COMPUTE!'s

2nd

# Book of Amiga

**More powerful games, tutorials, and much more for the Commodore Amiga personal computer.**

# COMPUTE!'s SECOND BOOK OF AMIGA

A **COMPUTE! Books** Publication

The following articles were originally published in *COMPUTE!* magazine, copyright 1987, COMPUTE! Publications, Inc.: "Jigsaw" (February); "Wari" (February); "Banner Printer" (March); "Euchre" (March); "3-D Surfaces" (March); "IFF Translator" (April); "Menu Planner" (April); "System Fonts" (April); "Rememory" (May); "Disk-Based Fonts" (June); "GraphiDemo" (June); "Weather Wizard" (July); "Zookeeper" (July); "Climber 5" (August); "Fractal Mountains" (August); "Monte Carlo" (September); "Ramdisker" (September); "SuperMenus" (September); "BASIC Button" (October); "Marbles" (October); "Word War" (October); "IFF to Icon Translator" (November); "Masked Input" (November); "Karma" (December).

The following articles were originally published in *COMPUTE!* magazine, copyright 1988, COMPUTE! Publications, Inc.: "Canfield" (January); "Tiles" (February); "Casino Blackjack" (March); "Math Draw" (March).

"Menu Driver" originally appeared in *COMPUTE!'s Amiga Applications,* by Brian Flynn, copyright 1987, COMPUTE! Publications, Inc.

*Laser Chess™*, a trademark of COMPUTE! Publications, Inc., originally appeared in *COMPUTE!* magazine (June), copyright 1987, COMPUTE! Publications, Inc.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-87455-122-6

COMPUTE! Books, Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is a Capital Cities/ABC, Inc. company and is not associated with any manufacturer of personal computers. Amiga is a trademark of Commodore-Amiga.

# Contents

# Foreword

From the moment the powerful Amiga appeared, COMPUTE! Books dedicated itself to supporting it. Following in the tradition that spawned our best-selling *First Book of Amiga, COMPUTE!'s Second Book of Amiga* offers a wide range of games, applications, tutorials, and programming tips.

For game players we've included the popular *Laser Chess*™, winner of COMPUTE!'s $10,000 programming contest. A game of strategy and skill, *Laser Chess* defies description. Also contained are "Karma," a futuristic battle for control of a society, and "Marbles," a wild, fast-paced arcade game of excitement. And for card sharks, we've included the best of computer solitaire, "Euchre," "Monte Carlo," "Canfield," and "Casino Blackjack."

Two invaluable applications, "Banner Printer" and "Menu Planner" help at home and at the office. What's the weather going to be like in your neck of the woods tomorrow? Keep ahead of the storm with "Weather Wizard."

Would you like to know about the Amiga's amazing graphics? *COMPUTE!'s Second Book of Amiga* is loaded with graphics utilities and demos. Use the "IFF Translator" to convert your IFF files for use with Amiga Basic. Enjoy fractal landscapes with "Fractal Mountains," and explore the third dimension with "3-D Surfaces."

If programming is your niche, you'll find "Ramdisker" and "Zookeeper" indispensable. And you can learn to enhance BASIC with "BASIC Button."

As with all COMPUTE! Books, *COMPUTE!'s Second Book of Amiga* contains only programs that have been rigorously and fully tested and that are ready to type in and use. If you prefer not to type them in, a companion disk which includes all the programs from the book—ready to load and run—is available. To order the disk, use the coupon in the back of this book.

# CHAPTER ONE

# Strategy and Thinking Games

# *Laser Chess*™

Mike Duppong
*Translation by Tim Midkiff*

Laser Chess™ *won First Prize in our $10,000 programming contest for* COMPUTE!'s Atari ST Disk & Magazine. *Awarded $5,000 for its originality and skillful* programming, Laser Chess *is a two-player strategy game patterned after traditional chess—with some fascinating new twists. The original version was written in Modula-2 for the Atari ST. Here we have provided a BASIC translation for the Amiga. The Amiga version requires at least 512K of memory.*

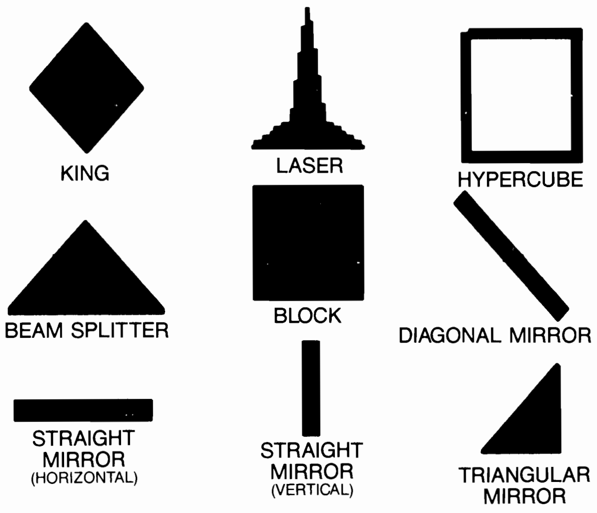
*Laser Chess*™, as the name implies, is a chesslike strategy game for two players. The goal is to manipulate a laser-firing piece and various reflective objects to eliminate your opponent's king. As in traditional chess, there are many ways to accomplish this. Type in and save a copy of the game before you play.

There are eight basic types of pieces in *Laser Chess,* and each has unique capabilities. Over time, you'll learn each piece's advantages and limitations. Obviously, the more you play *Laser Chess,* the more you'll understand the pieces in your arsenal, which in turn will make you a better player. So let's start with a description of the pieces.

## A Geometric Army

Figure 1-1 shows each piece and its name. Some sides of certain pieces are highlighted. This indicates a reflective surface. When a laser beam strikes a reflective surface, it bounces off without harming the piece. But if a piece is hit by a laser on a nonreflective surface, it is destroyed.

**Figure 1-1. The Game Pieces**



*These are the basic pieces in* Laser Chess.

A piece can also be removed from the board if it is captured by an opposing piece. This is similar to traditional chess; to capture a piece, you simply move one of your own pieces onto its square.

In addition to their ability to move from square to square, pieces with reflective surfaces can also be rotated in place in 90-degree increments. This lets you orient the piece to protect it against opposing laser shots or to set up bounce shots with your own laser.

The *king* is the most important piece in *Laser Chess.* When the king is eliminated, the other player wins the game. Since it has no reflective surfaces, it can be destroyed by a laser from any angle. It can also be captured by an opposing piece. The king is not totally defenseless, however. It can capture any opposing piece by moving onto its square. But you can use it for a capture only once per turn.

The second most important piece is the *laser*. This piece is your primary offensive weapon; it's the only piece which can fire a laser shot. To take aim, you can rotate it in place at 90-degree angles. Like the king, the laser is completely vulnerable to enemy laser strikes, because it has no reflective surfaces. If you lose your laser, the game is not over, but only the most skillful (or incredibly lucky) player can overcome its loss.

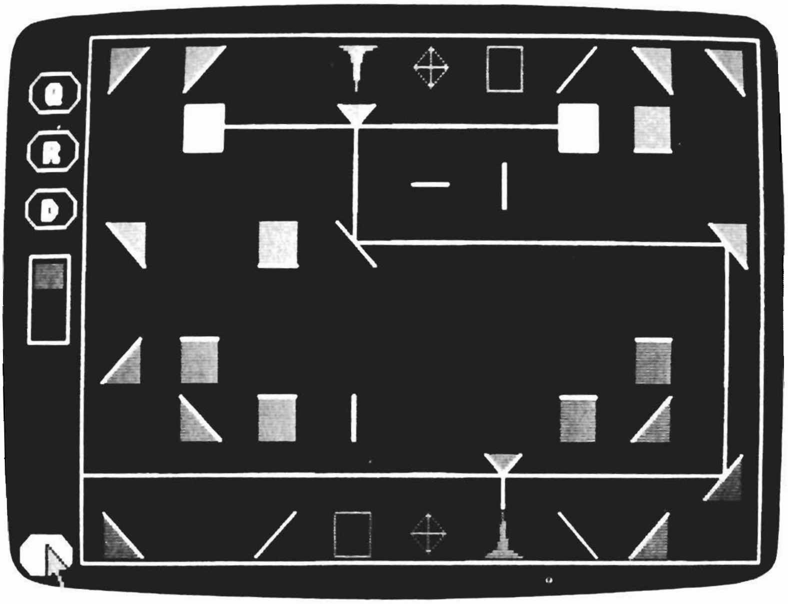
## Tricky Pieces

The *hypercube* is an interesting piece. It can't harm an opposing piece directly, but may very well do so indirectly. When the hypercube is moved onto another piece (even your own), that piece disappears from its original position and reappears on a randomly selected empty square. This can happen only once per turn. The hypercube can be a two-edged sword; it may relocate a piece to a vulnerable position, or it may make it possible for the piece to capture an important opposing piece on the next move. The hypercube has no reflective surfaces and cannot be rotated. It is invulnerable to laser shots, however, because it's made of transparent material—a laser beam passes right through it. Remember that.

The *beam splitter* is another tricky piece. When a laser beam strikes a splitter's vertex (the point opposite its base), the beam splits in two. The two new beams travel in opposite directions, perpendicular to the original beam's path (see Figure 1-2). When a laser shot hits one of the beam splitter's reflective surfaces, it bounces off at a 90-degree angle *without* splitting. If the beam splitter's base is hit by a laser shot, it is destroyed. The beam splitter can be rotated.

The *blocks* are fairly simple pieces. However, they may impose some complex situations. A block can capture any opposing piece by moving onto that piece's square, much like a king. But unlike a king, a block has one reflective side and can be rotated as the situation demands. Therefore, blocks can be used either offensively or defensively. A laser beam that hits the reflective surface of a block is deflected 180 degrees—bouncing the beam back where it came from.

**Figure 1-2. A Beam Splitter in Action**



*As seen in this magnified view, a beam splitter's vertex reflects a laser shot in two perpendicular directions.*

A *diagonal mirror* cannot be destroyed by a laser, because both of its surfaces are reflective. Diagonal mirrors can be removed from the board only when captured by a block or a king. When a laser beam strikes a diagonal mirror, the beam is deflected 90 degrees. Diagonal mirrors can be flipped to their opposite diagonal, but cannot be rotated to face horizontally or vertically.

The *horizontal mirrors* and *vertical mirrors* (known collectively as *straight mirrors*) are also invulnerable to lasers due to their reflective surfaces. When a laser hits a straight mirror on its flat surface, the beam is deflected 180 degrees. If the laser hits a straight mirror edgewise, the beam passes straight through it. (Look closely at Figure 1-2. A laser beam is passing through a horizontal mirror just to the left of the beam splitter.) Straight mirrors can be rotated to become either horizontal or vertical mirrors, but not diagonal mirrors.

The *triangular mirrors* deflect laser beams just as diagonal mirrors do, but they are vulnerable to hits on their two nonreflective sides. A triangular mirror can be rotated in 90-degree increments.

## Making Moves

*Laser Chess* requires 512K of memory and Microsoft Amiga Basic. At the beginning of the game, you can choose between filled and unfilled playing pieces by pressing F and U, respectively. This option affects only the appearance of the pieces.

As in the conventional game of chess, a move in *Laser Chess* consists of moving or otherwise manipulating a game piece. The same player always moves first in *Laser Chess.* There's no particular advantage or disadvantage to moving first.

*Laser Chess* is played with the mouse. To move a piece, position the mouse pointer over the desired piece and hold down the left mouse button. When the ghosted image of that piece appears, you can either drag the piece to a new location or rotate it by pressing a key. Release the mouse button to drop the piece in its new location.

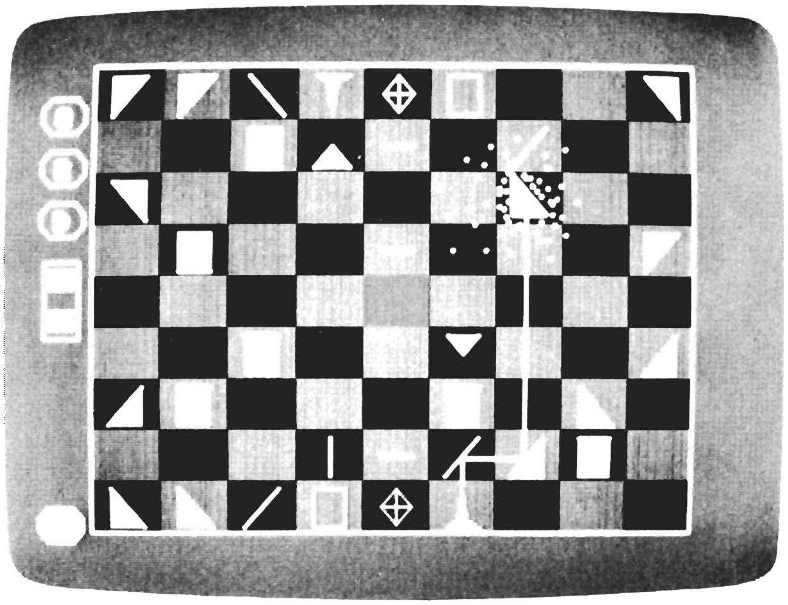
A turn consists of two moves. The number of moves remaining in a turn is indicated visually on the screen. The color of the playfield border indicates the number of turns remaining and whose turn it is. When the border around the sides of the board are doubled, the player of the border color has two moves left in the turn. If the border is single, the player has one move left in the turn.

Before you move or rotate a piece, you must select it. When a piece is selected, its appearance changes to a ghosted image.

If you accidentally select the wrong piece, you can deselect it by placing it back where you got it from and releasing it. Deselecting is usually done after rotating a piece—more on this later.

After you've selected a piece, your next decision is whether to move or rotate it. Moving a distance of one square takes one move; moving two squares takes two moves (although you can move a piece two squares in one step). Since

**Figure 1-3. Game Board and Controls**



*This full-screen view of* Laser Chess *shows its 9 × 9 board grid and game controls.*

you have only two moves per turn, the maximum distance a piece can be moved in one turn is two squares. The computer does not allow illegal moves.

Pieces can be moved forward, backward, left, or right, but not diagonally. You can effectively move a piece diagonally by using two moves—forward and right, for instance.

You cannot move a piece onto a square occupied by another piece. The only exceptions are captures with blocks and kings, and moves of the hypercube as described above.

## Rotating a Piece

The computer does not allow you to rotate a piece that's incapable of rotation. Otherwise, the piece rotates 90 degrees (one-quarter turn) clockwise. You may continue rotating the piece to any desired position before deselecting it. Rotating a piece to face any direction takes only one move, and the move is subtracted after the piece is deselected. If you deselect the piece in its original position, no move is subtracted.

You can combine a rotation and a move in a single action. First, select the piece. Then rotate it to the direction you wish it to face. Finally, move to any adjacent square (except a diagonal) as you would normally do. The piece moves to that square and faces in the direction you've chosen. Since rotating a piece and moving a piece each take one move, this uses up your turn.

## Special Features

At the center of the 9 × 9 board is a special square called a *hypersquare*. It absorbs laser beams and acts like a stationary hypercube. That is, if you try to move a piece onto it, the piece disappears from its original position and reappears on a randomly selected empty square. This can happen only once per turn, however.

There are three buttons to the left of the board. To select a button, move the mouse pointer over the button and press the left mouse button. The button labeled Q allows you to quit playing at any time. When selected, this option requires that you confirm your decision.

The restart button (R) lets you start a new game without finishing the current game. (For instance, a player may be so hopelessly behind that he or she wants to resign.) Again, the program asks that you confirm this choice.

## Firing the Laser

The last button is the laser trigger. When it's your turn, you can click this button to fire your laser. Firing your laser takes only one move, but it can be done only once per turn. Therefore, you may want to use your first move in a turn to aim the laser, rotate a reflecting piece to set up a bounce shot, or move another piece into position.

Of course, you won't necessarily be firing the laser on every turn. Much of the strategy in *Laser Chess* involves moving and rotating your pieces to set up complex shots. It's important to realize that *any* laser hit on a piece's nonreflective or nontransparent surface will destroy that piece. You can destroy your own pieces just as easily as you can destroy your opponent's. You can even zap your own laser, particularly if you fire directly into the 180-degree reflective surface of a straight mirror or block, or if you fail to anticipate the effects of a beam splitter. Be forewarned.

## *Laser Chess* Strategy

As in the conventional game of chess, much of the strategy in *Laser Chess* revolves around thoughtful placement of your pieces. However, the character of the game differs from that of chess in many ways. The laser, for example, can strike at long distances and in more than one direction at once. And the hypercube adds an extra element of uncertainty. The best strategy for any particular game depends to a great extent on the skill and personality of your opponent. However, here are some general tips you may find helpful.

Get your mirrors out early. Use them to gain the fullest potential of your laser. Try to position mirror networks on both sides of the beam splitter so you can inflict as much damage as possible.

Take advantage of the blocks. Since they "control" an area around them with their threat of capture, no other pieces can safely move within their range. Make your opponent work to displace them. Remember to rotate the reflective side of a block to the most probable direction of laser fire. If you can prevent a laser from destroying the block, your opponent will most likely have to gang up on it with two or more of his or her own blocks.

Use mirrors to protect your king. If you surround your king with straight and diagonal mirrors, there is no way it can be hit by a laser. Therefore, your opponent will have to break through your defense with blocks. (This is a pretty dirty trick when all of your opponent's blocks have been destroyed, since your king is almost invulnerable.) Defending your king with blocks is also a good strategy.

The hypercube should be used sparingly, since you have no idea where a relocated piece will reappear. Most players use the hypercube as a last resort: If a piece is going to be destroyed anyway, it doesn't hurt to take a chance and relocate it with the hypercube. Also, if your opponent's king is encircled with mirrors, you can march right in with your hypercube, followed by a block. This tactic may displace your opponent's defense, forcing your opponent to evacuate the king from its mirrored fortress. Escorting the hypercube with an adjacent block prevents the opponent from attacking the hypercube with his or her king. Your opponent's only options will be to flee or be displaced.

## *Laser Chess™*

Filename: LASER CHESS

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'Laser Chess←
'Copyright 1987 Compute! Publications, Inc.←
'All Rights Reserved←
CLEAR,25000:CLEAR ,50000&←
DEFINT a-z:DEFSNG r,g,b,mx:RANDOMIZE TIMER:SCREEN 1,
320,200,4,1←
WINDOW 3,,(0,0)-(311,186),16,1:WINDOW OUTPUT 3:COLOR
 ,0←
DIM sn(8,3,1,1),es(155,1),shape(155,87),piece(9,9),o
rient(9,9),cLr(9,9)←
DIM os(155),beamck(3,9,9),dirck(8,3,3),bmd0(158),bmd
1(22),shpt(8)←
DIM ddrcx(1,20),ddrcy(1,20),pt(14)←
DIM s(255),n(255),sq(255),freq(20,4),shptx(8,19),shp
ty(8,19)←
LOCATE 1,4:PRINT CHR$(169)"1987 Compute! Publication
s, Inc."←
LOCATE 3,11:PRINT"All Rights Reserved"←
LOCATE 12,9:PRINT"(F)illed or (U)nfilled?"←
WHILE NOT(k$="F" OR k$="U"):k$=UCASE$(INKEY$):WEND:f
L=k$="F"←
PALETTE 0,.15,.05,.5:PALETTE 1,.15,.25,.95←
FOR i=2 TO 14:PALETTE i,.15,.05,.5:NEXT:PALETTE 15,.
15,.25,.95←
ON TIMER(1) GOSUB CLock:ti=36-fL*10←
COLOR 1,0:CLS:LOCATE 10,14:PRINT "PLEASE  WAIT"←
LOCATE 12,18:PRINT "seconds":TIMER ON←
FOR i=0 TO 255:s(i)=127-i:NEXT:FOR i=0 TO 255:n(i)=1
27-RND*255:NEXT:WAVE 0,s←
FOR i=0 TO 127:sq(i)=127-RND*50:NEXT:FOR i=128 TO 25
5:sq(i)=-128+RND*50:NEXT←
cop(1)=4:cop(2)=6:GOSUB InitShapes:GOSUB InitObjects
```

```
:TIMER OFF:CLS←
RESTORE PaLetteData:FOR i=2 TO 14:READ r,g,b:PALETTE
 i,r,g,b:NEXT←
PaLetteData: ←
DATA Ø,Ø,Ø,.3,.3,.3←
DATA .6,Ø,Ø,1,Ø,Ø,Ø,.55,Ø,Ø,.9,Ø←
DATA 1,1,Ø,1,1,Ø,.6,.6,.6,1,1,1←
DATA 1,1,Ø,1,1,Ø,1,1,Ø←
Start:←
L(1)=1:L(2)=1:Lpx(1)=4:Lpy(1)=1:Lpx(2)=6:Lpy(2)=9←
COLOR ,Ø:GOSUB DrawBoard:k=Ø:pL=1←
Main:←
pL=pL XOR 3:px=5:py=5:move=2:hycube=Ø:hysq=Ø:taken=Ø
:fired=1←
LINE(4Ø,1Ø)-(288,186),cop(pL),b:LINE(42,12)-(286,184
),cop(pL),b←
MovePiece:←
WHILE MOUSE(Ø)>-1:WEND:x=MOUSE(3):y=MOUSE(4)←
px=INT((x-17)/27):py=INT((y+6)/19):moves=Ø←
IF NOT((px>Ø AND px<1Ø) AND (py>Ø AND py<1Ø)) THEN O
ptions←
IF cLr(px,py)<>pL THEN MovePiece←
piece=piece(px,py):rot=orient(px,py)←
obindex=oi(piece,rot):spx=px:spy=py←
IF NOT(obindex>Ø) THEN MovePiece←
OBJECT.X obindex,x-14:OBJECT.Y obindex,y-1Ø:OBJECT.O
N obindex


                                                    ←
WHILE MOUSE(Ø)<Ø←
OBJECT.X obindex,MOUSE(1)-14:OBJECT.Y obindex,MOUSE(
2)-1Ø←
IF INKEY$<>"" THEN←
rot=(rot+1) AND turns(piece):j=obindex:obindex=oi(pi
ece,rot)←
OBJECT.X obindex,MOUSE(1)-14:OBJECT.Y obindex,MOUSE(
2)-1Ø←
OBJECT.OFF j:WAVE Ø,s:SOUND 4ØØØ,.1,255,Ø:OBJECT.ON
obindex←
END IF ←
WEND←
OBJECT.OFF obindex←
GOSUB EraseSquare←
px=INT((MOUSE(5)-17)/27):py=INT((MOUSE(6)+6)/19)←
GOSUB CheckMove←
GOSUB PutShape←
IF piece(px,py)=2 THEN Lpx(pL)=px:Lpy(pL)=py←
EndMove:IF k THEN EndGame←
move=move-moves:IF move=1 THEN LINE(4Ø,1Ø)-(288,186)
,Ø,b←
IF move>Ø THEN MovePiece←
```

```
GOTO Main←
←
CLock:ti=ti-1:LOCATE 12,14:PRINT STR$(ti)" ":RETURN←
←
InitShapes:←
LINE(Ø,Ø)-(Ø,Ø),1Ø,bf:GET(Ø,Ø)-(Ø,Ø),pt:PUT(Ø,Ø),pt←
LINE(Ø,Ø)-(Ø,18),1Ø:GET(Ø,Ø)-(Ø,18),bmdØ:PUT(Ø,Ø),bm
dØ←
LINE(Ø,Ø)-(26,Ø),1Ø:GET(Ø,Ø)-(26,Ø),bmd1:PUT(Ø,Ø),bm
d1←
LINE(Ø,Ø)-(26,18),2,bf:GET(Ø,Ø)-(26,18),es(Ø,Ø)←
LINE(Ø,Ø)-(26,18),3,bf:GET(Ø,Ø)-(26,18),es(Ø,1)←
RESTORE LaserDir:FOR i=Ø TO 3:READ dirx(i),diry(i):N
EXT←
LaserDir:DATA Ø,-1,1,Ø,Ø,1,-1,Ø←
RESTORE ShapePts←
k=Ø:x=Ø:y=Ø:FOR i=1 TO 8:READ turns(i),shpt(i)←
FOR j=Ø TO shpt(i)+1:READ shptx(i,j),shpty(i,j):NEXT
←
GOSUB GetShapes:NEXT←
RESTORE ShapeRefLect←
FOR i=1 TO 8:FOR j=Ø TO turns(i):FOR k=Ø TO 3:READ d
irck(i,j,k):NEXT k,j,i←
RETURN←
ShapePts:←
DATA 1,1,-1,17,17,1,Ø,Ø←
DATA 3,6,7,17,9,1,11,17,7,17,1,15,17,15,11,17,9,9←
DATA 1,1,-1,9,17,9,Ø,Ø←
DATA Ø,7,5,9,9,5,13,9,9,13,5,9,13,9,9,13,9,5,Ø,Ø←
DATA 3,6,1,2,17,2,17,17,1,17,1,2,-1,1,17,1,9,9←
DATA Ø,4,1,1,17,1,17,17,1,17,1,1,Ø,Ø←
DATA 3,6,2,1,16,1,9,8,2,1,-1,1,-9,9,17,1,9,4←
DATA 3,5,2,17,17,17,17,2,2,17,-1,17,17,1,13,13←
ShapeRefLect:←
DATA 1,Ø,3,2,3,2,1,Ø,-1,-1,-1,-1,-1,-1,-1,-1←
DATA -1,-1,-1,-1,-1,-1,-1,-1,2,1,Ø,3,Ø,3,2,1←
DATA -1,-1,-1,-1,-1,-1,Ø,-1,-1,-1,-1,1,2,-1,-1,-1←
DATA -1,3,-1,-1,Ø,1,2,3,-2,2,-1,2,3,-2,3,-1←
DATA -1,Ø,-2,Ø,1,-1,1,-2,-1,Ø,3,-1,-1,-1,1,Ø←
DATA 1,-1,-1,2,3,2,-1,-1←
←
GetShapes:←
FOR angLe=Ø TO turns(i):FOR bkgd=Ø TO 1:FOR pL=1 TO
2←
co=cop(pL):PUT(Ø,Ø),es(Ø,bkgd),PSET←
ON angLe+1 GOSUB rotateØ,rotate9Ø,rotate18Ø,rotate27
Ø←
sn(i,angLe,pL-1,bkgd)=k:GET(Ø,Ø)-(26,18),shape(Ø,k):
k=k+1←
NEXT pL,bkgd,angLe:RETURN←
←
```

```
rotate0:←
FOR j=1 TO shpt(i):IF shptx(i,j-1)<0 THEN hue=co+1 E
LSE hue=co←
LINE(ABS(shptx(i,j-1))+4+x,shpty(i,j-1)+y)-(ABS(shpt
x(i,j))+4+x,shpty(i,j)+y),hue:NEXT←
IF shptx(i,shpt(i)+1)>0 AND fL THEN PAINT(shptx(i,sh
pt(i)+1)+4+x,shpty(i,shpt(i)+1)+y),co,co←
RETURN←
rotate90:←
FOR j=1 TO shpt(i):IF shptx(i,j-1)<0 THEN hue=co+1 E
LSE hue=co←
LINE(18-shpty(i,j-1)+4+x,ABS(shptx(i,j-1))+y)-(18-sh
pty(i,j)+4+x,ABS(shptx(i,j))+y),hue:NEXT←
IF shptx(i,shpt(i)+1)>0 AND fL THEN PAINT(18-shpty(i
,shpt(i)+1)+4+x,shptx(i,shpt(i)+1)+y),co,co←
RETURN←
rotate180:←
FOR j=1 TO shpt(i):IF shptx(i,j-1)<0 THEN hue=co+1 E
LSE hue=co←
LINE(18-ABS(shptx(i,j-1))+4+x,18-shpty(i,j-1)+y)-(18
-ABS(shptx(i,j))+4+x,18-shpty(i,j)+y),hue:NEXT←
IF shptx(i,shpt(i)+1)>0 AND fL THEN PAINT(18-shptx(i
,shpt(i)+1)+4+x,18-shpty(i,shpt(i)+1)+y),co,co←
RETURN←
rotate270:←
FOR j=1 TO shpt(i):IF shptx(i,j-1)<0 THEN hue=co+1 E
LSE hue=co←
LINE(shpty(i,j-1)+4+x,18-ABS(shptx(i,j-1))+y)-(shpty
(i,j)+4+x,18-ABS(shptx(i,j))+y),hue:NEXT←
IF shptx(i,shpt(i)+1)>0 AND fL THEN PAINT(shpty(i,sh
pt(i)+1)+4+x,18-shptx(i,shpt(i)+1)+y),co,co←
RETURN←
←
InitObjects:←
k=1:si$=STRING$(26,0):POKE SADD(si$)+11,4:POKE SADD(
si$)+15,27←
POKE SADD(si$)+19,19:POKE SADD(si$)+21,24:POKE SADD(
si$)+23,15←
FOR piece=1 TO 8:FOR angLe=0 TO turns(piece):seLect=
sn(piece,angLe,0,0)←
PUT(0,0),es(0,0),PSET:PUT(0,0),shape(0,seLect)←
oi(piece,angLe)=k:GET(0,0)-(26,18),os←
sd$="":FOR i=3 TO 154:sd$=sd$+MKI$(os(i)):NEXT←
OBJECT.SHAPE k,si$+sd$:OBJECT.PLANES k,3,8←
k=k+1:NEXT angLe,piece:RETURN←
←
DrawBoard:←
COLOR 3,2:LINE(11,54)-STEP(16,11),,b:PAINT(12,55),2,
3:LOCATE 8,3:PRINT"Q"←
LINE(11,94)-STEP(16,10),,b:PAINT(12,95),2,3:LOCATE 1
3,3:PRINT"R"←
```

```
LINE(11,134)-STEP(16,10),,b:PAINT(12,135),2,3:LOCATE
 18,3:PRINT"L" <
FOR py=1 TO 9:FOR px=1 TO 9:GOSUB EraseSquare:NEXT p
x,py<
LINE(151,89)-(177,107),0,bf<
RESTORE ShapePos:FOR py=1 TO 2:FOR px=1 TO 9:cLr(px,
py)=1:cLr(px,py+7)=2<
READ piece(px,py),orient(px,py),orient(10-px,10-py)<
piece(10-px,10-py)=piece(px,py):NEXT px,py<
FOR px=1 TO 9:FOR py=1 TO 9:IF piece(px,py)>0 THEN G
OSUB PutShape<
NEXT py,px<
ShapePos:<
DATA 8,2,0,8,2,0,1,1,1,2,2,0,4,0,0,6,0,0<
DATA 1,0,0,8,3,1,8,3,1,8,3,1,5,2,0,5,2,0<
DATA 7,2,0,3,0,0,3,1,1,5,2,0,5,2,0,8,2,0<
RETURN<
<
PutShape:<
x=px*27+16:y=py*19-6:bkgd=(px+py+1) AND 1<
PUT(x,y),shape(0,sn(piece(px,py),orient(px,py),cLr(p
x,py)-1,bkgd)),PSET<
RETURN<
<
EraseSquare:<
x=px*27+16:y=py*19-6:bkgd=(px+py+1) AND 1:PUT(x,y),e
s(0,bkgd),PSET<
RETURN<
<
Fire:<
px=Lpx(pL):py=Lpy(pL):Lx(1)=px:Ly(1)=py:dir(1)=orien
t(px,py)<
FOR i=1 TO 3:aLive(i)=0:term(i)=0:NEXT:aLive(1)=1<
WHILE (aLive(1)=1) OR (aLive(2)=1) OR (aLive(3)=1)<
FOR i=1 TO 3:IF aLive(i)<1 THEN AdvBeam<
nLx(i)=Lx(i)+dirx(dir(i)):nLy(i)=Ly(i)+diry(dir(i))<
IF beamck(dir(i),Lx(i),Ly(i))=1 THEN EndBeam<
beamck(dir(i),Lx(i),Ly(i))=1:GOTO DrawBeam<
Hit:term(i)=1:drk(i)=tdir:IF d THEN EndBeam<
tx=px:ty=py:px=Lx(i):py=Ly(i):IF piece(px,py)=4 THEN
 k=k+cLr(px,py)<
IF piece(px,py)=2 THEN L(cLr(px,py))=0<
x=px*27+16:y=py*19-6<
m=piece(px,py):shpt(0)=shpt(m):FOR j=0 TO shpt(0)+1:
shptx(0,j)=shptx(m,j)<
shpty(0,j)=shpty(m,j)::NEXT:t=i:i=0:co=8<
ON orient(px,py)+1 GOSUB rotate0,rotate90,rotate180,
rotate270<
i=t:px=tx:py=ty<
EndBeam:aLive(i)=-1<
AdvBeam:NEXT:WEND<
```

```
RETURN<
<
DrawBeam:<
x=Lx(i)*27+29:y=Ly(i)*19+3<
ON dir(i) GOTO BRt,BDn,BLt<
BUp:PUT(x,y-19),bmdØ:GOTO CkBeam<
BRt:PUT(x,y+1),bmd1:GOTO CkBeam<
BDn:PUT(x+1,y),bmdØ:GOTO CkBeam<
BLt:PUT(x-27,y),bmd1:GOTO CkBeam<
<
CkBeam:<
IF (nLx(i)>9) OR (nLy(i)>9) OR (nLx(i)<1) OR (nLy(i)
<1) THEN EndBeam<
IF nLx(i)=5 AND nLy(i)=5 THEN EndBeam<
Lx(i)=nLx(i):Ly(i)=nLy(i):IF piece(nLx(i),nLy(i))=Ø
THEN AdvBeam<
tdir=dir(i):dir(i)=dirck(piece(Lx(i),Ly(i)),orient(L
x(i),Ly(i)),dir(i))<
IF dir(i)=-1 THEN Hit<
IF dir(i)>-2 THEN AdvBeam<
IF aLive(2)=Ø THEN j=2 ELSE j=3<
aLive(j)=1:Lx(j)=Lx(i):Ly(j)=Ly(i)<
dir(i)=tdir+1 AND 3:dir(j)=tdir-1 AND 3<
GOTO AdvBeam<
<
Laser:<
k=Ø:d=Ø:GOSUB Fire<
FOR i=Ø TO 3:FOR x=1 TO 9:FOR y=1 TO 9:beamck(i,x,y)
=Ø:NEXT y,x,i<
FOR i=1 TO 3<
IF term(i)=1 THEN<
IF piece(Lx(i),Ly(i))>Ø THEN<
tx=px:ty=py:px=Lx(i):py=Ly(i):GOSUB ExpLode:px=tx:py
=ty<
END IF<
END IF<
NEXT<
TIMER OFF:d=1:GOSUB Fire<
FOR i=1 TO 3<
IF term(i)=1 THEN<
IF piece(Lx(i),Ly(i))>Ø THEN<
tx=px:ty=py:px=Lx(i):py=Ly(i):piece(px,py)=Ø:cLr(px,
py)=Ø<
GOSUB EraseSquare:px=tx:py=ty<
END IF<
END IF<
NEXT<
FOR i=Ø TO 3:FOR x=1 TO 9:FOR y=1 TO 9:beamck(i,x,y)
=Ø:NEXT y,x,i<
RETURN<
<
ExpLode:<
```

```
FOR j=Ø TO 4:vol(4-j)=(j+1)*4Ø:NEXT:ch=Ø←
FOR j=Ø TO 2Ø:t=9ØØ-INT(RND*8)*1ØØ:FOR m=Ø TO 4:freq
(j,m)=t:NEXT m,j←
Lv=12Ø:cx=px*27+29:cy=py*19+3:WAVE Ø,n:WAVE 1,n←
IF dirx(drk(i))=Ø THEN←
FOR j=Ø TO 2Ø:ddrcy(Ø,j)=INT(RND*1Ø)*diry(drk(i))+cy
←
ddrcx(Ø,j)=cx+1Ø-INT(RND*2Ø)←
ddrcy(1,j)=INT(RND*2Ø)*diry(drk(i))+cy←
ddrcx(1,j)=cx+2Ø-INT(RND*4Ø):NEXT←
ELSE←
FOR j=Ø TO 2Ø:ddrcx(Ø,j)=INT(RND*1Ø)*dirx(drk(i))+cx
←
ddrcy(Ø,j)=cy+1Ø-INT(RND*2Ø)←
ddrcx(1,j)=INT(RND*2Ø)*dirx(drk(i))+cx←
ddrcy(1,j)=cy+2Ø-INT(RND*4Ø):NEXT←
END IF←
GOSUB EraseSquare←
FOR j=Ø TO 2Ø:PUT(ddrcx(Ø,j),ddrcy(Ø,j)),pt:IF (j AN
D 4)=4 THEN GOSUB ExpSnd←
NEXT←
FOR j=Ø TO 2Ø:PUT(ddrcx(Ø,j),ddrcy(Ø,j)),pt:PUT(ddrc
x(1,j),ddrcy(1,j)),pt←
IF (j AND 4)=4 THEN GOSUB ExpSnd←
NEXT:FOR j=Ø TO 2Ø:PUT(ddrcx(1,j),ddrcy(1,j)),pt:NEX
T←
RETURN←
←
ExpSnd:←
ch=1-ch:FOR m=Ø TO 4:SOUND freq(j,m),.Ø5,vol(m),ch:N
EXT:RETURN←
←
←
CheckMove:←
dx=ABS(px-spx):dy=ABS(py-spy)←
moves=dx+dy+ABS(rot<>orient(spx,spy))←
IF dx=Ø AND dy=Ø THEN VaLidMove←
IF NOT(px>Ø AND px<1Ø AND py>Ø AND py<1Ø) THEN InVaL
idMove←
IF moves>move THEN InVaLidMove←
IF moves=2 THEN←
midx=(px+spx)/2:midy=(py+spy)/2←
IF midx=5 AND midy=5 THEN InVaLidMove←
IF dx=2 THEN IF piece(midx,py)<>Ø THEN InVaLidMove←
IF dy=2 THEN IF piece(px,midy)<>Ø THEN InVaLidMove←
IF dx=1 AND dy=1 THEN←
IF ((piece(px,spy)<>Ø) OR (px=5 AND spy=5)) AND ((pi
ece(spx,py)<>Ø) OR (spx=5 AND py=5)) THEN InVaLidMov
e←
END IF←
END IF  ←
```

```
IF piece(px,py)<>Ø THEN←
IF piece=4 OR piece=5 THEN←
IF taken THEN InVaLidMove←
IF piece(px,py)=4 THEN k=cLr(px,py)←
IF piece(px,py)=2 THEN L(cLr(px,py))=Ø←
WAVE Ø,n:WAVE 1,n←
FOR i=255 TO 1Ø STEP -2Ø:SOUND 4ØØ,.1,i,Ø:SOUND 4ØØ,
.1,i,1:NEXT←
taken=1:GOTO VaLidMove←
ELSEIF piece=6 THEN←
IF hycube THEN InVaLidMove←
hycube=1:GOTO HyperCube←
ELSE←
GOTO InVaLidMove←
END IF←
END IF←
IF NOT(px=5 AND py=5) THEN VaLidMove←
IF hysq THEN InVaLidMove←
WHILE (px=5 AND py=5) OR piece(px,py)<>Ø←
px=INT(RND*9+1):py=INT(RND*9+1)←
WEND←
WAVE Ø,n:FOR i=25Ø TO Ø STEP -2:SOUND 1ØØ+i*2,.Ø3,i,
Ø:NEXT:WAVE 1,n←
GOSUB VaLidMove:FOR i=Ø TO 25Ø STEP 2:SOUND 1ØØ+5ØØ-
i*2,.Ø3,i,1:NEXT←
hysq=1:GOSUB PutShape←
RETURN←
HyperCube:←
nx=INT(RND*9+1):ny=INT(RND*9+1)←
IF (nx=5 AND ny=5) OR piece(nx,ny)<>Ø THEN HyperCube
←
WAVE Ø,n:FOR i=25Ø TO Ø STEP -2:SOUND 1ØØ+i*2,.Ø3,i,
Ø:NEXT:WAVE 1,n←
piece(nx,ny)=piece(px,py):orient(nx,ny)=orient(px,py
):cLr(nx,ny)=cLr(px,py)←
GOSUB VaLidMove:FOR i=Ø TO 25Ø STEP 2:SOUND 1ØØ+5ØØ-
i*2,.Ø3,i,1:NEXT←
GOSUB PutShape:piece(spx,spy)=Ø:cLr(spx,spy)=Ø:px=nx
:py=ny←
RETURN←
 ←
VaLidMove:←
piece(px,py)=piece:orient(px,py)=rot:cLr(px,py)=cLr(
spx,spy)←
IF dx>Ø OR dy>Ø THEN piece(spx,spy)=Ø:cLr(spx,spy)=Ø
←
RETURN←
InVaLidMove:←
px=spx:py=spy:moves=Ø:RETURN←
←
Confirm:←
WINDOW 2,,(114,82)-(216,1Ø5),Ø,1:WINDOW OUTPUT 2:PRI
```

```
NT"Are you sure?"←
COLOR 3,2:LINE(27,14)-STEP(16,10),,b:PAINT(28,15),2,
3:LOCATE 3,5:PRINT"Y"←
LINE(59,14)-STEP(16,10),,b:PAINT(68,15),2,3:LOCATE 3
,9:PRINT"N"←
CkCon:WHILE MOUSE(0)>-1:WEND:x=MOUSE(3):y=MOUSE(4):c
o=POINT(x,y)←
IF NOT(co=2 OR co=3) THEN CkCon←
IF x>27 AND x<43 THEN←
c=1←
ELSEIF x>59 AND x<75 THEN←
c=0←
ELSE←
GOTO CkCon←
END IF←
WINDOW CLOSE 2:WHILE MOUSE(0)<>0:WEND:RETURN←
←
Options:←
co=POINT(x,y):moves=0←
IF NOT(co=2 OR co=3) THEN MovePiece←
IF y>133 AND y<145 AND fired AND L(pL) THEN←
fired=0:PALETTE 10,1,1,0:PALETTE 15,1,1,0←
WAVE 2,sq:WAVE 3,sq:ON TIMER(1) GOSUB LSnd:Lv=200:GO
SUB LSnd:TIMER ON←
GOSUB Laser:moves=1←
PALETTE 10,.6,.6,.6:PALETTE 15,.15,.25,.95←
ELSEIF y>93 AND y<105 THEN←
GOSUB Confirm:IF c THEN Restart←
ELSEIF y>53 AND y<66 THEN←
GOSUB Confirm:IF c THEN SCREEN CLOSE 1:WINDOW CLOSE
3:CLEAR,25000:END←
END IF←
GOTO EndMove←
←
LSnd:SOUND 120,18.2,Lv,2:SOUND 121,18.2,Lv,3:RETURN←
←
Border:LINE(40,10)-(288,186),,b:LINE(42,12)-(286,184
),,b:RETURN←
←
Restart:←
COLOR ,0:CLS:FOR i=1 TO 9:FOR j=1 TO 9:piece(i,j)=0:
cLr(i,j)=0:NEXT j,i←
GOTO Start←
←
EGOpt:←
co=POINT(x,y)←
IF co=2 OR co=3 THEN←
IF y>93 AND y<105 THEN Restart←
IF y>53 AND y<66 THEN SCREEN CLOSE 1:WINDOW CLOSE 3:
CLEAR,25000:END←
END IF←
```

```
GOTO EndGWait←
←
EndGame:←
IF k=3 THEN←
COLOR 1Ø,Ø:GOSUB Border:COLOR 11:LOCATE 1,19:PRINT"D
raw"←
ELSE←
IF k=2 THEN COLOR 4,Ø:GOSUB Border:COLOR 5:LOCATE 1,
16:PRINT"Red";←
IF k=1 THEN COLOR 6,Ø:GOSUB Border:COLOR 7:LOCATE 1,
15:PRINT"Green";←
PRINT" victory"←
END IF←
EndGWait:←
WHILE MOUSE(Ø)>-1:WEND:x=MOUSE(3):y=MOUSE(4)←
px=INT((x-8)/27):py=INT((y+6)/19):moves=Ø←
IF NOT((px>Ø AND px<1Ø) AND (py>Ø AND py<1Ø)) THEN E
GOpt←
GOTO EndGWait←
```

# Karma

Todd Heimarck and Rhett Anderson

*"Karma" is a mind-twisting strategy game for two players. You control the collective morale of a society. Depending on your ability, the people thrive or just survive. The program demonstrates the power of the Amiga hardware and Amiga Basic. At least 512K of RAM is required.*

Imagine that you have the power to make people very happy. Perhaps you're a vice president in charge of awarding college scholarships. Or you're a billionaire who enjoys giving someone ten thousand bucks. Or maybe you just have a nice smile.

Paradoxically, while you're being altruistic and are dispensing gifts to a grateful and increasingly happy world, you're greedy, too. You want to gain the approval and adoration of the beneficiaries of your largesse. You want people to like you.

Unfortunately, there's another philanthropist who has the same power as you. While you're dispensing your gifts and making people happy, your opponent is doing the same thing. You're locked in a popularity contest from which only one victor will emerge.

"Karma" is a two-player strategy game in which you and your opponent struggle for territory. Four different scenarios—each with a different goal—are included. Players take turns using the mouse to add happiness to households. When a certain level of happiness builds up, an explosion takes place. When one of the players achieves an explosion, that player captures all of the surrounding regions. Karma is easy to play, but difficult to win.

## Getting Started

Karma is written in Amiga Basic. Type it in and save a copy to disk. When you're ready to play the game, run it. When you play, you'll first be asked to choose one of the four karmic variations: Capture All, Four Corners, Two Pies, or 2500 Points. Game play is identical for each game, although the goal is different. To select a game, press one of the number keys (1–4) from the keyboard or the numeric keypad. The standard game is Capture All, which you select by pressing the 1 key.

## Levels of Happiness

The screen is divided into three parts: the big map, the small map, and the scoreboard. The small map shows you which player owns which territories. The big map on the left contains the most important information; it tells you the relative levels of happiness within each household in the city of Karma. Table 1-1 shows the color of the moods that you'll see.

**Table 1-1. The Color of Moods in Karma**

| Level | Mood | Color |
|---|---|---|
| 1 | Gloomy | Deep Blue |
| 2 | Content | Deep Purple |
| 3 | Pleased | Maroon |
| 4 | Joyous | Red |
| 5 | Ecstatic | Bright Red |

The black player moves first; white, second. During your turn, you may move the mouse pointer to any household on the big map, but the household must be on your side. Click the left button once (you may have to hold down the button for a second or two to make sure the click registers).

Whichever block you select will instantly increase one step in happiness. A blue transforms to purple, purple becomes maroon, and so on.

It may strike you that you're not gaining a lot of popularity if you can give happy points only to the households that are already on your side. You click the mouse pointer on your followers, and your opponent clicks on his or her followers. How do you move into neutral (or unfriendly) territory? Good question.

## The Power of Gossip

The levels of glee stop at ecstatic; there is no more blissful state. That's because ecstasy has a curious effect on the citizens of Karma. When their happiness hits level 5, they immediately tell all of their next-door neighbors. This is known as a *gossip explosion.* Three things happen: The ecstatic household drops back down to a lower level of glee (1, 2, or 3, depending on the type of house), but at the same time, each of the neighbors jumps up one level in happiness. The neighbors also move over to your side. If you watch the two maps, you'll see the happy colors change on the big map. You'll also see your own color spread outward on the smaller map. Table 1-2 shows the colors for each player.

**Table 1-2. The Players and Their Colors**

| Player | Color |
|---|---|
| Player 1 | Black |
| Player 2 | White |
| Neutral | Gray |

As the game begins, a majority of cells are neutral, but once a household is converted to one side or the other, it can never again become neutral.

You win and lose games by controlling strategically located joyous households. If you click on a red piece, it affects all of the neighboring pieces. If a neighbor is also joyous, it explodes. It's fairly common to see long strings of chain reactions as gossip spreads through a block of neighbors and gradually affects every house in the city.

As you plan your strategy, remember this: If you own a joyous Karmalite, color red, and your own Karmalite lives next door to another joyous Karmalite on your enemy's side, either one of you can capture both of them (plus all of their neighbors).

## From Condos to Suburbs

The city of Karma offers elegant living arranged as four types of dwelling units as shown in Table 1-3.

**Table 1-3. Points for Happiness**

| Unit | Points | Minimum Happiness |
|---|---|---|
| Condos | 3 | Content/2 |
| Houses | 4 | Gloomy/1 |
| Ranches | 4 | Gloomy/1 |
| Estates | 2 | Pleased/3 |

The condos appear on the screen as four pie-shaped units of eight wedge-shaped condos. Each condo has three neighbors and is worth three points. A group of eight condos looks circular like a pie and is commonly referred to as a *condo pie.*

Houses and ranches have four neighbors and a value of 4. There are nine houses, which are square in shape. The house at the very top of the city is connected with the house on the southern edge. Likewise, the east and west houses are neighbors. The eight ranches are the five-sided shapes on the fringe of Karma. Each ranch borders on two houses, one condo, and an estate.

In the outer corners, you'll see the four estates. They have only two neighbors (both of which are ranches) and are worth two points.

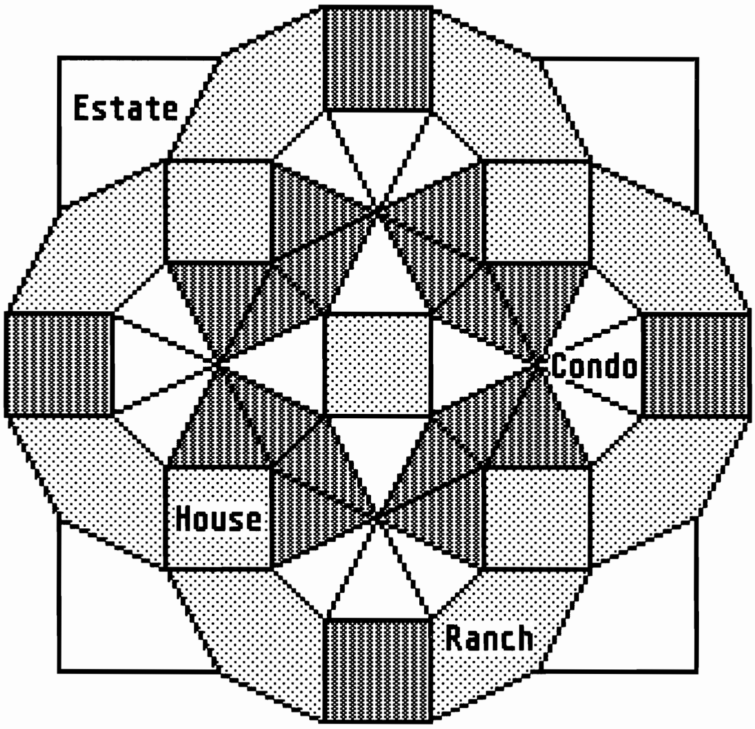
## Scoring and Winning

At the end of each turn, both players are awarded popularity points according to which households they've swayed to their sides. The points accumulate as the game progresses. If you control 12 condos, three houses, a ranch, and two estates, you'll gain 56 points: $(12 \times 3) + (3 \times 4) + (1 \times 4) + (2 \times 2)$.

Underneath the score is a second number that indicates how many households are on your side. If this number dwindles to 0, the game automatically ends, because you can only click on households you currently own. If you don't own any, you can't make a move.

In the first three games, the points are irrelevant, except to provide the loser with some consolation in the case that he

**Figure 1-4. The Karma Playfield**



*There are four types of properties in "Karma."*

or she loses while leading in points. The fourth game (2500 points) is just what you might think. The first person to reach 2500 wins.

In game 1 (Capture All), the goal is to send your opponent packing. As soon as one player has no more friendly households, the game ends.

Game 2 (Two Pies) takes a little less time, since the purpose is to capture two complete eight-unit condo pies. There are four condo blocks, so you might believe a tie—two blocks each—could happen, but it's impossible. Say player 1 made a move that yielded complete control of two blocks (16 condos) and that the other player also owned two blocks at the end of the turn. Player 2 can't capture any cells during player 1's turn, so for a tie to occur, player 2 would have had to own two complete blocks before player 1 started his or her turn. But in that case, player 2 would have won the game before player one moved the mouse. Ties are impossible.

In the Four Corners game (game 3), your aim is to capture all four corner estates. Each corner has only two neighbors, so this is a game where defense is crucial. Once you control a corner, you can—and should try to—hold on to it for as long as you can.

## Strategies and Tactics

The joyous households are on the verge of exploding with gossip, so watch them. At the beginning of Karma, you may want to set off several strategic explosions, in order to gain more territory to develop.

In the middle game, push a few isolated cells (households in an unhappy neighborhood) up to the red level and then leave them as an investment in the future. There's nothing worse than setting off a chain reaction that leaves the board in a situation where your opponent simply replies with another chain reaction that decimates your troops. If you have nothing but blues, you can't do much to get back.

The final few moves are crucial. You'll often see a city where one move creates a small chain reaction, while another move removes your opponent from play.

Although reds are primed to explode, maroons will often receive gossip from two directions. If three reds are immediate neighbors, all three will explode. If a maroon is next to two of the reds, it will receive gossip from two directions and will also explode.

## Karma

Filename: KARMA

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'Copyright 1987←
'COMPUTE! Publications, Inc.←
'All Rights Reserved.←
←
DEFINT a-z:DEFSNG r,g,b←
DIM sides(52),xcord(52,5),ycord(52,5),numadjacents(5
2),neighbors(52,3)←
DIM owner(52),renter(52),update(52),start(20),xfind(
52),yfind(52)←
DIM r(15),g(15),b(15),ToDo(100)←
gamenum=1←
←
RANDOMIZE TIMER←
←
SCREEN 1,320,200,4,1:WINDOW 3,"",(0,0)-(311,186),16,
1:WINDOW OUTPUT 3←
←
newgame:←
←
COLOR 1,0:CLS:score(1)=10:score(2)=10←
COLOR 1,2:LOCATE 8,13:PRINT"     Karma     "←
COLOR 1,0:PRINT:PRINT"  Copyright 1987 Compute! Publ
., Inc."←
PRINT"          All Rights Reserved"←
PRINT:PRINT:COLOR 1,2←
PRINT"              Choose game.              ":COLOR
 0,1:PRINT←
←
PRINT "              1. Capture All              "←
PRINT "              2. Four Corners             "←
PRINT "              3. Two Pies                 "←
PRINT "              4. 2500 Points              "←
←
GetAKey:←
a$=INKEY$:IF a$="" THEN GetAKey←
IF a$<"1" OR a$>"4" THEN GetAKey←
gamenum=VAL(a$):LOCATE 21,19:PRINT " "a$" " ←
←
RESTORE findpoints←
FOR i=0 TO 52←
 READ x,y:xfind(i)=x*10:yfind(i)=y*10←
NEXT i←
←
RESTORE Karma←
FOR i=0 TO 52←
 READ sides(i)←
 FOR ii=Ø TO sides(i)-2←
  READ xc,yc←
  xcord(i,ii)=xc*12          ←
  ycord(i,ii)=yc*12←
 NEXT ii←
 READ numadjacents(i)←
 FOR ii=Ø TO numadjacents(i)-1←
  READ neighbors(i,ii)←
 NEXT ii←
NEXT i←
←
RESTORE thecolors←
FOR i=Ø TO 15←
 READ r,g,b:r(i)=r/1ØØ:g(i)=g/1ØØ:b(i)=b/1ØØ←
 PALETTE i,r(i),g(i),b(i)←
NEXT i←
←
thecolors:←
←
DATA 5Ø,4Ø,3Ø←
DATA 16,16,16←
DATA Ø,Ø,Ø  ←
DATA Ø,5,4Ø←
DATA 25,5,3Ø←
DATA 5Ø,5,2Ø←
DATA 75,5,1Ø←
DATA 1ØØ,5,Ø←
DATA 1ØØ,55,Ø←
DATA 3Ø,3Ø,3Ø←
DATA Ø,Ø,Ø←
DATA 7Ø,7Ø,7Ø←
DATA Ø,Ø,Ø←
DATA Ø,Ø,Ø←
DATA Ø,Ø,Ø←
DATA Ø,Ø,Ø←
←
COLOR 1,Ø:FOR i=Ø TO 24:PRINT:NEXT i←
←
LOCATE 1,8:COLOR 1Ø,11:PRINT" K a r m a "←
COLOR 11,1Ø:LOCATE 1,25←
←
IF gamenum=1 THEN PRINT " Capture All "←
IF gamenum=2 THEN PRINT " Four Corners "←
IF gamenum=3 THEN PRINT " Two Pies "←
IF gamenum=4 THEN PRINT " 25ØØ Points "←
←
RESTORE start←
FOR i=Ø TO 19←
 READ start(i)←
NEXT i←
←
```

```
start:←
DATA 4,13,5,14,11,15,23,33,24,34,25,35,18,8,30,20,37
,36,39,38 ←
←
FOR i=0 TO 52←
 owner(i)=0:renter(i)=0:update(i)=0←
NEXT i←
←
FOR i=0 TO 19 ←
 owner(start(i))=(i AND 1)+1:renter(start(i))=1←
NEXT i←
←
FOR i=0 TO 52←
 GOSUB DoOne←
NEXT i←
←
player=2:play$(1)="black":play$(2)="white"←
←
game:←
←
player=3-player←
LOCATE 7,25:COLOR 9+player,9:PRINT " ";play$(player)
;"'s turn "←
←
←
←
loop:←
←
WHILE MOUSE(0)=0:WEND←
x=MOUSE(1):y=MOUSE(2):hue=POINT(x,y)←
IF hue<3 OR hue>8 THEN loop←
PALETTE 15,r(hue),g(hue),b(hue)←
PAINT (x,y),15,2←
←
which=-1←
FOR i=0 TO 52←
 IF POINT(xfind(i),yfind(i))=15 THEN which=i←
NEXT i←
IF which<0 THEN STOP←
←
IF owner(which)<>player THEN PAINT (x,y),hue,2:GOTO
loop←
←
SOUND WAIT←
SOUND 130,10,,0:SOUND 130.5,10,,2←
SOUND RESUME←
←
FOR real=0 TO 1 STEP .02←
 h=hue:r=real←
 PALETTE 15,r(h)+.25*r,.05,b(h)-.1*r←
NEXT real←
←
```

```
MaxToDo=Ø←
←
again:←
←
renter(which)=renter(which)+1←
IF renter(which)+1>numadjacents(which) THEN ←
 FOR i=Ø TO numadjacents(which)-1←
  MaxToDo=MaxToDo+1:t=neighbors(which,i):ToDo(MaxToD
o)=t←
  REM  PAINT (xfind(t),yfind(t)),POINT(xfind(t),yfin
d(t))+1,2←
 NEXT i:SOUND WAIT:SOUND 2ØØ+which*16,1,,Ø:SOUND 2ØØ
+which*8,1,,2:SOUND RESUME ←
 renter(which)=renter(which)-numadjacents(which)←
END IF ←
←
i=which←
IF owner(i)=3-player THEN score(3-player)=score(3-pl
ayer)-1←
IF owner(i)<>player THEN score(player)=score(player)
+1←
owner(i)=player:GOSUB DoOne:SOUND 2ØØ+6*which,.15,8Ø
,1←
IF score(1)=Ø OR score(2)=Ø THEN gameover←
IF MaxToDo<>Ø THEN which=ToDo(MaxToDo):MaxToDo=MaxTo
Do-1:GOTO again←
←
WHILE MOUSE(Ø)<Ø:WEND←
←
IF gamenum=2 AND ((owner(Ø) AND owner(1) AND owner(2
) AND owner(3))<>Ø) THEN gameover←
IF gamenum=3 THEN←
 win1=Ø:win2=Ø←
 FOR j=Ø TO 3←
  garbage=owner(j*8+4)←
  FOR k=1 TO 7←
   garbage=owner(j*8+4+k) AND garbage←
  NEXT k←
  IF garbage=1 THEN win1=win1+1←
  IF garbage=2 THEN win2=win2+1←
 NEXT j←
 IF win1>=2 OR win2>=2 THEN gameover←
END IF   ←
FOR j=1 TO 2←
 FOR i=Ø TO 52←
  IF owner(i)=j THEN points(j)=points(j)+numadjacent
s(i)←
 NEXT i←
NEXT j←
LOCATE 23,25:COLOR 1Ø,Ø:PRINT points(1)←
LOCATE 23,32:COLOR 11,Ø:PRINT points(2)←
```

```
IF gamenum=4 AND (points(1)>2499 OR points(2)>2499)
THEN gameover←
 ←
GOTO game←
←
SCREEN CLOSE 1←
←
GOTO doIt←
←
END←
←
DoOne:←
←
si2=sides(i)-2:COLOR 7-(numadjacents(i)-renter(i)),Ø
←
AREA (xcord(i,si2)+12,ycord(i,si2)+12)←
FOR ii=Ø TO si2←
 AREA (xcord(i,ii)+12,ycord(i,ii)+12)←
NEXT ii←
AREAFILL←
COLOR 2,1←
PSET (xcord(i,si2)+12,ycord(i,si2)+12)←
FOR ii=Ø TO si2←
 LINE -(xcord(i,ii)+12,ycord(i,ii)+12)←
NEXT ii←
←
DoOne2:←
←
si2=sides(i)-2:COLOR owner(i)+9,1←
AREA (xcord(i,si2)/2+2Ø2,ycord(i,si2)/2+9Ø)←
FOR ii=Ø TO si2←
 AREA (xcord(i,ii)/2+2Ø2,ycord(i,ii)/2+9Ø)←
NEXT ii←
AREAFILL←
COLOR 1,1←
PSET (xcord(i,si2)/2+2Ø2,ycord(i,si2)/2+9Ø)←
FOR ii=Ø TO si2←
 LINE -(xcord(i,ii)/2+2Ø2,ycord(i,ii)/2+9Ø)←
NEXT ii←
RETURN←
←
gameover:←
FOR i=Ø TO 52←
 GOSUB DoOne2←
NEXT i ←
FOR i=Ø TO 4Ø←
 FOR j=Ø TO 3←
  SOUND RND*i*1Ø,2,,j←
 NEXT j←
NEXT i  ←
FOR i=4Ø TO Ø STEP -1←
```

```
 FOR j=Ø TO 3←
  SOUND RND*i*1Ø,2,,j←
 NEXT j←
NEXT i  ←
FOR i=Ø TO 1ØØØØ:NEXT i←
RUN←
←
Karma:←
←
DATA 5, 1,1, 4,1, 3,3, 1,4←
DATA 2, 43,44←
←
DATA 5, 1Ø,1, 13,1, 13,4, 11,3←
DATA 2, 45,46←
←
DATA 5, 11,11, 13,1Ø, 13,13, 1Ø,13←
DATA 2, 4Ø,47←
←
DATA 5, 1,1Ø, 3,11, 4,13, 1,13←
DATA 2, 41,42←
←
←
DATA 4, 6,2, 8,2, 7,4←
DATA 3, 5,11,39←
←
DATA 4, 8,2, 9,3, 7,4←
DATA 3, 4,6,45←
←
DATA 4, 9,3, 9,5, 7,4←
DATA 3, 5,7,5Ø←
←
DATA 4, 9,5, 8,6, 7,4←
DATA 3, 6,8,19←
←
DATA 4, 8,6, 6,6, 7,4←
DATA 3, 7,9,52←
←
DATA 4, 6,6, 5,5, 7,4←
DATA 3, 8,1Ø,29←
←
DATA 4, 5,5, 5,3, 7,4←
DATA 3, 9,11,49←
←
DATA 4, 5,3, 6,2, 7,4←
DATA 3, 4,1Ø,44←
←
←
DATA 4, 9,5, 11,5, 1Ø,7←
DATA 3, 13,19,5Ø←
←
DATA 4, 11,5, 12,6, 1Ø,7←
DATA 3, 12,14,46←
```

```
<
DATA 4, 12,6, 12,8, 10,7<
DATA 3, 13,15,36<
<
DATA 4, 12,8, 11,9, 10,7<
DATA 3, 14,16,47<
<
DATA 4, 11,9, 9,9, 10,7<
DATA 3, 15,17,51<
<
DATA 4, 9,9, 8,8, 10,7<
DATA 3, 16,18,21<
<
DATA 4, 8,8, 8,6, 10,7<
DATA 3, 17,19,52<
<
DATA 4, 8,6, 9,5, 10,7<
DATA 3, 7,12,18<
<
<
DATA 4, 6,8, 8,8, 7,10<
DATA 3, 21,27,52<
<
DATA 4, 8,8, 9,9, 7,10<
DATA 3, 17,20,22<
<
DATA 4, 9,9, 9,11, 7,10<
DATA 3, 21,23,51<
<
DATA 4, 9,11, 8,12, 7,10<
DATA 3, 22,24,40<
<
DATA 4, 8,12, 6,12, 7,10<
DATA 3, 23,25,37<
<
DATA 4, 6,12, 5,11, 7,10<
DATA 3, 24,26,41<
<
DATA 4, 5,11, 5,9, 7,10<
DATA 3, 25,27,48<
<
DATA 4,5,9, 6,8, 7,10<
DATA 3, 20,26,31<
<
<
DATA 4, 3,5, 5,5, 4,7<
DATA 3, 29,35,49<
<
DATA 4, 5,5, 6,6, 4,7<
DATA 3, 9,28,30<
<
```

```
DATA 4, 6,6, 6,8, 4,7←
DATA 3, 29,31,52←
←
DATA 4, 6,8, 5,9, 4,7←
DATA 3, 27,3Ø,32←
←
DATA 4, 5,9, 3,9, 4,7←
DATA 3, 31,33,48←
←
DATA 4, 3,9, 2,8, 4,7←
DATA 3, 32,34,42←
←
DATA 4, 2,8, 2,6, 4,7←
DATA 3, 33,35,38←
←
DATA 4, 2,6, 3,5, 4,7←
DATA 3, 28,34,43←
←
←
DATA 5, 12,6, 14,6, 14,8, 12,8←
DATA 4, 14,38,46,47←
←
DATA 5, 6,12, 8,12, 8,14, 6,14←
DATA 4, 24,39,4Ø,41←
←
DATA 5, Ø,6, 2,6, 2,8, Ø,8←
DATA 4, 34,36,42,43←
←
DATA 5, 6,Ø, 8,Ø, 8,2, 6,2←
DATA 4, 4,37,44,45←
←
←
DATA 6, 9,11, 11,11, 1Ø,13, 8,14, 8,12←
DATA 4, 2,23,37,51←
←
DATA 6, 3,11, 5,11, 6,12, 6,14, 4,13←
DATA 4, 3,25,37,48←
←
DATA 6, Ø,8, 2,8, 3,9, 3,11, 1,1Ø←
DATA 4, 3,33,38,48←
←
DATA 6, 1,4, 3,3, 3,5, 2,6, Ø,6←
DATA 4, Ø,35,38,49←
←
DATA 6, 4,1, 6,Ø, 6,2, 5,3, 3,3←
DATA 4, Ø,11,39,49←
←
DATA 6, 8,Ø, 1Ø,1, 11,3, 9,3, 8,2←
DATA 4, 1,5,39,5Ø←
←
DATA 6, 11,3, 13,4, 14,6, 12,6, 11,5←
DATA 4, 1,13,36,5Ø←
```

```
←
DATA 6, 12,8, 14,8, 13,1Ø, 11,11, 11,9←
DATA 4, 2,15,36,51←
←
←
DATA 5, 3,9, 5,9, 5,11, 3,11←
DATA 4, 26,32,41,42←
←
DATA 5, 3,3, 5,3, 5,5, 3,5←
DATA 4, 1Ø,28,43,44←
←
DATA 5, 9,3, 11,3, 11,5, 9,5←
DATA 4, 6,12,45,46←
←
DATA 5, 9,9, 11,9, 11,11, 9,11←
DATA 4, 16,22,4Ø,47←
←
←
DATA 5, 6,6, 6,8, 8,8, 8,6←
DATA 4, 8,18,2Ø,3Ø←
←
findpoints:←
←
DATA 4,4, 16,5, 16,15, 4,15←
DATA 1Ø,4, 11,5, 11,6, 11,7, 1Ø,8, 9,7, 8,6, 9,5←
DATA 13,8, 14,9, 15,1Ø, 15,11, 13,11, 12,11, 12,1Ø,
12,8←
DATA 1Ø,11, 11,12, 11,13, 11,15, 1Ø,15, 9,14, 8,13,
8,12←
DATA 6,8, 7,9, 8,1Ø, 8,11, 6,11, 5,11, 5,1Ø, 5,8←
DATA 17,1Ø, 1Ø,17, 3,1Ø, 1Ø,3←
DATA 13,16, 7,16, 4,13, 4,7, 7,3, 12,3, 16,7, 16,12←
DATA 6,13, 6,6, 13,6, 13,13←
DATA 1Ø,1Ø←
              ←
```

