOR i=Ø TO 55:READ a$:d1(i)=VAL("&H"+a$):NEXT←
DATA 18,D,2,18,Ø,18,EØ,1Ø←
DATA 8Ø,31,FØØ1,3F,38ØØ,3Ø,4ØØØ,3Ø←
DATA CØØØ,3F,8ØØØ,3F,8Ø7Ø,41,8ØC6,1←
DATA 8Ø7D,Ø,Ø,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,1,EØØØ,1,38ØØ,Ø←
DATA 7EØØ,Ø,E3ØØ,1,E3ØØ,3C,BEØØ,7E←
DATA 8ØØØ,62,Ø,63,Ø,3E,Ø,Ø←
←
DaveLeft:←
DIM d2(55):RESTORE DaveLeft←
FOR i=Ø TO 55:READ a$:d2(i)=VAL("&H"+a$):NEXT←
DATA 19,D,2,Ø,Ø,3ØØ,6Ø,3ØØ←
DATA 18,CØ,7E,EØ,Ø,1131,Ø,2E18←
DATA Ø,407C,Ø,FEFE,Ø,83C1,3E,18Ø←
DATA 63,18Ø,3E,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,1ØØØ←
```

```
DATA Ø,2800,Ø,400C,Ø,7E3E,Ø,FF3F€
DATA Ø,C763,Ø,C663,Ø,7C3E,Ø,Ø€
€
DaveLeftWheel:€
DIM d3(55):RESTORE DaveLeftWheel€
FOR i=Ø TO 55:READ a$:d3(i)=VAL("&H"+a$):NEXT€
DATA 18,D,2,18,18,18,8C,8€
DATA C,F8C,FD,1CFC,82,20C,Ø,30C€
DATA Ø,1FC,Ø,1FC,80,182,Ø,180€
DATA 3D,Ø,46,Ø,7C,Ø,Ø,Ø€
DATA Ø,Ø,Ø,780,Ø,1C80,Ø,7E00€
DATA Ø,C700,Ø,C780,Ø,7D3C,Ø,17E€
DATA Ø,46,Ø,C6,Ø,7C,Ø,Ø€
€
Auto:€
DIM aØ(19):RESTORE Auto€
FOR i=Ø TO 19:READ a$:aØ(i)=VAL("&H"+a$):NEXT€
DATA C,8,2,1F80,2040,4020,FFFØ,E070€
DATA FFFØ,Ø,Ø,1F80,2040,4020,FFFØ,9F90€
DATA FFFØ,C030,C030,Ø€
€
Hoop:€
DIM hp(45):RESTORE Hoop€
FOR i=Ø TO 45:READ a$:hp(i)=VAL("&H"+a$):NEXT€
DATA A,15,2,1E00,3300,6180,6180,C0C0€
DATA C0C0,C0C0,C0C0,C0C0,C0C0,C0C0,C0C0,C0C0€
DATA C0C0,C0C0,C0C0,C0C0,6180,6180,3300,1E00€
DATA 1E00,3300,6180,6180,C0C0,C0C0,C0C0,C0C0€
DATA C0C0,C0E0,C0E0,C0E0,C0E0,C0E0,C0E0,C0C0€
DATA C0C0,6180,6180,3300,1E00,A00,400€
€
RETURN€
€
```

Laser Strike

Barbara Schulak
Translation by Tim Midkiff

In this strategy game for one or two players, your mission is to locate and vaporize your opponent's hidden space force before your force suffers the same fate. Includes speech synthesis and stereo sound effects.

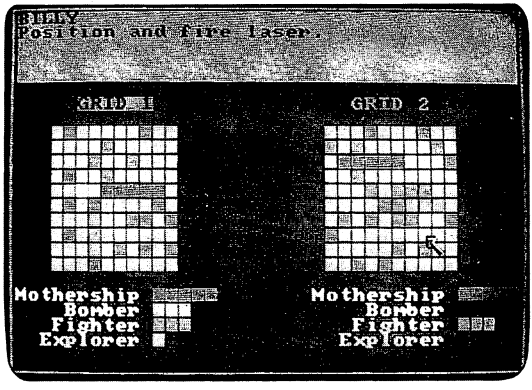
"Laser Strike" is a strategy game based on several popular board games (Battleship is probably the most famous). However, unlike the board games, the action in Laser Strike occurs in outer space. Two players secretly deploy their spaceships around the galaxy and then try to locate the opponent's ships by firing laser strikes on the two-dimensional galaxy grid. The first player to find and destroy all the opponent's ships is the winner.

Type in the program and save it to disk. Before running the program, set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. To run the program, select *Start* from the *Run* menu.

Beginning Play

Laser Strike begins by asking each player to enter his or her name. If you wish to play against the computer, press RETURN without typing anything at the first name prompt. If you press RETURN at both name prompts, the computer plays the entire game by itself.

The program then displays two grids, one for each player. In the first stage of the game, each player decides where to locate the ships in his or her grid. The deployment must be secret, so the second player needs to look away from the screen while the first player deploys ships, and vice versa. To deploy a ship, simply move the cursor to the desired location on the screen with the cursor keys and press the space bar. After choosing the location, you must also decide whether to deploy the ship horizontally or vertically. You cannot place a ship so



"Laser Strike," a strategy game with stereo sound effects.

that it overlaps the border. If you attempt to place a ship illegally, the program warns you and gives you another chance.

After both players' ships have been placed, the contest begins. The players alternate firing laser shots on each other's grids. To fire a shot, move the cursor to the desired location; then press the space bar. If you hit a ship, that square of the grid is marked in the color of the ship you hit. If you miss, the square is marked in a neutral color. The game continues until one player has found and eliminated all of the other player's ships. At that point you can exit the program or play a new game.

To hear the stereo effect, make sure that both of the Amiga's audio channels are connected to your monitor or amplifier as explained in the user's manual.

Laser Strike

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
DEFINT a-z:DEFSNG r,g,b:SAY ""←
DIM g(1,9,9),c(1,9,9),nh(1,5),xh(1,5),yh(1,5),dx(1,
5),dy(1,5),ta(1,5),s(27,8),w1(255),w2(255)←
SCREEN 1,320,200,3,1:WINDOW 3,"",(0,0)-(311,186),16
,1:WINDOW OUTPUT 3:COLOR 6,0←
FOR i=0 TO 255:w1(i)=RND*255-128:w2(i)=RND*255-128:
NEXT←
RESTORE PaletteData:FOR i=0 TO 7:READ r,g,b:PALETTE
i,r,g,b:NEXT←
PaletteData:←
DATA .13,0,.73,.13,0,.73,1,.73,0,.8,0,.93,.33,.87,0
,.8,.2,0,.9,.9,.9,0,.73,.73←
```

```
RESTORE VoiceData:FOR i=Ø TO 8:READ v%(i):NEXT←
VoiceData:←
DATA 11Ø,Ø,15Ø,Ø,22ØØ,64,1Ø,1,Ø←
WIDTH 4Ø:CLS:RANDOMIZE TIMER←
FOR co=1 TO 7:CLS:LINE(1,1)-(7,7),co,BF:GET(1,1)-(8
,8),s(Ø,co):NEXT←
CLS:LOCATE 11,14:CALL Echo("LASER STRIKE",v%()):PRI
NT←
gx(Ø)=24:gx(1)=192:gy=64←
b$="                                                "←
RESTORE ShipNames:FOR i=2 TO 5:READ s$(i):NEXT←
ShipNames:←
DATA Explorer,Fighter,Bomber,Mothership←
FOR i=Ø TO 1←
PRINT "Player"+STR$(i+1);:INPUT p$(i):IF p$(i)="" T
HEN cp(i)=1:p$(i)="Computer"←
NEXT:CLS:LOCATE 7,6:COLOR 1,Ø:PRINT"GRID 1"SPC(15)"
GRID 2"←
GOSUB DrawGrid:LOCATE 2Ø,1:COLOR 6,Ø ←
FOR n=5 TO 2 STEP-1:l=1Ø-LEN(s$(n))←
PRINT SPC(l)s$(n)SPC(13+l)s$(n):NEXT←
←
DeployShips:←
d=Ø:FOR pp=Ø TO 1:p=ABS(pp-1):ls(p)=5:FOR n=5 TO 2
STEP-1:er=1←
WHILE er=1←
LOCATE 1,1:PRINT p$(pp)←
PRINT"Deploy your "s$(n)".";←
FOR i=1 TO n:PUT(i*8+POS(Ø)*8,8),s(Ø,n),PSET:NEXT←
IF cp(pp)=Ø THEN GOSUB Human ELSE x=INT(RND*(1Ø-n))
:y=INT(RND*(1Ø-n))←
GOSUB SelectDir←
WEND←
GOSUB ClearTop:NEXT:GOSUB DrawGrid:NEXT←
p=1:d=1:GOSUB ClearTop:PALETTE 1,.5,.5,.5←
FOR i=Ø TO 1:px(i)=Ø:py(i)=Ø:NEXT←
←
MainLoop:←
WHILE th(p)<14:p=ABS(p-1):pp=p:WHILE INKEY$<>"":WEN
D←
LOCATE 7,6:COLOR p,ABS(p-1):PRINT"GRID 1";:COLOR AB
S(p-1),p:PRINT SPC(15)"GRID 2"←
LOCATE 1,1:COLOR Ø,1:PRINT p$(p)←
PRINT"Position and fire laser."←
IF cp(p)=Ø THEN GOSUB Human ELSE GOSUB Computer←
GOSUB FireLaser:GOSUB PutFigure:GOSUB ClearTop←
WEND←
←
EndGame:←
COLOR 5,Ø:CLS:LOCATE 11,15:CALL Echo("GAME OVER",v%
())←
```

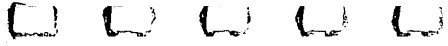
```
COLOR 4:LOCATE 13,20-LEN(p$(p))/2:CALL Echo(p$(p),v
%())←
LOCATE 14,9:CALL Echo("Has freed the galaxy!",v%())
←
COLOR 2:LOCATE 17,12:PRINT"Play again [Y/N]"←
SAY TRANSLATE$("play again"),v%:c=1←
WHILE k$<>"Y" AND k$<>"N"←
k$=UCASE$(INKEY$):c=ABS(c-1)←
COLOR 5+c:LOCATE 11,15:PRINT"GAME OVER":FOR i=1 TO
200:NEXT←
LOCATE 11,1:PRINT b$:FOR i=1 TO 200:NEXT←
WEND←
IF k$="Y" THEN RUN←
IF k$="N" THEN CLS:END←
←
Delay:FOR i=1 TO 1500:NEXT:RETURN←
←
ClearTop:LOCATE 1,1:COLOR,d:FOR i=1 TO 5:PRINT b$:N
EXT:RETURN←
←
PutFigure:PUT(x*8+gx(p),y*8+gy),s(0,co),PSET:pc=c(p
,x,y):c(p,x,y)=co:RETURN←
←
DrawGrid:←
co=7:FOR y=0 TO 9:FOR x=0 TO 9:FOR p=0 TO 1:IF c(p,
x,y)<>1 THEN GOSUB PutFigure←
NEXT p,x,y:RETURN←
←
CheckXY:←
IF x<0 OR x>9 THEN er=1←
IF y<0 OR y>9 THEN er=1←
RETURN←
←
FitShip:←
x1=x←
80 x1=x1+1:IF x1<=9 THEN IF g(p,x1,y)<>1 THEN 80←
xp=x1-x:x1=x←
90 x1=x1-1:IF x1>=0 THEN IF g(p,x1,y)<>1 THEN 90←
xn=x-x1:tx=xn+xp-1:y1=y←
100 y1=y1+1:IF y1<=9 THEN IF g(p,x,y1)<>1 THEN 100←
yp=y1-y:y1=y←
110 y1=y1-1:IF y1>=0 THEN IF g(p,x,y1)<>1 THEN 110←
yn=y-y1:ty=yn+yp-1:RETURN←
←
Human:←
x=px(p):y=py(p):x1=x:y1=y:co=6:GOSUB PutFigure:k$="
"←
WHILE k$<>" ":k$=INKEY$←
IF k$=CHR$(30) THEN IF x1<9 THEN x1=x1+1←
IF k$=CHR$(31) THEN IF x1>0 THEN x1=x1-1←
IF k$=CHR$(28) THEN IF y1>0 THEN y1=y1-1←
```

```
IF k$=CHR$(29) THEN IF y1<9 THEN y1=y1+1€
co=pc:GOSUB PutFigure:x=x1:y=y1:co=6:GOSUB PutFigur
e€
WEND:px(p)=x:py(p)=y:RETURN€
€
Computer:€
h=Ø:FOR n=2 TO 5:IF nh(p,n)>Ø AND nh(p,n)<n THEN sh
=n:h=1€
NEXT:IF h=Ø THEN 4Ø€
x=xh(p,sh):y=yh(p,sh):IF nh(p,sh)>1 THEN 2Ø€
FOR i=Ø TO 3:pr(i)=Ø:NEXT€
1Ø r=INT(RND*4):IF pr(r)=1 THEN 1Ø€
dx(p,sh)=Ø:dy(p,sh)=Ø:er=Ø:pr(r)=1:GOSUB FitShip€
IF r=Ø THEN dx(p,sh)=1:IF tx<sh OR xp<xn THEN er=1€
IF r=1 THEN dx(p,sh)=-1:IF tx<sh OR xn<xp THEN er=1
€
IF r=2 THEN dy(p,sh)=1:IF ty<sh OR yp<yn THEN er=1€
IF r=3 THEN dy(p,sh)=-1:IF ty<sh OR yn<yp THEN er=1
€
x=x+dx(p,sh):y=y+dy(p,sh)€
GOSUB CheckXY:IF er=Ø THEN IF g(p,x,y)<>1 THEN 5Ø€
x=x-dx(p,sh):y=y-dy(p,sh):GOTO 1Ø€
2Ø x=x+dx(p,sh):y=y+dy(p,sh)€
er=Ø:GOSUB CheckXY:IF er=1 THEN GOSUB 3Ø:GOTO 2Ø€
IF g(p,x,y)=1 THEN GOSUB 3Ø:GOTO 2Ø€
IF g(p,x,y)=Ø THEN GOSUB 3Ø€
GOTO 5Ø€
3Ø IF ta(p,sh)=Ø THEN dx(p,sh)=-dx(p,sh):dy(p,sh)=-
dy(p,sh):ta(p,sh)=1€
RETURN€
4Ø x=INT(RND*1Ø):y=INT(RND*1Ø)€
IF ((x+y) AND 1)=Ø THEN 4Ø€
IF g(p,x,y)=1 THEN 4Ø€
GOSUB FitShip:IF tx<ls(p) AND ty<ls(p) THEN 4Ø€
5Ø co=6:GOSUB PutFigure:RETURN€
€
SelectDir:€
IF g(p,x,y)<>Ø THEN 7Ø€
k$=CHR$(29+INT(RND*2))€
IF cp(pp)=Ø THEN €
PRINT"Horizontal [right] or vertical [down]?"€
k$="":WHILE k$<>CHR$(3Ø) AND k$<>CHR$(29):k$=INKEY$
:WEND€
END IF:er=Ø€
IF k$=CHR$(3Ø) THEN 6Ø€
IF y+n-1>9 THEN 7Ø€
FOR i=y TO y+n-1:IF g(p,x,i)<>Ø THEN er=1€
NEXT:IF er=1 THEN 7Ø€
y1=y:FOR y=y1 TO y1+n-1:g(p,x,y)=n:IF cp(pp)=Ø THEN
co=n:GOSUB PutFigure€
NEXT:RETURN€
```

```
60 IF x+n-1>9 THEN 70€
FOR i=x TO x+n-1:IF g(p,i,y)<>0 THEN er=1€
NEXT:IF er=1 THEN 70€
x1=x:FOR x=x1 TO x1+n-1:g(p,x,y)=n:IF cp(pp)=0 THEN
co=n:GOSUB PutFigure€
NEXT:RETURN€
70 er=1:IF cp(pp)=0 THEN LOCATE 4,1:COLOR 5:PRINT"I
NVALID CHOICE":GOSUB Delay€
COLOR 6:co=pc:GOSUB PutFigure:GOSUB ClearTop:RETURN
€
€
FireLaser:€
WAVE 0,SIN:WAVE 1,SIN:k=1€
FOR i=250 TO 1 STEP -75:k=ABS(k-1):SOUND 660,.5,i,k
€
FOR j=1 TO 500:NEXT:SOUND 0,0,0,k:FOR j=1 TO 500:NE
XT:NEXT€
n=g(p,x,y)€
IF n=0 THEN PRINT"MISS!":GOSUB Delay:co=1:g(p,x,y)=
1:RETURN€
IF n=1 THEN PRINT"ALREADY HIT":co=pc:GOSUB Delay:RE
TURN€
co=n:PRINT"DIRECT HIT!"€
WAVE 0,w1:WAVE 1,w2€
FOR i=255 TO 10 STEP-10:SOUND 100,.1,i,0:SOUND 100,
.1,i,3:FOR j=1 TO RND*20:NEXT:NEXT€
nh(p,n)=nh(p,n)+1:th(p)=th(p)+1€
g(p,x,y)=1:PUT((10+23*p+nh(p,n))*8,(24-n)*8),s(0,n)
,PSET€
IF n<>nh(p,n) THEN €
xh(p,n)=x:yh(p,n)=y€
ELSE€
FOR i=2 TO 5:IF nh(p,i)=0 THEN ls(p)=i€
NEXT€
IF n<>4 THEN €
SAY TRANSLATE$(s$(n)+" deestroyed"),v%:c=1€
ELSE €
SAY TRANSLATE$("bommer deestroyed"),v%:PRINT  s$(n)
€
END IF€
FOR i=1 TO 10:c=ABS(c-1):COLOR n+(6-n)*c,c€
LOCATE 4,20-LEN(s$(n))/2:PRINT UCASE$(s$(n))€
LOCATE 5,15:PRINT"DESTROYED!":FOR j=1 TO 50:NEXT:NE
XT€
END IF€
FOR i=1 TO 500:NEXT:RETURN€
€
SUB Echo(s$,v%(1)) STATIC€
SAY TRANSLATE$(s$),v%:PRINT s$:END SUB€
€
€
```

CHAPTER TWO

Education



Pioneer

Martin Mathis

A one- to four-player game, "Pioneer" tests your geography skills. Includes a medium-resolution map of the United States, speech, and mouse control. Requires 512K memory.

"Pioneer" is an educational geography game. The program uses medium-resolution graphics (for a map of the United States), speech and windows, and part of the game is mouse controlled. There are several ways to play Pioneer depending on the number of players, and it includes an Atlas option for practice or information.

The game is easy to play: Simply answer the questions about the states. In addition to testing your knowledge of geographical facts, you'll learn how to spell names like Tallahassee or Pennsylvania.

Typing In and Starting Pioneer

Pioneer is written in Amiga BASIC. It uses DATA statements for the geographical information, and uses the *x* and *y* coordinates of the lines and points to draw the map. Be careful typing these statements. One missing or misplaced DATA statement can cause the program not to run or, even worse, cause the program to be confused about which questions go with which answers.

After you've finished typing, be sure to save a copy. Before running the program set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. Then type RUN or select *Start* from the *Run* menu and turn up the volume. It takes some time for initialization.

Any system request windows will be behind the Pioneer screen. Right after starting Pioneer, *put the Workbench disk in the drive so the speech file can be loaded.*

Using the Atlas Option

Selecting F1 in the main menu takes you to the *atlas mode.* You'll see a prompt and the map. At the prompt you can enter one of the following:

- The name of a state or its mail code.
- The name of the capital or the largest city of a state.
- A number from 1 to 50 (to look for the state by its size rank).

Or you can select a state by pressing the mouse button at the approximate center of a state. In all of the above cases, the chosen state will be painted red on the map, and an information window about it will appear, displaying the state's capital, largest city, area (and rank in brackets), and its percentage of the total United States area. Click the mouse button or press the space bar to close the window. Hit ESC to exit the atlas mode.

The program will protest if you try to choose the ocean, a borderline, text, or if you type in an invalid city name.

Sometimes the program doesn't recognize a state if you click the mouse too close to the border. Imagine a rectangle centered within the state's borders. You can select the state with your mouse only if you click inside this rectangle. For that reason, for example, you must place the mouse pointer in the eastern part of Michigan and the western half of California to choose those states. Just practice a bit to get the idea.

Playing Alone

If you select the single-player option by pressing F2 from the main menu, you'll have to decide on the difficulty level. Levels 1, 2, and 3 present the easiest questions, but level 3 allows fewer wrong answers. Levels 4, 5 and 6 have more difficult questions, and levels 7, 8, and 9 contain the most difficult questions. The more difficult the question, the more points you'll get for a correct response.

The idea is to make it from Washington D.C. to either the Midwest, the South, or the West Coast (depending on the level) by answering questions and scoring points. Each point moves you closer to your destination. Correctly answering an easy question gives you one point; a medium-level question is worth two points, and a hard one yields four points.

If you don't know the correct answer press the HELP key,

which counts as a wrong answer. At the top of the screen, your current location and the distance remaining are displayed. Each point equals about 60 miles.

At a "Show me..." question, point to the state with the mouse pointer and press the button. The other answers have to be typed in. The game is over after you reach your destination or when too many questions are answered incorrectly. Your performance will then be rated.

Two to Four Players

If you select F3 from the menu you'll need to also indicate the number of players and their names. The object is to win as many states (and their land area) as possible. The final score is the number of states multiplied by the percentage of land area owned. The bigger the state you want to go for, the harder the questions become and the more of them you'll have to answer correctly. Small states are useful, too, because they can multiply your score and are easier to get. Just pick the state you want to win with your mouse. Note that there will never be questions about the state you're currently playing for.

At the beginning of a new round you can press the ESC key to abort the game. Otherwise the game ends after all 50 states are taken. The HELP key is also active again.

When the game is over each player gets his or her own score window, which will be behind the Hall of Fame window. Use the drag bar and the size gadget to view the other windows.

Pioneer

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
←
←
CLS:CLEAR,30000:RANDOMIZE TIMER:OPTION BASE 1←
DIM s$(50,5),abc$(50),abc(50),xpos(50),ypos(50),w(5
0),e(50),n(50),s(50),xtr(89),ytr(89),lct$(89)←
usa=3539341&←
clr$=">                              ":clr1$="A:
"←
SCREEN 1,320,200,2,1:WINDOW 1,"",(0,0)-(311,185),16
,1←
PALETTE 0,.3,.2,1←
PALETTE 1,.33,.87,0←
PALETTE 2,.95,.25,0←
PALETTE 3,.3,.2,1←
```

```
babble "just a moment please"←
FOR i=1 TO 50:FOR j=1 TO 5:READ s$(i,j):NEXT←
abc$(i)=s$(i,1):abc(i)=VAL(s$(i,5))←
IF s$(i,4)="s" THEN s$(i,4)=s$(i,3)←
NEXT←
shell=25←
shell: array=0:FOR i=1 TO 50-shell←
IF abc(i)<abc(i+shell) THEN SWAP abc$(i),abc$(i+she
ll):SWAP abc(i),abc(i+shell):array=1←
NEXT:IF array=1 THEN shell←
shell=INT(shell/2):IF shell>=1 THEN shell←
RESTORE FillDat:FOR i=1 TO 50:READ xpos(i),ypos(i):
NEXT←
RESTORE PickDat:FOR i=1 TO 50:READ w(i),e(i),n(i),s
(i):NEXT←
x2=62:y2=37:RESTORE USBorderDat:GOSUB DrawUS←
COLOR 2,0:LOCATE 3,10:h$="Welcome to the":p$="U.S.A
."←
FOR i=1 TO 14:FOR j=1 TO 3:LOCATE j,8+i:PRINT MID$(
h$,i,1)←
IF j>1 THEN LOCATE j-1,8+i:PRINT " "←
FOR k=0 TO 74:NEXT k,j,i←
RESTORE StateBorderDat:GOSUB DrawState←
x2=40:y2=124:RESTORE AKDat:GOSUB DrawAK←
FOR i=50 TO 1 STEP -1:PAINT(xpos(i),ypos(i)),1,3←
IF i=11 THEN c=1:GOSUB 98←
IF i=22 THEN c=1:GOSUB 99←
NEXT←
FOR i=1 TO 0 STEP -.002:PALETTE 3,i/3,i/5,i:NEXT
←
FOR i=1 TO 6:FOR j=1 TO 5:LOCATE j,24+i:PRINT MID$(
p$,i,1)←
IF j>1 THEN LOCATE j-1,24+i:PRINT " "←
FOR k=0 TO 124:NEXT k,j,i←
babble "this is the gaim. off py.onier."←
GOSUB clr:LOCATE 5,25:PRINT "       "←
RESTORE TrackDat:FOR i=1 TO 89:READ xtr(i),ytr(i),t
ct$(i):NEXT                                    ←
←
MainMenu:←
WINDOW 2,"Pioneer Menu",(15,20)-(142,52),0,1:COLOR
2,0:WINDOW OUTPUT 2←
PRINT "F1 Atlas Mode"←
PRINT "F2 One  Pioneer"←
PRINT "F3 Some Pioneers"←
PRINT "F4 Quit";←
babble "pleez select"←
key1:←
b$=INKEY$:IF b$<CHR$(129) OR b$>CHR$(132) THEN key1
←
ON ASC(b$)-128 GOTO Atlas,Pion1,Some,Kwit←
```

```
←
Atlas:←
WINDOW CLOSE 2:WINDOW OUTPUT 1:c$=""←
IF tgl=Ø THEN babble "at.lass mode":tgl=1←
LOCATE 1,14:COLOR 2,3:PRINT "Atlas mode":COLOR 2,Ø←
3 LOCATE 3,1:PRINT "> "c$;←
b$=INKEY$:GOSUB Pick:nogo=Ø:babble x$:IF b$=CHR$(13
) AND c$<>"" THEN 4←
IF b$=CHR$(8) AND c$<>"" THEN c$=LEFT$(c$,LEN(c$)-1
):LOCATE 3,1:PRINT clr$:GOTO 3←
IF b$=CHR$(27) THEN tgl=Ø:GOSUB clr:GOTO MainMenu←
IF b$>="Ø" AND b$<="9" AND VAL(c$+b$)<51 AND LEN(c$
)<2 THEN c$=c$+b$:GOTO 3←
IF (b$<"A" OR b$>"z") AND b$<>" " AND b$<>"." THEN
3←
c$=c$+b$:IF LEN(c$)>19 THEN c$="":LOCATE 3,1:PRINT
clr$←
GOTO 3←
4 IF UCASE$(c$)="PORTLAND" THEN port=1←
IF VAL(c$)>Ø AND VAL(c$)<51 THEN c$=abc$(VAL(c$))←
FOR i=1 TO 5Ø:FOR j=1 TO 4←
IF UCASE$(s$(i,j))=UCASE$(c$) THEN ii=i:jj=j:GOTO 5
←
NEXT j,i:babble "sorry, try again.":GOSUB clr:GOTO
Atlas←
5 x$=s$(ii,1)+","+s$(ii,2):PAINT(xpos(ii),ypos(ii))
,2,3 ←
IF ii=11 THEN c=2:GOSUB 98←
IF ii=22 THEN c=2:GOSUB 99←
GOTO StateInfo←
←
Pion1:←
didit=Ø:e$="":d$="questions":xdc=242:ydc=93:curloc$
="Washington DC"←
CLS:PRINT:PRINT "  Choose level":PRINT "      (1-9)
?"←
1Ø b$=INKEY$:IF b$<"1" OR b$>"9" THEN 1Ø ELSE CLS:P
RINT:PRINT "  Level chosen":PRINT SPACE$(6)"("b$")"
←
lev=VAL(b$):e$="easy ":xd=194:yd=82:dest$="Mid West
":mile=86Ø:pstep=61.46:frm=1:tom=14←
IF lev>3 THEN e$=e$+"and meedium ":xd=157:yd=135:de
st$="South":mile=149Ø:pstep=55.19:frm=15:tom=41←
IF lev>6 THEN e$=e$+"and hard ":xd=61:yd=12Ø:dest$=
"West Coast":mile=295Ø:pstep=61.43:frm=42:tom=89←
IF lev=1 OR lev=4 OR lev=7 THEN jsm=1.3←
IF lev=2 OR lev=5 OR lev=8 THEN jsm=1.2←
IF lev=3 OR lev=6 OR lev=9 THEN jsm=1.1←
points=tom-frm+1:abspts=INT(points*jsm+.5):mtg=mile
←
```

```
e$=e$+d$:babble e$←
WINDOW CLOSE 2:WINDOW OUTPUT 1←
LOCATE 2,3:COLOR 2:PRINT "Your destination is the "
dest$"."←
LOCATE 4,3:COLOR 1:PRINT "You start from Washington
D.C."←
FOR i=Ø TO 2←
FOR j=Ø TO 2:CIRCLE(xdc,ydc),j,3:CIRCLE(xd,yd),j,3:
NEXT:FOR k=Ø TO 999:NEXT←
FOR j=Ø TO 2:CIRCLE(xdc,ydc),j,2:CIRCLE(xd,yd),j,2:
NEXT:FOR k=Ø TO 999:NEXT k,i←
babble "try to make it from washington dc to the "+
dest$←
curloc=frm-1:lev=INT(lev/3+.9)←
town$(1)="Chicago":town$(2)="Dallas":town$(3)="Los
Angeles"←
cmnt$(1)="not getting killed by Al Capone.":cmnt$(2
)="competing with J.R. Ewing."←
cmnt$(3)="messing with Mexican immigrants.":cmnt$(4
)="getting into Hollywood's society."←
GameLoop:←
FOR i=Ø TO 9:b$=INKEY$:NEXT←
IF curloc>=frm THEN ←
FOR i=1 TO 50:IF lct$(curloc)=s$(i,2) THEN curloc$=
s$(i,1):ii=i←
NEXT←
END IF←
GOSUB StatLin←
question=INT(RND(1)*lev)+1:ON question GOSUB easy,m
edium,hard←
IF question=3 THEN question=4←
IF q1=Ø AND question=1 THEN GOSUB EasPick:GOTO 2Ø←
c$="":GOSUB GetAnswer:r1$=c$:eas=Ø←
2Ø GOSUB Check←
abspts=abspts-question:IF ans=1 THEN←
babl=INT(RND(1)*3)←
IF babl=Ø THEN babble "yoo are right"←
IF babl=1 THEN babble "that's correct."←
IF babl=2 THEN babble "you've got that one."←
points=points-question:mile=mile-question*pstep:cur
loc=curloc+question←
IF curloc>tom THEN curloc=tom←
FOR i=frm TO curloc:PSET(xtr(i),ytr(i)),2:SOUND RND
*5ØØ+9Ø,1:NEXT←
curloc$=lct$(curloc)←
IF curloc$="D" THEN mile=Ø:didit=1:curloc$=dest$:GO
SUB StatLin:GOTO Gover←
GOTO GameLoop←
END IF←
IF b$<>CHR$(139) THEN babble "oops."←
```

```
IF eas=Ø THEN LOCATE 4,1:COLOR 3:PRINT "The right a
nswer is ";:COLOR 2:PRINT r$;:COLOR 3:PRINT ".":FOR
i=Ø TO 6999:NEXT←
IF abspts<=Ø THEN didit=Ø:GOSUB StatLin:GOTO Gover←
GOTO GameLoop←
Gover:←
LOCATE 3,12:COLOR 2,3:PRINT "The Game is Over"←
babble "game over":FOR i=Ø TO 7999:NEXT:GOSUB clr←
IF didit=Ø THEN←
IF curloc >=frm THEN:FOR j=Ø TO 1:CIRCLE(xtr(curloc
),ytr(curloc)),j,2:NEXT←
LOCATE 1,1:COLOR 1:PRINT "You didn't make it to you
r destination."←
COLOR Ø:PRINT "You wind up in ";:COLOR 2:PRINT curl
oc$;:COLOR Ø←
PRINT " after":COLOR 2:PRINT INT(mtg-mile);:COLOR Ø
:PRINT "miles (";←
COLOR 2:PRINT STR$(INT((mtg-mile)/mtg*1ØØ+.5))"% ";
←
COLOR Ø:PRINT "), where you"←
job=INT(RND(1)*5):IF job=Ø THEN PRINT "build a log
cabin and become a hunter."←
IF job=1 THEN PRINT "get into the lumberjack busine
ss."←
IF job=2 THEN PRINT "build a house and start farmin
g."←
IF job=3 THEN PRINT "settle down and raise a family
."←
IF job=4 THEN PRINT "die lonely a couple of years l
ater."←
babble "well, that's tough luck, py.onier.":GOTO Ja
clyn←
END IF←
LOCATE 1,1:COLOR 1:PRINT "You made it to the "dest$
" !!!"←
COLOR Ø:PRINT "You build the town of ";:COLOR 2:PRI
NT town$(lev)←
COLOR Ø:PRINT "where you have a hard time"←
IF lev=3 AND RND(1)<.5 THEN lev=4←
PRINT cmnt$(lev):FOR i=Ø TO 3999:NEXT←
babble "you are faimous and rich, py.onier"←
Jaclyn:←
babble "hit any key to continue."←
FOR i=Ø TO 9:a$=INKEY$:NEXT←
3Ø a$=INKEY$:FOR i=frm TO curloc:PSET(xtr(i),ytr(i)
),2:NEXT←
FOR i=frm TO curloc:PSET(xtr(i),ytr(i)),3:NEXT←
IF a$="" THEN 3Ø←
COLOR 1,Ø:CLS:x2=62:y2=37:RESTORE USBorderDat:GOSUB
DrawUS←
RESTORE StateBorderDat:GOSUB DrawState←
```

```
x2=40:y2=124:RESTORE AKDat:GOSUB DrawAK←
FOR i=50 TO 1 STEP -1:PAINT(xpos(i),ypos(i)),1,3←
IF i=11 THEN c=1:GOSUB 98←
IF i=22 THEN c=1:GOSUB 99←
NEXT:GOTO MainMenu                                    ←
←
StatLin:←
GOSUB clr:COLOR 0,1:FOR i=1 TO 40:LOCATE 1,i:PRINT
" ":NEXT←
LOCATE 1,2:PRINT "Loc: "curloc$:mile$=STR$(INT(mile
+.5))←
LOCATE 1,21:PRINT "/ "MID$(mile$,2,LEN(mile$)-1)" m
iles to go"←
RETURN←
←
Pion2:←
round=round+1:p2=1:FOR g=1 TO p:r1$="":IF sta>49 TH
EN GameOver←
COLOR 2,3:LOCATE 1,1:PRINT "Round"round←
PRINT "Pioneer "p$(g)" >":COLOR 1,0←
babble "your turn "+p$(g)←
7 IF MOUSE(0)<>0 THEN 7←
FOR i=0 TO 9:b$=INKEY$:NEXT←
8 b$=INKEY$←
IF b$=CHR$(27) AND g=1 AND round>1 THEN round=round
-1:babble "game aborted after"+STR$(round)+" rounds
":GOTO GameOver←
GOSUB Pick:IF nogo=1 THEN nogo=0:babble x$:GOTO 8 ←
h=ii:are=VAL(s$(h,5)):GOSUB clr:COLOR 2,3:LOCATE 1,
1:PRINT s$(h,1)←
GOSUB easy:IF q1=0 THEN GOSUB EasPick←
c$="":IF q1>0 THEN GOSUB GetAnswer:r1$=c$:eas=0←
GOSUB Check:IF ans=0 THEN nay  ←
IF are>9200 THEN←
babble "alrite, but there's more to answer"←
GOSUB clr:COLOR 2,3:LOCATE 1,1:PRINT s$(h,1)←
GOSUB medium:c$="":GOSUB GetAnswer:r1$=c$:GOSUB Che
ck:eas=0←
IF ans=0 THEN nay←
END IF←
IF are>59000& THEN←
babble "really, you're getting closer"←
GOSUB clr:COLOR 2,3:LOCATE 1,1:PRINT s$(h,1)←
GOSUB medium:c$="":GOSUB GetAnswer:r1$=c$:GOSUB Che
ck:eas=0←
IF ans=0 THEN nay←
END IF←
IF are>130000& THEN←
babble "yes, but the last question is a hard one."←
GOSUB clr:COLOR 2,3:LOCATE 1,1:PRINT s$(h,1)←
```