# Rememory

Charles Harbert
*Translation by Tim Midkiff*

*How good is your memory? Find out for sure as you're tested by the Amiga in this Concentration-style thinking game. Your goal: match the pairs of colorful graphic images with one another. "Rememory" lets you test your powers of recall against the computer or a friend.*

"Rememory" is a game that will push your powers of concentration and memorization to the limit. Type in the program listed and save it. When you're ready to play, simply load and run the program.

## Playing Rememory

Rememory is played on a grid containing 54 boxes arranged in a 9 × 7 matrix. Each box contains a graphics shape, and there are many matching shapes within the grid. The object of the game is to find all of the matches in the playing grid by selecting any two boxes at a time.

The mouse pointer indicates your current position on the game screen. Move the mouse pointer to the desired box and press the left mouse button to select it. When you click on the box, the computer displays the shape which it contains. In the two-player game, the colors of the window border change to indicate whose turn it is. To add to the interest and difficulty, the program uses color cycling to change the colors of the graphics shapes.

A turn consists of two selections. After you select both boxes, the computer displays both of them briefly. If the two shapes you select are identical, you have scored one match, and those shapes remain visible on the board. If the shapes do not match, the computer erases them, and it's your job to remember where those shapes were located. The computer scrambles the shapes at the beginning of each game, so you won't know where a given shape is found until you uncover it.

Rememory can be played with one or two players. When you play alone, the object is to match all the shapes in the fewest number of turns. For a two-player game, the goal is to score more matches than your opponent. You get an extra turn every time you succeed in making a match. If you set a time limit for each move (for instance, 20 or 30 seconds), Rememory can be a fast-paced, exciting, two-player game.

When you run the program, it asks how many players will play the game. Enter the number of players—either 1 or 2. Then the program asks how many matches will be required to finish the game. If you enter the maximum number, 27, you will have to match every pair of shapes in the grid to finish. If you choose a lower number, the game ends when you achieve the designated number of matches. The right side of the screen displays the current score.

**Rememory**
Filename: REMEMORY

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.←
'  All Rights Reserved←
←
DEFINT a-z:DEFSNG r,g,b,mx:RANDOMIZE TIMER:SCREEN 1,
320,200,5,1←
WINDOW 3,"",(0,0)-(311,186),16,1:WINDOW OUTPUT 3←
DIM bn(5,8),cb(26),r(11),b(11),df(7),aLt(7),hor(7),v
er(7),sL(7),rn(7),ck(7)←
RESTORE PaletteData:FOR i=0 TO 15:READ r,g,b:PALETTE
 i,r,g,b:NEXT←
FOR i=20 TO 21:READ r,g,b:PALETTE i,r,g,b:NEXT←
FOR i=0 TO 5:READ r(i),b(i):PALETTE i+22,r(i),0,b(i)
:NEXT←
FOR i=0 TO 5:r(11-i)=b(i):b(11-i)=r(i):NEXT←
PaletteData: ←
DATA 0,0,0,.5,.5,.5,.5,.5,.5,.6,0,0←
DATA 0,.6,0,.6,.6,0,.6,0,.6,0,.6,.6←
DATA 0,0,.6,.9,.9,.9,.9,0,0,0,.9,0←
DATA 0,0,.9,.9,.9,0,.9,0,.9,.5,.5,.5←
DATA 0,0,0,0,0,0←
DATA .6,0,.8,0,1,0,.8,0,.6,0,.5,.3←
FOR i=0 TO 26:READ cb(i):NEXT←
DATA 9,4,13,15,3,0,9,6,12,9,4,20,13,0,0,0,22,12,9,13
,15,15,9,8,9,3,10←
```

```
e$=SPACE$(3):ON TIMER(1) GOSUB CycLe←
FOR i=Ø TO 7←
df(i)=&HFFFF:ver(i)=&HAAAA:rn(i)=RND*&HFFFF←
IF (i AND 1) THEN hor(i)=&HFFFF:aLt(i)=&HAAAA ELSE a
Lt(i)=&H5555←
IF (i AND 4) THEN ck(i)=&HFØFØ ELSE ck(i)=&HFØF←
NEXT←
FOR i=Ø TO 3:READ sL(i):sL(i+4)=sL(i):NEXT←
DATA &H3333,&H6666,&hcccc,&H9999←
Start:←
COLOR 2,Ø:CLS:GOSUB InPLayers:LOCATE 13,9:GOSUB InMa
tches←
GOSUB RandBoard:mf=Ø:ts=Ø:FOR i=Ø TO 1:tr(i)=Ø:sc(i)
=Ø:NEXT←
GOSUB DrawBoard:pL=Ø:m=1:sw=Ø:TIMER ON←
WHILE ts<nm←
IF np THEN PALETTE 1,.6*(1-pL),.6*pL,Ø←
GOSUB SelectBox:GOSUB ShowPic:r1=ro:c1=co:GOSUB Sele
ctBox:GOSUB ShowPic←
IF bn(r1,c1)=bn(ro,co) THEN←
sc(pL)=sc(pL)+1:ts=ts+1:m=Ø←
ELSE←
FOR i=1 TO 4ØØØ:NEXT:GOSUB HidePic←
END IF←
tr(pL)=tr(pL)+1:GOSUB UpdateScore:IF m THEN pL=pL XO
R np←
m=1←
WEND←
COLOR 15,Ø:LOCATE 9,1Ø:PRINT "Another game (y/n)?"←
EndLp: k$=UCASE$(INKEY$):IF k$="Y" THEN Start←
IF k$="N" THEN TIMER OFF:SCREEN CLOSE 1:WINDOW CLOSE
 3:END ELSE EndLp←
←
SelectBox:←
WHILE MOUSE(Ø)=Ø:WEND:px=MOUSE(1):py=MOUSE(2):WHILE
MOUSE(Ø)<>Ø:WEND←
IF POINT(px,py)<>2 THEN SelectBox←
px=px AND &HFFEØ:py=py AND &HFFEØ:ro=INT(py/32):co=I
NT(px/32):py=py+8           ←
RETURN←
←
ShowPic:←
n=bn(ro,co):COLOR ,cb(n):GOSUB Hide←
ON n+1 GOTO 1,1,1,2,3,4,5,6,7,8,9,1Ø,11,12,13,14,15,
16,17,18,19,2Ø,21,22,23,24,1←
LOCATE cy,cx:PRINT STR$(bn(ro,co))←
1 RETURN←
2 COLOR Ø:LOCATE cy+1,cx+1:PRINT CHR$(214):PAINT(px+
12,py+12):RETURN←
3 COLOR 13:a$=CHR$(191)+CHR$(63):LOCATE cy,cx:PRINT
a$CHR$(191)←
```

```
LOCATE cy+1,cx:PRINT CHR$(63)a$:LOCATE cy+2,cx:PRINT
 a$CHR$(191);:RETURN ←
4 PATTERN ,sL:FOR i=Ø TO 7:COLOR i+7←
LINE(px+2*i,py+2*i)-(px+23-2*i,py+23-2*i),,bf:NEXT:P
ATTERN ,df:RETURN←
5 COLOR 1Ø:AREA(px+2,py+12):AREA STEP(1Ø,-1Ø):AREA S
TEP(1Ø,1Ø)←
AREA STEP(-1Ø,1Ø):AREA STEP(-1Ø,-1Ø):PATTERN ,hor:AR
EAFILL:PATTERN ,df:RETURN←
6 COLOR 14:GOSUB Triangle:AREAFILL:RETURN ←
7 COLOR 13:CIRCLE(px+19,py+5),2:PAINT(px+2Ø,py+5)←
PATTERN ,sL:GOSUB Triangle:AREAFILL:PATTERN ,df:RETU
RN←
8 COLOR ,3:LOCATE cy,cx+1:PRINT SPACE$(1):LOCATE cy+
1,cx:PRINT e$←
LOCATE cy+2,cx+1:PRINT SPACE$(1);:RETURN←
9 FOR i=Ø TO 2:FOR j=Ø TO 2:COLOR 2Ø+((i+j) AND 1):x
=px+5*i+7:y=py+5*j+7←
CIRCLE(x,y),2:NEXT j,i:RETURN←
1Ø COLOR 21:GOSUB Box:PATTERN ,aLt:AREAFILL:PATTERN
,df:RETURN←
11 COLOR 6:GOSUB Box:PATTERN ,sL:AREAFILL:PATTERN ,d
f:RETURN←
12 COLOR 14:PATTERN ,hor:x=px+4:y=py+4:GOSUB Diamond
:AREAFILL←
x=px:y=py+12:GOSUB Diamond:AREAFILL←
x=px+12:y=py+16:GOSUB Diamond:AREAFILL:PATTERN ,df:R
ETURN←
13 FOR i=Ø TO 11:COLOR (i MOD 6)+22:LINE(px,py+2*i)-
(px+23,py+23-2*i):NEXT←
RETURN←
14 FOR i=Ø TO 11:COLOR (i MOD 6)+22:LINE(px,py)-(px+
2*i,py+23-2*i)←
LINE(px+23,py+23)-(px+2*i,py+23-2*i):NEXT:RETURN←
15 COLOR 24:GOSUB Box:PATTERN ,aLt:AREAFILL:COLOR ,Ø
←
LOCATE cy+1,cx+1:PRINT SPACE$(1):PATTERN ,df:RETURN←
16 COLOR 2Ø:x=px+8:y=py+12:GOSUB Diamond:AREAFILL:x=
x+4←
CIRCLE(x,y),2,21:PAINT(x,y),21,21:RETURN←
17 COLOR 12:GOSUB Triangle:PATTERN ,ver:AREAFILL:PAT
TERN ,hor:COLOR 9,12←
AREA(px,py):AREA STEP(23,Ø):AREA STEP(Ø,23):AREA STE
P(-23,-23)←
AREAFILL:PATTERN ,df:RETURN←
18 PATTERN ,rn:PAINT(px,py),22,Ø:PATTERN ,df:RETURN←
19 PATTERN ,ck:PAINT(px,py),Ø,Ø:PATTERN ,df:RETURN←
2Ø COLOR 15,Ø:LINE(px,py+7)-(px+23,py+7),Ø←
LOCATE cy+1,cx:PRINT CHR$(24Ø)CHR$(245)CHR$(24Ø):RET
URN←
```

```
21 FOR i=1 TO 11:LINE(px,py+i*2)-(px+23,py),10←
LINE(px,py+23)-(px+23,py+i*2),12:NEXT:RETURN←
22 FOR i=4 TO 20:LINE(px,py+i)-(px+23,py+i),(i MOD 4
)+28:NEXT:RETURN←
23 COLOR 0:LINE(px,py+17)-(px+6,py+9):LINE -STEP(6,2
):LINE -STEP(6,6)←
LINE -STEP(5,0):PAINT(px,py),0,0:LINE(px,py+16)-(px+
6,py+8),28←
LINE -STEP(6,2),29:LINE -STEP(6,6),30:LINE -STEP(5,0
),31:RETURN←
24 LINE(px,py+8)-(px+23,py+16),0:PAINT(px,py+23),5,0
:RETURN←
←
Triangle:←
AREA(px,py):AREA STEP(23,23):AREA STEP(-23,0):AREA S
TEP(0,-23):RETURN←
←
Diamond:←
AREA(x,y):AREA STEP(4,-4):AREA STEP(4,4):AREA STEP(-
4,4):AREA STEP(-4,-4)←
RETURN←
←
Box:←
AREA(px,py):AREA STEP(23,0):AREA STEP(0,23):AREA STE
P(-23,0)←
AREA STEP(0,-23):RETURN←
←
HidePic:←
COLOR ,2:GOSUB Hide:ro=r1:co=c1:GOSUB Hide:RETURN←
Hide: cx=4*co+1:cy=4*ro+2←
FOR i=0 TO 2:LOCATE cy+i,cx:PRINT e$;:NEXT:RETURN←
←
UpdateScore:←
COLOR 0,pL+3:pr=8*pL-4*np+13:LOCATE pr,37:s$=STR$(tr
(pL))←
PRINT RIGHT$("00"+RIGHT$(s$,LEN(s$)-1),3)←
LOCATE pr+2,37:s$=STR$(sc(pL))←
PRINT RIGHT$("00"+RIGHT$(s$,LEN(s$)-1),3)←
RETURN←
←
DrawBoard:←
CLS:COLOR ,2:FOR i=0 TO 23←
IF (i AND 3)<>0 THEN←
FOR j=0 TO 8:PRINT e$SPC(1);:NEXT←
END IF←
IF i<23 THEN PRINT←
NEXT←
FOR pL=0 TO np:COLOR 0,pL+3:FOR j=0 TO 6:LOCATE 8*pL
-4*np+10+j,37←
PRINT e$:NEXT:LOCATE 8*pL-4*np+11,37:PRINT STR$(pL+1
):GOSUB UpdateScore:NEXT←
```

```
RETURN←
←
RandBoard:←
i=Ø:FOR j=Ø TO 4 STEP 2:FOR k=Ø TO 8:bn(j,k)=i:bn(j+
1,k)=i:i=i+1:NEXT k,j←
FOR j=Ø TO 5:FOR k=Ø TO 8:sj=INT(RND*5):sk=INT(RND*9
)←
t=bn(sj,sk):bn(sj,sk)=bn(j,k):bn(j,k)=t:NEXT k,j←
RETURN←
←
InPLayers:←
LOCATE 11,9:PRINT "Number of players (1/2)?"←
GetKey:k$=INKEY$:IF k$="" OR (k$<>"1" AND k$<>"2") T
HEN GetKey←
np=VAL(k$)-1←
RETURN←
←
InMatches:←
INPUT "Number of matches (1-27)? ",s$←
nm=VAL(s$):IF nm<1 OR nm>27 THEN nm=27←
RETURN←
←
CycLe:←
nsw=sw XOR 1:PALETTE 2Ø,Ø,sw*.9,Ø:PALETTE 21,Ø,nsw*.
9,Ø←
sw=(sw+1) MOD 2:cc=(cc+1) MOD 12←
FOR cn=28 TO 31:PALETTE cn,1,1,1:NEXT:PALETTE (cc MO
D 4)+28,Ø,Ø,1←
FOR cn=Ø TO 5:ck=(cc+cn) MOD 12:PALETTE cn+22,r(ck),
Ø,b(ck):NEXT←
RETURN←
```

# Wari

Don Donati
*Translation by Pat Parrish*

*"Wari," based on an ancient strategy game, lets you match your wits against the computer. Move the counters around the board and capture the computer's pieces. A finely detailed BASIC program, Wari requires well-planned strategies in order to win. The program runs on any Amiga computer.*

Wari is a strategy game which has been played for centuries in Africa and the Middle East. The object of the game is to capture as many of your opponent's pieces as you can, while trying to prevent the capture of your own pieces. Type in the listing and save a copy before you run the program.

## Electronic Beans

Wari is played on a board which has 12 compartments arranged in two rows of 6 (the arrangement is similar to that of an egg carton). In the original versions of this game, the compartments were actual depressions in a board or simply holes scooped into the ground, and the game was played by moving counters (beans, pebbles, or other small objects) among the various compartments. In the computerized version of Wari, the compartments are rectangles drawn on the screen, and the counters are represented by numbers. If the number 4 appears in a compartment, that compartment holds four counters, and so on.

Six of the compartments are yours and the other 6 belong to your opponent, which is always the computer. When the game begins, four counters are placed in each of the 12 compartments, for a total of 48 counters. You each, then, start the game with 24 counters. Once play begins, however, counters are considered yours when they rest in one of the 6 compartments on your side of the board. Next, the program asks whether you want a limited game or an unlimited one. For limited games, a maximum number of moves are allowed. There's more about that below. The program then asks whether you or the computer should make the first move.

## Counterclockwise Movement

A move consists of taking all the counters from one compartment on your side of the board and sowing, or distributing, one counter into each of the adjacent compartments in a counterclockwise direction. In the original game, this was done by picking up the counters and sowing them by hand. In this version of Wari, you simply indicate which compartment you wish to sow by pressing the key corresponding to the letter printed by that compartment. The computer automatically sows that compartment's counters for you leaving the original compartment empty.

Depending on which compartment you choose, the sowing can wrap around from one side of the board to the other. Should you sow 12 or more counters, you return to the compartment where you started: In that case, the original compartment is skipped and sowing resumes in the next one.

## Captures

You score points in Wari by capturing counters. A capture occurs when you sow your last counter in the computer player's compartment which previously contained either one or two counters. The counters from that compartment are then removed from play. Each captured counter is equal to one point. If the previous compartment in line also contains two or three counters at the end of the move, its counters are captured, as well. This process continues until no more counters can be captured in that turn. In some cases, it's possible to capture all of the computer's counters in a single move.

The game can end in several different ways. Play must end whenever the board is empty (all counters have been captured) or whenever one player has no more counters to move. The game also ends when one player has captured more than half of the counters (if you have more than 24 counters, it's numerically impossible for the computer to win the game).

Wari also permits a stalemate, where you and the computer chase each other around the board fruitlessly; a stalemate game should be ended at your discretion.

At the end of the game, your score is increased by the number of counters remaining on your side of the board. The computer automatically totals the score and announces the winner. You may end the game at any time by pressing the Q key.

## Game Limits

As mentioned above, at the beginning of each game, the computer also asks whether you wish to play a limited or unlimited game. A limited game consists of a set number of moves for each player; an unlimited game lasts until the game ends in one of the ways described above. Press L for a limited game or U for an unlimited game.

Limits of 25–35 moves make for interesting games. In a very short limited game (say, 5 moves) it's usually more important to protect counters on your side of the board than to capture the computer player's counters; when the game ends, you are awarded all the counters on your side.

### Wari

Filename: WARI

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.←
'  All Rights Reserved←
←
setup:←
CLEAR ,25000←
CLEAR ,65536&←
SCREEN 1,320,200,3,1:WINDOW 3,"",(0,0)-(311,186),16,
1←
WINDOW OUTPUT 3←
PALETTE 0,0,0,0:PALETTE 1,.1,.2,.8   ' black, blue←
PALETTE 2,.33,.9,0:PALETTE 3,0,.93,.87   ' green, aq
ua←
PALETTE 4,1,.1,.27:PALETTE 5,.8,0,.93    ' red, purp
le←
PALETTE 6,1,1,.13:PALETTE 7,1,1,1   ' yellow, white←
DIM a(12),b(12),sc(80),i(80)←
RANDOMIZE TIMER:ds$=STRING$(39,32)←
DIM waveform%(255)←
```

```
wavedata=-128←
FOR n=Ø TO 255:waveform%(n)=wavedata←
wavedata=wavedata+1:NEXT←
WAVE Ø,waveform%←
←
restart:←
CLS:LOCATE 1,1Ø:COLOR 6,Ø:PRINT STRING$(21,42)←
PRINT TAB(1Ø)"*";:COLOR Ø,1:PRINT STRING$(19,32);←
COLOR 6,Ø:PRINT "*"←
PRINT TAB(1Ø)"*";:COLOR Ø,1:PRINT "    Amiga  Wari
   ";←
COLOR 6,Ø:PRINT"*" ←
PRINT TAB(1Ø)"*";:COLOR Ø,1:PRINT STRING$(19,32); ←
COLOR 6,Ø:PRINT "*"←
PRINT TAB(1Ø)STRING$(21,42)←
PRINT:COLOR 3:PRINT TAB(13)"Computer's Side":cp=Ø:mc
=Ø ←
COLOR 5:PRINT TAB(13)"Move #";:COLOR 1:PRINT mc←
COLOR 5:PRINT TAB(13)"Score:";:COLOR 4:PRINT cp←
COLOR 2:PRINT:PRINT "    <<F<<<<E<<<<D<<<<C<<<<B<<<<
A<<<"←
GOSUB dash:PRINT "    ";←
FOR i=1 TO 6:PRINT CHR$(124)"----";:NEXT:PRINT CHR$(
124)←
GOSUB dash←
COLOR 2:PRINT "    >>G>>>>H>>>>I>>>>J>>>>K>>>>L>>>"←
PRINT:COLOR 7:PRINT TAB(13)"Player's Side":pl=Ø:mp=Ø
←
COLOR 5:PRINT TAB(13)"Move #";:COLOR 1:PRINT mp←
COLOR 5:PRINT TAB(13)"Score:";:COLOR 4:PRINT pl←
FOR t=1 TO 12:a(t)=4:b(t)=4:GOSUB placepieces:NEXT t
←
←
gametype:←
LOCATE 23,1:COLOR Ø,6:PRINT"L";:COLOR 6,Ø←
PRINT "imited or ";:COLOR Ø,6:PRINT "U";←
COLOR 6,Ø:PRINT "nlimited game? ";←
←
type:←
GOSUB getchar:IF b$<>"L" AND b$<>"U" THEN type←
PRINT b$:FOR tm=1 TO 4ØØ:NEXT:ml=Ø←
IF b$="U" THEN COLOR Ø,1:LOCATE 4,16:PRINT "Unlimite
d":GOTO first←
←
moves:←
GOSUB cline:PRINT "Move Limit";:INPUT ml←
ml=INT(ml):IF ml<=Ø THEN moves←
LOCATE 4,15:COLOR Ø,1:PRINT "Limited:"+STR$(ml)←
←
first:←
LOCATE 23,1:COLOR 3,Ø:PRINT "Who goes first: ";:←
```

```
COLOR Ø,3:PRINT "C";:COLOR 3,Ø:PRINT "omputer or ";←
COLOR Ø,3:PRINT "P";:COLOR 3,Ø:PRINT "layer? ";←
←
getfirst:←
GOSUB getchar:IF b$<>"C" AND b$<>"P" THEN getfirst←
PRINT b$:FOR tm=1 TO 4ØØ:NEXT tm←
IF b$="C" THEN computer←
←
player:←
p$="P":p=Ø:sp=Ø:mp=mp+1←
COLOR 1:LOCATE 2Ø,19:PRINT mp←
FOR f=7 TO 12:p=p+a(f):NEXT f←
entry:←
COLOR 7:LOCATE 23,1:PRINT ds$:LOCATE 23,1←
PRINT "Player's turn. Move counters (G-L)? ";←
IF p=Ø THEN nocounters←
GOSUB getchar:i=ASC(b$)-64:COLOR 3:PRINT b$←
IF b$="Q" THEN quit←
IF b$>="G" AND b$<="L" THEN IF a(i)<>Ø THEN okmove←
COLOR 4:GOSUB cline:PRINT "Illegal move!!!"←
FOR tm=1 TO 15ØØ:NEXT:GOTO entry←
okmove:←
ds=1:GOSUB movecounters:FOR tm=1 TO 9ØØ:NEXT tm←
pt=Ø:FOR f=1 TO 12:a(f)=b(f):pt=pt+a(f):NEXT f←
IF mp=ml AND mc=ml THEN award←
IF pt=Ø THEN nocounters←
IF pl>24 THEN award←
←
computer:←
p$="C":pa=Ø:pb=Ø:ia=Ø:ib=Ø:p=Ø:mc=mc+1←
COLOR 1:LOCATE 8,19:PRINT mc←
FOR f=1 TO 6:p=p+a(f):NEXT f←
COLOR 3:GOSUB cline←
PRINT "Computer's turn (Move counters A-F). ";←
IF p=Ø THEN nocounters←
←
checkmoves:←
FOR g=1 TO 12←
sc(g)=Ø:i(g)=Ø←
IF g=7 THEN p$="P"←
IF a(g)=Ø THEN skip←
sc=Ø:sp=Ø:i=g←
ds=Ø:GOSUB movecounters←
sc(g)=s1 OR s2:i(g)=tØ←
skip:←
NEXT g←
p$="C"←
←
pickbest:←
FOR f=1 TO 6←
IF pa>=sc(f) THEN ahead←
```

```
ia=f:pa=sc(f)←
ahead:←
IF pb>=sc(f+6) OR a(i(f+6))=Ø THEN skip2←
ib=i(f+6):pb=sc(f+6)←
skip2:←
NEXT f←
IF ia=Ø AND ib=Ø THEN skip3←
i=ia:IF pb>pa THEN i=ib←
GOTO printit←
skip3:←
p=Ø:FOR f=7 TO 12:p=p+a(f):NEXT f←
IF p<>Ø THEN random←
FOR f=1 TO 5←
IF a(f)<>Ø AND a(f)<=6-f THEN i=f:GOTO printit←
NEXT f←
random:←
i=INT(RND*6)+1:IF a(i)=Ø THEN random←
←
printit:←
COLOR 7:PRINT CHR$(i+64):sc=Ø←
ds=1:GOSUB movecounters:FOR tm=1 TO 15ØØ:NEXT tm←
pt=Ø:FOR f=1 TO 12:a(f)=b(f):pt=pt+a(f):NEXT f←
IF mp=ml AND mc=ml THEN award←
IF pt=Ø THEN nocounters←
IF cp>24 THEN award←
GOTO player←
←
nocounters:←
FOR tm=1 TO 6ØØ:NEXT←
COLOR 4:GOSUB cline:PRINT "No counters!!! ";←
gameend:←
IF pt=Ø THEN PRINT "Game over."←
FOR tm=1 TO 15ØØ:NEXT←
IF pt=Ø THEN winner←
←
award:←
COLOR 4:GOSUB cline:PRINT "Game over. ";←
COLOR 6:PRINT "Award counters."←
FOR f=1 TO 6←
p$="C":sc=a(f):t=f:b(t)=Ø:GOSUB placepieces ←
p$="P":sp=a(f+6):t=f+6:b(t)=Ø:GOSUB placepieces←
NEXT←
←
winner:←
GOSUB cline←
COLOR 3:IF pl=cp THEN PRINT "A draw. ";:GOTO another
←
IF pl>cp THEN COLOR 2:PRINT "Player wins. ";:GOTO an
other←
COLOR 4:PRINT "Computer wins. ";←
←
```

```
another:←
COLOR 7:PRINT "Another game (Y/N)? ";←
another2:←
GOSUB getchar:IF b$<>"Y" AND b$<>"N" THEN another2←
PRINT b$:IF b$="Y" THEN restart←
WINDOW CLOSE 3←
SCREEN CLOSE 1←
WINDOW 1,"Wari",,31,-1←
CLEAR ,25000←
END  ←
←
quit:←
FOR tm=1 TO 200:NEXT←
COLOR 4:GOSUB cline←
PRINT "Quit game. Are you sure (Y/N)? ";←
again:←
GOSUB getchar:IF b$<>"Y" AND b$<>"N" THEN again←
IF b$<>"Y" THEN entry←
GOSUB cline:pt=0:GOTO gameend←
←
placepieces:←
COLOR 7←
IF t<7 THEN tb=36-5*t:GOTO place←
IF t>6 THEN tb=5*(t-6)+1←
place:←
LOCATE 13-2*(t>6),tb:PRINT b(t)←
FOR tm=1 TO 1000:NEXT tm←
IF b(t)<>0 THEN RETURN←
IF p$="P" THEN GOSUB playerscore:RETURN←
IF p$="C" THEN GOSUB computerscore:RETURN←
←
movecounters:←
t=i:s1=0:s2=0←
FOR f=1 TO 12:b(f)=a(f):NEXT←
b(t)=0:IF ds THEN GOSUB placepieces←
FOR f=1 TO a(t)←
t=t+1←
IF t>12 THEN t=1←
b(t)=b(t)+1:IF ds THEN GOSUB placepieces←
NEXT f:t0=t←
←
captures:←
IF b(t0)<2 OR b(t0)>3 THEN RETURN←
IF p$="P" AND t0<=6 THEN total←
IF p$="C" AND t0>=7 THEN total←
RETURN←
←
total:←
ls=1:IF p$="C" THEN ls=7←
FOR f=t0 TO ls STEP -1←
IF b(f)<2 OR b(f)>3 THEN RETURN←
IF p$="P" THEN sp=b(f):s2=s2+sp:GOTO total2←
IF p$="C" THEN sc=b(f):s1=s1+sc←
total2:←
b(f)=Ø:IF ds THEN t=f:GOSUB placepieces←
NEXT f:RETURN←
←
getchar:←
b$=UCASE$(INKEY$):IF b$<>"" THEN getchar←
getchar2:←
b$=UCASE$(INKEY$):IF b$="" THEN getchar2 ELSE RETURN
←
←
playerscore:←
IF sp=Ø THEN RETURN←
FOR h=pl+1 TO pl+sp←
COLOR 4:LOCATE 21,19:PRINT h←
SOUND 44Ø,2,255,Ø:FOR tm=1 TO 5ØØ:NEXT←
NEXT h:pl=pl+sp:RETURN←
←
computerscore:←
IF sc=Ø THEN RETURN←
FOR h=cp+1 TO cp+sc←
COLOR 4:LOCATE 9,19:PRINT h←
SOUND 22Ø,2,255,Ø:FOR tm=1 TO 5ØØ:NEXT←
NEXT h:cp=cp+sc:RETURN←
←
dash:←
COLOR 1:FOR j=1 TO 2:FOR i=1 TO 7:PRINT "     "CHR$(1
24);←
NEXT:PRINT:NEXT:RETURN←
←
cline:←
LOCATE 23,1:PRINT ds$:LOCATE 23,1:RETURN←
```

# CHAPTER TWO

# Classic Card Games

# Euchre

David Shimoda
*Translation by Tim Midkiff*

*Here's a finely detailed implementation of the classic card game of Euchre. The Amiga's stunning graphics powers are combined with its speed and precision to create a visually exciting and challenging game. The Amiga also provides a partner and two opponents in this keyboard-controlled, one-player version.*

"Euchre" is a four-handed translation of the popular card game of the same name. In this version, you play with a computer partner against two computer opponents. The computer will deal the cards, keep score, and play your partner's as well as your opponents' hands. Even better, it never gets bored or commits blunders such as trumping your ace. Nearly all the subtleties of the original card game are reproduced faithfully, including lone hands, short suits, and more. You can even choose different personalities for your partner and opponents. Type in the program and save a copy before you run it.

## Computer Personalities

The game begins by asking you to choose personalities for your partner and your opponents. Move the reverse-video cursor to your choices, and make selections by pressing the Return key.

The normal personality plays a more cautious game, while the aggressive personality tends to take more risks. Both opponents must have the same personality, but the partner's personality is chosen separately. This makes the game much more varied than if the computer players always stick to the same, predictable strategy. One of the more difficult combinations is to choose a normal partner and aggressive opponents. Of course, your own style of play will have an impact on which combination you prefer.

## Dealing and Trump

This Euchre variation uses only 24 cards from the standard 52-card deck. Each suit includes only the 9, 10, jack, queen, king, and ace (the ace is high). Before actual play begins, the first dealer must be selected. This is done by dealing out cards until a black jack is thrown. The first person who receives a black jack becomes the first dealer. Press the Return key for the dealing to begin. After each hand, the position of dealer passes to the next player in clockwise order.

The dealer distributes 5 cards to each player and then places 1 card, face up, on the center of the table. As a consequence of this scheme, only 21 of the 24 cards are in play for any given hand. (Three cards are always left unplayed.)

The next step is to choose *trump;* the trump suit is the most powerful of the four suits for the current hand. Trump is determined by moving around the table in clockwise order, giving each player an opportunity to choose whether the dealer should pick up the center card. Each player can either pass or *order up*—order the dealer to pick up the center card. When the dealer is forced to take the center card, that card's suit becomes trump, and the dealer discards one card. The computer players, of course, decide for themselves whether to pass or to order up in this phase of the game.

If no player chooses to order up in the first circuit of the table, each player then has a chance to pick any other suit as trump. If no player chooses trump on the second circuit, the hand is thrown out completely, and another is dealt.

## Lone Hands

On certain occasions, a player may choose to exclude his partner from play, a tactic which is known as playing *lonehand.* The player who chooses trump must choose at the same time whether or not to play lonehand. If a player orders up a card into his partner's hand, the player who ordered up must play lonehand. (If your partner is the dealer and you order up, you must play lonehand.)

For instance, you might want to play lonehand in a case where you hold most of the high cards in a suit, your partner is the dealer, and the center card is a high card of your strong suit. By excluding your partner and playing lonehand, you are in a very strong position to take most or all of the tricks.

## Tricks and Hands

A hand consists of five *tricks.* A trick consists of all players laying down one card. The player to the left of the dealer throws down the first card in the first trick. Subsequent tricks are begun by the winner of the previous trick. Suit must be followed within a trick. That is, you must throw a card of the suit which was led, as long as you have any card of that suit. To play a card, use the right and left cursor keys to move the special hand icon over the one you wish to throw. When it's in place, press the Return key to play the card.

If no trump cards are thrown in a trick, the trick is won by the player who laid the highest card of the leading suit. If trump is thrown, then the highest trump card takes the trick.

For all suits except the trump suit, the rank of the cards follows the usual order. (The 9 is low, and the ace is high.) For the trump suit, however, the jack is the highest-ranking card. The jack of the same color, but different suit, is considered part of the trump suit—and it is the second highest ranking card. For example, if the trump suit is chosen as clubs, it follows this ranking:

jack of clubs
jack of spades
ace of clubs
king of clubs
queen of clubs
10 of clubs
9 of clubs

A hand is won by the side which wins a majority of tricks (three or more). If you or your partner orders up a card, your side must take the majority of tricks in that hand or else be *euchred,* meaning that the opposite side gets two extra points.

## Scoring

A game of Euchre ends when one side accumulates ten or more points. You score one point for winning a hand, two points for winning all the tricks in a hand, and four points for winning all the tricks lonehand.

## Euchre

Filename: EUCHRE

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.←
'  All Rights Reserved←
←
Euchre:←
DEFINT a-z:DEFSNG r,g,b,cy:RANDOMIZE TIMER←
SCREEN 1,320,200,4,1:WINDOW 3,"",(Ø,Ø)-(311,186),16,
1:WINDOW OUTPUT 3:COLOR 3,Ø←
RESTORE PaLetteData:FOR i=Ø TO 15:READ r,g,b:PALETTE
 i,r,g,b:NEXT←
PaLetteData: ←
DATA .2,.2,.9,Ø,Ø,Ø,Ø,Ø,Ø,.8,Ø,Ø←
DATA .8,.8,.8,Ø,Ø,1,1,1,Ø,.93,.93,Ø←
DATA .87,.87,Ø,Ø,.8,Ø,Ø,1,Ø,1,1,Ø←
DATA .3,.3,.3,.5,.5,.5,.6,.6,.6,.7,.7,.7←
GOSUB InitiaLize:GOSUB DispLay:GOSUB PLayers:GOSUB P
ickDeaLer←
NewHand:←
GOSUB Bidding←
IF tp=4 THEN←
COLOR Ø,4:LOCATE 12,29:PRINT "NO BIDDERS"←
LOCATE 13,28:PRINT "HAND  DUMPED"←
GOSUB WaitKey:x=2Ø:FOR i=Ø TO 4:y=i*5+2:GOSUB EraseC
ard:NEXT ←
dL=FNnp(dL):x=8:y=12:GOSUB EraseCard←
COLOR ,Ø:GOSUB CLRMess:GOTO NewHand←
END IF:COLOR 5,4:LOCATE 12,29:PRINT "TRUMP : "←
LOCATE 13,29:PRINT "BIDDER: ";←
IF bd=Ø THEN ←
PRINT "you";←
ELSE←
PRINT "p";RIGHT$(STR$(bd),1)←
END IF:PUT(288,88),sb(Ø,tp),PSET←
IF ku<>Ø THEN ←
IF dL<=Ø THEN ←
COLOR 11,2:LOCATE 15,28:PRINT "PICK DISCARD"←
GOSUB UPickCard:COLOR ,4:LOCATE 15,28:PRINT "
      ";←
ELSE←
```

```
GOSUB UDiscard←
END IF:c(dL,f)=kc:s(dL,f)=ks:GOSUB PutHand←
END IF:x=8:y=12:GOSUB EraseCard←
GOSUB ResetHand:GOSUB PLayHand:COLOR Ø,4←
IF NOT((pw(Ø)>9) OR (pw(1)>9)) THEN ←
x=21:y=31:n=Ø:GOSUB PrintScore←
x=21:y=37:n=Ø:GOSUB PrintScore←
x=7:y=31:n=pw(Ø):GOSUB PrintScore←
x=7:y=37:n=pw(1):GOSUB PrintScore←
dL=FNnp(dL):GOSUB ExitLoopPD:GOTO NewHand←
END IF:wt=Ø:IF pw(1)>=1Ø THEN wt=1←
x=7:y=28+wt*6:n=1:GOSUB PrintScore←
x=7:y=31+wt*6:n=pw(wt)-1Ø:GOSUB PrintScore←
FOR de=1 TO 2ØØØ:NEXT←
GOSUB CLearWindow:COLOR 11,3:LOCATE 13,28:PRINT "  Y
OU";←
IF wt=Ø THEN PRINT " WIN!  "←
IF wt=1 THEN PRINT " LOSE  "←
COLOR 4,Ø:LOCATE 2,2:PRINT "Play again?"←
Lo=11:hi=12:xp=2:yp=14:GOSUB SeLection←
SCREEN CLOSE 1:WINDOW CLOSE 3←
IF an=11 THEN RUN ←
END←
←
DispLay: ←
GOSUB InitShapes:WIDTH 4Ø:CLS:COLOR ,Ø←
LINE(Ø,Ø)-(3,3),2,bf:GET(Ø,Ø)-(3,3),en←
LINE(Ø,Ø)-(3,3),11,bf:GET(Ø,Ø)-(3,3),nb←
GET(4,4)-(4Ø,44),ec:LINE(4,4)-(4Ø,44),4,bf←
GET(4,4)-(4Ø,44),cb:GET(4,4)-(17,21),eh:CLS←
FOR i=Ø TO 3:j=i*2:LINE(216+j,j)-(311-j,23-j),i+12,b
f:NEXT←
COLOR 2,4:FOR i=Ø TO 6 STEP 2:LINE(224,7+i)-(3Ø3,7+i
):NEXT←
LOCATE 2,31:PRINT "EUCHRE":LINE(224,15)-(3Ø3,15)←
COLOR 1Ø,2:LOCATE 4,28:PRINT "   POINTS   ←
LOCATE 18,28:PRINT "   TRICKS   "←
COLOR 9,2:LOCATE 5,28:PRINT " YOU   COMP "←
LOCATE 19,28:PRINT " YOU   COMP "←
FOR i=Ø TO 4:LOCATE 6+i,28:PRINT w$:LOCATE 2Ø+i,28:P
RINT w$;:NEXT ←
LINE(216,4Ø)-(311,4Ø),9:LINE(216,152)-(311,152),9←
LINE(262,32)-(262,79),9:LINE(262,144)-(262,191),9←
COLOR ,4:LOCATE 11,28:PRINT w$:LOCATE 17,28:PRINT w$
:GOSUB CLearWindow←
FOR i=Ø TO 3:LINE(216,8Ø+i)-(311,8Ø+i),i+12←
LINE(216,132+i)-(311,132+i),15-i:NEXT ←
n=Ø:x=7:y=31:GOSUB PrintScore:y=37:GOSUB PrintScore←
x=21:y=31:GOSUB PrintScore:y=37:GOSUB PrintScore←
COLOR 4,Ø:RETURN←
←
```

```
InitiaLize:<
DIM c(3,4),s(3,4),ms(6,3),cL(7,3),dc(23),ds(23),sp(3
,5),ns(3,5),me$(15)<
DIM en(19),nb(19),eh(75),hb(75),ec(507),cb(507),sb(3
0,3)<
DEF FNnp(x)=((x+1)/4-INT((x+1)/4))*4<
bL$=SPACE$(10):w$=SPACE$(12):c$="9 10J Q K A J J  91
0 J Q K A J J"<
FOR i=0 TO 3:READ co(i):NEXT:DATA 3,2,3,2<
FOR i=0 TO 6:READ nf(i):NEXT:DATA 4,0,1,2,3,4,0<
FOR j=0 TO 3:FOR i=0 TO 5:dc(j*6+i)=i:ds(j*6+i)=j:NE
XT i,j<
FOR i=0 TO 3:READ px(i),py(i):NEXT:DATA 14,12,8,7,2,
12,8,17<
FOR i=0 TO 3:READ cx(i),cy(i):NEXT:DATA 18,11,11,4,4
,11,11,18<
FOR i=0 TO 5:READ cp(i):NEXT:DATA 1,1,8,1,2,-1<
FOR i=0 TO 13:READ me$(i):NEXT<
DATA "pass     ","order up","pass     ","pick up","  PA
SS   ","diamonds"<
DATA "clubs    ","hearts  ","spades   ","normal     ","
aggressive","yes","no ","yes"<
FOR i=0 TO 3:READ mx(i),my(i):NEXT:DATA 1,1,8,3,2,11
,8,19<
FOR i=0 TO 6:READ ob(i),ou(i),PU(i),ms(i,0),ms(i,1),
ms(i,2),ms(i,3),ga(i):NEXT<
DATA 99,99,99,99,99,99,99,99<
DATA 99,99,99,99,99,99,99,99<
DATA 99,99,14,14,14,13,13,99<
DATA 20,12,8,8,8,8,7,19<
DATA 14,0,0,0,0,0,0,16<
DATA 0,0,0,0,0,0,0,14<
DATA 0,0,0,0,0,0,0,0<
FOR i=0 TO 9:READ pt&(i):NEXT<
DATA 16768479&,13421772&,16719823&,16764879&,1343433
3&<
DATA 16764703&,16768799&,13421791&,16768991&,1342255
9& <
RETURN<
<
PutCard: <
IF (s=tp) AND (c=6) THEN s=s+2:s=(s/4-INT(s/4))*4<
NC=c*2+1:COLOR co(s),4:px=(y-1)*8-2:py=(x-1)*8-1<
PUT(px,py),cb,PSET::px=px+1:py=py+2:PUT(px,py+8),sb(
0,s),PSET<
px=px+2:py=py-1:PUT(px+22,py+21),sb(0,s),PSET<
LOCATE x,y:PRINT MID$(c$,NC,2):LOCATE x+4,y+2:PRINT
MID$(c$,NC+16,2);<
RETURN <
<
PutHand: FOR u=0 TO 4:x=20:y=u*5+2<
```

```
c=c(Ø,u):s=s(Ø,u):GOSUB PutCard:NEXT:RETURN←
 ←
DeaLCards: FOR i=Ø TO 23:j=INT(RND(1)*24):t=dc(i):dc
(i)=dc(j):dc(j)=t←
t=ds(i):ds(i)=ds(j):ds(j)=t:NEXT:FOR j=Ø TO 3:FOR i=
Ø TO 4←
c(j,i)=dc(j*5+i):s(j,i)=ds(j*5+i):NEXT i,j:kc=dc(2Ø)
:ks=ds(2Ø):RETURN←
←
PickDeaLer: ←
COLOR Ø,4:LOCATE 13,28:PRINT "FIRST  BLACK":LOCATE 1
4,29:PRINT "JACK DEALS"←
GOSUB DeaLCards:dL=Ø:cc=Ø←
LoopPD: c=dc(cc):s=ds(cc):x=cx(dL):y=cy(dL):GOSUB Pu
tCard←
FOR de=1 TO 5ØØ:NEXT ←
IF (dc(cc)=2) AND ((ds(cc) AND 253)=1) THEN ExitLoop
PD←
x=cx(dL):y=cy(dL):GOSUB EraseCard←
FOR de=1 TO 1ØØ:NEXT ←
cc=cc+1:dL=FNnp(dL):GOTO LoopPD←
ExitLoopPD: GOSUB CLearWindow:COLOR Ø,4:LOCATE 12,29
←
IF dL<>Ø THEN ←
PRINT " PLAYER";STR$(dL)←
LOCATE 13,3Ø:PRINT " DEALS";←
ELSE←
PRINT "YOUR  DEAL";←
END IF:GOSUB WaitKey←
x=cx(dL):y=cy(dL):GOSUB EraseCard:COLOR 7,1:RETURN←
←
WaitKey: COLOR 3,2:LOCATE 15,28:PRINT " HIT RETURN "
:SOUND 2ØØØ,6←
WHILE INKEY$<>CHR$(13):WEND:GOSUB CLearWindow:COLOR
7,1:RETURN←
←
CLearWindow: COLOR ,4:FOR i=12 TO 16:LOCATE i,28:PRI
NT w$:NEXT:RETURN←
←
PrintScore:←
tx=(y-1)*8:py=(x-1)*8:bt&=1:i&=pt&(n)←
FOR q=Ø TO 5:FOR p=Ø TO 3:px=tx+p*4←
IF (bt& AND i&)<>Ø THEN PUT(px,py),nb,PSET ELSE PUT(
px,py),en,PSET←
px=tx+p*4:bt&=bt&*2:NEXT p:py=py+4:NEXT:RETURN←
←
SetPoints: ←
FOR i=Ø TO 3:sp(p,i)=fc(p AND 253):ns(p,i)=Ø:NEXT ←
FOR i=Ø TO 4:s=s(p,i):c=c(p,i):sp(p,s)=sp(p,s)+cp(c)
:ns(p,s)=ns(p,s)+1←
```

```
IF c=2 THEN s=s+2:s=(s/4-INT(s/4))*4:sp(p,s)=sp(p,s)
+6:ns(p,s)=ns(p,s)+1←
IF c=5 THEN ←
FOR j=Ø TO 3:sp(p,j)=sp(p,j)+4:NEXT ←
END IF:NEXT:ss=Ø:FOR i=Ø TO 4:IF ns(p,i)=Ø THEN sp(p
,i)=Ø:ss=ss+1←
NEXT:FOR i=Ø TO 3:sp(p,i)=sp(p,i)+ss:NEXT ←
IF p=dL THEN ←
IF kc=5 THEN sp(p,ks)=sp(p,ks)+4←
sp(p,ks)=sp(p,ks)+cp(kc):ns(p,ks)=ns(p,ks)+1←
END IF:RETURN ←
←
UOrderUp: Lo=Ø:hi=1:xp=14:yp=1Ø:GOSUB SeLection←
IF an=1 THEN tp=ks←
RETURN ←
←
ULoneHand: LOCATE 14,9:PRINT "Lonehand";←
Lo=12:hi=13:xp=14:yp=18:GOSUB SeLection:Lh=Ø←
IF an=13 THEN Lh=1:LOCATE mx(bd),my(bd):PRINT "Loneh
and";←
LOCATE 14,9:PRINT "        ";←
RETURN ←
←
CGoALone: Lh=Ø:IF sp(p,tp)>ga(ns(p,tp)) THEN Lh=1←
RETURN←
 ←
COrderUp: IF FNnp(FNnp(p))=dL THEN GOSUB CGoALone:f=
Lh:GOTO ExitCOU←
f=Ø:IF kc<>2 THEN ←
IF sp(p,ks)>ou(ns(p,ks)) THEN f=1←
END IF:IF sp(p,ks)>ob(ns(p,ks)) THEN f=1←
IF (f=Ø) OR (p<>FNnp(dL)) THEN ExitCOU←
sb=cp(kc):IF kc=5 THEN sb=3←
FOR i=Ø TO 3:IF i<>ks THEN IF sp(p,i)>=(sp(p,ks)-sb)
 THEN f=Ø←
NEXT ←
ExitCOU: IF f=1 THEN tp=ks←
RETURN ←
←
UPickUp: Lo=2:hi=3:xp=14:yp=1Ø:GOSUB SeLection←
IF an=3 THEN tp=ks←
RETURN ←
←
CPickUp: IF sp(p,ks)>PU(ns(p,ks)) THEN tp=ks←
RETURN ←
←
UMake: Lo=4:hi=8:xp=14:yp=1Ø:GOSUB SeLection←
IF an-5=ks THEN UMake←
IF an>4 THEN tp=an-5←
RETURN ←
←
CMake: df=Ø:FOR i=Ø TO 3←
```

```
IF i<>ks THEN←
IF sp(p,i)-ms(ns(p,i),ps)>=df THEN df=sp(p,i)-ms(ns(
p,i),ps):tp=i←
END IF:NEXT←
RETURN←
←
CLRMess: FOR i=1 TO 3:FOR j=Ø TO 2:LOCATE mx(i)+j,my
(i):PRINT SPACE$(8);←
NEXT j,i:RETURN←
←
SeLection: ←
an=Lo:k$="":WHILE k$<>CHR$(13)←
x1=xp:y1=yp:FOR i=Lo TO hi:COLOR 4,Ø←
IF i=an THEN ←
IF (an=Lo) OR (hi-Lo=1) THEN COLOR Ø,4 ELSE COLOR ,c
o(i-Lo-1)←
END IF:LOCATE x1,y1:PRINT me$(i);:x1=x1+1:NEXT←
WaitS: k$=INKEY$:IF k$="" THEN WaitS ←
IF k$=CHR$(28) THEN ←
an=an-1:IF an<Lo THEN an=hi←
ELSEIF k$=CHR$(29) THEN ←
an=an+1:IF an>hi THEN an=Lo←
END IF:WEND:x1=xp:y1=yp:COLOR 4,Ø←
FOR i=Lo TO hi:LOCATE x1,y1:PRINT bL$;:x1=x1+1:NEXT←
RETURN←
←
Bidding: ←
GOSUB DeaLCards:GOSUB PutHand:p=FNnp(dL):tp=4:bd=Ø:k
u=Ø←
x=8:y=12:c=kc:s=ks:GOSUB PutCard:COLOR 4,Ø←
IF dL<>Ø THEN LOCATE mx(dL),my(dL):PRINT "dealer"←
5 GOSUB SetPoints←
IF p=Ø THEN GOSUB UOrderUp:GOTO 2Ø←
IF ABS(p-dL)<>2 THEN 7 ←
GOSUB CGoALone:IF Lh=1 THEN tp=ks:GOTO 1Ø←
7 GOSUB COrderUp←
1Ø LOCATE mx(p),my(p):COLOR 4,Ø←
IF tp=4 THEN PRINT " pass":GOTO 2Ø←
bd=p:PRINT "order up"←
2Ø p=FNnp(p):IF (p<>dL) AND (tp=4) THEN 5←
p=dL:GOSUB SetPoints:IF tp<>4 THEN 45←
IF dL=Ø THEN GOSUB UPickUp:GOTO 3Ø←
GOSUB CPickUp:LOCATE mx(dL),my(dL)←
IF tp=4 THEN PRINT "turned":LOCATE mx(dL)+1,my(dL):P
RINT " down":GOTO 3Ø←
bd=p:PRINT "picked":LOCATE mx(dL)+1,my(dL):PRINT "
up"←
3Ø FOR de=1 TO 2ØØØ:NEXT←
IF (bd=Ø) AND (tp<>4) THEN 45 ←
x=8:y=12:GOSUB EraseCard:IF tp<>4 THEN 45←
GOSUB CLRMess:ps=Ø←
```