```
GOSUB hard:c$="":GOSUB GetAnswer:r1$=c$:GOSUB Check
:eas=0←
IF ans=0 THEN nay←
END IF←
yup:←
babble "o.k! you won the state."←
sta=sta+1:score(g)=score(g)+are:sta(g)=sta(g)+1:GOT
O goon←
nay:←
IF b$<>CHR$(139) THEN babble "nope"←
PAINT(xpos(h),ypos(h)),1,3←
IF h=11 THEN c=1:GOSUB 98←
IF h=22 THEN c=1:GOSUB 99←
IF eas=0 THEN COLOR 2,0:LOCATE 4,1:PRINT "The corre
ct answer is ";:COLOR 1:PRINT r$;:COLOR 2:PRINT "."
←
goon:←
GOSUB Windw:GOSUB clr:NEXT:GOTO Pion2←
←
GameOver:←
GOSUB clr:COLOR 2,3:LOCATE 2,14:PRINT "GAME OVER":C
OLOR 3,0:LOCATE 3,13←
PRINT round"Rounds":babble"gaim over":Wind=1:g=1←
WINDOW 2,p$(1),(5,13)-(170,73),16,1:WINDOW OUTPUT 2
:GOSUB Wind←
WINDOW 3,p$(2),(136,30)-(306,90),16,1:WINDOW OUTPUT
3:GOSUB Wind←
IF p>2 THEN WINDOW 4,p$(3),(5,103)-(170,163),16,1:W
INDOW OUTPUT 4:GOSUB Wind←
IF p>3 THEN WINDOW 5,p$(4),(136,120)-(306,180),16,1
:WINDOW OUTPUT 5:GOSUB Wind←
WINDOW 6," Pioneers' Hall of Fame",(60,40)-(250,160
),19,1:WINDOW OUTPUT 6←
SOUND 1100,6:FOR i=1 TO p:per$=STR$(score(i)):GOSUB
Percentage←
score(i)=VAL(per$):ttl(i)=sta(i)*score(i):NEXT←
FOR i=1 TO p-1:FOR j=i TO p←
IF ttl(i)<ttl(j) THEN SWAP p$(i),p$(j):SWAP ttl(i),
ttl(j):SWAP sta(i),sta(j):SWAP score(i),score(j)←
NEXT j,i←
FOR i=1 TO p:IF i=1 THEN COLOR 2,3 ELSE COLOR 1,3←
PRINT:PRINT STR$(i)". "p$(i)" "←
COLOR 2,0:PRINT sta(i)"X"score(i)"="ttl(i)←
NEXT:COLOR 1:PRINT:PRINT "Hit any key to continue";
←
FOR i=0 TO 9:b$=INKEY$:NEXT←
11 b$=INKEY$:IF b$="" THEN 11←
FOR i=2 TO 6:WINDOW CLOSE i:NEXT:WINDOW OUTPUT 1←
COLOR 1:GOSUB clr:Wind=0←
FOR i=50 TO 1 STEP -1:PAINT(xpos(i),ypos(i)),1,3←
IF i=11 THEN c=1:GOSUB 98←
```

```
IF i=22 THEN c=1:GOSUB 99←
NEXT←
p2=0:GOTO MainMenu←
←
GetAnswer:←
COLOR 3,0:LOCATE 3,1:PRINT clr1$:COLOR 1,0:LOCATE 3
,3:PRINT c$←
GA:←
b$=INKEY$:IF b$=CHR$(13) AND c$<>"" THEN RETURN←
IF b$=CHR$(139) THEN c$="":eas=0:babble "help":RETU
RN←
IF b$=CHR$(8) AND c$<>"" THEN c$=LEFT$(c$,LEN(c$)-1
):GOTO GetAnswer←
IF (b$>"z" OR b$<"0") AND b$<>" " AND b$<>"." THEN
GA←
c$=c$+b$:IF LEN(c$)>19 THEN c$=""←
GOTO GetAnswer←
←
Check:←
ans=0←
IF port=1 AND (UCASE$(r1$)="OREGON" OR UCASE$(r1$)=
"MAINE") THEN ans=1:port=0←
IF UCASE$(r$)=UCASE$(r1$) THEN ans=1←
IF hard=0 THEN RETURN←
hard=0:IF q1=0 THEN←
r1=VAL(r$):r2=VAL(r1$):r1=r1-r1/100*15:rh=r1+r1/100
*15←
IF r1<=r2 AND r2<=rh THEN ans=1:RETURN←
END IF←
IF q1=1 THEN←
FOR i=1 TO 50:IF UCASE$(abc$(i))=UCASE$(r1$) THEN i
i=i←
NEXT:IF ii=VAL(ar$) THEN ans=1←
END IF←
RETURN←
←
EasPick:←
r1$="":←
9 IF MOUSE(0)<>1 THEN 9←
x=MOUSE(3):y=MOUSE(4):FOR i=1 TO 50←
IF x>w(i) AND x<e(i) AND y>n(i) AND y<s(i) THEN r1$
=s$(i,1)←
NEXT:IF r1$="" THEN 9←
LOCATE 3,3:PRINT r1$:eas=1←
RETURN←
←
Windw:←
WINDOW 2,p$(g),(g*30,35+g*18)-(170+g*30,100+g*18),0
,1:WINDOW OUTPUT 2←
Wind:←
COLOR 3,0:PRINT "States owned:":COLOR 2:PRINT sta(g
```

```
)←
COLOR 3:PRINT "Area owned in sq.mi.:":COLOR 2:PRINT
score(g)←
COLOR 3:PRINT "Land area of USA:":COLOR 2:per$=STR$
(score(g))←
GOSUB Percentage:PRINT " "per$" %";←
IF Wind=1 THEN ←
PRINT:PRINT INT(sta(g)/round*1000)/1000"States/turn
";←
g=g+1:SOUND g*200,4:FOR i=0 TO 2999:NEXT:RETURN←
END IF←
COLOR 1:PRINT:PRINT:PRINT "Hit SPACE";←
FOR i=0 TO 9:b$=INKEY$:NEXT←
key3:←
b$=INKEY$:IF b$<>" " AND MOUSE(0)<>1 THEN key3←
WINDOW CLOSE 2:WINDOW OUTPUT 1:RETURN←
←
Some:←
CLS:PRINT:PRINT "How many players"←
PRINT "    (2 to 4) ?"←
key2:←
b$=INKEY$:IF b$<"2" OR b$>"4" THEN key2←
p=VAL(b$):babble b$+" py.oniers."←
CLS:PRINT "Enter names :"←
FOR i=1 TO p:babble "player"+STR$(i):INPUT p$(i)←
IF p$(i)="" THEN p$(i)="Player"+STR$(i)←
p$(i)=UCASE$(LEFT$(p$(i),15)):score(i)=0:sta(i)=0:N
EXT←
WINDOW OUTPUT 1:WINDOW CLOSE 2←
babble "fine. let's start."←
round=0:sta=0:GOTO Pion2←
←
StateInfo:←
GOSUB clr:av=(w(ii)+e(ii))/2:ar=1←
IF av>160 THEN WINDOW 2,x$,(8,25)-(140,98),0,1←
IF av<161 THEN WINDOW 2,x$,(170,25)-(302,98),0,1←
WINDOW OUTPUT 2←
COLOR 3,0:PRINT "Capital:":COLOR 2:PRINT s$(ii,3)←
COLOR 3:PRINT "Largest City:":COLOR 2:PRINT s$(ii,4
)←
COLOR 3:PRINT "Area in sq.mi.:":COLOR 2:PRINT s$(ii
,5)" ";←
FOR i=1 TO 50:IF abc$(i)=s$(ii,1) THEN ar=i←
NEXT:PRINT "("ar")"←
per$=s$(ii,5):GOSUB Percentage:PRINT per$"% of USAr
ea"←
COLOR 1:PRINT:PRINT "Hit SPACE";←
FOR i=0 TO 9:b$=INKEY$:NEXT←
6 b$=INKEY$:IF b$<>" " AND MOUSE(0)<>1 THEN 6←
WINDOW CLOSE 2:WINDOW OUTPUT 1:PAINT(xpos(ii),ypos(
ii)),1,3←
```

```
IF ii=11 THEN c=1:GOSUB 98←
IF ii=22 THEN c=1:GOSUB 99←
IF port=1 THEN port=0:ii=37:GOTO 5←
GOTO Atlas←
←
Percentage:←
a$=STR$(INT(VAL(per$)/usa*10000)/100):per$=RIGHT$(a
$,LEN(a$)-1)←
RETURN←
←
Pick:←
x$="":i=0:s=0:IF MOUSE(0)=1 THEN ←
x=MOUSE(3):y=MOUSE(4):pt=POINT(x,y)←
IF x<4 OR x>308 OR y<4 OR y>182 THEN Pick←
IF pt=0 THEN x$="salt water. yuch!":nogo=1:RETURN←
IF pt=3 THEN x$="watch the border patrol!":nogo=1:R
ETURN←
IF pt=2 THEN x$="nice try kiddo!":nogo=1:RETURN←
PAINT(x,y),2,3:FOR i=1 TO 50←
IF x>w(i) AND x<e(i) AND y>n(i) AND y<s(i) THEN x$=
s$(i,1)+","+s$(i,2):jj=1:ii=i:GOTO 1←
NEXT:PAINT(x,y),1,3:nogo=1←
1 IF i=11 THEN c=2:GOSUB 98←
IF i=22 THEN c=2:GOSUB 99←
IF nogo=1 OR p2=1 THEN RETURN←
GOTO StateInfo←
END IF←
nogo=1:RETURN ←
←
easy:←
q=INT(RND*50)+1:IF q=ii THEN easy←
q1=INT(RND*3)←
IF q1=0 THEN q$="Show me "+s$(q,1)+" on the map.":r
$=s$(q,1)←
IF q1=1 THEN q$="Mail code for "+s$(q,1)+"?":r$=s$(
q,2)←
IF q1=2 THEN q$="What state's "+s$(q,2)+" short for
?":r$=s$(q,1)←
GOTO qa←
medium:←
q=INT(RND*50)+1:IF q=ii THEN medium←
q1=INT(RND*4)←
IF q1=0 THEN q$=s$(q,4)+" is the largest city of?":
r$=s$(q,1):IF s$(q,4)="Portland" THEN port=1←
IF q1=1 THEN q$=s$(q,3)+" is the capital of?":r$=s$
(q,1)←
IF q1=2 THEN q$="What's the capital of "+s$(q,1)+"?
":r$=s$(q,3)←
IF q1=3 THEN q$="Largest city of "+s$(q,1)+" is?":r
$=s$(q,4)←
```

```
GOTO qa←
hard:←
q=INT(RND*50)+1:IF q=ii THEN hard←
q1=INT(RND*2):hard=1←
IF q1=0 THEN q$="Area of "+s$(q,1)+" (+/- 15%)?":r$
=s$(q,5)←
IF q1=1 THEN ar$=STR$(INT(RND*48)+2):q$="The"+ar$+"
. largest state ?":r$=abc$(VAL(ar$))←
qa:←
COLOR 3,0:LOCATE 2,1:PRINT "Q:";:COLOR 2,0:PRINT q$
←
COLOR 3,0:LOCATE 3,1:PRINT "A:";:COLOR 1,0←
RETURN←
←
Kwit:←
WINDOW OUTPUT 1:WINDOW CLOSE 2←
FOR i=0 TO 1 STEP .002:PALETTE 3,i/3,i/5,i:NEXT←
FOR i=0 TO 1 STEP .002:PALETTE 1,.33,.87-i/1.43,i:N
EXT←
LOCATE 11,14:COLOR 2:PRINT "Bye, bye ..."←
babble "thanx for bothering my bits. hurry back"←
FOR i=0 TO 124 STEP .2:PALETTE 2,RND,RND,RND:LINE(8
0+i,10)-(80+i,175),0:NEXT←
SYSTEM←
←
clr:←
LOCATE 1,1:FOR i=0 TO 3:PRINT "
"←
NEXT:RETURN←
←
DrawUS:←
READ x1,y1:IF x1=-1 THEN RETURN←
LINE(x1,y1)-(x2,y2),3:x2=x1:y2=y1:GOTO DrawUS←
←
DrawState:←
READ x1,y1,x2,y2:IF x1=-1 THEN RETURN←
LINE(x1,y1)-(x2,y2),3:GOTO DrawState←
←
DrawAK:←
READ x1,y1:IF x1=-1 THEN RETURN←
LINE(x1,y1)-(x2,y2),3:x2=x1:y2=y1:GOTO DrawAK←
←
98 PAINT(78,143),c,3:PAINT(69,138),c,3:PAINT(98,168
),c,3:PAINT(94,158),c,3←
RETURN ←
99 PAINT(205,75),c,3:PAINT(195,60),c,3:RETURN←
←
GeoData:←
DATA Alabama,AL,Montgomery,Birmingham,50766,Alaska,
AK,Juneau,Anchorage,570833,Arizona,AZ,Phoenix,s,113
510←
```

```
DATA Arkansas,AR,Little Rock,s,53187,California,CA,
Sacramento,Los Angeles,156297¢
DATA Colorado,CO,Denver,s,103598,Connecticut,CT,Har
tford,Bridgeport,4872¢
DATA Delaware,DE,Dover,Wilmington,1933,Florida,FL,T
allahassee,Jacksonville,54157,Georgia,GA,Atlanta,s,
58060¢
DATA Hawaii,HI,Honolulu,s,6427,Idaho,ID,Boise,s,824
13,Illinois,IL,Springfield,Chicago,55646¢
DATA Indiana,IN,Indianapolis,s,35936,Iowa,IA,Des Mo
ines,s,55965,Kansas,KS,Topeka,Wichita,81783¢
DATA Kentucky,KY,Frankfort,Louisville,39674,Louisia
na,LA,Baton Rouge,New Orleans,44520¢
DATA Maine,ME,Augusta,Portland,30995,Maryland,MD,An
napolis,Baltimore,9838¢
DATA Massachusetts,MA,Boston,s,7826,Michigan,MI,Lan
sing,Detroit,56959¢
DATA Minnesota,MN,St.Paul,Minneapolis,79548,Mississ
ippi,MS,Jackson,s,47234¢
DATA Missouri,MO,Jefferson City,St.Louis,68945,Mont
ana,MT,Helena,Billings,145388¢
DATA Nebraska,NE,Lincoln,Omaha,76639,Nevada,NV,Cars
on City,Las Vegas,109895¢
DATA New Hampshire,NH,Concord,Manchester,8992,New J
ersey,NJ,Trenton,Newark,7468¢
DATA New Mexico,NM,Santa Fe,Albuquerque,121336,New
York,NY,Albany,New York,47379¢
DATA North Carolina,NC,Raleigh,Charlotte,48843,Nort
h Dakota,ND,Bismarck,Fargo,69299¢
DATA Ohio,OH,Columbus,Cleveland,41004,Oklahoma,OK,O
klahoma City,s,68656¢
DATA Oregon,OR,Salem,Portland,96187,Pennsylvania,PA
,Harrisburg,Philadelphia,44892¢
DATA Rhode Island,RI,Providence,Providence,1054,Sou
th Carolina,SC,Columbia,s,30207¢
DATA South Dakota,SD,Pierre,Sioux Falls,75956,Tenne
ssee,TN,Nashville,Memphis,41154¢
DATA Texas,TX,Austin,Houston,262015,Utah,UT,Salt La
ke City,s,82076,Vermont,VT,Montpelier,Burlington,92
73¢
DATA Virginia,VA,Richmond,Norfolk,39700,Washington,
WA,Olympia,Seattle,66512¢
DATA West Virginia,WV,Charleston,s,24124,Wisconsin,
WI,Madison,Milwaukee,54424¢
DATA Wyoming,WY,Cheyenne,Casper,96988¢
¢
USBorderDat:¢
DATA 62,37,110,43,164,46,182,52,174,59,187,56,205,6
0,192,65,192,80,196,83,198,80,199¢
DATA 65,205,62,214,82,228,68,252,54,254,39,262,38,2
70,50,260,65,263,69,251,80 ¢
```

```
DATA 247,101,251,108,228,136,239,159,238,163,230,16
1,218,143,210,146,194,145←
DATA 190,153,171,152,155,162,157,170,148,166,136,15
1,130,150,128,155,112,137←
DATA 101,139,65,126,53,113,44,77,56,37,62,43,62,37,
-1,-1←
←
StateBorderDat:←
DATA 46,71,99,83,81,106,135,113,135,112,166,111,86,
81,75,130,63,76,59,91←
DATA 59,91,78,116,106,88,101,139,112,137,129,137,12
9,137,129,112,129,115←
DATA 142,115,142,115,142,125,142,125,167,128,167,12
8,170,151,130,44,126,89←
DATA 84,40,75,77,128,69,101,66,101,66,98,86,98,86,1
35,90,135,90,134,112←
DATA 87,40,90,62,90,62,100,69,153,46,156,86,156,86,
165,102,165,102,167,130←
DATA 128,63,153,64,127,79,155,80,135,95,162,96,174,
59,171,68,171,68←
DATA 181,81,181,81,178,95,178,95,190,108,190,108,18
6,120,186,120,182,132←
DATA 182,132,183,143,183,143,190,143,190,143,193,14
6,194,81,195,103,195,103←
DATA 190,108,180,80,192,79,156,76,176,76,160,92,178
,92,166,114,187,115←
DATA 168,133,181,133,197,82,212,80,206,82,207,96,20
7,96,199,103←
DATA 199,103,195,103,253,53,260,64,260,66,248,69,24
8,69,244,58,251,68←
DATA 252,55,248,70,251,78,250,73,257,70,257,70,259,
72,255,71,256,74←
DATA 251,78,241,72,241,72,220,77,220,77,225,89,225,
89,245,84,245,84←
DATA 244,75,243,84,249,92,248,91,245,91,245,91,242,
85,247,95,238,93←
DATA 238,93,237,89,237,89,229,90,229,90,229,88,207,
96,217,98,217,98←
DATA 224,87,234,90,227,102,227,102,219,103,219,103,
216,99,218,102,213,108←
DATA 188,113,246,102,222,107,212,117,186,121,228,11
4,228,114,240,121,217,116←
DATA 231,131,206,118,212,140,228,139,201,141,201,14
1,200,145,194,119←
DATA 197,145,52,51,79,58,181,58,192,65←
DATA 97,164,103,171,103,171,95,176,95,176,94,168,94
,168,98,164,91,153←
DATA 97,158,97,158,92,159,92,159,90,154,84,146,89,1
47,85,151,86,152,79←
DATA 140,81,144,81,144,76,144,76,144,75,141,75,141,
```

```
78,139,71,136,69,139€
DATA 69,139,65,137,65,137,68,134,68,134,72,136,-1,-
1,-1,-1€
AKDat:€
DATA 40,124,53,136,46,153,59,166,58,168,47,158,35,1
54,32,159,26,158€
DATA 12,164,14,161,25,153,16,147,18,130,29,124,40,1
24,-1,-1€
€
FillDat:€
DATA 200,130,34,135,90,120,180,120,60,100,120,100,2
55,75,246,89,230,150€
DATA 220,130,78,143,80,70,190,90,200,90,170,80,140,
100,210,100,180,140€
DATA 260,50,240,90,255,70,205,75,160,60,190,130,170
,100,110,60,140,90€
DATA 70,90,255,60,250,80,120,120,240,70,240,110,140
,50,215,90,150,120€
DATA 70,60,230,80,257,72,230,120,140,70,200,115,150
,140,90,90,250,60€
DATA 240,100,70,50,225,95,180,70,110,80€
€
PickDat:€
DATA 197,207,119,140,15,59,124,168,81,102,110,136,1
69,183,115,133,45,60€
DATA 75,120,106,135,90,109,251,257,73,78,244,248,87
,93,202,240,141,163€
DATA 213,224,123,139,67,105,135,177,77,99,68,78,184
,196,80,103,196,207€
DATA 82,98,161,178,77,92,136,163,96,111,201,216,101
,108,170,184,133,152€
DATA 257,271,39,59,238,243,86,94,252,264,67,70,195,
214,59,80,156,172,45,76€
DATA 185,196,125,143,166,179,92,114,91,128,41,66,13
6,157,80,95,68,82,80,102€
DATA 253,259,61,67,245,252,79,85,105,130,112,137,22
2,246,59,72,222,251€
DATA 106,114,130,154,45,64,207,220,82,96,143,167,11
2,125,47,76,59,71€
DATA 225,245,77,85,256,260,71,75,228,236,119,128,12
9,155,64,79,196,214€
DATA 112,118,130,167,129,170,85,105,87,106,247,252,
58,69,231,249,96,102€
DATA 52,80,38,51,222,228,90,102,180,193,65,79,102,1
27,70,86€
€
TrackDat:€
DATA 240,90,MD,238,88,MD,235,88,MD,231,89,MD,227,90
,WV,223,89,WV,219,88,OH€
DATA 215,88,OH,210,89,OH,205,90,IN,201,90,IN,199,87
,IN,196,85,IN,194,82,D€
```

```
DATA 241,98,VA,240,101,VA,240,105,NC,239,109,NC,238
,113,NC,238,117,NC←
DATA 235,120,SC,232,122,SC,229,124,SC,225,126,GA,22
1,127,GA,217,128,GA←
DATA 213,129,GA,209,130,AL,205,130,AL,201,130,AL,19
7,131,AL,193,132,MS←
DATA 189,133,MS,185,134,MS,181,135,LA,177,135,LA,17
3,135,LA,169,136,LA←
DATA 165,136,TX,161,136,TX,157,135,D,240,96,VA,236,
95,VA,232,95,VA,228,95,WV←
DATA 224,96,WV,220,96,WV,216,97,OH,212,98,KY,208,98
,KY,204,98,IN,200,99,IN←
DATA 196,100,IN,192,100,IL,188,101,IL,184,102,MO,18
0,102,MO,176,103,MO←
DATA 172,104,MO,168,105,MO,164,106,KS,160,107,KS,15
6,108,KS,153,110,KS←
DATA 151,113,OK,149,115,OK,145,116,OK,141,116,TX,13
7,117,TX,133,117,TX←
DATA 130,118,TX,125,118,NM,121,118,NM,117,117,NM,11
3,117,NM,109,116,NM←
DATA 105,115,NM,101,114,AZ,97,113,AZ,93,112,AZ,89,1
11,AZ,85,110,AZ←
DATA 81,110,AZ,77,110,NV,74,113,CA,71,115,CA,68,117
,CA,65,119,CA,61,120,D←
←
SUB babble (a$) STATIC←
SAY TRANSLATE$(a$)←
END SUB←
←
```

Hickory, Dickory, Dock

Barbara Schulak
Translation by John Krause

Fun and educational, "Hickory, Dickory, Dock" helps children learn the concepts of telling time by relating a digital clock display to a conventional clock face. Includes speech synthesis.

"Hickory, Dickory, Dock" offers an enjoyable way for children to learn how to tell time. Type in the program; then save a copy before running it. You'll also need to set Preferences to 60 columns.

When you run Hickory, Dickory, Dock, it displays a round clock face as well as a digital display. Four different activities are available. The first option lets youngsters practice telling time. As the positions of the clock hands change on the screen, the digital clock display changes as well. This shows the relationship between the spatial position of hands on a clock face and the numeric representation of time.

The other three activities test a youngster's time-telling ability for hours only, hours and half-hours, or five-minute intervals. Pick the option you want from the menu bar using the right mouse button. Move the hands to the correct position by using the mouse pointer and left button to change the hours or minutes. Button detection can be slow; be sure to hold the button down for about one full second. After five correct answers, the program plays a brief song and displays some graphics as a reward. After three incorrect choices, the program automatically moves the clock hands to the correct position.

Hickory, Dickory, Dock

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
CLEAR,30000←
GOSUB init←
loop:←
MENU 1,0,1←
IF MENU(0)=1 THEN ON MENU(1) GOSUB practice,hours,h
alf,five,quit←
IF p THEN GOSUB SetClock←
GOTO loop←
practice:←
COLOR 3,0:LOCATE 2,17:PRINT "      Practice  "←
SAY TRANSLATE$("praktis.")←
p=1:hour=0:minute=0:GOSUB UpdateAnalog←
HourDigital=0:MinuteDigital=0:GOSUB UpdateDigital←
RETURN←
hours:←
p=0:COLOR 3,0:LOCATE 2,17:PRINT "     Hours Test "←
SAY TRANSLATE$("hours test.")←
FOR count=1 TO 5←
MinuteDigital=0:HourDigital=INT(RND*12)*12←
GOSUB GetAnswer←
NEXT←
GOSUB music←
RETURN←
half:←
p=0:COLOR 3,0:LOCATE 2,17:PRINT "Half Hours Test"←
SAY TRANSLATE$("half hours test.")←
FOR count=1 TO 5←
MinuteDigital=CINT(RND)*72:HourDigital=INT(RND*12)*
12←
GOSUB GetAnswer←
NEXT←
GOSUB music←
RETURN←
five:←
p=0:COLOR 3,0:LOCATE 2,17:PRINT "  5 Minute Test"←
SAY TRANSLATE$("five minut test.")←
FOR count=1 TO 5←
MinuteDigital=INT(RND*12)*12:HourDigital=INT(RND*12
)*12←
GOSUB GetAnswer←
NEXT←
GOSUB music←
RETURN←
quit:←
SYSTEM←
SetClock:←
answer=0←
```

```
IF MOUSE(Ø)=1 THEN€
IF MOUSE(3)>22Ø AND MOUSE(3)<29Ø THEN€
IF MOUSE(4)>27 AND MOUSE(4)<36 THEN€
GOSUB IncHour€
IF p THEN GOSUB IncHourDigital€
END IF  €
IF MOUSE(4)>45 AND MOUSE(4)<54 THEN€
GOSUB IncMinute€
IF p THEN GOSUB IncMinuteDigital€
END IF  €
IF MOUSE(4)>63 AND MOUSE(4)<72 AND p=Ø THEN€
COLOR 1,Ø€
IF INT(hour/12)=HourDigital/12 AND minute=MinuteDig
ital THEN€
LOCATE 1Ø,23:PRINT "Correct!"€
answer=2:SAY TRANSLATE$("corekt.")€
ELSE€
LOCATE 1Ø,23:PRINT "Wrong!"€
answer=1:SAY TRANSLATE$("wrong.")€
END IF€
FOR i=Ø TO 2ØØØ:NEXT€
LOCATE 1Ø,23:PRINT SPACE$(9)€
END IF€
END IF€
END IF€
RETURN€
GetAnswer:€
MENU 1,Ø,Ø€
GOSUB UpdateDigital:wrong=Ø€
loop1: GOSUB SetClock€
IF answer=Ø THEN loop1€
IF answer=2 THEN€
RETURN€
ELSE€
wrong=wrong+1€
IF wrong<3 THEN€
GOTO loop1€
ELSE€
WHILE MinuteDigital<>minute:GOSUB IncMinute:WEND€
WHILE HourDigital/12<>INT(hour/12):GOSUB IncHour:WE
ND€
count=count-1€
FOR t=Ø TO 4ØØØ:NEXT€
END IF€
END IF€
RETURN€
music:€
FOR i=Ø TO 27:SOUND f(i),t(i):SOUND Ø,1:NEXT:RETURN
€
IncHourDigital:€
HourDigital=(OldHourDigital+12) MOD 144€
```

```
GOSUB UpdateDigital←
RETURN←
IncMinuteDigital:←
MinuteDigital=(OldMinuteDigital+12) MOD 144←
HourDigital=(OldHourDigital+1) MOD 144←
GOSUB UpdateDigital←
RETURN←
IncHour:←
hour=(OldHour+12) MOD 144←
GOSUB UpdateAnalog←
RETURN←
IncMinute:←
minute=(OldMinute+12) MOD 144←
hour=(OldHour+1) MOD 144←
GOSUB UpdateAnalog←
RETURN←
number:←
COLOR 1,2←
LOCATE r(i),c(i)←
IF i>9 THEN PRINT "1";←
PRINT CHR$(48+(i MOD 10))←
RETURN←
UpdateDigital:←
GOSUB DrawDigital←
OldHourDigital=HourDigital:OldMinuteDigital=MinuteD
igital←
GOSUB DrawDigital←
RETURN←
DrawDigital:←
IF OldHourDigital<12 THEN OldHourDigital=OldHourDig
ital+144←
IF OldHourDigital>119 THEN PUT (165,139),d(0,1)←
PUT (200,139),d(0,(OldHourDigital\12) MOD 10)←
PUT (245,139),d(0,INT(OldMinuteDigital*5/120))←
PUT (280,139),d(0,(OldMinuteDigital*5\12) MOD 10)←
RETURN←
ClrAnalog:←
GOSUB MinuteHand←
COLOR 2:AREAFILL←
GOSUB HourHand←
AREAFILL←
i=CINT(OldMinute/12):IF i=0 THEN i=12←
GOSUB number←
RETURN←
UpdateAnalog:←
GOSUB ClrAnalog←
OldMinute=minute:GOSUB MinuteHand←
COLOR 1:AREAFILL←
OldHour=hour:GOSUB HourHand←
AREAFILL←
RETURN←
```

```
MinuteHand:€
AREA (cx,cy)€
AREA (cx+60*x((OldMinute+143) MOD 144),cy+60*y((Old
Minute+143) MOD 144))€
AREA (cx+80*x(OldMinute MOD 144),cy+80*y(OldMinute
MOD 144))€
AREA (cx+60*x((OldMinute+1) MOD 144),cy+60*y((OldMi
nute+1) MOD 144))€
RETURN€
HourHand:€
AREA (cx,cy)€
AREA (cx+40*x((OldHour+142) MOD 144),cy+40*y((OldHo
ur+142) MOD 144))€
AREA (cx+50*x(OldHour MOD 144),cy+50*y(OldHour MOD
144))€
AREA (cx+40*x((OldHour+2) MOD 144),cy+40*y((OldHour
+2) MOD 144))€
RETURN€
init:€
SAY ""€
SCREEN 2,320,200,2,1€
WINDOW 2," Hickory, Dickory, Dock ",,2,2€
PALETTE 0,.6,.5,.3€
PALETTE 1,0,0,0€
PALETTE 2,.7,.7,.7€
PALETTE 3,0,0,0€
DIM s(300,6),d(300,9),x(143),y(143),r(12),c(12),f(2
7),t(27)€
MENU 1,0,1,"Test       "€
MENU 1,1,1,"Practice   "€
MENU 1,2,1,"Hours      "€
MENU 1,3,1,"Half hours"€
MENU 1,4,1,"5 minute   "€
MENU 1,5,1,"Quit       "€
MENU 2,0,0,"":MENU 3,0,0,"":MENU 4,0,0,""€
RANDOMIZE TIMER€
pi=4*ATN(1):p=1€
FOR i=0 TO 143€
x(i)=COS(pi*(i/72-.5))€
y(i)=SIN(pi*(i/72-.5))*.84€
NEXT€
FOR i=0 TO 6€
CLS:READ k€
FOR j=1 TO k:READ x,y:AREA (x,y):NEXT€
AREAFILL:GET (0,0)-(32,48),s(0,i)€
NEXT  €
FOR j=0 TO 9€
CLS:READ a$€
FOR i=1 TO 7€
IF MID$(a$,i,1)="1" THEN PUT (0,0),s(0,i-1)€
NEXT€
```

```
GET (0,0)-(32,48),d(0,j)  ←
NEXT←
CLS:ERASE s←
FOR i=1 TO 12:READ r(i),c(i):NEXT←
FOR i=0 TO 27:READ f(i),t(i):NEXT←
cx=100:cy=85←
CIRCLE (cx,cy),100,1:PAINT (cx,cy),1←
CIRCLE (cx,cy),90,2:PAINT (cx,cy),2←
COLOR 1,2←
FOR i=1 TO 12:GOSUB number:NEXT←
FOR i=0 TO 143 STEP 12←
CIRCLE (cx+84*x(i),cy+84*y(i)),3←
PAINT (cx+84*x(i),cy+84*y(i))←
NEXT  ←
FOR i=0 TO 59←
dx=COS(pi*(i/30-.5)):dy=SIN(pi*(i/30-.5))←
LINE (cx+82*dx,cy+69*dy)-STEP(4*dx,3*dy)←
NEXT
←
LINE (185,135)-(320,200),2,bf←
CIRCLE (236,151),3:PAINT (236,151)←
CIRCLE (236,171),3:PAINT (236,171)←
GOSUB DrawDigital←
LOCATE 4,23:PRINT " Hours "←
LOCATE 6,23:PRINT "Minutes"←
LOCATE 8,23:PRINT "  OK?  "←
SAY TRANSLATE$("welcome to hickory dickory dock.")←
GOSUB practice←
RETURN←
DATA 4,0,1,0,21,4,19,4,5 ←
DATA 4,1,0,26,0,22,4,5,4←
DATA 4,27,1,27,21,23,19,23,5←
DATA 4,27,23,27,43,23,39,23,25←
DATA 4,26,44,1,44,5,40,22,40←
DATA 4,0,43,0,23,4,25,4,39←
DATA 6,1,22,6,20,22,20,26,22,21,24,5,24←
DATA 1111110,0011000,0110111,0111101,1011001←
DATA 1101101,1101111,0111000,1111111,1111101←
DATA 4,14,7,17,10,18,13,17,16,14,17,11←
DATA 16,7,13,4,10,3,7,4,4,7,3,10←
DATA 783.99,2,783.99,2,783.99,2,880,2,880,2,880,2,7
83.99,11←
DATA 783.99,2,659.26,5,659.26,2,698.46,5,698.46,2,6
59.26,11←
DATA 659.26,2,523.25,5,523.25,2,659.26,5,659.26,2,5
87.33,5,587.33,2,880,8←
DATA 783.99,2,880,2,783.99,2,698.46,2,659.26,2,587.
33,2,523.25,11←
←
```

Switchbox

Todd Heimarck
Translation by Philip I. Nelson

A challenging two-player game of strategy, "Switchbox" looks easy at first, but takes time to master and permits many variations. Includes speech and stereophonic sound effects. Requires 512K of memory.

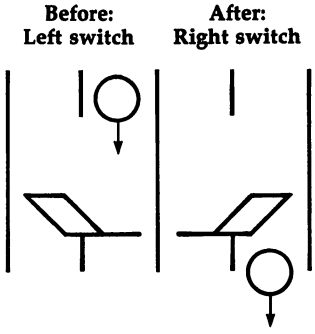
Playing "Switchbox" is like putting dominos in place for a chain reaction—either you're setting them in position or you're knocking them over. Winning requires skill and a sense of when to go for points and when to lie back and wait for a better board. The goal is simple: You try to score more points than your opponent by dropping balls into a box full of two-way switches. Each switch has a trigger and a platform. If the ball lands on an empty platform, it stops dead. But if it hits a trigger, it reverses the switch and continues. In many cases dropping a single ball creates a cascading effect—one ball sets another in motion, which sets others in motion, and so on, all the way down.

Type in the program listing and save a copy before you run it. Before running the program, make sure that the Amiga's display is set for 80 columns of text (if it isn't, the numbers printed on the screen won't match the rest of the display). If you've previously changed the display to 60 columns, open the Preferences icon and change it back to 80; then close Preferences, activate BASIC, and run the program as usual.

A Box of Switches

Switchbox is a tale of 2's: Each switch has two parts, two positions, two states, two paths in, and two paths out. The two parts are the platform and the trigger. A switch can lean to the left (platform left, trigger right) or to the right (platform right, trigger left):

Figure 1. Switch States



The trigger is weak, and always allows balls to pass. But the platform is strong enough to hold a single ball. So the platform either holds a ball—it's full—or it does not and is empty. When a ball sits on a platform, the switch is said to be loaded, or full.

Figure 2. Loaded Switch

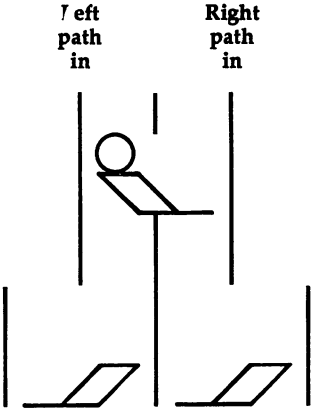


Figure 2 shows a full switch over two empty switches. The platform holds a ball and leans to the left. The trigger extends to the right. Note that the switch on top has two pathways leading in, the left path and the right, and that the right

path leading out is the left path into one of the switches below. The left path of the top switch leads into the right path of the switch below and to the left. If you drop a ball down the righthand path, it hits the trigger and flips that switch to the right. Then it continues down, hits the lefthand trigger below, and flips that switch as well.

In the meantime, the ball on the platform is set in motion (when the switch is flipped); it then hits the trigger. The top switch is reset to point to the left. The second ball then drops a level to the platform below, where it stops. The playing field is composed of five levels, with four switches in the first level and eight in the bottom level. At the beginning of the game, there are no balls on the field—all platforms are empty—and the position of each switch is chosen randomly.

Moving down the Path

Players alternate dropping balls into one of eight entry points. These balls (and others) may or may not make it all the way through the switchbox to one of the 16 exit paths. Balls fall straight down (with one exception), so a ball's movement is always predictable. When it hits an empty switch, one of two things can happen. If it lands on the empty platform, it stops dead in its tracks. But if it lands on a trigger, it falls through to the next level below.

Moving balls always make it through loaded switches. Triggers allow balls to continue, and move the switch to the other position. If the switch is loaded, the dead ball on the platform is put into motion and it hits the trigger that just moved over. This makes the switch go back to its original position, but with an empty platform. So when a ball hits the trigger of a loaded switch, its motion continues unabated. The switch moves, the ball on the platform begins to fall, and it hits the newly placed trigger. The newly emptied switch moves back again, and the two balls drop to the next level.

There's one more possibility: a ball dropping onto a platform that already holds a ball. A platform can't hold any more than one ball, so when this happens one of the balls slides over to the trigger. So the ball does not move straight down—it slides over to the next pathway. This is the exception to the rule that balls drop in a straight line. Of course, when the ball hits the trigger, the switch changes position, causing the other ball to drop and hit the trigger.

The Chain Reaction

At the game's start, all platforms are empty, so four of eight entry paths are blocked. Remember that your turn ends when a ball hits an empty platform and stops. As the switches fill up, the chances increase that a ball will descend through several levels. The goal is to score points by getting balls to pass all the way through the maze of the switchbox. The best way to collect a lot of points is to cause a chain reaction.

A ball that hits a loaded switch from either side continues on its way. And the previously inert ball on the platform starts moving. One enters; two exit. If both of those balls encounter full platforms, four drop from the switches. The pathways are staggered, so the effects can spread outward, with more and more balls cascading toward the bottom.

Rather than taking an easy point or two, it's often worthwhile to build up layers of loaded switches. Watch out for leaving yourself vulnerable, though. Because players take turns, you'll want to leave positions where your opponent's move gives you a chance to create a chain reaction. The best strategy is to play defensively. Look ahead a move or two, and watch for an opening that allows you to score several points at once.

Four Quarters

A game of Switchbox always lasts four rounds. In the first (equality), each exit counts for 2 points. Your goal is to score 10 points. The second quarter has more points available, as well as a higher goal. If you look at the exits, you'll see that the further they are away from the middle, the higher their point value. The numbers increase in a Fibonacci sequence: 1, 2, 3, 5, 8, and so on. Each number is the sum of the previous two ($1 + 2 = 3$, $2 + 3 = 5$, $3 + 5 = 8$, and so on). The target score in round 2 is 40.

In round 3 the numbers are a bit lower. They increase arithmetically (1, 2, 3, 4, and up to 8 in the corners). A goal of 20 points brings you to round 4, where you can score big. Here the numbers are squares: 1, 4, 9, 16, 25—all the way to 64 at the edges. In rounds 2–4, it's sometimes prudent to leave a middle path open for your opponent to score a few points, in order to gather a high score on the big numbers to the left and right.

Each round lasts until one player has reached the goal. At

that point the other player has one last turn before the round ends. It's possible to win the round on this last-chance play; watch out for barely topping the goal and leaving a chain reaction open for the other player. An arrow points to the scoreboard of the player whose turn it is. On the other side of the screen, you'll see a number where the arrow should be. That's the goal for the current round (the Amiga version displays the goal on both sides of the screen, below the scoreboards).

Bonus points are awarded at the conclusion of each round. Four numbers appear below the scoreboards. The first is simply the total at any given point in the game. The second is the total plus a bonus of the goal for the round if the player's points are equal to or greater than the goal. For example, if the goal is 20 and you get 18, there's no bonus. If you score 22, the bonus is the goal for that round (20) and you'll have 42 points. The third number under the scoreboard is the difference between scores for the rounds. If you win by 2 points, 2 points are added to your score (and 2 subtracted from the other player's). The final number is the grand total of the first three scores and bonuses. Rounds 1 and 3 are fairly low-scoring, with low goals. You may want to seed the field with extra balls during these quarters, so you can collect more points in the second and fourth quarters.

Variations

Although the goal of the game is to score the most points, there's no reason you couldn't agree to play for low score. In a "lowball" game, you would try to avoid scoring points. You wouldn't necessarily play backwards; you would have to adjust the strategy of where to place the balls. Fill up the board as much as possible and leave your opponent in a situation where he or she is forced to score points.

The Setup routine determines the goal for each round and the point values for the exit paths. You can prolong the game by doubling the goals; this also dilutes the value of a big score at the beginning of a round, preventing one player from winning on the first or second turn. An interesting variation is to assign negative values to some slots. If some paths score negative points, you are forced to think harder about where the balls will drop.

In addition to the numbered keys (1–8), the plus (+) and

minus (−) keys are active. Pressing plus drops a ball at random down one of the eight entry paths. Pressing minus allows you to pass your turn to your opponent.

Once you've mastered the regular game, you can add some new rules. Each player gets three passes per half, similar to the three timeouts in a football game. If you don't like the looks of the board, press the minus key to use one of your passes. After one player has skipped a turn, the other player must play (this prevents the possibility of six passes in a row). It's also a good idea to make a rule that a player can't pass on two consecutive turns. You could also give each player two random moves to be played for the opponent. In other words, after making a move, you could inform your opponent that you're going to give him or her one of your random moves, and then press the plus key.

Here's one more change you could make: Instead of alternating turns, allow a player to continue after scoring. When a player drops a ball and scores some points, the other player will have to pass (by pressing the minus key). If the first player scores again, the opponent passes again, and so on until no more points are scored.

Playing Solitaire

To drop a ball, press a numbered key (1–8). The numeric keypad is convenient for choosing a move. By using the pass and random-turn options, you can play against the computer. Here are the rules for solitaire play:

1. The computer always scores first. At the beginning of every round, the computer plays randomly until at least one point is acquired. Press the plus key for the computer's turn. You must continue passing (skip your turn with the minus key) until the computer puts points on the board.
2. After the first score by the computer, you can begin to play. When the computer has a turn, press the plus key for a random move.
3. Whenever you make points, you must pass again until the computer scores. When the computer gets more points, you can begin to play again. This rule means you should hold back on the easy scores of a few points; wait until there's an avalanche available.
4. If you're the first to reach the goal, the computer gets a last

chance. Don't make this move randomly; figure out the best opportunity for scoring and play that move for the last-chance turn.

In the interest of keeping these programs to a manageable length, no attempt has been made to provide an "intelligent" computer opponent. Once you become familiar with the game, you might find it an interesting project to try adding some routines that give the computer a rational basis for picking one move over another.

Switchbox

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
'Switchbox for 512K Amiga←
'Set Preferences for 80 columns←
←
Restart:←
CLEAR:GOSUB Setup←
←
Main:←
FOR Round=1 TO 4←
PUT (80,7+Round*8),Ball←
PUT (515,7+Round*8),Ball←
GOSUB Values     ←
SAY TRANSLATE$(Intro$(Round))←
Keepgoing:←
Who=1-Who 'alternate players←
GOSUB Taketurn ←
IF SC(1-Who,Round)=>Points(Round,0) THEN Nextround←
GOTO Keepgoing←
←
Nextround:←
FOR j=0 TO 1:FOR k=5 TO 8←
SC(j,k)=0:NEXT:NEXT←
FOR j=0 TO 1:FOR k=1 TO 4←
gx=Points(k,0):ac=SC(j,k)←
SC(j,5)=SC(j,5)+ac←
SC(j,6)=SC(j,6)-(ac=>gx)*gx←
SC(j,7)=SC(j,7)+SC(j,k)-SC(1-j,k)←
NEXT:NEXT←
FOR j=0 TO 1:FOR k=6 TO 7←
SC(j,k)=SC(j,k)+SC(j,5)←
NEXT:NEXT←
FOR j=0 TO 1:FOR k=5 TO 7←
SC(j,8)=SC(j,8)+SC(j,k)←
NEXT:NEXT←
FOR j=0 TO 1 ←
FOR k=5 TO 8:y$=STR$(SC(j,k))←
```

```
x=LEN(y$):tx=8+j*64-x:ty=4+k€
LOCATE ty,tx-1:PRINT SPACE$(2)€
LOCATE ty,tx:PRINT y$€
NEXT:NEXT€
NEXT Round €
€
Gohome:€
LINE (240,70)-(362,100),2,bf€
LOCATE 11,32:PRINT " Play again? "€
text$=Who$(ABS(SC(1,8)>SC(0,8)))€
text$=text$+" wins this game.."€
text$=text$+"How about another?"€
SAY TRANSLATE$(text$)€
FOR j=0 TO 10:x$=INKEY$:NEXT€
Again:€
x$=INKEY$:IF x$="" THEN Again€
SAY TRANSLATE$("OK.")€
IF x$="y" OR x$="Y" THEN WINDOW CLOSE 2:GOTO Restar
t€
SAY TRANSLATE$(" Bye-bye.")€
WINDOW CLOSE 2€
END€
Taketurn:€
FOR j=0 TO nb:LB(j,0)=0:NEXT:nb=1€
SAY TRANSLATE$(Who$(Who)+CHR$(46))€
PUT (140,5),Larrow:PUT (440,5),Rarrow€
FOR j=0 TO 9:x$=INKEY$:NEXT€
Getkey:€
a$=INKEY$:IF a$="-" THEN RETURN€
IF a$="+" THEN a$=STR$(INT(RND(1)*8+1))€
a=VAL(a$):IF (a<1) OR (a>8) THEN Getkey€
LB(0,0)=1€
FOR j=1 TO 3:LB(0,j)=0:NEXT€
LB(0,4)=a+3€
Moreballs:€
ex=1:FOR j=0 TO nb€
IF LB(j,0) THEN ex=0:GOSUB Moveone€
NEXT:IF ex=0 THEN Moreballs€
x=0:FOR j=13 TO 7 STEP -3:FOR k=x TO 15-x€
PUT (Column(k),Row(j)+1),Blank,AND€
NEXT:x=x+1:NEXT:RETURN€
€
Moveone:€
dy=LB(j,0):dx=LB(j,1):LY=LB(j,2)€
ny=LB(j,3):nx=LB(j,4)€
IF ny THEN€
PUT (Column(nx),Row(ny+(LY*3))+1),Blank,AND€
END IF€
LB(j,3)=(ny+1) MOD 3€
ON ny+1 GOTO Pos0,Pos1,Pos2€
```

```
€
Pos0:€
IF LY>4 THEN LB(j,0)=0:GOTO Score€
vx=0:GOSUB Whichway€
IF (SW(wx,wy,1)) AND (SW(wx,wy,0)=sd) THEN€
vx=1-2*sd:LB(j,3)=ny+1:LB(j,4)=nx+vx€
GOTO Putball€
END IF€
IF SW(wx,wy,0)=sd THEN€
LB(j,0)=0 €
SW(wx,wy,1)=1:ny=ny+1€
GOTO Putball€
END IF€
LB(j,3)=ny+1:GOTO Putball€
€
Pos1:€
LB(j,1)=0:LB(j,4)=nx+dx:GOTO Putball€
€
Pos2:€
LB(j,2)=LY+1:GOSUB Whichway€
SW(wx,wy,0)=1-SW(wx,wy,0) €
IF SW(wx,wy,1) THEN€
PUT (Column(LB(j,4)+1-sd*2),Row(ny+(LY*3))),Blank,A
ND€
LB(nb,0)=1:LB(nb,1)=0:LB(nb,2)=LY€
LB(nb,3)=0:LB(nb,4)=nx+1-sd*2:nb=nb+1€
SW(wx,wy,1)=0€
END IF€
sx=Xpos(wx,wy):sy=Ypos(wx,wy)€
wp=SW(wx,wy,0)€
'Always fall thru to switch€
€
Switch:€
PUT (sx,sy),Swblank, AND€
ON wp+1 GOTO Left,Right€
Left:€
PUT (sx,sy),Lswitch,OR:GOTO Bop€
Right:€
PUT (sx,sy),Rswitch,OR€
Bop:€
SOUND 100,1,64,1€
€
SOUND 250,1,64,3€
RETURN€
€
Putball:€
SOUND INT(RND(1)*10)*(30*LY)+200,1,64,0€
PUT (Column(nx),Row(ny+(LY*3)+1)),Ball,OR€
RETURN€
€
Whichway:€
```

```
wx=LY:wy=INT((nx+LY-4)/2):sd=(nx+LY) AND 1:RETURN
€
Score:€
sf=Points(Round,nx+1):sg=SC(Who,Round)+sf€
tx=8+63*Who+(sg>9)+(sg>99)+(sg>999)€
ty=2+Round:a$=MID$(STR$(sg),2)€
LOCATE ty,tx:PRINT a$€
SC(Who,Round)=sg€
FOR j=1600 TO 200 STEP -300€
SOUND j,1,64,0€
SOUND j+400,1,64,3€
NEXT:RETURN€
€
Values:€
FOR j=0 TO 1€
k=2+70*j:LOCATE 15,k€
PRINT SPACE$(3):LOCATE 15,k€
PRINT RIGHT$(STR$(Points(Round,0)),3)€
NEXT€
FOR j=1 TO 16:k=Points(Round,j)€
m=6+j*3.75€
IF k>9 THEN€
x=INT(k/10)€
x$=MID$(STR$(x),2,1)€
ELSE€
x$=CHR$(32)€
END IF€
LOCATE 22,m:PRINT x$;€
LOCATE 23,m:PRINT RIGHT$(STR$(k),1);€
NEXT:RETURN€
€
Setup:€
RANDOMIZE TIMER€
Greet$="Hi. Welcome to Switchbox."€
PRINT Greet$:SAY TRANSLATE$(Greet$)€
SCREEN 2,640,200,2,2€
PALETTE 0, 0, 0, 0 €
PALETTE 1, 1, 1, 1€
PALETTE 2, 0, .1, .7 €
PALETTE 3, 1, 1, .13€
WINDOW 2,"Switchbox",,0€
DIM Larrow(30),Rarrow(30),Wav%(256),Lefthunk(400)€
DIM Righthunk(400),Swblank(100),Rswitch(200)€
DIM Lswitch(200),Column(16),Row(25)€
DIM Blank(70),Ball(60),Piece(80)€
DIM SW(8,8,1),LB(32,4),Points(4,16),SC(1,8)€
FOR j=0 TO 10:LINE (0,5)-(10,j),3€
NEXT€
LINE (10,3)-(20,7),3,bf€
GET (0,0)-(20,10),Larrow€
PUT (0,0),Larrow€
```

```
FOR j=Ø TO 1Ø€
LINE (2Ø,5)-(1Ø,j),3€
NEXT€
LINE (Ø,3)-(1Ø,7),3,bf€
GET (Ø,Ø)-(2Ø,1Ø),Rarrow€
PUT (Ø,Ø),Rarrow€
GET (8,2)-(22,9),Blank€
CIRCLE (15,4),7,1€
PAINT (16,4),1€
GET (8,Ø)-(22,9),Ball€
PUT (8,Ø),Ball€
FOR j=Ø TO 127:Wav%(j)=-127€
Wav%(j+128)=127:NEXT€
FOR j=Ø TO 3:WAVE j,Wav%€
NEXT€
DATA 1Ø,"round 1. equal scores."€
DATA 2,2,2,2,2,2,2,2€
DATA 4Ø,"round 2. fibonachie seequence."€
DATA 1,2,3,5,8,13,21,34€
DATA 2Ø,"round 3. arithmetic seequence."€
DATA 2,3,4,5,6,7,8,9€
DATA 8Ø,"round 4. seequence of squares."€
DATA 1,4,9,16,25,36,49,64€
FOR j=1 TO 4:READ Points(j,Ø)€
READ Intro$(j)€
FOR k=1 TO 8:READ x€
Points(j,k+8)=x:Points(j,9-k)=x€
NEXT k:NEXT j€
a=215:b=2€
FOR j=Ø TO 4€
a=a-3Ø:b=b+3Ø€
FOR k=Ø TO j+3€
Xpos(j,k)=a+k*6Ø€
Ypos(j,k)=b€
NEXT:NEXT€
k=Ø€
FOR j=7Ø TO 52Ø STEP 3Ø€
Column(k)=j€
k=k+1:NEXT€
k=Ø€
FOR j=4 TO 154 STEP 1Ø€
Row(k)=j:k=k+1:NEXT€
€
start:€
SAY TRANSLATE$("First player's name?")€
INPUT"Name of Player 1";pØ$€
SAY TRANSLATE$("Second player's name?")€
INPUT"Name of Player 2";p1$€
Who$(Ø)=LEFT$(pØ$,6):Who$(1)=LEFT$(p1$,6)€
text$=Who$(Ø)+" plays "+Who$(1)+". Is this correct"
€
```

```
PRINT text$;:SAY TRANSLATE$(text$)←
INPUT query$:an$=LEFT$(query$,1)←
IF LEN(an$)=0 OR an$="y" OR an$="Y" THEN Draw←
GOTO start←
←
Draw:←
SAY TRANSLATE$("OK.")←
CLS←
LOCATE 1,6:PRINT Who$(0)←
LOCATE 1,66:PRINT Who$(1)←
x=4:FOR j=0 TO 1 'score boxes←
LINE (x,12)-(x+110,60),2,bf'shadow←
LINE (x+6,10)-(x+120,58),3,bf'outline←
LINE (x+16,14)-(x+110,48),0,bf'inside←
x=x+480:NEXT ←
x=1:FOR j=24 TO 50 STEP 3.7←
LOCATE 2,j:PRINT x←
x=x+1:NEXT←
LINE (180,0)-(182,40),,bf←
GET (180,0)-(182,40),Piece←
LINE (180,0)-(420,0)←
FOR j=210 TO 420 STEP 60←
LINE (j,0)-(j+2,12),,bf←
PUT (j,40),Piece←
PUT (j,100),Piece←
NEXT←
FOR j=180 TO 420 STEP 60←
PUT (j,0),Piece,OR←
PUT (j,70),Piece←
PUT (j,126),Piece←
NEXT←
PUT (120,126),Piece←
PUT (150,100),Piece←
PUT (450,100),Piece←
PUT (480,126),Piece←
ERASE Piece 'reclaim memory←
FOR j=30 TO 570 STEP 30←
LINE (j,155)-(j+2,170),1,bf←
NEXT←
LINE (176,4)-(186,32),2,bf←
LINE (416,4)-(426,32),2,bf←
LINE (176,32)-(156,42),2←
LINE STEP(0,0)-STEP(-10,0),2←
LINE STEP(0,0)-STEP(35,-32),2←
PAINT (175,31),2←
LINE (426,32)-(446,42),2←
LINE STEP(0,0)-STEP(10,0),2←
LINE STEP(0,0)-STEP(-36,-32),2←
PAINT (427,32),2←
GET (136,12)-(186,69),Lefthunk←
GET (416,12)-(456,62),Righthunk←
```

```
l=106:r=446:k=42←
FOR j=1 TO 4←
PUT (l,k),Lefthunk,OR←
PUT (r,k),Righthunk,OR←
l=l-30:r=r+30:k=k+30←
NEXT←
ERASE Lefthunk,Righthunk←
LINE (26,153)-(36,165),2,bf←
LINE (564,153)-(576,165),2,bf←
GET (245,32)-(299,40),Swblank←
FOR j=0 TO 18←
LINE (270+j,40)-(280+j,32),3←
NEXT←
LINE (245,39)-(280,40),3,bf←
GET (245,32)-(298,40),Rswitch←
PUT (184,32),Swblank,AND←
FOR j=0 TO 20←
LINE (184+j,32)-(193+j,40),3←
NEXT←
LINE (193,39)-(236,40),3,bf←
GET (184,32)-(236,40),Lswitch←
FOR m=0 TO 4:FOR n=0 TO m+3←
sx=Xpos(m,n):sy=Ypos(m,n)←
wp=INT(RND(1)*2)←
SW(m,n,0)=wp←
SW(m,n,1)=0←
Who=1-Who:GOSUB Switch←
NEXT n:NEXT m←
PUT (140,5),Larrow←
RETURN←
←
```

Amiga Puzzle

Bill Boegelein

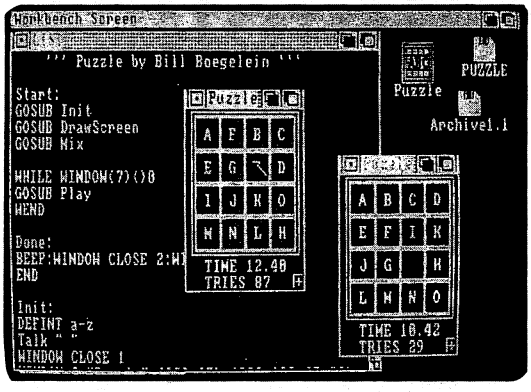
"Amiga Puzzle" provides you with an entertaining puzzle game that pops up as a small window on the Workbench screen. But more importantly, it demonstrates some interesting and powerful programming techniques in Amiga BASIC.

A popular game that used to keep kids occupied for hours in the back of the station wagon was known as the "slide puzzle." This was simply a plastic frame with 15 numbered or lettered tiles arranged in a 4 × 4 pattern with one square left vacant. The object was to slide one tile at a time into the vacant slot in an attempt to restore the puzzle to its proper numeric or alphabetic order. "Amiga Puzzle" is the Amiga BASIC equivalent of the slide puzzle.

Amiga Puzzle works on any Amiga with Amiga Microsoft BASIC. To get started, run Amiga BASIC, enter Program 1 below, and save a copy on disk. Be sure to set Preferences for a 60-column screen before running Amiga Puzzle. When you run the program, it pops open as a small window on the Workbench screen and scrambles the tiles. You can begin when the program announces "Ready" using the Amiga's built-in speech capability.

To move a tile into the vacant slot, point to the tile with the mouse and press the left mouse button. If you try to cheat by moving a tile diagonally, the program will scold you. The gadget in the lower right corner of the window, normally reserved for window resizing, has been replaced with a plus sign (+) in Amiga Puzzle. Clicking on this gadget rescrambles the puzzle and starts a new game. The other gadgets (front/back, the move bar, and close window) are all active as usual.

As a final touch, the Puzzle window displays the elapsed time and number of moves since the start of the game.



This photo shows two copies of "Amiga Puzzle" running simultaneously, an example of the Amiga's multitasking capabilities.

How It Works

Amiga Puzzle was adapted from a similar puzzle game available on the Macintosh. Those of you familiar with Microsoft BASIC 2.0 on the Macintosh will immediately see many similarities in Amiga BASIC. In fact, many Macintosh BASIC programs can be converted to Amiga BASIC with little effort. Both languages support windows, pull-down menus, labels, many types of graphics commands, and other features.

Amiga Puzzle is divided into seven subroutines named Init, DrawScreen, Mix, Play, CheckCheat, More, and Done. There are also two small subprograms, Talk and Position. Here are descriptions of what these subroutines do:

Init initializes variables and user-defined functions, and loads the puzzle's characters A–O into the two-dimensional array *c()*.

DrawScreen displays the puzzle's squares and characters.

Mix mixes only adjacent squares and keeps track of the blank square's position in variables *blankX* and *blankY*.

Play is the main subroutine where the current mouse position is compared to coordinates of each square stored in the three-dimensional array *rat*.

CheckCheat makes sure the attempted move is a legal one (adjacent squares only, no diagonal moves).

More checks whether the mouse is clicked on the plus-sign gadget in Amiga Puzzle's lower right corner. If so, the program starts a new game by jumping back to Start.

Done ends the program and returns control to BASIC.

Special Features

Most of the program is standard Microsoft BASIC. The only lines that merit special attention are the user-defined functions in the Init subroutine. One function operates the puzzle's timer, and the other determines whether the puzzle has been solved.

A quick way to time any event is to define a function that subtracts the current time from the initial starting time. This is done with the function FNlaps. The current time is obtained from TIMER and the initial time stored in the variable *starttime.*

The other defined function, FNwin, determines whether the puzzle has been solved by comparing the letter in each tile—stored in the array *c( )*—with the characters A–O. Each letter in the correct position returns a value of −1 (true). So the puzzle has been solved when all 15 letters are sorted, returning a value of −15.

A very powerful feature of Amiga BASIC is its subprogram capability. In effect, subprograms let you add new commands to the language. The word *TALK* is not a command in Amiga BASIC, but has been added to Amiga Puzzle as a subprogram. It lets you execute both a SAY command and the text-to-speech TRANSLATE$ function by simply typing TALK followed by the desired string. Similarly, POSITION is a subprogram that expects parameters of the *x* and *y* coordinates as the location to PRINT the desired string.

Multitasking with BASIC

Since multitasking is built into the Amiga's operating system as a standard feature, no special programming techniques are required to write a BASIC program that's capable of running simultaneously with other tasks. Feel free to move Amiga Puzzle over or under other windows running different programs without causing interference. If your computer has at least 512K RAM, you can even click on the Amiga Puzzle icon a second time and run two of the games at once.

Amiga Puzzle

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
←
Start:←
GOSUB Init←
```

```
GOSUB DrawScreen←
GOSUB Mix←
←
WHILE WINDOW(7)<>Ø←
GOSUB Play←
WEND←
←
Done:←
BEEP:WINDOW CLOSE 2:WINDOW 1←
END←
←
Init:←
DEFINT a-z←
Talk " " ←
WINDOW CLOSE 1←
WINDOW 2,"Puzzle",(23Ø,45)-(23Ø+138,45+96),3Ø←
tries=Ø:RANDOMIZE TIMER←
FOR y=Ø TO 3←
FOR x=Ø TO 3←
c(x,y)=x+y*4+ASC("A") 'load chars'←
NEXT x←
NEXT y←
blankX=x-1:blankY=y-1←
c(blankX,blankY)=ASC(" ")←
DEF FNlaps!=((TIMER-starttime!)\6Ø)+(((TIMER-startt
ime!) MOD 6Ø)/1ØØ)←
DEF FNa=(c(Ø,Ø)=65)+(c(1,Ø)=66)+(c(2,Ø)=67)+(c(3,Ø)
=68)←
DEF FNb=(c(Ø,1)=69)+(c(1,1)=7Ø)+(c(2,1)=71)+(c(3,1)
=72)←
DEF FNc=(c(Ø,2)=73)+(c(1,2)=74)+(c(2,2)=75)+(c(3,2)
=76)←
DEF FNd=(c(Ø,3)=77)+(c(1,3)=78)+(c(2,3)=79)+(c(3,3)
=8Ø)←
DEF FNwin=(FNa+FNb+FNc+FNd)    'won if = -15'←
RETURN←
←
DrawScreen:←
FOR y=Ø TO 3←
FOR x=Ø TO 3←
Position (x+1)*3,(y+1)*2,CHR$(c(x,y))    'print char
s'←
x1=x*3Ø+1Ø:y1=y*18+3    'draw boxes'←
LINE (x1,y1)-(x1+3Ø,y1+18),1,b←
LINE (x1-1,y1-1)-(x1+3Ø+1,y1+18+1),1,b←
LINE (1Ø-5,3-3)-(4*3Ø+1Ø+5,4*18+3+3),1,b←
moreX=128:moreY=89←
LINE (moreX,moreY)-(moreX+1Ø,moreY+1Ø),1,bf ←
Position 14,11,"+"          'new game gadget'←
rat(x,y,Ø)=x1:rat(x,y,1)=y1←
NEXT x←
```

```
NEXT y€
Position 3,10,"TIME  0.00" €
Position 3,11,"TRIES 0"€
RETURN€
€
Mix:€
x=blankX:y=blankY€
FOR mixing=333 TO 1 STEP -1 €
IF (mixing AND 1)=0 THEN€
x=INT(RND*4):y=blankY 'even'€
ELSE €
y=INT(RND*4):x=blankX 'odd'€
END IF€
GOSUB CheckCheat€
NEXT mixing€
Talk "Ready."€
starttime!=TIMER€
RETURN€
€
Play:€
LOCATE 10,8:PRINT USING "##.##";FNlaps!;€
WHILE MOUSE(0)<>0€
mouseX=MOUSE(3):mouseY=MOUSE(4)€
FOR y=0 TO 3€
FOR x=0 TO 3€
IF (mouseX>rat(x,y,0) AND mouseX<rat(x,y,0)+30) AND
(mouseY>rat(x,y,1) AND mouseY<rat(x,y,1)+18) THEN G
OSUB CheckCheat:RETURN€
NEXT x€
NEXT y€
GOSUB More€
WEND€
RETURN€
€
CheckCheat:€
IF (ABS(x-blankX)>1 OR ABS(y-blankY)>1) OR ((x<>bla
nkX AND y<>blankY)) THEN€
IF mixing=0 THEN Talk "Cheater."     'cheating'€
ELSE                                 'not cheating'€
SWAP c(x,y),c(blankX,blankY)€
Position (x+1)*3,(y+1)*2,CHR$(c(x,y))€
SWAP x,blankX:SWAP y,blankY€
Position (x+1)*3,(y+1)*2,CHR$(c(x,y))€
END IF€
IF mixing=0 THEN€
tries=tries+1€
LOCATE 11,8:PRINT tries;€
WHILE MOUSE(0)<>0:WEND€
IF FNwin=-15 THEN Talk "We have a winner.":GOTO Mor
e€
```

```
END IF←
RETURN←
←
More:←
WHILE MOUSE(Ø)<>Ø OR FNwin=-15    'another game?'←
mouseX=MOUSE(3):mouseY=MOUSE(4)←
IF MOUSE(Ø)=Ø AND (mouseX>moreX AND mouseX<moreX+1Ø
) AND (mouseY>moreY AND mouseY<moreY+1Ø) THEN GOTO
Start←
IF WINDOW(7)=Ø THEN Done←
WEND←
RETURN←
←
SUB Talk(a$) STATIC←
SAY TRANSLATE$(a$)←
END SUB←
←
SUB Position(x,y,a$) STATIC←
LOCATE y,x:PRINT a$;←
END SUB←
←
←
```

Hex War

Todd Heimarck
Translation by Philip I. Nelson

You float high above a distant planet, controlling robot armies below. Can you take control of the priceless mining turf planetside, or will your opponent's robot crews prevail? To win at this thoughtfully designed, engaging strategy game, you'll need foresight and conceptual skills rather than a quick hand on the joystick. Requires 512K memory and features synthesized speech.

"Hex War" is a two-player strategy game that can be played five different ways, and there are limitless variations. But the basic premise is always the same: You and an opponent move armies on a field of hexagons, attempting to capture territory.

The goal of the first two games is simple: capture the capital city of the other player. In game 1, the capital cities are far apart; you must devote some of your armies to defending your own capital while attempting to breach the walls of the other capital. Game 2 puts the capitals near each other, so offense and defense tend to merge in this scenario. Most of the action takes place within a small area of the battlefield.

Games 3 and 4 spread the action over a wider area. In the third game, your object is to occupy 8 of the 12 cities on the game board. Six cities occupy the periphery, and 6 are in the center of the playing field. Game 4 requires actual control of 6 cities; you must have an army in the city, one that's not involved in a battle, before you're credited with control (this version will probably take the most amount of time to play).

Although the first four scenarios encourage a commitment to battle, you employ different tactics in the fifth. The goal here is to acquire 40 of the 61 hexes, so you need some free armies to move around. As soon as you claim 40 hexes, you win the game.

Typing It In

Hex War is written in Amiga BASIC, with some important information in DATA statements. Type in the program and be sure to save a copy. Before running the program set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. After the game has been saved, and the column width set, select *Start* from the *Run* menu.

When you first run Hex War, the computer pauses to set up the screen, then displays a menu of five choices. The five different games are explained in detail below. If you're new to the game, press the 1 key to choose game 1. There will be another short pause while the variables are initialized, and then you'll see a playing field with 61 hex shapes, containing four armies on each side.

Hexes and Hexadecimal

A chess board has 64 squares arranged in a rectilinear grid. Hex War gives you a playing field of 61 hexagons (almost as many spaces as a chess board), but they're part of a six-sided honeycomb field. If you've played war games before, you may recognize the hexes.

Press the cursor-up key to go northeast, the cursor-right key to go southeast, the cursor-down key to go southwest, and the cursor-left key to go northwest. At first, the cursor movement may seem unusual. If you can turn the keyboard 45 degrees in a clockwise direction, these directions will seem more natural.

Use the space bar to select or set an army. Press ESC to end your turn before all your armies have been moved. A player indicator appears in the lower right corner to indicate which player has the current turn.

Each hex has six neighbors, so an army can move in six possible directions. To travel left and right, you'll have to press two cursor keys (for example, up and right cursor keys to move one hex to the right, which counts as one movement).

Army strengths are listed in hexadecimal (base 16) numbers, so the four armies labeled 40 actually have strengths of 64 (the hexadecimal value 40 equals 64 in our everyday decimal numbering system). At the beginning of a turn, any army has exactly three movement points. It requires one point to move an army into a neutral or enemy-controlled zone. To

move *through* the same zone also requires a point. To move into and through a friendly hex requires a total of one point. This means you can move a single army through two neutral or enemy hexes in any one turn, but the same army can move through up to three friendly zones during a turn.

Select an army by moving the cursor onto it. Press the space bar once; then position the cursor on a neighboring hex and press the space bar again. If you wish to stop, press the space bar again, and two plus signs (++) will appear, signaling that no more movement can occur. Otherwise, position the cursor on another neighboring hex and press the space bar.

Zones of Control

Each army controls the six contiguous hexes surrounding its resident hex. If you enter an enemy's zone of control, you forfeit any additional moves and must prepare for battle. In addition, an army that begins the turn in a zone of control cannot move until the battle is resolved.

Robots vs. Robots

In this game, you aren't really on the planet, but are parked high above it in a remote mothership. You've landed some robots to explore the area, and they've encountered robots belonging to another explorer. Your robots, or *bots*, as you call them, follow your orders to advance toward the other bots. Each bot has a mining laser which can stop or disable the other bots. Also, your bots have disrupter beams which can daze another bot, temporarily confusing it. When two bot-groups come close to each other, they shoot lasers and disrupters until one army of bots is disabled.

Three things can happen to a robot which suffers a hit. If the robot suffers a direct hit by a laser in its logic unit, it is vaporized. It is destroyed forever and never reappears in play.

The second thing that can happen is injury. If the laser beam is deflected, the robot is out of commission until it can be transported back to a botspital. An injured bot is frozen in place until the battle is finished, after which the victorious army carts away the injured bots to be repaired and reused.

Thus, winning a battle means you evacuate both the friendly injured and the enemy injured. After all of the injured bots recover, *they join the force in whose botspital they were*

healed. In effect, injured bots eventually become members of the army which won the battle in which they were damaged.

The third possibility is confusion: The robot is temporarily disoriented for two turns. When the time has passed, the robot is ready again.

Reprogramming Bots

Moving the cursor onto an army of robots brings up a status window in the upper left corner of the screen. The number in reverse video is unimportant; it's the army number (which may change as the game progresses).

The four numbers underneath are significant, however. The first is the army's active strength (in decimal). The second is the number of injured robots, which will be transported to the botspital of whichever side wins the battle. The third—on the line below—is the number of disrupted robots that will be available for combat in the next turn. The fourth number is the number of robots that can join the active force two turns from now.