```
35 p=FNnp(p)<
IF p=Ø THEN GOSUB UMake:GOTO 4Ø<
GOSUB CMake:LOCATE mx(p),my(p)<
FOR de=1 TO 6ØØ:NEXT<
IF tp=4 THEN PRINT "  pass";:GOTO 4Ø<
bd=p:PRINT me$(tp+5);<
4Ø IF (p<>dL) AND (tp=4) THEN ps=ps+1:GOTO 35<
GOTO 5Ø<
45 ku=1:IF (bd=Ø) AND (dL=2) THEN Lh=1:GOTO 6Ø<
5Ø IF tp=4 THEN 7Ø<
IF (Lh=1) AND (bd<>Ø) THEN 6Ø<
IF bd=Ø THEN GOSUB ULoneHand:GOTO 7Ø<
GOSUB CGoALone<
IF Lh=Ø THEN 7Ø<
6Ø LOCATE 1,1:PRINT "Lonehand"<
7Ø FOR de=1 TO 2ØØØ:NEXT<
GOSUB CLRMess:RETURN<
<
UPickCard: <
f=Ø:WHILE c(Ø,f)=-1:f=f+1:WEND:g=f<
PrintHand: x=(g*5+2)*8+1:PUT(x,167),eh,PSET:x=(f*5+2
)*8+1:PUT(x,167),hb,PSET<
GetKeyUPC: k$=INKEY$:IF k$="" THEN GetKeyUPC ELSE IF
 k$=CHR$(13) THEN ExitUPC<
g=f:IF k$<>CHR$(31) THEN 1ØØ<
9Ø f=nf(f):IF c(Ø,f)<Ø THEN 9Ø<
GOTO PrintHand<
1ØØ IF k$<>CHR$(3Ø) THEN GetKeyUPC<
11Ø f=nf(f+2):IF c(Ø,f)<Ø THEN 11Ø<
GOTO PrintHand<
GOTO GetKeyUPC<
ExitUPC: RETURN<
<
UDiscard:<
FOR i=Ø TO 4<
IF (s(p,i)=tp) AND (c(p,i)=2) THEN <
c(p,i)=7<
ELSE <
IF ((s(p,i) AND 253)=(tp AND 253)) AND (c(p,i)=2)THE
N c(p,i)=6:s(p,i)=tp<
END IF:NEXT <
FOR i=Ø TO 4:FOR j=Ø TO 3<
IF NOT(s(p,j)>s(p,j+1)) THEN<
IF s(p,j)=s(p,j+1) THEN IF NOT(c(p,j)>c(p,j+1)) THEN
<
t=c(p,j):c(p,j)=c(p,j+1):c(p,j+1)=t<
t=s(p,j):s(p,j)=s(p,j+1):s(p,j+1)=t<
END IF<
END IF:NEXT j,i<
FOR i=Ø TO 4:pt(i)=Ø<
IF s(p,i)=tp THEN <
```

```
pt(i)=c(p,i)+10←
ELSE←
IF c(p,i)=5 THEN←
pt(i)=9←
ELSE←
IF (s(p,i)<>s(p,nf(i))) AND (s(p,i)<>s(p,nf(i+2))) T
HEN pt(i)=-1←
END IF←
END IF:NEXT ←
L=99:FOR i=0 TO 4:IF pt(i)<L THEN f=i:L=pt(i)←
NEXT:RETURN ←
←
ResetHand:←
FOR i=0 TO 3:FOR j=0 TO 3:ns(i,j)=0:NEXT:FOR j=0 TO
4←
IF c(i,j)=2 THEN ←
IF s(i,j)=tp THEN←
c(i,j)=7←
ELSE←
IF ABS(s(i,j)-tp)=2 THEN c(i,j)=6:s(i,j)=tp←
END IF←
END IF:ns(i,s(i,j))=ns(i,s(i,j))+1:NEXT j,i←
RETURN ←
←
PLayers:←
LOCATE 2,2:PRINT "Partner?";:Lo=9:hi=10:xp=2:yp=12:G
OSUB SeLection←
fc(0)=0:IF an=10 THEN fc(0)=2←
LOCATE 2,2:PRINT "Opponents?";:Lo=9:hi=10:xp=2:yp=14
:GOSUB SeLection←
fc(1)=0:IF an=10 THEN fc(1)=2←
LOCATE 2,2:PRINT "           ";:RETURN←
←
PLayHand: ←
FOR i=0 TO 7:FOR j=0 TO 3:cL(i,j)=0:NEXT j,i:cL(2,tp
 AND 253)=1←
FOR i=0 TO 3:sL(i)=0:NEXT ←
Ld=FNnp(dL):dm=4:tr(0)=0:tr(1)=0:IF Lh<>0 THEN←
IF bd=2 THEN x=20:FOR i=0 TO 4:y=i*5+2:GOSUB EraseCa
rd:NEXT ←
dm=FNnp(FNnp(bd))←
IF Lh=1 THEN IF Ld=dm THEN Ld=FNnp(Ld)←
END IF:FOR tk=0 TO 4:p=Ld:ps=0:tL=0:IF dm=p THEN p=F
Nnp(p)←
GOSUB PLayCard:wp=p:IF Lh=1 THEN ps=ps+1←
sL(s(p,pc(p)))=1←
IF s(p,pc(p))=tp THEN tL=1←
FOR i=1 TO 3:p=FNnp(p):IF p=dm THEN 130←
ps=ps+1:GOSUB PLayCard:IF tL=0 THEN 120←
```

```
IF s(p,pc(p))=tp THEN IF c(p,pc(p))>c(wp,pc(wp)) THE
N wp=p←
GOTO 130←
120 IF s(p,pc(p))=tp THEN wp=p:tL=1:GOTO 130←
IF s(p,pc(p))=s(wp,pc(wp)) THEN IF c(p,pc(p))>c(wp,p
c(wp)) THEN wp=p←
130 NEXT:FOR de=1 TO 400:NEXT←
x=px(wp):y=py(wp):GOSUB Winner←
FOR de=1 TO 3000:NEXT:Ld=wp:wt=wp AND 253:tr(wt)=tr(
wt)+1←
COLOR 2,3:x=21:y=31+6*wt:n=tr(wt):GOSUB PrintScore←
FOR i=0 TO 3:x=px(i):y=py(i):GOSUB EraseCard:c(i,pc(
i))=-1:NEXT i,tk←
COLOR ,0:LOCATE 1,1:PRINT "          ";←
bt=bd AND 253:LOCATE 15,28←
COLOR 2,3:IF NOT(tr(bt)<3) THEN←
IF NOT(tr(bt)<5) THEN ←
pw(bt)=pw(bt)+2+Lh*2←
IF bt=0 THEN PRINT "  YOU  WON  ":LOCATE 16,28:PRINT
 " ALL TRICKS ":GOTO 140←
IF bt=1 THEN PRINT "COMPUTER WON":LOCATE 16,28:PRINT
 " ALL TRICKS ":GOTO 140←
END IF:pw(bt)=pw(bt)+1←
IF tr(0)>2 THEN PRINT "YOU WON HAND";:GOTO 140←
IF tr(1)>2 THEN PRINT "  COMPUTER  ":LOCATE 16,28:PR
INT "  WON HAND  ":GOTO 140←
END IF:pw(1-bt)=pw(1-bt)+2←
IF tr(0)<3 THEN PRINT "YOU'VE  BEEN":LOCATE 16,28:PR
INT "  EUCHRED!  ":GOTO 140←
IF tr(1)<3 THEN PRINT "  COMPUTER  ":LOCATE 16,28:PR
INT "  EUCHRED!  "←
140 FOR de=1 TO 4000:NEXT←
RETURN←
 ←
PLayCard: ←
IF p<=0 THEN ←
COLOR 11,2:LOCATE 15,28:PRINT " YOUR  PLAY ";:GOSUB
UPickCard←
150 Ls=s(Ld,pc(Ld))←
IF NOT((ps=0) OR (s(p,f)=Ls) OR (ns(p,Ls)=0)) THEN←
GOSUB PrintHand:GOTO 150←
x=20:y=f*5+2:GOSUB EraseCard:GOTO 160←
END IF:COLOR ,4:LOCATE 15,28:PRINT "            ";←
x=20:y=f*5+2:GOSUB EraseCard:GOTO 160←
END IF:IF tk>=5 THEN ←
FOR k=0 TO 4:IF c(p,j)>-1 THEN f=i←
NEXT:GOTO 160←
END IF:ON (ps+1) GOSUB 4000,4100,4200,4200←
160 pc(p)=f:y=py(p):x=px(p):c=c(p,f):s=s(p,f):GOSUB
PutCard←
ns(p,s(p,f))=ns(p,s(p,f))-1:cL(c(p,f),s(p,f))=1←
```

```
RETURN←
 ←
4000 IF ns(p,tp)<>5-tk THEN 4015←
sp=tp:GOSUB 5200:IF f=1 THEN GOTO 5150←
GOTO 5160←
4015 IF (Lh<>1) OR (bd<>p) THEN 4030←
IF ns(p,tp)>0 THEN sp=tp:GOTO 5150←
GOTO 5050←
4030 GOSUB 5000:IF (f=1) AND (ABS(bd-p)=2) THEN sp=t
p:GOTO 5150←
GOSUB 5250:IF (f<>1) OR (p<>bd) THEN 5050←
GOSUB 5200:IF i=1 THEN sp=tp:GOTO 5150←
IF ns(p,tp)>2 THEN sp=tp:GOTO 5160←
GOTO 5050←
4100 IF ns(p,s(Ld,pc(Ld)))=0 THEN 4115←
GOSUB 5300:sp=s(Ld,pc(Ld)):IF f=1 THEN 5150←
GOTO 5160←
4115 IF ns(p,tp)=5-tk THEN sp=tp:GOTO 5160←
IF ns(p,tp)=0 THEN 5100←
IF c(Ld,pc(Ld))=5 THEN sp=tp:GOTO 5160←
IF bd<>p THEN sp=tp:GOTO 5160←
GOSUB 5250:IF f=1 THEN sp=tp:GOTO 5160←
GOTO 5100←
4200 IF ns(p,s(Ld,pc(Ld)))=0 THEN 4235←
sp=s(Ld,pc(Ld))←
IF (sp<>tp) AND (tL=1) THEN 5160←
IF ABS(wp-p)<>2 THEN 4225←
GOSUB 5300:IF f=1 THEN GOSUB 5350:IF f=0 THEN 5150←
GOTO 5160←
4225 GOSUB 5300:IF f=1 THEN 5150←
GOTO 5160←
4235 IF ns(p,tp)<5-tk THEN 4270←
sp=tp:IF ABS(wp-p)=2 THEN 5160←
IF tL=0 THEN 5160←
GOSUB 5300:IF f=1 THEN 5400←
GOTO 5160←
4270 IF ns(p,tp)=0 THEN 5100←
IF ABS(wp-p)<>2 THEN 4310←
IF (tL=1) OR (ps=3) THEN 5100←
IF c(wp,pc(wp))=5 THEN 5100←
IF c(wp,pc(wp))<4 THEN sp=tp:GOTO 5160←
GOSUB 5250:IF f=1 THEN sp=tp:GOTO 5160←
GOTO 5100←
4310 IF tL=0 THEN sp=tp:GOTO 5160←
GOSUB 5300:IF f=1 THEN sp=tp:GOTO 5400←
GOTO 5100←
5000 f=0:FOR a=0 TO 4:IF c(p,a)>5 THEN f=1←
NEXT:RETURN ←
5050 f=-1:FOR a=0 TO 4:IF (sL(s(p,a))=0) AND (s(p,a)
<>tp) THEN IF c(p,a)=5 THEN f=a←
NEXT:IF f>-1 THEN 5070←
```

```
Lc=-1:FOR a=Ø TO 4:IF s(p,a)<>tp THEN IF c(p,a)>Lc T
HEN Lc=c(p,a):f=a←
NEXT←
5Ø7Ø RETURN ←
51ØØ IF ns(p,tp)>Ø THEN 5125←
sp=-1:FOR a=Ø TO 4←
IF s(p,a)<>tp THEN IF (c(p,a)=5) AND (ns(p,s(p,a))>1
) THEN sp=s(p,a)←
NEXT:IF sp>-1 THEN 516Ø←
GOTO 518Ø←
5125 v=4:f=-1:FOR a=Ø TO 4←
IF s(p,a)=tp THEN 5135←
IF (ns(p,s(p,a))<>1) OR (sL(s(p,a))=1) THEN 5135←
IF (c(p,a)>=Ø) AND (c(p,a)<v) THEN v=c(p,a):f=a←
5135 NEXT:IF f=-1 THEN 518Ø←
RETURN ←
515Ø IF ps=3 THEN 54ØØ←
v=-1:FOR a=Ø TO 4:IF s(p,a)=sp THEN IF c(p,a)>v THEN
 v=c(p,a):f=a←
NEXT:RETURN ←
516Ø v=1Ø:FOR a=Ø TO 4←
IF s(p,a)=sp THEN IF (c(p,a)>=Ø) AND (c(p,a)<v) THEN
 v=c(p,a):f=a←
NEXT:RETURN ←
518Ø v=1Ø:FOR a=Ø TO 4:IF s(p,a)<>tp THEN IF c(p,a)>
-1 THEN IF c(p,a)<v THEN v=c(p,a):f=a←
NEXT:RETURN ←
52ØØ ht=8:f=Ø←
52Ø5 ht=ht-1:IF ht>Ø THEN IF cL(ht,tp)=1 THEN 52Ø5←
IF ht<Ø THEN 524Ø←
FOR a=Ø TO 4:IF s(p,a)=tp THEN IF c(p,a)=ht THEN f=1
←
NEXT ←
524Ø RETURN ←
525Ø f=1:FOR a=Ø TO 4:IF c(p,a)>-1 THEN IF (s(p,a)<>
tp) AND (c(p,a)<5) THEN f=Ø←
NEXT:RETURN ←
53ØØ f=Ø:FOR a=Ø TO 4:IF s(p,a)=s(wp,pc(wp)) THEN IF
 c(p,a)>c(wp,pc(wp)) THEN f=1←
NEXT:RETURN ←
535Ø f=Ø:FOR a=Ø TO 4:IF s(p,a)=s(wp,pc(wp)) THEN IF
 c(p,a)-c(wp,pc(wp))=1 THEN f=1←
NEXT:RETURN ←
54ØØ d=1Ø:FOR a=Ø TO 4←
IF s(p,a)=s(wp,pc(wp)) THEN e=c(p,a)-c(wp,pc(wp)):IF
 (e<d) AND (e>Ø) THEN d=e:f=a←
NEXT:RETURN ←
←
EraseCard: PUT((y-1)*8-2,(x-1)*8-1),ec,PSET:RETURN←
←
Winner:←
```

```
x1=y-1:y1=x-1:x=(x1+2)*8:y=(y1+2)*8+3:x1=x1*8-3:y1=y
1*8-2←
CIRCLE (x,y),8,Ø:PAINT (x,y),Ø←
FOR i=1 TO 1ØØ:NEXT←
FOR i=1 TO 3:LINE (x1-i,y1-i)-(x1+i+38,y1+i+42),i+5,
b:NEXT←
FOR i=3 TO 1 STEP -1:CIRCLE (x,y),i*2,i+5:PAINT (x,y
),i+5:NEXT ←
r=1:FOR i=Ø TO 5:r=r-.Ø7:cy(i)=r:NEXT←
FOR i=1 TO 5Ø:FOR p=1 TO 2ØØ:NEXT:j=i MOD 6←
PALETTE (i MOD 3)+6,cy(j),cy(j),Ø:NEXT←
FOR i=3 TO 1 STEP -1:LINE (x1-i,y1-i)-(x1+i+38,y1+i+
42),Ø,b:FOR j=1 TO 5Ø:NEXT j,i ←
CIRCLE (x,y),8,4:PAINT (x,y),4←
RETURN←
←
InitShapes:←
RESTORE InitShapes←
FOR j=Ø TO 3:FOR i=Ø TO 3Ø:←
READ a$:sb(i,j)=VAL("&H"+a$):NEXT i,j←
RESTORE Hand:FOR i=Ø TO 75:←
READ a$:hb(i)=VAL("&H"+a$):NEXT:RETURN←
←
Diamond: DATA B,9,3,4ØØ,EØØ,1FØØ,3F8Ø,7FCØ←
DATA 3F8Ø,1FØØ,EØØ,4ØØ,4ØØ,EØØ,1FØØ,3F8Ø←
DATA 7FCØ,3F8Ø,1FØØ,EØØ,4ØØ,FBEØ,F1EØ,EØEØ←
DATA CØ6Ø,8Ø2Ø,CØ6Ø,EØEØ,F1EØ,FBEØ,Ø←
←
CLub: DATA B,9,3,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,EØØ,1FØØ,1FØØ,7FCØ←
DATA FFEØ,FFEØ,75CØ,EØØ,1FØØ,F1EØ,EØEØ,EØEØ←
DATA 8Ø2Ø,Ø,Ø,8A2Ø,F1EØ,EØEØ,Ø←
←
Heart: DATA B,9,3,71CØ,FBEØ,FFEØ,FFEØ,7FCØ←
DATA 3F8Ø,1FØØ,EØØ,4ØØ,71CØ,FBEØ,FFEØ,FFEØ←
DATA 7FCØ,3F8Ø,1FØØ,EØØ,4ØØ,8E2Ø,4ØØ,Ø←
DATA Ø,8Ø2Ø,CØ6Ø,EØEØ,F1EØ,FBEØ,Ø←
←
Spade: DATA B,9,3,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,4ØØ,4ØØ,EØØ,1FØØ←
DATA 3F8Ø,7FCØ,75CØ,EØØ,1FØØ,FBEØ,FBEØ,F1EØ←
DATA EØEØ,CØ6Ø,8Ø2Ø,8A2Ø,F1EØ,EØEØ,Ø←
←
Hand: ←
DATA E,12,4,Ø,2CØ,96Ø,15AØ,588←
DATA A3C,23C,2ØC,413C,2Ø54,1168,58,12Ø←
DATA 4Ø,14Ø,2AØ,54Ø,AAØ,6ØØ,34Ø,196Ø←
DATA CAØ,CA8,46AC,66A4,66A4,37EC,1FFC,FF8←
DATA FF8,FFØ,FFØ,17EØ,7EØ,F6Ø,AAØ,FFFC←
DATA FFFC,FFFC,FFFC,FFFC,FFFC,FFFC,FFFC,FFFC←
DATA FFFC,FFFC,FFFC,FFFC,FFFC,FFFC,FFFC,FFFC←
DATA FFFC,6ØØ,FCØ,3FEØ,3FEØ,1FF8,DFFC,EFFC←
DATA EFFC,FFFC,7FFC,3FF8,1FF8,1FFØ,1FFØ,1FEØ←
DATA FEØ,F6Ø,AAØ,Ø←
```

# Casino Blackjack

John Hamilton
*Translation by George Miller*

*Win your ticket home or lose your shirt in this accurate simulation of the most popular Las Vegas card game. "Casino Blackjack" is written in Amiga Basic and requires 512K of memory. Type in the listing and save a copy to disk before running the program.*

"Casino Blackjack" is *real* blackjack, played the same way it's played in the best Las Vegas casinos. You start with a bankroll of $100 and play against an intelligent computer dealer. Just like in the casinos, you can "split" your hand, double down, or buy insurance.

## In the Cards

The rules for Casino Blackjack are simple to learn. The object of the game is to get more points than the dealer—without exceeding 21 points. An ace can be counted as either 1 point or 11 points. Jacks, queens, and kings are all worth 10 points. All other cards count as their face value.

Before each hand, you decide how much of your bankroll you want to bet. Next, four cards are dealt—two to you and two to the computer dealer. The dealer's bottom card is face down and the top card is face up, so you can only guess at the total number of points the dealer has. Both your cards are dealt face up.

If you've been dealt a *blackjack* (21 points), you automatically win one and a half times your bet, unless the dealer was also dealt a blackjack, in which case no money exchanges hands.

After you've taken a look at your cards, decide whether you want to take a *hit* (take another card) or *stand* with the cards you have. If you take a hit and exceed 21 points, you have *gone bust* (lost the hand). If you take a card and still have less than 21 points, you're free to take another hit if you like. This process continues until you stand or bust.

If you stand, the dealer may get a card. The dealer takes a hit if the hand has 16 or fewer points and stands on 17 or more points. If the dealer doesn't go bust, the cards are inspected to see who came closest to 21. A tie is known as a *push.* In the case of a push, no money is lost; otherwise, the winner takes the money.

## Special Plays

If the dealer's top card is an ace, you may elect to insure yourself against the dealer having blackjack. You pay half your original bet for insurance. If the dealer does have blackjack, you keep your bet. You lose the insurance money, whether the dealer has blackjack or not.

If you're dealt a total of 10 or 11 points, you may double your bet. This is known as *doubling down.* If you choose to double down, you're dealt one more card, and then you automatically stand. If you think the dealer will win the hand, you may choose to surrender instead. Surrendering costs half your bet.

If you're dealt two cards of the same denomination, you may elect to *split* your hand. When you split, you double your original bet and play the two cards as two separate hands, one after the other. Each hand has a chance to beat the dealer.

### Casino Blackjack

Filename: CASINO BLACKJACK

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
Blackjack:←
'Copyright 1988 COMPUTE! Publications, Inc.←
'All Rights Reserved←
DEFINT a-z:DEFSNG r,g,b,cy:RANDOMIZE TIMER←
DIM cd$(52),v$(13),m(52),sb(36,3),card(507,4)←
sp$=SPACE$(40)←
SCREEN 1,320,200,4,1:WINDOW 3,"",(0,0)-(311,186),16,
1:WINDOW OUTPUT 3:COLOR 3,0←
Start:s=1:GOSUB DoScreen:GOSUB InitiaLize:GOSUB Disp
Lay←
Main:IF s<>1 THEN GOSUB HoLdscrn←
COLOR ,10:CLS:GOSUB TitLe:h=0:sp=0:s=0←
IF ba<>0 THEN ←
GOTO DoIt←
ELSE←
```

```
LOCATE 10,10:COLOR 4,10:PRINT "Sorry, you are broke!
":LOCATE 12,12←
PRINT "Play again? (Y/N)"←
Answer:GOSUB GetKey←
IF a$="Y" THEN GOTO Start ELSE IF a$="N" THEN GOTO E
ndGame←
GOTO Answer←
END IF   ←
DoIt:COLOR 7,10:LOCATE 6,12:←
ba=INT(ba*100)/100←
PRINT USING " Bank = $$########.##";ba ←
LOCATE 8,12:PRINT "Playing Hand #"+STR$(ha+1)←
LOCATE 10,15:COLOR 11,15:PRINT " Q ":LOCATE 12,15:PR
INT " R "←
COLOR 11,10:LOCATE 10,19:PRINT "to Quit":LOCATE 12,1
9:PRINT "to Restart"←
GetBet:LOCATE 16,1:PRINT sp$:COLOR 1,10:LOCATE 16,5←
INPUT "Enter your bet";bet$:bet=INT(VAL(bet$)*100)/1
00←
bet$=UCASE$(bet$)←
IF LEFT$(bet$,1)="Q" THEN GOTO EndGame←
IF LEFT$(bet$,1)="R" THEN CLS:GOTO Start←
IF bet<=0 OR bet>ba THEN GOTO GetBet←
GOSUB GameScreen:IF h=1 GOTO Main←
ChecKey:GOSUB GetKey←
IF a$="P" THEN GOSUB SpLit←
IF a$="D" THEN GOSUB DoubLe←
IF a$="H" THEN GOSUB Hit←
IF a$="S" THEN GOSUB Stand:GOTO Main←
IF a$="E" THEN ba=ba-bet*.5:GOTO Main←
IF h=1 THEN h=0:GOTO Main←
GOTO ChecKey←
DeLay:FOR i=1 TO 5000:NEXT:RETURN←
Stand:GOSUB ShowDeaLer:GOSUB VdeaLer←
IF sp<>1 THEN COLOR 4,10:LOCATE py+2,px:PRINT "Stand
":GOSUB PvaLue:GOSUB VdeaLer←
IF dt1=21 AND d=2 THEN LOCATE dy+2,dx:PRINT "Blackja
ck!":h=1:ba=ba-bet:RETURN←
DeaLerpLays:←
IF dt1>21 THEN←
LOCATE dy+2,dx:COLOR 4,10:PRINT "Busted!":ba=ba+bet:
GOSUB Pwin:h=1←
RETURN←
END IF←
IF dt1<17 OR dt2<17 THEN GOSUB GetCard:GOSUB VdeaLer
:GOTO DeaLerpLays←
GOSUB CheckCards:RETURN←
GetCard:FOR t= 1 TO 3000:NEXT←
c=c+1:dl$(d)=cd$(m(c)):tp$=dl$(d):d=d+1:c=c+1:x=dx*8
-1:y=dy*8-11←
xL=dx:yL=dy:dx=dx+6:GOSUB ShowCard:RETURN←
```

```
CheckCards:←
IF sp=1 THEN GOSUB SpLitHand←
IF dt1<=21 THEN dt=dt1 ELSE dt=dt2←
IF dt2>dt1 AND dt2<=21 THEN dt=dt2←
IF pt1<=21 THEN pt=pt1 ELSE pt=pt2←
IF pt2>pt1 AND pt2<=21 THEN pt=pt2←
COLOR 4,10:IF dt=pt THEN LOCATE 12,10:PRINT "Push, N
o winner":GOSUB DeLay←
IF dt>pt AND dt<=21 THEN GOSUB HouseWin:ba=ba-bet←
IF pt>dt AND pt<=21 THEN GOSUB Pwin:ba=ba+bet←
RETURN←
SpLitHand:←
IF vt1<=21 AND pt1<=21 AND vt1>pt1 THEN pt1=vt1←
IF vt2<=21 AND pt2<=21 AND vt2>pt2 THEN pt2=vt2←
RETURN←
Hit:c=c+1:pl$(p)=cd$(m(c)):tp$=pl$(p):p=p+1:x=px*8-1
:y=py*8-4:xL=px←
yL=py+1:px=px+6←
GOSUB ShowCard:GOSUB PvaLue←
IF pt1>21 AND pt2>21 THEN ←
LOCATE py+2,px:COLOR 4,10:PRINT "Busted!"←
IF sp=0 THEN←
GOSUB ShowDeaLer:GOSUB HouseWin:ba=ba-bet:GOSUB DeLa
y:h=1←
ELSE←
sp=0←
END IF←
END IF←
RETURN←
DoubLe:←
GOSUB PvaLue:IF pt1<10 OR pt1>11 THEN a$="":RETURN←
LOCATE 10,12:COLOR 4,10:PRINT "Double Down!":bet=bet
*2:GOSUB Hit:a$="S":RETURN←
ShowDeaLer:x=1*8-1:y=5*8-11:xL=1:yL=5:tp$=dl$(0):GOS
UB ShowCard:RETURN←
HouseWin:LOCATE dy+2,dx:COLOR 4,10:PRINT "House Wins
!":RETURN←
Pwin:LOCATE py+2,px:COLOR 4,10:PRINT "Winner!":RETUR
N
                                              ←
PvaLue:pt2=0:pt1=0:FOR x=0 TO p-1:g$=pl$(x):GOSUB Va
Lue:pt1=pt1+v1←
pt2=pt2+v2:NEXT:RETURN←
VdeaLer:dt2=0:dt1=0:FOR x=0 TO d-1:g$=dl$(x):GOSUB V
aLue:dt1=dt1+v1←
dt2=dt2+v2:NEXT←
IF dt1<21 THEN dt2=dt1←
IF dt1>21 THEN dt1=dt2←
RETURN←
VaLue:v1=0:v2=0:FOR y=1 TO 13←
IF LEFT$(g$,1)=v$(y) THEN v=y+1←
```

```
NEXT←
IF v=14 THEN v1=11:v2=1:RETURN←
IF v>10 THEN v1=10:v2=10:RETURN←
v1=v:v2=v←
RETURN←
SpLit:←
IF LEFT$(pl$(Ø),1)<>LEFT$(pl$(1),1) THEN RETURN←
LINE(55,124)-(88,16Ø),1Ø,bf:r=Ø←
c=c+1:sp=1:px=1:py=1Ø:tp$=pl$(1):x=px*8-1:y=py*8-4:x
L=px←
yL=py+1:p=2:GOSUB ShowCard:px=px+6←
temp$=pl$(Ø):pl$(Ø)=pl$(1):pl$(1)=cd$(m(c)):c=c+1:tp
$=pl$(1)←
x=px*8-1:y=py*8-4:xL=px:yL=py+1:GOSUB ShowCard:px=px
+6:←
pt1=Ø:pt2=Ø:p=2:GOSUB PvaLue←
SpLitLup:GOSUB GetKey←
IF a$="H" THEN GOSUB SPHit←
IF a$="S" AND r=1 THEN  COLOR 4,1Ø:LOCATE py+2,px:PR
INT "Stand":RETURN←
IF a$="S" AND r=Ø THEN vt1=pt1:vt2=pt2:COLOR 4,1Ø:LO
CATE py+2,px:PRINT "Stand":GOSUB SecondHand←
IF pt1>21 AND pt2>21 THEN ←
LOCATE py+2,px:COLOR 4,1Ø:PRINT "Busted!":a$=""←
IF r=1 THEN a$="S":RETURN←
IF r=Ø THEN vt1=pt1:vt2=pt2:GOSUB SecondHand←
END IF←
GOTO SpLitLup←
SPHit:c=c+1:pl$(p)=cd$(m(c)):tp$=pl$(p):x=px*8-1:y=p
y*8-4←
xL=px:yL=py+1:px=px+6←
GOSUB ShowCard:p=p+1:GOSUB PvaLue←
RETURN←
SecondHand:←
pt1=Ø:pt2=Ø:r=1:px=7:py=16:pl$(Ø)=temp$:c=c+1:pl$(1)
=cd$(m(c)):p=2←
tp$=pl$(1):x=px*8-1:y=py*8-4:xL=px:yL=py+1:px=px+6:G
OSUB ShowCard:GOSUB PvaLue:LOCATE 1,1:PRINT pt1,pt2:
RETURN←
GameScreen:GOSUB ShuffLeCards:CLS:GOSUB TitLe←
FOR i=Ø TO 3:j=i*2:LINE(1+j,168+j)-(31Ø-j,19Ø-j),i+1
2,bf:NEXT←
COLOR 2,15:LOCATE 23,3:PRINT "it  tand  ouble-down S
 lit Surr nder"←
COLOR 12,15:LOCATE 23,2:PRINT "H":LOCATE 23,6:PRINT
"S":LOCATE 23,12:PRINT "D"←
LOCATE 23,25:PRINT "p":LOCATE 23,34:PRINT "e":p=Ø:d=
Ø←
dx=1:dy=5:px=1:py=16:FOR i=1 TO 4:tp$=cd$(m(i))←
IF i=1 OR i=3 THEN←
x=px*8-1:y=py*8-4:xL=px:yL=py+1:px=px+6:pl$(p)=tp$:p
```

```
=p+1←
ELSE                ←
x=dx*8-1:y=dy*8-11:xL=dx:yL=dy:dx=dx+6:dl$(d)=tp$:d=
d+1←
END IF←
IF i=2 THEN tp$="  "←
GOSUB ShowCard:NEXT:c=4←
Insurance:←
IF LEFT$(dl$(1),1)="A" THEN←
LOCATE 12,12:COLOR 4,10:PRINT "Purchase insurance? (
Y/N)":GOSUB GetKey←
LOCATE 12,1:PRINT sp$:←
IF a$="Y" THEN←
IF bet-bet-bet*.5<ba THEN←
ins=1:ba=ba-bet*.5←
ELSE←
LOCATE 12,1:PRINT "You can't afford insurance.":GOSU
B DeLay←
LOCATE 12,1:PRINT sp$←
END IF←
ELSE←
IF a$<>"N" THEN←
GOTO Insurance←
END IF   ←
END IF←
GOSUB VdeaLer:←
IF dt1=21 THEN ←
h=1:GOSUB ShowDeaLer:LOCATE dy+2,dx:COLOR 4,10:PRINT
 "Blackjack!":GOSUB DeLay:ba=ba-bet←
IF ins<>1 THEN ba=ba-bet:RETURN ELSE RETURN←
ELSE←
LOCATE 12,8:PRINT "Dealer does not have Blackjack":G
OSUB DeLay←
LOCATE 12,1:PRINT sp$←
END IF←
END IF←
GOSUB PvaLue←
IF pt1=21 AND p=2 THEN←
COLOR 4,10:LOCATE py+2,px:PRINT "Blackjack!":h=1:GOS
UB ShowDeaLer:GOSUB VdeaLer←
IF dt2<>21 THEN←
ba=bet*1.5+ba:GOSUB DeLay:RETURN←
ELSE←
LOCATE dy+2,dx:PRINT "Blackjack!":LOCATE 12,10:PRINT
 "Push, No winner":h=1:GOSUB DeLay←
END IF←
END IF←
GOSUB DeLay←
RETURN←
ShowCard:←
IF tp$="  " THEN PUT (x,y),card(0,4),PSET:RETURN←
```

```
j=ASC(RIGHT$(tp$,1))-65←
PUT (x,y),card(Ø,j),PSET:COLOR 1,4←
LOCATE yL,xL+1:PRINT LEFT$(tp$,1):LOCATE yL+3,xL+4:P
RINT LEFT$(tp$,1)←
RETURN←
TitLe:FOR i=Ø TO 3:j=i*2:LINE(64+j,j)-(256-j,23-j),i
+12,bf:NEXT←
COLOR 2,15: LOCATE 2,13:PRINT "Casino  Blackjack":RE
TURN←
GetKey:a$=INKEY$←
IF a$<>"" THEN←
a$=UCASE$(a$)←
ELSE←
GOTO GetKey←
END IF←
RETURN←
ShuffLeCards:FOR i=1 TO 52:m(i)=i:NEXT:FOR i=1 TO 52
:x=INT(52*RND)+1←
tp=m(i):m(i)=m(x):m(x)=tp:NEXT:CLS:ha=ha+1:RETURN←
InitiaLize:ha=Ø:GOSUB InitShapes:ba=1ØØ:RESTORE card
s:FOR x=Ø TO 3←
FOR t=1 TO 13:READ cd$(x*13+t):cd$(x*13+t)=cd$(x*13+
t)+CHR$(x+65)←
NEXT:RESTORE cards:NEXT:FOR t = 13 TO 1 STEP-1:READ
v$(t):NEXT:RETURN←
cards:DATA A,K,Q,J,T,9,8,7,6,5,4,3,2←
DispLay:WIDTH 4Ø:CLS:COLOR ,Ø←
FOR x=Ø TO 3:LINE (8Ø+x*2,4Ø+x*2)-(225-x*2,63-x*2),1
2+x,bf:NEXT←
COLOR 2,15:LOCATE 7,12:PRINT "Casino Blackjack"←
COLOR 2,1Ø:LOCATE 12,4:PRINT "Copyright 1988 COMPUTE
! Pub., Inc."←
LOCATE 13,11:PRINT "All Rights Reserved":FOR x= 1 TO
 1Ø:GOSUB DeLay:NEXT←
COLOR 4,Ø:RETURN←
InitShapes:RESTORE InitShapes:check=Ø←
LINE(4,4)-(37,4Ø),12,bf:GET (4,4)-(37,4Ø),card(Ø,4):
FOR j=Ø TO 3:FOR i=Ø TO 3Ø←
READ a$:sb(i,j)=VAL("&H"+a$):check=check+sb(i,j):NEX
T i,j←
IF check<>178748& THEN PRINT "Error in Shape Data":G
OSUB DeLay:STOP←
FOR j=Ø TO 3←
LINE(4,4)-(37,4Ø),4,bf:PUT (14,6),sb(Ø,j),PSET:PUT (
17,3Ø),sb(Ø,j),PSET←
GET (4,4)-(37,4Ø),card(Ø,j):CLS:NEXT j:RETURN←
diamond:DATA B,9,3,4ØØ,EØØ,1FØØ,3F8Ø,7FCØ←
DATA 3F8Ø,1FØØ,EØØ,4ØØ,4ØØ,EØØ,1FØØ,3F8Ø←
DATA 7FCØ,3F8Ø,1FØØ,EØØ,4ØØ,FBEØ,F1EØ,EØEØ←
DATA CØ6Ø,8Ø2Ø,CØ6Ø,EØEØ,F1EØ,FBEØ,Ø←
CLub:DATA B,9,3,Ø,Ø,Ø,Ø,Ø←
```

```
DATA Ø,Ø,Ø,Ø,EØØ,1FØØ,1FØØ,7FCØ←
DATA FFEØ,FFEØ,75CØ,EØØ,1FØØ,F1EØ,EØEØ,EØEØ←
DATA 8Ø2Ø,Ø,Ø,8A2Ø,F1EØ,EØEØ,Ø←
Heart:DATA B,9,3,71CØ,FBEØ,FFEØ,FFEØ,7FCØ←
DATA 3F8Ø,1FØØ,EØØ,4ØØ,71CØ,FBEØ,FFEØ,FFEØ←
DATA 7FCØ,3F8Ø,1FØØ,EØØ,4ØØ,8E2Ø,4ØØ,Ø←
DATA Ø,8Ø2Ø,CØ6Ø,EØEØ,F1EØ,FBEØ,Ø←
Spade:DATA B,9,3,Ø,Ø,Ø,Ø,Ø←
DATA Ø,Ø,Ø,Ø,4ØØ,4ØØ,EØØ,1FØØ←
DATA 3F8Ø,7FCØ,75CØ,EØØ,1FØØ,FBEØ,FBEØ,F1EØ←
DATA EØEØ,CØ6Ø,8Ø2Ø,8A2Ø,F1EØ,EØEØ,Ø←
DoScreen:WINDOW OUTPUT 3:COLOR 3,Ø:WIDTH 4Ø:RESTORE
PaLetteData←
FOR i=Ø TO 15:READ r,g,b:PALETTE i,r,g,b:NEXT:RETURN
←
PaLetteData:DATA Ø,.7,Ø,Ø,Ø,Ø,Ø,Ø,Ø,1,Ø,Ø,1,1,1,Ø,Ø,
Ø,1,Ø,Ø,1,1,1,Ø,Ø,Ø←
DATA 1,Ø,Ø,Ø,.7,Ø,1,Ø,Ø,1,Ø,Ø,1,.5,.1,1,.7,.1,1,.6,.
1←
HoLdscrn:LOCATE 12,6:COLOR 5,1Ø:PRINT "Press any key
 to continue....":GOSUB GetKey:RETURN←
EndGame:CLS:COLOR 1,1Ø:LOCATE 12,1Ø:PRINT "Thank you
 for playing":COLOR 11←
LOCATE 14,12:PRINT "Casino Blackjack":GOSUB DeLay:WI
NDOW CLOSE 3←
SCREEN CLOSE 1:END←
←
```

# Canfield

Ed Reynolds
*Translation by George Miller*

*Play a hand of this high-stakes turn-of-the-century solitaire game on your computer. You start with $500, and Fortune is beckoning. You can break the bank or lose your shirt in one of the best computer card games we've ever published.*

Among the many games that people play, solitaire card games have long been among the most popular. Such games are ideal candidates for computerization. With this in mind, I set out to find a unique solitaire game. After a bit of research, I discovered *Canfield,* a game that was in vogue around the turn of the century. Although I did eventually find a few people who still play the game, it is basically unknown nowadays. *Canfield* is as intriguing and challenging as any solitaire game around, and it has a fascinating history.

In the latter part of the nineteenth century, there existed in Saratoga Springs, New York, a lavish casino—the Saratoga Springs Club House. It was known as the "Monte Carlo of America" and entertained European royalty, U.S. senators, and scores of American millionaires. Its founder and proprietor, Richard A. Canfield, retired a multimillionaire when a wave of antigambling sentiment eventually caused the permanent closing of the casino in 1914.

One of the more popular games offered by the casino was a solitaire game invented by and named after the casino's founder. The player would purchase a deck of cards from the casino for $50 (a princely sum in those days). Under the watchful eye of one of Mr. Canfield's croupiers, the player would deal the layout and try to beat the odds. The object was to get all 52 cards (or as many as possible) on four foundation piles. For each card placed upon a foundation pile, the player would receive $5. Sound simple? Just wait.

This version of *Canfield* is written in Amiga Basic. Type it in and save a copy before playing it. Press F1 to quit. Press F10 to start a new game with a new deck.

The computerized version of *Canfield* presented here preserves the spirit of the original game, but the tedious tasks of shuffling and dealing the layout have been usurped by the computer.

As with all card games, *Canfield*'s rules seem more complicated in print than when you're playing the game. Since the computer won't let you make an illegal move, you can learn to play by trial and error. For those who want to know what they're getting into, complete rules are presented below.

## The Rules of the Game

A standard deck of 52 cards is shuffled. (In the computer version, a *T* is used to designate the number 10 cards.) Then, 13 cards are counted off, face down, into a pile, which is then placed face up to the player's left, to form the stock. One card is dealt face up above and to the right of the stock for the first foundation. Then, four cards are placed face up in a row, to the right of the stock, to form the tableau. The remaining 34 cards (held face down) constitute the pack.

In play, the cards in the pack are turned up in batches of three and placed on a talon pile to the right of the pack. The top card of the talon is available for play. When all of the pack has been played onto the talon, the cards are then turned over and become the pack once again.

The other three cards of the same rank as the first foundation card are also foundation cards and, if they become available during play, must be placed up alongside the first. You must then build up on the foundations in suit and sequence until each foundation pile contains 13 cards.

Note that the ranking in each suit is circular—the ranking *wraps around* the king and ace. For example, if the queen of hearts is the foundation, you would build hearts on this pile by playing next the king, then the ace, then the deuce, and so on. Your computer will give you a little help here. Whenever a foundation card is exposed during play, the program will automatically place the card in the proper place in the foundation row.

On the tableau piles, you build downward in alternate colors. The top cards can be played only on foundations. To build on another tableau pile, you must move an entire pile as a unit. If any pile is moved away leaving a space, the top card of the stock must be used to fill the space. Here again, your computer will help by moving the card automatically until the stock is exhausted. Once the stock is exhausted, spaces can be filled from the talon, but at this time, a space may be kept open as long as you want.

In this version of Canfield, your computer will shuffle the deck and deal the layout. You move cards to and from screen locations by pressing specific keys (see Table 2-1).

**Table 2-1. Keypress Reference Guide**

| Key | Action |
|---|---|
| **S** | Move a card from the stock |
| **T** | Move a card from the talon |
| **F** | Move a card to its foundation |
| **P** | Turn over the pack of cards |
| **1** | Move a card to or from tableau 1 |
| **2** | Move a card to or from tableau 2 |
| **3** | Move a card to or from tableau 3 |
| **4** | Move a card to or from tableau 4 |

## Computer Canfield

Canfield provides a way for you to keep your money across games. To accomplish this, Canfield checks a data file on disk to see how much money you had when you last quit the game. When you're asked for your name, that name will be used as a filename to store your account on disk. If it's your first game, or if you drop to $0, you'll start with $500. Remember that it costs $50 to buy a deck of cards to play the game.

## Canfield

Filename: CANFIELD

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
<
' COPYRIGHT (C) 1988 COMPUTE! PUBLICATIONS, INC.<
'                ALL RIGHTS RESERVED<
DEFINT a-Z:DEFSNG r,g,b:RANDOMIZE TIMER<
DIM sb(36,3),CD$(52),M(52),ST$(13),FD$(4,13),TB$(4,2
5),pk$(34),v$(13)<
t$="  "<
<
INPUT"What is your name";NAM$<
yorn:<
PRINT "Do you have an account";:INPUT yorn$<
IF UCASE$(LEFT$(yorn$,1))="Y" THEN<
 OPEN NAM$ FOR INPUT AS #1<
 INPUT#1,bank<
 CLOSE 1 <
ELSE<
 IF UCASE$(LEFT$(yorn$,1))<>"N" THEN yorn<
END IF<
GOSUB DoScreen<
GOSUB InitiaLize<
GOSUB GameScreen<
ChecKey:<
GOSUB CheckTotaL:GOSUB GetKey<
IF key=129 THEN endgame<
IF key=138 THEN GOSUB Busted:GOSUB GameScreen:GOTO C
hecKey<
COLOR 2,15:LOCATE 22,9:PRINT a$<
IF a$="P" THEN GOSUB PacK:GOTO ChecKey<
IF a$="T" THEN GOSUB DoTalon:GOTO ChecKey<
IF a$>="1" AND a$<="4" THEN GOSUB TtoTA:GOTO ChecKey
<
IF a$="S" THEN GOSUB StackPLay:GOTO ChecKey ELSE Che
cKey<
<
GameScreen:<
COLOR 2,10:CLS:WIDTH 40<
FOR c=0 TO 2:FOR x=0 TO 3:LINE (200+x*2,20+(70*c)+x*
2)-(305-x*2,70+(60*c)-x*2),12+x,bf:NEXT x,c<
COLOR 2,15:LOCATE 5,27:PRINT"F1  Quit":LOCATE 7,27:P
RINT"F10 Concede"<
LOCATE 14,28:COLOR 12,15:PRINT "Canfield"<
COLOR 2,10:LOCATE 1,9:PRINT "oundation":LOCATE 5,9<
PRINT"1   2   3   4":LOCATE 9,2:PRINT"tock":LOCATE 2
2,28<
COLOR 2,15:PRINT"Bankroll"<
bank=bank-50:LOCATE 23,29:PRINT"$";bank<
```

```
FOR x=Ø TO 3:LINE (9+x*2,16Ø+x*2)-(77-x*2,191-x*2),1
2+x,bf:NEXT x<
LOCATE 22,3:PRINT"From:":LOCATE 23,5:PRINT"To:"<
COLOR 2,1Ø:LOCATE 23,13:PRINT"ack";:LOCATE 23,18:PRI
NT"alon";<
COLOR 3,1Ø:LOCATE 1,8:PRINT"F":LOCATE 9,1:PRINT"S":L
OCATE 23,12:PRINT"P"<
LOCATE 23,17:PRINT"T":COLOR 2,1Ø<
GOSUB ShuffLeDeck<
<
RETURN<
<
CheckTotaL:<
COLOR 2,15:LOCATE 22,9:PRINT" ":LOCATE 23,9:PRINT" "
;<
TOTAL=Ø:FOR y=1 TO SU :TOTAL=TOTAL+F(y):NEXT:IF TOTA
L<52 THEN RETURN<
CLS:COLOR 14,1:LOCATE 8,1Ø:PRINT"Congratulations, "p
layer$"!"<
LOCATE 1Ø,7:PRINT"You've beaten the odds!"<
LOCATE 12,1Ø:PRINT"Play again (Y/N)?"<
GOSUB GetKey:<
rs:<
IF a$="Y" THEN<
GOSUB GameScreen:GOTO ChecKey<
ELSEIF a$="N" THEN<
endgame<
ELSE <
GOTO rs<
END IF<
<
Busted:<
IF bank>5Ø THEN RETURN<
CLS:COLOR 1,15:LOCATE 1Ø,2Ø:PRINT"Sorry, you only ha
ve $";bank;" remaining.":LOCATE 12,2Ø:PRINT"Would yo
u like to play again? (y/n)"<
Brs:<
GOSUB GetKey:IF a$="Y" THEN <
bank=5ØØ:GOTO GameScreen:GOTO ChecKey<
ELSEIF a$="N" THEN<
GOTO endgame<
ELSE<
GOTO Brs<
END IF<
<
InitiaLize:<
WIDTH 4Ø:COLOR 2,1Ø:CLS<
FOR x=Ø TO 3:LINE (1ØØ+x*2,4Ø+x*2)-(2Ø5-x*2,63-x*2),
12+x,bf:NEXT x<
COLOR 2,15:LOCATE 7,16:PRINT "Canfield"<
COLOR 2,1Ø:LOCATE 14,4:PRINT"Copyright 1988 COMPUTE!
 Pub., Inc."<
```

```
LOCATE 15,11:PRINT"All Rights Reserved":<
<
SetBank:<
IF bank<=Ø THEN bank=5ØØ<
RESTORE cards<
FOR x=Ø TO 3:FOR t=1 TO 13<
READ CD$(x*13+t):CD$(x*13+t)=CD$(x*13+t)+CHR$(x+65):
NEXT<
RESTORE cards:NEXT<
FOR t=13 TO 1 STEP-1:READ v$(t):NEXT:<
GOSUB InitShapes<
RETURN<
<
cards:<
DATA A,K,Q,J,T,9,8,7,6,5,4,3,2<
<
ShuffLeDeck:<
LOCATE 15,5:COLOR 2,1Ø:PRINT"Shuffling Deck...":FOR
i=1 TO 52:M(i)=i:NEXT:FOR i=1 TO 52:x=INT(52*RND)+1:
tp=M(i):M(i)=M(x):M(x)=tp:NEXT<
LOCATE 15,5:PRINT"                  ":pk=34:TN=Ø<
FOR x=1 TO 13:ST$(x)=CD$(M(x)):NEXT:tp$=ST$(13)<
xL=1Ø:yL=2:GOSUB ShowCard:ST=13:ST$(Ø)="  "<
FD$(1,1)=CD$(M(14)):tp$=FD$(1,1):xL=2:yL=7<
GOSUB ShowCard:F=7:SU=1:GOSUB VBank:FOR x=15 TO 18<
TB$(x-14,1)=CD$(M(x)):NEXT<
FOR x=19 TO 52:pk$(x-18)=CD$(M(x)):NEXT:pk$(Ø)="  "<
GOSUB ShowDeck<
<
xL=6:FOR Q=1 TO 4 :tp$=TB$(Q,1):t(Q)=1:F(Q)=Ø<
yL=5+(4*Q):GOSUB ShowCard:NEXT:F(1)=1<
STtoFD:<
IF LEFT$(ST$(ST),1)<>LEFT$(FD$(1,1),1) THEN SetT<
GOSUB DeLaY:SU=SU+1:FD$(SU,1)=ST$(ST):GOSUB NewSuit<
GOSUB DeLaY:i=SU:F=F+4:tp$=ST$(ST):xL=2:yL=F<
GOSUB ShowCard:GOSUB VBank:ST=ST-1<
tp$=ST$(ST):xL=1Ø:yL=2:GOSUB ShowCard:GOTO STtoFD<
SetT:<
t=Ø<
T1tF:<
t=t+1:tp$=TB$(t,1):IF LEFT$(tp$,1)=LEFT$(FD$(1,1),1)
 THEN SetI<
IF t<4 THEN T1tF ELSE RETURN<
SetI:<
i=t:GOSUB DeLaY:GOSUB AddSuit:GOSUB ShowCard<
GOSUB VBank:GOSUB YcalC:tp$="  ":GOSUB ShowCard<
STK:<
IF ST<>Ø THEN <
tp$=ST$(ST):GOSUB YcalC:GOSUB ShowCard<
ST=ST-1:TB$(t,1)=tp$:tp$=ST$(ST):xL=1Ø:yL=2<
GOSUB ShowCard:GOTO STtoFD<
END IF<
```

```
AddSuit:←
SU=SU+1:FD$(SU,1)=tp$:LOCATE 20,18:COLOR 2,10:PRINT
TN←
GOSUB NewSuit:F=F+4:xL=2:yL=F:RETURN←
YcalC:←
xL=6:yL=5+(t*4):RETURN←
←
PacK:←
IF pk=0 THEN pk=TN:TN=0←
IF pk<3 THEN TN=TN+pk:pk=0 ELSE pk=pk-3:TN=TN+3:GOSU
B ShowDeck:GOTO PTaLon←
←
PBLank:←
xL=21:yL=12:COLOR 7,6:tp$="  ":GOSUB ShowCard←
COLOR 2,0:LOCATE 20,11:IF pk<>0 THEN PRINT pk ELSE P
RINT "  "←
GOSUB ShowDeck←
←
PTaLon:←
xL=21:yL=19:tp$=pk$(TN):GOSUB ShowCard←
LOCATE 20,18:COLOR 2,0:PRINT TN←
IF LEFT$(tp$,1)<>LEFT$(FD$(1,1),1) THEN RETURN←
TN=TN-1:IF TN<>0 THEN GOSUB ISuit←
IF TN=0 THEN xL=20:yL=18:COLOR 0,2:tp$="  ":PRINT tp
$:xL=21:yL=19:GOSUB ShowCard←
IF pk=0 THEN xL=20:yL=15:COLOR 0,2:tp$="  ":GOSUB Sh
owCard←
GOTO PBLank←
←
ISuit:←
SU=SU+1:FD$(SU,1)=tp$:LOCATE 20,18:COLOR 2,0:PRINT T
N:GOSUB NewSuit:GOSUB DeLaY←
F=F+4:xL=2:yL=F:GOSUB ShowCard:GOSUB VBank←
IF TN>0 THEN tp$=pk$(TN):xL=21:yL=19:GOSUB ShowCard:
GOSUB ShowDeck←
DPack:←
FOR i=TN+1 TO 33:t=i+1:pk$(i)=pk$(t):NEXT:RETURN←
←
DoTalon:←
GOSUB GetKey←
a=VAL(a$):COLOR 2,15:LOCATE 23,9:PRINT a$:tp$=pk$(TN
) ←
IF a>=1 AND a<=4 AND t(a)=0 THEN TB$(a,1)=tp$:GOTO N
ewCard←
IF a$="F" THEN GOSUB ChkFDTP:IF HIT=1 THEN GOSUB LTa
Lon:HIT=0←
IF a=0 THEN RETURN←
r=a:y=t(a):IF a>=1 OR a<=4 THEN GOSUB ChkTable:IF HI
T=1 THEN GOSUB TnMiN←
GOSUB ShowDeck←
RETURN←
←
```

```
NewCard:←
xL=6:yL=5+(4*a):t(a)=1:GOSUB TnMiN:IF TN=Ø THEN GOSU
B NCard ELSE RETURN←
←
NCard:←
tp$="  ":xL=21:yL=19:GOSUB ShowCard←
IF pk<>Ø THEN GOSUB ShowDeck←
RETURN←
←
DeLaY:←
←
FOR DeLaY=Ø TO VaLC:NEXT:RETURN←
←
ShowCard:←
COLOR 1Ø,1Ø:LOCATE xL,yL:PRINT t$:GOSUB CheckCoLor←
IF tp$="  " THEN←
GOTO Space←
END IF←
LOCATE xL,yL:PRINT LEFT$(tp$,1)←
x=yL*8:y=xL*8-8:j=ASC(RIGHT$(tp$,1))-65←
PUT (x,y),sb(Ø,j),PSET←
PUT (x-8,y+8),sb(Ø,j),PSET←
LOCATE xL+1,yL+1:PRINT LEFT$(tp$,1):RETURN←
Space:←
COLOR 1Ø,1Ø:LOCATE xL,yL:PRINT tp$:LOCATE xL+1,yL:PR
INT tp$:RETURN←
←
←
CheckCoLor:←
IF tp$="  " THEN COLOR 5,6:RETURN←
IF RIGHT$(tp$,1)=CHR$(65) OR RIGHT$(tp$,1)=CHR$(66)
THEN←
COLOR 12,2←
ELSE←
COLOR 5,2←
END IF←
RETURN←
←
VBank: bank=bank+5:COLOR 2,15:LOCATE 23,29:PRINT "
   "←
LOCATE 23,3Ø:PRINT"$"bank:RETURN←
ChkFDTP:←
FOR Q=1 TO SU←
IF RIGHT$(FD$(Q,F(Q)),1)=RIGHT$(tp$,1) THEN F$=FD$(Q
,F(Q)):s=Q←
NEXT:IF s<>Ø THEN IsAce ELSE RETURN←
IsAce:←
IF LEFT$(F$,1)="A" THEN v=1:GOSUB VaLC:GOTO VC←
GOSUB VaLG:GOSUB VaLC←
VC:←
IF v+1<>c THEN RETURN←
F(s)=F(s)+1:FD$(s,F(s))=tp$:xL=2:yL=3+(4*s):HIT=1←
```

```
GOSUB ShowCard:GOSUB VBank:RETURN<
LTaLon:<
xL=21:yL=19:TN=TN-1:GOSUB DPack:GOTO PBLank<
ChkTable:<
IF ASC(RIGHT$(tp$,1))<=66 AND ASC(RIGHT$(TB$(r,y),1)
)<=66 THEN RETURN<
IF ASC(RIGHT$(tp$,1))=>67 AND ASC(RIGHT$(TB$(r,y),1)
)=>67 THEN RETURN<
F$=TB$(r,y)<
IF LEFT$(tp$,1)="A" THEN c=1:GOSUB VaLG ELSE GOSUB V
aLG:GOSUB VaLC<
IF v-1<>c THEN RETURN<
t(r)=t(r)+1:xL=6+(t(r)-1):yL=5+(4*r)<
TB$(r,y+1)=tp$:HIT=1:RETURN<
TnMiN:<
GOSUB ShowCard:TN=TN-1:tp$=pk$(TN)<
xL=21:yL=19:GOSUB ShowCard:GOSUB DPack:HIT=0:GOTO PB
Lank<
              <
               <
endgame:<
CLS:LOCATE 10,8:PRINT"You have $"bank" remaining."<
LOCATE 12,5:PRINT"Thank you for playing ";:COLOR 6:P
RINT"CANFIELD"<
OPEN NAM$ FOR OUTPUT AS #1<
PRINT#1,bank<
CLOSE 1<
WINDOW CLOSE 3:SCREEN CLOSE 1:END<
StackPLay:<
IF ST=0 THEN RETURN<
tp$=ST$(ST)<
GOSUB GetKey<
IF a$="F" THEN GOSUB ChkFDTP:GOSUB NewStock:RETURN<
IF a$>="1" AND a$<="4" THEN r=VAL(a$):y=t(r):GOSUB C
hkTable:GOSUB NewStock:RETURN<
RETURN<
NewStock:<
IF HIT=0 THEN RETURN<
GOSUB ShowCard:ST=ST-1:tp$=ST$(ST)<
xL=10:yL=2:GOSUB ShowCard:HIT=0<
IF ST=0 THEN COLOR 0,2:GOSUB ShowCard:RETURN<
ReChK:<
IF LEFT$(ST$(ST),1)<>LEFT$(FD$(1,1),1) THEN RETURN<
GOSUB DeLaY:SU=SU+1:FD$(SU,1)=ST$(ST):GOSUB NewSuit<
GOSUB DeLaY:i=SU:F=F+4:tp$=ST$(ST)<
xL=2:yL=F:GOSUB ShowCard:GOSUB VBank:ST=ST-1<
tp$=ST$(ST):xL=10:yL=2:GOSUB ShowCard:GOTO ReChK<
<
TtoTA:<
a=VAL(a$):GOSUB GetKey:b$=a$:b=VAL(a$)<
LOCATE 23,9:PRINT b$:t=t(a):s=a<
```

```
IF a>=1 AND a<=4 THEN tp$=TB$(a,t)←
IF b$<>"F" THEN AeB←
IF LEFT$(tp$,1)<>LEFT$(FD$(1,1),1) THEN NoM←
IF SU<4 THEN SU=SU+1 ELSE NoM←
F(SU)=1:FD$(SU,1)=tp$←
NoM:←
GOSUB ChkFDTP←
IF HIT<>1 THEN RETURN←
HIT=Ø←
IF a>=1 AND a<=4 THEN t(a)=t(a)-1:IF t(a)=Ø THEN NoT
A←
tp$="  ":xL=6+t(a):yL=5+(a*4):GOSUB ShowCard←
tp$=TB$(a,t(a)):xL=6+(t(a)-1):GOSUB ShowCard:RETURN←
NoTA:←
tp$="  ":xL=6+t(a):yL=5+(a*4):GOSUB ShowCard←
IF ST<>Ø THEN t=a:GOSUB STK:t(a)=1:RETURN←
AeB:←
IF a=b THEN RETURN←
IF b>=1 AND b<=4 THEN F$=TB$(b,t(b)):GOTO PLyTabLe←
RETURN←
PLyTabLe:←
IF a>=1 AND a<=4 THEN tp$=TB$(a,1)←
IF t(b)=Ø THEN v=1:c=Ø:GOTO cv←
IF ASC(RIGHT$(tp$,1))<=66 AND ASC(RIGHT$(F$,1))<=66
THEN RETURN←
IF ASC(RIGHT$(tp$,1))=>67 AND ASC(RIGHT$(F$,1))=>67
THEN RETURN←
IF LEFT$(tp$,1)="A" THEN c=1:GOSUB VaLG:GOTO cv←
GOSUB VaLC:GOSUB VaLG←
cv:←
IF v-1<>c THEN RETURN←
IF a>=1 AND a<=4 THEN yL=5+a*4:t(a)=Ø←
tp$="  "←
FOR y=t TO Ø STEP-1:xL=7+(y-1):GOSUB ShowCard:NEXT←
yL=5+(b*4):TA=b:sb=t(b)←
←
FOR ty=1 TO t←
TB$(TA,sb+ty)=TB$(s,ty):t(b)=t(b)+1←
tp$=TB$(s,ty):xL=5+(sb+ty)←
GOSUB ShowCard←
NEXT ty←
←
IF t(a)=Ø AND ST<>Ø THEN NoTA←
RETURN←
←
NewSuit:←
F(SU)=1:RETURN←
VaLG:←
g$=F$:GOSUB Vcards:v=g:RETURN←
VaLC:←
g$=tp$:GOSUB Vcards:c=g:RETURN←
```