If one side is able to reduce the other player's active force to zero, two things will happen. The winner sends all injured bots away to be repaired. The winning side also collects all enemy bots (injured or dazed) and sends them to the reinforcement center to be reprogrammed. Eventually, all these bots will be available to the winner of this particular battle for future engagements.

Reinforcements and Mergers

At the start of the game, you'll see some armies positioned outside the hex field. These are reinforcements and reserves in transit to the battle. Player 1's reinforcements enter at the bottom right corner; player 2's enter at the top left. The line of new armies moves counterclockwise; the army next to the entry point is the next to enter the battlefield.

However, the reinforcements cannot enter the battlefield if an army (friendly or enemy) is blocking their way. Keep your armies off your own reinforcement point, and try to block your opponent's armies from this area if you can. If the entry hex is owned, but not occupied, by your opponent, you'll lose some reinforcements.

After completing a turn, you're credited with additional

reinforcements according to how much territory you own. Passing over a hex allows you to claim it; the hex changes color to indicate ownership. Each piece of property provides enough ore and energy to build a new robot, available for use two turns in the future. The numbers in the line of reinforcements are updated after you move to show additional robots being built.

Winning a battle also provides additional armies in the line of reinforcements. As mentioned above, a victorious army captures any dazed enemy bots, which are reprogrammed and available in three turns. At the same time, the winner evacuates injured bots of both sides. Transportation and repair take five turns for friendly bots, seven for enemy bots. The two additional turns are needed for reprogramming the opponent's forces.

If you're losing a battle, the number of injured robots (displayed in the status window) will begin to rise. Remember that if your opponent reduces your active strength to zero, he or she will capture all of your injured bots; they'll be reprogrammed and added to future reinforcements. To prevent this from happening, you're allowed to bring in a second army for merging. Simply move another army on top of the army with which you want to merge. There's just one rule: One or both of the armies must have a strength less than 32 decimal (less than 20 hex).

Turning Speech On and Off

Hex War includes synthesized speech to emphasize various events and provide information during the game. For instance, when you select an army, the computer tells you the number (in decimal) of robots in that army. If you're not familiar with hexadecimal numbers, this feature can help you learn hex notation.

Either player can turn the speech off or on at any time. Simply click the mouse button twice: A small window appears and the computer announces the current voice status. If speech was previously activated, it now shuts off, and vice versa. Click the mouse button once to erase the window and resume play. A similar window appears to announce the outcome at the end of each game.

Hex War

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
' Hex War for 512K Amiga←
←
CLEAR ,25000←
CLEAR ,65536&←
←
Restart:←
GOSUB Setup←
←
mainloop:←
GOSUB Reveille←
GOSUB DrawField←
GOSUB PlaceTroops←
GOSUB TakeTurn←
CLS:talk$="Thinking"←
LOCATE 12,17:PRINT talk$←
GOSUB talk←
GOSUB Battle←
GOSUB Resolve←
GOSUB Battle←
GOSUB Prisoners←
GOSUB Reinforcements←
GOSUB Outcome←
pn=1-pn:ft=0:pp=0←
GOTO mainloop←
←
DrawField:←
CLS←
FOR r=11 TO 1 STEP -2←
FOR c=12-r TO r+26 STEP 4←
PUT (c*8,r*8),s10←
NEXT c,r←
FOR r=13 TO 21 STEP 2←
FOR c=r-10 TO 49-r STEP 4←
PUT (c*8,r*8),s10←
NEXT c,r←
FOR r=12 TO 1 STEP -2←
FOR c=14-r TO r+28 STEP r+28-(14-r)-1←
LOCATE r,c←
PRINT CHR$(32)←
NEXT c,r←
FOR r=13 TO 21 STEP 2←
FOR c=r-11 TO 53-r STEP 53-r-(r-11)-1←
LOCATE r,c←
PRINT CHR$(32)←
NEXT c,r←
FOR j=2 TO 23 STEP 21←
LOCATE j,12←
```

```
PRINT SPACE$(19)€
NEXT€
FOR j=1 TO 12€
GOSUB 710€
NEXT€
j=1€
IF gn=1 THEN GOSUB 718:j=2:GOSUB 715€
IF gn=2 THEN GOSUB 718:j=3:GOSUB 715€
LOCATE 1,1€
WHILE INKEY$<>"":WEND€
RETURN€
710 k=cit(j,0)€
l=cit(j,1)€
x=(k-1)*2+19€
y=(12-(k+1))*2+3€
PUT (x*8+3,y*8+3),s2,PSET€
RETURN€
715 k=cit(j,0)€
l=cit(j,1)€
x=(k-1)*2+19€
y=(12-(k+1))*2+3€
PUT (x*8,y*8),s3,PSET€
RETURN€
718 k=cit(j,0)€
l=cit(j,1)€
x=(k-1)*2+19€
y=(12-(k+1))*2+3€
PUT (x*8,y*8),s4,PSET€
RETURN€
€
TakeTurn:€
IF nx(pn)<2 THEN RETURN€
ht=4:hb=4:GOSUB 1000€
810 mv=0:ct=0€
cb=0:pk=0:k=0€
FOR j=1 TO nx(pn)-1€
IF army(j,0,pn)>0 AND army(j,6,pn)<1 THEN€
k=1€
j=nx(pn)-1€
END IF€
NEXT j€
IF k=0 THEN RETURN€
CheckIt:€
IF a$=CHR$(27) THEN a$="":RETURN€
ReadMouse:€
IF MOUSE(0)<>2 THEN NoFlag€
' left button clicked twice€
WINDOW 4,"Speech",(65,70)-(225,110),16,1€
IF TalkFlag=1 THEN€
talk$="Now I can talk."€
PRINT talk$€
```

```
TalkFlag=1-TalkFlag€
GOSUB talk€
GOTO ClearMouse€
END IF€
IF TalkFlag=Ø THEN€
talk$="OK, I'll be quiet."€
PRINT talk$€
GOSUB talk€
TalkFlag=1-TalkFlag€
END IF€
ClearMouse:€
WHILE MOUSE(Ø)<>Ø:WEND€
PRINT "Press button once"€
PRINT "to continue..."€
' wait for one click€
WHILE MOUSE(Ø)<>1:WEND€
' purge keyboard, too€
WHILE INKEY$<>"":WEND€
WINDOW CLOSE 4€
NoFlag:€
qv=Ø:a$=INKEY$:IF a$="" THEN ReadMouse€
IF UCASE$(a$)="Q" THEN€
GetOut:€
WINDOW CLOSE 3€
SCREEN CLOSE 1€
WINDOW 1,"Hex War",,31,-1€
WINDOW OUTPUT 1€
CLEAR ,25ØØØ€
END€
END IF€
IF a$=CHR$(3Ø) THEN qv=3:GOTO Codeit€
IF a$=CHR$(31) THEN qv=7:GOTO Codeit€
IF a$=CHR$(28) THEN qv=1:GOTO Codeit€
IF a$=CHR$(29) THEN qv=5€
Codeit:€
j=qv-128*(a$=" ")€
IF j=Ø THEN CheckIt€
IF (j AND 128) THEN 11ØØ€
IF (j AND 1)=Ø THEN CheckIt€
j=(j-1)/2€
IF j AND 1 THEN b1=hb+j-2:t1=ht ELSE t1=ht+1-j:b1=h
b€
IF t1<Ø OR t1>8 THEN CheckIt€
IF b1<Ø OR b1>8 THEN CheckIt€
s1=t1+b1:IF s1<4 OR s1>12 THEN CheckIt€
hb=b1:ht=t1:GOSUB 1ØØØ€
LOCATE 1,1€
FOR z=1 TO 6€
PRINT SPACE$(8)€
NEXT€
qn=map(ht,hb,Ø)€
```

```
IF qn=Ø THEN LOCATE 1,1:GOTO CheckIt ELSE q1=map(ht
,hb,1)-1€
LOCATE 1,1€
PRINT USING "####";qn:PRINT"------"€
FOR j=Ø TO 3€
PRINT USING "####";army(qn,j,q1)€
NEXT€
LOCATE 1,1€
GOTO CheckIt€
€
1ØØØ sx=146+16*(ht-hb)€
sy=21Ø-16*(ht+hb)€
IF map(ht,hb,2)=1 THEN€
PUT (ox,oy),s8€
PUT (sx,sy),s7€
ox=sx:oy=sy€
pp=1€
RETURN€
END IF€
IF pp THEN€
PUT (ox,oy),s7€
PUT (sx,sy),s8€
ox=sx:oy=sy€
pp=Ø€
RETURN€
END IF€
IF ft THEN€
PUT (ox,oy),s8€
PUT (sx,sy),s8€
ox=sx:oy=sy€
RETURN€
END IF€
PUT (sx,sy),s8€
ox=sx:oy=sy:ft=1€
RETURN€
€
11ØØ IF pk=1 THEN 12ØØ€
IF map(ht,hb,1)<>pn+1 OR map(ht,hb,Ø)=Ø THEN 81Ø€
an=map(ht,hb,Ø)€
IF army(an,6,pn)<>Ø THEN 81Ø€
pk=1:ct=ht:cb=hb€
cs=army(an,Ø,pn):SOUND 11ØØ,1Ø€
talk$=STR$(cs)+CHR$(32)+"roahbohts.":GOSUB talk€
GOTO CheckIt€
€
12ØØ j=(ht=ct) AND (hb=cb)€
IF j AND mv=Ø THEN 81Ø€
IF j AND mv>Ø THEN 142Ø€
ax=army(map(ht,hb,Ø),Ø,pn)€
IF (ax>me AND cs>me) OR (map(ht,hb,1)-1=1-pn AND ax
```

```
>Ø) THEN CheckIt←
dt=ABS(ct-ht)←
db=ABS(cb-hb)←
tl=db+dt←
IF NOT (tl=1 OR (ct+cb=ht+hb AND dt=1)) THEN CheckI
t←
mg=map(ht,hb,Ø)←
IF mg=Ø THEN 13ØØ←
FOR j=Ø TO 3←
army(mg,j,pn)=army(mg,j,pn)+army(an,j,pn)←
army(an,j,pn)=Ø←
NEXT←
army(mg,6,pn)=1←
map(ct,cb,Ø)=Ø←
cs=army(mg,Ø,pn)←
an=mg:mv=mm+1←
GOTO 138Ø←
←
13ØØ n8=map(ht,hb,1)-1←
mv=mv+1←
IF n8<>pn THEN mv=mv+1←
map(ct,cb,Ø)=Ø←
map(ht,hb,Ø)=an←
map(ht,hb,1)=pn+1←
army(an,4,pn)=ht←
army(an,5,pn)=hb←
IF mv>=mm THEN army(an,6,pn)=1←
k=Ø←
FOR j=-1 TO 1 STEP 2←
j1=ht+j:j2=hb+j:j3=hb-j←
IF j1<Ø OR j1>8 THEN 134Ø←
IF map(j1,hb,Ø)>Ø THEN←
IF map(j1,hb,1)=2-pn THEN←
k=1:j=1:GOTO 136Ø←
END IF←
END IF←
134Ø IF j2<Ø OR j2>8 THEN 135Ø←
IF map(ht,j2,Ø)>Ø THEN←
IF map(ht,j2,1)=2-pn THEN←
k=1:j=1:GOTO 136Ø←
END IF←
END IF←
135Ø IF j3<Ø OR j3>8 OR j1<Ø OR j1>8 THEN 136Ø←
IF map(j1,j3,Ø)>Ø THEN←
IF map(j1,j3,1)=2-pn THEN k=1:j=1←
END IF←
136Ø NEXT j←
IF k=1 THEN army(an,6,pn)=1:mv=mm+1←
138Ø a=pn:j=ct←
k=cb:c=Ø:d=Ø←
```

```
GOSUB 1830€
j=ht:k=hb€
c=cs:d=army(an,6,pn)€
GOSUB 1830€
ct=ht:cb=hb€
IF mv<mm THEN CheckIt€
1420 army(an,6,pn)=1€
j=ht:k=hb€
c=cs:d=1€
GOSUB 1830€
GOTO 810€
€
€
1500 RESTORE Strengths€
FOR j=0 TO 1€
nx(j)=5€
FOR k=1 TO 4€
READ a,b,c€
army(k,0,j)=a€
army(k,4,j)=b€
army(k,5,j)=c€
map(b,c,0)=k€
map(b,c,1)=j+1€
NEXT k,j€
Strengths:€
DATA 64,2,8,64,3,7,64,5,6,64,6,6€
DATA 64,2,2,64,3,2,64,5,1,64,6,0€
FOR j=0 TO 1€
FOR k=0 TO 20€
a=INT(RND(1)*k*3)€
FOR l=1 TO 5€
a=a+INT(RND(1)*21-8)€
NEXT l€
IF a<16 THEN a=0 ELSE a=(a+k*8) AND 254€
fq(k,j)=a€
NEXT k,j€
RETURN€
€
PlaceTroops:€
FOR j=0 TO 8€
FOR k=0 TO 8€
a=map(j,k,1)€
IF a THEN a=a-1:GOSUB 1800€
NEXT k,j€
FOR a=0 TO 1€
e=13+a*12:f=a*22€
dx=2-4*a:d=0€
FOR j=0 TO 8€
c=fq(j,a):GOSUB 1840€
e=e+dx*2€
IF j>3 THEN f=f+dx:e=e-dx€
```

```
NEXT j,a€
IF pn THEN€
PUT (280,160),s5,PSET€
ELSE€
PUT (280,160),s6,PSET€
END IF€
RETURN€
€
1800 b=map(j,k,0)€
c=army(b,0,a)€
d=army(b,6,a)€
1830 e=(j-k+10)*2-1€
f=(13-j-k)*2+1€
1840 IF a THEN€
PUT (e*8+1,f*8),s5,PSET€
LOCATE f+a+1,e+1€
PRINT SPACE$(2);€
GOTO 1850€
END IF€
PUT (e*8+1,(f+1)*8),s6,PSET€
LOCATE f+a+1,e+1€
PRINT SPACE$(2);€
1850 IF c=0 THEN RETURN€
LOCATE f+a+1,e+1€
PRINT RIGHT$("0"+HEX$(c),2);€
IF d AND a=0 THEN€
LOCATE f+2,e+1€
PRINT SPACE$(2);€
PUT (e*8+1,(f+1)*8+1),s11€
RETURN€
END IF€
IF d AND a=1 THEN€
LOCATE f+1,e+1€
PRINT SPACE$(2);€
PUT (e*8+1,f*8+1),s9€
END IF€
RETURN€
€
Reveille:€
sw=0:e=nx(pn)-1€
IF e<1 THEN RETURN€
FOR j=1 TO e-1€
IF army(j,0,pn)>=1 THEN 1970€
t=army(j,4,pn)€
b=army(j,5,pn)€
IF map(t,b,0)=j THEN map(t,b,0)=0€
FOR k=j TO e€
FOR l=0 TO 6€
army(k,l,pn)=army(k+1,l,pn)€
army(k+1,l,pn)=0€
NEXT l€
```

```
t=army(k,4,pn)€
b=army(k,5,pn)€
map(t,b,Ø)=k€
NEXT k€
nx(pn)=nx(pn)-1€
j=e: sw=1€
1970 NEXT j€
IF sw THEN Reveille€
FOR j=1 TO e€
army(j,Ø,pn)=army(j,Ø,pn)+army(j,2,pn)€
army(j,2,pn)=army(j,3,pn)€
army(j,3,pn)=Ø€
army(j,6,pn)=Ø€
NEXT j€
k=nx(1-pn)€
FOR j=1 TO k€
army(j,6,1-pn)=Ø€
NEXT€
GOSUB 2400€
2050 IF bp>Ø THEN€
FOR j=Ø TO 1€
FOR k=1 TO bp€
a=BTL(k,j,Ø)€
army(a,6,j)=army(a,6,j)+1€
NEXT€
NEXT€
END IF€
RETURN€
€
Battle:€
GOSUB 2400€
a=nx(Ø)€
IF nx(1)>a THEN a=nx(1)€
FOR j=Ø TO 1€
FOR k=1 TO a€
army(k,6,j)=Ø€
NEXT k,j€
GOSUB 2050€
RETURN€
€
Prisoners:€
FOR j=Ø TO 1€
a=1-j€
b=nx(j)-1€
FOR k=1 TO b€
IF army(k,Ø,j)>=1 THEN 2280€
fq(2,a)=fq(2,a)+army(k,2,j)+army(k,3,j)€
IF fq(2,a)>255 THEN€
c=fq(2,a)-255€
fq(3,a)=fq(3,a)+c€
fq(2,a)=255€
```

```
END IF€
fq(6,a)=fq(6,a)+army(k,1,j)€
IF fq(6,a)>255 THEN€
c=fq(6,a)-255€
fq(7,a)=fq(7,a)+c€
fq(6,a)=255€
END IF€
IF map(army(k,4,j),army(k,5,j),Ø)=k AND map(army(k,
4,j),army(k,5,j),1)=j+1 THEN€
map(army(k,4,j),army(k,5,j),Ø)=Ø€
END IF€
FOR l=Ø TO 6€
army(k,l,j)=Ø€
NEXT€
2280 IF army(k,6,j)>=1 THEN 2320€
fq(4,j)=fq(4,j)+army(k,1,j)€
army(k,1,j)=Ø€
IF fq(4,j)>255 THEN€
c=fq(4,j)-255€
fq(5,j)=fq(5,j)+c€
fq(4,j)=255€
END IF€
2320 NEXT k,j€
RETURN€
€
2400 bp=Ø€
FOR j=Ø TO 8€
j1=(j-4)*(4-j>Ø)€
j2=8-(j>4)*(4-j)€
FOR k=j1 TO j2€
a=map(j,k,Ø)€
r=map(j,k,1)€
IF a=Ø OR r=Ø THEN 2490€
IF army(a,Ø,r-1)<1 THEN 2490€
t=j+1€
b=k: GOSUB 2500€
b=b-1: GOSUB 2500€
t=t-1: GOSUB 2500€
2490 NEXT k,j€
RETURN€
€
2500 IF t<Ø OR b<Ø OR t>8 OR b>8 THEN RETURN€
pa=map(t,b,Ø)€
IF pa=Ø THEN RETURN€
IF map(t,b,1)=r THEN RETURN€
IF army(pa,Ø,2-r)<1 THEN RETURN€
bp=bp+1€
BTL(bp,r-1,Ø)=a€
BTL(bp,2-r,Ø)=pa€
RETURN€
€
```

```
Resolve:€
IF bp=Ø THEN RETURN€
FOR j=1 TO bp€
FOR k=Ø TO 1€
a=1-k€
an=BTL(j,k,Ø)€
ax=army(an,Ø,k)€
ht=army(an,6,k)€
ct=INT(ax/ht)+1€
BTL(j,a,1)=INT(ct*ka+1)€
BTL(j,a,2)=INT(ct*kb+1)€
BTL(j,a,3)=INT(ct*kc+1)€
NEXT k,j€
FOR j=1 TO bp€
jØ=BTL(j,Ø,Ø)€
j1=BTL(j,1,Ø)€
GOSUB 31ØØ€
army(jØ,Ø,Ø)=army(jØ,Ø,Ø)-a*BTL(j,Ø,1)€
army(j1,Ø,1)=army(j1,Ø,1)-b*BTL(j,1,1)€
GOSUB 31ØØ€
c=a*BTL(j,Ø,2)€
army(jØ,Ø,Ø)=army(jØ,Ø,Ø)-c€
army(jØ,1,Ø)=army(jØ,1,Ø)+c€
c=b*BTL(j,1,2)€
army(j1,Ø,1)=army(j1,Ø,1)-c€
army(j1,1,1)=army(j1,1,1)+c€
GOSUB 31ØØ€
c=a*BTL(j,Ø,3)€
army(jØ,Ø,Ø)=army(jØ,Ø,Ø)-c€
army(jØ,3,Ø)=army(jØ,3,Ø)+c€
c=b*BTL(j,1,3)€
army(j1,Ø,1)=army(j1,Ø,1)-c€
army(j1,3,1)=army(j1,3,1)+c€
NEXT j€
RETURN€
€
Reinforcements:€
a=1-pn:b=Ø€
FOR j=Ø TO 8:FOR k=Ø TO 8€
IF map(j,k,1)=pn+1 THEN b=b+1€
NEXT k,j€
fq(1,pn)=fq(1,pn)+b€
IF fq(1,pn)>255 THEN€
b=fq(1,pn)-255€
fq(2,pn)=fq(2,pn)+b€
fq(1,pn)=255€
END IF€
t=4:b=pn*8€
IF map(t,b,Ø)<>Ø THEN RETURN€
IF map(t,b,1)=pn+1 THEN€
fq(Ø,a)=Ø:fq(1,a)=Ø€
```

```
GOTO 3060€
END IF€
j=nx(a)€
IF j>31 THEN RETURN€
j1=fq(0,a)€
IF j1<1 THEN 3060€
nx(a)=nx(a)+1€
map(t,b,0)=j€
map(t,b,1)=a+1€
army(j,0,a)=j1€
FOR k=1 TO 3€
army(j,k,a)=0€
NEXT k€
army(j,4,a)=t€
army(j,5,a)=b€
3060 FOR k=0 TO 19€
fq(k,a)=fq(k+1,a)€
NEXT k€
fq(20,a)=0€
RETURN€
€
3100 a=0:FOR m=1 TO 6€
IF RND(1)<.5 THEN a=a+1€
NEXT m:b=6-a€
RETURN€
€
3200 talk$="press 1 through 5 to choose seenaireeo.
"€
GOSUB talk€
WINDOW 4,"Scenario: Press 1-5",(65,70)-(255,120),16
,1€
PRINT "1> Capture capital/far"€
PRINT "2> Capture capital/near"€
PRINT "3> Occupy  8/12 cities"€
PRINT "4> Control 6/12 cities"€
PRINT "5> Occupy  40/61 hexes"€
GrabKey:€
a$=INKEY$:IF a$="" THEN GrabKey€
gn=VAL(a$)€
IF gn<1 OR gn>5 THEN GrabKey€
WINDOW CLOSE 4€
talk$="seenaireeo"+STR$(gn)+CHR$(46):GOSUB talk€
RETURN€
€
Outcome:€
a=0:ON gn GOSUB 3430,3450,3480,3490,3580€
IF a=0 THEN RETURN€
en$=c$:ea=a€
GOSUB DrawField€
GOSUB PlaceTroops€
a=ea€
```

```
WINDOW 4,"Outcome",(25,70)-(300,120),16,1€
PRINT "Player "a" wins"€
MaybeOut:€
PRINT c$€
PRINT "Press Q to quit, RETURN to play."€
a$=""€
WHILE a$=""€
a$=INKEY$€
WEND€
WINDOW CLOSE 4€
IF UCASE$(a$)="Q" THEN GetOut€
WINDOW CLOSE 2:WINDOW CLOSE 1€
CLEAR ,25000€
RUN€
€
3430 IF map(cit(2,0),cit(2,1),1)=1 THEN€
a=2€
c$="Red captured the capital"€
GOSUB Announce€
RETURN€
END IF€
GOTO 3460€
€
3450 IF map(cit(3,0),cit(3,1),1)=1 THEN€
a=2€
c$="Red captured the capital"€
GOSUB Announce€
RETURN€
END IF€
€
3460 IF map(cit(1,0),cit(1,1),1)=2 THEN€
a=1€
c$="Yellow captured the capital"€
GOSUB Announce€
RETURN€
END IF€
RETURN€
€
3480 l=8:GOTO 3500€
3490 l=6€
3500 c(1)=0:c(2)=0€
FOR j=1 TO 12:t=cit(j,0):b=cit(j,1)€
r=map(t,b,1):c(r)=c(r)+1€
IF gn=4 THEN€
an=map(t,b,0)€
IF r>0 THEN IF an=0 OR army(an,6,r-1)>0 THEN c(r)=c
(r)-1€
END IF€
NEXT j€
IF c(1)=>1 THEN€
a=2€
```

```
c$="Red captured"+STR$(c(1))+" cities"€
GOSUB Announce€
RETURN€
END IF€
IF c(2)=>1 THEN€
a=1€
c$="Yellow captured"+STR$(c(2))+" cities"€
GOSUB Announce€
END IF€
RETURN€
€
3580 c(1)=0:c(2)=0€
FOR j=0 TO 8:FOR k=0 TO 8€
r=map(j,k,1):c(r)=c(r)+1€
NEXT k,j€
IF c(1)=>40 THEN€
a=2€
c$="Red occupies"+STR$(c(1))+" hexes"€
talk$="REHD AA4KYUWPAYZ":SAY talk$€
talk$=STR$(c(1))+"hehxes"€
GOSUB talk:RETURN€
END IF€
IF c(2)=>40 THEN€
a=1€
c$="Yellow occupies"+STR$(c(2))+" hexes"€
talk$="YEHLOH AA4KYUWPAYZ":SAY talk$€
talk$=STR$(c(2))+"hehxes"€
GOSUB talk:RETURN€
END IF€
RETURN€
€
Setup:€
DEFINT s€
SCREEN 1,320,200,2,1€
' open window 3 with no €
' gadgets or title bar€
WINDOW 1,"",(0,0)-(311,25),16,1€
WINDOW 3,"",(0,0)-(311,185),16,1€
WINDOW OUTPUT 3€
PALETTE 0,0,0,0€
PALETTE 1,.5,1,1€
PALETTE 2,1,0,0€
PALETTE 3,1,1,.1€
WIDTH 40€
CLS€
DIM Voice%(8)€
RESTORE VoiceData€
FOR j=0 TO 8€
READ Voice%(j)€
NEXT€
RESTORE€
```

```
' speech will be synchronous€
VoiceData:€
DATA 110,0,170,0,22200,64,10,1,0€
talk$="Welcome to Hex War."€
LOCATE 13,11€
PRINT talk$€
GOSUB talk€
Temp$="Click button twice to turn"€
LOCATE 15,8:PRINT Temp$€
Demp$=" speech off or on during game."€
LOCATE 16,6:PRINT Demp$€
talk$=Temp$+Demp$€
GOSUB talk€
' hex shape€
LINE (0,0)-(2,0):LINE (13,0)-(15,0)€
LINE (0,1)-(3,1):LINE (12,1)-(15,1)€
LINE (3,2)-(12,2):LINE (4,3)-(11,3)€
FOR j=4 TO 11€
LINE (6,j)-(9,j)€
NEXT€
LINE (4,12)-(11,12):LINE (3,13)-(12,13)€
LINE (0,14)-(3,14):LINE (12,14)-(15,14)€
LINE (0,15)-(2,15):LINE (13,15)-(15,15)€
DIM s10(225)€
GET (0,0)-(15,15),s10€
PUT (0,0),s10€
' cursor shape€
FOR k=0 TO 1€
GOSUB Bracket€
PAINT (32,42),2+k,1€
LINE (30,35)-(48,44),0,bf€
LINE (32,32)-(47,47),0,bf€
GOSUB Bracket€
IF k=0 THEN€
DIM s8(400)€
GET (26,26)-(53,49),s8€
PUT (26,26),s8€
END IF€
IF k=1 THEN€
DIM s7(400)€
GET (26,26)-(53,49),s7€
PUT (26,26),s7€
END IF€
NEXT k€
GOTO Blip€
Bracket:€
PUT (16,32),s10€
PUT (32,48),s10€
PUT (48,32),s10€
PUT (32,16),s10€
RETURN€
```

```
Blip:€
' city shape€
FOR j=Ø TO 1€
LINE (2,j)-(7,j)€
LINE (2,j+8)-(7,j+8)€
LINE (j,2)-(j,7)€
LINE (j+8,2)-(j+8,7)€
NEXT€
PSET (1,1):PSET (8,1)€
PSET (1,8):PSET (8,8)€
DIM s2(1ØØ)€
GET (Ø,Ø)-(9,9),s2€
PUT (Ø,Ø),s2€
' capital shape€
FOR k=Ø TO 1€
FOR j=Ø TO 3€
LINE (4,j)-(11,j),2+k€
LINE (4,j+12)-(11,j+12),2+k€
LINE (1,4+j)-(14,4+j),2+k€
LINE (1,8+j)-(14,8+j),2+k€
NEXT€
LINE (3,6)-(5,9),Ø,bf€
LINE (6,3)-(9,5),Ø,bf€
LINE (6,1Ø)-(9,12),Ø,bf€
LINE (1Ø,6)-(12,9),Ø,bf€
IF k=1 THEN€
DIM s3(225)€
GET (Ø,Ø)-(15,15),s3€
PUT (Ø,Ø),s3€
END IF€
IF k=Ø THEN€
DIM s4(225)€
GET (Ø,Ø)-(15,15),s4€
PUT (Ø,Ø),s4€
END IF€
NEXT k€
'army shape€
FOR j=Ø TO 14€
LINE (7,Ø)-(j,7),3€
NEXT€
FOR j=4 TO 1Ø€
LINE (7,2)-(j,5),Ø€
NEXT€
DIM s5(64)€
GET (Ø,Ø)-(14,7),s5€
PUT (Ø,Ø),s5€
' other army shape€
FOR j=Ø TO 14€
LINE (7,7)-(j,Ø),2€
NEXT€
FOR j=4 TO 1Ø€
```

```
LINE (7,5)-(j,2),Ø€
NEXT€
DIM s6(64)€
GET (Ø,Ø)-(14,7),s6€
PUT (Ø,Ø),s6€
' crosses                          €
FOR k=Ø TO 1€
FOR j=Ø TO 1€
LINE (Ø,2+j)-(13,2+j),2+k€
LINE (10+j,Ø)-(10+j,5),2+k€
LINE (2+j,Ø)-(2+j,5),2+k€
NEXT€
FOR j=Ø TO 1€
LINE (6+j,Ø)-(6+j,5),Ø€
NEXT€
IF k=Ø THEN€
DIM s11(150)€
GET (Ø,Ø)-(13,13),s11€
PUT (Ø,Ø),s11€
END IF€
IF k=1 THEN€
DIM s9(150)€
GET (Ø,Ø)-(13,13),s9€
PUT (Ø,Ø),s9€
END IF€
NEXT k€
DIM army(31,6,1),BTL(64,1,3)€
DIM map(9,9,2),fq(20,1),nx(1),c(2)€
cn=12:DIM cit(cn,1)€
RANDOMIZE TIMER€
€
pn=1:me=31€
mm=3 ' Maximum number of moves€
ka=1/48:kb=1/48:kc=1/32€
RESTORE Whatsit€
FOR j=1 TO cn€
FOR k=Ø TO 1€
READ cit(j,k)€
NEXT€
map(cit(j,Ø),cit(j,1),2)=1€
NEXT€
Whatsit:€
DATA 8,4,Ø,4,8,Ø,Ø,8,4,Ø,4,8€
DATA 5,5,3,3,6,3,2,5,5,2,3,6€
CLS€
GOSUB 32ØØ€
CLS€
GOSUB 15ØØ€
RETURN€
€
```

```
Announce:€
talk$=c$€
€
talk:€
IF TalkFlag=Ø THEN SAY TRANSLATE$(talk$),Voice%€
RETURN€
€
```

Chain Reaction

Mark Tuttle
Translation by Tim Midkiff

In this explosive strategy game, the contest is never finished until your last bomb has been thrown. Requires 512K of memory.

"Chain Reaction" is a clever strategy game for one or two players. Whether you play against the computer or another human, the objective is the same: to eliminate all of your opponent's bomb-shaped pieces from the field of play. The game is played on a 5 × 6 grid of squares, and the players alternate turns, placing one bomb in a square on each turn.

The results of a move depend on how many bombs are already in the chosen square and adjacent squares. Whenever any square reaches "critical mass," it explodes and sends its bombs into neighboring squares. If those squares are already loaded to capacity, they explode too, creating a chain reaction that can engulf a large area of the board.

Type in the program and save a copy of it. Before running the program set the screen to 80 columns by clicking on the Preferences icon and setting the proper width. Select *Start* from the *Run* menu to begin. Use the cursor keys to control movement and the space bar to place a bomb.

Bomb Begets Bomb

When you run Chain Reaction, it begins by asking whether you wish to play with one or two players. If you've never played before, you may want to play a game or two against the computer to learn what sort of strategies it favors. When you choose to play against the computer, the program also asks whether you'd like the computer to take the first turn.

The first part of most games involves placement of initial pieces, without many explosions. As the board fills up, however, explosions occur with increasing frequency. Play continues until one player's pieces are completely eliminated from the board.

The position of a square in the grid determines how many bombs it requires to create an explosion. A corner square can hold a maximum of one bomb. When you place a second bomb in a corner square that already holds one, both bombs explode, sending a bomb of your color into two neighboring squares. After an explosion, the original square is emptied.

Other squares require more bombs to create an explosion. A border square that isn't on a corner can hold a maximum of two bombs. When you place a third bomb in a border square, its explosion sends three bombs into the squares that adjoin it. Squares in the center of the game board hold the most bombs and also create the most devastating explosions. When you place a fourth bomb in a central square, it sends four bombs into squares which adjoin that position.

When an explosion sends bombs into adjacent squares, any bombs in that square change color to match the color of the exploding bombs. Should one of the adjoining squares surpass its limit, that square, too, will explode, creating the potential for even more explosions. This process continues until no more explosions are possible.

Thus, the situation in Chain Reaction is often volatile. The lead frequently seesaws back and forth between players, as each creates increasingly more widespread chain reactions. Even if defeat seems almost certain, you can often regain the lead with clever play. When a game ends, the program announces the winner and permits you to play a new game or quit.

Like other games of strategy and placement, Chain Reaction rewards the player who can think ahead. At first, you may be tempted to start making explosions as quickly as possible. But that's not always the best long-term tactic. By spreading bombs of your color throughout the board, you may be able to survive chain reactions that would otherwise wipe you out.

Chain Reaction

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
DEFINT a-z:DEFSNG r,g,b←
SCREEN 1,320,200,3,1:WINDOW 3,"",(0,0)-(311,186),16
,1:WINDOW OUTPUT 3:COLOR 3,0←
DIM s(42,1),u(4,5),p(4,5),j(4,5),bx(4,5),by(4,5),n(
1,5),er(528),w1(255),w2(255),rt(30),tr(30),tx(30),t
y(30)←
```