```
Vcards:<
FOR i=13 TO 1 STEP-1:IF LEFT$(g$,1)=v$(i) THEN g=i+1
<
1520 NEXT:RETURN<
'<
InitShapes:<
RESTORE InitShapes<
FOR j=0 TO 3:FOR i=0 TO 36:<
READ a$:sb(i,j)=VAL("&H"+a$):NEXT i,j<
RETURN<
<
'heart<
DATA 8,8,4,FF00,FF00,FF00,FF00,FF00<
DATA FF00,FF00,FF00,FF00,9900,8100,8100,8100<
DATA C300,E700,FF00,FF00,9900,8100,8100,8100<
DATA C300,E700,FF00,0,6600,7E00,7E00,7E00<
DATA 3C00,1800,0,0,0<
<
'diamond<
DATA 8,8,4,FF00,FF00,FF00,FF00,FF00<
DATA FF00,FF00,FF00,FF00,E700,C300,8100,8100<
DATA C300,E700,FF00,FF00,E700,C300,8100,8100<
DATA C300,E700,FF00,0,1800,3C00,7E00,7E00<
DATA 3C00,1800,0,0,0<
<
'club<
DATA 8,8,4,FF00,E700,E700,9900,9900<
DATA E700,C300,FF00,FF00,E700,E700,9900,9900<
DATA E700,C300,FF00,FF00,E700,E700,9900,9900<
DATA E700,C300,FF00,0,1800,1800,6600,6600<
DATA 1800,3C00,0,0,0<
<
<
'spade<
DATA 8,8,4,FF00,E700,C300,8100,8100<
DATA 8100,E700,FF00,FF00,E700,C300,8100,8100<
DATA 8100,E700,FF00,FF00,E700,C300,8100,8100<
DATA 8100,E700,FF00,0,1800,3C00,7E00,7E00<
DATA 7E00,1800,0,0,0<
<
<
GetKey:<
a$=INKEY$<
IF a$<>"" THEN<
a$=UCASE$(a$)<
key=ASC(a$)<
ELSE<
GOTO GetKey<
END IF<
RETURN<
<
```

```
ShowDeck:←
LOCATE 2Ø,11:COLOR 2,1Ø:PRINT pk←
IF pk=Ø THEN←
COLOR 1Ø,1Ø←
ELSE←
COLOR 15,15←
END IF←
tp$="  ":LOCATE 21,12:PRINT tp$←
LOCATE 22,12:PRINT tp$←
RETURN←
←
DoScreen:←
SCREEN 1,32Ø,2ØØ,4,1:WINDOW 3,"",(Ø,Ø)-(311,186),16,
1:←
WINDOW OUTPUT 3:COLOR 3,Ø:WIDTH 4Ø:RESTORE PaLetteDa
ta:FOR i=Ø TO 15←
READ r,g,b:PALETTE i,r,g,b:NEXT←
RETURN←
←
PaLetteData:←
DATA Ø,.7,Ø←
DATA Ø,Ø,Ø←
DATA 1,1,1←
DATA 1,Ø,Ø←
DATA 1,1,1←
DATA Ø,Ø,Ø←
DATA 1,Ø,Ø←
DATA 1,1,1←
DATA Ø,Ø,Ø←
DATA 1,Ø,Ø←
DATA Ø,.7,Ø←
DATA 1,Ø,Ø←
DATA 1,Ø,Ø←
DATA 1,.5,.1←
DATA 1,.7,.1←
DATA 1,.6,.1←
←
←
```

# Monte Carlo

Thomas Carlson
*Translation by Bill Chin*

*Casino excitement at home can be yours when you type in this solitaire card game. In this recreation of the classic millionaire's gambit, choose pairs of cards and beat the odds. Type in the accompanying listing and save a copy before you run it.*

Monte Carlo is the name of a town in Monaco that is famous as a gambling resort. It also gave birth to the name of a card game with simple rules and complex strategies.

To play "Monte Carlo," shuffle a deck of cards and deal 20 cards into a grid 5 cards across by 4 cards down. The goal of the game is to remove all the cards from the grid. Cards can be removed only in pairs. The cards must be of the same face value for you to remove them. In addition, they must be adjacent vertically, horizontally, or diagonally. The remaining cards slide to the left to fill in the gaps. Gaps at the right edge are filled in by moving cards up from the row below. When all the cards have been moved, two cards from the deck are dealt into the empty slots at the bottom right corner of the board. Eventually, all cards in the deck are exhausted. You win the game if you're able to remove the remaining cards from the board.

As you can imagine, all this moving about of cards can take quite a long time if you're playing with a real deck of cards. Letting the computer do the busywork makes the game more enjoyable. As you play, you'll develop strategies. Each pair you remove can destroy other pairs on the board and create new ones. But you might also have fun just removing the first pair of cards that you see. This game can be played by those who would rather leave their fate to the wind, those who analyze the consequences of each and every move, and all those players in between. Three difficulty levels are included. The level affects the number of rows on the the grid.

*Easy* is six rows, *Medium* is five, and *Hard* is four. Choose the difficulty at the beginning of the game.

Monte Carlo uses the mouse to select cards. To choose a card, move the mouse pointer to the first card and click the left mouse button. Then move the pointer to the other card and click again. The cards are removed and the other cards slide into place. If you wish to deselect a card, point to the selected card and click on it again. Press Q at any time for a new game.

When you've selected a card, move to the matching card (remember, the two cards you wish to remove must have either a corner or an edge in common) and press RETURN again. The computer will not let you make an illegal move.

## Monte Carlo

Filename: MONTE CARLO

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
' Copyright 1987 COMPUTE! Publications, Inc.  All ri
ghts reserved.←
GOSUB DefinePLayFieLd←
start:←
 DEFINT a-z←
 CLS:PRINT " Copyright 1987 COMPUTE! Publications":P
RINT TAB(10)"All rights reserved."←
 LOCATE 4,14:PRINT "MONTE CARLO":PRINT←
 DIM  c$(52),v$(13),t$(4),c2(52)←
 c$(52)=" -"←
 RESTORE:FOR i=1 TO 13←
 READ a$:v$(i)=a$:NEXT←
 DATA " A"," 2"," 3"," 4"," 5"," 6"," 7"," 8"," 9","
10"," J"," Q"," K"←
 PRINT TAB(12);"Shuffling Deck"←
 RANDOMIZE TIMER←
 FOR i=1 TO 4:FOR j=1 TO 13←
 p=INT(RND(1)*52)←
skip2: IF c$(p)="" THEN c$(p)=v$(j):c2(p)=(i-1)*43:G
OTO skip1←
 p=p+1:IF p=52 THEN p=0←
 GOTO skip2:←
skip1: NEXT:NEXT←
 GOSUB suitshapes←
 SOUND 440,1.5←
 PRINT:PRINT←
 PRINT TAB(14)"Click on:":PRINT←
 PRINT TAB(16)"HARD"←
 PRINT TAB(16)"MEDIUM"←
 PRINT TAB(16)"EASY"←
skip4: WHILE MOUSE(Ø)=Ø:WEND←
 x=MOUSE(1):y=MOUSE(2)←
 IF x<12Ø OR x>17Ø OR y<8Ø OR y>1Ø3 THEN skip4←
 IF y<1Ø4 THEN c=6←
 IF y<96 THEN c=5←
 IF y<88 THEN c=4←
 WHILE MOUSE(Ø)<>Ø:WEND←
display:←
 ct=Ø:CLS:LOCATE 5,1←
FOR i=1 TO c:t=4←
 FOR j=1 TO 5←
  PRINT SPC(t);c$(ct);:t=5←
  IF c$(ct)<>" -" THEN PUT (j*56-6,i*24+8),suit(c2(c
t))←
  ct = ct+1←
 NEXT←
 PRINT:PRINT:PRINT ←
NEXT←
PRINT TAB(12);"EXIT";SPC(8);"RESTART";←
 IF c$(Ø)=" -" THEN LOCATE 2,12:PRINT"Tableau is cle
ared":GOTO winner←
 x=Ø:y=Ø←
 cr$=CHR$(62):GOSUB cursor:x1=x:y1=y:p1=y*5+x:SOUND
44Ø,1.5←
 cr$=CHR$(187):GOSUB cursor:x2=x:y2=y:p2=y*5+x←
 IF p1=p2 THEN display←
 IF (LEFT$(c$(p1),2))<>(LEFT$(c$(p2),2)) THEN invaLi
d←
 IF ABS(x2-x1)>1 OR ABS(y2-y1)>1 THEN invaLid←
 SOUND 66Ø,1.2←
 FOR i=p1 TO 51←
 c$(i)=c$(i+1):c2(i)=c2(i+1):NEXT←
 IF p2>p1 THEN p2=p2-1←
 FOR i=p2 TO 51←
 c$(i)=c$(i+1):c2(i)=c2(i+1):NEXT←
 GOTO display←
winner: ←
 FOR i=1 TO 11←
 SOUND 44Ø,1:SOUND 66Ø,1:a$=INKEY$←
 NEXT:game=1←
 GOSUB cursor←
invaLid:←
 SOUND 15Ø,3:GOTO display←
cursor:←
 WHILE MOUSE(Ø)=Ø:WEND←
 xx=(MOUSE(1)-12)/8:yy=(MOUSE(2)-36)/12←
 x=(xx-6)/7:y=yy/2←
 IF y<>c OR (yy MOD 2)=1 THEN skip3←
  IF xx>9 AND xx<14 THEN CLS:END←
  IF xx>2Ø AND xx<29 THEN CLEAR ,,25ØØØ:GOTO start←
 SOUND 19Ø,1.9:GOTO cursor ←
skip3: ←
 IF game THEN SOUND 19Ø,1.9:GOTO cursor ←
 IF (xx MOD 7)<3 OR (yy MOD 2)<>Ø THEN SOUND 19Ø,1.9
:GOTO cursor←
 x=(xx-6)/7:y=yy/2←
 IF x>4 OR x<Ø OR y<Ø OR y>=c THEN SOUND 19Ø,1.9:GOTO
  cursor←
 tx=x*56+3Ø:ty=y*24+3Ø:LINE (tx,ty)-(tx+4Ø,ty+1Ø),3,
b ←
 WHILE MOUSE(Ø)<>Ø:WEND←
 RETURN←
 DefinePLayFieLd:←
  SCREEN 1,32Ø,2ØØ,2,1←
  WINDOW 1,"MONTE CARLO",,2,1←
  PALETTE  Ø,.5,.5,.9←
  PALETTE  1,Ø,Ø,Ø←
  PALETTE  2,1,Ø,Ø←
  PALETTE  3,.9,.9,Ø←
 RETURN←
suitshapes:←
 DIM suit(299):RESTORE suitshapes←
 m= 42 :m2= 1Ø :GOSUB ReadCompressed←
 DATA  9, 7, 2, 6144, 1536Ø, 32256,-256,-256 ←
 DATA  6144, 1536Ø, 4ØØ33 ←
 m= 42 :m2= 1Ø :GOSUB ReadCompressed←
 DATA  8, 7, 2, Ø, 6144, 6144, 32256, 26112 ←
 DATA  6144, 1536Ø, 4ØØ33 ←
 m= 42 :m2= 11 :GOSUB ReadCompressed←
 DATA  8, 7, 2, 4ØØØ7, 2Ø48, 7168, 15872, 32512 ←
 DATA  15872, 7168, 2Ø48, 4ØØ26 ←
 m= 42 :m2= 11 :GOSUB ReadCompressed←
 DATA  8, 7, 2, 4ØØØ7, 27648,-512,-512,-512 ←
 DATA  31744, 14336, 4Ø96, 4ØØ26 ←
RETURN←
ReadCompressed:←
 FOR j=Ø TO m2←
 READ t& ←
 IF t&<4ØØØØ& THEN suit(i2)=t&:i2=i2+1 ELSE FOR i=Ø
TO t&-4ØØØØ&:suit(i+i2)=Ø:NEXT:i2=i2+t&-4ØØØØ&←
 NEXT ←
RETURN←
```

# CHAPTER THREE

# Action Arcade Games

# Climber 5

James Rogers
*Translation by Bill Chin*

*This fast-moving arcade game is brilliantly conceived. As a construction worker at a building behind the ballpark, you want to retrieve home-run balls hit by your favorite all-stars. Time is of the essence, but you must avoid moving hooks of various sizes. This keyboard-controlled, one-player game requires at least 512K of memory.*

"Play ball," the umpire cries, and the game begins. As a construction worker at the site of a new building behind the ballpark, your task seems simple: Retrieve stray balls and add them to your collection. The number 5 on the back of your working uniform shows that at heart you're part of the team. When a ball flies out of the park onto the building, the game grinds to a halt while a stadium full of fans and players cheers your valiant effort to get it down.

You're used to the pressure, but that doesn't make the waiting any easier. The first inning passes, followed by the second and third, without any problem. Then, at the top of the fourth inning, the first batter swats a towering homer over the left-field wall. Up, up it goes, so high that you grab your field glasses to track it. Yes, it's outside the stadium. It looks like the ball lodged right at the top of the building, which is still under construction. As you rush to retrieve the prize, you'll have to duck and dodge to avoid obstacles on the construction site.

## About the Program

The listing that accompanies this article is ready to type in. Be sure to save a copy before you run it. The shapes for the animated climber and moving hooks were first drawn with *Deluxe Paint* and then were converted for BASIC with the "IFF

Translator'' program also included in this book. A simple compression algorithm is used to reduce the amount of shape data you need to type in. The subroutine ReadCompressed handles data for one shape each time it is called. Data elements with a value less than 40000 are stored directly in the shape array. Any number greater than 40000 represents a group of zero values. The routine subtracts 40000 from the value to determine how many zeros to place in the array. For instance, the number 40015 means that we need to put 15 zeros in the array. The number 40000 is used because a normal shape data value would never be greater than 32768. One reason the shape data contains many zeros is that not all the colors are used in every shape.

## Playing the Game

The Climber 5 screen consists of several horizontal levels. When the game begins, you are at the bottom right corner of the screen, and the ball is at the upper left. Your job is to climb and run to where the ball lies, avoiding the moving hooks along the way. Since some of the hook objects are speedy indeed, that's more difficult than you might imagine at first. You can move left or right with cursor controls. There are short and long hooks. The long hooks must be avoided by running to a different place on the construction frame. The short hooks may be evaded by ducking down (press the down cursor key to duck).

Each level is connected to the next by one or more ladders. The cursor controls allow you to move up or down a ladder; of course, you must be perfectly aligned with the ladder in order to ascend or descend on it.

You have a total of five players when the game begins. Whenever you get ''hooked'' (hit by a moving hook), you lose one player. That really hurts. You can enjoy the Amiga speech emulator as it reports your demise with an ungrateful ''OUCH!'' The game ends when you have lost all your players. If you reach the ball without being hit, the program displays a congratulatory message and lets you try the next skill level, where everything becomes more difficult.

## Climber 5

Filename: CLIMBER5

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.←
'  All Rights Reserved←
←
'Climber 5←
'Copyright 1987←
'Compute! Publications, Inc.←
'All Rights Reserved.←
DEFINT a-z←
men=5:SAY TRANSLATE$("climer five!")←
DIM map(45,24),mdat(100),chk&(7),ex(4),ey(4),ty(4),o
ld(2),mindex(4):endmap=100←
GOSUB DefinePLayFieLd←
RESTORE mapdata:t=1:cm=1←
WHILE t<>-1←
 READ t:mdat(c)=t :c=c+1←
 IF t=0 THEN mindex(cm)=c:cm=cm+1←
 WEND←
←
mapdata:←
DATA 1,6, 1,12, 1,18, 2,2, 2,9, 2,15, 3,6, 3,12, 3,1
8, 4,2, 4,9, 4,15, 0←
DATA 1,3, 1,16, 2,5, 2,15, 3,9, 3,11, 4,7, 4,14, 0
←
DATA 1,9, 2,3, 2,13, 3,7, 3,17, 4,14, 0←
DATA 1,7, 2,17, 3,3, 3,10, 4,15, 0←
DATA 1,10, 2,4, 3,13, 4,9, -1←
 ←
stand=0:c1= 867:rt1=313:rt2=420:duck=103:baLL=654:gi
rd=603:hook=685←
 c2=974:Lt1=1081:Lt2=1188:shorthook=1371←
GOSUB makeshapes:←
NewBoard:←
GOSUB MakeMap:px=266:py=138:psn=313←
PUT (px,py),shape(psn):cf=0←
FOR i=0 TO 4:ex(i)=320:NEXT←
main:←
WHILE vnext=0←
'keyboard input←
 kf=33:dx=0:dy=0:a=0←
WHILE dx=0 AND dy=0 AND kf>0←
 a$=INKEY$:a=ASC(a$+" ")←
 IF a=29 THEN dy=8:GOTO keypressed←
 IF a=28 THEN dy=-8:GOTO keypressed←
 IF a=31 THEN dx=-8:GOTO keypressed←
 IF a=30 THEN dx=8←
 kf=kf-1←
```

```
keypressed:←
 WEND ←
 GOSUB pLayer←
'hooks←
 FOR n = Ø TO 4 ←
  IF ex(n)<318 THEN onscreen←
  PUT(ex(n),ey(n)),shape(hook),XOR←
  t!=RND(1):IF t!>.3 THEN eskip←
  IF  t!>.2 THEN esn(n)=hook ELSE esn(n)=shorthook←
  ex(n)=Ø:ty(n)=n*4+1:ey(n)=n*32+9←
  PUT (ex(n),ey(n)),shape(esn(n)),XOR←
 onscreen:←
  th=esn(n) ←
  PUT (ex(n),ey(n)),shape(th),XOR←
  ex(n) = ex(n)+16←
  PUT (ex(n),ey(n)),shape(th),XOR←
  tx=ex(n)/8:ty=ty(n)←
  IF th=hook AND ( map(tx-1,ty+1)=2 OR map(tx,ty+1)=
2 OR map(tx-2,ty+1)=2 ) THEN GOSUB kiLLed←
  IF map(tx-1,ty)=2 OR map(tx,ty)=2 OR map(tx-2,ty)=
2 THEN GOSUB kiLLed←
 eskip: ←
 NEXT←
WEND←
 t$="I got the ball":IF building>Ø THEN t$=t$+" agai
n."←
 IF building=4 THEN t$="I am getting tired."←
 IF vnext=2 THEN SAY TRANSLATE$(t$)←
 vnext=Ø←
 IF men>Ø THEN NextMap←
 SAY TRANSLATE$("game over.")←
 LOCATE 1,1:PRINT "  (C)ontinue ":PRINT "  (R)estart
  ":PRINT "  (Q)uit "←
 men=5←
 Loopg: a$=INKEY$:IF a$="r" THEN building=Ø:GOTO New
Board←
 IF a$="c" THEN GOTO NewBoard←
 IF a$="q" THEN CLS:CLEAR:STOP←
 GOTO Loopg←
NextMap:←
 building=(building+1) MOD 5 ←
 GOTO NewBoard←
kiLLed:←
 SAY TRANSLATE$("ouch.")←
 PUT (px,py),shape(psn),XOR←
 FOR i=Ø TO 2:map (mx,my+i-2)=old(i):old(i)=Ø:NEXT
←
 PUT (ex(4),ey(4)),shape(esn(4)),XOR←
 px=266:py=138←
 ex(4)=318:psn=stand:cf=Ø←
 PUT (px,py),shape(psn):men=men-1←
 IF men=<Ø THEN vnext=1:men=Ø←
 LOCATE 23,8:PRINT " Climbers ";men;" Level ";buildi
ng+1;←
 RETURN←
  ←
pLayer:←
 WHILE INKEY$<>"":WEND←
 mx=px/8:my=(py+17)/8←
 map (mx,my-2)=old(Ø)←
 map (mx,my-1)=old(1)←
 map (mx,my)=old(2) ←
 IF dy AND cf=Ø THEN GOSUB trycLimb←
 ON cf GOTO cLimbing,ducking ←
 IF dx THEN moveplay←
 sf=sf+1:REM standing on girder←
 IF sf>2 THEN PUT (px,py),shape(psn),XOR:PUT (px,py)
,shape(stand):sf=1:psn=stand←
 GOTO oLdmap←
←
trycLimb:←
 t=my:IF dy>Ø THEN t=my+1←
 IF map(mx,t)=1 THEN cf=1:RETURN←
 IF dy>Ø THEN PUT (px,py),shape(psn),XOR:PUT (px,py)
,shape(duck):psn=duck:cf=2←
 RETURN←
  ←
movepLay:←
 sf=Ø :pb=rt1:IF dx<Ø THEN pb=Lt1←
 tx=px+dx :ty=py←
 IF tx<8 OR tx>3ØØ THEN tx=px←
 IF tx=1Ø AND ty=1Ø THEN fr=1:pb=duck:vnext=2←
 GOTO animate←
 ←
ducking:←
 IF dx OR dy<Ø THEN cf=Ø:psn=duck:RETURN←
 GOSUB oLdmap ←
 map(mx,my-2)=Ø←
 RETURN←
  ←
cLimbing:←
 IF dy=Ø THEN oLdmap←
 pb=c1:tx=px:ty=py+dy:tmy=(ty+17)/8 ←
 IF (tmy AND 3)=3 THEN cf=Ø:pb=stand:fr=1 ←
 GOTO animate←
  ←
animate:←
 PUT (px,py),shape(psn),XOR←
 fr=1-fr:psn=pb+fr*1Ø7←
 PUT (tx,ty),shape(psn)←
 px=tx:py=ty:mx=px/8:my=(py+17)/8←
oLdmap: ←
```

```
 old(Ø)=map(mx,my-2):map(mx,my-2)=2<
 old(1)=map(mx,my-1):map(mx,my-1)=2<
 old(2)=map(mx,my):map(mx,my)=2<
 RETURN <
  <
MakeMap:<
 FOR i= Ø TO 39:FOR j=Ø TO 24<
  map(i,j)=Ø:NEXT:NEXT<
 c = mindex(building):CLS :yc=Ø<
 FOR y=yc TO yc+18Ø STEP 32:FOR x=Ø TO 319 STEP 8<
  PUT (x,y),shape(gird):NEXT:NEXT<
 WHILE mdat(c)>Ø<
  y2=mdat(c):x2=mdat(c+1)*2:my=y2*4<
  FOR j=my TO my+3:map(x2,j)=1:NEXT <
  tx=x2*8:ty=y2*32-5<
  LINE (tx,ty)-(tx+16,ty+36),Ø,bf<
  LINE (tx,ty)-(tx+2,ty+36),8,bf<
  LINE (tx+14,ty)-(tx+16,ty+36),8,bf<
  FOR j=ty+4 TO ty+28 STEP 8<
   LINE (tx+2,j)-(tx+14,j),8,b:NEXT j<
 temp: <
  c=c+2<
  WEND<
  PUT (2Ø,28),shape(baLL):map(Ø,Ø)=-1  <
  LOCATE 23,8:PRINT " Climbers ";men;" Level ";build
ing+1;                             <
 RETURN <
makeshapes:<
i2=Ø :RESTORE makeshapes:CLS<
LOCATE 1Ø,2:PRINT "Copyright 1987 Compute! Publicati
ons"<
LOCATE 12,1Ø:PRINT "All rights reserved." <
FOR i=Ø TO 7:READ chk&(i):NEXT<
' checksums<
DATA 445496, 818859, 1393851<
DATA 19Ø2864, 2ØØ21Ø9, 21ØØ91Ø<
DATA 24Ø6221, 26Ø626Ø <
DIM shape(1599)<
'shape Ø  man standing still<
m= 1Ø2 :m2= 58:GOSUB ReadCompressed<
DATA 14 , 22 , 4 , 4ØØØ2 , 256 , 896 , 14Ø8 , 3968 <
DATA 1792 , 816Ø , 16368 , 3Ø664 ,-4124 , 32756 , 15
736 , 8176 <
DATA 3Ø4Ø , 24ØØ , 736 , 224Ø , 2784 , 38Ø8 , 792Ø ,
 1548Ø <
DATA 4ØØØ2 , 768 , 3968 , 128Ø , 384Ø , 768 , 384Ø ,
 816Ø <
DATA 16368 , 3Ø68Ø , 14296 , 6832 , 2Ø8Ø , 1Ø24 , 16
64 , 3Ø72 <
DATA 1568 , 1Ø24 , 4ØØØ7 , 128Ø , 384Ø , 768 , 1536
, Ø <
```

```
DATA 2080 , 4112 , 0 , 2720 , 4064 , 1728 , 1728 , 3
136 <
DATA 1632 , 1088 , 40037 <
'shape 1  cLimbing<
m= 102 :m2= 43:GOSUB ReadCompressed<
DATA 14 , 22 , 4 , 40007 , 128 , 448 , 640 , 1920 <
DATA 896 , 8176 , 16376 , 16376 , 16376 , 16376 , 81
76 , 8176 <
DATA 7280 , 7280 , 31868 , 40007 , 384 , 1984 , 640
, 1920 <
DATA 128 , 896 , 8176 , 5008 , 4368 , 12312 , 4112 ,
 40013 <
DATA 640 , 1920 , 128 , 896 , 256 , 0 , 3104 , 15288
 <
DATA 6448 , 2080 , 2080 , 40036 <
'shape 2  ducking<
m= 106 :m2= 56:GOSUB ReadCompressed<
DATA 14 , 23 , 4 , 40003 , 8192 , 29440 , 30592 , 22
400 <
DATA 29440 , 27872 , 24720 , 13640 , 6728 , 3832 , 3
432 , 7664 <
DATA 7024 , 5808 , 4848 , 6896 , 7920 , 7920 , 7776
, 3072 <
DATA 40004 , 8960 , 1920 , 8192 , 8960 , 13056 , 163
52 , 8176 <
DATA 2032 , 2032 , 656 , 528 , 1152 , 2112 , 3072 ,
1024 <
DATA 40008 , 8192 , 40002 , 768 , 4864 , 0 , 2048 ,
1056 <
DATA 0 , 640 , 4080 , 3296 , 3168 , 3104 , 3168 , 30
72 <
DATA 40038 <
'shape 3 run right<
m= 106 :m2= 63:GOSUB ReadCompressed<
DATA 13 , 23 , 4 , 0 , 512 , 3328 , 8064 , 7424 <
DATA 8064 , 3856 , 5688 , 14200 , 25584 ,-8288 , 327
04 , 16128 <
DATA 7936 , 8064 , 6080 , 9184 , 19952 , 23008 ,-277
12 ,-2176 <
DATA 29568 , 14816 , 0 , 1536 , 3840 , 8128 , 1280 ,
 1920 <
DATA 3584 , 1552 , 7728 , 16224 , 28480 , 14080 , 66
56 , 3072 <
DATA 0 , 2048 , 7168 , 12288 , 8192 , 24576 , 40007
, 1280 <
DATA 1920 , 3584 , 1552 , 16 , 0 , 64 , 0 , 2048 <
DATA 3072 , 768 , 2944 , 7616 , 12384 , 8384 , 24960
 , 40038 <
'shape 4  running right second frame<
m= 182 :m2= 95:GOSUB ReadCompressed<
DATA 25 , 20 , 4 , 40002 , 32 , 0 , 112 , 0 <
DATA 254 , 0 , 232 , 0 , 252 , 0 , 120 , 0 <
DATA 240 , 0 , 944 , 0 , 1919 ,-32768 , 3519 ,-16384
 <
DATA 4031 ,-32768 , 2040 , 0 , 1023 , 0 , 383 ,-1638
4 <
DATA 15871 ,-16384 , 32739 ,-8192 ,-64 ,-512 ,-6208
, 32256 <
DATA 384 , 28672 , 40002 , 48 , 0 , 120 , 0 , 254 <
DATA 0 , 40 , 0 , 60 , 0 , 112 , 0 , 112 <
DATA 0 , 224 , 0 , 688 , 0 , 1759 ,-32768 , 608 <
DATA 0 , 768 , 40003 , 128 , 40019 , 40 , 0 , 60 <
DATA 0 , 112 , 0 , 48 , 40004 ,-32768 , 513 ,-16384
<
DATA 0 ,-32768 , 768 , 0 , 752 , 0 , 239 , 0 <
DATA 227 ,-32768 , 1216 ,-16384 , 1920 , 16384 , 384
 , 40063 <
'shape 5  girder<
m= 50 :m2= 25:GOSUB ReadCompressed<
DATA 8 , 9 , 4 , 0 , 1024 ,-1024 , 30720 , 13056 <
DATA -30976 ,-12544 , 4096 , 40002 , 8192 , 40005 ,-
32256 , 0 <
DATA -256 ,-256 , 768 ,-30976 ,-13312 , 30720 , 1228
8 ,-256 <
DATA -256 , 40021<
'shape 6  ball<
m= 30 :m2= 14:GOSUB ReadCompressed<
DATA 5 , 4 , 4 , 30720 , 18432 , 22528 , 30720 , 122
88 <
DATA 30720 , 30720 , 12288 , 18432 , 40002 , 18432 ,
 40016  <
'shape 7  hook<
m= 182 :m2= 53:GOSUB ReadCompressed<
DATA 15 , 20 , 4 , 4088 , 2032 , 992 , 448 , 448 <
DATA 448 , 448 , 448 , 448 , 992 , 480 , 448 , 4032
<
DATA 16352 , 30720 ,-4096 ,-4096 , 28672 , 14340 , 4
088 , 4096 <
DATA 2496 , 1152 , 512 , 40006 , 512 , 0 , 4128 , 16
384 <
DATA -32768 , 2048 , 0 ,-30720 , 17416 , 4100 , 4096
 , 2048 <
DATA 1024 , 512 , 40006 , 512 , 0 , 4128 , 16384 ,-3
2768 <
DATA 2048 , 0 ,-30720 , 17416 , 4100 , 40120 <
PUT (0,0),shape(206):PUT (30,0),shape(rt1)<
PUT (60,0),shape(rt2):PUT (90,0),shape(hook)<
GET (90,12)-(105,20),shape(shorthook)<
GET (0,0)-(14,23),shape(c1)<
x2=0:y2=0:y3=40:psn=c2:nx=14:ny=23:GOSUB ReverseBob<
x2=30:psn=Lt1:nx=13:ny=23:GOSUB ReverseBob<
x2=60:psn=Lt2:nx=25:ny=20:GOSUB ReverseBob<
RETURN<
```

```
ReverseBob:←
 FOR i= Ø TO nx-1←
  FOR j=Ø TO ny-1←
   t=POINT (i+x2,j+y2)←
   PRESET (x2+nx-i,j+y3),t←
  NEXT j←
 NEXT i←
 GET (x2,y3)-(x2+nx,y3+ny),shape(psn)←
 RETURN←
ReadCompressed: ←
FOR j=Ø TO m2 ←
 READ t&:s&=s&+t&←
 IF t&<4ØØØØ& THEN shape(i2)=t&:i2=i2+1 ELSE FOR i=Ø
 TO t&-4ØØØØ&:shape(i+i2)=Ø:NEXT:i2=i2+t&-4ØØØØ&←
NEXT←
IF chk&(ns)<>s& THEN PRINT "error in checksum";ns:PR
INT " or in shape";ns;"data statements":STOP←
ns=ns+1←
RETURN←
←
DefinePLayFieLd:←
  SCREEN 1,32Ø,2ØØ,4,1←
  WINDOW 1,"Climber 5",,2,1←
  RESTORE DefinePLayFieLd←
  FOR i=Ø TO 7←
  READ a!,b!,c!←
  PALETTE i,a!,b!,c!←
  PALETTE i+8,a!,b!,c!←
  NEXT :PALETTE 8,.25,.25,.25←
  DATA .45, .45, .6, .1, .1, .1, .85, .85, .8, .8, .
7, .7←
  DATA .85, .1, .1, .6, .45, .4, .45, .4, .3, 1, .6,
 .5←
  RETURN ←
```

# Marbles

Stephen Stout
*Translation by Tim Midkiff*

*"Marbles" is a hypnotic action program in which you try to catch marbles in a bucket. Act quickly—and don't panic—in this clever arcade-style game. A joystick and Workbench 1.1 or a recent Workbench 1.2 are required. For 1–5 players.*

In two weeks, the Statewide Marbles Championship will be held in Localsville. You've been practicing for nearly a year, and now, at the last minute, you find you've lost your favorite marble.

"Marbles" is written in BASIC. Type it in and save it to disk. After you have a copy safely tucked away, simply load and run the program.

To play Marbles, you must have a Workbench 1.1 disk or the most recent version of Workbench 1.2. The first release of 1.2 does not properly allocate sprites 6 and 7.

A joystick plugged into port 1 controls the buckets at the bottom of the screen. You must catch the red marbles in the red bucket and the blue marbles in the blue bucket. The arrows at the intersections of the pipes control the flow of marbles. Press the fire button to change the direction of all the arrows on the screen. The arrows move counterclockwise.

The game of Marbles can be played by up to five players. The one-player game starts with 3 marbles on the screen at once. After every ten points, another marble is added until there are 6 marbles on the screen at once. The game ends when 40 marbles have been caught. Your score is the percentage of marbles you caught.

In the multiplayer game (two to five players), all contestants play the same level. After each has had a turn, the player with the lowest score is dropped from the game. When only one player remains, he or she is declared the winner. If two or more players tie for low score, the level is played again.

Before running Marbles, you will either have to move or create a file called graphics.bmap. If you use Workbench 1.1, you must first copy the "graphics.bmap" file to your boot disk. Follow the procedure given for the "System Fonts" article on page 216.

Users of the 1.2 version of the Extras disk should see the instructions in Appendix B.

## Marbles

Filename: MARBLES

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'Marbles<
'Copyright 1987 COMPUTE! Publications, Inc.<
'All Rights Reserved.<
CLEAR ,25000:CLEAR ,50000&<
DEFINT a-z:RANDOMIZE TIMER<
LIBRARY "graphics.library"<
DECLARE FUNCTION GetSprite&() LIBRARY<
DECLARE FUNCTION ALLocRaster&() LIBRARY<
DIM sc(21,40),pno(21,40),pipmap(21,40),arrow(24,2),x
i(30),yi(30)<
DIM di(30),ti(30),fbb(12),fbr(12),w(255)<
SCREEN 1,320,200,4,1:WINDOW 3,,(0,0)-(311,186),16,1:
WINDOW OUTPUT 3<
RESTORE CoLors:FOR i=0 TO 15:READ r!,g!,b!:PALETTE i
,r!,g!,b!:NEXT<
CoLors:<
DATA 0,0,0,.7,.7,.7,.3,.3,.3,.8,0,.8,.6,0,.6,.2,.2,.
2,0,0,.8,0,0,.65<
DATA 0,0,.5,0,0,.4,.65,0,0,.5,0,0,.4,0,0,.25,0,0,.5,
.5,.5,.6,.6,.6<
PALETTE 22,0,0,1:PALETTE 23,0,0,.6:PALETTE 26,1,0,0<
PALETTE 27,.6,0,0:PALETTE 30,0,0,1:PALETTE 31,0,0,.6
<
COLOR 1,5:CLS:GOSUB InitSprites<
ON BREAK GOSUB CLoseSprites:BREAK ON<
DirDat:RESTORE DirDat:FOR i=0 TO 2:READ dx(i),dy(i):
NEXT:DATA -1,0,0,1,1,0<
RESTORE LeftArrow:FOR i=0 TO 24:READ a$:arrow(i,0)=V
AL("&H"+a$):NEXT<
RESTORE DownArrow:FOR i=0 TO 24:READ a$:arrow(i,1)=V
AL("&H"+a$):NEXT<
RESTORE RightArrow:FOR i=0 TO 24:READ a$:arrow(i,2)=
VAL("&H"+a$):NEXT<
RESTORE CupData:FOR i=1 TO 202:READ a:cup$=cup$+CHR$
(a):NEXT<
```

```
OBJECT.SHAPE 1,cup$←
LINE(2,Ø)-(6,Ø),7:LINE(Ø,1)-(8,1),8:GET(Ø,Ø)-(8,1),f
bb←
LINE(2,Ø)-(6,Ø),11:LINE(Ø,1)-(8,1),12:GET(Ø,Ø)-(8,1)
,fbr←
NewGame:COLOR 1,5:CLS:PRINT TAB(12) "Copyright 1987"
 ←
PRINT TAB(7) "COMPUTE! Publications, Inc.":PRINT TAB
(1Ø) "All Rights Reserved."←
PRINT:PRINT:PRINT"Number of Players (1-5)"←
PlNum:k$="":WHILE k$="":k$=INKEY$:WEND:IF k$<"1" OR
k$>"5" THEN PlNum←
CLS:numpL=VAL(k$)-1:pts=Ø:pn=numpL←
FOR i=Ø TO 255:w(i)=INT(RND*255-128):NEXT:WAVE 3,w←
IF numpL=Ø THEN←
LOCATE 23,6:PRINT"Marbles:":LOCATE 23,25:PRINT"Ratin
g:":npip=3←
ELSE←
LOCATE 1Ø,1:PRINT"Enter the initials of player"←
FOR i=Ø TO numpL:LOCATE 1Ø,29:PRINT LEFT$(STR$(i+1),
2);:INPUT nam$(i)←
nam$(i)=LEFT$(nam$(i)+SPACE$(3),3):LOCATE 1Ø,29:PRIN
T SPACE$(11):NEXT←
CLS:ofst=INT(34/numpL):npip=7-numpL←
FOR i=Ø TO numpL:cp(i)=i:LOCATE 23,i*ofst+2:PRINT na
m$(i):NEXT←
FOR i=Ø TO numpL:j=INT(RND*(numpL+1)):SWAP cp(i),cp(
j):NEXT←
END IF←
NewMap:←
FOR i=Ø TO 2:ck(i)=Ø:NEXT←
FOR i=2 TO 2Ø:FOR j=2 TO 38:sc(i,j)=Ø:pno(i,j)=Ø:pip
map(i,j)=Ø:NEXT j,i←
IF numpL>Ø THEN←
LOCATE 23,cp(pn)*ofst+2:COLOR 4,5:PRINT nam$(cp(pn))
←
LOCATE 21,1:PRINT SPACE$(4Ø):IF pn=numpL THEN LOCATE
 24,1:PRINT SPACE$(39);←
END IF←
pipe=Ø:LINE(Ø,8)-(312,159),Ø,bf←
NewPipe:←
row=2:pipe=pipe+1:COLOR 1,2←
Start:←
coL=INT(RND*37)+2:IF (sc(2,coL-1) OR sc(2,coL) OR sc
(2,coL+1))>Ø THEN Start←
GOSUB Down:IF Ln<3 THEN Start ←
ballx(pipe-1)=coL:baLLy(pipe-1)=row:pipmap(2,coL)=2←
LOCATE 2,coL:PRINT" ";:sc(2,coL)=pipe:Ln=2:pdir=1:di
r=1:GOTO PLot←
GetDir:←
dir=INT(RND*3):ck(dir)=ck(dir)+1←
IF ck(Ø)>1 AND ck(1)>1 AND ck(2)>1 THEN←
IF pno(row,coL)=pipe THEN<
LOCATE row,coL:COLOR ,Ø:PRINT" ";:COLOR ,2<
sc(row,coL)=Ø:pno(row,coL)=Ø:pipmap(row,coL)=Ø<
END IF<
im=im+1:IF im>1Ø THEN NewMap<
row=row-dy(pdir):coL=coL-dx(pdir)<
ck(pdir)=2:ck((pdir+1) MOD 2)=Ø:ck((pdir+2) MOD 2)=Ø
<
END IF<
IF sc(row+dy(dir),coL+dx(dir))=pipe OR ck(dir)>1 THE
N GetDir<
IF dir=Ø THEN GOSUB Left<
IF dir=1 THEN GOSUB Down<
IF dir=2 THEN GOSUB Right<
IF Ln>1 THEN ck(Ø)=Ø:ck(1)=Ø:ck(2)=Ø:IF Ln>2 THEN im
=Ø<
Ln=INT(RND*(Ln+1))<
PLot:                <
IF pdir<>dir AND Ln>1 THEN pipmap(row,coL)=dir+1:pdi
r=dir<
WHILE Ln>1<
row=row+dy(dir):coL=coL+dx(dir):LOCATE row,coL:PRINT
" ";<
sc(row,coL)=pipe:pno(row,coL)=pno(row,coL)+pipe<
Ln=Ln-1:WEND<
IF row<2Ø THEN GetDir<
IF pipe<npip THEN NewPipe<
GOSUB GetIntersect:IF ni<1 OR ni>3Ø THEN NewMap<
GOSUB MoveBaLLs:COLOR ,5<
IF numpL=Ø THEN<
pts=pts+ct-ms:IF pts<Ø THEN pts=Ø<
LOCATE 23,14:PRINT pts"   ":tot=tot+ct+ms:ctot=ctot+
ct<
LOCATE 23,32:PRINT INT(ctot/tot*1ØØ)"    "<
npip=INT(pts/1Ø)+3:IF npip>6 THEN EndGame<
ELSE<
LOCATE 23,cp(pn)*ofst+2:PRINT nam$(cp(pn))<
ppts(cp(pn))=ppts(cp(pn))+ct:LOCATE 24,cp(pn)*ofst+2
:PRINT ppts(cp(pn));<
IF pn=Ø THEN<
FOR i=numpL TO 1 STEP-1:FOR j=Ø TO i-1<
IF ppts(cp(j))<ppts(cp(j+1)) THEN SWAP cp(j),cp(j+1)
<
NEXT j,i<
IF ppts(cp(numpL))<ppts(cp(numpL-1)) THEN<
npip=npip+1:COLOR 1Ø,9:LOCATE 21,1<
PRINT SPACE$(4Ø):LOCATE 21,13:PRINT nam$(cp(numpL))"
 Eliminated"<
LOCATE 23,cp(numpL)*ofst+2:COLOR Ø,5:PRINT nam$(cp(n
umpL))<
FOR i=Ø TO numpL:ppts(cp(i))=Ø:NEXT:numpL=numpL-1:IF
 numpL=Ø THEN EndGame<
```

```
END IF<
pn=numpL<
ELSE<
pn=pn-1<
END IF<
END IF<
GOTO NewMap<
<
EndGame:<
FOR i=Ø TO 5ØØØ:NEXT:COLOR 1Ø,9:LOCATE 21,1:PRINT SP
ACE$(4Ø)<
LOCATE 21,5:PRINT"End of Game.  Play Again (Y/N)?"<
k$="":WHILE k$="":k$=UCASE$(INKEY$):WEND<
IF k$="Y" THEN CLS:GOTO NewGame<
IF k$="N" THEN GOSUB CLoseSprites<
GOTO EndGame<
<
Left:<
i=coL:Ln=Ø<
LeftCk:<
Ln=Ln+1<
IF sc(row+1,i)>Ø THEN IF sc(row+1,i-1)>Ø THEN RETURN
<
IF sc(row-1,i)>Ø THEN IF sc(row-1,i-1)>Ø THEN RETURN
<
i=i-1:IF i<2 THEN RETURN<
GOTO LeftCk<
<
Right:<
i=coL:Ln=Ø<
RightCk:<
Ln=Ln+1<
IF sc(row+1,i)>Ø THEN IF sc(row+1,i+1)>Ø THEN RETURN
<
IF sc(row-1,i)>Ø THEN IF sc(row-1,i+1)>Ø THEN RETURN
<
i=i+1:IF i>38 THEN RETURN<
GOTO RightCk<
<
Down:<
i=row:Ln=Ø<
DownCk:    <
Ln=Ln+1<
IF sc(i,coL+1)>Ø THEN IF sc(i+1,coL+1)>Ø THEN RETURN
<
IF sc(i,coL-1)>Ø THEN IF sc(i+1,coL-1)>Ø THEN RETURN
<
i=i+1:IF i>2Ø THEN RETURN<
GOTO DownCk<
<
GetIntersect:<
ni=Ø<
```

```
FOR row=3 TO 19:FOR coL=1 TO 39←
IF sc(row,coL)>Ø AND sc(row-1,coL)>Ø THEN←
sdn=ABS(sc(row+1,coL)>Ø):slt=ABS(sc(row,coL-1)>Ø):sr
t=ABS(sc(row,coL+1)>Ø)←
IF sdn+slt+srt>1 THEN←
xi(ni)=(coL-1)*8:yi(ni)=(row-1)*8←
IF sdn=Ø THEN←
ti(ni)=Ø:di(ni)=INT(RND*2)*2←
ELSEIF slt=Ø THEN←
ti(ni)=1:di(ni)=INT(RND*2)+1←
ELSEIF srt=Ø THEN←
ti(ni)=2:di(ni)=INT(RND*2)←
ELSE←
ti(ni)=3:di(ni)=INT(RND*3)←
END IF←
pipmap(row,coL)=di(ni)+1:PUT(xi(ni),yi(ni)),arrow(Ø,
di(ni)),PSET:ni=ni+1←
END IF←
END IF←
NEXT coL,row←
ni=ni-1:RETURN←
←
RotateArrows:←
FOR i=Ø TO ni←
ON ti(i) GOTO rot1,rot2,rot3←
di(i)=2-di(i):GOTO 1Ø←
rot1:di(i)=3-di(i):GOTO 1Ø←
rot2:di(i)=1-di(i):GOTO 1Ø←
rot3:di(i)=(di(i)+1) MOD 3←
1Ø pipmap(INT(yi(i)/8)+1,INT(xi(i)/8)+1)=di(i)+1←
PUT(xi(i),yi(i)),arrow(Ø,di(i)),PSET←
NEXT:RETURN←
←
MoveBaLLs:←
ct=Ø:ms=Ø:cx=156:cy=162:OBJECT.X 1,156:OBJECT.Y 1,16
2:OBJECT.ON 1:i=STRIG(2)←
FOR i=Ø TO npip-1:CALL MoveSprite&(Ø,sprite&+48*i,ba
llx(i)*8-5,baLLy(i)*8-13)←
NEXT:WHILE STRIG(2)=Ø:cx=cx+STICK(2)*8:OBJECT.X 1,cx
:FOR i=Ø TO 3ØØ:NEXT:WEND←
bd=Ø←
WHILE bd=Ø:bd=1←
FOR j=Ø TO npip-1←
IF baLLy(j)<21 THEN←
bd=Ø←
IF pipmap(baLLy(j),ballx(j))>Ø THEN balldir(j)=pipma
p(baLLy(j),ballx(j))-1←
ballx(j)=ballx(j)+dx(balldir(j)):baLLy(j)=baLLy(j)+d
y(balldir(j))←
CALL MoveSprite&(Ø,sprite&+48*j,ballx(j)*8-5,baLLy(j
)*8-5)←
```

```
ELSEIF baLLy(j)=21 THEN←
IF j=2 OR j=3 THEN←
IF ballx(j)-INT(cx/8)=4 THEN GOSUB Catch ELSE GOSUB
Miss:PUT((ballx(j)-1)*8,170),fbr,PSET←
ELSE←
IF ballx(j)-INT(cx/8)=2 THEN GOSUB Catch ELSE GOSUB
Miss:PUT((ballx(j)-1)*8,170),fbb,PSET←
END IF←
baLLy(j)=22←
END IF←
NEXT←
IF STRIG(2) THEN GOSUB RotateArrows←
cx=cx+STICK(2)*8:OBJECT.X 1,cx←
WEND←
OBJECT.OFF 1:LINE(8,170)-(304,171),5,bf←
RETURN←
←
Catch:←
SOUND 2000,.5:CALL MoveSprite&(0,sprite&+48*j,-16,0)
:ct=ct+1:RETURN←
←
Miss:←
SOUND 20,.5,255,3:ms=ms+1:CALL MoveSprite&(0,sprite&
+48*j,-16,0):RETURN←
←
LeftArrow:←
DATA 7,7,3,1800,2800,4E00,8200,4E00←
DATA 2800,1800,FE00,EE00,CF00,8300,CF00,EE00←
DATA FE00,0,1000,3100,7D00,3100,1000,0←
DATA 0,0←
←
DownArrow:←
DATA 7,7,3,3800,2800,EE00,8200,4400←
DATA 2800,1000,FE00,EE00,EF00,8300,C700,EE00←
DATA FE00,0,1000,1100,7D00,3900,1000,0←
DATA 0,0←
←
RightArrow:←
DATA 7,7,3,3000,2800,E400,8200,E400←
DATA 2800,3000,FE00,EE00,E700,8300,E700,EE00←
DATA FE00,0,1000,1900,7D00,1900,1000,0←
DATA 0,0←
←
CupData:←
DATA 0,0,0,0,0,0,0,0←
DATA 0,0,0,4,0,0,0,32←
DATA 0,0,0,11,0,24,0,15←
DATA 0,0←
DATA 15,128,15,128,50,64,50,64←
DATA 127,252,127,252,255,253,255,253←
DATA 55,240,55,240,136,13,104,13←
DATA Ø,248,64,248,64,114,96,114←
DATA 99,228,99,228,15,8,15,8←
DATA Ø,96,Ø,96←
DATA 15,24Ø,15,24Ø,48,76,48,76←
DATA 112,6,112,6,96,2,96,2←
DATA 24Ø,12,48,12,255,252,127,252←
DATA 127,248,127,248,127,24Ø,127,24Ø←
DATA 63,224,63,224,15,Ø,15,Ø←
DATA Ø,Ø,Ø,Ø←
DATA 15,24Ø,15,24Ø,48,76,63,188←
DATA 112,6,79,25Ø,1Ø2,178,223,255←
DATA 242,172,255,255,255,252,143,243←
DATA 127,248,128,6,127,24Ø,Ø,14←
DATA 63,224,64,28,15,Ø,16,248←
DATA Ø,Ø,7,224←
DATA 15,24Ø,15,24Ø,63,188,63,252←
DATA 79,25Ø,127,254,217,79,249,79←
DATA 61,95,253,95,15,243,255,255←
DATA 128,6,255,254,Ø,14,127,254←
DATA 64,28,127,252,16,248,31,248←
DATA 7,224,7,224←
←
InitSprites:←
sprite&=ALLocRaster&(16,144):IF sprite&=Ø THEN CLose
Sprites←
CALL BLtCLear&(sprite&,288,Ø)
     ←
FOR i=2 TO 7:offset=48*(i-2)←
IF GetSprite&(sprite&+offset,i)<>i THEN CLoseSprites
1←
POKEW sprite&+offset+4,7:POKEW sprite&+offset+6,-17←
RESTORE BaLLData:FOR j=16 TO 42 STEP 2:READ k:POKE s
prite&+offset+j,k:NEXT←
CALL ChangeSprite&(Ø,sprite&+offset,sprite&+offset+1
2)←
NEXT:RETURN←
←
BaLLData:←
DATA Ø,6Ø,Ø,126,3,255,1←
DATA 255,15,255,6,126,28,6Ø←
←
CLoseSprites:←
FOR i=2 TO 7:CALL FreeSprite&(i):NEXT←
CLoseSprites1:←
CALL FreeRaster&(sprite&,16,144)←
WINDOW CLOSE 3:SCREEN CLOSE 1:ON ERROR GOTO Ø:END←
←
```

# CHAPTER FOUR

# Family Fun and Educational Games

# Word War

Peter Crosby

*This simple but challenging word game pits you against a friend and is bound to provide hours of entertainment. The game requires 512K of RAM, Amiga Basic, and Kickstart version 1.2.*

"Word War" is a quick-paced word game that takes advantage of the Amiga's advanced features—its mouse, many colors, and excellent sound. The object is simple: Players guess letters and words to take over the Power Bar gauge located at the top of the screen. Whenever a player correctly guesses a letter or word, his or her color eats up more of the Power Bar. When one player's color overwhelms the other, the game is over. In essence, Word War combines the playing concepts of both hangman and tug-of-war.

## Typing It In

Word War is written in Amiga Basic. Type it in and save a copy to disk. When you're ready to play, load and run the program.

Enter the names of the two players. The game screen will appear. Take a moment to familiarize yourself with the screen. Near the top is the Power Bar. At the beginning of the game, the left half of the bar is red and the right is blue. As the players score points, the border between the colors moves to the left or to the right. Eventually, one player wins the game by taking over the entire bar.

Below the Power Bar are two boxes which hold the names of the players. Player 1 always plays red, while player 2 plays blue. During player 1's turn, the red box lights up. During player 2's turn, the blue name box is lit. Both players should watch these boxes to be sure that they do not play their opponent's turn.

The Action Board can be found directly below the names of the players. At the bottom of this board is the alphabet. A mystery word appears in the small box which appears to hang from the Board. At first the word is made up of black squares in the place of the mystery word letters. Players use the mouse to point at and click on the letter of the alphabet that they wish to guess. If the chosen letter can be found in the word, each occurrence of the letter lights up in the word, and the player gets another turn. In addition, the Power Bar moves slightly in favor of the player. If the letter cannot be found in the word, the other player gets a turn.

In the center of the board is a box labeled *I Know It.* A player who wants to take a guess at the mystery word can use the mouse to point at and click on this box. After selecting this box, the player types in his or her guess. If it's correct, the Power Bar changes to favor the player. If the guess is incorrect, the player's opponent gains points instead.

When one player needs to make a desperate recovery or wants to jump far ahead in score, he or she clicks on one of his or her three Revival Squares. (This can be done only during the player's turn.) These squares are located on the extreme left (for the red player) and the extreme right (for the blue player) of the Action Board. During the revival, a series of scrambled words appears at the top of the screen. The player must quickly unscramble these words and type them in. When time runs out, the game returns to the main screen, and the revival points are placed into the power bar. During the revival, the Delete key is inactive. To correct a typing error, use the space bar as a Delete key.

The accompanying listing has a vocabulary of 50 words. If you wish to use more, you must make a few changes to the program. The arrays Word$( ) and RevWord( ) must be dimensioned to the number of words in your word data. (The words are in DATA statements at the end of the program.) The 49 in the FOR X=0 TO 49 statement of the InitWords subroutine needs to be changed to the number of words you are using minus 1. The number 50 in the RND statements of lines 100 and 550 should also be changed to reflect the revised number of words. At the start of each game, BASIC reads words from the word data into the array word, checking for duplication as it proceeds. The greater the number of words used, the longer the delay at the start of a game.