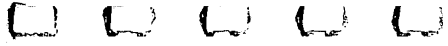
```
FOR i=Ø TO 255:w1(i)=RND*255-128:w2(i)=RND*255-128:
NEXT←
RESTORE PaletteData:FOR i=Ø TO 7:READ r,g,b:PALETTE
i,r,g,b:NEXT←
PaletteData: ←
DATA Ø,Ø,.7,Ø,Ø,Ø,.8,.8,Ø,.7,.7,.7,.33,.87,Ø,.9,.9,
.9,.6,Ø,Ø,Ø,.6,Ø←
WIDTH 40:CLS:RANDOMIZE TIMER←
GOSUB InitShapes:e=Ø←
FOR i=1 TO 3:FOR j=1 TO 4:j(i,j)=4:NEXT:j(i,Ø)=3:j(
i,5)=3:NEXT←
RESTORE Corners:FOR j=1 TO 4:j(Ø,j)=3:j(4,j)=3:READ
a,b:j(a,b)=2:NEXT←
Corners: DATA Ø,Ø,Ø,5,4,Ø,4,5←
RESTORE BombPos:FOR i=1 TO 4:FOR j=1 TO i:READ bx(i
,j),by(i,j):NEXT j,i←
bx(4,5)=bx(4,4):by(4,5)=by(4,4)←
BombPos: DATA 13,9,6,9,20,9,13,5,4,15,22,15,13,3,13
,17,4,9,22,9←
LOCATE 8,14:PRINT "CHAIN REACTION"←
LOCATE 12,9:PRINT "Number of players (1/2)?";←
WHILE np<>1 AND np<>2:np=VAL(INKEY$):WEND←
IF np<>2 THEN←
LOCATE 16,10:PRINT "Computer first (Y/N)?";←
WHILE k$<>"Y" AND k$<>"N":k$=UCASE$(INKEY$):WEND←
tu=ABS(k$="Y")←
END IF←
CLS:COLOR 3,1:LOCATE 1,13:PRINT " CHAIN REACTION ";
←
GOSUB DrawGrid←
←
MainLoop:←
WHILE e=Ø:tu=-tu+1:co=tu+6←
IF np=1 AND tu=Ø THEN←
GOSUB Computer←
ELSE←
GOSUB Human←
WHILE p(y,x)<>tu+1 AND p(y,x):GOSUB Human:WEND←
END IF←
u(y,x)=u(y,x)+1:FS(tu)=FS(tu)+1:IF p(y,x)=Ø THEN p(
y,x)=tu+1←
GOSUB PlaceBomb←
IF u(y,x)=j(y,x) THEN ←
CheckGrid: e=Ø:fg=Ø:FOR p=Ø TO 4:FOR q=Ø TO 5:y=p:x
=q←
IF u(y,x)>=j(y,x) AND e=Ø THEN fg=1:GOSUB FullSquar
e←
NEXT q,p:IF fg=1 AND e=Ø THEN CheckGrid←
END IF←
WEND←
←
```

```
EndGame:←
COLOR 3,1:LOCATE 24,15:PRINT " GAME OVER ";:FOR i=1
TO 10000:NEXT←
LOCATE 24,5:PRINT " Press space bar to play again.
";←
k$="":WHILE k$<>" ":k$=INKEY$:WEND←
SCREEN CLOSE 3:WINDOW CLOSE 3:RUN←
←
Human:←
WHILE INKEY$<>"":WEND:x=hx(tu):y=hy(tu):dx=0:dy=0:G
OSUB DrawCursor:k$=""←
WHILE k$<>" ":k$=INKEY$←
IF k$=CHR$(28) THEN IF y>0 THEN dy=-1←
IF k$=CHR$(29) THEN IF y<4 THEN dy=1←
IF k$=CHR$(31) THEN IF x>0 THEN dx=-1←
IF k$=CHR$(30) THEN IF x<5 THEN dx=1←
IF dx<>0 OR dy<>0 THEN←
co=0:GOSUB DrawCursor←
x=x+dx:y=y+dy:co=tu+6:GOSUB DrawCursor←
dx=0:dy=0←
END IF←
WEND:hx(tu)=x:hy(tu)=y←
co=0:GOSUB DrawCursor:co=tu+6:RETURN←
←
FullSquare:←
r=0:yy=32*y+15:xx=36*x+50:WAVE 0,w1:WAVE 1,w2←
FOR i=1 TO 4:PUT(xx+2,yy+2),er,PSET←
FOR j=1 TO 4:n(1,j)=INT(RND*3)-1:NEXT←
r=-r+1:k=u(y,x)+1:bn=co-6:IF k=6 THEN k=5←
ON k GOSUB b0,b1,b2,b3,b4,b4←
FOR m=255 TO 10 STEP-20:SOUND 100,.1,m,0←
SOUND 100,.1,m,3:FOR n=1 TO RND*20:NEXT n,m:NEXT←
tx=x:ty=y:J1=0:K1=k-1:y1=32*y+15+by(1,1):x1=36*x+50
+bx(1,1)←
IF tx>0 THEN x=tx-1:dx=-1:dy=0:GOSUB ExplodeBombs:G
OSUB AddBomb←
IF tx<5 THEN x=tx+1:dx=1:dy=0:GOSUB ExplodeBombs:GO
SUB AddBomb←
x=tx:IF ty>0 THEN y=ty-1:dy=-1:dx=0:GOSUB ExplodeBo
mbs:GOSUB AddBomb←
IF ty<4 THEN y=ty+1:dy=1:dx=0:GOSUB ExplodeBombs:GO
SUB AddBomb←
IF FS(0)<1 OR FS(1)<1 THEN e=1←
y=ty:u(y,x)=u(y,x)-j(y,x):GOSUB PlaceBomb:IF u(y,x)
=0 THEN p(y,x)=0←
RETURN←
←
AddBomb:←
IF p(y,x)<>tu+1 THEN FS(tu)=FS(tu)+u(y,x):FS(-tu+1)
=FS(-tu+1)-u(y,x)←
```

```
p(y,x)=tu+1:u(y,x)=u(y,x)+1:GOSUB PlaceBomb:RETURN←
←
DrawGrid:←
FOR y=Ø TO 4:yy=32*y+16:FOR x=Ø TO 5:xx=36*x+51←
LINE(xx,yy)-(xx+34,yy+3Ø),2,b←
NEXT x,y←
GET(xx+1,yy+1)-(xx+33,yy+29),er:RETURN←
←
DrawCursor:←
yy=32*y+15:xx=36*x+5Ø←
LINE(xx,yy)-(xx+36,yy+32),co,b←
LINE(xx+2,yy+2)-(xx+34,yy+3Ø),co,b ←
RETURN←
←
PlaceBomb:←
yy=32*y+15:xx=36*x+5Ø:r=Ø:bn=co-6←
PUT(xx+2,yy+2),er,PSET:k=u(y,x)+1:IF k=6 THEN k=5←
ON k GOTO bØ,b1,b2,b3,b4,b4←
bØ: RETURN←
b1: PUT(xx+bx(1,k-1),yy+by(1,k-1)),s(Ø,bn):RETURN←
b2: FOR j=1 TO k-1:PUT(xx+bx(2,j)+n(r,j),yy+by(2,j)
+n(r,j)),s(Ø,bn):NEXT:RETURN←
b3: FOR j=1 TO k-1:PUT(xx+bx(3,j)+n(r,j),yy+by(3,j)
+n(r,j)),s(Ø,bn):NEXT:RETURN←
b4: FOR j=1 TO k-1:PUT(xx+bx(4,j)+n(r,j),yy+by(4,j)
+n(r,j)),s(Ø,bn):NEXT:RETURN←
←
CheckNeighbor:←
fp=1:IF y>Ø THEN IF p(y-1,x)=2 THEN ay=y-1:RETURN←
IF x>Ø THEN IF p(y,x-1)=2 THEN ax=x-1:RETURN←
IF x<5 THEN IF p(y,x+1)=2 THEN ax=x+1:RETURN←
IF y<4 THEN IF p(y+1,x)=2 THEN ay=y+1:RETURN←
fp=Ø:RETURN←
←
Computer:←
xt=Ø:FOR y=Ø TO 4:FOR x=Ø TO 5←
IF p(y,x)<>2 THEN xt=xt+1:ty(xt)=y:tx(xt)=x←
NEXT x,y←
LOCATE 24,15:COLOR 3,Ø:PRINT "Thinking...";←
FOR i=1 TO xt:rt(i)=Ø:tr(i)=Ø:y=ty(i):x=tx(i):GOSUB
CheckNeighbor←
IF fg=1 AND fp AND u(y,x)>Ø THEN EndComputer←
IF u(y,x)+1=j(y,x) THEN ←
IF fp=1 AND u(ay,ax)+1=j(ay,ax) THEN rt(i)=6:GOTO C
heckNext←
IF fp=1 THEN rt(i)=2:GOTO CheckNext←
IF fp=Ø THEN rt(i)=1:GOTO CheckNext←
END IF←
IF j(y,x)=2 THEN←
IF fp=Ø AND u(y,x)=1 THEN rt(i)=1:GOTO CheckNext←
```

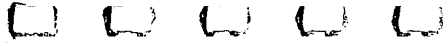
```
IF fp=Ø AND u(y,x)=Ø THEN rt(i)=4:GOTO CheckNext←
IF fp=1 AND u(y,x)=1 THEN rt(i)=4:GOTO CheckNext←
END IF←
IF u(ay,ax)+1=j(ay,ax) THEN rt(i)=1:GOTO CheckNext←
IF u(y,x)+2>=j(y,x) THEN←
IF fp=1 AND u(ay,ax)+1<j(ay,ax) THEN rt(i)=5:GOTO C
heckNext←
IF fp=Ø THEN rt(i)=3:GOTO CheckNext←
rt(i)=2:GOTO CheckNext←
END IF←
IF fp=Ø THEN rt(i)=2:GOTO CheckNext←
rt(i)=1←
CheckNext: NEXT:zt=Ø:ab=6←
WHILE zt=Ø←
FOR i=1 TO xt:IF rt(i)=ab THEN zt=zt+1:tr(zt)=i←
NEXT:ab=ab-1←
WEND←
dh=INT(zt*RND)+1:hd=tr(dh):y=ty(hd):x=tx(hd)←
EndComputer: LOCATE 24,15:PRINT "              ";:fg=f
g+1:RETURN←
←
ExplodeBombs:←
J1=J1+1:xx=x1-bx(1,1):yy=y1-by(1,1):s=1Ø87:bn=co-6←
WAVE Ø,SIN:SOUND 66Ø,.5,255←
FOR j=1 TO 5ØØ:NEXT:SOUND Ø,Ø,Ø←
PUT(xx+bx(K1,J1)+n(r,J1),yy+by(K1,J1)+n(r,J1)),s(Ø,
bn)←
IF dy=Ø THEN ←
X2=x1+35*dx:dx=dx*4:PUT(x1,y1),s(Ø,bn)←
FOR i=x1 TO X2 STEP dx:s=s-4Ø:SOUND s,1,5Ø←
PUT(i,y1),s(Ø,bn):PUT(i+dx,y1),s(Ø,bn):NEXT←
PUT(xx+3,yy+3),er←
ELSE←
Y2=y1+31*dy:dy=dy*4:PUT(x1,y1),s(Ø,bn)←
FOR i=y1 TO Y2 STEP dy:s=s-4Ø:SOUND s,1,5Ø←
PUT(x1,i),s(Ø,bn):PUT(x1,i+dy),s(Ø,bn):NEXT←
PUT(xx+3,yy+3),er←
END IF←
RETURN←
←
InitShapes:←
RESTORE RedBomb←
FOR j=Ø TO 1:FOR i=Ø TO 42←
READ a$:s(i,j)=VAL("&H"+a$):NEXT i,j:RETURN←
←
RedBomb: DATA B,D,3,2ØØ,4ØØ,4ØØ,Ø,18ØØ←
DATA 3ØØØ,1BØØ,AØØ,4ØØ,AØØ,1BØØ,Ø,Ø←
DATA Ø,Ø,Ø,EØØ,2780,4FCØ,E4EØ,F5EØ←
DATA FBEØ,F5EØ,64CØ,3F8Ø,EØØ,2ØØ,4ØØ,4ØØ←
DATA EØØ,3F8Ø,7FCØ,E4EØ,F5EØ,FBEØ,F5EØ,64CØ←
DATA 3F8Ø,EØØ,3F8Ø←
```

```
←
GreenBomb: DATA B,D,3,200,400,400,E00,3F80←
DATA 7FC0,FFE0,FFE0,FFE0,FFE0,7FC0,3F80,E00←
DATA 0,0,0,E00,2780,4FC0,E4E0,F5E0←
DATA FBE0,F5E0,64C0,3F80,E00,200,400,400←
DATA E00,3F80,7FC0,E4E0,F5E0,FBE0,F5E0,64C0←
DATA 3F80,E00,3F80←
←
←
```



CHAPTER THREE

BASIC Programming



Amiga BASIC Style

Jim Butterfield

Learn to manage custom menus and output windows, read mouse input, trap background events, and master other techniques which give Amiga BASIC its unique character. The article also highlights some of the differences between Amiga BASIC and earlier BASICs, and includes a useful program for calculating mortgages.

There's a different style to BASIC programming on the Amiga. You should take a close look at new features; you'll discover concepts that lead to a radically different style of programming and user interaction.

To illustrate some of these concepts, let's construct a simple Amiga BASIC program which analyzes the five important variables in a home mortgage: principal (amount borrowed), interest rate, period of loan, monthly payment, and balance due. Since interest-compounding schedules are different in Canada from in the United States, the program includes an option for choosing either schedule. We'll discuss elements of the program as we go through it.

Initialization

```
REM Mortgage (Version 1)←
DIM title$(6),site$(2),pudef$(5),value#(5),peryear(
2),compound(2)←
cal=4:site=1←
←
```

The REM identifies the program and version. The DIM statement defines the six arrays used in the program, which we'll discuss as we go along. Note that there are no line numbers in Amiga BASIC. They are not needed. Even with GOTO or GOSUB, it's usual to identify a line with a label, not a number. (You may include line numbers if you like—a feature included for the sake of compatibility with other BASICs—but since the line numbers are treated simply as labels, numeric

order is irrelevant.)

Also, notice that we use descriptive words for variable names. In the versions of BASIC on earlier Commodore computers, only the first two characters of the variable name were significant (HO$ and HOUSEHOLD$ would be considered the same name). In Amiga BASIC, names can be up to 40 characters long with every character significant (Householdbudget1 and Householdbudget2 are recognized as distinct names). Descriptive variable names make the program much easier to understand and reduce the need for explanatory REM statements. We also set the default value of the two variables that determine which menu items are selected. The loan variable to be calculated (*cal*) is 4, the payment amount. The default interest compounding schedule (*site*) is that for country 1, Canada. Change either of these if you wish.

```
DATA Principal,Rate,Years,Payment,Balance,Quit←
MENU 5,Ø,1,"Calculate"←
FOR j=1 TO 6:READ title$(j)←
MENU 5,j,1-(j=cal)," "+title$(j):NEXT j←
←
```

The DATA statement contains the items for the first of our custom menus, as well as the captions for the output window (the array *title$*). One of the most significant features of Amiga BASIC is that the programmer can easily construct custom menus.

We'll choose menu 5 for our first custom menu so that menus 1–4 can retain their default uses: Project, Edit, Run, and Windows. The first MENU statement sets Calculate as the title for the menu; then the FOR-NEXT loop reads the DATA items into the corresponding menu slots. Note the expression *1−(j=cal)* for the third parameter of the MENU statement in the loop. Just like earlier Commodore BASICs, Amiga BASIC interprets a true expression as −1 and a false expression as 0, so 1−(j=cal) will evaluate to 1−(−1) = 2 when the value of *j* equals the value of *cal*, and 1−(0) = 1, otherwise. A value of 2 for this parameter puts a check to the left of the menu item, so this feature is used to indicate which calculation option is currently selected. A value of 1 displays the menu item without a checkmark, but still makes it active; a value of 0 deactivates the menu item, leaving it dimmed, or *ghosted*, and impossible to select.

```
DATA Canada,2,6,USA,12,1←
MENU 6,0,1,"Country"←
FOR j=1 TO 2:READ site$(j),peryear(j),compound(j)←
MENU 6,j,1-(j=site),"  "+site$(j):NEXT j←
```

Different rules are used in the U.S. and Canada to work out a monthly interest rate based on the annual interest figure. In the U.S., the annual amount is simply divided by 12. In Canada, semiannual compounding is used, which involves dividing by 2 to get the semiannual rate and then using a more complex formula. The user will be able to pick the appropriate system from menu 6, which is titled Country. It would not be too hard to add extra menu items, such as compounding quarterly (the numeric DATA items would be 4,3). The FOR-NEXT loop here uses the same technique for flagging the current menu selection as the one above.

Format with PRINT USING

```
←
DATA "#,###,###.##"←
DATA "   ###.###% "←
DATA "   ###.###  "←
DATA "#,###,###.##"←
DATA "#,###,###.##"←
FOR j=1 TO 5:READ pudef$(j):NEXT j←
```

These are the PRINT USING templates that tell how the numeric values of the five loan variables are to be printed. The principal amount, for example, is printed as a dollars-and-cents value. The annual interest rate, in contrast, will be shown to three decimal places with a percent sign.

```
←
DATA 10000,10,10,0,0←
FOR j=1 TO 5:READ value#(j):NEXT j←
```

These are just arbitrary figures to appear on the initial screen. I've picked a principal amount of $10,000 at 10 percent over ten years. Your own default values may be substituted. Once the program is running, any of these values can easily be changed.

An important point: Note that the array into which the values are read, *value#*, has an extra symbol at the end. The # sign (pound sign, hash mark, or whatever you want to call it)

indicates that these variables are *double precision.* If you've worked with previous Commodore machines which offered only one level of numeric precision, you might be unclear about this issue. Here's the story: In earlier Commodore BASICs, variables worked to about ten digits of accuracy. That was enough—just barely enough—to do most home finance calculations. Normal (single-precision) Amiga BASIC variables—the type you usually get if you don't add a type identifier after the variable name—are reliable to only about seven digits. This means that it can't handle amounts of more than about $167,000 without losing pennies.

Computer scientists will tell you that single-precision Amiga BASIC variables have a 24-bit *mantissa,* as opposed to the 32-bit mantissa in earlier Commodore BASICs. This means that whenever you need to deal with dollars-and-cents values—or with other values requiring a high accuracy—you need to call for a double-precision variable. Such a variable will have more accuracy—enough to cover a federal budget and still be exact on the pennies. To specify double precision, add a # sign to the end of the variable name. Be careful to include the sign each time you use the variable name, however. Amiga BASIC will consider *value* and *value#* to be two different variables.

A Custom Window

```
←
WINDOW 2,"Mortgage",(10,10)-(400,100),8←
WINDOW OUTPUT 2←
GOSUB calc:GOSUB showval←
LOCATE 7,1←
PRINT "Use menu buttons to select option."←
PRINT "Click on existing values to change."←
GOSUB hang←
WINDOW CLOSE 2←
END←
```

Now we open a new window in which the calculations will appear. The only gadget we put on the window is the closing gadget (code 8). It's there so that the user can still put away the window manually in case the program is stopped. The window is not only created, but also selected for output. Then the initial calculations are displayed, along with brief instructions near the bottom of the window.

The program's main job is a subroutine called *hang*. We'll stay in that subroutine until the user wants to quit, at which time the window will be closed. Here is the *hang* subroutine:

```
←
hang:←
ON MENU GOSUB event←
ON MOUSE GOSUB event←
MOUSE ON←
MENU ON←
kwit=Ø←
WHILE kwit<>1:WEND←
MOUSE OFF←
MENU OFF←
MENU RESET←
RETURN←
```

We define an action for the mouse and for the menus we previously defined. Clicking the left mouse button or selecting a menu item invokes the *event* subroutine. These two activities are *interrupts* or *event traps*. After they are activated with MENU ON and MOUSE ON, they will remain in place, waiting for the appropriate event to happen, until they are canceled or turned off. While they are active, it doesn't matter what the program is doing; a suitable stimulus will immediately cause the program to jump to the specified subroutine.

A variable called *kwit* is used by the program to tell when it's time to quit. As long as it's 0, the program stays in the WHILE–WEND loop. How does it ever get out of this seemingly endless loop? Remember the event traps we just enabled. Pressing the left mouse button or selecting a menu item will trigger a GOSUB to the *event* routine, which in turn calls subroutines to process the button click or menu selection. One menu selection, the Quit option from the Calculate menu, will change the value of *kwit* to 1 to end the loop. After exiting the loop, we'll shut off the menu and mouse, disconnect the event traps, and return to the main program, which ties things up.

A Major Event

```
←
event:←
ms=MOUSE(Ø):mn=MENU(Ø)←
IF mn THEN GOSUB menuhit←
IF ms THEN GOSUB eek←
IF kwit=Ø THEN GOSUB calc:GOSUB showval←
RETURN←
```

Now let's look at the routine where the real action takes place. When we arrive at the *event* subroutine, we know that one of two things has happened. Either the left mouse button has been clicked or a menu item has been selected by using the right mouse button. The MOUSE and MENU functions are used to check which, and the appropriate service subroutine is called. Once the new value for *cal* or *site* has been established, we're ready to calculate new values, but first we check that *kwit* is still 0—we don't want to calculate values if the Quit option from the Calculate menu was selected. The new financial values are determined by calling the subroutine *calc*, then displayed using the *showval* subroutine. Keep in mind that we'll come back to this routine to recalculate anytime the data elements—or the rules—are changed.

```
←
calc:←
ON ERROR GOTO oops←
principal#=value#(1)←
r1#=(value#(2)/100/peryear(site)+1)^(1/compound(sit
e))←
rate#=r1#-1←
months=value#(3)*12←
payment#=value#(4)←
balance#=value#(5)←
ON cal GOSUB fprin,fintr,fper,fpay,fbal←
scale=100:IF cal=2 OR cal=3 THEN scale=1000←
value#(cal)=INT(value#(cal)*scale+.99)/scale←
ON ERROR GOTO 0←
RETURN←
```

The *calc* subroutine is where the dirty work begins. The principal, interest rate, number of periods, payment amount, and final balance are extracted from the *value#* array so that they can be used by the various calculation programs more easily. Note that in most cases, we retain double-precision accuracy with the # sign. The monthly interest rate is worked out by a fairly complex formula, and the number of months equals the number of years times 12.

The variable *cal* tells us what to calculate. Depending on its value, we'll call *fprin* (find principal), *fintr* (find interest rate), *fper* (find period), *fpay* (find payment), or *fbal* (find balance). The calculation with *scale* rounds any calculated value to the next highest penny, or, if not a money figure, to three decimal places.

The calculation subroutine also includes an error trap,

since some calculations are impossible or ridiculous (for example, how long would it take to pay off a $1,000 mortgage with a payment of $0 per month?). Problems are directed to an event trap named *oops*.

```
←
oops:←
value#(cal)=Ø←
RESUME oops2←
oops2:←
WINDOW 2←
RETURN←
←
```

If there's any calculation problem, we set the calculated value to 0 and give up. We do not go back to the detailed calculation program. Instead, using *oops2*, we return to the main *calc* routine. But first, it's necessary to reopen WINDOW 2, since the Amiga always closes any secondary windows when an error occurs. Notice that the message at the bottom of the window is not reprinted. So if you see the window blink and then reappear minus the message and with the value being calculated set to 0, an error has been trapped. If this occurs when you enter what seem to be legitimate values, it may indicate that you made an error while entering the program. For this reason you may want to omit the ON ERROR statements until you are confident that you have eliminated all typing mistakes in the program.

Here are the five calculation routines. We won't plunge into details of the math here, since it's rather complex.

```
fprin:←
value#(1)=(balance#+payment#*(r1#^months-1)/rate#)/
r1#^months←
RETURN←
←
fintr:←
rØ#=Ø:r1#=EXP(75/months):IF r1#>2 THEN r1#=2 ←
rate#=r1#-1:r9#=rate#*1ØØ←
pØ#=balance#+payment#*months-principal#←
p9#=(balance#+payment#*(r1#^months-1)/rate#)/r1#^mo
nths-principal#←
IF pØ#<Ø OR p9#>Ø THEN ←
r2#=Ø←
ELSE←
flop%=Ø←
WHILE ABS(r9#-rØ#)>.ØØ1←
flop%=1-flop%←
```

```
IF flop%>Ø THEN←
r2#=(rØ#+r9#)/2←
ELSE←
r2#=rØ#-pØ#*(r9#-rØ#)/(p9#-pØ#)←
END IF←
r1#=(1+r2#/1ØØ/peryear(site))^(1/compound(site))←
rate#=r1#-1←
p2#=(balance#+payment#*(r1#^months-1)/rate#)/r1#^mo
nths-principal#←
IF p2#>Ø THEN←
rØ#=r2#:pØ#=p2#←
ELSE←
r9#=r2#:p9#=p2#←
END IF←
WEND←
END IF←
value#(2)=r2#←
RETURN←
←
fper:←
value#(3)=LOG((payment#-rate#*balance#)/(payment#-r
ate#*principal#))/LOG(r1#)/12#←
RETURN←
←
fpay:←
value#(4)=rate#*(principal#*r1#^months-balance#)/(r
1#^months-1)←
RETURN←
←
fbal:←
value#(5)=principal#*r1#^months-payment#*(r1#^mon
s-1)/rate#←
RETURN←
←
```

The only one of the above routines that's lengthy is *fintr*. There's no simple formula for the interest rate, so we must zero in on the correct value by repeated calculations.

Displaying Results

Now to display the calculated values:

```
showval:←
FOR j=1 TO 5←
LOCATE j,1←
IF j=cal THEN ←
PRINT "*";←
ELSE←
PRINT " ";←
END IF←
```

```
PRINT title$(j);SPACE$(20)←
LOCATE j,12←
PRINT USING pudef$(j);value#(j)←
NEXT j←
RETURN←
←
```

For a good human interface, I wanted to distinguish between the calculated item and the entered values. The title for the value being calculated will be preceded by an asterisk. SPACE$ is used to generate a string of blanks to wipe out any old values.

A Choice Is Made

```
menuhit:←
ms=0←
IF mn>4 THEN←
mn1=MENU(1)←
ON mn-4 GOSUB newcalc,style←
END IF←
RETURN←
```

Here's the routine to handle menu selections. The value *mn*, given the value of MENU(0) in the calling routine, is used to determine which menu is involved. MENU(1) tells us which item from the menu has been selected. We then subtract 4 from *mn* to get an offset of 1 or 2 for the ON-GOSUB statement.

```
←
newcalc:←
IF mn1<6 THEN←
MENU 5,cal,1←
cal=mn1←
MENU 5,cal,2←
ELSE←
IF mn1=6 THEN kwit=1←
END IF←
RETURN←
←
style:←
IF mn1<3 THEN←
MENU 6,site,1←
site=mn1←
MENU 6,site,2←
END IF←
RETURN←
←
```

The *newcalc* subroutine is called when menu 5, the Calculate menu, is selected. If the item selected from that menu is 1–5, the previously selected menu item has its checkmark removed, and a checkmark is placed beside the newly selected item. The value of *cal* is updated to show which variable is now being calculated. If menu item 6, Quit, is chosen, we instead set the value of *kwit* accordingly. The *style* subroutine sets *site* to the selected country when an item is selected from menu 6, the Country menu.

```
eek:←
x=MOUSE(3):y=MOUSE(4)←
IF x>5 AND x<190 THEN←
v=INT((y+8)/8)←
IF v>0 AND v<6 AND v<>cal THEN←
LOCATE v,12:PRINT SPACE$(20)←
LOCATE v,12:INPUT value#(v)←
LOCATE v,12:PRINT USING pude f$(v);value#(v)←
END IF←
END IF←
RETURN←
←
```

When the left mouse button is clicked, the *eek* subroutine allows entry of a new value. It's important to read MOUSE(0) before reading the mouse's position, but in this case, that's already been done in the *event* routine that calls *eek*. The *x* and *y* coordinates of the mouse pointer's current position come from MOUSE(3) and MOUSE(4), since those functions return the position of the mouse when the button was clicked. MOUSE(1) and MOUSE(2) return the mouse's position at the time of the MOUSE(0) call, so either would probably give comparable results in this case. Remember that we are reading pixel positions, not character positions. Before recognizing a click as a request to enter input, we check that the pointer was reasonably close to one of the displayed values. One more limitation is that we won't allow an entry for the *cal* variable: The computer calculates that value.

Once we know it's a valid variable, we clear the old value using SPACE$, input a new value, and then print it neatly formatted in the space provided.

Maiden Voyage

Let's give the program a trial run. First, you'll see the window appear. If you have used the initial values suggested, you'll

notice that the program has calculated a payment of $131.04. That's the Canadian computation. Now press the right button, slide the mouse pointer up to the Country menu, and move down to *USA* before you release the button. The payment should change to $132.16.

This is a ten-year mortgage. Let's see what the balance would be after five years. Use the right button (also called the menu button, for obvious reasons) to select the Balance option from the Calculate menu. The balance will show a slightly negative amount. That's okay (each payment is rounded up a fraction of a penny, so the final payment will be slightly less than 0). Next, move the pointer up to the Years value in the display window menu and click the left button. The computer is inviting you to enter a new value: Enter 5 for five years. Observe that the balance still due after five years is a little over $6,000.00.

How long to pay it off at $150 a month? Select Years from the Calculate menu. Change the Balance value to 0 and the Payment value to 150. The answer is, a little more than eight years. If you change the interest rate to 12 percent, you'll see that it would take over nine years to pay off the loan. At 18 percent, you wouldn't live long enough to pay it off at $150 a month, and at 20 percent, it's impossible (note that the Years value is set to 0 to indicate the error). When you've snooped through the combinations enough to satisfy yourself, select Quit. And don't forget to save the program. If your answers don't match these, check the formulae for typographical errors.

After running through this exercise, think how different things would be on any eight-bit computer. It's not just the mortgage calculation; it's the style of the machine. With a fresh approach, you can make your Amiga more flexible and useful than any computer you've used before.

Foolproof Input for Amiga BASIC

Tom Bunker

Amiga BASIC programmers will find this routine quite handy—a routine that creates edit-field boxes for accepting various kinds of keyboard input. The routine also demonstrates how well-designed subprograms can, in effect, add new commands to Amiga BASIC.

Amiga BASIC's ability to use custom subprograms is one of its most valuable features: It allows programmers to accumulate a library of very useful routines that can be attached to virtually any BASIC program. The simple requester window subprogram presented later in this chapter is just one example. Another subprogram that should be in every programmer's collection is a foolproof input routine.

The ideal input routine would simulate the Amiga operating system's own edit-field boxes. An example of such an edit field appears when you select the *Save as* option in Amiga BASIC's *Project* menu. A similar routine in BASIC would give your programs much more control than provided by the standard INPUT statement. It would be helpful, for instance, to be able to limit the number of characters that can be entered or to limit numeric input to integers rather than print error messages after the fact. The input routine shown here has all of these capabilities and more.

Edit Fields in BASIC

The complete input routine consists of two subprograms: Getline, which gets a line of input from the keyboard, and Box, which Getline calls to draw an edit-field box and cursor on the screen. The Box subprogram is very useful in its own right and can be used independently of Getline.

Getline lets you create the equivalent of an edit-field box in Amiga BASIC. Here are some of its features:

- The main program which calls Getline sets the maximum length of input allowed.
- The Box subprogram draws an edit-field box of appropriate size.
- The cursor inside the box can be flashing or nonflashing.
- The main program can select the type of input allowed: alphanumeric characters, real numbers, or integers.
- The range of alphanumeric characters accepted for input can be adjusted.
- Pressing the ESCape key aborts the input operation.
- A single keystroke can erase all input within the edit-field box.
- The main program can display a default entry within the edit-field box which the user can edit.

Getline can be used any time your program needs to accept input from the keyboard, for entry of data, filenames, or whatever. To use Getline, your program should first print any desired prompt message and leave the cursor at the point on the screen where input is to begin. Then you must call Getline using this general format:

CALL Getline (*string$,maxlength%,inputtype%*)

The string variable *string$* holds whatever default text you want to display inside the edit-field box for the user to edit, and also returns the input entered by the user. For instance, if Getline is called as part of a save-data-to-disk routine, you could suggest a default filename or use a filename which the user has previously indicated. If you don't want to display anything within the edit-field box when it appears, set this string variable to a null string (" ") before calling Getline. In any case, Getline returns the user's input in this string variable after the subprogram passes control back to your main program.

The second parameter (maxlength%) is an integer which sets the maximum input length. For instance, if you want to limit input to 30 characters, you'd specify a 30 for this parameter by supplying either an integer variable or a constant.

The last parameter (inputtype%) is an integer which tells

Getline which type of input to accept. There are three possible values:

0 accepts all alphanumeric characters without restriction.
1 accepts real numbers—the digits 0–9 and the decimal point.
2 accepts integers—only the digits 0–9.

The real and integer types also accept the plus and minus signs, but only in the first character position. Getline simply ignores all keystrokes that do not conform to the type of input selected.

CALLing Getline

Here are a couple of examples. Let's say you want the user to enter his or her name, up to 14 characters long, and you want your program to store the information in the string variable NAME$. The proper CALL would be:

```
CALL Getline (NAME$,14,0)
```

If you want the user to enter a three-digit integer number (perhaps a telephone area code), the proper CALL would be:

```
CALL Getline (NUMBER$,3,2)
```

Note that Getline always returns the user's input in a string variable. If the input you're seeking is an integer or a real number, you can convert it from string to numeric form with the VAL function after Getline returns control to your main program.

Remember, too, that Amiga BASIC's CALL statement has an alternate syntax: You can omit the CALL keyword if you delete the parentheses surrounding the arguments. The following statements work the same as the examples above:

```
Getline NAME$,14,0
Getline NUMBER$,3,2
```

This syntax saves a bit of program space, but also sacrifices a certain amount of program clarity. If you include the CALL keyword, it is always clear to others that the program is calling a subprogram.

Special Keystrokes

When called, the Getline subprogram first draws an edit-field box the proper size to hold the input. If the string variable

supplied in the call is not a null string (two quotes with nothing between them), the subprogram prints the string inside the box. A flashing cursor indicates that the program is awaiting keyboard input. Like the Amiga operating system's own edit fields, Getline recognizes the following special keystrokes:

- ESCape exits the edit field and leaves the string variable with the value it had when Getline was called.
- RETURN exits the edit field and assigns the user's entry to the string variable.
- BACKSPACE deletes the character to the left of the cursor.
- DELete removes the entry currently in the edit field.
- CURSOR LEFT moves the cursor one space to the left.
- CURSOR RIGHT moves the cursor one space to the right.

The last four commands, of course, are valid only if at least one character is within the edit field.

Customizing Getline

Note that Getline is designed to work only when Amiga BASIC's default font is used and Preferences is set to 80 columns. If you're using a 60-column screen or a different font, the text doesn't appear properly within the edit-field box. You can modify the subprograms to solve this problem if you don't regularly use the default 80-column font.

If you don't want to bother with three parameters every time you call Getline, you can omit either the maximum string length or input type or both, as long as you also delete the corresponding items from the parameter list of the SUB statement. The Getline call can be made as simple as this:

```
Getline NAME$
```

In this case, the SUB statement would have to be changed to look for only one argument:

```
SUB Getline(inputstring$) STATIC
```

Getline substitutes default values for maxlength% or inputtype% when they are missing from the parameter list. Maxlength% defaults to 40, and inputtype% defaults to 0 (thus accepting all types of input). You can change these defaults too, if you wish.

Two variables in the Getline subprogram—*asc.low* and

asc.high—determine the ASCII range of characters that are accepted in the edit field. You can change these variables to make the subprogram accept any range of characters desired, even to the extent of restricting input to only one key. They could also be declared in a SHARED statement and set by your main program.

The ESCape key aborts the input and exits the edit field. If your main program needs to know whether or not the edit field was terminated by ESCape (as opposed to a RETURN with no other input), add the following line to the Getline subprogram immediately following the SUB statement:

```
SHARED K
```

After the subprogram ends, your main program can test the value of K. If K = 27, the ESCape key was pressed.

You can also program one or more of the special function keys to work in a similiar fashion by adding additional lines directly below the ESCape-key line to test for any other ASCII value. For example, the addition of

```
IF K>=129 AND K<=138 THEN EXIT SUB
```

makes all the function keys abort the input like ESCape. Your main program could then test to see if K is equal to the ASCII value of any of the function keys and take whatever action is desired.

By deleting a single line as instructed by comments within the subprogram, Getline will always start with an empty string. Other comments show how the flashing cursor can be changed to a nonflashing cursor and how the box around the edit field can be eliminated. To make these changes, it's not necessary to actually delete the lines which are indicated. Simply insert a REM at the beginning of the line to disable it; this has the same effect and is more easily reversed.

The Box Subprogram

To draw the box around the edit field, Getline calls the Box subprogram. This subprogram selects a rectangular area of the screen and alters it in one of four ways. You may find this technique useful for other purposes as well. Here is the general format of the Box subprogram call:

CALL Box (*wide%,high%,border%,mode%*)

or

Box ***wide%,high%,border%,mode%***

The first two parameters (wide% and high%) set the size of the boxed area by specifying the width and height in number of characters. The third parameter (border%) changes the size selected by increasing or decreasing the area on all four sides by the number of pixels specified. If this argument is 0, the perimeter of the area falls on the character boundaries. The last parameter (mode%) can range from 0 to 3:

0 fills the box interior using a PATTERN statement.
1 inverts the interior of the box.
2 outlines the area using the foreground color.
3 fills the box interior using the foreground color.

The Box subprogram can be very useful when you want to erase a word or clear any rectangular section of the screen. Consider this statement:

COLOR background#:Box 30,1,0,3:COLOR foreground#

This erases a section of the screen 30 characters long without affecting any surrounding text. It sets the foreground color equal to the background color, fills the area, and resets the color. Of course, you can achieve the same effect by printing spaces, but the Box subprogram works much faster.