### Word War

Filename: WORDWAR

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
' WordWar←
' Copyright 1987 COMPUTE! Publications, Inc.←
' All Rights Reserved.←
Header:←
  CLS:CLEAR←
  DIM waveform%(255),Word$(50),RevWord(50),PalCol(12
,3),UsedLetter(26),Jum$(12),jm(12),Used(12),Rguess$(
12)←
  GOSUB InitWords←
  GOSUB MenuSet  ←
  FOR n=0 TO 127←
  waveform%(n)=127:NEXT←
  WAVE 0,waveform%←
  SCREEN 2,320,200,5,1:WINDOW 2,"             Word
 War",(0,0)-(311,186),16,2←
  PRINT TAB(13)"Copyright 1987":PRINT TAB(7)"Compute
! Publications, Inc.":←
  PRINT TAB(11)"All Rights Reserved.":FOR t=1 TO 100
00:NEXT←
  GOSUB ReadColor←
  GOSUB RestColor←
  GOSUB Players ←
  LINE(30,20)-(282,32),31,bf:LINE(23,10)-(149,22),21
,bf:LINE(149,10)-(275,22),23,bf←
  LINE(22,9)-(276,23),30,b:LINE(21,8)-(277,24),20,b
 ←
  LINE(40,39)-(110,52),31,bf:LINE(207,39)-(278,52),3
1,bf←
  LINE(30,29)-(100,42),24,bf:LINE(29,28)-(101,43),20
,b←
  LINE(198,29)-(268,42),22,bf:LINE(197,28)-(269,43),
20,b←
  LOCATE 5,5:COLOR 30,21:PRINT Play$(1):COLOR 30,23:
LOCATE 5,26:PRINT Play$(2):COLOR 20,0  ←
  LINE(30,60)-(282,145),31,bf:LINE(23,50)-(275,135),
25,bf:LINE(23,50)-(275,135),20,b←
  LINE(35,136)-(40,150),28,bf:LINE(38,136)-(40,150),
2,bf:LINE(258,136)-(263,150),28,bf:LINE(261,136)-(26
3,150),2,bf←
  LINE(45,136)-(50,150),31,bf:LINE(268,136)-(273,150
),31,bf  ←
```

```
  LINE(3Ø,158)-(282,168),31,bf:LINE(23,15Ø)-(275,161
),29,bf:LINE(22,149)-(276,162),2Ø,b←
  COLOR 2Ø,Ø:LINE(99,81)-(199,1Ø3),27,b←
  LINE(98,8Ø)-(198,1Ø2),2,bf:COLOR 3Ø,2:LOCATE 12,15
:PRINT"I Know It"←
  FOR y= 66 TO 1Ø6 STEP 15←
  LINE(35,y+1)-(45,y+11),27,bf:LINE(33,y)-(43,y+1Ø),
24,bf←
  LINE(257,y+1)-(267,y+11),27,bf:LINE(255,y)-(265,y+
1Ø),22,bf←
  NEXT y←
  COLOR 2Ø,4←
  er$="          "←
  Marker=149←
  Pflag=2:GOSUB ChangePlay←
  WordNum=-1:BeepFlag=Ø←
  ←
SetUp:←
  COLOR 3Ø,Ø:WordNum=WordNum+1←
  LINE(45,11Ø)-(259,122),27,b←
  LINE(44,1Ø9)-(258,121),Ø,bf:LINE(44,1Ø9)-(258,121)
,2,b←
  FOR x=65 TO 9Ø:LOCATE 15,x-58:PRINT CHR$(x);:NEXT
   ←
  COLOR 2Ø,4←
  l=LEN(Word$(WordNum))←
  LetterLoc=INT((39-(l*2))/2)←
  FOR x=1 TO l←
    LOCATE 2Ø,LetterLoc+x*2:COLOR ,2+x:PRINT MID$(Wo
rd$(WordNum),x,1)←
  NEXT←
  high=3:NumRight=Ø:FOR x=1 TO 26:UsedLetter(x)=Ø:NE
XT←
←
MouseClick:←
  IF MOUSE (Ø)=Ø THEN MouseClick←
  Xletter=MOUSE(5)←
  Yletter=MOUSE(6)←
  IF Yletter>11Ø AND Yletter<121 AND Xletter>49 AND
Xletter<256 THEN GOSUB SelectLetter←
  IF Yletter>8Ø AND Yletter<1Ø2 AND Xletter>98 AND X
letter<198 THEN GOSUB GuessIt←
  IF Yletter>65 AND Yletter<12Ø AND Xletter>33 AND X
letter<43 THEN GOSUB Revive1←
  IF Yletter>65 AND Yletter<12Ø AND Xletter>255 AND
Xletter<265 THEN GOSUB Revive2←
  GOTO MouseClick←
   ←
GuessWord:←
  COLOR 2,Ø←
  Guess$=CHR$(Selection)←
  FOR g=1 TO 1←
    IF MID$(Word$(WordNum),g,1)=UCASE$(Guess$) THEN
high=g+2:GOSUB Highlight←
  NEXT g←
  IF BeepFlag=Ø THEN SOUND 8Ø,5,15Ø,Ø:GOSUB ChangePl
ay←
  BeepFlag=Ø←
  RETURN←
        ←
Highlight:←
  NumRight=NumRight+1←
  IF NumRight=1 THEN Winner←
  FOR x=5ØØ TO 1ØØØ STEP 5Ø←
  SOUND x,1,15Ø,1:NEXT    ←
  FOR x=.1 TO 1 STEP .1←
  PALETTE high,x,x,Ø:FOR y=Ø TO 25Ø:NEXT y,x←
  Score=3:GOSUB Score←
  BeepFlag=1←
  RETURN←
←
SelectLetter:←
   Selection=(INT(Xletter/8)-5)+64←
   IF UsedLetter(Selection-64)=1 THEN GOTO ClearMous
e←
   UsedLetter(Selection-64)=1←
   COLOR Ø,Ø:LOCATE 15,Selection-58:PRINT CHR$(Selec
tion)←
   GOSUB GuessWord←
   GOTO ClearMouse←
   ←
ClearMouse:←
  IF MOUSE(Ø)<>Ø THEN ClearMouse←
  RETURN←
←
GuessIt:←
  COLOR 2,25←
  a$=INKEY$:IF a$<>"" THEN GuessIt←
  IF a$=CHR$(13) THEN GuessIt←
  LOCATE 8,7:INPUT"Your Guess? ",gw$←
  IF UCASE$(gw$)=Word$(WordNum) THEN Score=(1-NumRig
ht)*5:GOTO Winner←
  FOR x=2ØØ TO 5Ø STEP -1Ø:SOUND x,1,15Ø,2:NEXT←
  LINE(24,51)-(274,65),25,bf←
  Score=(1-NumRight)*5:GuessWrong=1:GOSUB ChangePlay
←
  GOSUB ClearMouse←
  RETURN←
        ←
Winner:←
   s1=16ØØ:s2=1ØØØ:s3=8ØØ:s4=6ØØ:st=-1ØØ←
   FOR loop= 1 TO 4←
   FOR x=s1 TO s2 STEP st<
   SOUND x,1,150,1:NEXT<
   FOR x=s3 TO s4 STEP st<
   SOUND x,1,150,1:NEXT   <
   FOR high=3 TO 1+3<
   FOR x=.1 TO 1 STEP .1<
   PALETTE high,x,x,0:NEXT x<
   PALETTE high,.4,.4,0<
   NEXT high<
   s1=400:s2=600:s3=800:s4=1400:st=100<
   NEXT loop<
   GOSUB Score<
   COLOR 0,0:LINE(24,51)-(274,65),25,bf<
   LINE(23,150)-(275,161),29,bf   <
   GOSUB RestColor<
   GOSUB ClearMouse<
   BeepFlag=0<
   GOTO SetUp<
     <
RestColor:<
  FOR x=3 TO 16:PALETTE x,0,0,0:NEXT x:PALETTE 20,0,
0,0<
  RETURN<
  <
InitWords:<
  RESTORE WordData<
  RANDOMIZE TIMER<
  FOR x=0 TO 49<
100 y=(INT(RND*50)):IF Word$(y)<>"" THEN 100<
    READ Word$(y)<
    NEXT x<
    RETURN<
<
ReadColor:<
  PALETTE 0,0,.3,.3<
  PALETTE 1,0,.3,.3<
  RESTORE ColorData<
  FOR x=0 TO 11<
    FOR y=0 TO 2<
      READ PalCol(x,y)<
    NEXT y<
  NEXT x    <
  FOR x=0 TO 11 <
  PALETTE x+20,PalCol(x,0),PalCol(x,1),PalCol(x,2)<
  NEXT x<
  RETURN<
<
Players:<
  COLOR 2,21<
  LINE(34,61)-(284,76),31,bf:LINE(24,51)-(274,66),21
,bf:LINE(24,51)-(274,66),20,b<
  LOCATE 8,5:PRINT"Player #1: ";:COLOR 30,21:LINE IN
```

```
PUT"",Play$(1) ←
  COLOR 2,23←
  LINE(34,92)-(284,108),31,bf:LINE(24,82)-(274,98),2
3,bf:LINE(24,82)-(274,98),20,b←
  LOCATE 12,5:PRINT"Player #2: ";:COLOR 30,23:LINE I
NPUT"",Play$(2)←
  COLOR 20,0←
  FOR x=1 TO 2←
  IF LEN(Play$(x))>8 THEN Play$(x)=MID$(Play$(x),1,8
)←
  lp=LEN(Play$(x))←
  Play$(x)=SPACE$((8-lp)/2) + Play$(x)←
  NEXT←
  CLS←
  RETURN←
←
ChangePlay:←
  IF Pflag=1 THEN Switch1←
  IF Pflag=2 THEN Switch2←
  ←
Switch2:  ←
  PALETTE 22,0,0,.5:PALETTE 24,1,0,0←
  COLOR 30,24:LOCATE 5,5:PRINT Play$(1):COLOR 20,22:
LOCATE 5,26:PRINT Play$(2)←
  COLOR 20,0:Pflag=1←
  IF GuessWrong=1 THEN GOSUB Score←
  RETURN←
←
Switch1: ←
  PALETTE 22,0,0,1:PALETTE 24,.3,0,0←
  COLOR 20,24:LOCATE 5,5:PRINT Play$(1):COLOR 30,22:
LOCATE 5,26:PRINT Play$(2) ←
  COLOR 20,0:Pflag=2←
  IF GuessWrong=1 THEN GOSUB Score←
  RETURN←
←
Score:←
  IF Pflag=1 THEN GOSUB Play1 ELSE GOSUB Play2←
  Score=0:GuessWrong=0    ←
  RETURN←
←
Play2:  ←
  IF Marker-Score=<23 THEN ←
    LINE(23,10)-(275,22),23,bf←
    GOTO EndGame←
  END IF←
  Marker=Marker-Score:LINE(Marker,10)-(275,22),23,bf
←
  RETURN←
  ←
Play1:  ←
```

```
    IF Marker+Score=> 275 THEN ←
      LINE(23,10)-(275,22),21,bf←
      GOTO EndGame←
    END IF←
    Marker=Marker+Score:LINE(23,10)-(Marker,22),21,b
f  ←
    RETURN←
←
EndGame:←
  FOR x=1 TO 15←
  FOR y=0 TO 1 STEP .1←
  PALETTE 0,y,y,y:FOR z=1 TO 50:NEXT←
  NEXT←
  NEXT←
  PALETTE 0,0,.3,.3←
  LINE(23,10)-(275,22),20,bf:COLOR 30,29:LOCATE 20,9
:PRINT "  " Play$(Pflag) " has WON!!!    "←
200 LINE(24,51)-(274,65),25,bf←
    COLOR 20,25:LOCATE 8,7:INPUT"Another Game? ",An$
←
    IF UCASE$(An$)="N" THEN 300←
    IF UCASE$(An$)="Y" THEN GOSUB ClearMouse:GOTO He
ader←
    GOTO 200←
300 MENU RESET←
    WINDOW CLOSE 2←
    SCREEN CLOSE 2←
    SCREEN CLOSE 1←
    WINDOW CLOSE 1      ←
    END←
    ←
NewGame:←
    GOSUB ClearMouse←
    GOTO Header←
      ←
Revive1:←
  IF (Pflag=2) OR (Rev(1)>2) THEN RETURN←
  IF Yletter>65+(Rev(1)*15) AND Yletter<75+(Rev(1)*1
5) THEN 400←
  RETURN←
400 LINE(33,65+(Rev(1)*15))-(45,77+(Rev(1)*15)),25,b
f←
    COLOR 20,0:Rev(1)=Rev(1)+1←
    GOSUB ClearMouse←
    GOSUB ReviveMe←
    RETURN←
←
Revive2:←
  IF (Pflag=1) OR (Rev(2)>2) THEN RETURN←
  IF Yletter>65+(Rev(2)*15) AND Yletter<75+(Rev(2)*1
5) THEN 500←
  RETURN←
```

```
500 LINE(255,65+(Rev(2)*15))-(267,77+(Rev(2)*15)),25
,bf←
      COLOR 20,0:Rev(2)=Rav(2)+1←
      GOSUB ClearMouse←
      GOSUB ReviveMe←
      RETURN←
←
ReviveMe:←
   RANDOMIZE TIMER←
   FOR x=0 TO 49:RevWord(x)=0:NEXT←
   GOSUB ResetVal:Score=0:rw=1:pit=0←
   WINDOW 3,"",(0,0)-(311,186),16,2:IF Pflag=1 THEN C
ol=24 ELSE Col=22←
   LINE(0,0)-(311,186),20,bf:LINE(15,0)-(296,182),0,b
f←
   LINE(46,21)-(286,32),31,bf:LINE(39,14)-(279,25),29
,bf:LINE(38,13)-(280,26),20,b←
   FOR x=1 TO 5←
550 rn=INT(RND*50)←
   FOR y=1 TO 5:IF rn=Used(y) THEN 550←
   NEXT y←
   IF RevWord(rn)=1 THEN 550←
   Used(x)=rn:RevWord(rn)=1:Rev$(x)=Word$(rn)←
   NEXT x←
   GOSUB TimeBar←
   COLOR 20,0:LOCATE 13,4:PRINT "TIME":COLOR 30,0←
   LINE(83,122)-(225,174),31,bf:LINE(73,112)-(215,164
),27,bf:LINE(72,111)-(216,165),20,b←
   LINE(240,120)-(290,153),31,bf:LINE(230,110)-(280,1
43),Col,bf←
   LINE(229,110)-(281,144),20,b:LINE(229,120)-(281,12
0),20←
   LOCATE 15,30:COLOR 20,Col:PRINT "POINTS":COLOR 30,
0←
   GOTO RevLoop←
   ←
ResetVal:←
   ln=35:FOR x=1 TO 12:jm(x)=0:Used(x)=0:Jum$(x)="":N
EXT←
   Jumble$="":Unjum$="":ln=35←
560 c$=INKEY$←
      IF c$<>"" THEN 560←
      IF c$=CHR$(13) THEN 560←
      RETURN←
   ←
TimeBar:  ←
   LINE(40,40)-(50,92),31,bf:LINE(35,35)-(45,90),Col,
bf←
   RETURN←
←
RevLoop: ←
```

```
  lw=LEN(Rev$(rw)):LOCATE 1,10:PRINT "Press A Key To
 Begin.":b$=INKEY$        ←
  IF b$="" THEN RevLoop ELSE LINE(40,0)-(260,10),0,b
f:LINE(70,48)-(260,100),0,bf←
  FOR x=1 TO lw←
570 rn=INT(RND*lw+1):IF rn=0 THEN 570  ←
  FOR y=1 TO lw:IF rn=jm(y) THEN 570 ←
  NEXT y
        ←
  jm(x)=rn:Jum$(x)=MID$(Rev$(rw),rn,1)←
  NEXT x←
  FOR x=1 TO lw:Jumble$=Jumble$+Jum$(x)+" ":NEXT:Cen
t=INT((24-(lw*2))/2)←
  COLOR 30,29:LOCATE 3,9+Cent:PRINT Jumble$:COLOR 30
,0←
  FOR x=1 TO lw:s=70←
600 a$=INKEY$←
    IF a$=CHR$(32) THEN ←
       LOCATE 11,13+x:PRINT a$←
       x=x-1:IF x<1 THEN x=1←
       LOCATE 11,13+x←
       PRINT a$:a$="":GOTO 600←
    END IF   ←
    IF a$=CHR$(13) THEN a$="":GOTO 600←
    LOCATE 11,13+x:PRINT UCASE$(a$);:COLOR 25,0:PRIN
T "_":COLOR 30,0←
    IF a$="" THEN ←
      LINE(35,INT(ln/25)+32)-(45,INT(ln/25)+32),0:ln
=ln+1←
      LINE(46,INT(ln/25)+37)-(50,INT(ln/25)+37),0←
      IF ln>1460 THEN 660 ELSE 600←
    END IF←
    Rguess$(x)=a$:a$="":NEXT       ←
    FOR x= 1 TO lw:Unjum$=Unjum$+Rguess$(x):NEXT←
    IF UCASE$(Unjum$)=Rev$(rw) THEN 650 ELSE 670←
650 LOCATE 8,11:PRINT" That's Correct! ":FOR x=1000
TO 2500 STEP 100:SOUND x,1,150,1:NEXT:Score=Score+20
:Ucol=30:GOSUB PrintIt:GOTO 680←
660 LOCATE 8,11:PRINT"Sorry, Out Of Time!":SOUND 100
,5,250,1:Ucol=20:GOSUB PrintIt:GOTO 680←
670 LOCATE 8,11:PRINT"That Is Incorrect.":SOUND 100,
5,250,1:Ucol=20:GOSUB PrintIt:GOTO 680  ←
680 LOCATE 17,31:COLOR 30,Col:PRINT Score←
700 GOSUB ResetVal:COLOR 30,0:GOSUB TimeBar:rw=rw+1:
IF rw<=5 THEN LINE(39,14)-(279,25),29,bf:GOTO RevLoo
p←
 ←
EndLoop:←
  FOR x= 1 TO 3000:NEXT←
  CLS:LOCATE 7,3:PRINT "Total Number of Revival Poin
ts":LOCATE 9,16:PRINT Score←
```

```
  FOR x= 1 TO 5000:NEXT  ←
  WINDOW CLOSE 3←
  WINDOW OUTPUT 2←
  GOSUB Score←
  GOSUB ChangePlay←
  RETURN←
  ←
PrintIt:←
  LINE(39,14)-(279,25),29,bf←
  FOR pit=0 TO lw-1←
  COLOR 30,29:LOCATE 3,9+Cent+(pit*2):It$=MID$(Rev$(
rw),pit+1,1):It$=It$+" "←
  PRINT It$:NEXT              ←
  FOR j=1 TO 4000:NEXT←
  LOCATE 15+rw,13:COLOR Ucol,27:PRINT Rev$(rw)←
  pit=0←
  RETURN←
←
MenuSet:←
  MENU 1,0,1, " "←
  MENU 2,0,1, " "←
  MENU 3,0,1, " "←
  MENU 4,0,1, " "←
  RETURN←
←
ColorData:←
  DATA 0,0,0,1,0,0,.3,0,0,0,0,1,0,0,.3,1,.5,0,.7,0,1
,.5,.2,0,.6,.6,.6←
  DATA .7,.1,0,1,1,1,0,.1,0.02         ←
    ←
WordData:←
  DATA "FORTHRIGHT","CONSULTANT","REMEMBER","HIGHLIG
HT","AUDIBLE"←
  DATA "PERSONAL","ADVANCED","EXECUTE","VARIATION","
RESULTS"←
  DATA "FREQUENCY","EVALUATION","INDICATE","DURATION
","ARGUMENT"←
  DATA "GENERATE","PRECISE","INFORMED","SYSTEM","ENC
OUNTERED"←
  DATA "FOREIGN","CONSUMER","WASHTUB","SUMMER","FEAT
URE"←
  DATA "TELEPHONE","PROGRAM","MICROWAVE","TEMPERATUR
E","CONCLUDE"←
  DATA "RECOMMEND","MOTIVATION","ECONOMICS","CAPTAIN
","DOUBTLESS"←
  DATA "CONVENIENCE","MINIMIZE","TRANSMISSION","MECH
ANICAL","CONDITION"←
  DATA "ELECTRICAL","CHEMICAL","ENGINEER","EFFECTIVE
","HORIZONTAL"←
  DATA "CONCRETE","LANGUAGE","FUNCTION","STARTLE","K
NOWLEDGE"←
      ←
    ←
```

# Tiles

Rick Harrison
*Translation by Tim Midkiff*

*See if you can repeat the pattern of colored tiles with this challenging one- or two-player memory builder. "Tiles" is fun for kids and adults alike. The program requires 512K of RAM.*

"Tiles" is a game of concentration and observation. Putting your memory to the test, Tiles offers amusement for game buffs of all ages. To gain points, players must memorize and reproduce different patterns of colored tiles. The number of tiles increases as the game proceeds, making tile memorization more and more difficult.

As a one-player game, Tiles offers a stimulating challenge. With two players, Tiles becomes a competitive memory test as each player strives to attain a higher score than his or her opponent.

## Playing the Game

When the game begins, a colored tile is randomly placed in a 6 × 10 grid. The current player must memorize the position of the tile in as little time as possible. A player's score decreases by 20 points for approximately every second that the tile is displayed. The time penalty becomes 40 points per second if the player's score is between 2000 and 5000, and it changes to 60 points per second if the player's score is above 5000. Here, time is not money; it's points.

When you're finished viewing the tile pattern, the colored tiles disappear. You must relocate the positions of the tiles in the grid by moving the pointer. If you're right, you gain 100 points, but if you're wrong, you lose 100 points. There is no time limit or order in which hidden tiles must be found.

An extra tile is added to the pattern every round. The number of tiles that you must find is displayed on the screen. The game is over when your score drops to 0, or if you reach the objective, which is to successfully conquer a pattern of 30 tiles—no small task. A high score is kept for each player.

To view the random pattern of tiles, point to the words *Click to view* in the dialog box at the bottom of the screen and press the left mouse button. The tiles on the screen will change from the default blue color, and a random pattern of colored tiles will appear. After you click, the words *Click when ready* appear in the dialog box, and the clock starts. When you're done memorizing the tiles, click in the dialog box again, and the tiles return to the default blue color. Now you must click on the tiles that appeared. To select a tile, use the mouse to point to the desired tile and press the left mouse button. The current player's score box is highlighted in the two-player version.

At the end of the game you are asked if you wish to play again. If you do, high scores are retained and transferred to the next game.

## Tiles

Filename: TILES

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'Tiles ←
'Copyright 1988 ←
'COMPUTE! Publications, Inc.←
'All Rights Reserved.←
←
CLEAR ,25ØØØ:DEFINT a-z:RANDOMIZE TIMER←
DIM hit(59),nums(59),hrow(3Ø),hcoL(3Ø),pat(1)←
SCREEN 1,32Ø,2ØØ,4,1:WINDOW 3,,(Ø,Ø)-(311,186),16,1:
WINDOW OUTPUT 3←
PRINT "Copyright 1988":PRINT"COMPUTE! Publications,
Inc.":PRINT"All Rights Reserved"←
FOR tt=1 TO 3ØØØ:NEXT tt←
RESTORE Colors:FOR i=Ø TO 15:READ r,g,b:PALETTE i,r/
1Ø,g/1Ø,b/1Ø:NEXT←
Colors:←
DATA 1,Ø,3,6,6,6,2,2,6,7,7,Ø←
DATA 5,Ø,Ø,8,Ø,Ø,Ø,8,Ø,4,4,4←
DATA 8,8,Ø,8,Ø,8,Ø,8,8,5,Ø,Ø←
DATA Ø,5,Ø,5,5,Ø,5,Ø,5,Ø,5,5←
FOR i=Ø TO 59:nums(i)=i:NEXT:ON TIMER(1) GOSUB DecSc
ore←
FOR i=Ø TO 1Ø:ci(i)=i+5:NEXT:pat(1)=&HFFFF←
NewGame:←
```

```
FOR i=Ø TO 1:hitnum(i)=Ø:hnum(i)=Ø:score(i)=5ØØ:high
(i)=5ØØ:NEXT:GOSUB Board<
Start:<
pL=numpL-pL:IF score(pL)=Ø THEN Start<
IF numpL=1 THEN<
i=1-pL<
LINE (247,31+64*i)-(311,72+64*i),Ø,b<
LINE (249,33+64*i)-(3Ø9,7Ø+64*i),Ø,b<
LINE (247,31+64*pL)-(311,72+64*pL),7,b<
LINE (249,33+64*pL)-(3Ø9,7Ø+64*pL),7,b<
END IF <
LINE(17,173)-(227,185),4,bf:COLOR 3,4<
LOCATE 23,4:PRINT SPACE$(5)"Click to view"SPACE$(6):
k=Ø<
WHILE k<>3 AND k<>4:WHILE MOUSE(Ø)<1:WEND:k=POINT(MO
USE(1),MOUSE(2)):WEND<
GOSUB ShowTiLes:LINE(17,173)-(227,185),4,bf<
LOCATE 23,8:PRINT "Click when ready":COLOR 3,Ø:TIMER
 ON:k=Ø<
WHILE k<>3 AND k<>4 AND score(pL)>Ø<
IF MOUSE(Ø)>Ø THEN k=POINT(MOUSE(1),MOUSE(2))<
WEND<
TIMER OFF:LINE(17,173)-(227,185),Ø,bf<
GOSUB HideTiLes:LOCATE 23,8:PRINT SPACE$(4)"Tiles ="
hitnum(pL)+1 SPACE$(3)<
IF score(pL)>Ø THEN GOSUB DoSearch<
IF score(pL)>Ø THEN<
hnum(pL)=hnum(pL)+hitnum(pL):hitnum(pL)=hitnum(pL)+1
<
ELSE<
GOSUB UncoverTiLes<
IF score(numpL-pL)=Ø THEN<
GOTO Endgame<
ELSE<
LINE(17,173)-(227,185),4,bf:COLOR 3,4<
LOCATE 23,4:PRINT SPACE$(4)"Click to continue"SPACE$
(3):k=Ø<
WHILE k<>3 AND k<>4:WHILE MOUSE(Ø)<1:WEND:k=POINT(MO
USE(1),MOUSE(2)):WEND<
END IF <
END IF<
IF score(pL)>high(pL) THEN high(pL)=score(pL):p=1:GO
SUB PrintScore:p=Ø<
GOSUB HideTiLes:IF hitnum(pL)=3Ø THEN Endgame ELSE S
tart<
<
DecScore:<
dec(pL)=4Ø<
IF score(pL)<2ØØØ THEN dec(pL)=2Ø ELSE IF score>5ØØØ
 THEN dec(pL)=6Ø<
score(pL)=score(pL)-dec(pL):SOUND 9ØØØ,.5<
```

```
IF score(pL)<=Ø THEN score(pL)=Ø←
←
PrintScore:←
t$=STR$(score(pL)):LOCATE pL*8+p*2+6,34←
PRINT RIGHT$(SPACE$(4)+RIGHT$(t$,LEN(t$)-1),5):RETUR
N←
←
ShowTiLe:←
x=coL*24+6:y=row*24+16:LINE(x,y)-(x+16,y+16),ci(coL)
,bf:RETURN←
←
Square:←
x=coL*24+6:y=row*24+16:LINE(x,y)-(x+16,y+16),2,bf:RE
TURN←
←
Board:←
CLS:LINE(Ø,1Ø)-(245,159),1,b←
FOR row=Ø TO 5:FOR coL=Ø TO 9:GOSUB Square:NEXT coL,
row←
LINE(16,172)-(228,186),1,b←
LOCATE 23,4:COLOR 3:PRINT"Number of players (1/2)?"←
k$="":WHILE k$<>"1" AND k$<>"2":k$=INKEY$:WEND:numpL
=ASC(k$)-49←
LOCATE 4,32:PRINT"PLAYER 1":LINE (248,32)-(31Ø,71),1
,b←
IF numpL=1 THEN LOCATE 12,32:PRINT"PLAYER 2":LINE (2
48,96)-(31Ø,135),1,b←
FOR pL=Ø TO numpL:p=Ø:GOSUB PrintScore:p=1:GOSUB Pri
ntScore:NEXT←
pL=numpL:p=Ø←
RETURN←
←
DoSearch:←
hits=Ø←
GetMouse:WHILE MOUSE(Ø)<1:WEND←
x=MOUSE(3):y=MOUSE(4):IF POINT(x,y)<>2 THEN GetMouse
←
row=INT((y-16)/24):coL=INT((x-6)/24):tiLe=row*1Ø+coL
←
IF hit(tiLe)=1 THEN←
GOSUB ShowTiLe:score(pL)=score(pL)+1ØØ:hits=hits+1:h
it(tiLe)=2←
ELSE←
SOUND 9ØØØ,.5:score(pL)=score(pL)-1ØØ:IF score(pL)<=
Ø THEN score(pL)=Ø←
END IF←
GOSUB PrintScore:IF hits>hitnum(pL) OR score(pL)=Ø T
HEN RETURN ELSE GetMouse←
←
ShowTiLes:←
FOR i=Ø TO 59:hit(i)=Ø:NEXT←
FOR i=Ø TO 1Ø:r=INT(RND*11):t=ci(r):ci(r)=ci(i):ci(i
)=t:NEXT<
FOR i=Ø TO hitnum(pL):r=INT(RND*6Ø):t=nums(r):nums(r
)=nums(i):nums(i)=t:NEXT<
FOR i=Ø TO hitnum(pL):hit(nums(i))=1:hrow(i)=INT(num
s(i)/1Ø)<
hcoL(i)=nums(i) MOD 1Ø:row=hrow(i):coL=hcoL(i):GOSUB
 ShowTiLe:NEXT<
RETURN<
<
UncoverTiLes:<
FOR i=Ø TO hitnum(pL):tiLe=nums(i)<
IF hit(tiLe)=1 THEN<
pat(Ø)=&HAAAA:pat(1)=&H5555:PATTERN ,pat<
PAINT (hcoL(i)*24+6,hrow(i)*24+16),ci(hcoL(i)),Ø<
pat(Ø)=&HFFFF:pat(1)=&HFFFF:PATTERN ,pat<
END IF<
NEXT:RETURN<
<
HideTiLes:<
FOR i=Ø TO hitnum(pL):row=hrow(i):coL=hcoL(i):GOSUB
Square:NEXT:RETURN<
<
Endgame:<
LOCATE 23,7:PRINT"Play Again (Y/N)?"<
k$="":WHILE k$<>"Y" AND k$<>"N":k$=UCASE$(INKEY$):WE
ND<
IF k$="Y" THEN GOTO NewGame ELSE WINDOW CLOSE 3:SCRE
EN CLOSE 1:END<
<
```

# Jigsaw

Walter Bulawa

*This short, elegant program is not only an entertaining activity for the whole family, but also a demonstration of valuable techniques for programming graphics in Amiga Basic. The author discusses how he created the program.*

"Jigsaw" is a simple, but absorbing, BASIC game for the Amiga. The program requires you to put together a puzzle after its pieces have been scattered around the screen. The Amiga keeps track of the number of moves you make and the total amount of time you take to complete the puzzle.

## Unshuffle the Pieces

Type in the program and save a copy to disk before you run it. The program begins by drawing a puzzle shape in a small window in the center of the screen. Wait until you see a shape that you like and then press the space bar. The Amiga then divides the picture into a number of equally sized pieces, capturing each piece in a small square on the screen. While this is being done, you should take advantage of the opportunity to memorize the puzzle's shape. After every piece has been captured, the computer shuffles them at random. Begin playing when the center of the screen is cleared.

Your goal is to reconstruct the picture by placing each piece in its original position, using the mouse to move pieces. To pick up a piece, move the mouse pointer over the piece; then press the left button and hold it down. The piece blinks briefly, and the computer emits a beep to indicate that you have the piece. Continue to hold down the mouse button as you move the piece to its destination. When you have positioned the piece, release the button. The square blinks a second time to signal that it has been placed. Continue this process until the entire picture is constructed. When you solve the puzzle, the program lets you play again or quit.

The bottom of the screen contains two counters: a timer that updates continuously and a move counter that shows how many turns you have taken. To increase the game's difficulty and add to its visual appeal, the computer also continuously rotates the palette colors of the puzzle pieces.

You must place each piece reasonably close to the desired destination square, but you need not line it up exactly. If the piece is close enough for the computer to tell which location you intend, the program automatically "snaps" it into perfect alignment.

If you find yourself stumped, you can peek at the original puzzle for a moment and then return to the puzzle screen. This is done with the back-window and front-window gadgets located at upper left of the window border. To peek at the original, unscrambled puzzle, click the left button once on the back-window gadget. To return to the puzzle screen, click the left button once on the front-window gadget. There is no penalty for peeking. However, keep in mind that the timer continues to tick while you study the original shape.

Beginning puzzlers should avoid puzzles that include large areas of blank space. Blank squares may look identical to you, but the computer remembers the original location of each piece and won't end the game until you place each one in the correct spot. Thus, a puzzle that contains mostly blank space can be nearly impossible to solve.

## Bobs and OBJECT

Programmers may wish to study the way that this program moves and places graphic shapes on the screen. One technique that might have been used is to GET each shape into a variable and PUT it on the screen wherever desired. But PUT and GET create slow, flickery animation in BASIC. Instead of PUT and GET, this program makes each puzzle piece into a *bob* and animates it with OBJECT commands. The result is much smoother animation. You can still notice slight jerkiness in the piece's motion when you carry it with the pointer, but this is due to delays created by background routines activated by ON TIMER.

The process of creating a bob involves several steps. First, GET is used to capture all the graphic data for each shape in an integer array. This integer array is then converted into a string array. The string array, in turn, is concatenated into a general string array that holds the bob's features and is used to animate the bob with OBJECT.DRAW commands.

## Jigsaw

Filename: JIGSAW

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.←
'  All Rights Reserved←
←
DEFINT a-z←
DEFSNG coLrs←
DEF FNXYfmRC(cr,w)=(cr-1)*w←
DEF FNRCfmXY(xy,w)=INT((xy+w)/w)←
faLse=Ø:true=-1←
ncoLs.pzL=5:nrows.pzL=4←
xmin.pzL=Ø:xmax.pzL=149:ymin.pzL=Ø:Move.piece=99←
xwidth=(xmax.pzL-xmin.pzL+1)/ncoLs.pzL←
ywidth=(Move.piece-ymin.pzL+1)/nrows.pzL←
getsize=3+INT((16+xwidth-1)/16)*ywidth*5←
rmin=1:rmax=7:cmin=1:cmax=1Ø←
xmin=FNXYfmRC(cmin,xwidth)←
xmax=FNXYfmRC(cmax,xwidth)←
ymin=FNXYfmRC(rmin,ywidth)←
ymax=FNXYfmRC(rmax,ywidth)←
ncoLrs=9:coLrmin=6:coLrmax=coLrmin+ncoLrs-1←
vmin=1:vmax=3←
←
DIM coLrs(ncoLrs,3),a(getsize,1),b(getsize)←
DIM pcoL(ncoLs.pzL-1,nrows.pzL-1),prow(ncoLs.pzL-1,n
rows.pzL-1)←
DIM coLs(ncoLs.pzL*nrows.pzL-1),rows(ncoLs.pzL*nrows
.pzL-1)←
DIM s$(ncoLs.pzL-1,nrows.pzL-1)←
←
PALETTE Ø,Ø,.3,.6←
PALETTE 1,1,1,1←
←
d=5←
s1$=STRING$(26,Ø)←
POKE SADD(s1$)+11,d←
POKE SADD(s1$)+15,xwidth←
POKE SADD(s1$)+19,ywidth←
POKE SADD(s1$)+21,24←
POKE SADD(s1$)+23,2^d-1←
←
RESTORE Nu.CoLors←
FOR i=Ø TO ncoLrs   'Get new palette coLors from DAT
A  ←
  FOR j=1 TO 3←
  READ coLrs(i,j)←
NEXT j,i←
←
RESTORE CoLs.Rows←
FOR i=Ø TO ncoLs.pzL*nrows.pzL-1←
  READ coLs(i):READ rows(i)←
NEXT←
    ←
SCREEN 1,32Ø,2ØØ,d,1←
WINDOW 2,"Jigsaw",,28,1←
←
Restart:←
CLS:RANDOMIZE TIMER:moves=Ø←
←
coLr.index=ncoLrs:GOSUB CoLr.Shift←
←
p$="Press space bar to stop puzzle":LOCATE 23,2Ø-INT
(LEN(p$)/2):PRINT p$;←
WINDOW 3,"Jigsaw",(8Ø,7Ø)-(229,169),16,1←
PAINT (1Ø,1Ø),2←
GOSUB Make.PuzzLe←
WINDOW OUTPUT 2←
LOCATE 23,2Ø-INT(LEN(p$)/2):PRINT STRING$(LEN(p$),"
");←
←
'Make Bob strings and place pieces on the screen←
cLast=ncoLs.pzL-1:rLast=nrows.pzL-1←
FOR irow=Ø TO rLast ←
  FOR icoL=Ø TO cLast←
    WINDOW OUTPUT 3←
    x=FNXYfmRC(icoL+1,xwidth):y=FNXYfmRC(irow+1,ywid
th)←
    GET (x,y)-(x+xwidth-1,y+ywidth-1),a(Ø,Ø)←
    s$(icoL,irow)=""←
    iLast=getsize-1←
    FOR i=3 TO iLast:s$(icoL,irow)=s$(icoL,irow)+MKI
$(a(i,Ø)):NEXT←
    WINDOW OUTPUT 2←
    i=icoL+ncoLs.pzL*irow←
    x=FNXYfmRC(coLs(i),xwidth):y=FNXYfmRC(rows(i),yw
idth)←
    PUT (x,y),a(Ø,Ø)←
    pcoL(icoL,irow)=coLs(i):prow(icoL,irow)=rows(i)←
```

```
NEXT:NEXT←
WINDOW 2←
←
' Shuffle the pieces←
FOR i=Ø TO 2Ø←
Pick.RC:←
  FOR j=1 TO 2←
    coL(j)=INT(ncoLs.pzL*RND):row(j)=INT(nrows.pzL*R
ND)←
  NEXT j←
  IF coL(1)=coL(2) AND row(1)=row(2) THEN GOTO Pick.
RC←
  FOR j=1 TO 2 ←
    x(j)=FNXYfmRC(pcoL(coL(j),row(j)),xwidth)←
    y(j)=FNXYfmRC(prow(coL(j),row(j)),ywidth)←
    GET (x(j),y(j))-(x(j)+xwidth-1,y(j)+ywidth-1),a(
Ø,j-1)←
    LINE (x(j),y(j))-(x(j)+xwidth-1,y(j)+ywidth-1),Ø
,bf←
  NEXT j←
  PUT (x(1),y(1)),a(Ø,1):PUT (x(2),y(2)),a(Ø,Ø)←
  SWAP pcoL(coL(1),row(1)),pcoL(coL(2),row(2))←
  SWAP prow(coL(1),row(1)),prow(coL(2),row(2))←
NEXT    ←
'←
' Main loop←
'←
t!=TIMER:ON TIMER (1) GOSUB Show.Time:TIMER ON←
done=faLse:seLection.made=faLse←
GOSUB Beap←
WHILE NOT done←
IF MOUSE(Ø)=-1 THEN←
SeLect.Piece:←
    x=MOUSE(5):y=MOUSE(6) 'get x & y of mouse←
    GOSUB Fit2Scn 'see if on screen←
    coL=FNRCfmXY(x,xwidth):row=FNRCfmXY(y,ywidth)←
    GOSUB WhatsThere←
    IF piece THEN ←
      coL.piece=cp:row.piece=rp←
      pcoL(coL.piece,row.piece)=-1←
      prow(coL.piece,row.piece)=-1←
      GOSUB Beap ←
      xp=FNXYfmRC(coL,xwidth):yp=FNXYfmRC(row,ywidth
)←

      xdif=xp-x:ydif=yp-y←
      GET (xp,yp)-(xp+xwidth-1,yp+ywidth-1),a(Ø,Ø)←
      LINE (xp,yp)-(xp+xwidth-1,yp+ywidth-1),Ø,bf←
      OBJECT.SHAPE 1,s1$+s$(coL.piece,row.piece)←
      OBJECT.X 1,xp:OBJECT.Y 1,yp←
      OBJECT.ON 1←
      seLection.made=true←
    END IF←
END IF '(mouse)←
WHILE seLection.made←
  WHILE MOUSE(Ø)=-1←
  x=MOUSE(5):y=MOUSE(6)←
  GOSUB Fit2Scn←
  IF x<>xp-xdif OR y<>yp-ydif THEN←
    xp=x+xdif:yp=y+ydif←
    OBJECT.X 1,xp:OBJECT.Y 1,yp←
   END IF←
  WEND←
  ←
  GOSUB Fit2Scn←
  coL=FNRCfmXY(x,xwidth)←
  row=FNRCfmXY(y,ywidth)←
  GOSUB WhatsThere←
  IF NOT piece THEN←
    x=FNXYfmRC(coL,xwidth)←
    y=FNXYfmRC(row,ywidth)←
    OBJECT.OFF 1:PUT (x,y),a(Ø,Ø)←
    seLection.made=faLse←
    pcoL(coL.piece,row.piece)=coL←
    prow(coL.piece,row.piece)=row←
    GOSUB Beap←
    moves=moves+1:LOCATE 23,13:PRINT "Moves:";moves;
←
    rØ=prow(Ø,Ø):cØ=pcoL(Ø,Ø):count=Ø←
    FOR r=Ø TO nrows.pzL-1←
      FOR c=Ø TO ncoLs.pzL-1←
      IF (prow(c,r)-rØ)=r THEN←
        IF (pcoL(c,r)-cØ)=c THEN count=count+1←
      END IF←
    NEXT c,r←
    IF count=nrows.pzL*ncoLs.pzL THEN done=true←
  END IF '(not piece)←
WEND '(seLection)←
WEND '(done)←
TIMER OFF←
FOR i=Ø TO 1Ø:GOSUB Beap:NEXT←
p$="Again (Y/N)?"←
COLOR 1,Ø:LOCATE 23,25:PRINT p$;←
p$="":FOR i=Ø TO 1ØØØ:NEXT:WHILE p$="":p$=INKEY$:WEN
D←
IF p$="y" OR p$="Y" THEN GOTO Restart←
SCREEN CLOSE 1←
END←
←
Beap:←
  SOUND 8ØØ,1,1ØØ,Ø:SOUND 1ØØØ,1,1ØØ,Ø←
  RETURN←
  ←
```

```
Fit2Scn:←
  IF x<xmin THEN x=xmin←
  IF x>xmax THEN x=xmax←
  IF y<ymin THEN y=ymin←
  IF y>ymax THEN y=ymax←
  RETURN←
←
WhatsThere:←
  piece=false:cLast=ncols.pzL-1:rLast=nrows.pzL-1←
  FOR c=Ø TO cLast←
    FOR r=Ø TO rLast←
    IF pcoL(c,r)=coL THEN←
      IF prow(c,r)=row THEN piece=true:cp=c:rp=r:RET
URN←
    END IF←
  NEXT:NEXT←
  RETURN←
←
CoLr.Shift:←
  coLr.index=(coLr.index+1) MOD ncoLrs←
  FOR j=Ø TO ncoLrs-1←
    i=(coLr.index+j) MOD ncoLrs←
    PALETTE j+6,coLrs(i,1),coLrs(i,2),coLrs(i,3)←
  NEXT←
  RETURN←
←
Make.PuzzLe:←
FOR i=Ø TO 1←
x(i)=xmax.pzL*RND:y(i)=Move.piece*RND←
v:←
vx(i)=2*vmax*RND-vmax:vy(i)=2*vmax*RND-vmax←
IF vx(i)=Ø OR vy(i)=Ø THEN GOTO v←
NEXT←
coLr=coLrmin←
WHILE INKEY$=""←
FOR i=Ø TO 1←
  x(i)=x(i)+vx(i)←
  y(i)=y(i)+vy(i)←
  IF x(i)<=xmin.pzL OR x(i)>=xmax.pzL THEN←
    vx(i)=-SGN(vx(i))*(RND(vmax)+vmin)←
  END IF←
  IF y(i)<=ymin.pzL OR y(i)>=Move.piece THEN←
    vy(i)=-SGN(vy(i))*(RND(vmax)+vmin)←
  END IF←
NEXT←
coLr=coLr+1:IF coLr>coLrmax THEN coLr=coLrmin←
LINE (x(Ø),y(Ø))-(x(1),y(1)),coLr←
WEND   ←
RETURN←
←
Show.Time:←
```

```
  T2!=TIMER←
  LOCATE 23,1:PRINT "Time:";CINT(T2!-t!);←
  GOSUB CoLr.Shift←
RETURN←
  ←
  Nu.CoLors:←
  DATA .99,.Ø5,.Ø3←
  DATA .99,.7Ø,.Ø3←
  DATA .59,.99,.Ø3←
  DATA .Ø3,.99,.11←
  DATA .Ø3,.99,.81←
  DATA .Ø3,.51,.99←
  DATA .22,.Ø3,.99←
  DATA .89,.Ø3,.99←
  DATA .99,.Ø3,.4Ø←
  ←
  CoLs.Rows:←
  DATA 1,1,2,2,1,3,2,4,1,5,2,6,1,7←
  DATA 9,1,1Ø,2,9,3,1Ø,4,9,5,1Ø,6,9,7←
  DATA 3,1,4,2,5,1,6,2,7,1,8,2←
              ←
```

# CHAPTER FIVE

# Applications

# Banner Printer

Walter Bulawa

*Here's a banner-printing program with an interesting feature. In addition to the usual Amiga characters, you can use any of the Amiga's disk-based custom character fonts. A dot-matrix or laser printer is required.*

This Amiga Basic program allows you to construct and print a banner of enlarged letters using any of the 13 fonts present on the Workbench disk. You can use any combination of fonts on the same banner.

Program 1, the banner printing program, requires that two special files called *graphics.bmap* and *diskfont.bmap* be present. Users of Workbench 1.2 should follow the procedure outlined in Appendix B. Users of Workbench 1.1 should type COPY Extras:BasicDemos/graphics.bmap TO Libs: at the CLI prompt. To create diskfont.bmap, users of Workbench 1.1 can run Program 2. (This program is taken from *Advanced Amiga BASIC* by Tom Halfhill and Charles Brannon, available from COMPUTE! Books.) Once you've used Program 2 to create *diskfont.bmap,* you won't need Program 2 again except to create additional copies of that file.

## Banner Construction

When you run Program 1, it opens a window where you can construct a banner. The white area near the bottom of the window represents the printer paper, with the left edge of the display corresponding to the top edge of the printer paper. The small vertical line is the cursor.

Letters that you type on the keyboard appear in the work area with the current character font. You can move the cursor to any position in the white work area by dragging it with the mouse pointer. The Mouse menu allows you to use the mouse for two other purposes as well: drawing and erasing pixels in the work area. This facility lets you add graphics to text or erase text that you wish to eliminate.

The upper portion of the window indicates which font is currently in use. To change fonts, simply choose the desired font from the Font Selection menu. Except for the Topaz fonts, which are contained in ROM, new fonts will be loaded from the Workbench disk. Once the program has found the font, it identifies and displays the font on the screen. You may then type in the work area with that font.

It is important to remember that the white area represents the banner as it will be produced on the printer. So, should you wish larger letters for the banner, use the Box Height menu to select a narrower height for the white area. The more you shrink the work area, the larger the characters appear on the paper. Changing the work area's height always erases the work area completely.

This program ordinarily uses the X character to form the banner characters. However, you can select a different printing character with the Change Printer Char option of the Action menu. Simply type the new character when prompted.

## Printing

Once you have finished writing on the work area, choose the Print Banner option from the Action menu to print the banner. Make sure that the printer is connected and turned on before you take this action. It's also important that you use the correct printer driver for your printer. To check or change the printer driver, click on the Preferences tool from the Workbench and choose the Change Printer option.

You can abort the printing at any time by pressing the ESC key. Printing always begins at the left margin of the work area; to avoid wasting paper, it's usually best to locate the first character close to the left margin. If your banner message doesn't completely fill the text window, the printer will print blank lines representing the unused portion. To avoid wasting paper, you may want to press ESC to halt printing as soon as all of your message has been printed.

When the banner is printed, the characters tend to look somewhat stretched compared to their appearance on the screen. The Printer lines/display column option allows you to correct for the stretching effect, depending on what height is selected under the Box Height menu. Good results can often be obtained by using a value about half as large as the default value.

**Program 1. Banner Printer**
Filename: BANNER

*For instructions on entering these programs, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.<
'  All Rights Reserved<
<
'<
DEFINT a-z<
LIBRARY "graphics.library"<
LIBRARY "diskfont.library"<
DECLARE FUNCTION OpenDiskFont&() LIBRARY<
DECLARE FUNCTION OpenFont&() LIBRARY<
DIM TextAttr&(1),FontName$(13),FontSize(13)<
DIM choice(4)<
xmin=Ø:xmax=615:ymin=1Ø7:ymax=184:LpC%=1:CpL%=1<
false=Ø:true=-1:kwit=false<
pen=1:choice(2)=1:choice(3)=4<
xe=7Ø:ye=2Ø<
PChar$="X"<
<
MENU 1,Ø,1,"Action"<
MENU 1,1,1,"  Print Banner"<
MENU 1,2,1,"  Erase Box"<
MENU 1,3,1,"  Change Printer Char"<
MENU 1,4,1,"  Change Ptr lines/col"<
MENU 1,5,1,"  Quit"<
<
MENU 2,Ø,1,"Mouse"<
MENU 2,1,2,"  Draw"<
MENU 2,2,1,"  Erase"<
MENU 2,3,1,"  Position cursor"<
<
MENU 3,Ø,1,"Box Height"<
MENU 3,1,1,"  1Ø pixels"<
MENU 3,2,1,"  2Ø pixels"<
MENU 3,3,1,"  4Ø pixels"<
MENU 3,4,2,"  8Ø pixels"<
```

```
←
MENU 4,Ø,1,"Font Selection"←
FOR i%=1 TO 13←
  READ FontName$(i%),FontSize(i%)←
  MENU 4,i%,1,"  "+FontName$(i%)+STR$(FontSize(i%))←
NEXT i%  ←
←
'Set-up the Screen Display←
COLOR 3,Ø←
LOCATE 3,1:PRINT "Font:"←
LOCATE 7,1:PRINT "Printer character:"←
PRINT "Printer lines/display column:"←
COLOR 1,Ø←
LOCATE 7,2Ø:PRINT PChar$←
LOCATE 8,3Ø:PRINT LpC%←
'Get Info on current font←
Rp&=WINDOW(8)←
CALL AskFont&(Rp&,VARPTR(TextAttr&(Ø)))←
  FontSize=TextAttr&(1)\65536&←
  FontName$="topaz"←
  Font.Default&=OpenFont&(VARPTR(TextAttr&(Ø)))←
  Font.Active&=Font.Default&←
  BaseLine=PEEKW(Rp&+62)←
GOSUB Show.Example←
GOSUB Erase.Box 'Display Banner Box←
←
'Main Loop←
ON MOUSE GOSUB Mouse.Action:MOUSE ON←
ON MENU GOSUB Menu.Request:MENU ON←
←
WHILE NOT kwit←
  c$=INKEY$←
  IF c$<>"" THEN←
    IF ASC(c$)=13 THEN←
      GOSUB Erase.Cursor←
      xc=xmin:yc=yc+FontSize+3←
      GOSUB Yc.Check←
      GOSUB Move.Cursor←
    ELSE←
      GOSUB Erase.Cursor←
      COLOR 2,1←
      PRINT RIGHT$(c$,1);←
      xc=PEEKW(Rp&+36):yc=PEEKW(Rp&+38)←
      GOSUB Show.Cursor 'display new cursor←
    END IF←
  END IF  ←
WEND←
←
Done:←
  COLOR 1,Ø←
  MENU RESET←
  CALL CloseFont&(Rp&,Font.Active&)←
  CALL SetFont&(Rp&,Font.Default&)←
  LIBRARY CLOSE←
  END←
←
Mouse.Action:←
  WHILE MOUSE(Ø)<>Ø←
  mx=MOUSE(1):my=MOUSE(2)←
  IF mx<xmin THEN mx=xmin←
  IF mx>xmax THEN mx=xmax←
  IF cursor.mode THEN           'Move cursor w/mouse←
    GOSUB Erase.Cursor←
    yc=my:xc=mx←
    GOSUB Yc.Check←
    GOSUB Move.Cursor←
  ELSE                          'Draw w/mouse  ←
    IF my<ymin THEN my=ymin←
    IF my>ymax THEN my=ymax←
    PSET (mx,my),pen←
  END IF←
  WEND←
RETURN←
←
Menu.Request:←
mnu=MENU(Ø):item=MENU(1)←
IF choice(mnu)<>Ø THEN MENU mnu,choice(mnu),1←
choice(mnu)=item←
MENU mnu,choice(mnu),2 ←
ON mnu GOSUB Menu.1,Menu.2,Menu.3,Menu.4←
c$=""←
RETURN←
←
Menu.1:←
  ON item GOSUB Print.it,Erase.Box,Choose.PChar,Choo
se.LpC,Quit←
RETURN←
←
Menu.2:←
  IF item=1 THEN pen=2←
  IF item=2 THEN pen=1←
  cursor.mode=false←
  IF item=3 THEN cursor.mode=true←
  RETURN←
←
Menu.3:←
  LINE (xmin,ymin)-(xmax,ymax),Ø,bf 'erase old box←
  BoxHeight%=1Ø*2^(item-1)←
  ymax=ymin+BoxHeight%-1←
  GOSUB Erase.Box  'display new box←
  LpC%=8Ø/BoxHeight%:CpL%=LpC%←
  CALL SetFont&(Rp&,Font.Default&)←
  COLOR 1,Ø:LOCATE 8,3Ø:PRINT LpC%←
  CALL SetFont&(Rp&,Font.Active&)←
  GOSUB Move.Cursor←
RETURN←
←
Menu.4:←
  F=Ø←
  TextAttr&(Ø)=SADD(FontName$(item)+".font"+CHR$(Ø))
←
  TextAttr&(1)=FontSize(item)*65536&←
  IF item<3 THEN←
    F&=OpenFont&(VARPTR(TextAttr&(Ø))) 'ROM fonts←
  ELSE←
    F&=OpenDiskFont&(VARPTR(TextAttr&(Ø))) 'Disk fon
ts←
  END IF←
  IF F&=Ø THEN RETURN←
  GOSUB Erase.Example:GOSUB Erase.Cursor←
  IF Font.Active&<>Ø AND Font.Active&<>Font.Default&
 THEN CALL CloseFont&(Rp&,Font.Active&)←
  Font.Active&=F&←
  FontSize=FontSize(item)←
  FontName$=FontName$(item)←
  CALL SetFont&(Rp&,Font.Active&)←
  BaseLine=PEEKW(Rp&+62)←
  GOSUB Show.Example←
  GOSUB Yc.Check←
  GOSUB Move.Cursor←
RETURN←
  ←
Choose.PChar:←
  CALL SetFont&(Rp&,Font.Default&)←
  COLOR 1,Ø:LOCATE 1Ø,1:PRINT "Enter new printer cha
racter: _"←
  c$="":WHILE c$="":c$=INKEY$:WEND←
  IF ASC(c$)>32 THEN←
    PChar$=c$←
    LOCATE 7,2Ø:PRINT PChar$←
  END IF←
  LOCATE 1Ø,1:PRINT STRING$(3Ø,32)←
  CALL SetFont&(Rp&,Font.Active&)←
  GOSUB Move.Cursor←
  GOSUB Delay←
  RETURN←
←
Choose.LpC:←
  CALL SetFont&(Rp&,Font.Default&)←
  COLOR 1,Ø:LOCATE 1Ø,1:PRINT "Enter lines/col: _"←
  c$="":WHILE c$="":c$=INKEY$:WEND←
  IF VAL(c$)<LpC% AND VAL(c$)>Ø THEN←
    LpC%=VAL(c$)←
    LOCATE 8,30:PRINT LpC%←
  END IF←
  LOCATE 10,1:PRINT STRING$(30,32)←
  CALL SetFont&(Rp&,Font.Active&)←
  GOSUB Move.Cursor←
  GOSUB Delay←
  RETURN←
    ←
Print.it:←
 Prt.Stop=false←
 OPEN "PRT:" FOR OUTPUT AS #1←
 PRINT #1,CHR$(27);"[0z";CHR$(27);"#3";←
 CALL SetFont&(Rp&,Font.Default&)←
 COLOR 1,0:LOCATE 10,1:PRINT"Press ESC to Abort Prin
t"←
 FOR x=xmin TO xmax←
    p$=""←
    IF INKEY$=CHR$(27) THEN GOTO Print.Done←
    FOR y=ymax TO ymin STEP -1←
      IF POINT(x,y)<>1 THEN←
       c$=PChar$←
      ELSE←
       c$=" "←
      END IF←
      FOR i=1 TO CpL%:p$=p$+c$:NEXT←
    NEXT y←
    FOR i=1 TO LpC%←
    PRINT #1,p$←
    NEXT i←
  NEXT x          ←
Print.Done:←
  CLOSE #1←
  LOCATE 10,1:PRINT STRING$(30," ")←
  CALL SetFont&(Rp&,Font.Active&)←
  GOSUB Move.Cursor←
  RETURN←
←
Erase.Box:←
  LINE (xmin,ymin)-(xmax,ymax),1,bf←
  xc=xmin:yc=ymin+BaseLine←
  GOSUB Move.Cursor←
  RETURN←
  ←
Quit:←
  kwit=true←
  RETURN←
←
Move.Cursor:←
  CALL move&(Rp&,xc,yc)←
Show.Cursor:←
  ytemp=yc-BaseLine←
  LINE (xc,ytemp)-(xc,ytemp+FontSize-1),3←
  RETURN←
←
Yc.Check:←
  ymn=ymin+BaseLine←
  IF yc<ymn THEN yc=ymn←
  ymx=ymax-FontSize+BaseLine+1←
  IF yc>ymx THEN yc=ymx←
  RETURN←
  ←
Erase.Example:←
  COLOR Ø,Ø←
  CALL move&(Rp&,xe,ye)←
  CALL ClearEOL&(Rp&)←
  RETURN←
  ←
Show.Example:←
  COLOR 2,1←
  CALL move&(Rp&,xe,ye)←
  c$=FontName$+STR$(FontSize)←
  CALL Text&(Rp&,SADD(c$),LEN(c$))←
  RETURN←
←
Erase.Cursor:←
  ytemp=yc-BaseLine←
  LINE (xc,ytemp)-(xc,ytemp+FontSize-1),1←
  RETURN←
  ←
Delay:←
  FOR j%=1 TO 1ØØØ:NEXT←
  RETURN←
            ←
FontTypes:←
DATA topaz,8,topaz,9←
DATA diamond,12←
DATA garnet,9,garnet,16←
DATA ruby,8,ruby,12←
DATA emerald,2Ø←
DATA opal,11←
DATA sapphire,14,sapphire,15,sapphire,18,sapphire,19
←
         ←
```

**Program 2. Diskfont.bmap Filemaker**
Filename: FONTBMAP.MAKER

```
'  Copyright 1987 COMPUTE! Publications, Inc.<
'  All Rights Reserved<
<
'DiskfontMaker<
file$="libs:Diskfont.bmap"<
READ filesize,checksum<
PRINT "Checking DATA statements...":PRINT <
FOR i=1 TO filesize<
  READ a$:a=VAL("&h"+a$)<
  check=check+a<
NEXT i<
RESTORE DiskFontData<
IF check<>checksum THEN PRINT "Checksum mismatch --
error in typing.":END<
PRINT "DATA ok, creating the file."<
ON ERROR GOTO CreationError<
OPEN file$ FOR OUTPUT AS #1<
FOR i=1 TO filesize<
  READ a$:a=VAL("&h"+a$)<
  PRINT#1,CHR$(a);<
NEXT i<
CLOSE#1<
PRINT "Finished."<
END<
CreationError:<
PRINT "ERROR #";ERR:END<
<
DATA 34,3196<
DiskFontData:<
DATA 4F,70,65,6E,44,69,73,6B,46,6F,6E,74,00,FF,E2,09
<
DATA 00,41,76,61,69,6C,46,6F,6E,74,73,00,FF,DC,09,01
<
DATA 02,00<
```

# Menu Planner

W. M. Shockley
*Translation by Tim Midkiff*

*For those who like to plan ahead, here's an easy-to-use BASIC program that lets you plan dinner menus for an entire week in advance. Easy to use and feature packed, "Menu Planner" is a simply conceived and structurally sound program that is open-ended for user customizing. A disk drive is required. A printer is optional.*

In a world where there's almost always too much to do, a computer can be a wonderful ally. "Menu Planner" is a time-saving program which allows you to plan an entire week of daily menus in advance, with only a few minutes at the keyboard. When the menu is complete, it can be printed out for use in shopping and in preparing meals. Previous menus can be loaded from disk and reviewed, reused, or edited. Since the entire program is written in BASIC, it's also a simple matter to customize it for your own particular needs. Type in the listing. Be sure to save a copy before running the program.

## Main Menu

Menu Planner is a menu-driven program, meaning that you make program choices by selecting from convenient, onscreen menus. When you first run the program, it prints the main menu on the screen. By pressing the indicated letter keys, you can make a new menu, review the current menu, print the current menu, save the current menu, load a previous menu from disk, edit the current menu, or quit the program.

For instance, the first time you use Menu Planner, you will want to press M to make a new menu. The computer clears the screen and prints a new display with onscreen prompts that tell you what to do.

You will be creating a dinner menu for each day of the week, beginning with Sunday. You begin with the entree:

Choosing from the program's built-in list of entrees, Menu Planner prints the first entree choice on the screen. To move to the next entree, press the space bar. Continue cycling through entree selections until you find the one you want; then press RETURN to select it.

Under the entree section is a special offering called *Eat Out*. If you select this choice, the program jumps to the next day of the week. The last selection in each category is called *Choice*. This selection allows you to type in any food that is not offered in the program's built-in list.

After you choose an entree, the same process is repeated for three additional categories: a vegetable dish, a starch or bread dish, and a dessert. When you have completed an entire meal, the entire cycle repeats until you have made an entire week of menus. At this point, the main menu reappears.

## Other Options

The Review option lets you review the entire current menu, one meal at a time. Press the space bar to continue to the next meal.

The Print Out option prints a menu to a printer with a couple of extra items to aid you in shopping for the week. Be sure that the printer is connected and turned on before you select this option.

The Save Current Menu option saves the current menu to disk. When you select this option, Menu Planner prompts you to enter a number for the menu. Enter a 1 for the very first menu, 2 for the second, and so on. You do not have to type in a filename: Menu Planner automatically creates a filename using the number you supply. Do not use the same number twice.

The Load Old Menu option loads an existing menu from disk. This selection should be used whenever you want to review, edit, or print out an old menu. Again, you don't need to type in a filename. Simply enter the number of the menu you wish to load.

The Edit option allows you to change an existing menu. This is done by reentering the selections for an entire day.

Note that you must have a menu in memory before you select this option (if you don't, Menu Planner has nothing to edit). Type in the day of the week when prompted; you must spell out the day completely (for instance, type MONDAY instead of MON). When the menu has been altered, the main program menu reappears. This feature simplifies the process of making similar menus. Simply reload an old menu, change the meals for one or more days, and then resave it.

The Quit option ends the program. If you have not already saved the current menu, be sure to do so before quitting.

## Customizing Menu Planner

The program can be changed easily for personal preference. For example, you may prefer to start the week's menus with Monday rather than Sunday. To do so, rearrange the order of the days in line 90. If you don't like some of the foods offered in the built-in lists, substitute different foods of your own. This is done by editing the DATA statements that contain the names of dishes. If you make this change, be sure that the total number of items matches the values in lines 10–40. These lines contain REMarks to indicate which category they control. For example, to change the total number of entrees from 40 to 45, change the 40 in line 10 to 45. A more significant modification might be to add an entirely new category—say, a second vegetable dish or a beverage.

### Menu Planner

Filename: MENUP

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.←
'  All Rights Reserved←
←
5 DEFINT A-Z:SCREEN 1,320,200,4,1:WINDOW 3,"",(0,0)-
(311,186),16,1:WINDOW OUTPUT 3←
10 NI(0)=40:REM NUMBER OF ENTREES←
20 NI(1)=20:REM NUMBER OF VEGETABLES←
30 NI(2)=20:REM NUMBER OF STARCH/BREADS←
40 NI(3)=20:REM NUMBER OF DESSERTS←
50 FOR I = 1 TO 17:EL$=EL$+CHR$(32):NEXT←
60 T=0:FOR I = 0 TO 3:IF NI(I) > T THEN T=NI(I)←
```

```
70 NEXT:DIM MN$(3,T+1),SE$(3,7)<
80 FOR N = 1 TO 7:READ DA$(N):NEXT<
90 DATA SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRI
DAY,SATURDAY<
100 FOR I = 0 TO 3:READ TI$(I):TI$(I)=TI$(I)+":":NEX
T<
110 DATA ENTREE,VEGETABLE,STARCH/BREAD,DESSERT<
120 FOR I = 0 TO 3:FOR N = 1 TO NI(I):READ MN$(I,N):
NEXT:MN$(I,N)="CHOICE":NEXT<
130 CH$="MRPSLEQ":GOTO 350<
140 N=1<
150 RW=1:CL=14:GOSUB 900:PRINT"MENU PLANNER"<
160 RW=3:CL=0:GOSUB 910:PRINT DA$(N)<
170 RW=12:CL=11:GOSUB 910:PRINT"<C> TO CANCEL"<
180 RW=13:GOSUB 910:PRINT"<RETURN> TO SELECT"<
190 RW=14:GOSUB 910:PRINT"<SPACE BAR> TO CYCLE"<
200 I=0:FOR K = 0 TO 3:SE$(K,N)="":NEXT<
210 RW=5+I:CL=0:GOSUB 910:PRINT TI$(I)<
220 R=1<
230 RW=5+I:CL=15:GOSUB 910:PRINT MN$(I,R);EL$<
240 GOSUB 930<
250 IF A$<>CHR$(13) THEN 280<
260 IF R=NI(I)+1 THEN GOSUB 470:SE$(I,N)=IM$:GOTO 31
0<
270 SE$(I,N)=MN$(I,R):GOTO 310<
280 IF A$="C" THEN GOTO 150<
290 IF A$=CHR$(32) THEN R=R+1:IF R>NI(I)+1 THEN 220<
300 GOTO 230<
310 IF SE$(0,N)="EAT OUT" THEN 330<
320 I=I+1:IF I<4 THEN 210<
330 IF FL=1 THEN FL=0:GOTO 350<
340 N=N+1:IF N<8 THEN 150<
350 RW=1:CL=14:GOSUB 900:PRINT"MENU PLANNER"<
360 RW=5:CL=9:GOSUB 910:PRINT"<M>AKE A MENU"<
370 RW=6:GOSUB 910:PRINT"<R>EVIEW"<
380 RW=7:GOSUB 910:PRINT"<P>RINT OUT"<
390 RW=8:GOSUB 910:PRINT"<S>AVE CURRENT MENU"<
400 RW=9:GOSUB 910:PRINT"<L>OAD OLD MENU"<
410 RW=10:GOSUB 910:PRINT"<E>DIT"<
420 RW=11:GOSUB 910:PRINT"<Q>UIT"<
430 RW=15:CL=6:GOSUB 910:PRINT"TYPE LETTER OF YOUR C
HOICE"<
440 GOSUB 930:FOR L = 1 TO 7:IF A$<>MID$(CH$,L,1) TH
EN 450<
445 ON L GOTO 140,600,600,700,750,500,800<
450 NEXT<
460 GOTO 440<
470 GOSUB 920:RW=10:CL=0:GOSUB 910:INPUT"CHOICE";IM$
<
480 IF IM$=CHR$(32) OR IM$="" THEN 470<
490 R=46:RW=10:CL=0:GOSUB 910:PRINT EL$;EL$:RETURN<
500 RW=2:CL=5:GOSUB 900:INPUT"DAY TO EDIT";DA$<
```

```
510 FOR N = 1 TO 7:IF DA$=DA$(N) THEN FL=1:GOTO 150←
520 NEXT:RW=4:CL=3:GOSUB 910:PRINT"DAY MUST BE SPELL
ED OUT IN FULL"←
530 FOR R = 1 TO 2000:NEXT:GOTO 350←
600 B$=A$:RW=0:CL=17:GOSUB 900←
610 IF B$="P" THEN OPEN "LPT1:" FOR OUTPUT AS #1:PRI
NT #1,SPC(18) ELSE OPEN "SCRN:" FOR OUTPUT AS #1←
620 PRINT #1,"MENU:":IF B$="P" THEN PRINT #1,SPC(45)
"ON HAND      NEED TO BUY"←
630 FOR N = 1 TO 7:PRINT #1,CHR$(13):PRINT #1,SPC(20
-LEN(DA$(N))/2);DA$(N):PRINT #1,CHR$(13)←
640 FOR I = 0 TO 3:PRINT #1,SPC(4);TI$(I);SPC(14-LEN
(TI$(I)));SE$(I,N):NEXT←
650 IF B$="R" AND INKEY$="" THEN 650←
660 NEXT:CLOSE #1:GOTO 350←
700 RW=2:CL=3:GOSUB 900:INPUT "NUMBER OR MENU TO SAV
E";NO$:IF NO$="" THEN 350←
710 OPEN "MENU"+NO$ FOR OUTPUT AS #1←
720 FOR N=1 TO 7:FOR I=0 TO 3:PRINT #1,SE$(I,N):NEXT
 I,N←
730 CLOSE #1:GOTO 350←
750 RW=2:CL=3:GOSUB 900:INPUT "NUMBER OR MENU TO LOA
D";NO$:IF NO$="" THEN 350←
760 OPEN "MENU"+NO$ FOR INPUT AS #1←
770 FOR N=1 TO 7:FOR I=0 TO 3:INPUT #1,SE$(I,N):NEXT
 I,N←
780 CLOSE #1:GOTO 350←
800 WINDOW CLOSE 3:SCREEN CLOSE 1:END←
900 CLS←
910 LOCATE RW+1,CL+1:RETURN←
920 WHILE INKEY$<>"":WEND:RETURN←
930 A$=UCASE$(INKEY$):RETURN←
1000 DATA HOT DOGS,HAMBURGERS,PORK CHOPS,STEAK,ROAST
 BEEF,ROAST PORK,LAMB CHOPS←
1010 DATA VEAL CUTLETS,ROAST CHICKEN,FRIED CHICKEN,C
HICKEN SALAD,SPAGHETTI←
1020 DATA LASAGNA,TACOS,TUNA PATTIES,TUNA SALAD,TUNA
 NOODLE CASSEROLE,BEEF STEW←
1030 DATA TURKEY,POTATO CHEESE SOUP,COTTAGE CHEESE,P
EA SOUP,OMELETTE←
1040 DATA FETTUCINI,CANNELLONI,MANICOTTI,CREPES,SCRA
MBLED EGGS,BURRITOS←
1050 DATA MEAT LOAF,LEG OF LAMB,VEAL SCALOPPINE,HALI
BUT,FISH,LOBSTER,CRAB←
1060 DATA CHOW MEIN,CHOP SUEY,BEEF VEGETABLE SOUP,EA
T OUT←
1099 REM VEGETABLES:←
1100 DATA CORN,PEAS,TOMATOES,GREEN BEANS,SPINACH,GRE
EN SALAD,FRUIT SALAD,CARROTS←
1110 DATA BROCCOLI,CAULIFLOWER,SQUASH,ZUCCHINI,LETTU
CE,CUCUMBER,PICKLE,MUSHROOMS←
```

```
1120 DATA CELERY,ONIONS,GREENS,LIMA BEANS←
1199 REM STARCH/BREADS:←
1200 DATA BREAD,FRIED POTATOES,NOODLES,ROLLS,PILAF,P
ITA,KASHA,MATZOHS,CRACKERS←
1210 DATA SWEET POTATOES,RICE,FRENCH FRIES,MASHED PO
TATOES,BAKED POTATO←
1220 DATA BREAD STICKS,GARLIC BREAD,FRIED RICE,BOILE
D POTATOES,BUNS,PASTA←
1299 REM DESSERTS:←
1300 DATA ICE CREAM,POPSICLE,CAKE,DOUGHNUT,FRUIT,CHE
ESE CAKE,COOKIES,PUDDING←
1310 DATA JELLO,SORBET,BROWNIES,FUDGE,FRUIT PIE,CREA
M PIE,SWEET ROLL,TORTE←
1320 DATA BOMBE,BAKED ALASKA,DRIED FRUIT,JELLY ROLL←
←
```

# Weather Wizard

John R. Wetsch
*Translation by Bill Chin*

*This weather-forecasting program is written entirely in BASIC. Originally a generic program for a multitude of microcomputers, this version was created especially for the Amiga. Stay current on the most popular subject in the world with your own personal "Weather Wizard."*

"Weather Wizard" is a simple, easily modified BASIC program for forecasting the weather. It uses a questionnaire format to get the data needed to predict the weather. You provide the information, and Weather Wizard automatically does the rest. Type in the accompanyimg listing and save a copy before you run it.

## Entering Data

Weather Wizard is largely self-prompting, so you will not need extensive instructions to use it. The program begins by asking you to enter several items of information about current weather conditions, beginning with the current month, day, and year. Enter a number from 1 to 12 for the month, a number from 1 to 31 for the day, and so on.

The program then displays instructions for entering the next item of information, the wind direction. This is done by typing a number according to the categories shown on the screen.

Next, you must enter the current barometric pressure, expressed in inches of mercury. This will be a number in the range 27–33.

After the barometric pressure, you must enter the barometric activity (whether the barometer is steady, rising fast, and so forth). Again, the computer displays a menu indicating which number to type.

The next item of information is the prevailing cloud type. As prompted by the computer, enter a 1 for cirrus clouds, a 2 for cumulus, and so on. If you are not familiar with the various cloud types, refer to the brief explanations which follow.

Finally, enter the current humidity and temperature when prompted. When you are finished entering data, the computer prints a complete forecast for a 6- to 36-hour period. The period of the forecast depends on what sort of weather is expected. To calculate another forecast, answer *Y* at the final prompt.

## Cloud Types

*Cumulus* clouds are puffy, white, and cottonlike in appearance with a clearly defined outline. Perhaps the most familiar cloud type, these are usually found at lower altitudes, with cloud tops seldom exceeding 5000 feet.

*Altocumulus* clouds are small, semitransparent, cumulus-type cloudlets that appear in layers. This type of cloud evolves from the lifting of lower clouds. They often appear connected, and you can see the sky through them. Rounded and regularly arranged, they are usually found at an altitude of 10,000–17,000 feet.

When a cumulus cloud develops both extreme height and mass, it evolves into a *cumulonimbus* cloud. Although it makes up the most beautiful cloud mass, sweeping up into a towering column, these are the most dangerous clouds. Capable of producing heavy rain, hail, lightning, and strong, gusty winds, they occasionally mask a tornado. These clouds may be easily identified by their massive appearance, vertical development (often in excess of 20,000 feet), anvil-shaped top, and thunder and lightning. Occasionally the tops of extremely powerful cumulonimbus clouds will exceed 30,000 feet.

*Stratus* clouds are gray, featureless sheets, sometimes layered in appearance. This type of cloud produces only light precipitation, if any, and may reveal the sun through its thinnest parts. When in contact with the earth (at an altitude of 50 feet or less) this cloud type is called *fog*. The tops of stratus clouds rarely exceed 10,000 feet, although one variety, *nimbostratus*, may have cloud tops reaching to 15,000 feet.

The word *nimbus* is a Latin word meaning *violent rain* or *black rain cloud.* Nimbostratus clouds produce continous precipitation. This cloud type is often gray and is always thick enough to obscure the sun. Nimbostratus clouds are usually found near weather fronts, and although classified as a middle-altitude cloud, with tops ranging to 15,000 feet, the cloud base may be quite low.

*Altostratus* clouds are a smooth, uniform, gray sheet of cloud cover, and consist mostly of ice crystals, although the lowest portion may be water droplets. The sun may appear as though seen through ground glass, and objects on the ground will not cast a shadow. Any precipitation associated with this cloud type is continous. Altostratus clouds are considered middle-level clouds, and generally occur between 6500 and 23,000 feet in altitude.

*Cirrostratus* clouds are thin, white clouds that appear in sheets. The sun and moon are hardly ever obscured by these clouds, which often indicate severe weather to come. They are typically found high in the atmosphere, usually above 25,000 feet.

*Cirrus* clouds are wispy and white, generally occurring between 16,500 feet and 45,000 feet. A buildup of cirrus clouds may indicate an approaching warm front.

## Weather Wizard

Filename: WEATHERWIZ

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
5 SCREEN 1,320,200,4,1:WINDOW 3,"",(Ø,Ø)-(311,186),1
6,1:WINDOW OUTPUT 3←
1Ø REM Copyright 1987 COMPUTE! Publications, Inc.  A
11 rights reserved←
2Ø GOSUB 152Ø←
3Ø PRINT "The Weather Wizard"←
4Ø PRINT ←
5Ø PRINT "Copyright 1987 COMPUTE! Publications":PRIN
T"All rights reserved"←
6Ø FOR j=1 TO 3ØØØ:NEXT j←
7Ø GOSUB 152Ø←
8Ø PRINT "Enter month (use number)";:min=1:max=12:GO
SUB 139Ø:m = e←
9Ø PRINT "Enter day";:min=1:max=31:GOSUB 139Ø:d = e←
1ØØ PRINT "Enter the year";:min=Ø:max=21ØØ:GOSUB 139
```

```
0:y = e←
110 GOSUB 1520←
120 PRINT"Please input numbers when prompted by ?"←
130 PRINT ←
140 PRINT " From the menu enter the number corres-"←
150 PRINT"ponding to the wind direction"←
160 PRINT ←
170 PRINT "N= North, S= South, E= East, W= West"←
180 PRINT "(Note: The directions are where the"←
190 PRINT "wind comes from, not where it is going."←
200 PRINT "So, 'NE' means a wind blowing out of"←
210 PRINT "the northeast.)"←
220 PRINT ←
230 PRINT "--------- WIND DIRECTION MENU ---------"←
240 PRINT ←
250 PRINT "1. N"SPC(4)"2. NE"SPC(3)"3. E"←
260 PRINT ←
270 PRINT "4. SE"SPC(3)"5. S"SPC(4)"6. SW"←
280 PRINT ←
290 PRINT "7. W"SPC(4)"8. NW"SPC(3)"9. No wind"←
300 PRINT ←
310 PRINT←
320 min=1:max=9:GOSUB 1390:w = e←
330 GOSUB 1520←
340 PRINT "Enter barometric pressure":PRINT "in inch
es of Hg (ex. 29.95) ";←
350 min=27:max=33:GOSUB 1480:bp = e←
360 GOSUB 1520 ←
370 PRINT "Please enter the number that describes"←
380 PRINT "current barometric activity."←
390 PRINT "Barometer is: 1. steady"←
400 PRINT  SPC(14)"2. rising fast"←
410 PRINT  SPC(14)"3. slowly rising"←
420 PRINT  SPC(14)"4. falling fast"←
430 PRINT  SPC(14)"5. slowly falling"←
440 PRINT ←
450 min=1:max=5:GOSUB 1390:bd =e←
460 GOSUB 1520←
470 PRINT  "Please enter the number for the main"←
480 PRINT  "cloud type for your location."←
490 PRINT ←
500 PRINT "1. Cirrus":PRINT "2. Cumulonimbus":PRINT
"3. Stratus" ←
510 PRINT "4. Nimbostratus":PRINT "5. Altostratus":P
RINT "6. Altocumulus"←
520 PRINT "7. Cirrostratus":PRINT "8. Cumulus":PRINT
 "9. No clouds":PRINT ←
530 min=1:max=9:GOSUB 1390:c = e←
540 GOSUB 1520←
550 PRINT "Input Humidity":PRINT "(ex. input 50 for
50% humidity)";←
560 min=0:max=100:GOSUB 1480:h = e←
```

```
570 GOSUB 1520←
580 PRINT "Input Temperature in Fahrenheit":PRINT"(e
x. 63 = 63 deg. F)";←
590 min=-130:max=130:GOSUB 1480:t = e←
600 GOSUB 1520←
610 REM forecasts←
620 IF bp >29.9 THEN 710←
630 IF (w>=1 AND w<=5) AND bd=4 AND bp>29.7 THEN 860
←
640 IF (w>=6 AND w<=8) AND (bd>=1 AND bd<=3) AND t>7
5 AND h>68 THEN 910←
650 IF (w>=6 AND w<=8) AND (bd=2 OR bd=3) THEN 930←
660 IF bp>29.7 THEN 810←
670 IF (bd=2 OR bd=3) THEN 1040←
680 IF (w>=1 AND w<=5) AND bd=4 THEN 1110←
690 IF (w=2 OR w=3) AND bd=5 THEN 1120←
700  GOTO 810←
710 IF (w>=6 AND w<=8) AND (bd=1 OR bd=3) AND t>75 A
ND h>70 THEN 950←
720 IF (w>=6 AND w<=8) AND (bd=1 OR bd=3) THEN 970←
730 IF w=4 AND bd=5 THEN 1000←
740 IF w=4 AND bp>30.5 AND (bd=4 OR bd=5) THEN 1020←
750 IF (w=4 OR w=5) AND bd=4 THEN 1060←
760 IF (w=4 OR w=5) AND bd=5 THEN 1080←
770 IF (w=3 OR w=4) AND bd=4 THEN 1090←
780 IF w=5 AND (bd=4 OR bd=5) THEN 860←
790 IF (w=1 OR w=2) AND bd=4 THEN 1140←
800 IF (w=1 OR w=2) AND bd=5 THEN 1160←
810 ON c GOTO 1180, 1210, 1260, 1280, 1300, 820, 131
0, 840, 840←
820 IF t>75 AND h>68 THEN 950←
830  GOTO 1310←
840 IF t>75 AND h>68 THEN 1330←
850 GOTO 1340←
860 PRINT "FORECAST: STRONG WINDS and HEAVY"←
870 PRINT "PRECIPITATION CAN BE EXPECTED within"←
880 PRINT "the next 6-24 hours.  Lower tempera-"←
890 PRINT "tures are also forecast."←
900 GOTO 1530←
910 PRINT"[SMALL CHANCE OF THUNDERSTORMS TODAY]"←
920 PRINT ←
930 PRINT "FORECAST: FAIR weather can be expected"←
940 PRINT "for the next 24 to 36 hours.":GOTO 1530←
950 PRINT "[SLIGHT CHANCE OF THUNDERSTORMS TODAY]"←
960 PRINT ←
970 PRINT "FORECAST: continued FAIR weather for"←
980 PRINT "the next 24 hours.  Temperatures will"←
990 PRINT "remain nearly constant.":GOTO 1530←
1000 PRINT "FORECAST: PRECIPITATION can be expected
←
```

```
1010 PRINT  "in 24-48 hours.":GOTO 1530←
1020 PRINT "FORECAST: WARMER temperatures for the"←
1030 PRINT "next 24 hours.":GOTO 1530←
1040 PRINT "FORECAST: Clear weather ahead with"←
1050 PRINT "cooler temperatures.":GOTO 1530←
1060 PRINT "FORECAST: HIGH WINDS and PRECIPITATION"←
1070 PRINT "within 12 hours.":GOTO 1530←
1080 PRINT "FORECAST: PRECIPITATION within 24 hours.
":GOTO 1530←
1090 PRINT "FORECAST: HIGH WINDS and PRECIPITATION"←
1100 PRINT "within 24 hours.":GOTO 1530←
1110 PRINT "FORECAST: SEVERE weather; stormy with":P
RINT "high winds.":GOTO 1530←
1120 PRINT "FORECAST: continuing precipitation can"←
1130 PRINT "be expected for the next 12 to 24 hours.
":GOTO 1530←
1140 PRINT "FORECAST: STORMS with HIGH WINDS within"
←
1150 PRINT "24 hours.  Cooler temperatures as well."
:GOTO 1530←
1160 PRINT "FORECAST: Precipitation within 24-36"←
1170 PRINT "hours, slightly cooler temperatures.":GO
TO 1530←
1180 PRINT "FORECAST: If cirrus clouds are"←
1190 PRINT "increasing and becoming more dominant"←
1200 PRINT "we can expect warmer weather.":GOTO 1530
←
1210 PRINT "FORECAST: 50-80% chance of preci-"←
1220 PRINT "pitation within the next 24-36 hours."←
1230 PRINT "Chance of precipitation will increase"←
1240 PRINT "with the accumulation of more cumulo-"←
1250 PRINT "nimbus type clouds.":GOTO 1530←
1260 PRINT "FORECAST: 10-20% chance of ";:IF t>35 TH
EN PRINT "showers":GOTO 1530←
1270 PRINT "snow flurries":GOTO 1530←
1280 PRINT "FORECAST: 90% chance of HEAVY precipi-"←
1290 PRINT  "tation within 12-24 hours.":GOTO 1530←
1300 PRINT "FORECAST: Chance of light precipitation.
":GOTO 1530←
1310 PRINT "FORECAST: PRECIPITATION with HEAVY"←
1320 PRINT "winds in 24-48 hours.":GOTO 1530←
1330 PRINT "[POSSIBILITY OF THUNDERSTORMS TODAY]"←
1340 PRINT "FORECAST: FAIR weather today with light"
←
1350 PRINT "to moderate winds.":GOTO 1530←
1360 REM integer input routine←
1370 PRINT "enter an integer between";min;" and ";ma
x←
1380 PRINT ", try again"←
1390 e$="":INPUT e$:e = VAL (e$)←
1400 IF e=0 AND LEFT$(e$,1)<>"0" THEN 1370←
1410 IF e<>INT(e) THEN PRINT "please enter an intege
```

```
r";:GOTO 138Ø←
142Ø IF min>e THEN PRINT "too low";:GOTO 138Ø←
143Ø IF e>max THEN PRINT "too high";:GOTO 138Ø←
144Ø RETURN←
145Ø REM real number input routine←
146Ø PRINT  "enter a number between";min" and ";max←
147Ø PRINT ", try again"←
148Ø e$="":INPUT e$:e = VAL (e$):IF e=Ø AND LEFT$(e$
,1)<>"Ø" THEN 146Ø←
149Ø IF min>e THEN PRINT "too low";:GOTO 147Ø←
15ØØ IF e>max THEN PRINT "too high";:GOTO 147Ø←
151Ø RETURN←
152Ø CLS:RETURN←
153Ø PRINT :PRINT m;"/";d;"/";y:PRINT ←
154Ø PRINT "Temperature is:";t;"deg. F":PRINT ←
155Ø PRINT "Humidity is: ";h;"%":PRINT ←
156Ø PRINT "Barometric pressure is: ";bp;" in. Hg" ←
157Ø FOR j=1 TO 35ØØ:NEXT j←
158Ø PRINT :PRINT :PRINT :PRINT :PRINT ←
159Ø PRINT "press <RETURN> to make another forecast"
←
16ØØ INPUT p$←
161Ø GOSUB 152Ø←
162Ø GOTO 12Ø←
←
←
←
```

# CHAPTER SIX

# Graphics

# GraphiDemo

Stefan Lindahl

*This intriguing graphics program, written by a COMPUTE! magazine reader in Sweden, demonstrates the Amiga's tremendous graphics processing power as well as the speed of Amiga Basic. The program requires 512K of memory.*

"GraphiDemo" demonstrates just how easy it is to create impressive graphics in Amiga Basic. Type it in and save a copy of the program; then run it. GraphiDemo begins by displaying a help screen that explains all of the program's options. You can recall this screen at any time by pressing the Help key. Take a moment to look at all the different options; you'll want to try them all.

GraphiDemo's options can be invoked in two different ways. If you press the right mouse button and examine the menus at the top of the screen, you'll see that every option can be selected from a menu, using the mouse pointer. However, GraphiDemo uses all of the Amiga's colors, which can make the menus unreadable at times. Thus, you can also select any option by pressing the key indicated in the help screen. If you forget which key is assigned to which option, simply press Help. When you exit the help screen, the main screen is restored to its original condition.

Since the program is entirely self-prompting, no elaborate explanations are necessary. Just run it, follow the prompts, and enjoy the show. If you're interested in graphics programming, the program provides examples of how to draw different shapes and control the color palette for various effects.

## GraphiDemo

Filename: GRAPHIDEMO

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
REM ** Copyright 1987 Compute! Publications, Inc.  *
*<
REM ** All Rights Reserved **<
<
CLEAR ,13000          :REM * Release basic memory to s
ystem *<
DEFINT b-y            :REM * Integer definition *<
<
RANDOMIZE TIMER       :REM * New random seed *<
<
b=5                   :REM * Maximum step length *<
cm=0                  :REM * Circlemode off *<
depth=4               :REM * No of bitlayers *<
<
SCREEN 2,640,200,depth,2<
WINDOW 2,,,16,2<
maxcoLor=2^depth-1<
<
GOSUB SetcoL<
<
GOSUB CLrmenu<
<
ch=2:ch2temp=3:GOSUB 10:GOSUB 20  :REM * Set menus &
 check marks *<
<
ON MENU GOSUB Mnuche<
MENU ON<
<
ON MOUSE GOSUB Chkmus<
MOUSE ON<
<
GOSUB Info       :REM * Display info-window *<
<
VarvaL:<
x1=50+RND*540:y1=50+RND*120:x2=50+RND*540:y2=50+RND*
120<
IF x2<x1 OR y2<y1 THEN VarvaL<
xs1=(1+RND*b):xs2=(1+RND*b):ys1=(1+RND*b):ys2=(1+RND
*b)<
minx=0:maxx=629:miny=0:maxy=195<
<
Main:<
FOR doit=-1 TO 1 STEP 2<
<
  FOR cc=maxcoLor*-(doit=-1) TO maxcoLor*-(doit=1) S
TEP doit<
```

```
    oLdx1=x1:oLdy1=y1:x1=x1+xs1:y1=y1+ys1←
    oLdx2=x2:oLdy2=y2:x2=x2+xs2:y2=y2+ys2←
←
    IF x1<minx OR x1>maxx THEN x1=x1-xs1:xs1=(1+RND*
b)*-SGN(xs1)←
    IF x2<minx OR x2>maxx THEN x2=x2-xs2:xs2=(1+RND*
b)*-SGN(xs2)←
    IF y1<miny OR y1>maxy THEN y1=y1-ys1:ys1=(1+RND*
b)*-SGN(ys1)←
    IF y2<miny OR y2>maxy THEN y2=y2-ys2:ys2=(1+RND*
b)*-SGN(ys2)←
    key=ASC(INKEY$+CHR$(Ø)):IF key THEN GOSUB Keyche
←
    ON choice GOSUB Serpent,Lines,Boxes,Filledboxes←
  NEXT cc←
NEXT doit←
 ←
GOTO Main    ←
←
Serpent:←
    COLOR cc:IF cm THEN←
      GOSUB CircLepos:AREA STEP (Ø,Ø):AREA (x1,y1):A
REA (x2,y2)←
      AREAFILL←
    ELSE←
      AREA (x1,y1):AREA (oLdx1,oLdy1):AREA (x2,y2):A
REA (oLdx2,oLdy2)←
      AREAFILL←
      AREA (x1,y1):AREA (x2,y2):AREA (oLdx1,oLdy1):A
REA (oLdx2,oLdy2)←
      AREAFILL←
    END IF←
  RETURN←
    ←
Lines:←
    IF cm THEN GOSUB CircLepos ELSE PSET (x1,y1),cc←
    LINE -(x2,y2),cc←
  RETURN←
    ←
Boxes:←
    IF cm THEN GOSUB CircLepos ELSE PSET (x1,y1),cc←
    LINE -(x2,y2),cc,b←
  RETURN←
    ←
Filledboxes:←
    IF cm THEN GOSUB CircLepos ELSE PSET (x1,y1),cc←
    LINE -(x2,y2),cc,bf←
  RETURN←
←
CircLemode:←
    CLS:cm=ABS(cm-1)←
  RETURN←
```

```
<
CircLepos:<
    api=api+.Ø5:PSET (314+2ØØ*SIN(api),9Ø+7Ø*COS(api
)),cc<
  RETURN<
     <
Mnuche:<
    IF MENU(Ø)=2 THEN menu2     :REM * Menu 1 or 2 ?
<
menu1:<
    ch=MENU(1)<
    IF ch>4 THEN ON ch-4 GOTO SetcoL,Info,Resetprog<
1Ø  GOSUB Setmenu<
    choice=ch:MENU 1,choice,2<
    RETURN<
menu2:<
    ch2temp=MENU(1)<
2Ø  IF ch2=ch2temp THEN RETURN ELSE ch2=ch2temp<
21  GOSUB menuNo2:MENU 2,ch2,2<
    GOTO SetcoL<
    <
Keyche:                   <
    IF key=27 THEN Resetprog                :REM * Esc
 key *<
    IF key=139 THEN Info                    :REM * Help
 key *<
    IF key=133 THEN SetcoL                  :REM * F5
 key *<
    IF key=127 THEN CLS                     :REM * Del
 key *<
    IF (key AND 223)=67 THEN CircLemode     :REM * 'C'
 key *<
    IF key>133 AND key<137 THEN coLormode<
    IF key<129 OR  key>132 THEN RETURN <
    ch=key-128:GOTO 1Ø          :REM * F1 - F4 *<
coLormode:<
    ch2temp=key-133:GOTO 2Ø     :REM * F6 - F8 *<
  RETURN<
      <
        <
Setmenu:<
    MENU 1,Ø,1,"   Main:"<
    MENU 1,1,1,"   Serpent      "<
    MENU 1,2,1,"   Lines        "<
    MENU 1,3,1,"   Boxes        "<
    MENU 1,4,1,"   Filledboxes  "<
    MENU 1,5,1,"   New Colors   "<
    MENU 1,6,1,"   Help         "<
    MENU 1,7,1,"   Stop program"<
  RETURN<
```

```
menuNo2:←
    MENU 2,Ø,1,"   ColorOptions:"←
    MENU 2,1,1,"   RGB <-> BLACK shading"←
    MENU 2,2,1,"   Random color shading "←
    MENU 2,3,1,"   Random colors        "←
  RETURN ←
←
CLrmenu:←
    MENU 1,Ø,Ø,""←
    MENU 2,Ø,Ø,""←
    MENU 3,Ø,Ø,""         '  Clear standard menus←
    MENU 4,Ø,Ø,""←
  RETURN←
    ←
SetcoL:←
    ON ch2 GOTO SetcoL1,SetcoL2,SetcoL3←
←
SetcoL1:←
    CLS←
    c=RND*7+.5 ←
    c1=SGN(c AND 1)←
    c2=SGN(c AND 2)←
    c3=SGN(c AND 4)    ←
    FOR c=Ø TO maxcoLor←
      PALETTE c,(c/16)*c1,(c/16)*c2,(c/16)*c3←
    NEXT c←
    RETURN←
←
SetcoL2:←
    CLS←
    FOR w=1 TO 3:a1(w)=RND:a2(w)=RND:NEXT w←
    FOR w=1 TO 3:adeL(w)=(a2(w)-a1(w))/(maxcoLor+1):
NEXT w←
    ←
    FOR w=Ø TO maxcoLor←
      PALETTE w,a1(1),a1(2),a1(3)←
      FOR w1=1 TO 3:a1(w1)=a1(w1)+adeL(w1):NEXT w1←
    NEXT w←
  RETURN  ←
←
SetcoL3:←
    FOR c=Ø TO maxcoLor←
      PALETTE c,RND,RND,RND←
    NEXT c←
  RETURN←
  ←
  ←
Resetprog:←
    MENU RESET←
    PALETTE Ø,.4375,.125,.1875←
    PALETTE 1,1,.56,Ø←
    PALETTE 2,1,.1,.6<
    PALETTE 3,.44,.6,.94<
    WINDOW CLOSE 2<
    SCREEN CLOSE 2<
    CLS<
    END<
<
Chkmus:<
    IF inf THEN inf=Ø ELSE CLS<
  RETURN<
<
Info:<
    MENU STOP:inf=1:REM* To tell mouse-trapping rout
ine <
    WINDOW 3,,(1ØØ,1Ø)-(517,175),Ø,2  :REM that we'r
e in Info<
    CLS:COLOR maxcoLor-2<
    PRINT SPACE$(5);"Copyright 1987 Compute! Publica
tions, Inc."<
    PRINT SPACE$(16);"All Rights Reserved":PRINT<
    PRINT SPACE$(2Ø);"GRAPHIDEMO"<
    PRINT SPACE$(8);"F1 or Menu ................. Se
rpent"<
    PRINT SPACE$(8);"F2 or Menu ..................
Lines"<
    PRINT SPACE$(8);"F3 or Menu ..................
Boxes"<
    PRINT SPACE$(8);"F4 or Menu ............ Filled
boxes"<
    PRINT SPACE$(8);"F5 or Menu ............. New C
olors"<
    PRINT :PRINT SPACE$(8);"F6 or Menu ... RGB <-> B
LACK shading"<
    PRINT SPACE$(8);"F7 or Menu .... Random color sh
ading"<
    PRINT SPACE$(8);"F8 or Menu .......... Random c
olors"<
    PRINT SPACE$(8);"'C' key ........ toggle 'Circle
mode' "<
    PRINT:PRINT "Clear Screen with Left Mouse button
 or the DEL key."<
    PRINT "  Stop the Program with the ESC key or fr
om Menu."<
    PRINT:PRINT "Get this window back with the HELP
key or from Menu."<
    PRINT:PRINT SPACE$(14);"PRESS ANY KEY TO CONTINU
E";<
Waithere:<
```

```
    IF INKEY$="" AND inf=1 THEN Waithere    :REM * Ch
eck for key or  ←
    WINDOW CLOSE 3                          :REM * mo
usebutton←
    MENU ON    ←
  RETURN←
    ←
    ←
    ←
```

# Fractal Mountains

Matthew Timmerman

*With this landscape-generating program and a 512K Amiga computer, you can create fascinating graphics displays based on the principles of fractal geometry.*

"Fractal Mountains" is an intriguing graphics program that draws landscapelike scenes containing rough, crinkled-looking mountain shapes. Although you can run it and simply enjoy the pictures it creates, the program is based on highly advanced mathematical concepts which you may wish to learn more about. Type in the program and save a copy before you run it.

When you run Fractal Mountains, it asks you to enter a number for the random seed. This value determines which landscape the program generates. There are 65,536 possible landscapes, so you needn't repeat a landscape very often unless you find one you particularly like. Enter a number in the range indicated by the onscreen prompt. (The first time you run the program, try entering 70 for the random seed.)

The second prompt asks you to enter a number for the maximum variation. This value determines the cragginess of the mountains. Although the program prompts you to enter a value in the range 0–2, you aren't necessarily limited to that range. A variation of 0 gives you a perfectly flat plane (no variation), while a value of 2 gives you extremely high, rugged mountain peaks. (The first time you run the program, try entering 1.96 for the maximum variation.)

After you've entered those values, the program draws the landscape. Please be patient while the process is underway. Although Amiga Basic is one of the fastest microcomputer BASICs, it still takes considerable time to perform the tens of thousands of calculations this program requires.

Once the picture is complete, you can save the picture to disk by pressing the S key. To show you what is happening, the program draws a white line on the screen indicating which line of the picture it is saving. The picture is saved in ILBM (InterLeaved BitMap) format, which allows you to load it into *Deluxe Paint, Graphicraft,* and other IFF-compatible art programs. To exit the program, press the space bar.

If you find a mountain that you like, but it is too smooth, use the same random seed value with a higher maximum variation. If you get a landscape which is mostly water, try it again using a negative maximum variation. The negative value inverts the landscape: What was once land becomes water, and vice versa. Since gravity is meaningless in this universe, pictures look as good upside down as they do right side up. Don't be afraid to experiment with different values. Not all combinations give pleasing results, but exploration is one of the interesting aspects of using a program like this.

## Why Fractals?

A fractal is an object with a *fractional dimension*—a value between 1 and 2, for instance, or between 2 and 3. In his book *The Fractal Geometry of Nature,* Benoit Mandelbrot tells us to imagine a piece of aluminum foil. When it's flat, it is a simple plane with two dimensions. As you begin to crinkle the foil, it can no longer be confined to two dimensions, but it is not yet three-dimensional. Therefore, it has a dimension somewhere 2 and 3. The aluminum foil becomes more crinkled until, eventually, it becomes a solid block filling three dimensions.

The same analogy can be applied to a straight line becoming bent until it becomes infinitely bent and complex, completely darkening the surface on which it is drawn. At that point, the line is no longer a line, but a two-dimensional plane.

Another aspect of fractals is self-similarity, meaning that the big parts of the object look like the little parts. To take a rough example, if you break a chunk of rock from a mountain, the rock looks like a miniature mountain. It's not the same, but it has the same general look. This phenomenon occurs throughout nature: in tree bark, snowflakes, coastlines of continents, trees, clouds, surfaces of proteins, all types of turbulence, and the positioning of stars, planets, solar systems, and galaxies—all of which are only a few of the countless possibilities.

The algorithm used to create these pictures is derived from one described in the September 1984 issue of *Scientific American*. However, whereas that formula was based on a triangle, this one is based on a square (for display purposes).

To understand how the program works, imagine that you begin with a square, putting a point in the center of all four sides of the shape. Next, raise or lower those points a random amount, as much as half the length of the square. At that point, you put a point in the center of the square and give it the average height of all the corners. Raise or lower this point by a random amount, using the same conditions as in the previous adjustment, and connect the points.

You now have four smaller squares. By repeating this process, you obtain smaller and smaller squares, eventually obtaining a good approximation of a natural landscape surface.

The most difficult part of making a convincing mountain is putting the picture on the screen. This program draws the mountains as if the sun were directly above them, for a couple of reasons. Since the algorithm produces no overhanging pieces, you don't have to worry about one part of the landscape shadowing another part. Secondly, the program already takes a considerable time to work, without adding the extra complication and delay of computing the effect of light falling at an angle.

The landscape is stored as a square grid of height values in an array named *lv*. In order to draw the landscape, each surface in the array must be given a *shade value* in relation to how bright the surface appears. The subroutine *Getshade* calculates the shade value as the slope of the plane in relation to the light source.

The problem is that the four points do not necessarily fall all on the same plane. For this reason, a point is placed in the center of each square and a separate shade value is calculated for all four triangles that are formed. What you get is an aerial view of mountains, islands, or whatever happens to come out. The waterline is at 0, meaning that all points in the viewing area with negative values are covered with water. In addition, snow covers all peaks more than three-quarters as high as the highest point in the picture.

## Fractal Mountains

Filename: FRACTALMTNS

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'Copyright 1987 Compute! Publications, Inc.←
'All Rights Reserved.←
CLEAR 32767←
DEFSNG a-z←
DIM Lv(64,64)←
DIM cmap$(31)←
PRINT"    Copyright 1987"←
PRINT"Compute! Publications, Inc."←
PRINT"  All Rights Reserved.":PRINT←
RANDOMIZE←
PRINT "Enter maximum variation (Ø-2) (1 is nice) ";:
INPUT max←
PRINT "Enter a filename to save picture under."←
INPUT "(Saving at end is optional.)   ";fiL$←
FOR a = 1 TO 1Ø←
PRINT RND←
NEXT←
SCREEN 2,32Ø,2ØØ,5,1←
WINDOW 3,"Mountain",(Ø,Ø) - (311,186),28,2←
FOR a = Ø TO 15←
PALETTE a,a/15,a/25,a/5Ø←
PALETTE a+16,a/15,a/15,a/15←
a$ = CHR$(a*17)←
cmap$(a) = a$+CHR$(a*1Ø.2)+CHR$(a*5.1)←
cmap$(a+16) = a$+a$+a$←
NEXT←
PALETTE 16,Ø,.25,.5←
cmap$(16) = CHR$(Ø)+CHR$(64)+CHR$(128)←
COLOR 15←
maxLv = Ø←
MakeMount:←
FOR iter = 6 TO 1 STEP -1←
sk = 2 ^ iter←
hL = sk/2←
PRINT "Doing Iteration";iter←
Dotops:←
PRINT "Tops & Bottoms  ";←
FOR y = Ø TO 64 STEP sk←
FOR x = hL TO 64 STEP sk←
ran = (RND-.5)*max*sk←
oLd = (Lv(x-hL,y) + Lv(x+hL,y))/2←
```

```
Lv(x,y) = oLd + ran←
NEXT x←
NEXT y←
Dobottoms:←
PRINT "Sides  ";←
FOR x = Ø TO 64 STEP sk←
FOR y = hL TO 64 STEP sk←
ran = (RND-.5)*max*sk←
oLd = (Lv(x,y-hL) + Lv(x,y+hL))/2←
Lv(x,y) = oLd + ran←
NEXT y←
NEXT x←
Docentres:←
PRINT "Centers  "←
FOR x = hL TO 64 STEP sk←
FOR y = hL TO 64 STEP sk←
ran = (RND-.5)*max*sk←
oLd1 = (Lv(x+hL,y-hL) + Lv(x-hL,y+hL))/2←
oLd2 = (Lv(x-hL,y-hL) + Lv(x+hL,y+hL))/2←
oLd = (oLd1 + oLd2)/2←
Lv(x,y) = oLd + ran←
IF Lv(x,y) > maxLv THEN maxLv = Lv(x,y)←
NEXT y←
NEXT x←
NEXT iter←
snowLine = maxLv - maxLv/4←
drawmount:←
CLS←
xm = 4←
ym = 1←
xshift = .5←
yp = 7Ø←
FOR x = Ø TO 64←
IF Lv(x,Ø) < Ø THEN Lv(x,Ø) = Ø←
NEXT x←
FOR y = Ø TO 63←
IF Lv(Ø,y) < Ø THEN Lv(Ø,y) = Ø←
FOR x = Ø TO 63←
IF Lv (x+1,y+1) < Ø THEN Lv(x+1,y+1) = Ø←
Lv = Lv(x,y) + Lv(x+1,y) +Lv(x,y+1)←
Lv = (Lv + Lv(x+1,y+1))/4←
a=x:b=y←
rx1 = xm * a + xshift * b←
ry1 = ym * b + yp -Lv(a,b)←
GOSUB getshade:←
shade1 = shade←
a=x+1←
rx2 = xm * a + xshift * b←
ry2 = ym * b + yp -Lv(a,b)←
GOSUB getshade:←
shade2 = shade←
a=x:b=y+1←
```

```
rx3 = xm * a + xshift * b←
ry3 = ym * b + yp -Lv(a,b)←
GOSUB getshade:←
shade3 = shade←
a=x+1←
rx4 = xm * a + xshift * b←
ry4 = ym * b + yp -Lv(a,b)←
GOSUB getshade:←
shade4 = shade←
a=x+.5:b=y+.5←
rx = xm * a + xshift * b←
ry = ym * b + yp←
a=x:b=y←
ry = ry - Lv←
AREA (rx,ry)←
AREA (rx1,ry1)←
AREA (rx2,ry2)←
COLOR shade1←
AREAFILL←
AREA (rx,ry)←
AREA (rx2,ry2)←
AREA (rx4,ry4)←
COLOR shade2←
AREAFILL←
AREA (rx,ry)←
AREA (rx1,ry1)←
AREA (rx3,ry3)←
COLOR shade3←
AREAFILL←
AREA (rx,ry)←
AREA (rx3,ry3)←
AREA (rx4,ry4)←
COLOR shade4←
AREAFILL←
 NEXT x←
NEXT y←
ender:←
a$ = INKEY$←
IF a$ = "s" THEN GOTO savepic←
IF a$ <> " " THEN GOTO ender←
end2:←
WINDOW CLOSE 3←
SCREEN CLOSE 2←
WINDOW OUTPUT 1←
END←
getshade:←
c = x + 1 - (b-y)←
d = y + (a-x)←
xc = x+.5←
yc = y+.5←
xrun1 = xc - a←
xrun2 = xc - c←
yrun1 = yc - b←
```

```
yrun2 = yc - d←
rise1 = Lv - Lv(a,b)←
rise2 = Lv - Lv(c,d)←
yrise = ABS(rise1*xrun2 - rise2 *xrun1)←
yrun = ABS(yrun1*xrun2 - xrun1*yrun2)←
IF yrun = yrise THEN yrun = 1:yrise = 1←
xrise = ABS(rise1*yrun2 - rise2*yrun1)←
xrun = ABS(xrun1*yrun2 - yrun1*xrun2)←
IF xrun = xrise THEN xrun = 1:xrise = 1←
xrise = xrise / 2←
yrise = yrise / 2←
xshade = 1-ABS(xrise / (xrun + xrise))←
yshade = 1-ABS(yrise / (yrun + yrise))←
shade = 14*xshade*yshade+1 ←
IF Lv > snowLine THEN shade = shade + 16←
IF Lv <= Ø THEN shade = 16←
RETURN←
savepic:←
rastport& = WINDOW(8)←
bitmap& = PEEKL(rastport&+4)←
topLine = 6Ø - INT(maxLv)←
IF topLine < Ø THEN topLine = Ø←
topadd = topLine * 4Ø←
FOR a = Ø TO 4←
pLane&(a) = PEEKL(bitmap& + 8 + a*4)+topadd←
NEXT←
bottomLine = 144←
Lines = bottomLine - topLine←
OPEN fiL$ FOR OUTPUT AS 1←
a$ = MKL$(Lines * 4Ø * 5 + 144)←
PRINT#1,"FORM";a$;"ILBMBMHD";MKL$(2Ø);←
PRINT#1,MKI$(32Ø);MKI$(Lines);MKL$(Ø);←
PRINT#1,CHR$(5);MKI$(Ø);CHR$(Ø);←
PRINT#1,MKI$(Ø);CHR$(1Ø);CHR$(11);←
PRINT#1,MKI$(32Ø);MKI$(2ØØ);←
PRINT#1,"CMAP";MKL$(96);←
FOR a = Ø TO 31←
PRINT#1,cmap$(a);←
NEXT←
PRINT#1,"BODY";MKL$(Lines * 4Ø * 5);←
FOR a = 1 TO Lines←
FOR p = Ø TO 4←
FOR b = Ø TO 39 STEP 4←
PRINT#1,MKL$(PEEKL(pLane&(p) + b));←
NEXT b←
POKEL pLane&(p),-1←
pLane&(p) = pLane&(p) + 4Ø←
NEXT p←
NEXT a←
CLOSE←
GOTO end2←
  ←
```

# IFF Translator

Michael Barron

*This program translates IFF image files from programs such as* Deluxe Paint *into files which can be used in your own Amiga Basic programs. It works only in conjunction with a drawing program that creates IFF-format image files (see below).*

Amiga Basic provides a number of powerful OBJECT commands for animating sprite and bob shapes. In order to use these commands, you must first have an image to animate. The Amiga Extras disk includes an object editor for creating sprite and bob images; however, this simple program has a number of limitations. Commercial drawing programs such as *Deluxe Paint, Aegis Images,* and *Graphicraft* are much more powerful and convenient to use, but the image files which they create can't be used directly in Amiga Basic.

"IFF Translator" allows you to translate standard IFF Amiga graphics files for use in BASIC programs. (IFF stands for Interchange File Format, a standard file structure developed jointly by Electronic Arts and Commodore-Amiga. Most commercial drawing programs for the Amiga store images according to the IFF standards.)

## Creating an Image

Before you can use IFF Translator, you must create an image file with *Deluxe Paint* or any other IFF-compatible program. This article will discuss the procedure to follow with *Deluxe Paint;* other programs use a similar process. Once you have created the image file, you can run IFF Translator to convert the file to a form usable by BASIC.

Before you create the image, set the drawing program to the type of screen which you'll need in BASIC. For instance, if you wish the image to be compatible with the default (Workbench) screen, the drawing program should be in medium resolution with two bit planes. In *Deluxe Paint,* the screen setting is determined by the command you use to start the program from the CLI. Table 6-1 summarizes the commands used to select different screens in *Deluxe Paint.*

**Table 6-1. *Deluxe Paint* Command Summary**

```
1> DPAINT <resolution> <#planes>
```

| resolution: | | #planes: | |
|---|---|---|---|
| LO | 320*200 | 5 | 5 bit planes, 32 colors (low res only) |
| MED | 640*200 | 4 | 4 bit planes, 16 colors |
| HI | 640*400 | 3 | 3 bit planes, 8 colors |
| | | 2 | 2 bit planes, 4 colors |
| | | 1 | 1 bit plane, 2 colors |

The default *Deluxe Paint* screen could be entered as:

```
1> DPAINT LO 5
```

A Workbench-type screen could be entered as:

```
1> DPAINT MED 2
```

IFF Translator also incorporates the color palette from the original image file. To change the color palette in *Deluxe Paint,* click the left mouse button on the foreground color indicator and adjust the color sliders of the palette window as needed.

After you have drawn the desired image, it must be saved as a brush. To select an image as a brush within *Deluxe Paint,* click the left mouse button on the brush-selection tool and drag a selection box around the image. To save a brush file, select the Save As option from the program's Brush menu.

Exit the drawing program. At this point, the image has been saved as a brush file on disk. To minimize disk-swapping, you may want to copy the brush file to your BASIC work disk. IFF Translator will store the BASIC-compatible files in the same folder as the original brush file.

## Using IFF Translator

Type in and save IFF Translator; then run it. The program asks you to enter the name of an IFF file to translate. Enter the correct name and allow the program to process the file. IFF Translator reads and writes to disk until the translation is complete, so do not disturb the destination disk.

After the translation is complete, the program recreates the screen that was in effect when you created the original image. Then it displays the image on the screen and permits you to move it around with the mouse pointer for examination. Press any key to exit the program.

IFF Translator creates two new files for BASIC. The first is a file of the same name with the extension _Field. The second has the same name with the extension _Image. For example, if the original file is named FLOWER, IFF Translator creates files named FLOWER_Field and FLOWER_Image.

The _Field file is a BASIC subroutine which recreates the original screen from which the image was saved. This subroutine, named DefinePlayField:, is stored in plain ASCII form and can be merged with an existing program or used as the basis of a new program.

The _Image file is compatible with OBJECT commands in Amiga Basic. Specifically, it is used with OBJECT.SHAPE to define an image. This image can be used exactly like any image created with the BASIC object editor program. The following code demonstrates how to read and define an image contained in an _Image file:

**OPEN "*filename*_Image" FOR INPUT AS 1**
**OBJECT.SHAPE 1,INPUTS$(LOF(1),1)**
**CLOSE 1**

Of course, you should replace *filename* in the OPEN command with the name used for your file. Once the object has been defined, it can be manipulated like any other object, as explained in the Amiga Basic manual.

## IFF Translator

Filename: IFFTRANS

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
<
<
'TranslateIFF <
'Use IFF brush files with AmigaBASIC object commands
<
'Copyright 1987 Compute! Publications, Inc<
'All Rights Reserved<
<
<
```

```
Mainline:←
  GOSUB Initialize←
  OPEN FileIFF$ FOR INPUT AS 1←
    PRINT "Reading from file     : ";FileIFF$←
    GOSUB TranslateChunks←
  CLOSE 1←
  OPEN FileIFF$+"_Field" FOR OUTPUT AS 1←
    PRINT "Writing to file       : ";FileIFF$+"_Field
"←
    GOSUB CreateFieldFile←
  CLOSE 1←
  OPEN FileIFF$+"_Image" FOR OUTPUT AS 1←
    PRINT "Writing to file       : ";FileIFF$+"_Image
"←
    GOSUB CreateImageFile←
  CLOSE 1←
  GOSUB DisplayBOB←
  END←
←
Initialize:←
  WINDOW 1,"TranslateIFF",(Ø,136)-(45Ø,186),15←
  DEFINT a-z←
  DIM R!(31),G!(31),B!(31)←
  True=-1←
  False=Ø←
  INPUT "IFF file to translate: ",FileIFF$←
  RETURN←
←
TranslateChunks:←
  ckID$=INPUT$(4,1)←
  SkipData$=INPUT$(4,1)←
  ckType$=INPUT$(4,1)←
  IF ckID$<>"FORM" OR ckType$<>"ILBM" THEN←
    PRINT "File is not a FORM ILBM chunk."←
    STOP←
  END IF←
  FoundBMHD=False←
  FoundCMAP=False←
  WHILE True←
    ckID$=INPUT$(4,1)←
    ckLength&=CVL(INPUT$(4,1))←
    IF ckID$="BMHD" THEN←
      GOSUB TranslateBMHD←
      FoundBMHD=True←
    ELSEIF ckID$="CMAP" THEN←
      GOSUB TranslateCMAP←
      FoundCMAP=True←
    ELSEIF ckID$="BODY" THEN←
      IF FoundBMHD AND FoundCMAP THEN←
        GOSUB TranslateBODY←
        RETURN←
      ELSE←
```

```
        PRINT "Context chunks are missing."←
        STOP←
      END IF←
    ELSE←
      SkipData$=INPUT$(ckLength&,1)←
      SkipData$=""←
    END IF←
    IF ckLength& MOD 2 THEN←
      SkipData$=INPUT$(1,1)←
    END IF←
  WEND←
  ←
TranslateBMHD:←
  Wide&=CVI(INPUT$(2,1))←
  Height&=CVI(INPUT$(2,1))←
  SkipData$=INPUT$(4,1)←
  Depth&=ASC(INPUT$(1,1))←
  Masking=ASC(INPUT$(1,1))←
  Compression=ASC(INPUT$(1,1))←
  SkipData$=INPUT$(1,1)←
  TransColor=CVI(INPUT$(2,1))←
  SkipData$=INPUT$(2,1)←
  PageWidth=CVI(INPUT$(2,1))←
  PageHeight=CVI(INPUT$(2,1))←
  ScnMode=PageWidth/320+2*(PageHeight/200-1)←
  PlanePick=2^Depth&-1←
  IF Masking<>2 THEN←
    PRINT "Unknown masking technique used."←
    STOP←
  END IF←
  IF Compression=0 THEN←
    FileCompressed=False←
  ELSEIF Compression=1 THEN←
    FileCompressed=True←
  ELSE←
    PRINT "Unknown compression technique used."←
    STOP←
  END IF←
  IF TransColor<>0 THEN←
    PRINT "Register zero is not the transparent colo
r."←
    STOP←
  END IF←
  Header$=MKL$(0)+MKL$(0)+MKL$(Depth&)+MKL$(Wide&)+M
KL$(Height&)←
  Header$=Header$+MKI$(24)+MKI$(PlanePick)+MKI$(0)←
  RETURN←
  ←
TranslateCMAP:←
  ColorCount=ckLength&/3-1←
  FOR Register=0 TO ColorCount←
```

```
    R!(Register)=INT(ASC(INPUT$(1,1))/12)/20←
    G!(Register)=INT(ASC(INPUT$(1,1))/12)/20←
    B!(Register)=INT(ASC(INPUT$(1,1))/12)/20←
  NEXT Register←
  RETURN←
←
TranslateBODY:←
  BytesPerRow=2*INT((Wide&+15)/16)←
  BytesPerPlane=BytesPerRow*Height&←
  ReqBytes=BytesPerPlane*Depth&←
  BitMap$=STRING$(ReqBytes,CHR$(0))←
  FOR RowNo=1 TO Height&←
    Pointer=1+BytesPerRow*(RowNo-1)←
    FOR PlaneNo=1 TO Depth&←
      Offset=BytesPerPlane*(PlaneNo-1)←
      IF FileCompressed THEN←
        Row$=""←
        WHILE LEN(Row$)<BytesPerRow←
          UByte=ASC(INPUT$(1,1))←
          ControlByte=UByte-2*(UByte AND 128)←
          IF ControlByte<-127 THEN←
            'No Operation←
          ELSEIF ControlByte<0 THEN←
            Row$=Row$+STRING$(-ControlByte+1,INPUT$(
1,1))←
          ELSEIF ControlByte<128 THEN←
            Row$=Row$+INPUT$(ControlByte+1,1)←
            END IF←
        WEND←
      ELSE←
        Row$=INPUT$(BytesPerRow,1)←
      END IF←
      MID$(BitMap$,Pointer+Offset,BytesPerRow)=Row$←
    NEXT PlaneNo←
  NEXT RowNo←
  RETURN←
  ←
CreateFieldFile:←
  PRINT#1,"DefinePlayField:"←
  PRINT#1,USING "  SCREEN 1_,###_,###_,#_,#";PageWid
th,PageHeight,Depth&,ScnMode←
  PRINT#1, "  WINDOW 1,";CHR$(34);FileIFF$;CHR$(34);
",,3,1"←
  FOR Register=0 TO ColorCount←
    PRINT #1,USING "  PALETTE ##_,#.##_,#.##_,#.##";
Register,R!(Register),G!(Register),B!(Register)←
  NEXT Register←
  PRINT #1,"  RETURN"←
  RETURN←
  ←
```