Getline Input Routine

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
SUB Getline(inputstring$, maxlength%, type%) STATIC
←
'Value of type% should be Ø for character, 1 for re
al, 2 for integer←
'Set default maximum length:←
defaultlength=4Ø←
IF maxlength%=Ø THEN maxlength%=defaultlength  ←
y=CSRLIN:x=POS(o):a$=""←
asc.low=32:asc.high=125 'Set ASCII limits←
'Delete next line to disable edit mode:←
a$=inputstring$ ←
cursor=LEN(a$):strlength=LEN(a$)←
'Delete next line to eliminate input box:←
Box maxlength%,1,2,2 ←
Print.line:←
LOCATE y,x:PRINT a$+SPACE$(maxlength%-LEN(a$))←
Getkey:←
```

```
k$=INKEY$ ←
'Delete next line for nonflashing cursor:←
count=count-1 ←
IF count<=Ø AND cursor<maxlength% THEN←
LOCATE y,x+cursor:Box 1,1,Ø,1←
count=1ØØ 'Set cursor flash rate:←
END IF←
IF k$="" THEN Getkey←
k=ASC(k$):count=Ø←
IF k=13 THEN inputstring$=a$:GOTO Done 'Return key←
IF k=27 THEN Done 'ESCape key←
IF k>=asc.low AND k<=asc.high AND strlength<maxleng
th% THEN←
IF type%>Ø THEN 'Check if real or integer←
IF k<43 OR k>57 OR k=44 OR k=47 THEN Print.line←
IF (k=43 OR k=45) AND cursor>Ø THEN Print.line←
IF type%>1 AND k=46 THEN Print.line←
END IF  ←
LOCATE y,x+cursor:cursor=cursor+1:strlength=strleng
th+1    ←
a$=LEFT$(a$,cursor-1)+k$+MID$(a$,cursor)←
PRINT MID$(a$,cursor):GOTO Getkey←
END IF←
IF k=31 AND cursor>Ø THEN 'Cursor left←
cursor=cursor-1 ←
ELSEIF k=3Ø AND cursor<strlength THEN  'Cursor righ
t←
cursor=cursor+1←
ELSEIF k=127 THEN 'Delete entry←
a$="":cursor=Ø:strlength=Ø ←
ELSEIF k=8 AND cursor>Ø THEN 'Backspace key←
cursor=cursor-1:strlength=strlength-1←
a$=LEFT$(a$,cursor)+MID$(a$,cursor+2)←
END IF←
GOTO Print.line←
Done:←
LOCATE y,x←
PRINT inputstring$+SPACE$(maxlength%-LEN(inputstrin
g$))←
END SUB←
←
SUB Box(wide%, high%, border%, mode%) STATIC←
'wide% and high% set size expressed as number of ch
aracters←
'border% is to be given as number of pixels←
'mode% - use Ø for pattern fill; 1 to invert area←
'mode% - use 2 for area outline; 3 to fill area wit
h foreground color←
y=CSRLIN*8-9-border%:y1=y:IF y1<Ø THEN y1=Ø←
x=POS(o)*8-9-border%:x1=x:IF x1<Ø THEN x1=Ø←
x2=x+wide%*8+1+2*border% ←
```

```
IF x2>=WINDOW(2) THEN x2=WINDOW(2)-1 ←
y2=y+high%*8+1+2*border% ←
IF y2>=WINDOW(3) THEN y2=WINDOW(3)-1←
IF x1>x2 THEN x1=x2 ←
IF y1>y2 THEN y1=y2←
IF mode%=2 THEN LINE (x1,y1)-(x2,y2),,b:EXIT SUB←
IF mode%=3 THEN LINE (x1,y1)-(x2,y2),,bf:EXIT SUB←
AREA (x1,y1):AREA (x2,y1):AREA (x2,y2):AREA (x1,y2)
←
AREAFILL mode%←
END SUB←
←
```

Amiga Math Graphics

Warren Block

Is math boring? Before you answer, take a look at this Amiga BASIC program. It creates graceful, multicolored graphic designs based on a variety of interesting mathematical functions.

As one of my first Amiga programming projects, I decided to convert several Apple II+ hi-res graphics routines to run on my new machine. Originally, all these routines were written as one-liners: That is, the entire program would fit (just barely, sometimes) on one BASIC line. "Amiga Math Graphics" combines all of them into a single program. At the very least, these routines demonstrate the speed and power of the Amiga, while creating a pleasing visual display. At their best, perhaps they will convince you to explore the field of microcomputer graphics—a field which many people avoid because it seems difficult. Pictures are a fundamental part of communication, and being able to use graphics on the computer will improve your ability to communicate through that medium.

Type in the program and save a copy before you run it. The small ← character indicates where each program line ends. Don't try to type this character—we deliberately chose one that's not on the Amiga keyboard. The ← character merely shows where you should press RETURN to end one program line and start another.

Labeled Subroutines

Although the routines in this program were originally one-liners, it seemed a shame to keep them that way when AmigaBASIC makes it so easy to write neat, readable code. Each routine is marked with a descriptive label. Let's look at each of them in turn.

RightOvals. The basic formula used in this routine forms the basis for several different plotting routines. They all involve drawing a line from the perimeter of one oval to the perimeter of another. In this case, the line is drawn from a point on the first oval to a point halfway along the other.

SideOvals. Only minor changes were made to RightOvals to produce this interesting display. The second oval was tilted with respect to the first, and the line is plotted with an offset added to the *x* coordinate of the second oval.

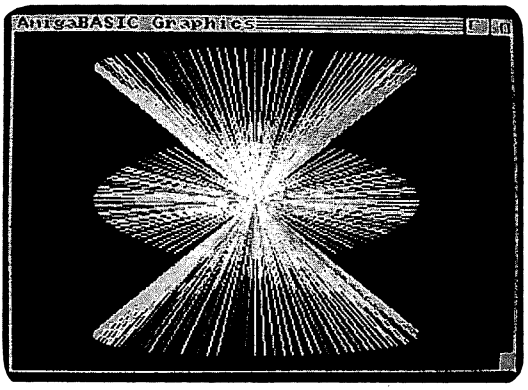
Scaling Graphic Shapes

When the trigonometric functions sine and cosine are used for graphics, a problem arises because both of these functions return only values between 0 and 1. Without scaling (adjusting) the figures to fit the computer's display, you would see only three or four pixels in the middle of the screen. Scaling the display involves multiplying a set of coordinates by a constant amount. However, if you multiply both the *x* (horizontal) and *y* (vertical) coordinates by the same amount, the graph appears to be squashed horizontally on the screen. This occurs because the Amiga's aspect ratio (the ratio of horizontal to vertical pixels) is greater than 1. In plain English, there are more pixels across the screen than there are from top to bottom. To adjust for the aspect ratio, you must make the horizontal scaling factor larger than the vertical factor.

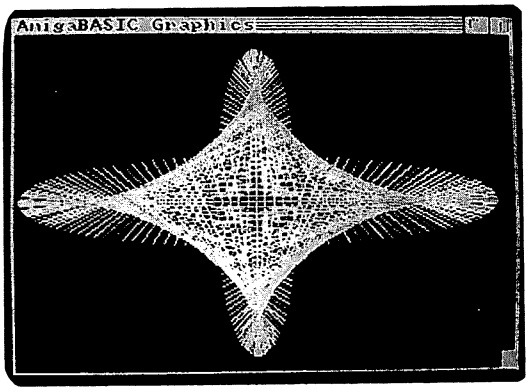
Other factors influence aspect ratio, including the type of monitor you have and the physical shape and relative locations of the pixels it displays. Some experimentation is required to find the best scaling values for any given display. In this program, the R variables (R1, R2, and so on) set the scaling factors for various routines. By changing these values, you can squash the shapes vertically or horizontally.

TwistedBand. Using a minor variation on the double-oval effect, this routine creates a display that looks remarkably like a twisted loop of paper. The only real difference from SideOvals is that an offset is added to the *y* coordinate of the second oval, not to its *x* coordinate.

MultiLobe. This routine employs a common polar function which involves multiplying an angle, theta, by a fixed constant, then using this new value to compute the R value (theta and R are discussed at the end of this article). The effect is that of several squashed, distorted lobes instead of a plain



"Amiga Math Graphics" creates these graceful shapes with short routines based on polar functions.



circle. By setting the variable Lobes to 4, eight lobes are drawn. Try changing Lobes to different values (including nonintegers) for some interesting variations.

Show Your Colors

Before you bought an Amiga, you may have heard that it can display 4096 different colors. The low-resolution graphics screen lets you display as many as 32 different colors at once. If you're familiar with earlier computers, the Amiga's color system may seem confusing at first. On a Commodore 64, for example, color 2 is always red, and so on. But the Amiga, like the PC/PCjr, allows you to assign any color to color 2. The

PALETTE statement allows you to define color 2 as black, magenta, or whatever you like. The color number simply provides a means for referring to that color—however you define it.

To use PALETTE, imagine that you have three cans of paint: one red, one green, and one blue. By mixing various portions of these cans together, you can create almost any conceivable color. For example, to make a bright red, take 90 percent of the paint in the red can and mix it with 20 percent of the paint in the green can (you don't need any blue). By coincidence, this is just the way the PALETTE statement works. The statement PALETTE 5,.90,.20,0 assigns a bright red color to color 5. (Strictly speaking, color mixing in Amiga BASIC is more like mixing colors of *light* than colors of paint. Thus, the statement PALETTE 5,1,1,1 sets red, green, and blue to maximum intensity, creating a white color. If you mix red, green, and blue pigments of equal intensities, the result is a very dark brown or black.)

SpiralCone. Using a method similar to that used by MultiLobe, this routine multiplies the theta value by 3, resulting in a six-lobed figure. However, only the *x* coordinate for this figure is used. The *y* coordinate is calculated using the normal value of theta. A conelike shape is formed by drawing all lines from the center of the display to the calculated points.

SideSpiralCone. This is merely SpiralCone drawn sideways, with different scaling values. The difference in appearance is substantial enough to prevent most viewers from detecting the similarities.

The last two routines in the program rely on similar functions, but produce patterns that look very different on the screen.

Circles. This routine defines a small circle surrounded by a larger one; then it picks 6 equally spaced points on the inner circle. The final design is created by drawing a line from each of those points to 20 or so equally spaced points on the outer circle.

Spikes. Although this routine looks nearly identical to Circles, the shape it draws is completely different.

There's a System to This

You can enjoy and experiment with this program without understanding the math that underlies the graphics. For those who are interested, here's a further explanation of how it works.

In the field of mathematics, there are many systems for expressing the location of a point in a plane. Generally, the center of the system is referred to as the *origin*. The origin is simply a reference point; the location of all other points is defined with respect to the origin.

Most people are familiar with the Cartesian coordinate system, in which the location of any point is expressed in terms of x and y coordinates. The x value represents the point's horizontal distance from the point of origin. Similarly, the y coordinate represents the point's vertical distance from the origin.

The Cartesian system works well for representing two- and three-dimensional shapes on a two-dimensional surface such as the computer's display screen. However, the *polar* coordinate system is much more convenient when you're using trigonometric functions such as sine and cosine. In this scheme, a point's location is expressed as a distance from the origin (conventionally labeled R) and an angle (usually labeled *theta*, or with the Greek letter θ) from a reference line.

Polar Functions

The routines in this program are all based on polar functions. Since Amiga BASIC commands use Cartesian coordinates (roughly—see below), it's necessary to convert from polar to Cartesian coordinates. In general, this operation can be performed by the expressions X=R*COS(theta) and Y=R*SIN (theta).

There are a few difficulties in adapting the graph of a polar function to a computer display. The easiest problem to allow for is the fact that most graphics displays (including the Amiga's) use an upside-down Cartesian system: That is, a point's y coordinate specifies how far *down* the screen the point lies—the exact opposite of the normal Cartesian system. Since all of our shapes are vertically symmetrical, this problem can simply be ignored.

Another difficulty arises because the Amiga's display does not allow for negative coordinates. The Amiga's origin point is in the upper left corner of the screen, not the center of the viewing area as in the Cartesian system. This can easily be corrected by considering the middle of the display to be the origin. In the calculations, all this involves is adding an x and y offset to the points you wish to plot.

Amiga Math Graphics

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
MathGraphics:←
GOSUB Initialize←
' Repeat until the user presses a key.
←
WHILE INKEY$=""←
' Module 1:RightOvals                    ←
R1=150←
R2=25                                  ←
R3=25 ←
R4=85←
Inc=Pi/64←
FOR Theta=0 TO 2*TwoPi STEP Inc        ←
X1=FNPolarX(R1,Theta)←
Y1=FNPolarY(R2,Theta)←
X2=FNPolarX(R3,Theta+Pi)←
Y2=FNPolarY(R4,Theta+Pi)←
LINE(X2,Y2)-(X1,Y1),INT(RND*31)+1    ←
NEXT←
Pause←
' Module 2:SideOvals--←
' Same thing, only different.←
R1=150←
R2=35←
R3=65 ←
R4=85←
Inc=Pi/64←
Offset=Pi/3←
FOR Theta=0 TO 3*TwoPi STEP Inc←
X1=FNPolarX(R1,Theta)←
Y1=FNPolarY(R2,Theta)←
X2=FNPolarX(R3,Theta+Offset)←
Y2=FNPolarY(R4,Theta)←
LINE(X1,Y1)-(X2,Y2),INT(RND*31)+1←
NEXT←
Pause←
' Module 3:TwistedBand←
' Yet another variation on the double oval theme.←
R1=150←
R2=35←
R3=65 ←
R4=85←
Inc=Pi/64←
Offset=Pi/3←
FOR Theta=0 TO 3*TwoPi STEP Inc←
X1=FNPolarX(R1,Theta)←
Y1=FNPolarY(R2,Theta)←
X2=FNPolarX(R3,Theta)←
```

```
Y2=FNPolarY(R4,Theta+Offset)←
LINE(X1,Y1)-(X2,Y2),INT(RND*31)+1←
NEXT←
Pause←
' Module 4:Multilobe←
R1=100←
Inc=Pi/128←
Lobes=4←
FOR Theta=0 TO 2*TwoPi STEP Inc←
R2=R1*SIN(Lobes*Theta)←
X1=FNPolarX(R2,Theta)←
Y1=FNPolarY(R2,Theta)←
LINE (XCenter,YCenter)-(X1,Y1),INT(RND*31)+1←
NEXT←
Pause←
' Module 5:SpiralCone←
R1=100←
R2=85←
Inc=Pi/160←
Lobes=3←
FOR Theta=0 TO 2*TwoPi STEP Inc←
X1=FNPolarX(R1,Theta*Lobes)←
Y1=FNPolarY(R2,Theta)←
LINE (XCenter,YCenter)-(X1,Y1),INT(RND*31)+1←
NEXT←
Pause←
' Module 6:SideSpiralCone←
R1=130←
R2=80←
Inc=Pi/160←
Lobes=3←
FOR Theta=0 TO 2*TwoPi STEP Inc←
X1=FNPolarX(R1,Theta)←
Y1=FNPolarY(R2,Theta*Lobes)←
LINE (XCenter,YCenter)-(X1,Y1),INT(RND*31)+1←
NEXT←
Pause←
' Module 7:Circles←
R1=115←
R2=85←
R3=40←
R4=45←
Inc1=Pi/3←
Inc2=Pi/20←
FOR Theta1=0 TO TwoPi STEP Inc1←
FOR Theta2= 0 TO TwoPi STEP Inc2←
X1=FNPolarX(R1,Theta2)←
Y1=FNPolarY(R2,Theta2)←
X2=FNPolarX(R3,Theta1)←
Y2=FNPolarY(R4,Theta1)←
LINE (X1,Y1)-(X2,Y2),INT(RND*31)+1←
```

```
NEXT←
NEXT←
Pause←
' Module 8:Spikes←
R1=115←
R2=85←
R3=40←
R4=45←
Inc1=Pi/3←
Inc2=Pi/18←
FOR Theta1=0 TO TwoPi STEP Inc1←
FOR Theta2= 0 TO TwoPi STEP Inc2←
X1=FNPolarX(R1,Theta2)←
Y1=FNPolarY(R2,Theta1)←
X2=FNPolarX(R3,Theta1)←
Y2=FNPolarY(R4,Theta2)←
LINE (X1,Y1)-(X2,Y2),INT(RND*31)+1←
NEXT←
NEXT←
Pause←
WEND←
' Shut everything down and quit.←
WINDOW CLOSE 2                          ←
SCREEN CLOSE 2←
WINDOW OUTPUT 1←
END←
←
SUB Pause STATIC←
FOR Delay=1 TO 5000←
NEXT←
CLS←
END SUB←
←
Initialize:←
' Set up a 32 color low-res screen.←
SCREEN 2,320,200,5,1←
WINDOW 2,"AmigaBASIC Graphics",(0,0)-(297,185),23,2
←
CLS←
' Color 0 (background) is black.←
PALETTE 0,0,0,0←
' Set up the other 31 colors as random combinations
.←
FOR L=1 TO 31                           ←
PALETTE L,RND,RND,RND←
NEXT←
' Keep the random sequence random.←
RANDOMIZE TIMER←
' Define constants.←
Pi=3.14159                              ←
TwoPi=2*Pi←
```

```
XCenter=151€
YCenter=93€
' Define polar to Cartesian conversion functions.€
DEF FNPolarX(R,Theta)=R*COS(Theta)+XCenter€
DEF FNPolarY(R,Theta)=R*SIN(Theta)+YCenter€
RETURN €
```

Requester Windows in Amiga BASIC

Tom R. Halfhill

Add your own custom requester windows to any Amiga BASIC program. Like dialog boxes on the Macintosh, requester windows allow your programs to flag errors or request confirmation before carrying out important functions.

Amiga BASIC is the most powerful BASIC interpreter supplied with any personal computer on the market. Written by Microsoft, it combines in a single language almost every feature found in IBM PC Advanced BASIC plus Microsoft BASIC for the Macintosh. In fact, many IBM BASICA and Macintosh BASIC programs will run on the Amiga with minor modifications.

However, Amiga BASIC does lack two key statements found in Macintosh BASIC: DIALOG and BUTTON. Both are important for writing BASIC programs which retain the mouse-and-window user interface common to the Macintosh and Amiga Workbench. Fortunately, both commands can be simulated fairly easily with Amiga BASIC's WINDOW and MOUSE statements.

In Macintosh BASIC, the DIALOG command lets a program open a *dialog box* (a small window) like those displayed by the Macintosh's operating system whenever the user must choose between two or more options. Dialog boxes also flag errors and alert users when they're about to activate a function that has irreversible consequences—such as quitting a program without saving the data on disk. For example, if the user pulls down a menu and selects Quit, a dialog box might open up and ask, "Quit program? (Data file not saved.)" Below this message is usually a pair of small boxes or circles called *buttons* which might be labeled OK and CANCEL. Pointing and

clicking the mouse on the OK button exits the program; pointing and clicking on the CANCEL button cancels the Quit function and returns to the main program so the user can save his or her data if desired.

In Amiga BASIC, the DIALOG and BUTTON commands must be simulated by a routine that uses the WINDOW and MOUSE statements. For greater convenience, the routine can be written as a *subprogram*, another advanced feature included in Amiga BASIC. Subprograms are similar to subroutines, except they can have *local variables*. These are variables which are independent of the main program. For instance, if your main program uses a variable X for some purpose, a subprogram can also use a variable named X, and it is treated as a separate variable. If the subprogram changes the value of its variable X, the main program's variable X is unaffected, and vice versa. On the other hand, a subprogram can also specify *shared variables*, sometimes known as *global* variables—those which are common to both the subprogram and the main program.

A major advantage of subprograms is that you can build up a library of useful routines on disk and add them to any new programs you write. This saves you the trouble of writing the same subprograms again and again. Although you can do the same thing with ordinary BASIC subroutines, there's always the chance that a subroutine variable might conflict with an identically named variable in your main program. Since subprogram variables are local, you're freed from this worry. Subprograms are truly programs within a program.

The Requester Subprogram

On the Amiga, dialog boxes are called *requesters*. Probably the most frequently encountered requester is the one that pops up when the Amiga asks you to insert a different disk. For the sake of consistency, an Amiga requester generally appears as a small window in the upper left corner of the screen, has a title bar labeled System Request, has two or three buttons, does not have a resizing gadget or close gadget, and cannot be moved elsewhere on the screen.

The "Requester Window Subprogram" listed below duplicates most of these features. It creates a window that appears in the upper left corner of the screen (or up to the full width of the screen in low-resolution modes); the window has a title

bar labeled Program Request (to distinguish it from System Request windows); there is no resizing gadget or close gadget; and the window cannot be moved elsewhere on the screen. Unlike system requesters, this requester always displays two buttons, and they're always labeled OK and CANCEL.

The subprogram lets you display one or two lines of your own text in the Program Request window. The maximum number of characters allowed in each line depends on whether the Amiga has been set for 60- or 80-column text with the Preferences tool. If Preferences is set for 60 columns, each requester line can be up to 31 characters long. If Preferences is set for 80 columns, each line can be up to 39 characters. (You can adjust the subprogram for either mode by changing a single program statement; see the remarks in the listing.) If you try to display a line of text which exceeds these limits, the subprogram leaves off the extra characters. Since you won't know how Preferences is set if you're writing programs that might be used by other people, it's safest to assume 60 columns and restrict each line of your message to 31 characters.

Opening a Program Request window is this simple:

```
request1$="This is the first line."
request2$="This is the second line."
CALL Requester
```

The two lines of your message are defined in the string variables *request1$* and *request2$*, and the CALL statement runs the subprogram (similar to GOSUB). The subprogram opens the requester window and waits for the user to click on the OK or CANCEL button. Clicks outside the buttons are ignored, although a click outside the requester window itself deselects it as the active window. It can be reselected, of course, by clicking within the window.

If the user clicks on OK, the subprogram returns a value of 1 in the variable *answer*. If the user clicks on CANCEL, *answer* equals 0. In either case, the subprogram closes the requester window after the button click and passes control back to the line following the CALL Requester statement. By testing *answer*, your program can branch to different routines to handle the user's response as required.

Hints for Use

Here's an example. Suppose your BASIC program sets up a Project menu with a Quit selection (a consistent feature in Amiga software). When your MENU statement detects that Quit has been selected, it can GOSUB Quit:

```
Quit:
 MENU OFF:CLS
 request1$="Quit program?"
 request2$="(OK exits to Workbench or CLI.)"
 CALL Requester
 IF answer=0 THEN RETURN
 SYSTEM
```

If the user selects Quit by accident or changes his mind, he can click on CANCEL and no harm is done—the Quit routine merely RETURNs. Otherwise, a click on OK stops the program and exits BASIC with the SYSTEM command. Of course, you could also include a check to see if any data created with the program has been saved, and if necessary, prompt the user to save it before quitting.

There are only two more details to keep in mind when using the requester routine. First, the WINDOW statement near the beginning of the subprogram opens WINDOW 2. If there's a chance that your program might already have two or more windows open when the requester is called, change this statement to WINDOW 3, or WINDOW 4, or whatever is necessary to avoid a conflict.

Second, the WINDOW statement defaults to the primary (Workbench) screen. That means the requester window always pops up on the primary screen. If your main program creates a secondary screen with the SCREEN statement, you'll want the requester window to appear on that screen instead of the primary screen. Otherwise, the requester will be invisible. To make the requester window appear on your program's secondary screen, append the screen's number to the WINDOW statement.

For instance, if your program creates a secondary screen with a statement such as this:

```
SCREEN 1,320,200,1,1
```

change the WINDOW statement in the requester subprogram as follows:

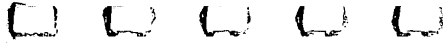
WINDOW 2,"Program Request",(0,0)-(311,45),16,1

This makes sure the requester will be visible.

Requester Window Subprogram

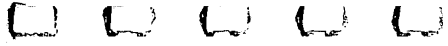
The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
RequesterSub:←
SUB Requester STATIC←
SHARED request1$,request2$,answer:' Global variable
s.←
' Add screen parameter if needed to next line.←
WINDOW 2,"Program Request",(0,0)-(311,45),16←
' If Preferences is set for 60 columns,←
' use maxwidth=INT(WINDOW(2)/10) for next line;←
' otherwise use maxwidth=INT(WINDOW(2)/8).←
maxwidth=INT(WINDOW(2)/8) ←
request1$=LEFT$(request1$,maxwidth)←
request2$=LEFT$(request2$,maxwidth)←
PRINT request1$:PRINT request2$←
' This section draws buttons.←
LINE (12,20)-(50,38),1,b←
LINE (152,20)-(228,38),1,b←
LOCATE 4,1:PRINT PTAB(20);"OK";←
PRINT PTAB(160);"CANCEL"←
' This section gets input.←
reqloop:←
WHILE MOUSE(0)=0:WEND:' Wait for button click. ←
m1=MOUSE(1):m2=MOUSE(2)←
IF m1>12 AND m1<50 AND m2>20 AND m2<38 THEN←
answer=1:' OK was selected.←
LINE (12,20)-(50,38),1,bf:' Flash OK box.←
WHILE MOUSE(0)<>0:WEND:' Wait for button release.←
WINDOW CLOSE 2:EXIT SUB←
ELSE←
IF m1>152 AND m1<228 AND m2>20 AND m2<38 THEN←
answer=0:' CANCEL was selected.←
LINE (152,20)-(228,38),1,bf:' Flash CANCEL box.←
WHILE MOUSE(0)<>0:WEND:' Wait for button release.←
WINDOW CLOSE 2:EXIT SUB←
ELSE←
GOTO reqloop←
END IF←
END IF←
GOTO reqloop←
END SUB←
←
```



CHAPTER FOUR

Beyond BASIC



Meet ED, the AmigaDOS Editor

Christopher J. Flynn

AmigaDOS—the command-driven operating system which underlies the graphics-oriented Workbench—contains two text editors. Although they aren't full-fledged word processors, these editors are ideal for entering program source code, creating batch files, and even writing short documents.

The Amiga comes with more software than most people realize. Besides Amiga BASIC, Electronic Arts' *Kaleidoscope*, Mindscape's *Amiga Tutor*, the RAM disk, the speech synthesizer, the printer drivers, the icon editor, the calculator, the clock, and numerous demo programs, there are also three complete text editors. Most people know about the Notepad because it's available from the Workbench. But the other two text editors—ED and EDIT—don't show up as icons and must be run from an AmigaDOS CLI (Command Line Interface) window.

The most powerful of these text editors is ED. Although it doesn't handle multiple fonts and styles like the Notepad, it has many more editing functions and is the ideal tool for writing AmigaDOS batch files or program source code. EDIT, on the other hand, is a little more specialized. It is a sequential file editor. In practice, EDIT is best used to make changes to an existing disk file. You'll probably prefer to use ED for composing new text.

We'll be exploring ED version 1.10. Future releases of ED may change things around a little and introduce new features, so keep this in mind.

Starting ED

Where is ED hiding? Even if you peek through every nook and cranny of the Workbench, you will not find an icon for ED. It turns out that ED is actually an AmigaDOS command.

This means that you have to start ED from a CLI window.

If you've never used a CLI window before, your first step will be to activate the CLI. Open the Workbench and check the contents of the System drawer. If CLIs are activated, you'll see a cube-shaped icon labeled CLI in this drawer. If the icon is not present, point to the Preferences icon and double-click the mouse's left button. Look for the CLI On/Off selector on the Preferences screen and click on the On box; then exit Preferences by specifying Save (*not* Use). Now when you reopen the System drawer, it should contain a CLI icon. If not, go back to Preferences and make sure CLI is turned on. (If you find yourself using the CLI often, you may want to drag the CLI icon from the System drawer into the main Workbench window to avoid the extra step of opening the System drawer.) To open a CLI window, double-click on the CLI icon. Now you'll have a window in which you can type AmigaDOS commands.

ED can be started in two ways:

ED *filename* [SIZE *n*]

RUN ED *filename* [SIZE *n*]

The first method starts ED from the CLI which you've just activated. It ties up the CLI until you're finished with ED. In other words, you have to leave ED before issuing other AmigaDOS commands. When you specify RUN ED, AmigaDOS automatically starts another CLI task for you and starts ED in this new CLI. Thus, you can temporarily suspend ED by moving the mouse to another window. You can go back to the original CLI and issue other AmigaDOS commands. If you are adventuresome, you can even have multiple ED sessions in progress at the same time. (What you're really doing is multitasking more than one AmigaDOS command simultaneously.)

In either case, the ED command requires a filename. You can either supply the name of an existing disk file you wish to edit, or create a new file by specifying a new filename. Remember that Amiga filenames can be up to 30 characters long. So, choose filenames that take advantage of this feature. It helps you recognize your files later on.

There is a SIZE option for the ED command. (Don't type in the brackets, by the way. Brackets just signify options.) A text document must be able to fit entirely in memory. ED just

cannot handle a document partly on disk and partly in memory. The SIZE option gives you a way of telling ED how much memory you want to set aside for working on the document. If you don't type in SIZE, ED will set aside 40K for you. The maximum SIZE is determined by the amount of memory you have.

Here are a few examples of commands for starting up ED:

ED GROCERY-LIST

ED WAR-PEACE-BOOK-REPORT SIZE 90000

When SIZE is used, type out the number. Note that 90,000 bytes is typed as 90000 and not as 90,000 or 90K.

Leaving ED

When ED has been successfully started, its display occupies the entire screen. So, how can you return to the CLI? There is no close gadget on ED's window. There is nothing to point at and click. Instead, ED requires either a Quit or an Exit command. Press the ESC (escape) key. An asterisk appears on the last line of the display. Type either Q for Quit or X for Exit and then RETURN. That's all there is to it.

There is a difference between Quit and Exit. Q leaves the editor without saving the document to disk. Anything you have typed will be lost. ED recognizes that this can be quite an inconvenience, so if you do type Q, ED displays the following warning message:

Edits will be lost - type Y to confirm:

Pressing Y at this point gets you out of ED, and no text is saved. If you type anything else, ED lets you continue working on your document.

ESC-X, the Exit option, *does* save the document on disk, using the filename you specified when you started ED. No messages are given. When ED finishes, you're back in the CLI and can then proceed with other AmigaDOS commands. When you've finished with the CLI, type ENDCLI. If you've got only one CLI window running, this returns you to the Workbench.

ED Commands

There are two types of editor commands in ED. The more direct ones are called *immediate commands* because you can enter them while typing text. Examples are line insertions and deletions. Immediate commands are always CTRL key combinations. The other category—*extended commands*—can be typed only when in the command mode. ESC-Q and ESC-X are examples. Pressing ESC opens the lowest display line on your screen for these extended commands.

When ED starts, it positions the cursor at the upper left corner of the screen. If you are working on a new document, the screen is blank. Otherwise, the screen shows the first page of the document.

If you're creating a new document, just start typing. Notice what happens when the text approaches the right side of the screen. If a word is too long to fit on the remainder of the line, ED pulls the word down to the next line. You can keep typing without being concerned about hitting RETURN at the end of a line as you would on a typewriter.

There are several ways of correcting typos. The BACK SPACE key deletes the character to the left of the cursor. DEL deletes the character under the cursor. Table 1 lists other ways of deleting text.

ED is a full-screen editor, so you can move the cursor wherever you want with the arrow keys. To insert text, position the cursor at the desired location and begin typing. Notice that ED does not have a strikeover mode. Unwanted text has to be deleted—you can't just type over it.

The Insertion Gotcha

Try typing a few fairly long lines. Now, move the cursor to the beginning of the text. Start typing again. The existing text on the current line is moved to the right off the edge of the screen. During insertions, ED neither brings the excess text down to the next line nor enforces margins.

The disappearing text is not lost, however. ED has made one long line. The long line can be split at any point by placing the cursor where you want and pressing RETURN. If you're working with ordinary text, not source code or batch files, this may leave gaps of several spaces between sentences. To clean up the appearance, the extra spaces will have to be removed. Some other lines may need adjusting as well.

Using the Extended Commands

Extended commands (Table 2) can be typed only when ED is in the extended-command mode, entered by pressing the ESC key. The cursor appears on the last line of the display. At this point, you can type one or more extended commands. It's quite handy to be able to give ED a series of commands separated by semicolons (;). When you press RETURN, ED acts on the command or commands you've requested.

Extended commands can move the cursor, mark blocks of text for certain operations, and perform searches and exchanges. Some of the operations are tricky and require care. Cursor commands apply only to the cursor position in the text and not to the command line. This is fine except that you can't see the cursor in the text. You have to remember where the cursor is before you use some of the extended commands.

Sections of text can be marked by block start (BS) and block end (BE) commands. Blocks can be deleted, copied elsewhere in the document, or saved to disk. Marking a block involves moving the cursor to the first line in the block and executing the BS extended command. The end of the block is marked similarly with BE. Unfortunately, there is no visible indication of the defined text. Be very careful of cursor movements. The only help ED offers is the show (SH) command. It displays the first and last line of the block and some other information.

Text search-and-exchange operations work without a hitch. You can search forward (F) or backward (BF) through the document. You can exchange (E or EQ) one text string for another. Lowercase text can be treated as matching uppercase text (UC), or it can be treated as not matching (LC).

The repeat (RP) command is often used for exchanges. RP causes the command following it to be executed repeatedly until something (an error, for example) stops it. Thus, RP E carries out multiple exchange operations. Here is an example:

```
T; RP E /Compute/COMPUTE/
```

Here, the typing of COMPUTE is being corrected. T moves the cursor to the top of the document so that the entire document will be examined. RP precedes the exchange command. Note that the two text strings are delimited by slashes. This is ED's convention when text strings are used. A "Search failed" error occurs when *Compute* can no longer be found in

the text. This halts the repeat command, and the entire document will have been corrected.

The save command (SA) saves the document to disk without exiting ED. You should do this periodically to prevent disasters in the event of a power failure.

Overall, ED is an excellent general-purpose text editor. You can use it when programming, since it works with any language that accepts ASCII text files as input (including Amiga BASIC). ED can also prepare data files or help you write short letters and notes. It's not a fancy word processor, but it can handle smaller, less complex tasks quite well.

Table 1. ED Immediate Commands

Command	Description
Special Keys	
BACK SPACE	Deletes the character to the left of the cursor.
DEL	Deletes the character under the cursor.
ESC	Switches to extended-command mode.
RETURN	Ends the line at the cursor and starts a new line.
TAB	Moves the cursor right, adding spaces, to the next tab position.
up-arrow	Moves the cursor up one line.
down-arrow	Moves the cursor down one line.
left-arrow	Moves the cursor one character position to the left.
right-arrow	Moves the cursor one character position to the right.
Control Key Combinations	
CTRL-A	Inserts a line after the line on which the cursor is located.
CTRL-B	Deletes the line on which the cursor is located.
CTRL-D	Scrolls the text down 12 lines toward the beginning of the document.
CTRL-E	If the cursor is at the top of the screen, moves the cursor to the bottom of the screen. If the cursor is at the bottom of the screen, moves the cursor to the top of the screen.
CTRL-F	Switches the case (upper- to lower- or lower- to upper-) of the character under the cursor.
CTRL-G	Repeats the last extended command which was issued.
CTRL-H	Deletes the character to the left of the cursor. Equivalent to the BACK SPACE key.
CTRL-I	Moves the cursor right to the next tab position. Equivalent to the TAB key.

CTRL-M	Equivalent to the RETURN key.
CTRL-O	If the cursor is on a nonblank character, deletes all characters from the cursor to the first space. If the cursor is on a space, deletes all spaces from the cursor to the first nonblank character.
CTRL-R	Moves the cursor left to the first space after the previous word on the current line.
CTRL-T	Moves the cursor right to the first character of the next word on the current line.
CTRL-U	Scrolls the text up 12 lines toward the end of the document.
CTRL-V	Redisplays (verifies) the screen. Insures that all the text is visible and is useful after moving or sizing the display window.
CTRL-Y	Deletes all characters on the line starting with the character under the cursor.
CTRL-[	Switches to the extended-command mode. Equivalent to the ESC key.
CTRL-]	If the cursor is at the start of the line, moves the cursor to the end of the line. If the cursor is at the end of the line, moves the cursor to the start of the line.

Table 2. ED Extended Commands

Note: /s/ refers to a single text string (/this is a string/).
/s/t/ refers to two text strings (/brown/blue/).

Command	Description
A /s/	Inserts the string on a new line after the current line.
B	Moves the cursor to the end (bottom) of the document.
BE	Places an end-of-block marker at the cursor.
BF /s/	Searches the document for the string going in a direction from the cursor toward the beginning of the document (backward find).
BS	Places a start-of-block marker at the cursor.
CE	Moves the cursor to the end of the current line.
CL	Moves the cursor one character position to the left.
CR	Moves the cursor one character position to the right.
CS	Moves the cursor to the start of the line.
D	Deletes the current line. Moves all following lines up.
DB	Deletes the text marked by start-block and end-block markers.
DC	Deletes the character at the current cursor position.
E /s/t/	Replaces (exchanges) occurrences of the first string with the second string.

EQ /s/t/	The same as E, but asks you to confirm the replacement each time a match is found. Type Y or N in response to the Exchange ? prompt.
EX	Extends the right margin, allowing additional text to be typed.
F /s/	Searches the document for the string going in a direction from the cursor position toward the end of the document (find).
I /s/	Inserts the string on a new line before the current line.
IB	Inserts the block of text marked by start-block and end-block markers after the current line.
IF /s/	Inserts the contents of a file before the current line. The filename is given by /s/.
J	Joins the current line with the next line. This makes one new line where there were formerly two.
LC	Treats upper- and lowercase characters as different in searches.
M *n*	Moves the cursor to the line number given by *n*.
N	Moves the cursor to the starting position of the next line.
P	Moves the cursor to the starting position of the previous line.
Q	Quits ED without first saving the text. A warning message will be given stating that the text may be lost.
RP	Repeats commands. Commands are typed following RP. For example, T; RP E /brown/red/ moves the cursor to the top of the document. The Exchange command is repeated, thus changing all occurrences of *brown* to *red*. Repeat ends when an error is found. In this case, an error occurs after all the changes have been made since *brown* can no longer be found.
S	Splits the current line at the cursor location. This makes two lines where there was formerly one.
SA	Saves the document to the file specified by the original ED command. Use SA periodically to make sure you have a good copy of the text on disk.
SB	Shows the text block marked by start-block and end-block markers. The block (and any following text) will be displayed starting at the top of the screen.
SH	Shows the filename, tab distance, margin settings, first and last lines of any marked text block, and the buffer-full percentage.

SL *n*	Sets the left margin to the position specified by *n*. SL affects the margin setting for the entire document. New text will be typed within the margins. Existing text is not automatically reformatted when the margins change.
SR *n*	Sets the right margin to the position specified by *n*. SR affects the margin setting for the entire document. New text will be typed within the margins. Existing text is not automatically reformatted when the margins change.
ST *n*	Sets the distance the cursor moves when the TAB key is pressed.
T	Moves the cursor to the top of the document.
U	Undoes any changes made to the current line. This does not restore line deletes (D). It also does not work if you have moved the cursor from the current line.
UC	Treats upper- and lowercase characters as equivalent for searches (for example, *A* will match *a*).
WB /s/	Writes the text block marked by start-block and end-block markers to the file specified by /s/.
X	Exits ED, first making sure that the document has been saved on disk.

AmigaDOS Batch Files

Charles Brannon

AmigaDOS is more than a console-driven disk operating system. By executing a sequence of AmigaDOS commands stored in a file, AmigaDOS takes on some of the characteristics of a programming language. Whether you want to simplify repetitive disk commands or create personalized custom commands, batch files further extend the range and flexibility of AmigaDOS.

No matter how easy it is to use a program, the most popular programs are those that give users more power. And although a program may have scads of powerful commands, the most powerful programs are those which let users put the commands together in new ways—in effect, to write programs.

Instead of forcing you to always issue commands one at a time, a programmable application lets you create a script of commands to customize the behavior of the program. Whether we're talking about word processing macros, spreadsheet templates, relational database languages, or advanced machine language, programmability is the real key to software power. If you feel limited by a certain range of commands, you can combine the commands in new ways to create personalized features, just as we combine the vocabulary of English words to create a wealth of literature. Why just read when you can write?

Scripts, Sequences, and Batches

AmigaDOS is more than just a disk operating system—it's a programmable system that can process lists of its own commands as well as individual commands. In effect, AmigaDOS is a simple disk-oriented programming language.

A list of AmigaDOS commands can be stored in a disk

file variously known as a *script*, a *sequence*, or a *batch file*. The term *batch file* is most commonly used by those who work with PC-DOS, MS-DOS, and CP/M, which are also programmable disk operating systems. To keep things straight, we'll use *batch files* synonymously with *scripts* or *sequences*.

Even if you don't program in BASIC or any other language, you may be interested in learning about AmigaDOS batch files. The batch file "language" is simply made up of the same AmigaDOS commands you've probably been using all along. (For information about AmigaDOS see *COMPUTE!'s AmigaDOS Reference Guide* by Sheldon Leemon and Arlan R. Levitan, or *COMPUTE!'s Amiga Programmer's Guide*.) There are also a few AmigaDOS commands designed especially for batch files.

Creating and running batch files is easy. Using a text editor, you just type in a list of AmigaDOS commands. Then you save the list on disk under a filename. To run the batch file, you type EXECUTE *filename* at an AmigaDOS prompt. AmigaDOS reads the batch file and executes the list of commands, just as if you had typed them one by one yourself.

We won't cover some of the more advanced features of batch files, useful only to advanced C and machine language programmers. Instead, we'll concentrate on the everyday utility of batch-file programming.

A Quick Example

In a moment, we'll show how to create batch files with ED, the AmigaDOS full-screen text editor, but first there's a simpler way to create a short batch file (see the previous article or more complete instructions for using ED). Enter this line at an AmigaDOS prompt:

```
COPY * TO Hello
```

(Note that AmigaDOS commands can be entered in uppercase or lowercase.)

Although nothing seems to happen, AmigaDOS is waiting for you to enter some lines. We'll use the ECHO command to display a friendly message. ECHO displays any text that follows it within quotation marks, just like the PRINT statement in BASIC. One difference is that if you want to ECHO only a single word, the quotes aren't necessary.

At an AmigaDOS prompt, enter the following text, pressing RETURN after each line:

```
ECHO "Hello!"
ECHO "I am your friend, the Amiga"
ECHO "personal computer."
```

After the last line, press CTRL- \. The \ key is the one to the left of the BACKSPACE key. CTRL- \ tells AmigaDOS that you're finished, and that it should finish writing and close the file. This key represents EOF, for End Of File.

To confirm that you've typed the file correctly, enter

```
TYPE Hello
```

You should see the same lines you typed. Now you can start this simple program:

```
EXECUTE Hello
```

This should print on the screen:

```
Hello!
I am your friend, the Amiga
personal computer.
```

Using ED

It would be nice to have the Amiga actually speak this greeting. Rather than type in a whole new file, we'll use ED, the screen editor, to make the simple changes we're interested in. Enter

```
ED Hello
```

This runs ED and also loads the batch file named Hello. When you start ED, you can give it the name of any file to edit. If the filename doesn't exist, it will be created; otherwise the file is automatically displayed on the editor screen.

We'll make the Amiga speak the ECHO messages aloud by taking advantage of the system's built-in speech synthesis via the AmigaDOS SAY command (added to AmigaDOS version 1.1). To learn more about SAY, just enter SAY by itself to enter an interactive mode with onscreen instructions.

After you start ED by typing ED Hello, the batch file we previously entered should be on the screen, with the cursor at the beginning of the first line. ED is a full-screen text editor, so you can move the cursor anywhere within the file (but not

past the last line). To insert some text, just start typing. The DEL and BACKSPACE keys can be used to delete characters.

Move the cursor to the second ECHO line and press RETURN. This inserts a blank line. Cursor up to the blank line and enter

SAY HELLO!

You don't need to press RETURN at the end of the line, since you already did this to open up a line for typing.

Now cursor to the end of the file and type:

SAY I am your friend, the Amiga personal computer.

(Notice that SAY is the only AmigaDOS command that doesn't require you to enclose text containing spaces with quotes.) This is how your screen should look:

```
ECHO "Hello!"
SAY Hello!
ECHO "I am your friend, the Amiga"
ECHO "personal computer."
SAY I am your friend, the Amiga personal computer.
```

With the cursor at the end of the file, press the ESC key. An asterisk (*) should appear. Press the X key, then RETURN. This exits ED and saves your changes back to disk.

Finally, type EXECUTE Hello to try out your talking batch file.

Although these techniques are sufficient for simple editing, ED has dozens of editing commands. For example, CTRL-B (press CTRL and B at the same time) blanks out and deletes the line the cursor is on. ESC-J RETURN (press the ESCape and J keys; then press RETURN) joins two lines together. For a list of ED commands see the previous article.

Startup-Sequence

A special AmigaDOS batch file, called the *startup-sequence*, is executed automatically when you boot up an AmigaDOS or Workbench disk by inserting it at the Workbench prompt. Startup-sequence normally just displays a message, then launches the Workbench and ends the command line interface.

To edit this batch file, enter:

ED s/startup-sequence

This runs ED and calls up the file "startup-sequence" from the S subdirectory. This subdirectory, which can also be accessed as the S: device, is a convenient place for batch files. Just as AmigaDOS, by default, searches for AmigaDOS commands in the C subdirectory, the EXECUTE command first looks for a batch file in the S subdirectory. If AmigaDOS can't find the batch file in this subdirectory, it looks for it in the current directory. So no matter what your current directory is, you can always use your batch file if you place it in the S directory on your startup disk.

When you first load startup-sequence into ED, you'll see something like this:

```
ECHO "Workbench disk. Release 1.1"
ECHO " "ECHO "Use Preferences tool to set date"
ECHO " "
LoadWb
endcli > nil:
```

Since this message appears every time you start up your disk, you may want to change the ECHO statements for a personalized message. Likewise, if you'd rather use AmigaDOS instead of the Workbench, delete the last two lines. The "> nil:" sequence makes AmigaDOS throw away the output of a command; here, it's the message "CLI task 1 ending."

Startup-sequence is a good place to put personalized commands. For example, if you like to keep your command directory in RAM for speed and convenience, you could insert these lines above the LoadWb line:

```
makedir ram:c
copy c to ram:c all quiet
cd ram:c
```

This copies all of the AmigaDOS commands from the C subdirectory on the floppy disk into a C subdirectory on the RAM disk. It also changes the current directory to the C subdirectory in RAM:, so any AmigaDOS commands you type from that point on will be loaded from RAM: instead of from the floppy. In effect, this turns AmigaDOS into a memory-resident DOS, with all commands intrinsic instead of extrinsic. AmigaDOS responds much faster this way. However, this also uses up quite a bit of memory, so you may want to copy only the commands you use frequently.

Another useful startup action is to set the date and time. You can always do this with the Preferences tool or by opening a CLI and using the DATE command. However, it can be more convenient to enter the date when you first turn on your Amiga, allowing all files subsequently saved to be stamped with the current date and time. Just insert this line into startup-sequence:

```
DATE ?
```

The ? operator can be used in place of the parameter of a command. Instead of specifying the date, ? prompts the user to enter the date. It also displays the template for the DATE command (TIME,DATE, TO=VER/K:).If you like, use ECHO to display your own prompt, and > nil: to discard the template:

```
ECHO "Please enter the date and time."
ECHO "DD-MMM-YY HH:MM:SS"
date > nil: ?
```

From then on, whenever you boot up from this disk, you'll respond to the prompt by typing something like this:

```
27-jan-86 15:12
```

which automatically sets the system clock.

Variable Parameters

You can also send special options to your batch file. You enter these options on the command line along with the EXECUTE command. Just as with variables in BASIC, you can manipulate these parameters symbolically.

Let's say you'd like a batch file that gives you complete information on a file. It uses LIST to display the information about the file, and TYPE to display the file. You would use a command like EXECUTE SHOW RODEO to display the file RODEO. Use ED SHOW or COPY * TO SHOW to create this batch file:

```
.KEY name
LIST <name>
TYPE <name>
```

.KEY (don't forget the leading period) sets up a name for substitution text. Whatever you typed on the same line with EXECUTE is substituted wherever you use <name>. You must

use the angular brackets, or LIST and TYPE would look literally for the file "name."

After creating this batch file, type this at an AmigaDOS prompt:

EXECUTE SHOW S/STARTUP-SEQUENCE

The result is the same as if you had typed LIST S/STARTUP-SEQUENCE followed by TYPE S/STARTUP-SEQUENCE.

Other AmigaDOS commands let you check to see if the user has entered a specific string and check to see if a file exists. To prevent an error message, we can check to see whether the file exists before we use LIST and TYPE:

```
.KEY name
IF EXISTS <name>
LIST <name>
TYPE <name>
ELSE
ECHO "<name> does not exist!"
ENDIF
```

Notice the use of IF, ELSE, and ENDIF. Looks like Amiga BASIC, doesn't it? In fact, the AmigaDOS IF-ELSE-ENDIF commands function very much like BASIC's. When the IF condition is true, AmigaDOS executes the following statements; otherwise the following statements are ignored. ELSE executes the statements following it only if the preceding IF was false. ENDIF cancels conditional processing and returns to executing all commands.

Any Parameters Missing?

Here's how to use the IF EQ option to test for the existence of a command-line parameter. If there is no parameter, <name> is null, so "<name>z" is simply "z". We use NOT to reverse the test. If the parameter "<name>z" is NOT equal to "z", then we must have a command line parameter. (We can't just test IF <name> NOT EQ " ", since EQ requires two parameters, and the null string " " is not a parameter, but the absence of one.)

```
.KEY name
IF <name>z NOT EQ z
LIST <name>
TYPE <name>
```

```
ELSE
ECHO "You didn't give me anything to
 SHOW."
ENDIF
```

Although you can't use leading spaces in the actual batch file, it's easier to follow the IF-ENDIF structures when you use indentation. Just don't type in the leading spaces. This version of the batch file SHOW checks both for the existence of the filename and for the presence of the filename parameter:

```
.KEY name
IF <name>z NOT EQ z
  IF EXISTS <name>
   LIST <name>
   TYPE <name>
  ELSE
   ECHO "<name> does not exist!"
  ENDIF
ELSE
  ECHO "You didn't give me anything
  to SHOW."
ENDIF
```

You can use more than one parameter in the .KEY statement, just as many commands, such as DATE, accept two inputs.

If the user doesn't enter anything for the parameter, you can assign a default value using either .DEF or $. If you use .DEF, the default phrase is used throughout the batch file. In this example, SHOW displays itself if you don't give it a filename.

```
.KEY name
.DEF s/show
LIST <name>
TYPE <name>
```

You can use $ to substitute a default value only for the current substitution. Several batch commands may use the value in different ways, so each command may have its own default value. In the following example, LIST displays the whole directory if <name> is null, but TYPE types the file "TEMP" if <name> is null:

```
.KEY name
LIST <name>
TYPE <name$temp>
```

Labels and Branching

You can jump forward to a label with the SKIP command. You'd typically use SKIP along with an IF condition if you want to skip over a block of statements that shouldn't be executed if the IF was true. You declare the label with LAB. The SKIP command can't skip backward, only forward to a LAB statement. You can usually use IF and ELSE, though, to accomplish the same thing.

```
.KEY name
IF exists <name>
TYPE <name>
SKIP ToMyLou
ENDIF
ECHO "<name> doesn't exist."
LAB ToMyLou
ECHO "Finished."
```

An EXECUTE command can execute another batch file, or even itself. This permits backward looping to some degree. Nested batch files can be quite handy. You can test and debug individual batch programs, then execute them together from a master execute script:

```
EXECUTE Greeting
EXECUTE GetDate
EXECUTE Assignments
```

The individual files could themselves contain other EXECUTE references.

ASSIGNing Shortcuts

If you're using EXECUTE a lot, you may grow weary of typing it. You can always rename EXECUTE to something short like x, but other batch programs may contain EXECUTE statements, requiring you to rename it again. Instead, you can use the ASSIGN command to assign any filename to a device name.

```
ASSIGN x: sys:c/EXECUTE
```

You can now use x: whenever you want to use the EXECUTE command. (The prefix *sys:c/* makes sure that EXECUTE can be found no matter what directory you're in.)

The device name you create should not conflict with an existing one. To get a list of the current assignments, just type

ASSIGN. You may want to ASSIGN d: c:list for a convenient and quick shorthand for directories (c: is synonymous with the C directory). You can then just type d: to get a LIST.

ASSIGN can be so handy for this kind of thing that you'll probably want to include your own sequence of ASSIGN commands within startup-sequence. If you put your ASSIGN statement within startup-sequence, you'll get these assignments for every session. Just remember that ASSIGN can only be used to attach a device name to a particular filename. ASSIGN d: "c:list quick" doesn't seem to work. Although LIST is a filename in the c directory, the "quick" parameter is not part of the filename.

Printers for the Amiga

Charles Brannon

A printer is one of those optional but essential add-ons for your computer. It lets you reap something tangible from your word processor, terminal program, spreadsheet, or drawing program.

Most serial or parallel printers will work with the Amiga. The Amiga sports an RS-232 serial port as well as a standard parallel printer port. All it takes is the right cable to link the Amiga with almost any printer.

The commonly available IBM printer cables appear similar to Amiga cables except that the end of the cable that plugs into the IBM is a DB-25 male connector and the Amiga port is also a DB-25 male connector. Since printer cables specifically for the Amiga can be difficult to obtain, you might be tempted to use a "gender-changer" (a box or cable with a male connector on one end and a female connector on the other) to connect the IBM cable to your Amiga. *Don't do this.* Such an arrangement could damage your Amiga or your printer, or both.

The Amiga parallel port does not use exactly the same pin assignments as the IBM port. (Refer to page 7-13 of the *Introduction to Amiga* manual for a pinout chart.) Even more important, pin 23 on the Amiga parallel port is a +5-volt power supply, while pin 23 on an IBM-type printer cable may be connected to voltage ground. If the cable carries this voltage, and if the printer connector has a grounded pin at that position, the power supply in your Amiga may be damaged.

If you have a serial (RS-232) printer connected via the Amiga's serial port instead of the parallel port, a similar caution applies: Pins 14, 21, and 23 on the serial port carry power supply voltages. (Refer to page 7-12 of the *Introduction to Amiga* manual for a pinout chart.) Since these pins are often unused in devices like modems and printers, it *may* be safe to use IBM-type serial printer cables. Check the manual for your

printer carefully to be sure that your particular model does not make any connection to these pins. Again, a gender-changer plug will be required to attach an IBM-style cable to the Amiga serial port. It's best to check with your dealer before using a suspect cable.

Printer Drivers

Once you've hooked up the hardware, you need to "attach" the printer to your software. Although every printer manufacturer uses different specifications for software control over printing features, the Amiga is capable of adapting to a variety of popular printers.

What complicates things is that every printer has its own unique set of codes, even for common effects such as underlining or boldfacing. For example, the Epson MX-80 uses the ASCII sequence 27-53 ("ESC-4") to turn on italics mode, and 27-54 ("ESC-5") to turn off italics. On the other hand, the Okimate 20, which is similar in many other ways, uses the sequence 27-37-71 (ESC-%G) to turn on italics, and 27-37-72 (ESC-%H) to turn italics off.

When an Amiga program wants to print italics, it can't just use the code for one printer model, because the program would be incompatible with other printers. Instead of sending the actual code for italics, then, Amiga programs send a symbolic code for italics. If you tell the Amiga which *printer driver* to use, the driver translates these symbolic codes into the actual codes for your printer.

Use Preferences to install your printer driver, following the instructions given in *Introduction to Amiga,* pages 7-6 to 7-11. Printer drivers currently exist for the Alphacom Alphapro 101 (no longer in production); Commodore CBM-MPS 1000; Epson FX-80, RX-80; HP LaserJet/LaserJet Plus; Brother HR-15XL; Diablo Advantage C-150, D25, 630; Qume LetterPro-20; and Okimate-20. If your printer is not on this list, try some of the drivers to see if they work with your printer. For example, the Juki 5510 dot-matrix printer is Epson JX-80–compatible, so you can use the Epson JX-80 printer driver.

If none of the drivers work, select the Custom printer driver. If you have the version 1.1 operating system upgrade, by default the Custom selection looks for a printer driver named Generic. The Generic driver works with any properly cabled printer by ignoring all special printer codes. If your

printer won't respond to the codes used by any of the printers on the list, you can at least get a plain-vanilla text printout with the Generic driver.

Unfortunately, the Generic driver won't let you use any special printing effects such as underlining, boldface, italics, or bit-image graphics. You need a printer driver created especially for your brand of printer. Many people are working on drivers for unsupported printers, including one company that has developed a printer-driver builder that a nonprogrammer can use to design a new custom driver. Nevertheless, if you are looking for a printer for your Amiga, it's best to buy one that is compatible with one of the above printers.

Programming in Modula-2

Charles Brannon

There are a plethora of programming languages for the Amiga, giving programmers and developers a wide choice of programming styles and systems. There are various versions of BASIC, C compilers, macro assembler/editors, versions of Pascal, and even an implementation of LISP. Numerous programming tools, such as editors and debuggers, are also available.

A relative newcomer to the scene, TDI's *Modula-2*, is now getting some attention. Some programmers consider it easier to learn and use than C, since it shares many of the high-level aspects of Pascal while still retaining a machine-level interface for maximum efficiency.

Modula-2 is a descendent of the language Modula, which in turn is a descendent of Pascal. Nicklaus Wirth, the inventor of Pascal, designed Modula from the roots of Pascal, but purposely kept it very simple so that it could be used with very small computers—primarily for controlling hardware devices such as robot arms. The original Modula had little application outside a very specialized world, so Wirth put back most of the features of Pascal to create Modula-2. TDI has worked directly with Wirth to implement versions of Modula-2 for the Amiga and Atari ST.

Software Chips

The concept of Modula-2 is echoed in its name. It's a language designed specifically for the techniques of modular programming, just as Pascal was designed to make structured programming convenient and elegant. Modular programming—the art of breaking a large, complex problem into small, independent tasks—is at the heart of all programming, but Modula-2 tries to bring to software the modularity inherent in computer hardware, based on off-the-shelf chips and components. With "software chips," Wirth envisioned, software technology could

advance apace with the remarkable speed of hardware evolution.

If software chips are possible, they have to be based on program modules that can be truly independent, and hence, individually testable. You can compile a module without having to recompile the entire program. A module, once developed, becomes a "black box" routine that accepts input and/or provides output. You no longer need to know how this module works internally to use it—you just plug it in and go. Writing a program becomes a task of putting together these building blocks in the right way without ever needing to reinvent the wheel. Why solve a problem when someone else has already found the solution?

TDI's *Modula-2* comes with a standard library containing modules for input/output, math routines, and access to special machine features. You use only the routines you're interested in, and only these routines (and the underlying routines they are based on) need to be included in your compiled code. This lets you control the size of your final program.

You can easily add your own library modules. First, you write the *definition module*, which simply contains the procedure headers that specify the inputs and outputs of a module. The definition module primarily specifies the names of these procedures. It compiles to a symbol file for use by the compiler. The *implementation module* contains the actual code of the module. You compile the definition module separately from the implementation module.

You can change and recompile the implementation module without changing the definition module, as long as your procedure headings remain the same. When you're referencing library modules, the compiler can check the compact, compiled symbol file rather than the full-length definition module, speeding up compilation. After compilation, a linker combines your main program with the compiled implementation modules to create the final executable program.

Reminiscent of Pascal

One of the best ways to learn about a language is to study an example program. The program accompanying this column is written to demonstrate some of the features of Modula-2 without getting bogged down in tricky algorithms. It's a simple "guess my number" game. The RandomNumbers module thinks of a number from 1 to 100. The program then gives you ten

tries to guess the number, helping out with hints. If you guess too high, the program recommends that you try a smaller number. If you guess too low, you should try a higher number.

Here's how the program works. The first line declares the name of the module. Next, the IMPORT statements specify which external library calls we'll be using. Then we declare the variables. We define the procedure SkipEOL, used to strip away the rest of a line after getting a single-character response. The main loop follows, enclosed by the keywords BEGIN and END. (All Modula-2 keywords must be typed in uppercase, which can be annoying.)

Most of the program looks very much like Pascal, especially the use of := for assignments and the required semicolon at the end of each logical line. Also, you won't find GOTO anywhere in this or any Modula-2 program. Instead, you can control looping and program execution with statements like LOOP–EXIT–END, WHILE-END, REPEAT–UNTIL, and IF–THEN–ELSE–END.

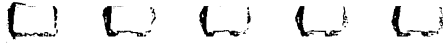
You might be interested to know that this program compiles in 35 seconds when the source code is stored in the RAM disk; it takes 37 seconds to compile when the source code is stored on a floppy disk. Linking takes 45 seconds from the RAM disk, and just one minute from a floppy disk. This is quite a bit faster than *Lattice C* and compares well with *Aztec C*.

There's much more to Modula-2 than this discussion can encompass. The language even permits procedures to run as multitasking programs. Our example doesn't show how easily Modula-2 can take advantage of the Amiga operating system—even a small program would be too large to demonstrate here—but the interface is similar to C's, using Pascal-style RECORDs instead of C structures. It's possible to develop modules that support the Amiga operating system on a higher level, using calls like Screen(320,200,5) to open a custom screen, as opposed to filling in the blanks of a NewScreen structure, opening the Intuition library, and calling OpenScreen( ). Some high-level modules are included in the library. When these modules are developed and shared between Modula-2 programmers, Amiga programming in Modula-2 can seem almost as easy as in BASIC, but with every advantage of a modern compiled language.

Guess the Number

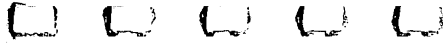
```
MODULE Example;

FROM InOut IMPORT EOL,Read,ReadInt,ReadString,
     WriteLn,WriteString,WriteInt;
FROM RandomNumbers IMPORT Random;

VAR
 MyNum, Guess, Tries : INTEGER;
 Again : CHAR;

(* Skips until end of line is reached *)
PROCEDURE SkipEOL;
  VAR temp : CHAR;
BEGIN
  REPEAT
    Read(temp);
  UNTIL temp=EOL;
END SkipEOL;

(* The main loop *)
BEGIN
  LOOP
    MyNum := Random(100)+1;
    WriteString("I'm thinking of a ");
    WriteString("number from 1 to 100");
    WriteLn;
    Tries :=0;
    LOOP
      Tries := Tries+1;
      IF Tries>10 THEN EXIT; END;
      WriteString("Guess #");
      WriteInt(Tries,2);
      WriteString("? ");
      ReadInt(Guess);
      WriteLn;
      IF (Guess=MyNum) OR (Guess=0) THEN EXIT; END;
      IF Guess<MyNum THEN
        WriteString("Try a larger number.");
      ELSE
        WriteString("Try a smaller number.");
      END; (* IF *)
      WriteLn; WriteLn;
    END; (* LOOP *)
    IF Tries>10 THEN
      WriteString("You only get 10 tries!");
    END; (* IF *)
    IF Guess=MyNum THEN
      WriteString("You guessed my number!");
```