```
CreateImageFile:←
  PRINT #1,Header$+BitMap$;←
  RETURN←
  ←
DisplayBOB:←
  SCREEN 1,PageWidth,PageHeight,Depth&,ScnMode←
  WINDOW 2,"DisplayBOB",,Ø,1←
  FOR Register=Ø TO ColorCount←
    PALETTE Register,R!(Register),G!(Register),B!(Re
gister)←
  NEXT Register←
  PRINT "Hit any key to return to BASIC."←
  PRINT "Manipulate BOB with mouse pointer."←
  OBJECT.SHAPE 1,Header$+BitMap$←
  OBJECT.X 1,Ø←
  OBJECT.Y 1,16←
  OBJECT.ON←
  WHILE INKEY$=""←
    IF MOUSE(Ø)<>Ø THEN←
      OBJECT.X 1,MOUSE(1)-Wide&←
      OBJECT.Y 1,MOUSE(2)-Height&←
    END IF←
  WEND←
  OBJECT.OFF←
  WINDOW 1←
  SCREEN CLOSE 1←
  RETURN←
  ←
```

# IFF to Icon Translator

Charles L. Baker

*Design your own Workbench icons using programs such as* Deluxe Paint, Aegis Images, *and* Graphicraft. *Requires Workbench 1.2.*

"IFF to Icon" lets you customize your Workbench icons by translating IFF image files into Workbench .info files. IFF (Interchange File Format) is a standard file structure developed jointly by Electronic Arts and Commodore-Amiga. Most commercial drawing programs for the Amiga store images according to the IFF standards.

In order for a file's icon to appear on the Workbench screen, there must be a corresponding .info file. The Preferences program, for example, has the Preferences.info file associated with it. Drawers, the Trashcan (a special type of drawer), and even disks use .info files to describe what their icons look like. By modifying .info files, we can redefine Workbench icons.

## Creating an Image

Before you can use this program, you must create an image file with *Deluxe Paint* or any other IFF-compatible program. This article will describe what you must do when using *Deluxe Paint;* other programs use a similar process. Once you have created the image file, you can run IFF to Icon to convert your image into a Workbench icon.

Before you create the image, set the drawing program to the type of screen which your Workbench uses—either medium-resolution or high-resolution (interlace). The icon's colors are ultimately determined by the Workbench and not the drawing program used to design the icon. You may use the Preferences program to change the Workbench colors. Remember that the Workbench only uses four colors. After you have drawn the desired image, it must be saved as a brush. To select an image as a brush within *Deluxe Paint,* click the left mouse button on the brush-selection tool and drag a selection box around the image. To save a brush file, select Save or Save As from the Brush menu.

Exit the drawing program. At this point, the image has been saved as a brush file on disk. To minimize disk swapping, you should copy the brush file to your BASIC work disk.

## Getting Started

Type in and save IFF to Icon. The program uses the system library file named icon.bmap. In order for Amiga Basic to use this library, it must have a file description of the library in a form which it understands. This form is called a *.bmap file.* The .bmap file is essentially a list of pointers that allow Amiga Basic to access library routines.

The file icon.bmap must be created before you can run IFF to Icon. If you have version 1.2 of the Amiga operating system (available as an inexpensive upgrade from any Amiga dealer), you can create icon.bmap quite easily. The BASICDemos disk for 1.2 contains a BASIC program named ConvertFD, as well as a directory named FD1.2. Run the ConvertFD program and enter the following information when prompted:

Enter name of .fd file to read >
  **Extras:fd1.2/icon_lib.fd**
Enter name of .bmap file to produce >
  **Libs:icon.bmap**

When the ConvertFD program is finished, the disk contains the icon.bmap file. Copy this file onto the same disk as the IFF to Icon program. When IFF to Icon is run, the icon.bmap file must be either in the current directory or in the directory named LIBS (LIBrarieS) on the disk used when you booted the system. The LIBS directory is a good place for .bmap files, since their purpose is to give you access to libraries. If you don't have the .bmap files in the correct place, BASIC will stop with a *file not found* error when you run IFF to Icon.

## Using the Program

Run IFF to Icon. The program asks you to enter the name of the IFF file to translate and the name of the .info file to modify. You must specify the disk and folder in which the programs are located. Do not include the .info extension when entering the second filename. The program does this for you. If you wish to change the trashcan's icon, for example, simply enter the filename TRASHCAN. The IFF to Icon program does not create new .info files; it modifies existing ones. So, the .info file must already exist on disk. After both filenames have been entered, IFF to Icon translates the IFF image, creates a temporary image file of its own, and finally modifies the specified .info file. To convert IFF image files, this program uses code from the IFF Translator program.

Even after modification, a file's original icon will stay on the Workbench screen until the file, drawer, or disk is closed and redrawn. For files and drawers, this means closing and reopening the window that the icon is located in. If you modified a disk's icon, you must close all drawers and windows from the disk, remove the disk from the drive, and reinsert the disk after the original icon has disappeared. In some cases, you may have to reboot in order to remove the original disk icon from the Workbench screen.

### IFF to Icon Translator

Filename: IFFICON

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'  Copyright 1987 COMPUTE! Publications, Inc.<
'  All Rights Reserved<
'<
'  IFF to Icon<
'<
DECLARE FUNCTION GetDiskObject& LIBRARY<
DECLARE FUNCTION PutDiskObject& LIBRARY<
DECLARE FUNCTION FreeDiskObject& LIBRARY<
<
LIBRARY "icon.library"<
<
MainLine:<
 GOSUB InitiaLize<
 OPEN FiLeIFF$ FOR INPUT AS 1<
 PRINT "Reading from file    : ";FiLeIFF$<
 GOSUB TransLateChunks←
 CLOSE 1←
 OPEN FiLeIFF$+"_Image" FOR OUTPUT AS 1←
 PRINT "Writing to file       : ";FiLeIFF$+"_Image"←
 GOSUB CreateImageFiLe←
 CLOSE 1←
 OPEN FiLeIFF$+"_Image" FOR INPUT AS 1←
 PRINT "Translating file      : ";FiLeIFF$+"_Image"←
 GOSUB TransLateImage←
 CLOSE 1←
 KILL FiLeIFF$+"_Image"←
 PRINT "Modifying file        : ";FiLeInfo$+".info"←
 GOSUB CreateInfoFiLe←
 LIBRARY CLOSE←
END←
←
InitiaLize:←
 WINDOW 1,"Brush To Icon",(Ø,12Ø)-(45Ø,186),15←
 PRINT "Copyright 1987 COMPUTE! Publications, Inc."←
 PRINT "           All Rights Reserved.":PRINT←
 DEFINT a-z←
 DIM R1(31), G1(31), B1(31)←
 true = -1←
 faLse = Ø←
 INPUT "IFF file to translate: ",FiLeIFF$←
 INPUT ".info file to modify : ",FiLeInfo$←
RETURN←
←
TransLateChunks:←
 ckID$=INPUT$(4,1)←
 SkipData$=INPUT$(4,1)←
 ckType$=INPUT$(4,1)←
 IF ckID$ <> "FORM" OR ckType$ <> "ILBM" THEN←
   PRINT "File is not a FORM ILBM chunk."←
   STOP←
 END IF←
 FoundBMHD = faLse←
 FoundCMAP = faLse←
 WHILE true←
  ckID$=INPUT$(4,1)←
  ckLength&=CVL(INPUT$(4,1))←
  IF ckID$="BMHD" THEN←
    GOSUB TransLateBMHD←
    FoundBMHD = true←
  ELSEIF ckID$="CMAP" THEN←
    GOSUB TranslateCMAP←
    FoundCMAP = true←
  ELSEIF ckID$="BODY" THEN←
   IF FoundBMHD AND FoundCMAP THEN←
     GOSUB TransLateBODY←
     RETURN←
   ELSE<
     PRINT "Context chunks are missing."<
     STOP<
   END IF<
   ELSE<
    SkipData$=INPUT$(ckLength&,1)<
    SkipData$=""<
    END IF<
    IF ckLength& MOD 2 THEN<
       SkipData$=INPUT$(1,1)<
    END IF<
 WEND<
<
TransLateBMHD:<
 Wide&=CVI(INPUT$(2,1))<
 Height&=CVI(INPUT$(2,1))<
 SkipData$=INPUT$(4,1)<
 depth&=ASC(INPUT$(1,1))<
 Masking=ASC(INPUT$(1,1))<
 Compression=ASC(INPUT$(1,1))<
 SkipData$=INPUT$(1,1)<
 TransCoLor=CVI(INPUT$(2,1))<
 SkipData$=INPUT$(2,1)<
 PageWidth=CVI(INPUT$(2,1))<
 PageHeight=CVI(INPUT$(2,1))<
 ScnMode=PageWidth/320+2*(PageHeight/200-1)<
 pLanepick=2^depth&-1<
 IF Masking<>2 THEN<
    PRINT "Unknown masking technique used."<
    STOP<
 END IF<
 IF Compression = 0 THEN<
    FiLeCompressed = faLse<
  ELSEIF Compression = 1 THEN<
   FiLeCompressed = true<
  ELSE<
    PRINT "Unknown compression technique used."<
    STOP<
  END IF<
 IF TransCoLor <>0 THEN<
   PRINT "Register zero is not the transparent color
."<
   STOP<
 END IF<
 Header$=MKL$(0)+MKL$(0)+MKL$(depth&)+MKL$(Wide&)+MK
L$(Height&)<
 Header$=Header$+MKI$(24)+MKI$(pLanepick)+MKI$(0)<
RETURN<
<
```

```
TranslateCMAP:←
 CoLorCount=ckLength&/3-1←
 FOR register=Ø TO CoLorCount←
  R1(register)=INT(ASC(INPUT$(1,1))/12)/2Ø←
  G1(register)=INT(ASC(INPUT$(1,1))/12)/2Ø←
  B1(register)=INT(ASC(INPUT$(1,1))/12)/2Ø←
 NEXT register←
RETURN←
←
TransLateBODY:←
   BytesPerRow = 2*INT((Wide&+15)/16)←
   BytesPerPLane = BytesPerRow*Height&←
   ReqBytes = BytesPerPLane*depth&←
   BitMap$=STRING$(ReqBytes,CHR$(Ø))←
   FOR RowNo=1 TO Height&←
   pointer=1+BytesPerRow*(RowNo-1)←
   FOR PLaneNo=1 TO depth&←
   Offset=BytesPerPLane*(PLaneNo-1)←
    IF FiLeCompressed THEN←
    Row$=""←
    WHILE LEN(Row$)<BytesPerRow←
    UByte=ASC(INPUT$(1,1))←
    ControLByte=UByte-2*(UByte AND 128)←
    IF ControLByte<-127 THEN←
    ' No operation←
    ELSEIF ControLByte<Ø THEN←
    Row$=Row$+STRING$(-ControLByte+1,INPUT$(1,1))←
    ELSEIF ControLByte<128 THEN←
    Row$=Row$+INPUT$(ControLByte+1,1)←
    END IF←
    WEND←
    ELSE←
    Row$=INPUT$(BytesPerRow,1)←
    END IF←
    MID$(BitMap$,pointer+Offset,BytesPerRow)=Row$←
    NEXT PLaneNo←
   NEXT RowNo←
RETURN←
←
CreateImageFiLe:←
  PRINT#1, Header$+BitMap$;←
RETURN←
←
TransLateImage:←
 garbage$     = INPUT$(8,1)         ' throw away colors
et and dataset←
 depth&       = CVL(INPUT$(4,1))    ' depth of screen i
n bitmaps←
 bwidth&      = CVL(INPUT$(4,1))    ' width of screen i
n pixels←
 bheight&     = CVL(INPUT$(4,1))    ' height of screen
```

```
in pixels←
 garbage$     = INPUT$(2,1)        ' drop masking flag
s←
 pLanepick%  = CVI(INPUT$(2,1))←
 pLaneonoff% = CVI(INPUT$(2,1))←
 bit$         = INPUT$(LOF(1)-26,1)  ' bitplane data←
RETURN←
←
CreateInfoFiLe:←
 diskobj& = GetDiskObject&(SADD(FiLeInfo$))←
 IF diskobj& = Ø THEN←
  PRINT "error opening ";FiLeInfo$;".info"←
  GOTO ending←
 END IF←
 POKEL diskobj& + 8, Ø     ' top corner at Ø,Ø←
 POKEW diskobj& + 12, bwidth&←
 POKEW diskobj& + 14, bheight&←
 imageptr& = PEEKL(diskobj& + 22)←
 POKEW imageptr& + 4,  bwidth&←
 POKEW imageptr& + 6,  bheight&←
 POKEW imageptr& + 8,  depth&←
 POKEL imageptr& + 1Ø, SADD(bit$)←
 POKE  imageptr& + 14, pLanepick%←
 POKE  imageptr& + 15, pLaneonoff%←
 erro&=PutDiskObject&(SADD(FiLeInfo$), diskobj&)←
 IF erro& = Ø THEN←
   PRINT "error on file writing "←
 END IF←
 erro&=FreeDiskObject&(diskobj&)←
 IF erro& <> Ø THEN←
   PRINT "error on memory clearing --> ";erro&←
 END IF←
ending:←
RETURN←
←
```

# Math Draw

Rhett Anderson

*Create complex geometric patterns with this menu-driven graphics program for the Amiga. When you've finished, you can save your picture as an IFF graphics file for use with* Deluxe Paint II *and other Amiga paint programs. Requires 512K.*

"Math Draw" shows you how to create intricate geometric designs and patterns, even if you don't understand the mathematics behind them. Use the special menu to select from a palette of 32 different colors and several different drawing options. When you've finished your picture, you can save it in IFF ILBM format, which is used by most paint programs.

Math Draw is written in Amiga Basic. Type it in and save it to disk. When you're ready to use the program, load and run it.

Math Draw starts drawing immediately. After you've seen the pattern it draws, activate the Amiga menus by pressing the right mouse button. Math Draw adds an Option menu to Amiga Basic's menus. While holding the right button, slide the mouse pointer up to the word Option to see the new menu options. To choose an option, release the button while pointing to the desired option. Here is a list of the new options:

- **Color Control.** When you activate this function, you'll be presented with a chart which shows 32 colors, each labeled with a number. Choose a new drawing color by entering a number. The 32 colors are fixed; they cannot be changed from within the program. If you wish to change the colors, you'll need to modify the numbers in the first set of DATA statements in the program.
- **Radius Control.** Math Draw works by moving an imaginary circle inside or outside another imaginary circle. The drawing pen is "attached" to the edge of the moving circle. You change the size of these circles (and thus the scale of your

drawing) by choosing the Radius Control menu selection. You'll be asked for new radius values for the circles.

- **Growth Control.** If you choose, the imaginary circles used to move the drawing pen can grow at a fixed rate. This feature can be used to create spirals and shell designs. Normally, you'll want to keep this parameter at its default value, 0. A reasonable growth rate is 0.01. A negative number will make the circles shrink.
- **Type Control.** This option lets you choose between drawing epicycloids and hypocycloids. In an epicycloid, the moving circle moves around the outside of the stationary circle. In a hypocycloid, the moving circle moves around the inside of the stationary one.
- **Pause.** This option halts drawing. After selecting this option, the drawing pen freezes. This is useful when you want to make several changes to the pen at once. For instance, you may want to change the pen color and the outer radius at the same time. Select Pause and then use the menu options to select the new color and the new radii.
- **Continue.** Ends Pause mode and starts the pen moving again.
- **Save Picture.** Saves your picture as an IFF file compatible with *Deluxe Paint II, Digi-Paint, Aegis Images,* and other Amiga paint programs. Since the save routine is written entirely in BASIC, it takes a few minutes to save a picture. Be sure not to drop any menus down while the picture is being saved, or else you'll see ghosts of the menus in your saved pictures.
- **Clear.** Clears the screen. Since your work could be lost if you accidentally select this option, you'll be asked whether you're sure you want to clear the screen.
- **Quit.** Exits to BASIC cleanly. You should always use this menu option to stop the program. If you quit another way, type MENU RESET to disable the Math Draw menu and SCREEN CLOSE 1 to free up the screen created by Math Draw. Since your work could be lost if you select this option accidentally, you'll be asked whether you're sure you want to quit.

### Your First Drawing

Let's create a picture. First, select Pause to freeze the drawing pen. Now select Clear to erase any picture that might be on the screen. Select Color Control and pick a color. Now select Radius Control. Type 30 at the first prompt and 8 at the second. Select Continue, and a pattern will be drawn. When the pattern begins to repeat, select Pause again and then change the color. Change the radii to 50 and 10. Repeat this process until you're satisfied with your picture. If you wish to save it, select the Save option.

When you're ready for a new effect, turn on the Growth control. A moderate value for Growth is 0.01, but try values like 0.001, 1, and 10 to see how Growth works.

Table 6-2 has six sets of parameters that produce attractive designs. After seeing what they do, try combining them.

**Table 6-2. Example Designs**

| Radius 1 | Radius 2 | Growth | Type |
|---|---|---|---|
| 120 | 35 | 0 | 2 |
| 75 | 25 | −.1 | 1 |
| 10 | 45 | 0 | 2 |
| 80 | 46 | 0 | 2 |
| 30 | 16 | 0 | 1 |
| 40 | 16 | 0 | 1 |

After you have saved a couple of pictures, go to a paint program and fill in different areas with different colors. From within the paint program, you can modify the palette colors if you like.

### Math Draw

Filename: MATHDRAW

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
'Math Draw←
'Copyright 1988←
'COMPUTE! Publications, Inc.←
'All Rights Reserved←
←
CLEAR 32767←
DEFSNG a-z : DEFDBL g,h,theta←
TRUE% = -1 : FALSE% =Ø : PI = 3.141593 : speed = PI/
1ØØ←
```

```
DIM cmap$(31)←
←
colours:←
 DATA Ø,Ø,Ø, 1Ø,1Ø,1Ø, 6,Ø,Ø, 9,Ø,Ø, 12,Ø,Ø, 15,Ø,Ø←
 DATA Ø,6,Ø, Ø,9,Ø, Ø,12,Ø, Ø,15,Ø, Ø,Ø,6, Ø,Ø,9←
 DATA Ø,Ø,12, Ø,Ø,15, 4,4,Ø, 7,7,Ø, 1Ø,1Ø,Ø, 13,13,Ø
←
 DATA 4,Ø,4, 7,Ø,7, 1Ø,Ø,1Ø, 13,Ø,13, 6,3,Ø, 8,4,Ø←
 DATA 1Ø,5,Ø, 12,6,Ø, 4,8,12, 8,4,12, 4,12,8, 12,8,4
←
 DATA 12,4,8, 8,12,4←
 ←
ON BREAK GOSUB NoMore←
BREAK ON←
SCREEN 1,32Ø,2ØØ,5,1:WINDOW 3,"",(Ø,Ø)-(311,186),16,
1:WINDOW OUTPUT 3←
RESTORE colours←
FOR i%=Ø TO 31←
 READ r,g,b←
 PALETTE i%,r/15,g/15,b/15←
 cmap$(i%)=CHR$(r*17)+CHR$(g*17)+CHR$(b*17)←
NEXT i%  ←
MENU 3,Ø,1,"Options"←
MENU 3,1,1,"Color Control"←
MENU 3,2,1,"Radius Control"←
MENU 3,3,1,"Growth Control"←
MENU 3,4,1,"Type Control"←
MENU 3,5,1,"Save Picture"←
MENU 3,6,1,"Pause"←
MENU 3,7,1,"Continue"←
MENU 3,8,1,"Clear"←
MENU 3,9,1,"Quit"←
ON MENU GOSUB checkmenu←
a=1ØØ:b=2Ø:type%=1:colour=1Ø:deltaA=Ø:deltaB=Ø←
starting = TRUE% : pause = FALSE%←
nextpoint:←
MENU ON←
WHILE pause:WEND←
MENU STOP←
 g = (a+b)/b : h = (a-b)/b←
 theta = theta+speed←
 a = a+deltaA←
 b = b+deltaB←
 oldx = x: oldy = y←
 IF type% = 1 THEN GOSUB Epicycloid ELSE GOSUB Hypoc
ycloid←
 IF NOT starting THEN LINE (16Ø+oldx,1ØØ+oldy*.88)-(
16Ø+x,1ØØ+y*.88),colour←
 starting = FALSE%←
GOTO nextpoint←
```

```
←
Epicycloid:←
 x = (a+b)*COS(theta)-b*COS(g*theta)←
 y = (a+b)*SIN(theta)-b*SIN(g*theta)←
RETURN←
←
Hypocycloid:←
 x=(a-b)*COS(theta)+b*COS(h*theta)←
 y=(a-b)*SIN(theta)-b*SIN(h*theta)←
RETURN ←
←
checkmenu:←
 menuid=MENU(Ø)←
 menuitem=MENU(1)←
 IF menuid=3 THEN←
  IF menuitem = 1 THEN←
   WINDOW 4,"",(Ø,Ø)-(26Ø,82),18,1:WINDOW OUTPUT 4←
   FOR xx%=Ø TO 7←
    FOR yy%=Ø TO 3←
     LINE (xx%*32+16,yy%*16+8)-(xx%*32+24,yy%*16+16)
,xx%+yy%*8,bf←
     LOCATE yy%*2+2,xx%*4+1←
     PRINT RIGHT$(STR$(xx%+yy%*8),2);←
    NEXT yy%←
   NEXT xx%←
   LOCATE 1Ø,4←
   INPUT "Choose a Color";colour : colour=colour MOD
 32←
   WINDOW OUTPUT 3←
   WINDOW CLOSE 4←
  END IF←
  IF menuitem = 2 THEN←
   WINDOW 4,"",(Ø,Ø)-(28Ø,65),18,1:WINDOW OUTPUT 4←
   PRINT "Current Radius":PRINT" of Stationary Circl
e:";a←
   INPUT "New Radius";a←
   PRINT "Current Radius":PRINT" of Moving Circle:";
b←
   INPUT "New Radius";b←
   a=ABS(a):b=ABS(b)←
   IF a=Ø THEN a=1←
   IF b=Ø THEN b=1←
   starting = TRUE%←
   WINDOW OUTPUT 3←
   WINDOW CLOSE 4←
  END IF←
  IF menuitem = 3 THEN←
   WINDOW 4,"",(Ø,Ø)-(3ØØ,5Ø),18,1:WINDOW OUTPUT 4←
   PRINT "Current Growth Rate:";deltaA←
   INPUT "New Rate";deltaA:IF a<>Ø THEN deltaB=delta
A*b/a←
```

```
  starting = TRUE%←
  WINDOW OUTPUT 3←
  WINDOW CLOSE 4←
 END IF←
 IF menuitem = 4 THEN←
  WINDOW 4,"",(Ø,Ø)-(25Ø,5Ø),18,1:WINDOW OUTPUT 4←
  PRINT "Choose a Type:"←
  PRINT "1. Epicycloid (Outside)"←
  PRINT "2. Hypocycloid (Inside)"←
  PRINT:PRINT"Current Type =";type%←
  INPUT "New Type";type%←
  IF type% = 2 THEN speed = PI/5Ø ELSE speed = PI/1
ØØ←
    starting = TRUE%←
    WINDOW OUTPUT 3←
    WINDOW CLOSE 4←
   END IF←
   IF menuitem = 5 THEN←
    WINDOW 4,"",(Ø,Ø)-(3ØØ,4Ø),18,1:WINDOW OUTPUT 4←
    INPUT "Filename For Save";fil$←
    WINDOW OUTPUT 3 : WINDOW CLOSE 4←
    IF fil$<>"" THEN GOSUB savepic←
   END IF←
   IF menuitem = 6 THEN pause = TRUE%←
   IF menuitem = 7 THEN pause = FALSE%←
   IF menuitem = 8 THEN←
    WINDOW 4,"",(Ø,Ø)-(25Ø,5Ø),18,1:WINDOW OUTPUT 4←
    PRINT "Are You Sure You Want":PRINT" To Erase the
 Screen?"←
    INPUT a$←
    WINDOW OUTPUT 3 : WINDOW CLOSE 4←
    IF UCASE$(LEFT$(a$,1))="Y" THEN CLS : starting =
TRUE     ←
   END IF←
   IF menuitem = 9 THEN←
    WINDOW 4,"",(Ø,Ø)-(25Ø,5Ø),18,1 : WINDOW OUTPUT 4
←
    PRINT "Are You Sure You Want To Quit?"←
    INPUT a$←
    IF UCASE$(LEFT$(a$,1))="Y" THEN GOTO NoMore←
    WINDOW OUTPUT 3 : WINDOW CLOSE 4 ←
   END IF ←
  END IF ←
RETURN←
←
savepic:←
 rastport& = WINDOW(8)←
 bitmap& = PEEKL(rastport&+4)←
 topLine = Ø←
 topadd = topLine * 4Ø←
 FOR i% = Ø TO 4←
  pLane&(i%) = PEEKL(bitmap& + 8 + i%*4)+topadd←
 NEXT←
 bottomLine = 2ØØ←
 Lines = bottomLine - topLine←
 OPEN fil$ FOR OUTPUT AS 1←
 a$ = MKL$(Lines * 4Ø * 5 + 144)←
 PRINT#1,"FORM";a$;"ILBMBMHD";MKL$(2Ø);←
 PRINT#1,MKI$(32Ø);MKI$(Lines);MKL$(Ø);←
 PRINT#1,CHR$(5);MKI$(Ø);CHR$(Ø);←
 PRINT#1,MKI$(Ø);CHR$(1Ø);CHR$(11);←
 PRINT#1,MKI$(32Ø);MKI$(2ØØ);←
 PRINT#1,"CMAP";MKL$(96);←
 FOR i% = Ø TO 31←
  PRINT#1,cmap$(i%);←
 NEXT←
 PRINT#1,"BODY";MKL$(Lines * 4Ø * 5);←
 FOR i% = 1 TO Lines←
  FOR p% = Ø TO 4←
   FOR j% = Ø TO 39 STEP 4←
    PRINT#1,MKL$(PEEKL(pLane&(p%) + j%));←
   NEXT j%←
   pLane&(p%) = pLane&(p%) + 4Ø←
  NEXT p%←
 NEXT i%←
 CLOSE←
 PALETTE Ø,1,1,1:SOUND 44Ø,2Ø,2ØØ,Ø:PALETTE Ø,Ø,Ø,Ø←
RETURN←
←
NoMore:←
 WINDOW CLOSE 3 : WINDOW CLOSE 4 : SCREEN CLOSE 1←
 MENU RESET←
 MENU OFF←
 END←
←
```

# 3-D Surfaces

Martin Staley

*Written entirely in Amiga Basic, this graphically impressive program allows you to plot three-dimensional shapes on the screen in any color combination you like. By making small changes, you can view the object from any vantage point or plot an entirely different graph.*

One of the most popular traditional applications for computer graphics is to plot three-dimensional graphs on the screen. That description may sound dull, but the resulting shapes are often quite beautiful in their own right as well as educational. The Amiga's outstanding graphics capabilities and fast processing speed make it ideal for such activity.

"3-D Surfaces" provides a convenient, powerful tool for anyone interested in creating such pictures. It draws 3-D graphs as *mesh perspectives*. That is, the shapes appear as rectangular grids that have been pushed up or down in various places to create a variety of different shapes. The program permits you to change many different aspects of the picture, including the fineness of the mesh, screen resolution, observation angle, low and high bounds of the function that creates the picture, and, of course, the function itself.

Type in and save the program. Since the program requires quite a bit of memory, it's best not to run any other programs while it's in operation.

## Using the Program

The program begins by computing all the data it needs to plot the current function. This process can take awhile, depending on the complexity of the shape. To inform you of its progress, the program prints a counter value on the screen. When the calculations are complete, the program draws the shape on the screen.

Once the shape is finished, you can change any of the screen colors by moving the color sliders in the upper left corner of the screen with the mouse pointer. To move a slider, place the mouse pointer on the slider, hold down the left mouse button, and then move the slider to the desired spot.

You can stop the program if necessary by selecting the *Quit* option from the *Actions* menu. This option automatically restores the original palette colors and closes the hi-res screen for your convenience.

## Creating New Shapes

This program is designed to give you great flexibility in plotting your own 3-D pictures. Apart from color changes (see above), this is done by changing one or more of the parameters defined at the beginning of the program. The best way to learn about these parameters is to experiment on your own. All of the controlling parameters are located immediately following the labels *Parameters* and *Equation*. If you're familiar with this type of activity, the comments in these lines may give you enough information to plot your own graphs. The remainder of this article discusses in more detail the significance and use of these parameters.

## Change the Equation

Each image created by this program is a two-dimensional representation of an equation or a mathematical function. It is the equation, more than any other factor, which controls the ultimate appearance of the graph. It's defined with the DEF FN statement in the line immediately after the label *Equation*. DEF FN, as you may know, creates a user-defined function for later use in the program in which it appears. To change the function, simply replace the portion on the right side of the equal sign (=). The result can be an entirely new shape. Here are some interesting functions to try:

```
←
←
←
(x^2+5*y^2)*EXP(1-x^2-y^2)/2-SIN(3*x^2y^2)/(x^2+y^2)
←
←
-x^3/10-(SIN(1-x^2-y^2)+COS(1-x^2-y^2))/2←
←
SIN(3*x)*SIN(3*y)/5+.7*SIN(2*x^2+3*y^2)/(x^2+y^2)←
←
COS(3*x)+2*SIN(x^2+y^2)/(x^2+y^2)-x/2←
←
.3*(SIN(x^2+y)+COS(y^2+x))←
←
(SIN(4*x^2+y^2)+2*SIN(x*y))/(4*x^2+y^2)←
←
SIN(3*x)+SIN(3*y)←
←
←
←
```

In each case, the new function definition should be substituted for the portion of the DEF FN statement that lies on the right side of the equal sign. For instance, to use the last example definition, the line following the label *Equation* should read as follows:

**DEF FNz(x,y)=SIN(3*x)+SIN(3*y)**

## The Plot Thickens

The first two variables in the *Parameters* section, *m* and *n*, control the number of grid rectangles in the *x* (horizontal) and *y* (vertical) directions. Simply put, these values control the fineness of the rectangular mesh of which the graph is composed. If you increase the value of *m* and/or *n*, the plot appears thicker and more finely detailed. The finer the resolution, the better the graph looks. However, more detailed plots take longer to create. Conversely, smaller values make the graph look coarser and less substantial. The coarser the mesh, the less time it takes to complete the necessary calculations. Setting both values to 31 is a reasonable tradeoff between time and accuracy.

Since the program utilizes two 2-dimensional arrays based on *m* and *n*, the values of these two variables are limited by the amount of available memory. On a 512K Amiga, I've used values as high as 75. At this degree of accuracy, the program requires about ten minutes for calculations; however, the results are worth it.

The values of *m* and *n* need not be equal. However, they should be set to an odd number. Both of these points are discussed in more detail below.

## Resolution

The next variable, *res,* controls the screen resolution. If *res* equals 1, the program draws the graph on Amiga Basic's default 640 × 200 output window. If you set *res* to 2, the program opens a custom output window in 640 × 400 resolution before it draws. The memory requirements of this window make it unusable on a 256K Amiga. Graphs drawn in the lower resolution always look coarser than those drawn in the highest resolution, particularly when the mesh size is small. However, even lower resolution screens look quite good.

## Accuracy

The variable *gt* stands for *graph type*. It controls the accuracy of the plot by selecting one of two drawing algorithms (formulas). The first algorithm draws a good estimate of the shape. The second algorithm draws the shape in actual, exact perspective from any direction, angle, and distance. Each method has advantages and disadvantages. The estimate method is less complex, more reliable, and faster. The exact perspective method is slower and requires many more intensive calculations (which can lead to error messages on rare occasions). However, drawing in exact perspective allows you to view a shape from different observation points. The estimate method causes some inaccuracy in the vertical scale, but exact perspective uses correct proportions, taking into account the fact that pixels (dots) on the Amiga screen are square, not round.

## Aspect and Height

Two of the parameter variables are used only with the estimate drawing method (see preceding section). The variable *asp* controls the apparent *x-y* ratio of the graph as it appears on the screen, regardless of the bounds you specify. Aspects that are too large or too small (say, larger than 4 or smaller than ¼) have the side effect of downgrading the quality of the estimate (the graph may look slightly distorted). The variable *h* controls the height factor, which affects the graph's vertical appearance. In general, height factors of less than 100 tend to make the apparent observation point higher in the *z* direction; as a result, graphs look a bit stubbier than expected. Larger height factors have the opposite effect (lower observation points and taller graphs). By enlarging the height factor, you can emphasize a graph's vertical qualities.

## Observation Angle and Distance

The graph's perspective is controlled by three parameter variables: *theta, phi,* and *d*. The variable *theta* equals the observation angle from the *x-y* axis moving counterclockwise in the *x-y* plane as viewed from the positive *z* direction. The variable *phi* is the observation angle with respect to the *x-y* plane. This variable is set up for both angles to be in degrees; if you would rather use radians, remove the conversions in the second program line under the label *Equations*. Any observation angle is possible if you keep *theta* in the range −180 to 180 and keep *phi* in the range −90 to 90. Other values may be used; however, it's usually best to keep the angle more than about one-tenth degree away from any positive or negative multiple of 90 degrees (including 0) to avoid overflow errors in the computation. Such extreme observation angles aren't very interesting, anyway, since you tend to lose most of the graph's three-dimensional quality.

The variable *d* controls the distance of the observation point—in the direction of the direction angles—from the graph's center (the point whose coordinates are the average *x*, *y*, and *z* coordinates of all the computed function values). The only formal restriction for *d* is that it cannot be 0. However, it should be large enough to place you a reasonable distance from the shape. Observing the graph from an extremely close location is a bit like viewing the Mona Lisa by putting your eye one millimeter away from the canvas. In addition, extremely small values for *d* can actually locate the observation point "inside" the graph. The program assumes that all graph points are within a 180-degree field of view while looking toward the center. If *d* is so close to the center that not all of the graph's points are within this view, the program's output is garbage. It's best to make *d* large enough so that the observation point is beyond the bounds of the function as specified by the four parameters discussed in the next section. Incidentally, specifying a very large distance won't make the graph look significantly smaller. As the distance becomes larger, perspective qualities such as the presence of a vanishing point become less pronounced. To avoid wasting screen resolution, the program always stretches the perspective until either the horizontal or vertical dimension becomes too large to fit on the screen.

## Bounds

The next four parameter variables set the low and high bounds of the graph in the *x* and *y* dimensions. This simply means that the four sides of the graph will be along those edges.

## Equation Notes

The most important parameter, of course, is the equation contained in the DEF FN statement. When defining new functions, keep in mind that the computer can't perform some operations, such as dividing by 0 or taking the square root of a negative number. However, functions which have what's known as a *limiting value* on the interval can usually be plotted. There are many rational functions whose numerators become 0 at the same time their denominators reach 0; and the ratio can be finite. But the computer doesn't know this and still generates a *Division by zero* error unless it just misses the coordinate in question.

To compute function values, the program increments between the low and high *x* bounds, and between the low and high *y* bounds, in step sizes such that a total of $m+1$ different *x* values and $n+1$ different *y* values are eventually put in the equation. If the increment sizes and the low and high bounds are such that the offending point is skipped, everything should work correctly. Odd values for *m* and *n* seem to work best, but problems are still rare when even values are used.

For some equations, the CLEAR,60000 statement in the second program line may cause an *Out of memory* error. You may be able to avoid this error by reducing the value in the CLEAR statement. That change reduces the amount of space available for BASIC arrays and variables, which may make it necessary to decrease the value of *m* and/or *n* as well.

## 3-D Surfaces

Filename: 3DSURF

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
<
'  3D Surfaces<
'  Copyright 1987 COMPUTE! Publications, Inc.<
'  All Rights Reserved<
<
   CLEAR,60000&:DEFINT i,j<
   pi=3.1415927#:e=2.7182818#<
Parameters:      <
   m=31:n=31              ' mesh size = m*n<
   res=1                  ' resolution: 1=640*200, 2=6
40*400<
   gt=1                   ' graph type: 1=estimate, 2=
real     <
   asp=1                  ' y/x ratio of graph (only f
or gt=1)<
   h=150                  ' height factor      (only f
or gt=1)<
   theta=30 :phi=20       ' observation angles in degr
ees (only for gt=2)<
   d=100                  ' distance from graph center
    (only for gt=2)<
   lox=-3 :hix=3          ' low & high bounds in x dir
ection<
   loy=-3 :hiy=3          ' low & high bounds in y dir
ection<
Equation:   <
   DEF FNz(x,y)=SIN(x^2+3*y^2)/(x^2+y^2)+(x^2+5*y^2)
```

```
*EXP(1-x^2-y^2)/2 ←
   theta=theta*pi/180:phi=phi*pi/180  ' conversion:
degrees to radians←
   a=d*COS(phi)*COS(theta)←
   b=d*COS(phi)*SIN(theta)←
   c=d*SIN(phi)      ←
   GOSUB Check←
   DIM xc(m+1,n+1),yc(m+1,n+1)  ←
   tx=(hix-lox)/m:ty=(hiy-loy)/n←
   PRINT:PRINT "computing values..."←
   LOCATE 4,7:PRINT m+1   ←
   IF gt=1 THEN GOSUB Estimate:ELSE GOSUB True←
Draw:←
   IF res=2 THEN SCREEN 1,640,400,2,4:WINDOW 2,"grap
h",,15,1:ELSE CLS ←
   DIM v(15),rgb(3,2)←
   p=0←
Rc:               ←
   FOR c=0 TO 3          ←
      READ r,g,b:PALETTE c,r,g,b←
      rgb(c,0)=r:rgb(c,1)=g:rgb(c,2)=b←
   NEXT c          ←
   DATA 0,0,0,.14,.14,.14,0,0,0,0,0,1←
   COLOR 2,3←
   FOR x=dfx TO dtx  STEP sx←
      FOR y=dfy TO dty STEP sy←
         x1=xc(x,y):x2=xc(x,y+1):x3=xc(x+1,y+1):x4=x
c(x+1,y)←
         y1=yc(x,y):y2=yc(x,y+1):y3=yc(x+1,y+1):y4=y
c(x+1,y)←
         AREA (x1,y1):AREA (x2,y2):AREA (x3,y3):AREA
 (x4,y4):AREAFILL←
         LINE (x1,y1)-(x2,y2),3:LINE (x2,y2)-(x3,y3)
,3←
         LINE (x3,y3)-(x4,y4),3:LINE (x4,y4)-(x1,y1)
,3  ←
      NEXT y←
   NEXT x   ←
Colors: ←
   FOR n=0 TO 15:v(n)=11+7.54*n:NEXT n←
   col=0:x1=11:x2=11:x3=11←
   IF res=1 THEN WINDOW 3,"colors",(0,0)-(130,50),18
,-1←
   IF res=2 THEN WINDOW 3,"colors",(0,0)-(130,50),18
,1←
   WINDOW OUTPUT 3←
   PRINT "R":PRINT "G":PRINT "B":PRINT:PRINT "C"←
   LINE (0,0)-(130,0):LINE (0,8)-(130,8)←
   LINE (0,16)-(130,16):LINE (0,24)-(130,24)←
   LINE (10,0)-(10,50):LINE (11,25)-(40,50),0,bf←
   LINE (40,25)-(70,50),1,bf:LINE (70,25)-(100,50),2
,bf←
```

```
   LINE (1ØØ,25)-(13Ø,5Ø),3,bf:LINE (11,1)-(19,7),3,
bf   ←
   LINE (11,9)-(19,15),3,bf:LINE (11,17)-(19,23),3,b
f ←
   i=2Ø:LINE (2Ø,32)-(3Ø,42),1,bf←
   MENU 1,Ø,1,"Action":MENU 1,1,1,"Quit":MENU ON←
Loop:←
   IF MENU(Ø)=1 AND MENU(1)=1 THEN←
      WINDOW CLOSE 3:MENU RESET:SCREEN CLOSE 1:STOP←
   END IF←
   x=MOUSE(1):y=MOUSE(2):IF MOUSE(Ø)>=Ø THEN c1=1:c2
=1:c3=1:GOTO Loop←
   IF x>1Ø AND x<131 THEN IF y>24 AND y<51 THEN Getc
ol:ELSE GOTO Check1←
   GOTO Loop←
Check1:←
   arg=.126Ø5Ø4*x-1.386551:IF y<1 OR y>7 OR c1=Ø THE
N Check2←
   LINE (11,1)-(13Ø,7),Ø,bf:LINE (v(arg),1)-(v(arg)+
8,7),3,bf:x1=v(arg)←
   rgb(col,Ø)=arg/15:PALETTE col,rgb(col,Ø),rgb(col,
1),rgb(col,2)←
   c1=1:c2=Ø:c3=Ø:GOTO Loop←
Check2:←
   IF y<9 OR y>15 OR c2=Ø THEN Check3←
   LINE (11,9)-(13Ø,15),Ø,bf:LINE (v(arg),9)-(v(arg)
+8,15),3,bf:x2=v(arg)←
   rgb(col,1)=arg/15:PALETTE col,rgb(col,Ø),rgb(col,
1),rgb(col,2)←
   c1=Ø:c2=1:c3=Ø:GOTO Loop←
Check3:←
   IF y<17 OR y>23 OR c3=Ø THEN Loop←
   LINE (11,17)-(13Ø,23),Ø,bf:LINE (v(arg),17)-(v(ar
g)+8,23),3,bf:x3=v(arg)←
   rgb(col,2)=arg/15:PALETTE col,rgb(col,Ø),rgb(col,
1),rgb(col,2)←
   c1=Ø:c2=Ø:c3=1:GOTO Loop←
Getcol:←
   LINE (i,32)-(i+1Ø,42),col,bf←
   IF x<4Ø  THEN i=2Ø :LINE (i,32)-(3Ø,42),1,bf :col
=Ø:GOTO Nst←
   IF x<7Ø  THEN i=5Ø :LINE (i,32)-(6Ø,42),2,bf :col
=1:GOTO Nst←
   IF x<1ØØ THEN i=8Ø :LINE (i,32)-(9Ø,42),3,bf :col
=2:GOTO Nst←
                  i=11Ø:LINE (i,32)-(12Ø,42),Ø,bf:col
=3←
Nst:   ←
   LINE (11,1)-(13Ø,23),Ø,bf:LINE (1Ø,8)-(13Ø,8):LIN
E (1Ø,16)-(13Ø,16)←
   c=1←
   FOR n=Ø TO 2:tl=111*rgb(col,n)+11←
   LINE (tl,c)-(tl+8,c+6),3,bf:c=c+8←
   NEXT n←
   GOTO Loop←
Check:    ←
   m=INT(m):n=INT(n)←
   asp=ABS(asp):h=ABS(h)←
   IF res<>1 AND res<>2 THEN res=1←
   IF res=1 THEN ht=186:hht=93 ←
   IF res=2 THEN ht=386:hht=193←
   IF lox>hix THEN SWAP lox,hix←
   IF loy>hiy THEN SWAP loy,hiy←
   dfx=1:dtx=m:sx=1:dfy=1:dty=n:sy=1←
   IF gt<>2 THEN gt=1←
   IF res<>2 THEN res=1←
   IF gt=2 THEN←
      IF a<Ø THEN dfx=m:dtx=1:sx=-1←
      IF b<Ø THEN dfy=n:dty=1:sy=-1←
   END IF←
   RETURN      ←
Estimate:←
   m1=31Ø/m:m2=16Ø/m:n1=31Ø/n:n2=16Ø/n:rd=18Ø/pi←
   x=24Ø/SQR(1+asp^2):y=24Ø*asp/SQR(1+asp^2)←
   spx=31Ø+.8886207*(x-y):spy=hht-.4586429*(x+y)←
   x1=1.777241*x/m:x2=1.777241*y/n:y1=.9172858*x/m:y
2=.9172858*y/n  ←
   i=Ø:x=lox-tx←
   WHILE i<m+1←
      i=i+1  :LOCATE 5,7:PRINT i;←
      x=x+tx :j=Ø:y=loy-ty←
      WHILE j<n+1←
         j=j+1←
         y=y+ty                 ←
         xc(i,j)=(spx+x2*j-x1*i)←
         yc(i,j)=spy+y2*j+y1*i-h*FNz(x,y)←
         IF yc(i,j)<smin THEN smin=yc(i,j)←
         IF yc(i,j)>smax THEN smax=yc(i,j)←
      WEND←
   WEND←
   IF smax<ht AND smin>Ø THEN RETURN←
   avg=(smax+smin)/2:smax=smax-avg:smin=smin-avg:mul
t=ht/(smax-smin)←
   FOR x=1 TO m+1←
      FOR y=1 TO n+1←
         yc(x,y)=mult*(yc(x,y)-avg)+hht←
      NEXT y←
   NEXT x  ←
   RETURN←
True:←
   DEF FNc(a,b,c,x,y,z)=(x*(b*(b-y)+c*(c-z))+(x-a)*(
b*y+c*z))/d←
```

```
    DEF FNang(x,y,z)=(px*x+py*y+pz*z)/(dp*SQR(x^2+y^2
+z^2))          ←
    px=FNc(a,b,c,Ø,Ø,1Ø):py=FNc(b,a,c,Ø,Ø,1Ø):pz=FNc(
c,b,a,1Ø,Ø,Ø)←
    dp=SQR(px^2+py^2+pz^2)←
    i=Ø:x=lox-tx←
    WHILE i<m+1←
       i=i+1  :LOCATE 5,7:PRINT i←
       x=x+tx :j=Ø:y=loy-ty←
       WHILE j<n+1←
          j=j+1←
          y=y+ty                      ←
          xc(i,j)=FNz(x,y):sum=sum+xc(i,j)←
       WEND←
    WEND←
    avg=sum/((m+1)*(n+1)):ym=loy-ty-(loy+hiy)/2←
    i=Ø:x=lox-tx-(lox+hix)/2←
    WHILE i<m+1←
       i=i+1  :LOCATE 6,7:PRINT i←
       x=x+tx :j=Ø:y=ym←
       WHILE j<n+1←
          j=j+1:y=y+ty:z=xc(i,j)-avg ←
          d=a*(a-x)+b*(b-y)+c*(c-z)←
          xc=FNc(a,b,c,x,y,z)←
          yc=FNc(b,a,c,y,x,z)←
          zc=FNc(c,b,a,z,y,x)←
          rad=SQR(xc^2+yc^2+zc^2)←
          s=1←
          IF SGN(a)<>SGN(yc*pz-zc*py) THEN ←
             s=-1←
          ELSEIF SGN(b)<>SGN(zc*px-xc*pz) THEN ←
             s=-1←
          ELSEIF SGN(c)<>SGN(xc*py-yc*px) THEN ←
             s=-1←
          END IF←
          cs=FNang(xc,yc,zc):sn=SQR(1.ØØØØ1-cs^2)←
          xc(i,j)=s*rad*sn:yc(i,j)=-rad*cs          ←
          IF xc(i,j)>xmax THEN xmax=xc(i,j)←
          IF xc(i,j)<xmin THEN xmin=xc(i,j)←
          IF yc(i,j)>ymax THEN ymax=yc(i,j)←
          IF yc(i,j)<ymin THEN ymin=yc(i,j)←
       WEND←
    WEND←
    ax=(xmax+xmin)/2:ay=(ymax+ymin)/2←
    IF res=1 THEN←
       hzy=93←
       IF ((ymax-ymin)/(xmax-xmin))>(6.75/1Ø.25) THEN
←
          my=168/(ymax-ymin):mx=168/(ymax-ymin)*2.2ØØ
899←
       ELSE←
```

```
          my=6Ø2/(xmax-xmin)/2.2ØØ899:mx=6Ø2/(xmax-xm
in)←
        END IF←
     ELSE   ←
        hzy=193←
        IF ((ymax-ymin)/(xmax-xmin))>(6.875/1Ø.25) THE
N←
          my=368/(ymax-ymin):mx=368/(ymax-ymin)*1.Ø92
Ø89←
        ELSE←
          my=6Ø2/(xmax-xmin)/1.Ø92Ø89:mx=6Ø2/(xmax-xm
in)←
        END IF←
     END IF←
     FOR x=1 TO m+1←
        FOR y=1 TO n+1←
          xc(x,y)=315+mx*(xc(x,y)-ax)←
          yc(x,y)=hzy+my*(yc(x,y)-ay)←
        NEXT y←
     NEXT x   ←
     RETURN←
          ←
```

# CHAPTER SEVEN

# Programming

# System Fonts

Daniel L. Stockton

*This short program illustrates how to select different system fonts and text styles under program control in Amiga Basic.*

Nearly every Amiga owner recognizes that there are two system text fonts. If you open the Preferences tool, you can switch between the 80-column font, actually called Topaz Eight Point, or the 60-column font, called Topaz Nine Point. The font which you select from Preferences is carried over when you activate Amiga Basic. Many Amiga Basic programs are written with the assumption that you have selected one font or the other—either Topaz Eight or Topaz Nine. If you run a program only to find that printing is not aligned correctly, the cause is usually that you're using the wrong font. The only remedy is to exit BASIC, open Preferences, select the other font, and run the program all over again.

In fact, there's no need to make users play "guess the font" when they run Amiga Basic programs. With a relatively simple system call, you can select either of the system fonts under program control. And the same system call can be used to select special text styles, including boldface, italics, and underlining. The subprogram included with this article makes it all easy to do.

## Opening the Library

To change fonts under program control, you must make a call to operating system routines. A group of system routines is known as a *library;* the particular library which we need is called *graphics.library*. The door to the library is opened with the BASIC statements LIBRARY and CALL.

Type in the program and save it. Before you run the program, you must make sure that a special system file is on your BASIC work disk. When BASIC executes the LIBRARY command in this program, the Amiga looks for a file named *graphics.bmap* which enables it to locate graphics.library routines. If graphics.bmap can't be found, the computer displays an error box.

Users of Workbench 1.2 should follow the procedure outlined in Appendix B for creating the graphics.bmap file.

The simplest way for users of Workbench 1.1 to make sure that graphics.bmap can be found is to copy it from the BasicDemos drawer (subdirectory) of your Extras disk to the LIBS subdirectory of your BASIC work disk. Enter these commands to perform the copy from the CLI:

```
COPY Extras:BasicDemos/graphics.bmap TO ram:
COPY ram:graphics.bmap TO Workbench:libs
DELETE ram:graphics.bmap
```

These commands assume that your BASIC work disk is named *Workbench*. If it is not, substitute your disk name for *Workbench* in the second AmigaDOS command. (If the disk name contains a space, you must enclose that portion of the command in quotation marks. For example, if your work disk is named *BASIC Programs*, then the command will be COPY ram:graphics.bmap TO "BASIC Programs:libs".) When it encounters LIBRARY in a program, the computer automatically searches the LIBS subdirectory for the designated file.

Another way to ensure access to the .bmap file is to use CHDIR before the LIBRARY statement. CHDIR can be used to specify a particular subdirectory as the current directory for DOS access. If you create a subdirectory devoted exclusively to .bmap files, then every program can perform a CHDIR to that directory before attempting a LIBRARY command. One advantage of the second method is that it avoids unnecessary duplication of .bmap files in a case where you keep BASIC programs in more than one subdirectory on a disk.

## Styles

Once graphics.bmap is located in the correct place, load and run the program. It prints eight different text styles in each of the two system fonts. The variable *style%* controls which style is selected. Table 7-1 contains explanations of the different values assigned to that variable.

**Table 7-1. The Values for the *style%* Variable**

| *style%* Value | Result |
|---|---|
| 0 | plain text |
| 1 | underlined |
| 2 | boldface |
| 3 | boldface, underlined |
| 4 | italics |
| 5 | italics, underlined |
| 6 | italics, boldface |
| 7 | italics, boldface, underlined |

The subprogram named FontChange is designed to be merged with your own programs. Once you know that the demonstration works correctly, you can delete everything but the FontChange subprogram and save it in ASCII form (type SAVE "FontChange",A and press RETURN). You can then add it to another program with a MERGE command.

## System Fonts

Filename: SYSFONTS

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
' Copyright 1987 Compute! Publications, Inc.←
' All Rights Reserved←
←
  REM The next three lines must be in the main body
of your program←
  REM and occur before any calls to subroutine FontC
hanger←
←
DECLARE FUNCTION AskSoftStyle& LIBRARY←
DECLARE FUNCTION OpenFont& LIBRARY←
  REM  Place optional CHDIR command here to point to
 the←
  REM  graphics.bmap file directory as next line exa
mple:←
  REM  CHDIR "ABC:D"←
  REM  which points to volume/disk ABC directory D←
LIBRARY "graphics.library"←
  ←
  REM  The format for a CALL to FontChange is:←
  REM        CALL FontChange ("topaz.font",height%,st
yle%)←
  REM        "topaz.font" is the name of the system f
ont←
  REM        height% is a short integer variable to b
```

```
e selected←
  REM           as 8 or 9←
  REM      style% is a short integer variable to be
 selected ←
  REM           as Ø through 7 ←
     ←
  REM  The nine lines of Testit below are for testin
g only←
  REM  and should be replaced by your application(Th
e main body←
  REM  of your program).←
  ←
Testit:←
  CLS←
  FOR height%=8 TO 9←
    FOR style%=Ø TO 7←
      CALL FontChange ("topaz.font",height%,style%)←
      PRINT  "This is the topaz font of height ";hei
ght%;" style ";style%←
    NEXT style%←
  NEXT height%←
  CALL FontChange ("topaz.font",8,Ø)←
  REM The library should be closed when no longer re
quired←
        ←
LIBRARY CLOSE←
END←
←
  REM  The subroutine FontChange should be placed at
 the bottom←
  REM  of your program after the END statement←
  ←
SUB FontChange(font$,height%,style%) STATIC←
  SHARED pFont&←
  IF pFont&<>Ø THEN CALL CloseFont(pFont&)←
  fontA$=font$+CHR$(Ø)←
  textAttr&(Ø)=SADD(fontA$)←
  textAttr&(1)=height%*65536& + style%*256 ←
  pFont&=OpenFont&(VARPTR(textAttr&(Ø)))←
  IF pFont& <> Ø THEN SetFont WINDOW(8),pFont&←
  permit%=AskSoftStyle&(WINDOW(8))←
  SetSoftStyle WINDOW(8),style%,permit%←
END SUB←
←
←
←
←
```

# Disk-Based Fonts

Daniel L. Stockton

*"Disk-Based Fonts" is an expansion of the previous article "System Fonts." Here we explore Amiga fonts even further. Also, you'll learn techniques to create your own customized DISKFONT.BMAP file. The accompanying program shows how to load custom text fonts from disk and use them from Amiga Basic.*

In the Utilities drawer of the Amiga Workbench disk is a useful tool called Notepad. One attractive feature of this mini–word processor is its ability to use a variety of text fonts. This article explains how to use those same fonts—or any disk-based text font—in Amiga Basic. Amiga Basic does not provide any direct means of loading a custom font from disk. However, this can be accomplished by calling system routines which are used by the computer itself.

## Where Are My Fonts?

The fonts used by Notepad are located in the Fonts directory of the Workbench disk. You can use the FILES command to get a listing of that directory. Type this command in the BASIC output window and press Return:

```
files "Workbench:Fonts"
```

Each font in the Fonts directory has its own subdirectory which contains the various sizes for that font. Font sizes are specified in units called *points,* which are equal to 1/72 inch. Thus, a 9-point font has characters 9/72 inch in size, and so on. The different text styles (italicized, boldface, and so on) are not stored in the font directory; these styles are generated by selectively distorting the shapes found in the basic font file.

The program included with this article can select any of the disk-based fonts, with eight styles for each font. Table 7-2 lists the fonts, and Table 7-3 lists the various styles. Version

1.2 of the Amiga operating system adds a few new sizes to existing fonts.

**Table 7-2. Font Sizes**

| Fonts | Sizes |
|---|---|
| Topaz | 8, 9, 11 |
| Ruby | 8, 12, 15 |
| Diamond | 12, 20 |
| Opal | 9, 12 |
| Emerald | 17, 20 |
| Garnet | 9, 16 |
| Sapphire | 14, 19 |

**Table 7-3. Font Styles**

| Number | Style |
|---|---|
| 0 | plain text (Workbench default) |
| 1 | underline |
| 2 | boldface |
| 3 | boldface and underline |
| 4 | italics |
| 5 | italics and underline |
| 6 | italics and bold |
| 7 | italics, bold, and underline |

The program uses two system libraries named graphics.library and diskfont.library. In order for Amiga Basic to use these libraries, it must have a file description of the library in a form which it understands. This form is called a *.bmap file*. The .bmap file is essentially a list of pointers that allow BASIC to access library routines.

Before you can run Disk-Based Fonts, you must make sure that the correct .bmap files are present on the same disk as Disk-Based Fonts. The first such file, graphics.bmap, is included in the BASICDemos drawer of the Amiga Extras disk. The second file, named diskfont.bmap, must be created.

If you have version 1.2 of the Amiga operating system (available as an inexpensive upgrade from any Amiga dealer), you can create diskfont.bmap quite easily. The BASICDemos disk for 1.2 contains a BASIC program named ConvertFd, as well as a directory named FD1.2. Run the ConvertFd program, using the file named diskfont.lib_fd. When the program is finished, the disk contains diskfont.bmap. If you haven't obtained the 1.2 upgrade, type in and run Program 2 from the article "Banner Printer," page 149. That program reads values from DATA statements, checks them for accuracy, and creates diskfont.bmap on the current disk.

Before you run the program Disk-Based Fonts, make sure it is on the same disk as both graphics.bmap and diskfont.bmap. The location of these files is important. They must be either in the current directory *or* in the directory named LIBS on the disk used when you booted the system. The LIBS (LIBrarieS) directory is a good place for .bmap files, since their purpose is to give you access to libraries. If you don't have the .bmap files in the correct place, BASIC will stop with a *File not found* error.

When you run the program, it displays the various fonts on the screen in different sizes and waits for you to click the mouse button before proceeding to the next font.

The most important part of the program is contained in the subprogram named FontSelect, which appears at the end of the program. After you have tested the program and have saved a copy, delete everything in the program except the FontSelect subprogram. Then save the subprogram under a new name as an ASCII file, so that it can be MERGEd with other programs. To save a program in ASCII form, add the characters *,a* to the end of a normal save command. For instance, to save the subprogram with the name FontSelect, you would type this command in the BASIC output window and press Return:

**save "FontSelect",a**

The FontSelect subprogram is invoked with a CALL statement in the main program. Three items of information are passed to FontSelect in the form of variables.

The first variable is a string named *font$*. This string contains the name of the font you wish to use (Garnet, Ruby, and so on). If the string is a null string ("", a string containing no characters), only a style change occurs.

The second and third variables passed to FontSelect are numeric variables of the short integer type. The variable *height%* defines the point size that you wish to use (see Table 7-2), and *style%* defines the style to use (Table 7-3).

## Opening and Closing Libraries

A few additional statements are needed to prepare for the CALL to FontSelect and to clean up afterward. The preparatory statements are grouped together at the beginning of the program, immediately after the first two REMark statements. The DEFLNG statement causes all simple variables to default to the long integer type. (Note that this declaration is overridden by the short integer type specifier attached to *height%* and *style%*.)

The LIBRARY and DECLARE FUNCTION statements actually give you access to library routines. These statements should appear in the initialization section of the program, before the first CALL to the FontSelect subprogram.

When the program is about to terminate, you should take some additional steps to close the fonts and the libraries. The CALL to the CloseFont function closes any fonts that were previously opened. (A bug in version 1.1 of the operating system prevents this CALL from working correctly. Version 1.2 corrects the bug. If you have version 1.1, omit the line containing the CALL to CloseFont or put a REM in front of the line.)

The final CALL to FontSelect resets the font to the system font, Topaz. While not absolutely necessary, it's considered good manners for programs which change the computer environment to restore the original environment as closely as possible before terminating.

The LIBRARY CLOSE statement closes libraries that were previously opened. If you omit these final housekeeping chores, the computer may not crash, but the libraries will remain open, wastefully occupying memory which would otherwise be freed for other tasks.

The program module named TestSection uses another system routine named *text* when printing words in boldface and italicized styles. This method prevents the characters in those words from overlapping, as they would if you printed the words with PRINT.

## Disk-Based Fonts

Filename: DISKFONTS

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
REM Copyright 1987 COMPUTE! Publications, Inc.←
REM All Rights Reserved←
REM Program  SelectFont←
REM Provides for use of Amiga disk based fonts from
Amiga Basic←
PRINT "Copyright 1987 COMPUTE! Publications, Inc."←
PRINT "All Rights Reserved"←
DEFLNG a-z        'all variables default to long int
eger←
 'Include optional CHDIR command here (CHDIR ":BMAPS
")←
LIBRARY "diskfont.Library"←
LIBRARY "graphics.Library"←
DECLARE FUNCTION OpenDiskFont LIBRARY←
DECLARE FUNCTION AskSoftStyLe LIBRARY←
 'the above commands must be placed in the main body
 of your program←
 ←
TestSection:←
  BREAK ON←
  ON BREAK GOSUB Housekeeping←
  READ t1$,t2$←
  FOR i=0 TO 6  'look at 7 fonts←
    READ font$,height%←
    CALL FontSeLect(font$,height%,styLe%)←
    FOR styLe%=0 TO 4 STEP 2   'look at 3 styles eac
h←
    CALL FontSeLect("",0,styLe%)←
      IF styLe%=0 THEN←
        LOCATE 1,1:PRINT  "This is the";height%;"poi
nt ";font$;" font"←
        PRINT  "Click left mouse button for the next
 font"←
        PRINT t1$:PRINT t2$←
      ELSEIF styLe%=2 THEN←
        a$="YOU ARE LOOKING AT BOLD STYLE"←
        PRINT←
        CALL text(WINDOW(8),SADD(a$),LEN(a$)):PRINT←
      ELSEIF styLe%=4 THEN←
        a$="THIS IS ITALICS STYLE"←
        PRINT←
        CALL text(WINDOW(8),SADD(a$),LEN(a$)):PRINT←
      END IF←
    NEXT styLe%←
      ←
    WaitForMouse:←
    IF NOT MOUSE(Ø) THEN WaitForMouse←
    CLS←
     'include the following CALL statement when usin
g workbench 1.2←
     'as it closes fonts and frees memory ←
     'CALL CloseFont(WINDOW(8),fontptr) ←
  NEXT i←
←
  Housekeeping:←
  CALL FontSeLect("topaz",8,Ø)  'return to default s
ystem font←
  LIBRARY CLOSE←
  END←
    ←
  DATA ABCDEFGHIJKLMNOPQRSTUVWXYZØ123456789!@#$%←
  DATA abcdefghijklmnopqrstuvwxyz^&*()=+\!/?<[{←
  DATA topaz,9,ruby,12,diamond,12,opal,11←
  DATA emerald,2Ø,garnet,16,sapphire,19←
REM End Of TestSection←
←
SUB FontSeLect(font$,height%,styLe%) STATIC  ←
  IF font$ <> "" THEN←
    textAttr(Ø)=SADD(font$+".font"+CHR$(Ø))←
    textAttr(1)=height%*65536&←
    fontptr=OpenDiskFont(VARPTR(textAttr(Ø)))←
    IF fontptr THEN SetFont WINDOW(8),fontptr←
  END IF←
  permited%=AskSoftStyLe&(WINDOW(8))←
  CALL SetSoftStyle (WINDOW(8),styLe%,permited%)←
END SUB←
←
←
←
```

# Zookeeper

Michael Barron

*Data for object images is stored in individual files—which can quickly lead to an unmanageable directory. This utility provides a solution: Programs not only initialize faster, but they are also easier to handle. Requires Workbench 1.2.*

The object commands in Microsoft Amiga Basic, which allow you to manipulate sprites, vsprites, and bobs, provide easy access to some of the Amiga's most powerful animation routines. However, you must store the data for each object image in its own, separate file. Thus, if your program uses six objects, you'll have to manage seven separate disk files (six object files, plus the program itself). Before long, your previously neat disk directory can start looking like a zoo.