```
      WriteLn;
      WriteString("After ");
      WriteInt(Tries,2);
      WriteString(" tries.");
    END; (* IF *)
    WriteLn; WriteLn;
    WriteString("Play again? (Y/N): ");
    Read(Again);
    SkipEOL; (* skip ahead to next line *)
    IF (Again='N') OR (Again='n') THEN EXIT; END;
  END; (* LOOP *)
END Example.
```



CHAPTER FIVE

Telecommunications



Getting Online

Charles Brannon

Your Amiga can be a terminal to other, much larger computers all over the world. Included here is a quick overview of the basics of telecommunications with your Amiga.

Any computer can become an information appliance with the addition of a modem. Hayes-compatible 1200-baud modems can be bought for under $200 now. You may find one small complication when connecting a modem to your Amiga. When purchasing a cable to connect the modem to your Amiga, you must pay close attention to the types of plugs on the cable. The Amiga serial-port connector—where you plug in the modem cable—is the gender opposite that of the IBM serial port. (The Amiga port uses a female connector while the IBM uses a male.) Since IBM-style modem cables are more common than Amiga modem cables, you may find it simpler to use an IBM cable with a *gender-changer* module. I'm using one with my Amiga at home. A gender-changer is a small box that attaches to the female plug on the end of the modem cable, terminating in a male connection that plugs into the female connector on the Amiga. Be aware, though, that there is voltage on pins 14, 21, and 23 on the Amiga port, although these pins are not normally used in most RS-232 cables. Check your modem manual to make sure these pins are not connected or grounded on your modem's connector.

When using a direct-connect modem, you are required to call your local phone company to register the modem, as it becomes part of the phone system when you plug it in. Have at hand the FCC registration and ringer-equivalence numbers, usually found on the bottom of the modem or in the manual.

Next comes terminal software. In its simplest form, this is a program that monitors the modem for input, which is displayed on your screen, and checks the keyboard for your typing, which is sent out over the phone lines. The Amiga BASIC Extras disk contains a simple terminal program in the Basic-Demos folder. More complex terminal programs allow you to

transmit a file (uploading) or store incoming data to disk (downloading). "AmigaTerm," included in this chapter, is a powerful terminal program written in BASIC. It includes most of the features most users want.

Error-Free and Automatic

Programs such as XMODEM allow error-free file transmission. XON/XOFF allows either computer to pause when necessary without missing any characters. Advanced modem software lets you create scripts to automate the process of calling a remote computer, entering your password, and seeking and downloading information—even if you aren't there to monitor your computer.

What can you do with a modem? First, you can call up local bulletin boards, including Amiga-specific ones. This is a great way to meet people. Public bulletin boards offer services where callers discuss everything from the nuts and bolts of computing to controversial political issues. Usually, there are also public domain programs for you to download. It's expected you'll upload some of your own programs in exchange.

Then there are the commercial information services such as CompuServe, The Source, Delphi, and GEnie. These services provide information such as stock quotes, daily news/weather/sports, and online encyclopedias and books. Via electronic mail, you can send and receive letters directly over the phone. Most of these services let you play games with other users. The popular CB simulation allows dozens of callers to talk via keyboard in a conversational free-for-all. You can also shop by phone, make airline and ticket reservations—even buy and sell commodities.

Always a popular part of these services is the forum specific to your machine. All these services have Commodore or Amiga forums, containing databases of the most popular public domain software. The forums allow you to exchange messages with other members. It's like belonging to an electronic user group. It's a great way to get help with a problem—just send a question and you'll likely be surprised by how many answers you get.

The Twenty-First Century and Beyond

Perhaps the most powerful option you have with an *auto-answer* modem—one that can pick up the phone and establish a connection automatically when called by another modem—is to set up your own bulletin board. You can buy bulletin board software or download public domain programs to help manage your own information service. You are the host here, providing your time and equipment to set up a local communications network. Callers will download software and expect to find interesting things to download. Of course, you must insure that you offer only noncopyrighted, public domain software on your board. If in doubt, leave it out. (Programs published in most magazines and books, including this book as well as *COMPUTE!* magazine, are *not* in the public domain.)

Technology is now significantly expanding our communications; we live in an age where we can have our own computers and hook them into a global intelligence net, offering the greatest possibilities yet for personal expression and free choice. Although there are limitations, telecommunications offer us a hint of what life will be like as the global village becomes a reality in the twenty-first century, and beyond.

Cutting Telecommunications Costs

Kathy Yakal

Learning to keep your telecommunications costs as low as possible is one of the secrets of online success. Here are a few tips to help you conserve your money while still enjoying the pleasures of telecomputing.

There's probably no other personal computer application that can be as costly overall as telecommunications. The initial costs are low: a modem, a cable, perhaps, and terminal software. And if you limit yourself to calling local electronic bulletin board systems (BBSs), your expenses can end there.

An Exciting and Varied World

But it's a rare computer owner who can resist moving out to explore the exciting and varied world of telecommunications. Your first taste of online activity usually leads to the desire to find out what's happening online across the country. So you start calling out-of-state BBSs, and maybe subscribe to an online service or two. Soon, you're facing startup subscription fees, monthly service and hourly online charges, and steadily climbing telephone bills.

However, there are ways to economize online.

Familiarize yourself thoroughly with whatever system you're on before attempting to accomplish anything there. Obviously, familiarizing yourself with the system is accomplishing something, but don't even attempt extensive online chatting before you understand the command system and menu structures. Fortunately, most systems offer a lot of help in this area, and encourage the user to spend some time getting acquainted. For example, the Delphi telecommunications system requires each new user to go on an online tour at the

first sign-on. QuantumLink, a Commodore-specific service, offers guided tours to new users at regularly scheduled times. In fact, most of the major telecommunications networks attempt some sort of introductory orientation for new users, whether it's through written instructions or online tours.

When you're using a system where this kind of automatic help isn't available, it's a good idea to download (have sent from the host computer to your disk drive) Help menus (usually accessed by typing H or a question-mark symbol at a command prompt), and later print them out and study them offline (disconnected from the host) until you know them well.

These menus often contain detailed explanations of each command, and give you a good idea of the quickest, easiest ways to get around the system. It may seem a lot of trouble, but learning them will save enormous frustration, time, and money. The system operators (sysops) at each network are usually available if you really get stuck, and most are quite happy to help. But remember, if you're trying to get help while online, you're generally paying for the connect time.

Once you've learned a system fairly well, consider using its Expert mode. Most systems, and even many BBSs, offer a mode for experienced users that allows them to bypass many of the menus and go straight to the desired area. You need to be sure you know your way around well before you start using this, or you could find yourself locked out of the menu structure and unable to go anywhere.

If you do a lot of downloading, seriously consider getting a 1200-baud modem. While 300 baud is a good speed for socializing, it can be frustratingly slow when you simply want to download a program into your computer. At the same time, 1200 baud is often too fast for chatting, if several people are participating online. There's no simple formula to help you determine whether or not the savings from fast downloads will justify the expense of a 1200-baud modem. While it's true that you're getting the information four times faster, most systems have a higher hourly charge for 1200-baud use. In the long run, however, computer users who opt for 1200-baud service generally don't choose to return to 300-baud.

Speed up your log-on time by using a more sophisticated terminal program. Many terminal programs let you create macros, small user-definable routines that set up an

automatic log-on procedure. If there is one area where you always go first, or one task you always perform (such as checking mail), you can add that to the macro and save some time and keystrokes. Here again, the savings may or may not be worth the extra expense of a new terminal program. But the extra convenience may play a part in your decision.

Try to confine the bulk of your downloading to times when the system is relatively quiet. Systems that operate on a 24-hour basis charge lower rates for off-peak hours (evenings and weekends), thus offering substantial savings. But even off-peak hours are busier at some times than at others, usually from about 8:00 until 11:00 in the evening. At those times, a system sometimes suffers from short delays, pauses between the time you type commands and the time they're executed. You'll save some money if you steer clear of those hours.

If telecommunications at off-peak hours, such as 2:00 in the morning, is impossible, there are programs that will automatically log you on to a system at a specified hour, do the tasks you've assigned them, and log you off when they've finished. This doesn't necessarily require you to leave your computer on all night. If your computer can be set to boot up automatically when the power comes on, you can leave your disk in the drive and get an automatic timer that will turn the computer on and off at predetermined hours.

Consider shopping for a new long distance telephone service that may have lower rates than your present system. This won't make any difference if you only call the major telecommunications services and live in an area with local-access numbers for services like Tymnet and Telenet, which act as connectors to the telecommunications services. But if you're calling a lot of BBSs long distance, you might be able to reap some fairly significant savings if you switch to a more economical long-distance service.

If you're downloading messages at 1200 baud, dump all of them to disk and search through them later. Searching through messages and deciding which ones you want to keep can be quite time-consuming. If you're at 1200 baud, it might actually save online charges to dump a whole group of messages without stopping to read them and deciding which to save and which to discard. After you've logged off, you can go through the file and keep only the ones you want.

Set an alarm clock next to your computer. This may sound rather silly, but it's easy to lose track of time when you're online, especially in your first few weeks of telecomputing. Even if you don't feel you need to set absolute limits for yourself, it will alert you as to when a set period of time has gone by. Some terminal software includes an alarm clock function.

There are no hard-and-fast rules when it comes to saving money online. The more experience you get in telecomputing, the more efficient you'll become. You'll also find that all of the telecommunications networks are trying to offer ways to increase their subscriber base and their percentage of online usage. As a part of this effort, rates are getting less expensive, systems are becoming easier and faster to use, and there are more services being offered within each network. Increasingly, the happy result is more telecomputing for the money.

AmigaTerm

Philip I. Nelson

With "AmigaTerm," a comprehensive telecommunications program for the 512K Amiga, you can communicate with other computers, call commercial information services, and even upload and download files. Written in Amiga BASIC, the program gives you full control over all RS-232 parameters and includes power features such as autodialing, macros, and configuration files. A modem is required.

Telecommunicating is an increasingly popular use for personal computers. With a modem and a personal computer, you can access thousands of public domain programs, dial up the latest stock market quotes, and exchange messages, electronic mail, and even a favorite recipe with other people many miles away. The information bonanza is going strong. But computer communications require a telecommunications program, often known as a *terminal* program. "AmigaTerm" is a terminal program that provides all the features you need to start telecommunicating with the Amiga. It has many of the same capabilities offered by commercial programs, but it's written entirely in Amiga BASIC, in a modular style that makes it easy to expand to add even more features.

Type in the program and save a copy to disk. Before you run the program, you must also copy the file named graphics.bmap from the Basicdemos drawer (subdirectory) of your "Extras" disk onto the disk where you saved AmigaTerm. This file must be present on every disk that contains AmigaTerm (it's needed for the program's special graphics effects). Don't copy graphics.bmap into a drawer on the disk, even if AmigaTerm itself is within a drawer.

Running AmigaTerm

AmigaTerm is entirely menu-driven for maximum convenience. You can select every program option from onscreen menus with the mouse pointer. When you run the program, it

spends a few moments initializing; then it displays a welcome message. When the welcome appears, AmigaTerm is ready to use. Make sure that your modem is connected to the computer and turned on before you proceed any further.

If you wish to call a commercial information service, simply dial the service and log on. AmigaTerm defaults to the communications settings used by most commercial services (see below). To access any of the menu options, hold down the right mouse button and move the pointer to the menu of your choice. A drop-down menu appears, displaying that menu's options. To select an option from the menu, move the pointer down and release the button when the option is highlighted.

When using AmigaTerm, you'll notice that program messages and prompts are displayed in dark, boldface characters, while ordinary text—incoming as well as outgoing information—is displayed with normal characters. This is done to highlight program information and prevent confusion when you are communicating with another computer system.

AmigaTerm offers many different options. Let's examine each of its menus in turn, beginning with the *Settings* menu.

Settings Menu

Before you can communicate with another computer, you must make sure that both machines are speaking the same "language." For telecommunications, that language is the RS-232 standard. AmigaTerm's *Settings* menu gives you control over all the important RS-232 settings. We'll explain them briefly here. If you're not familiar with these terms, see "RS-232 Standards" later in this article.

When you activate the menu with the right mouse button, the *Settings* menu displays the current settings for the baud rate, parity, word length, stop bits, and duplex mode. To change a setting, move the pointer down to that setting and release the button when the desired option is highlighted. AmigaTerm displays the new setting on the screen in boldface type as a reminder of the change. An explanation of each RS-232 parameter follows.

Baud rate. AmigaTerm can communicate at either 300 or 1200 baud. This option (like most others in the *Settings* menu) toggles between the two available settings. If the baud rate is 300, selecting the option changes it to 1200, and vice versa.

AmigaTerm defaults to 300 baud, the lower of the two speeds.

Parity. AmigaTerm defaults to even parity. This option allows you to select even, odd, or no parity.

Word length. AmigaTerm uses a default word length of seven bits. You can also change the word length to eight bits.

Stop bits. The default setting for this option is a stop bit of 1. It should almost never need to be changed. For special applications, the stop bit can be changed to 2.

Duplex. Full duplex, the default setting, is most commonly used for calling bulletin boards and information services. You may also select half-duplex mode for situations in which the remote computer does not echo the characters you are transmitting.

Special Menu

The *Special* menu controls two convenience features: automatic phone dialing and configuration files. Of course, the autodialing feature works only if your modem is capable of autodialing. An explanation of the options in the *Special* menu follows.

Phone number. When you select this option, AmigaTerm displays the current telephone number inside a custom dialog window. To enter a new number or change an existing one, simply type in the number when the dialog window appears. To exit without changing the number, press RETURN without typing anything.

Dial. To dial the current number, select the *Dial* option. AmigaTerm transmits a Hayes-format autodial command, followed by the current telephone number.

Save settings. This important option allows you to save all of the current RS-232 settings, plus the current phone number and macros (see below), in a disk file for future use. Once a configuration file has been saved, you can reconfigure AmigaTerm at any time simply by loading the file. This is particularly useful if you regularly call more than one information service.

To save the current settings, select the *Save Settings* option from the *Special* menu. AmigaTerm opens a dialog window and prompts you to enter a filename for the settings file (if you press RETURN without typing anything, AmigaTerm aborts the operation). Any legal Amiga filename may be used for the settings file. However, the name *setup* has a special

meaning in this program. When you run AmigaTerm, it automatically searches the root directory of the current disk for a file named *setup*. If it finds the file, the program loads the settings when it boots. This feature is very convenient if you call a particular bulletin board or information service regularly. As soon as AmigaTerm boots up, you're ready to dial the service. All you need to do is select the *Dial* option.

Load settings. This option loads a settings file into memory. Enter a filename in the dialog window when prompted (press RETURN to abort without loading a file). AmigaTerm always displays all of the current RS-232 settings in boldface when you exit this option.

Transfer Menu

The *Transfer* menu allows you to upload (send) and download (capture) a file. What you transfer is up to you—the file may contain a program, word processing document, personal message, or whatever.

Send file. To send a file, choose the *Send* option from the *Transfer* menu. AmigaTerm opens a dialog window and asks you to enter the name of the file you wish to transmit. To abort this function, press RETURN without entering a filename. If a disk error occurs, AmigaTerm signals an error and aborts the operation. Otherwise, it loads the file from disk and sends it out. Do not try to break out of the program while AmigaTerm is busy sending a file.The program lets you know when the transfer is complete.

Receive file. The *Receive* option of the *Transfer* menu permits you to capture a file sent from a remote computer. Select the option from the menu; then enter a filename in the dialog window. AmigaTerm opens a disk file of that name, then diverts all input to that file until you signal that the transfer is complete. To end the capture, select *Receive* a second time: AmigaTerm closes the disk file. No restrictions are placed on the type of information that you receive. AmigaTerm stores whatever it receives between the time you initiate the capture and the time you terminate it. AmigaTerm does not support error-checking protocols such as XMODEM or Kermit.

When capturing files, keep in mind that the Amiga's RAM: device, a native RAMdisk, is always available. To save a file to the RAMdisk rather than a floppy disk, simply include the prefix RAM: at the beginning of the filename, just as you

would include the prefix DF0: to specify drive 0 or DF1: to specify drive 1. Since the RAMdisk operates much faster than a floppy disk drive, this method eliminates the delays which otherwise occur when a long file is being written to disk. Once you have completed the capture, the file remains safely in RAM even after you've exited BASIC and returned to the Workbench. In version 1.2 of the Amiga's operating system (available only in Beta test form at this writing) the RAMdisk appears on the Workbench and its files can be manipulated with icons, like floppy disk files. In earlier versions of the operating system, you must open a CLI window and perform a COPY operation from AmigaDOS to transfer a file from the RAMdisk to a floppy disk. For instance, the AmigaDOS command COPY RAM:TEST TO DF0:TEST copies the file TEST from the RAMdisk to the disk in drive 0.

Macro Menu

The *Macro* menu, AmigaTerm's most advanced feature, makes often-used commands available in a convenient, onscreen menu. For instance, most bulletin boards require that you log on in some fashion by typing a password or user I.D. number. By storing such commands in a macro definition, you can transmit the desired sequence of characters by selecting the macro from an onscreen menu with the mouse—without having to type a single character. This saves time as well as minimizing typing errors. Since AmigaTerm saves macro definitions as part of a configuration file (see "Save Settings"), you can have as many as ten macros for each configuration file.

When you activate the *Macro* menu, ten macro selections appear beneath an option labeled *Edit Macro.* The *Edit* option is used to enter a new macro string or change an existing one. To see how it works, select *Edit Macro;* then enter 1 when it prompts you for a macro number. When the next prompt appears, enter a short string: It can be anything, since this is just for practice. When the dialog window disappears, AmigaTerm has recorded the macro. To transmit the macro, set the *Duplex* option of the *Settings* menu to half duplex (to insure that AmigaTerm echoes the transmitted characters on the screen); then select Macro 1 from the *Macro* menu. The macro string that you entered is simultaneously displayed on the screen and sent to the modem.

Automated Log-On

Macro definitions are automatically saved when you choose the *Save Settings* option. When you load a settings file, AmigaTerm loads that file's macros as well. This feature allows you to automate your telecommunications even further. To illustrate—say that you frequently call a fictional information service called ChompuSerf, and you ordinarily begin each session by entering the AmigaSIG area to read messages. After you've set the correct RS-232 parameters in the *Settings* menu and entered ChompuSerf's phone number with the *Phone* option, you can create several macros to take you all the way through the log-on procedure. Macro 1 is defined as CIS, the string that tells your local packet-switching network which service to call. Macro 2 is defined with your user I.D. number for ChompuSerf, and Macro 3 is defined with your password. Macro 4 is defined as GO AMIGASIG, the command that moves you from the top-level menu of ChompuSerf to the AmigaSIG section of the service.

Once everything is set as needed, save the settings, phone number, and macros in a single configuration file with the *Save Settings* option. When you reload the file in a later session, AmigaTerm is properly configured to communicate with ChompuSerf. You dial ChompuSerf with the *Dial* option. When the service comes online, you can reach your destination effortlessly by selecting the predefined macros from a screen menu. Macros not only save you the time and bother of typing repetitive commands, but decrease the risk of typing errors.

RS-232 Standards

Despite the enormous popularity of telecomputing, many computer owners—including some otherwise expert programmers—are intimidated by the unfamiliar jargon and seemingly technical nature of the process. In fact, there's nothing particularly mystical about communicating with another computer over the telephone lines. If you and the computer owner at the other end of the link are using terminal programs, it's usually just a matter of making sure that you both use the same baud rate and other RS-232 parameters. For those who are unfamiliar with this area, here is a fuller explanation of the RS-232 parameters which you can control in AmigaTerm.

Baud rate. This setting determines how fast you can communicate. Although the term *baud* is more frequently used, it's more accurate to refer to this parameter as *bps*, which stands for bits per second. For years, nearly all personal telecomputing was done at 300 bps due to the low cost of 300-bps modems. In recent times, the price of 1200-bps modems has dropped so dramatically that many bulletin boards and information services support either 1200 or 300 baud. The Amiga can support much higher baud rates, but Amiga BASIC is not fast enough to communicate reliably at speeds higher than 1200 bps.

Parity. The parity setting provides the computers at both ends of the link with a means of error-checking for each character which is transmitted. Fortunately, it's rarely necessary to worry about how this error-checking works. For commercial information services which communicate with a word length of seven bits, the parity is usually even. A setting of no parity is often used in communicating with a word length of eight bits. Odd parity is used infrequently.

Word length. The word-length setting indicates how many bits (binary digits, either a 1 or 0 value) are needed to make up one data word, which ordinarily represents one character of text. The bits are sent one by one, and the computer adds bits together to determine the code for the current character. For plain ASCII text, you can use either seven bits or eight bits, since seven bits is sufficient to express all the numerals, punctuation, and letters of the alphabet as standard ASCII codes. To send other types of information (for instance, a word processing document containing control codes), a word length of eight bits may be necessary. In many cases, the word length is not critical. However, it is very important that both computers use the same word-length setting. If you have established a link with the other computer, but the information appears garbled, try changing the parity and/or word length.

Stop bit. You will almost never have to change this setting, which provides the computer with a means for telling when it has reached the end of a character. A stop bit of 1 is nearly universal except for very slow baud rates.

Duplex. The duplex setting, called *echo* in some terminal programs, sounds mysterious but simply has to do with what your computer shows on the screen. Many information services automatically echo every character that you send to

them. If you type in GO AMIGASIG to enter the AmigaSIG area of the ChompuSerf information service, ChompuSerf sends back the characters G-O- -A-M-I-G-A-S-I-G one by one as you type them in. Among other things, the echoing lets you confirm immediately that the computer at the other end is receiving your transmission correctly.

When you set AmigaTerm to full duplex, it does not display characters which you send out—only those which it receives. This is the setting you should use for information services or any remote computer that echoes back what you send. When you select the half-duplex setting, AmigaTerm displays every outgoing character immediately. Half duplex is appropriate if you are communicating directly with a friend or in any other situation where the computer at the other end is not echoing your transmissions.

Generally, if you can't see what you type, you need to switch from full duplex to half duplex. Conversely, if AmigaTerm displays two copies of every character (GGOO AAMMIIGGAASSIIGG instead of GO AMIGASIG), you probably need to switch from half duplex to full duplex. Some modems also have a duplex switch for switching between half and full duplex. Under ordinary circumstances, the modem's duplex switch should be set for full duplex (no echoing) to avoid confusion.

AmigaTerm

The left-arrow symbols in this listing indicate when to press RETURN at the end of each program line. Do not attempt to type the arrows themselves.

```
'  AmigaTerm 1.Ø for 512K Amiga←
←
CLEAR ,65536&←
ON ERROR GOTO Ø←
GOSUB Setup←
←
Main:←
MenuChoice=MENU(1)←
ON MENU(Ø) GOSUB Menu1, Menu2, Menu3, Menu4, Menu5←
IF LOC(1) THEN x$=INPUT$(1,1):PRINT x$;:IF CaptureF
lag=1 THEN PRINT #2,x$;←
x$=INKEY$:IF x$>"" THEN←
IF x$=CHR$(27) THEN x$=CHR$(3)←
PRINT #1,x$;:IF Duplex THEN PRINT x$;←
END IF←
GOTO Main←
←
```

```
OpenUp:←
CLOSE 1←
Modem$="com1:"+STR$(Baud(Baud))+CHR$(44)+Parity$(Pa
rity)+CHR$(44)←
Modem$=Modem$+STR$(Length(Length))+CHR$(44)+STR$(St
Bit(StBit))←
ON ERROR GOTO TrapMe←
OPEN Modem$ AS 1 LEN=1←
ON ERROR GOTO Ø←
RETURN←
←
Menu1:←
MENU RESET←
WINDOW 1, , (Ø,Ø)-(616,186), 31←
CLOSE←
GOSUB Roman←
IF SaveHeight&=8 THEN←
CALL SetFont(WINDOW(8),topaz8&)←
GOTO Pill←
END IF←
IF SaveHeight&=9 THEN←
CALL SetFont(WINDOW(8),topaz9&)←
END IF←
Pill:←
LIBRARY CLOSE←
CLEAR ,25ØØØ←
END←
←
Menu2:←
ON MenuChoice GOTO Two1, Two2, Two3, Two4←
RETURN←
←
Two1:←
WINDOW 2,"Save Current Settings",(5Ø,9Ø)-(4ØØ,11Ø),
2Ø←
GOSUB Bold←
PRINT "Filename: ";←
GOSUB Roman←
LINE INPUT Filename$←
IF LEN(Filename$)=Ø THEN Wip←
Errorstatus=Ø←
ON ERROR GOTO TrapMe←
OPEN Filename$ FOR OUTPUT AS 2←
ON ERROR GOTO Ø←
IF Errorstatus=1 THEN Wip←
IF Phone$="" THEN Phone$="none"←
PRINT #2,Baud,Parity,Length,StBit,Duplex,Phone$←
FOR j=1 TO 1Ø←
PRINT #2,Mac$(j)←
NEXT←
Wip:←
```

```
CLOSE 2←
WINDOW CLOSE 2←
GOSUB Roman←
RETURN←
←
Two2:←
WINDOW 2,"Load New Settings",(50,90)-(400,120),20←
GOSUB Bold←
PRINT "Filename: ";←
GOSUB Roman←
LINE INPUT Filename$←
IF LEN(Filename$)=0 THEN Xip←
Errorstatus=0←
ON ERROR GOTO TrapMe←
OPEN Filename$ FOR INPUT AS 2←
ON ERROR GOTO 0←
IF Errorstatus=1 THEN Xip←
INPUT #2,Baud,Parity,Length,StBit,Duplex←
LINE INPUT #2,Phone$←
FOR j=1 TO 10←
LINE INPUT #2, Mac$(j)←
NEXT←
GOSUB OpenUp←
MENU 3, 1, 1, Baud$(Baud)←
MENU 3, 2, 1, Par$(Parity)←
MENU 3, 3, 1, Length$(Length)←
MENU 3, 4, 1, StBit$(StBit)←
MENU 3, 5, 1, Duplex$(Duplex)←
Xip:←
CLOSE 2←
WINDOW CLOSE 2←
GOSUB Bold←
PRINT Baud$(Baud)←
PRINT Par$(Parity)←
PRINT Length$(Length)←
PRINT StBit$(StBit)←
PRINT Duplex$(Duplex)←
PRINT "Phone # ";Phone$←
GOSUB Roman←
RETURN←
←
Two3:←
WINDOW 2,"Phone Number",(50,90)-(450,120),20←
GOSUB Bold←
PRINT "Current Number: ";Phone$←
PRINT "Press RETURN to exit, or enter new number"←
PRINT ": ";←
GOSUB Roman←
LINE INPUT Temp$←
IF LEN(Temp$)=0 THEN Zip←
```

```
Phone$=Temp$€
Zip:€
CLOSE 2€
WINDOW CLOSE 2€
GOSUB Roman€
RETURN€
€
Two4:€
Dial$="ATDT"+Phone$€
PRINT #1, Dial$€
IF Duplex THEN PRINT Dial$€
RETURN€
€
Menu3:€
ON MenuChoice GOTO Three1,Three2,Three3,Three4,Thre
e5€
RETURN€
€
Three1:€
Baud=1-Baud:MENU 3, 1, 1, Baud$(Baud)€
GOSUB OpenUp€
GOSUB Bold€
PRINT Baud$(Baud)€
GOSUB Roman€
RETURN€
€
Three2:€
Parity=Parity+1:IF Parity=3 THEN Parity=Ø€
MENU 3, 2, 1, Par$(Parity)€
GOSUB OpenUp€
GOSUB Bold€
PRINT Par$(Parity)€
GOSUB Roman€
RETURN€
€
Three3:€
Length=1-Length:MENU 3, 3, 1, Length$(Length)€
GOSUB OpenUp€
GOSUB Bold€
PRINT Length$(Length)€
GOSUB Roman€
RETURN      €
€
Three4:€
StBit=1-StBit€
MENU 3, 4, 1, StBit$(StBit)€
GOSUB OpenUp€
GOSUB Bold€
PRINT StBit$(StBit)€
GOSUB Roman€
RETURN€
```

```
←
Three5:←
Duplex=1-Duplex←
MENU 3, 5, 1, Duplex$(Duplex)←
GOSUB OpenUp←
GOSUB Bold←
PRINT Duplex$(Duplex)←
GOSUB Roman←
RETURN←
←
Menu4:←
ON MenuChoice GOTO Four1, Four2, Four3, Four4←
←
Four1:←
IF CaptureFlag=1 THEN ←
GOSUB Bold←
PRINT "Ending file capture."←
MENU 4,1,1,"Capture file"←
MENU 4,2,1←
CLOSE 2←
CaptureFlag=Ø←
GOSUB Roman←
RETURN←
END IF  ←
←
CaptureFlag=1←
WINDOW 2,"Capture file",(5Ø,9Ø)-(52Ø,12Ø),2Ø←
WIDTH(WINDOW(2)/8)←
GOSUB Bold←
PRINT "To end capture, choose same menu option."←
PRINT "Filename for your computer: ";←
GOSUB Roman←
LINE INPUT Filename$←
GOSUB Bold←
IF LEN(Filename$)=Ø THEN  CaptureFlag=Ø:GOTO YouQui
t←
ON ERROR GOTO TrapMe←
OPEN Filename$ FOR OUTPUT AS 2 LEN=1←
ON ERROR GOTO Ø←
IF Errorstatus=1 THEN←
Errorstatus=Ø←
GOTO CloseFour1←
END IF←
CloseFour1:←
MENU 4,1,1,"End capture"←
MENU 4,2,Ø←
WINDOW CLOSE 2←
GOSUB Bold←
PRINT "Beginning file capture."←
GOSUB Roman←
```

```
WIDTH(WINDOW(2)/8)←
RETURN←
←
YouQuit:←
WINDOW CLOSE 2←
GOSUB Bold←
PRINT "Cancelled at your request"←
GOSUB Roman←
RETURN←
←
Four2:←
CaptureFlag=0←
WINDOW 2,"Send file",(50,90)-(400,120),20←
WIDTH(WINDOW(2)/8)←
GOSUB Bold←
PRINT "Filename of Amiga file: ";←
GOSUB Roman←
LINE INPUT Filename$←
GOSUB Bold←
IF LEN(Filename$)=0 THEN YouQuit←
ON ERROR GOTO TrapMe←
OPEN Filename$ FOR INPUT AS 2←
ON ERROR GOTO 0←
IF Errorstatus=1 THEN←
Errorstatus=0←
WINDOW CLOSE 2←
GOSUB Roman←
WIDTH(WINDOW(2)/8)←
RETURN←
END IF←
MENU 4,0,0←
FileLength=LOF(2)←
BufferSize=FileLength*1.5←
IF BufferSize>32766 THEN BufferSize=32766←
PRINT "Sending ";Filename$;" : ";FileLength;"bytes.
....."←
Hunks=INT(FileLength/BufferSize)←
LeftOver=FileLength-Hunks*BufferSize←
j=1←
x$=""←
WHILE j<Hunks AND x$=""←
x$=INKEY$←
PRINT #1, INPUT$(BufferSize,2);←
j=j+1←
WEND←
IF x$="" THEN PRINT #1, INPUT$(LeftOver,2)←
PRINT:PRINT "Finished sending ";Filename$;"."←
CLOSE 2←
WINDOW CLOSE 2←
MENU 4,0,1←
GOSUB Roman←
```

```
WIDTH(WINDOW(2)/8)←
RETURN←
←
Menu5:←
IF MenuChoice=1 THEN Five1←
GOTO Five2←
←
Five1:←
WINDOW 2,"Edit macro",(50,90)-(400,120),20←
WIDTH(WINDOW(2)/8)←
GOSUB Bold←
PRINT "Macro to edit (1-10) : ";←
GOSUB Roman←
LINE INPUT Num$←
GOSUB Bold←
IF LEN(Num$)=0 OR VAL(Num$)<1 OR VAL(Num$)>10 THEN
YouQuit←
OverAgain:←
CLS←
GOSUB Bold←
PRINT "Macro ";Num$;":"←
PRINT Mac$(VAL(Num$));":";←
GOSUB Roman←
LINE INPUT Thing$←
IF LEN(Thing$)>30 THEN OverAgain←
Mac$(VAL(Num$))=Thing$←
WINDOW CLOSE 2←
GOSUB Bold←
PRINT Mac$(VAL(Num$))←
GOSUB Roman←
WIDTH(WINDOW(2)/8)←
RETURN←
←
Five2:←
PRINT #1,Mac$(MenuChoice-1)←
IF Duplex=1 THEN PRINT Mac$(MenuChoice-1)←
RETURN←
←
Bold:←
CALL SetSoftStyle& (WINDOW(8),2,255)←
RETURN←
←
Roman:←
CALL SetSoftStyle& (WINDOW(8),0,255)←
RETURN←
←
Setup:←
CLOSE←
DEFINT a-z←
DEFLNG B,FileLength←
```

```
PALETTE 0,1,1,1←
PALETTE 1,0,0,0←
PALETTE 3,0,.2,.8←
PALETTE 2,.9,.9,0←
←
ON ERROR GOTO LibTrap←
PRINT "Loading graphics library"←
LIBRARY "graphics.library"←
DECLARE FUNCTION OpenFont LIBRARY←
ON ERROR GOTO 0←
←
' Initialize menus←
Baud(0)=300←
Baud$(0)="Baud Rate"+STRING$(4,46)+"300"←
Baud(1)=1200←
Baud$(1)="Baud Rate"+STRING$(3,46)+"1200"←
←
Parity$(0)="n"←
Par$(0)="Parity"+STRING$(6,46)+"none"←
Parity$(1)="e"←
Par$(1)="Parity"+STRING$(6,46)+"even"←
Parity$(2)="o"←
Par$(2)="Parity"+STRING$(7,46)+"odd"←
←
Length(0)=8 ←
Length$(0)="Word Length"+STRING$(4,46)+"8"←
Length(1)=7←
Length$(1)="Word Length"+STRING$(4,46)+"7"←
←
StBit(0)=1←
StBit$(0)="Stop Bits"+STRING$(6,46)+"1"←
StBit(1)=2←
StBit$(1)="Stop Bits"+STRING$(6,46)+"2"←
←
Duplex$(0)="Duplex"+STRING$(6,46)+"full"←
Duplex$(1)="Duplex"+STRING$(6,46)+"half"←
←
Baud=1←
Parity=1←
Length=1←
StBit=0←
Duplex=0←
←
FOR j=1 TO 10←
Mac$(j)=CHR$(32)←
NEXT←
CLOSE←
ON ERROR GOTO FooledYou←
OPEN "setup" FOR INPUT AS 2←
←
```

```
FooledYou:←
IF ERR=Ø THEN FooledMe←
IF ERR=53 THEN RESUME ForReal←
ON ERROR GOTO Ø←
←
FooledMe:←
PRINT "Loading setup file"←
INPUT #2,Baud,Parity,Length,StBit,Duplex←
LINE INPUT #2,Phone$←
FOR j=1 TO 1Ø←
LINE INPUT #2,Mac$(j)←
NEXT←
←
ForReal:←
CLOSE←
ON ERROR GOTO TrapMe←
GOSUB OpenUp←
ON ERROR GOTO Ø←
←
Menus:←
MENU 1, Ø, 1, "Project"←
MENU 1, 1, 1, "Quit"+SPACE$(3)←
←
MENU 2, Ø, 1, "Special"+SPACE$(6)←
MENU 2, 1, 1, "Save Settings"←
MENU 2, 2, 1, "Load Settings"←
MENU 2, 3, 1, "Phone Number"+CHR$(32)←
MENU 2, 4, 1, "Dial"+SPACE$(9)←
←
MENU 3, Ø, 1, "Settings"+SPACE$(8)←
MENU 3, 1, 1, Baud$(Baud)←
MENU 3, 2, 1, Par$(Parity)←
MENU 3, 3, 1, Length$(Length)←
MENU 3, 4, 1, StBit$(StBit)←
MENU 3, 5, 1, Duplex$(Duplex)←
←
MENU 4, Ø, 1, "Transfer"+SPACE$(4)←
MENU 4, 1, 1, "Capture File"←
MENU 4, 2, 1, "Send File"←
←
DIM Macro$(11)←
MENU 5, Ø, 1, "Macros"+SPACE$(4)←
MENU 5, 1, 1, "Edit macro"←
FOR j=2 TO 11←
Macro$(j)="Macro"+CHR$(32)+STR$(j-1)←
MENU 5, j, 1, Macro$(j)←
NEXT←
←
' Sorry, AmigaBASIC won't permit←
' a borderless 8Ø-column window.←
WINDOW 1, "AmigaTerm 1.Ø", (Ø,Ø)-(631,186), 2Ø←
```

```
WIDTH 79←
GOSUB Bold←
CLS←
PRINT "Welcome to AmigaTerm 1.Ø  "←
PRINT "Press the right mouse button to view options
"←
GOSUB Roman←
RETURN←
←
TrapMe:←
GOSUB Bold←
Errorstatus=-1←
IF ERR=53 THEN PRINT "* File not found *":GOTO ErrO
ut←
IF ERR=61 THEN PRINT "* Disk is full *"  :GOTO ErrO
ut←
IF ERR=64 THEN PRINT "* Bad filename *"  :GOTO ErrO
ut←
IF ERR=7Ø THEN PRINT "Disk is write-protected *":GO
TO ErrOut←
PRINT "Error #" + STR$(ERR)←
ErrOut:←
GOSUB Roman←
RESUME NEXT←
←
LibTrap:←
IF ERR<>53 THEN CLOSE:END←
PRINT "Root directory of this disk does not contain
the file"←
PRINT "that I need to handle graphics. Exit BASIC a
nd copy"←
PRINT "GRAPHICS.BMAP from the BASICDEMOS directory
of your"←
PRINT "EXTRAS disk to this disk. Then rerun the pro
gram."←
CLOSE←
END←
←
```

Index

To order your copy of *COMPUTE!'s First Book of Amiga Disk*, call our toll-free US order line: 1-800-346-6767 (in NY 212-887-8525) or send your prepaid order to:

First Book of Amiga Disk
COMPUTE! Publications
P.O. Box 5038
F.D.R. Station
New York, NY 10150

All orders must be prepaid (check, charge, or money order). NC residents add 5% sales tax. NY residents add 8.25% sales tax.

Send _____ copies of *COMPUTE!'s First Book of Amiga Disk* at $15.95 per copy.

Subtotal $_____

Shipping and Handling: $2.00/disk $_____

Sales tax (if applicable) $_____

Total payment enclosed $_____

□ Payment enclosed
□ Charge □ Visa □ MasterCard □ American Express

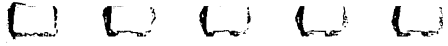
Acct. No. _____ Exp. Date _____
(Required)

Name _____

Address _____

City _____ State _____ Zip _____

Please allow 4-5 weeks for delivery.



COMPUTE! Books

Ask your retailer for these **COMPUTE! Books** or order directly from **COMPUTE!**.

Call toll free (in US) **1-800-346-6767** (in NY 212-887-8525) or write COMPUTE! Books, P.O. Box 5038, F.D.R. Station, New York, NY 10150.

Quantity	Title	Price*	Total
_____	COMPUTE!'s Beginner's Guide to the Amiga (025-4)	**$16.95**	_____
_____	COMPUTE!'s AmigaDOS Reference Guide (047-5)	**$16.95**	_____
_____	Elementary Amiga BASIC (041-6)	**$14.95**	_____
_____	COMPUTE!'s Amiga Programmer's Guide (028-9)	**$16.95**	_____
_____	COMPUTE!'s Kids and the Amiga (048-3)	**$14.95**	_____
_____	Inside Amiga Graphics (040-8)	**$17.95**	_____
_____	Advanced Amiga BASIC (045-9)	**$17.95**	_____
_____	COMPUTE!'s Amiga Applications (053-X)	**$16.95**	_____
_____	Learning C: Programming Graphics on the Amiga and Atari ST	**$18.95**	_____
_____	COMPUTE!'s First Book of Amiga (090-4)	**$16.95**	_____

*Add $2.00 per book for shipping and handling.
Outside US add $5.00 air mail or $2.00 surface mail.

NC residents add 5% sales tax _____
NY residents add 8.25% sales tax _____
Shipping & handling: $2.00/book _____
Total payment _____

All orders must be prepaid (check, charge, or money order).
All payments must be in US funds.
☐ Payment enclosed.
Charge ☐ Visa ☐ MasterCard ☐ American Express

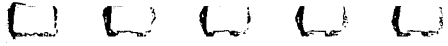
Acct. No._____ Exp. Date_____

Name_____

Address_____

City_____ State _____ Zip_____

*Allow 4–5 weeks for delivery.
Prices and availability subject to change.
Current catalog available upon request.



COMPUTE! Books

Ask your retailer for these **COMPUTE! Books** or order directly from **COMPUTE!**.
Call toll free (in US) **800-346-6767** (in NY 212-887-8525) or write COMPUTE! Books, P.O. Box 5038, F.D.R. Station, New York, NY 10150

Quantity	Title	Price*	Total
_____	Machine Language for Beginners (11-6)	**$16.95**	_____
_____	The Second Book of Machine Language (53-1)	**$16.95**	_____
_____	COMPUTE!'s Guide to Adventure Games (67-1)	**$14.95**	_____
_____	Computing Together: A Parents & Teachers Guide to Computing with Young Children (51-5)	**$12.95**	_____
_____	COMPUTE!'s Personal Telecomputing (47-7)	**$12.95**	_____
_____	BASIC Programs for Small Computers (38-8)	**$12.95**	_____
_____	Programmer's Reference Guide to the Color Computer (19-1)	**$12.95**	_____
_____	Home Energy Applications (10-8)	**$14.95**	_____
	The Home Computer Wars: An Insider's Account of Commodore and Jack Tramiel		
_____	Hardback (75-2)	**$16.95**	_____
_____	Paperback (78-7)	**$ 9.95**	_____
_____	The Book of BASIC (61-2)	**$12.95**	_____
_____	The Greatest Games: The 93 Best Computer Games of all Time (95-7)	**$ 9.95**	
_____	Investment Management with Your Personal Computer (005)	**$14.95**	_____
_____	40 Great Flight Simulator Adventures (022)	**$10.95**	_____
_____	40 More Great Flight Simulator Adventures (043-2)	**$12.95**	_____
_____	100 Programs for Business and Professional Use (017-3)	**$24.95**	_____
_____	From BASIC to C (026)	**$16.95**	_____
_____	The Turbo Pascal Handbook (037)	**$14.95**	_____
_____	Electronic Computer Projects (052-1)	**$10.95**	_____
	Flying on Instruments with Flight Simulator		
_____	perfect bound (091-2)	**$ 9.95**	_____
_____	wire bound (103-X)	**$12.95**	_____
	Jet Fighter School		
_____	perfect bound (092-0)	**$ 9.95**	_____
_____	wire bound (104-8)	**$12.95**	_____

* Add $2.00 per book for shipping and handling.
Outside US add $5.00 air mail or $2.00 surface mail.

NC residents add 5% sales tax. _____
NY residents add 8.25% sales tax _____
Shipping & handling: $2.00/book _____
Total payment _____

All orders must be prepaid (check, charge, or money order).
All payments must be in US funds.
☐ Payment enclosed.
Charge ☐ Visa ☐ MasterCard ☐ American Express

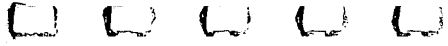
Acct. No. _____ Exp. Date _____ (Required)

Name _____

Address _____

City _____ State _____ Zip _____

*Allow 4–5 weeks for delivery.
Prices and availability subject to change.
Current catalog available upon request.



Amiga Power

COMPUTE!'s First Book of Amiga is a collection of some of the finest BASIC programs and tutorials written for the Commodore Amiga.

Continuing its support for this powerful computer, COMPUTE! Publications, the leader for home, educational, and recreational computing, presents the best programs and articles for the Amiga from *COMPUTE!* as well as some never-before-published programs. Here's a sample of what's inside:

- "AmigaTerm," a powerful terminal program that allows your Amiga, with a modem, to talk with other computers around the country and around the world.
- Strategy games such as "Beehive," "Hex Wars," "Chain Reaction," and more, that challenge your reasoning skills.
- Educational programs like "Pioneer" and "Hickory, Dickory, Dock" which entertain while they teach.
- "Amiga Math Graphics," which produces intricate and interesting displays on your monitor or TV.
- Tutorials that show you how to create batch files, use the built-in AmigaDOS editor, effectively program in BASIC, and more.

COMPUTE! Books is proud to add this first Amiga collection to its list of books on the Amiga. And since *COMPUTE!'s First Book of Amiga* is from COMPUTE! Publications, you can rely on high-quality programs that have been fully tested and are ready to run.

The programs in this book are available on a companion disk. See the coupon in the back for details.

ISBN 0-87455-090-4

COMPUTE!'s First Book of AMIGA

COMPUTE Books