"Zookeeper" offers a neat solution to this problem by converting image definition files into DATA statements which can be part of the main program. The DATA lines are organized into meaningful sections with commentary, bringing order to the object zoo.

Defining objects with DATA statements does make the program itself somewhat larger. But the advantages of this method definitely outweigh the drawbacks. Only one file need be duplicated when exchanging the program with a friend, and a printed listing of the program shows everything needed to make it run properly. The program also initializes faster, since it simply reads DATA statements already in memory, rather than seeking and loading separate files on disk. And, finally, it eliminates the problems that can occur when a program is not able to load the necessary object files.

## Using Zookeeper

Type in Program 1 and save a copy. Zookeeper can handle object definition files for both sprites and bobs, in the format produced by the object editor program on the Amiga Basic Extras disk. Let's demonstrate how it works using an example image file from the Extras disk.

Before we begin you'll have to create the exec.bmap file. See Appendix B for the complete instructions.

In the BASICDemos drawer of the Extras disk is an image definition file named ball. Copy that file onto the same disk (and directory, if applicable) where you saved the Zookeeper program.

Run Zookeeper and enter the appropriate filename at the prompt. The program then asks you to enter the maximum number of data elements for each DATA statement (eight is a good number). With this information, Zookeeper creates a text file consisting of commented DATA statements. The DATA lines contain the same information as the image definition file. This new file has the same filename as the one which you entered earlier, with the filename extension .ZOO. Thus, if you are converting the image file named ball, the new file is named ball.ZOO.

When the .ZOO file has been created, the Zookeeper program gives you the option to delete the original image definition file. Do not delete anything at this point.

## Demonstration

After you have created the ball.ZOO file, type in Program 2. This program will show you how easy it is to use the new image format. After you type in and save all the lines listed in Program 2, enter this command in the BASIC Output window:

```
MERGE "ball.ZOO"
```

The effect of this command is to merge the DATA lines from the ball.ZOO file with Program 2, which is already in memory. The DATA lines appear at the end of the current program.

Before you run the combined program, look at the subroutine named InitPlayer near the end of Program 2. That routine sets a variable named *ByteCount,* which is used to indicate how many items to read from the DATA statments. If you examine the comments at the beginning of the ball.ZOO data, you will see a comment indicating how many bytes this particular set of DATA lines contains.

Since we knew in advance that ball.ZOO would create 106 bytes of data, we were able to use that number in the statement from InitPlayer which sets *ByteCount*. For any other image, however, that value will be different. When using these routines to display your own objects, you will need to examine the comment at the beginning of the DATA lines and modify the statement in InitPlayer to match the number shown.

When you run the demonstration, notice how much faster the program begins. Not having to access the disk drive is a great advantage. Another advantage is easy accessibility to the image data itself. To see what we mean, find the twenty-second DATA number in the ball.ZOO data set, change that number from 24 to 48, and then rerun the program.

**Program 1. Zookeeper**
Filename: ZOOKEEPER

*For instructions on entering these programs, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
←
' ZooKeeper←
' Convert image definition files to DATA statements←
' Copyright 1987 COMPUTE! Publications, Inc.←
' All rights reserved.←
PRINT"Copyright 1987":PRINT"Compute! Publications, I
nc."←
PRINT"All Rights Reserved.":FOR X=1 TO 2000:NEXT X←
CLS←
MainLine:←
  GOSUB HouseKeeping←
  GOSUB ParameterEntry←
  GOSUB DefineFieLds←
  IF NoErrors THEN←
    GOSUB CreateZooFiLe←
    GOSUB FiLeMaintenance←
  END IF←
  GOSUB EndJob←
  END←
←
HouseKeeping:←
  DEFINT a-z←
  WINDOW 1,"The ZooKeeper",(0,56)-(500,186),15←
  TRUE= -1←
  FALSE= 0←
  HeaderBytes= 26←
  CoLorMapBytes= 6←
  RETURN←
←
ParameterEntry:←
  PRINT←
  INPUT "Enter name of AmigaBASIC object file: ",FiL
ename$←
  INPUT "Enter maximum number of data elements per s
tatement: ",MaxBytes←
  PRINT←
  RETURN←
←
DefineFieLds:←
  PRINT "Input file:  ";FiLename$←
  OPEN FiLename$ FOR INPUT AS 1←
    Image$=INPUT$(LOF(1),1)←
  CLOSE 1←
  Depth&= CVL(MID$(Image$,9,4))←
  Wide&= CVL(MID$(Image$,13,4))←
  Height&= CVL(MID$(Image$,17,4))←
  Flags= CVI(MID$(Image$,21,2))←
  BytesPerRow= 2*INT((Wide&+15)/16)←
  BytesPerPlane= BytesPerRow*Height&←
  BytesInBitmap= BytesPerPlane*Depth&←
  IF Flags AND 1 THEN←
    ObjectIsSprite= TRUE←
    ReqBytes= HeaderBytes+BytesInBitmap+CoLorMapByte
s←
  ELSE←
    ObjectIsSprite= FALSE←
    ReqBytes= HeaderBytes+BytesInBitmap←
  END IF←
  IF LEN(Image$)<>ReqBytes THEN←
    PRINT FiLename$;" is not compatible with the Zoo
Keeper."←
    NoErrors= FALSE←
  ELSE←
    NoErrors= TRUE←
  END IF←
  RETURN←
  ←
CreateZooFiLe:←
  PRINT "Output file: ";FiLename$;".ZOO"←
  PRINT←
  PRINT "Please wait..."←
  PRINT←
  OPEN FiLename$+".ZOO" FOR OUTPUT AS 1←
  PRINT # 1,←
  PRINT # 1, "ObjectData:"←
  IF ObjectIsSprite THEN←
    PRINT # 1, "' SPRITE Format"←
  ELSE←
```

```
    PRINT # 1, "' BOB Format"<
  END IF<
  PRINT # 1, "' Total Bytes: ";ReqBytes<
  PRINT # 1, "' Bit Planes:  ";Depth&<
  PRINT # 1, "' Pixels Wide: ";Wide&<
  PRINT # 1, "' Pixels Tall: ";Height&<
  CurrentByte= 1<
  Comment$= "' Object Header"<
  CALL FormatData(Comment$,MID$(Image$,CurrentByte,H
eaderBytes),MaxBytes,1)<
  CurrentByte= CurrentByte+HeaderBytes<
  FOR PLane=1 TO Depth&<
    Comment$= "' BitPlane "+ STR$(PLane)<
    CALL FormatData(Comment$,MID$(Image$,CurrentByte
,BytesPerPlane),MaxBytes,1)<
    CurrentByte= CurrentByte+BytesPerPlane<
  NEXT PLane<
  IF ObjectIsSprite THEN<
    Comment$="' Sprite Color Map"<
    CALL FormatData(Comment$,MID$(Image$,CurrentByte
,CoLorMapBytes),MaxBytes,1)<
  END IF<
  CLOSE 1<
  RETURN<
  <
FiLeMaintenance:<
  PRINT "Shall I delete ";FiLename$;" (y/n)";<
  INPUT Response$<
  IF UCASE$(Response$)= "Y" THEN<
    KILL FiLename$<
    PRINT FiLename$;" deleted."<
  END IF<
  PRINT<
  RETURN                <
  <
EndJob:<
  PRINT "Job complete: returning to AmigaBASIC."<
  PRINT<
  RETURN<
<
SUB FormatData(Note$,Dat$,DatLimit,FiLeNo) STATIC<
  PRINT # FiLeNo, Note$<
  NoBytes= LEN(Dat$)<
  NoFuLLLines= INT(NoBytes/DatLimit)<
  CurrentByte= 1<
  FOR LineOut= 1 TO NoFuLLLines<
    CALL PrintDataLine(MID$(Dat$,CurrentByte,DatLimi
t),FiLeNo)<
    CurrentByte= CurrentByte+DatLimit<
  NEXT LineOut<
  IF CurrentByte<= NoBytes THEN<
```

```
    BytesLeft= NoBytes-CurrentByte+1<
    CALL PrintDataLine(MID$(Dat$,CurrentByte,BytesLe
ft),FiLeNo)<
  END IF<
END SUB<
<
SUB PrintDataLine(Dat$,FiLeNo) STATIC<
  NoBytes= LEN(Dat$)<
  PRINT # FiLeNo, USING "  DATA   ###";ASC(MID$(Dat$
,1,1));<
  FOR ELement= 2 TO NoBytes<
    PRINT # FiLeNo, USING "_, ###";ASC(MID$(Dat$,ELe
ment,1));<
  NEXT ELement<
  PRINT # FiLeNo,<
END SUB<
```

**Program 2. Zookeeper Demonstration**
Filename: ZOOKEEPER.DEMO

```
<
' Zookeeper demonstration<
' Copyright 1987 COMPUTE! Publications, Inc.<
' All rights reserved<
PRINT"Copyright 1987":PRINT"Compute! Publications, I
nc."<
PRINT"All Rights Reserved.":FOR X=1 TO 2000:NEXT X<
CLS<
MainLine:<
  GOSUB HouseKeeping<
  GOSUB InitPLayer<
  WHILE INKEY$=""<
    IF MOUSE(0)<>0 THEN<
      OBJECT.X 1, MOUSE(1)<
      OBJECT.Y 1, MOUSE(2)<
    END IF<
  WEND<
  END<
  <
HouseKeeping:<
  DEFINT a-z<
  WINDOW 1,"Ball Demo",(0,136)-(450,186),15<
  PRINT "Manipulate object with mouse."<
  PRINT "Hit any key to exit."<
  RETURN<
<
InitPLayer:<
  ByteCount= 106<
  Image$= ""<
  FOR Loop= 1 TO ByteCount<
```

```
    READ ImageData←
    Image$=Image$+CHR$(ImageData)←
  NEXT Loop←
  OBJECT.SHAPE 1, Image$←
  OBJECT.ON←
  RETURN  ←
←
←
```

# SuperMenus

Rick Du Chateau

*Add new capabilities to Amiga Basic's MENU command with this powerful extension. Requires version 1.2 of Amiga Basic.*

Amiga Basic may be the most powerful BASIC interpreter available. However, while it does a reasonable job of providing access to the system software built into the Amiga, Amiga Basic doesn't support the system software as completely as compiled languages like C, Pascal, and Modula-2.

As an example, consider Amiga Basic's MENU command. With it, you can create custom menus and menu items, enable or disable the items, and place a check mark next to a menu item. But this is only a small fraction of the menu features and capabilities provided by Intuition. (Intuition is the portion of the Amiga operating system that controls the user interface—the windows, menus, and alert boxes.) Amiga Basic's MENU command doesn't provide for submenus, command keys, alternate ways of highlighting a selected menu, and the ability to turn a check mark on or off while excluding the other choices. For an example of how complex and useful menus can be, load *Deluxe Paint II* and wander through the menus and submenus.

Fortunately, the designers of Amiga Basic provided for these and future features by making Amiga Basic an extensible language through the use of subprograms and the LIBRARY command. By using LIBRARY, the Amiga Basic programmer has access to the multitude of the Amiga's operating system routines. And subprograms can actually add new commands to Amiga Basic.

"SuperMenus" illustrates the usefulness of subprograms by adding several menu-related commands to Amiga Basic. These new commands are INITIALIZE, SMENU, SUBMENU, and SMENUOFF. Adding SuperMenus to your own programs will give you access to the full power of the Amiga's menu system and will enable you to create more professional-looking programs. A short demonstration program is included that utilizes most of Supermenu's features.

## Getting Started

Type in and save Program 1. This is the SuperMenus routine. You will want to be able to merge this routine into programs you write yourself, so you must save the program in ASCII (text) format. This is accomplished by adding ,A to the end of the SAVE command:

```
SAVE "SuperMenus",A
```

To learn how to use the routine, type in and save a copy of Program 2, SuperMenus.Demo. Now go to the BASIC command window and type LOAD "SuperMenus.Demo"; then type MERGE "SuperMenus" to add the SuperMenus routine to the demonstration program. Note that Program 2 will not work unless you merge SuperMenus with it. If you failed to save Program 1 as an ASCII file, you'll see the error message *Bad file mode* when you give the MERGE command.

Run the resulting program. After a few seconds, you'll hear a beep and see the message *SuperMenus Ready!* Use the menu select button (the right mouse button) to view the various SuperMenu features. The demonstration will print any selections you make (when the right button is released). Select *Quit* to exit the program.

Follow these steps to incorporate SuperMenus into your own programs:

- After loading your Amiga Basic program, type MERGE "SuperMenus".
- Add the following lines to the beginning of your program:

```
DECLARE FUNCTION
  AllocMem&( ) LIBRARY
INITIALIZE
```

See the demonstration program for an example of how this is done.

- Follow the format described below to create a menu.
- Call the SUBITEM subprogram to test for a subitem selection (returned in variable called SubNum%).
- At the end of your program call the SMENUOFF subprogram.
- Save your program.
- Before running your program, be sure the files *exec.bmap* and *graphics.bmap* are in either your current directory or the Libs directory. Follow the procedure outlined in Appendix B for creating these files.

## The Commands

The first new command, INITIALIZE, should be used only once before the SMENU command. This subprogram initializes all of the variables used by SuperMenus and loads the necessary libraries from disk.

The SMENU command is the heart of SuperMenus. It's similar to Amiga Basic's MENU command. The format is

SMENU (*Menu%, Item%, SubMenu%, Flags%, MExclude&, CommandKey$, Text$, SelecText$*)

The items in italics in the SMENU command have the following meanings:

- *Menu%* is the menu position, in the range 1–10 (as in Amiga Basic MENU).
- *Item%* is the item position, in the range 1–19 (as in Amiga Basic MENU).
- *SubItem%* is the subitem position (no limit).
- *Flags%* is one or more of the flags listed in the table of flags.
- *MExclude&* is used in conjunction with the check% and toggle% flags in the table of flags. Choosing from Item1& through Item31& will exclude these items from selection when this item is active. For example, if you have a set of menu items 1, 2, and 3, and if you set MExclude&= Item2& + Item3& for the first menu item, then when item 1 is selected (indicated by a check mark), Intuition will erase any check marks on items 2 and 3. See the Amiga technical manuals for additional information.

- *CommandKey$* is an alphanumeric character to be used with the right Amiga key to select this item.
- *Text$* is the text to be used for this item.
- *SelecText$* is the text to be substituted for Text$ when this item is selected if you use the selectimage% highlighting flag.

Amiga Basic's MENU(0) and MENU(1) functions return the values of the selected menu and menu item respectively. So, in order to check for a selected subitem, a new command—SUBITEM—had to be added. SUBITEM returns a selected subitem number in the variable *SubNum%*.

The final new command, SMENUOFF, should be used at the end of your program to free up the memory used by Supermenus.

**Table 7-4. Available Flags**

| | |
|---|---|
| **You may select as many of these flags as you like:** | |
| *check%* | Puts a check mark to the left of the item or subitem |
| *text%* | Indicates that the item or subitem consists of text as opposed to a graphic image |
| *command%* | Uses this flag if this item or subitem is to have a command key associated with it |
| *toggle%* | Uses this flag in conjunction with *check%* to toggle the check mark on or off with selection |
| *enabled%* | Indicates the item or subitem is enabled as opposed to ghosted (off) |
| **You must select one and only one of the following highlighting flags:** | |
| *comp%* | Inverts the colors of an item or subitem during selection |
| *box%* | Draws a box around the item or subitem during selection |
| *selectimage%* | Uses a defined alternate image during selection |
| *none%* | Indicates no highlighting |

## Program 1. Supermenus

Filename: SUPERMENUS

*For instructions on entering these programs, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
' SuperMenus←
' Copyright 1987 COMPUTE! Publications, Inc.←
' All Rights Reserved.←
←
start:←
←
Initial:←
←
SUB INITIALIZE  STATIC←
  DEFLNG a-z←
  SHARED check%,text%,command%,toggLe%,enabLed%←
  SHARED image%,comp%,box%,none%←
  SHARED item1,item2,item3,item4,item5,item6,item7,i
tem8,item9,item10←
  SHARED item11,item12,item13,item14,item15,item16,i
tem17,item18,item19,item20←
  SHARED item21,item22,item23,item24,item25,item26,i
tem27,item28,item29,item30,item31←
       ←
     LIBRARY "exec.library"←
     LIBRARY "graphics.library"←
  ←
←
     'Flag Definitions←
      check%= 1←
      text%= 2←
      command%= 4←
      toggLe%= 8←
      enabLed%= 16←
      ←
      seLectimage%= 0←
      comp%= 64←
      box%= 128←
      none%= 192←
   ←
     'Mutual Exclude Flags←
      item1&= 1←
      item2&= 2←
      item3&= 2 ^ 2←
      item4&= 2 ^ 3←
      item5&= 2 ^ 4←
      item6&= 2 ^ 5←
      item7&= 2 ^ 6←
      item8&= 2 ^ 7←
      item9&= 2 ^ 8←
      item10&= 2 ^ 9←
      item11&= 2 ^ 10←
      item12&= 2 ^ 11←
      item13&= 2 ^ 12←
      item14&= 2 ^ 13←
      item15&= 2 ^ 14←
      item16&= 2 ^ 15←
      item17&= 2 ^ 16←
      item18&= 2 ^ 17←
      item19&= 2 ^ 18←
      item20&= 2 ^ 19←
      item21&= 2 ^ 20←
      item22&= 2 ^ 21←
      item23&= 2 ^ 22←
      item24&= 2 ^ 23←
      item25&= 2 ^ 24←
      item26&= 2 ^ 25←
      item27&= 2 ^ 26←
      item28&= 2 ^ 27←
      item29&= 2 ^ 28←
      item30&= 2 ^ 29←
      item31&= 2 ^ 30←
    ←
END SUB  ←
 ←
SMenus: ←
 ←
SUB SMENU(xpos%,ypos%,spos%,fLags%,mexcLude&,Key$,ti
tLe$,titLe2$) STATIC←
←
  SHARED Address&←
  ←
  DEFLNG a-z ←
←
  IF spos%=0 THEN←
  ←
  IF (fLags% AND 16) THEN active%= 1 ELSE active%= 0
     ←
  MENU xpos%,ypos%,active%,titLe$      ←
  staLL=TIMER:WHILE TIMER < (staLL + 1):WEND  'Give
AmigaBasic time to SetMenuStrip←
←
  GOTO SetKey←
  END IF←
 ←
 SetSPos:←
  ←
  Address= PEEKL(WINDOW(7) + 28)←
  IF xpos% >1 THEN CALL FindAddress(xpos%)←
  MenuAdd= Address←
  ←
  TextAttr=ALLocMem(12&,65539&)←
```

```
   CALL AskFont(WINDOW(8),TextAttr)<
   height%=PEEKW(TextAttr + 4)  'Find the height of c
urrent font.<
      <
      titLe$= titLe$ + CHR$(0)<
      wide%= LEN(titLe$) * 10<
      titLe= SADD(titLe$)<
       <
      Intuit= ALLocMem(20,65539&)<
      IF Intuit= 0 THEN PRINT "No Memory!!":STOP<
      <
      POKE Intuit + 1, 1  'Set BackPen to color regis
ter 1<
      <
      POKE Intuit + 2, 1  'Set DrawMode<
      POKEL Intuit + 12, titLe<
      <
      subitem= ALLocMem(34,65539&)<
      IF subitem= 0 THEN PRINT "No Memory!!":STOP<
      <
      POKEW subitem + 4,50  'Start the subitem 50 pix
els to the right of the Menu<
      POKEW subitem + 6,height% * spos%  'Figure the
top position<
      POKEW subitem + 8,wide%<
      POKEW subitem + 10,height%<
      POKEW subitem + 12,82<
      POKEL subitem + 18,Intuit<
      <
     Address= PEEKL(MenuAdd + 18)<
     IF ypos% >1 THEN CALL FindAddress(ypos%)<
     ItemAdd= Address<
     <
     IF spos% = 1 THEN Address= ItemAdd + 28:GOTO Ski
pIt<
     <
     Address= PEEKL(ItemAdd + 28)       <
     <
     IF spos% > 2 THEN<
     CALL FindAddress(spos% -1)<
     END IF<
<
SkipIt:<
     <
     POKEL Address,subitem<
<
     CALL FreeMem(TextAttr,12) <
     <
   SetKey:<
   <
     Address= PEEKL(WINDOW(7) + 28)    <
```

```
    IF xpos% > 1 THEN CALL FindAddress(xpos%)<
        <
    Address= PEEKL(Address + 18)<
    IF ypos% > 1 THEN CALL FindAddress(ypos%)<
    <
    IF spos% > Ø THEN<
    Address= PEEKL(Address + 28)<
    IF spos%= 1 THEN PokeKey<
   <
    CALL FindAddress(spos%)<
    END IF<
<
PokeKey:<
    <
    POKEW Address + 12,fLags%  'Set the Flags<
    POKEL Address + 14,mexcLude&<
    IF Key$ <> "" THEN POKE Address + 26,ASC(Key$)
'Poke in the Key<
<
    IF titLe2$ <> "" THEN<
<
     titLe2$= titLe2$ + CHR$(Ø)<
     wide%= LEN(titLe2$) * 1Ø<
     titLe= SADD(titLe2$)<
      <
     Intuit= ALLocMem(2Ø,65539&)<
     IF Intuit= Ø THEN PRINT "No Memory!!":STOP<
     <
     POKE Intuit + 1, 1  'Set BackPen to color regis
ter 1<
     <
     POKE Intuit + 2, 1  'Set DrawMode<
     POKEL Intuit + 12, titLe<
     POKEL Address + 22, Intuit<
     <
  END IF<
<
  OutaHere:<
    END SUB<
<
SUB FindAddress(Position%) STATIC<
  SHARED Address<
  <
      FOR counter%= 1 TO (Position% -1)<
      Address= PEEKL(Address)<
      NEXT          <
END SUB <
    <
TurnSmenuOff:<
<
SUB SMENUOFF STATIC<
```

```
 <
  MenuAdd= PEEKL(WINDOW(7) + 28)<
 <
Again:<
  IF MenuAdd= Ø THEN ALLDone<
  GOSUB ItemAddress<
  <
  MenuAddress:<
   MenuAdd= PEEKL(MenuAdd)<
   GOTO Again<
   <
ItemAddress:<
     IF PEEKL(MenuAdd + 18) = Ø THEN OutOfItems   <
<
       ItemAdd= PEEKL(MenuAdd + 18)<
       IF PEEKL(ItemAdd + 22) > Ø AND PEEKL(ItemAdd +
 22) <> PEEKL(ItemAdd + 22) THEN <
         Intuit= PEEKL(ItemAdd + 22)<
         CALL FreeMem(Intuit,2Ø)         'Free Select
IntuiText<
       END IF<
<
  KeepLooking:<
    <
       GOSUB LookForSub<
       ItemAdd= PEEKL(ItemAdd)<
       IF ItemAdd > Ø THEN KeepLooking<
OutOfItems:<
RETURN<
<
LookForSub:<
  IF PEEKL(ItemAdd + 28) = Ø THEN OutOfSubs<
    Address= PEEKL(ItemAdd + 28)<
<
  CLearMem:<
    NextAddress= PEEKL(Address)<
    Intuit= PEEKL(Address + 18)<
    CALL FreeMem(Intuit,2Ø)  'Free Intuitext<
    <
    IF PEEKL(Address + 22) > Ø AND PEEKL(Address + 2
2) <> PEEKL(Address + 18) THEN <
      Intuit= PEEKL(Address + 22)<
      CALL FreeMem(Intuit,2Ø)        'Free Select In
tuiText<
    END IF<
<
      CALL FreeMem(Address,34)       'Free Item Stru
cture   <
    <
    Address= NextAddress<
    IF Address > Ø THEN CLearMem<
    <
OutOfSubs:<
RETURN<
        <
ALLDone:<
      <
  LIBRARY CLOSE<
  MENU RESET<
  END SUB<
<
CheckSmenus:<
    <
SUB SUBMENU STATIC<
DEFLNG a-z<
SHARED SubNum%<
intuitmessage= PEEKL(WINDOW(7) + 94)<
menucode= intuitmessage + 24<
SubNum%= (PEEKW(menucode)/ (2^11) AND 31) + 1<
IF SubNum%= 32 THEN SubNum%= Ø  'No Sub if all bits
"on"<
<
END SUB<
<
```

**Program 2. Supermenus Demo**
Filename: SUPERMENUS.DEMO

```
<
'SuperMenu Demo<
'Copyright 1987 COMPUTE! Publications, Inc.<
'All Rights Reserved.<
<
DECLARE FUNCTION ALLocMem&() LIBRARY<
<
ON ERROR GOTO HandLeErrors<
<
INITIALIZE<
PRINT "Copyright 1987":PRINT"COMPUTE! Publications,
Inc."<
PRINT "All Rights Reserved."<
PRINT<
PRINT "Setting up SuperMenus, wait for the BEEP."<
<
MExcLude&=Ø    <
FLags%= text% + enabLed% + comp%<
SMENU 1,Ø,Ø,FLags%,MExcLude&,"","First",""<
SMENU 2,Ø,Ø,FLags%,MExcLude&,"","Second",""<
SMENU 3,Ø,Ø,Ø,MExcLude&,"","",""   'Blank out menus
3 and 4<
SMENU 4,Ø,Ø,Ø,MExcLude&,"","",""<
FLags%= text%  + enabLed% + comp%<
```

```
SMENU 1,1,Ø,FLags%,MExcLude&,"","ItemA",""←
SMENU 1,2,Ø,FLags%,MExcLude&,"","ItemB",""←
FLags%= text% + enabLed% + box% + command%←
SMENU 1,3,Ø,FLags%,MExcLude&,"Q","Quit    ",""←
←
MExcLude&= item2&←
FLags%= text% + enabLed%  + toggLe%  + check% + seLe
ctimage%←
SMENU 1,1,1,FLags%,MExcLude&,"","   SubItem1    ","
  ALternate"←
MExcLude&= item1&←
FLags%= text% + enabLed% + comp%  + toggLe%  + check
%←
SMENU 1,1,2,FLags%,MExcLude&,"","   SubItem2",""←
MExcLude&= Ø←
FLags%= text% + enabLed% + toggLe% + check% + comp%
+ command%←
SMENU 2,1,Ø,FLags%,MExcLude&,"T","   Menu2Item1     "
,""←
MExcLude&= Ø←
FLags%= text% + enabLed% + comp%←
SMENU 2,2,Ø,FLags%,MExcLude&,"","   Menu2Item2    ",
""←
MExcLude&= item2&←
FLags%= text% + enabLed% + toggLe% + check% + comman
d% + comp%←
SMENU 2,2,1,FLags%,MExcLude&,"A","   Item2Sub1    ","
"←
MExcLude&= item1&←
FLags%= text% + enabLed% + toggLe% + check% + box%←
SMENU 2,2,2,FLags%,MExcLude&,"","   Item2Sub2    ",""
←
MExcLude&= item1& + item2&←
FLags%= text% ←
SMENU 2,2,3,FLags%,MExcLude&,"","   Disabled",""←
←
BEEP←
←
PRINT←
PRINT "SuperMenus Ready!"←
←
CheckMenus:←
 ←
 MenuNum%= MENU(Ø)←
←
 IF MenuNum%= Ø THEN CheckMenus←
  ←
   ItemNum%= MENU(1)←
   SUBMENU←
   PRINT "MenuNum= ";MenuNum%,"ItemNum= ";ItemNum%,"
SubNum= ";SubNum%←
```

```
 ←
IF MenuNum%= 1 AND ItemNum% = 3 THEN CALL SMENUOFF:E
ND←
←
 GOTO CheckMenus←
←
HandLeErrors:←
←
     IF ERR= 53 THEN←
          PRINT "You must have the files 'exec.bmap'
 and 'graphics.bmap'"←
          PRINT "in either your current or Libs dire
ctory."←
    END IF←
END←
```

# Ramdisker

Jim Butterfield

*Learn about the powerful technique of using ramdisk files on the Amiga. The use of a ramdisk speeds up disk access tremendously, since ramdisk operations occur in the RAM (Random Access Memory) of the computer. A sample program is included.*

When most people think of data files, they think of collections of information stored on magnetic media—disk or tape. But files may also be stored, at least temporarily, in the computer's RAM (Random Access Memory). Special software can allow the computer to simulate the actions of a physical disk drive using a portion of memory for storage. Because the memory storage behaves just like a physical disk, it's frequently referred to as a *ramdisk*. This article explains simple sequential ramdisk file techniques for the Amiga. The Amiga's operating system has built-in support for ramdisks.

## Temporary Files

Many programs create temporary files. Such a file often contains partially processed data; the program has worked through the data concerned, but cannot finish the job until it has completed another pass through the data.

As an example, consider a program which collects a series of examination scores for a class of students. Among other things, its job is to print each student's performance compared with the class average. The program can't print the comparison at the time it receives each student's score since it can't know the average score until all the grades have been entered. One solution is to use a temporary file to store the scores as they come in.

Another classic data processing task that benefits from temporary files is assembling a machine language program. The assembler program can do part of its work as it reads the source code, but it can't finish the job until all the source code has been read. The assembler can create a partially processed temporary file. Then, on the next pass through the data, it can read back this file and fill in any missing information.

In cases like these, the use of a ramdisk can dramatically improve program performance. Files held in a ramdisk can be retrieved much more quickly than files on a disk in a mechanical drive—even a high-speed hard disk drive. For a ramdisk, the computer need only read data from memory—something it can do with blinding speed. Reading data from disk, on the other hand, involves a number of mechanical tasks in addition to the electronic communications. Before the first byte can be read, the drive must start the disk spinning, determine the position of the file on the disk surface, move the read/write head to the proper track, and wait for the first sector of the file to spin by.

Since the ramdisk is so much faster, you may wonder why it isn't used exclusively for program storage. The answer is that ramdisks have one very significant shortcoming: Since all the information is stored in memory, everything in a ramdisk is lost whenever the computer is turned off, even if the power interruption is accidental or only momentary. For this reason, ramdisks are generally used only for temporary files containing information that needs to be retrieved quickly, but could be reconstructed easily, if lost. Physical drives are still the best choice for permanent file storage.

"Ramdisker," the program at the end of this article, illustrates the use of ramdisk files in a routine to calculate prime numbers. Type in the program and save it to disk.

## Prime Numbers

One definition of a prime number is one which is not evenly divisible by any lower prime number, assuming that the first prime number is 2. This kind of definition is known as *recursive.* The series of prime numbers begins like this: 2, 3, 5, 7, 11, 13, 17, and so on. A more technical definition of a prime number is a number that is divisible only by 1 and itself. The number 1 is included in the prime series when we use this definition.

The example program uses the recursive definition to generate prime numbers. We begin with 2. For all following integers, the program tries dividing into the current value all previous primes it has found. If none of them divide evenly, the new value is added to the list of primes. To speed up the routine, the range of trial divisors is restricted to those less than or equal to the square root of the number being tested.

As each prime number is found, it is placed in a temporary file called PRIMES. For the first prime, 2, the file must be opened for writing. Subsequent numbers are tacked onto the end of the file by opening it for appending.

The performance of the routine is further enhanced by examining only odd numbers. All even numbers other than 2 are divisible by 2 and, hence, cannot be prime.

## Getting Started

The Amiga includes built-in support for ramdisks; however, to conserve memory, the operating system does not normally set up a ramdisk when the system is booted. Instead, it waits until you first request access to the ramdisk and then allocates the memory at that time. In Amiga Basic, you can establish a ramdisk file simply by referencing the device name *ram:*.

## Saving Your Work

If you've loaded the Workbench, you'll find a ramdisk icon on it's screen when you return there after running Ramdisker. Double-click on the ramdisk icon, and you'll see an icon for a file named *primes*. This is the temporary file containing the prime number values. If you wish to keep this file, you must copy the program to a physical disk. Drag the primes icon onto the icon of the disk or drawer where you want it to go.

If you're using the Amiga's CLI (Command Line Interpreter) instead of the Workbench, typing INFO will show you there's a ramdisk mounted. Use DIR RAM: or LIST RAM: to see its contents. To move the primes file to a more permanent place, use the COPY command. If you want the data file's icon to be visible on the Workbench, you must also copy the associated primes.info file.

## Without RAM

*Ramdisker* also works with a regular disk drive. For the sake of comparison, you might want to run the program in this manner. In *Ramdisker,* change the device name in the OPEN statement from ram: to df0:. You'll see the brakes go on as your program slows to the speed of the mechanical disk unit.

That's why using a ramdisk is so easy. No new or special programming techniques are required—and it sure speeds things up.

### Ramdisker

Filename: RAMDISKER

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
' Copyright 1987 COMPUTE! Publications, Inc.←
' All Rights Reserved.←
' PRIMES PROGRAM←
' DEMONSTRATES FILES TO RAM:←
PRINT "Copyright 1987":PRINT "COMPUTE! Publications,
 Inc."←
PRINT "All Rights Reserved.":FOR t=1 TO 10000:NEXT←
CLS←
p=2←
INPUT "Primes to what value";v←
OPEN "ram:primes" FOR OUTPUT AS 1←
GOSUB putprime←
CLOSE #1←
WHILE p=3←
 p=p+2←
 flag=0←
 d=0←
 q=SQR(p)←
 OPEN "ram:primes" FOR INPUT AS 1←
 WHILE NOT EOF(1) AND d<q AND NOT flag←
  INPUT #1,d←
  flag=(p MOD d = 0)←
 WEND←
 CLOSE #1←
 IF NOT flag THEN←
 OPEN "ram:primes" FOR APPEND AS 1←
 GOSUB putprime←
 CLOSE #1←
END IF←
WEND ←
END←
putprime:←
```

```
 PRINT #1,p←
 LIN=LIN+1←
 PRINT USING " #####";p;←
 IF LIN>9 THEN LIN=Ø:PRINT←
RETURN←
←
```

# Masked Input

Steve Michel

*Here's a versatile input routine for use in your Amiga Basic programs. Written as a subprogram, this input routine selectively masks out all unwanted characters. Whether you need numeric input or a simple Y/N response, "Masked Input" can do the job.*

Probably the most vulnerable part of a program is its input routine. If a program is going to crash, it usually does so here. To avoid such occurrences, input routines must carefully screen illegal and unacceptable keypresses. For example, when the program is expecting a numeric response, the input routine should accept only numeric data. Editing keys must be monitored as well. You don't want someone who is using the program to accidentally clear the screen or backspace over your input prompt simply because the wrong key is pressed.

"Masked Input" is an Amiga Basic subprogram that provides a welcome alternative to the INPUT statement. Because it is a subprogram, Masked Input easily can be included in your own programs. Besides being useful, Masked Input offers a good example of the use of subprograms and Amiga library routines.

## Getting Started

Type in the demo program and save a copy before you run it. This program contains both the Masked Input subprogram, named INPUTSTRING, and some preliminary code that demonstrates its use.

Masked Input makes use of the Amiga library file graphics.bmap. This file is included in the BasicDemos drawer of the Amiga Extras version 1.1 (if you use version 1.2 see Appendix B). Before you run the program, make sure that a copy of the graphics.bmap file is on the same disk as the demo program. The location of this file is important. It must be either in the current directory or in the directory named LIBS on the disk used when you booted the system. If you do not have this library file in the correct place, BASIC will stop with a *file not found* error when you run the program.

When run the demo program asks you to enter a string length and edit mask (see below for details). Next, the program calls the INPUTSTRING subprogram using your previous two entries, prompting you for your final input. After you press RETURN, the computer echoes your entry to the screen, waits for the RETURN key to be pressed, and then reruns the program.

## Using the Subprogram

The proper syntax for calling INPUTSTRING is

**CALL INPUTSTRING(*entry$,strlen,emask*)**

Amiga Basic also provides an alternative CALL syntax that allows the subprogram to be used like a new BASIC command:

**INPUTSTRING *entry$,strlen,emask***

INPUTSTRING requires three parameters: *entry$*, *strlen*, and *emask*. The string variable *entry$* returns the text entered by the user. The *strlen* parameter specifies the maximum length of input to be allowed. The *emask* parameter is an edit mask that determines the type of data that can be entered. Valid values for *emask* range from 0 to 127. Different mask values produce the results shown in Table 7-5.

**Table 7-5. Mask Value Results**

| Value | Function |
|---|---|
| 0 | All characters accepted |
| 1 | Numbers 0–9 |
| 2 | Punctuation marks ., +, and − |
| 4 | Upper- and lowercase letters *A–Z* |
| 8 | Blanks (spaces) allowed |
| 16 | Uppercase letters *A–Z* |
| 32 | Characters *Y* and *N* |
| 64 | Null input not allowed |

An important aspect of this method of masking is that the *emask* values may be added together to produce a cumulative effect. A value of 85 (1 + 4 + 16 + 64), for example, allows numbers, upper- and lowercase letters, and spaces, but not punctuation characters or a null input. This method of input masking puts the programmer in complete control.

INPUTSTRING uses less-than and greater-than symbols (< and >, respectively) to frame the area of input. This lets the user see exactly how many characters can be entered. All responses are returned in the variable *entry$*. If a numeric value is required, *entry$* may be converted to a number with the VAL function as illustrated in the demo program.

This subprogram is fully documented with remark statements. All comments that follow the apostrophes found at the end of lines are instructional and may be omitted. The comments following the remarks, however, should be left in place to document the different parameters that are necessary for using the subprogram.

## Editing Keys

In addition to the keys allowed by the edit mask, several other keys are available for editing input. The RETURN key terminates input. The cursor keys allow you to move through entered text. The current position of the cursor is denoted by an underline character. The BACK SPACE key deletes characters at the end of the input string. Pressing the DEL key erases the entire input field.

## Subprograms

As explained above, the entire input routine is contained in the subprogram named INPUTSTRING. A subprogram is a section of code that is called by the main program and interacts with the main program by passing data back and forth through variables called parameters. Parameters are listed in parentheses after the subprogram name. Other variables may also be held in common between the main program and the subprogram through the use of the SHARED statement. Except for passed parameters and shared variables, the subprogram acts as though it were in a world by itself. All other variables used within the subprogram are referred to as *local variables,* which means they are known only to the subprogram. Thus, the variable *LOOP.CNTR* in the main program and the variable *LOOP.CNTR* in a subprogram are treated as two different variables and do not interfere with each other.

Why use a subprogram instead of a subroutine to perform this input function? The main reason is efficiency. Once a subprogram has been written, debugged, and polished up, it can be attached to any program that requires its services. With a variety of prewritten subprograms, you no longer have to rewrite vital routines for each new program. Ideally, one could build and maintain a whole library of subprograms, each one designed for a specific application (inputting, sorting, reading a disk directory, and so on). Writing a program would then simply consist of splicing the appropriate subprogram into the main program. And because each subprogram acts independently, you do not have to worry about conflicting variable names.

It's easy to create a version of the *Masked Input* subprogram that you can add to your own programs. First, load the demonstration program and delete all the lines that come before the SUB INPUTSTRING statement. Next, save the subprogram text to disk as an ASCII file. Use a statement of the form

**SAVE "masked input",A**

When you want to add the subprogram to one of your own programs, load or type in that program and then use a command of the form

**MERGE "masked input"**

to add the subprogram text from disk. Then add the statements for access to the graphics library routines, as explained below. All that's left is to add CALL statements for the INPUTSTRING subprogram and your program is set up for customized input.

## Libraries

When the Amiga is first booted with Kickstart, approximately 192K of operating system is loaded into the upper part of the computer's memory. (Kickstart is found in ROM on the Amiga 500 and Amiga 2000.) This code contains, among many other things, a whole set of instructions that manage the Amiga's graphics. This set of instructions is organized into a neat collection of routines collectively known as the graphics library, which consists of such routines as ClearScreen( ), Draw( ), WritePixel( ), and SetSoftStyle( ).

Before any library routine can be used from BASIC, you must open the library with the LIBRARY command. In the case of our Masked Input routine, the command **LIBRARY "graphics.library"** is used. Executing this command instructs Amiga Basic to load the file graphics.bmap. (See Appendix B for instructions on creating .bmap files.)

To create an underlined cursor, INPUTSTRING uses the graphics library routine named SetSoftStyle( ). This routine allows you to change a font's type style. The syntax for SetSoftStyle is

**CALL SetSoftStyle&(WINDOW(8),*font.style,font.mask*)**

where WINDOW(8) is a pointer to the RastPort for the current window, *font.style* is a value in the range 0–7, and *font.mask* is a value that specifies which type styles are valid for a particular font. Not all fonts have the capability of producing every type style.

To ensure that Amiga Basic interprets this as a function and not as an array reference, a **DECLARE FUNCTION AskSoftStyle& LIBRARY** command is placed near the beginning of the demo program. At this point, we're ready to change a character's font style to produce an underlined cursor. Legal values for the *font.style* parameter are shown in Table 7-6.

**Table 7-6. Legal *font.style* Parameters**

| Value | Font |
|---|---|
| 0 | normal |
| 1 | underlined |
| 2 | boldface |
| 4 | italics |

These values may be added together to achieve multiple font styles. For example, a value of 3 produces underlined boldface type. For our purposes, however, we need only use a 1 for underline.

## Masked Input Demonstration

Filename: MASKED.INPUT

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
' Copyright 1987 COMPUTE! Publications, Inc.←
' All Rights Reserved←
←
demo.driver:←
  ' the following declaration must be made in the ca
lling program←
  DECLARE FUNCTION AskSoftStyle& LIBRARY←
 ←
  LIBRARY "graphics.library"      ' tell AmigaBASIC t
o read it←
←
start: ←
  CLS←
  PRINT"Copyright 1987 COMPUTE! Publications, Inc."←
  PRINT"           All Rights Reserved.":FOR tt=1 TO 3
5ØØ:NEXT tt←
  CLS ←
  strLen = 2: emask = 1                            ' set
 default values←
  LOCATE 2,2: PRINT "Enter string length ";        ' set
 up prompt ←
  CALL INPUTSTRING (entry$,strLen,emask)           ' get
 input←
  size = VAL(entry$): strLen = 3                   ' con
vert to # & reset length←
  LOCATE 4,2: PRINT "Enter edit mask (Ø - 127) ";←
  CALL INPUTSTRING (entry$,strLen,emask)            ←
  mask = VAL(entry$)                               ' con
vert to number←
  CLS: PRINT: PRINT "Enter input here => ";←
  CALL INPUTSTRING (entry$,size,mask)←
  PRINT: PRINT: PRINT "User input was   =>  ";entry$
←
  LOCATE 2Ø,1Ø: PRINT "PRESS ANY KEY"←
get.Loop:←
  g$ = INKEY$: IF g$ = "" THEN get.Loop←
  GOTO start:←
 ←
SUB INPUTSTRING (entry$,strLen,emask) STATIC:←
  REM entry$ = input string returned to calling prog
ram←
```

```
  REM strlen = maximum size of field to be input ←
  REM emask  = number (Ø-127) that determines input
field traits←
  REM emask  = see table at end of subprogram for va
lues & traits←
←
  poss.styLe% =  AskSoftStyle&(WINDOW(8))     ' get p
ossible styles ←
  IF emask < Ø OR emask > 127 THEN emask = Ø←
←
input.string:←
  g$ = INKEY$: IF g$ <> "" THEN input.string   ' cle
ar out keyboard buffer←
  yLine = CSRLIN: xcoL = POS(Ø)                ' get
 screen positions←
  PRINT "<";:LOCATE  yLine, xcoL + strLen + 1: PRINT
 ">";:LOCATE yLine, xcoL←
  pos.cntr = 1: Len.cntr = 1←
  entry$ = "": backspace$ = CHR$(8) ←
next.key: ←
  IF Len.cntr = pos.cntr AND Len.cntr <> strLen + 1
THEN←
     LOCATE yLine,xcoL + pos.cntr: PRINT "_";←
  END IF←
get.key: ←
  g$ = INKEY$: IF g$ = "" THEN get.key←
  ascii = ASC(g$) ←
  IF ascii =  13 THEN quit.sub     ' return ←
  IF ascii =   8 THEN back.up      ' backspace←
  IF ascii =  3Ø THEN move.right   ' cursor right←
  IF ascii =  31 THEN move.left    ' cursor left←
  IF ascii = 127 THEN wipe.out     ' del(ete)  ←
  IF Len.cntr = strLen + 1 AND Len.cntr = pos.cntr T
HEN get.key←
  IF emask = Ø OR emask = 64 THEN print.char←
 ←
' AND each bit of emask to determine edit functions
←
←
check.nums:  ←
  IF (emask AND 1) = Ø THEN check.punct←
  IF ascii >= 48 AND ascii <= 57 THEN print.char←
check.punct:←
  IF (emask AND 2) = Ø THEN check.upLow←
  IF ascii = 46 OR ascii = 43 OR ascii = 45 THEN pri
nt.char←
check.upLow:←
  IF (emask AND 12) = Ø THEN check.spaces←
  IF ascii < 65 OR (ascii > 9Ø AND ascii < 97) OR as
cii > 122 THEN check.spaces←
  IF (emask AND 8) THEN g$ = UCASE$(g$)←
  GOTO print.char←
```

```
check.spaces:←
  IF (emask AND 16) = Ø THEN check.yorn←
  IF g$ = " " THEN print.char←
check.yorn:←
  IF (emask AND 32) = Ø THEN bad.char←
  g$ = UCASE$(g$): IF g$ = "Y" OR g$ = "N" THEN prin
t.char←
bad.char:        ' invalid character based on edit ma
sk←
  GOTO get.key←
print.char:      ' valid character so print it←
  IF Len.cntr = pos.cntr THEN       ' at end of ente
red text ?←
     PRINT backspace$;g$;←
     entry$ = entry$ + g$←
     Len.cntr = Len.cntr + 1←
     pos.cntr = pos.cntr + 1←
  ELSE                              ' no, in middle
of entered text←
     MID$(entry$,pos.cntr,1) = g$←
     GOTO move.right←
  END IF←
  GOTO next.key←
back.up:         ' delete key action←
  IF entry$ = "" THEN get.key←
  IF pos.cntr <> Len.cntr THEN get.key←
  PRINT backspace$;" ";backspace$;←
  Len.cntr = Len.cntr - 1: pos.cntr = pos.cntr - 1←
  IF LEN(entry$) < 2 THEN entry$ = "": GOTO next.key
←
  entry$ = LEFT$(entry$,LEN(entry$)-1): GOTO next.ke
y ←
move.right:      'cursor right action←
  IF pos.cntr = Len.cntr THEN next.key←
  char$ = MID$(entry$,pos.cntr,1)←
  CALL SetSoftStyLe&(WINDOW(8),Ø,poss.styLe%)    ' fo
r underlined characters←
  LOCATE yLine, xcoL + pos.cntr←
  PRINT char$;←
  pos.cntr = pos.cntr + 1←
  char$ = MID$(entry$,pos.cntr,1)←
  CALL SetSoftStyLe&(WINDOW(8),1,poss.styLe%)←
  LOCATE yLine, xcoL + pos.cntr←
  PRINT char$;←
  CALL SetSoftStyLe&(WINDOW(8),Ø,poss.styLe%)←
  GOTO next.key←
move.left:       ' cursor left action←
  IF pos.cntr = 1 THEN get.key←
  IF (pos.cntr = Len.cntr) AND (Len.cntr <> strLen +
 1) THEN←
     LOCATE yLine, xcoL + pos.cntr←
     PRINT " ";←
  END IF<
  IF pos.cntr < Len.cntr THEN<
     char$ = MID$(entry$,pos.cntr,1)<
     CALL SetSoftStyLe&(WINDOW(8),Ø,poss.styLe%)<
     LOCATE yLine, xcoL + pos.cntr<
     PRINT char$;<
  END IF   <
  pos.cntr = pos.cntr - 1<
  char$ = MID$(entry$,pos.cntr,1)<
  CALL SetSoftStyLe&(WINDOW(8),1,poss.styLe%)<
  LOCATE yLine,xcoL + pos.cntr<
  PRINT char$;<
  CALL SetSoftStyLe&(WINDOW(8),Ø,poss.styLe%)<
  GOTO get.key<
wipe.out:         ' erase WHOLE input field & position
 at start of field<
  LOCATE yLine,xcoL+1: FOR wo = 1 TO strLen: PRINT "
 ";: NEXT wo<
  entry$ = "": pos.cntr = 1: Len.cntr = 1: LOCATE yL
ine, xcoL+1<
  GOTO next.key<
quit.sub:         ' return to calling program<
  IF (emask AND 64) AND entry$ = "" THEN next.key <
END SUB<
<
REM ===  EMASK values  ===<
<
REM   Ø = all characters <
REM   1 = numbers only<
REM   2 = .  +   -  punctuation<
REM   4 = A-Z , a-z  upper and lower <
REM   8 = A-Z force uppercase<
REM  16 = blank spaces allowed in input<
REM  32 = Y or N only (forced  uppercase )<
REM  64 = null input not allowed<
<
REM  all mask values may be added together for a cum
ulative effect<
REM  i.e.  an emask of 67 = forced entry of numbers
and punctuation<
<
```

# BASIC BUTTON

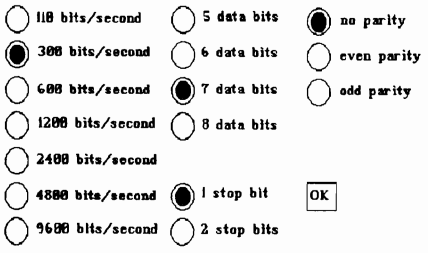
Robert Katz

*Take advantage of the mouse and icon interface with this subprogram that you can use in your own Amiga Basic programs. A demo that lets you change the speed and gender of the Amiga's voice is included.*

Perhaps more than any other BASIC, Amiga Basic is extensible. New commands are easily added to the language. "BUTTON" takes advantage of this flexibility to add a BUTTON command similar to the one in Microsft BASIC for the Macintosh. Buttons allow a menulike choice without forcing the user to hold down the right mouse button and perform a menu action. In general, you use a button when you want the choice to be intrusive; you use menus when you want the choice to be available only when the user looks for it.

Buttons are useful when you want several choices to be displayed and alterable at once. As an example, if you were writing a telecommunications terminal program, you would probably want all the options to be displayed on the screen simultaneously. (See Figure 7-1 for an example.) All at once, the user can see that the speed of communication varies between 110 and 9600; parity may be even, odd, or not selected; and the data and stop bits may be selected in a given number of ways. All of the current settings are clearly shown.

**Figure 7-1. Example of Button Choices**



110 bits/second [✓ 300 bits/second] 600 bits/second 1200 bits/second 2400 bits/second 4800 bits/second 9600 bits/second

5 data bits 6 data bits [✓ 7 data bits] 8 data bits

[✓ no parity] even parity odd parity

[✓ 1 stop bit] 2 stop bits

OK

*A display of telecommunications parameters shows current settings.*

The BUTTON command is implemented as a subprogram. Subprograms differ from subroutines in that subprograms allow local variables that cannot be accessed by the main program or other subprograms. When you use a subprogram, you don't have to worry about using a variable that is already being used by the main program; no side effects are possible (unless you specify that certain variables are to be shared among program and subprograms). Although the variables are independent, it's possible to pass values (called *parameters*) back and forth between the subprogram and its calling program.

A subprogram must begin with a SUB statement containing the name of the subprogram and a list of parameters. The END SUB statement marks the end of a subprogram. All statements between SUB and END SUB are considered part of the subprogram rather than part of the main program.

Program 1 is the subprogram to implement the BUTTON command. Type it in and save a copy to disk, but don't try to run it yet. This isn't a complete program, only a subprogram. Since you'll want to merge this with your own programs, be sure to save it in ASCII format. To do this, you must use a command of the form

**SAVE *"filename"*,A**

## Calling All Subprograms

There are two ways for the main program to access, or *call,* a subprogram. The formal method is with a statement of the form

**CALL *subprogram name(parameter list)***

However, Amiga Basic also permits you to omit the CALL and the parentheses surrounding the parameter list. This format makes the subprogram call look just like any other BASIC statement, so the subprogram essentially becomes a new BASIC command.

Here's the syntax for the BUTTON command:

**BUTTON *ID, state, title, x position, y position, type***

The *ID* and *state* parameters are two-way values. Setting them before you call the subprogram provides instructions to the subprogram, and reading their values after the subprogram is called returns information on the buttons. The other parameters are one-way: They provide information to the subprogram, but nothing is reported back through them. One difference between subprograms and actual commands is that you must supply some value for all the parameters in a subprogram call. None of the parameters are optional, even in cases where the parameter values aren't meaningful.

The *ID* value is a unique number by which you can identify each button. You may have any number of buttons on the screen at once. Allowable button numbers range from 1 up to the maximum number you specify.

An *ID* value of 0 has a special meaning. When the subprogram is called with this value in the first parameter position, it checks whether the mouse is currently positioned over any previously defined button. (In this case, the values of the other parameters are not important.) If the mouse is over a button, then the ID value of that button is returned in the *ID* parameter, the state of the button is inverted (from 1 to 2, or vice versa), and the new state of the button is returned in the *state* parameter. If the mouse isn't over any of the buttons, the *ID* parameter will still be 0 upon return.

Note that this function compares the current mouse pointer position against the coordinates for all defined buttons, without regard for whether the buttons are actually being displayed on the screen at the time of the test. To prevent spurious readings, it's important to clear button definitions when the corresponding buttons are erased from the screen.

The *state* parameter determines the state of the button specified by the *ID* parameter. If this parameter is set to 0 when the subprogram is called, then the current definition for the specified button is erased. This makes the button inactive, but doesn't remove it from the screen. If this parameter is set to 1, then the specified button is deselected (turned off). Setting this parameter to 2 causes the corresponding button to be selected (turned on). The remaining parameters are only meaningful the first time the subprogram is called for a particular *ID* value. *Title* is the name you assign to a button, *x position* and *y position* define the screen coordinates of the button, and *type* identifies the shape of the button as described below. Once a button is defined, these features cannot be changed.

The subprogram provides three different types of buttons. Type 1 is a rectangle with the name of the button inside it. When a type 1 button is selected, the button name appears in reverse video. Type 2 buttons are squares with their names beside them. When a type 2 button is selected, it's marked with an X. Type 3 buttons are circles with their names beside them. When a type 3 button is selected, a smaller filled circle is placed inside the circle.

Two preparatory steps are required when using the BUTTON subprogram in your own programs. First, your main program should begin with the statement DEFINT a–z to force all variables to default to the short integer type. Second, because the subprogram makes calls to operating system graphics routines, you must place a copy of the graphics.bmap file into the disk directory that holds your program. The graphics.bmap file can be found in the BasicDemo folder on the Extras disk that came with your Amiga. (If you're using Workbench version 1.2 see Appendix B.)

## Buttoned Speech

The best way to learn how to use the BUTTON subprogram is by looking at an example. Program 2 illustrates all three types of buttons provided by the BUTTON subprogram. It sets up a control panel for a speech synthesis demonstration. Buttons allow you to change the voice between male and female, and between fast and slow. There are also buttons to start or stop the voice. To try Program 2, type it in as listed and save a copy, but don't try to run it yet. First you must merge in the BUTTON subprogram. Use a command of the form

**MERGE *"filename"***

where *filename* is the name you used for the ASCII file containing the subprogram. After the subprogram is merged in, the combined program is ready to run. If you want, you can save a copy of the completed program to avoid having to merge the subprogram again.

The first BUTTON command in Program 2, in the FOR-NEXT loop, provides the initial definitions of the six buttons. The second BUTTON command, following the WEND, has the *ID* parameter set to zero to check whether the mouse pointer was over a button when the mouse button was clicked. If so, the variable *ID* will have a nonzero value upon return, and array element State(ID) will hold the new state of that variable. (Although Program 2 uses a variable named *ID* for the *ID* parameter, that's not mandatory. Any other variable name would work just as well.) The final BUTTON command is used to set the other button in the panel pair to the complementary state. Note that dummy values are supplied for the last three parameters in the latter two BUTTON commands. These values are necessary to satisfy the syntax of the subprogram call.

Since Program 2 defines only six buttons, the default value for the *maxbut* variable in the subprogram is sufficient. Should your application require more than ten buttons, simply increase *maxbut* to whatever value you need.

After examining Program 2, try using the BUTTON subprogram in your own programs. Remember that you must place a copy of the graphics.bmap file into the disk folder that holds your BASIC programs for the program to function properly.

## Technical Details

Because there is no easy way for Amiga Basic to print text at an arbitrary pixel location, the BUTTON subprogram uses an Amiga operating system graphic routine called Move. Now, characters can be placed outside of normal character boundaries. The Move routine is called like an Amiga Basic subprogram. The routine has three parameters. The first is the address of the current window's RastPort, a structure used by the operating system. We can find this address with the WINDOW(8) function. The other two parameters are the X and Y locations (in pixels) that specify where the text should begin. Here's an example of a call to Move:

```
CALL Move&(WINDOW(8), X%, Y%)
```

Note that X and Y are long integers (as the suffix % shows).

One other system routine is used in the BUTTON subprogram. This routine, SetDrMd, changes the default drawing mode. This is used to invert the colors when a type 1 button is selected. From BASIC, it's called like this:

**CALL SetDrMd&(WINDOW(8),*value*)**

*Value* represents one of the four available drawing modes. Table 7-7 shows the modes and their corresponding values.

**Table 7-7. Mode Values**

| **Mode** | **Value** |
|---|---|
| Jam1 | 0 |
| Jam2 | 1 |
| Complement | 2 |
| Inversvid | 3 |

The default mode is Jam2. It "jams" the foreground and background colors (as an 8 x 8 rectangle) onto the screen memory. Jam1 jams only the foreground color onto the screen, allowing you to overlay characters. Complement reverses the colors, changing all ones to zeros and all zeros to ones. For instance, if you are using a four-color screen, color 0 would switch places with color 3, and color 1 would switch places with color 2 (it makes more sense in binary). Inversvid exchanges the foreground color with the background color to make inverse characters.

**Program 1. BUTTON**
Filename: BUTTON

*For instructions on entering these programs, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
<
 <
  <
'BUTTON  subprogram<
' Copyright 1987, COMPUTE! Publications, Inc.<
' ALL rights reserved.<
<
SUB BUTTON(ID,State,TitLe$,X,Y,Type) STATIC<
  IF ID=Ø THEN GOTO CheckButton<
  IF numbut>Ø THEN GOTO Skipped<
  LIBRARY "graphics.library"<
  rastport&=WINDOW(8)<
  CALL SetDrMd&(rastport&,Ø)<
  maxbut=1Ø<
  DIM ID(maxbut),State(maxbut),TitLe$(maxbut),Type(m
axbut)<
  DIM X1(maxbut),Y1(maxbut),X2(maxbut),Y2(maxbut)<
  AREA(1Ø,1Ø):AREAFILL<
<
Skipped:<
  IF (ID<Ø OR ID>maxbut OR State<Ø OR State>2 OR Typ
e>3) THEN EXIT SUB <
  IF State=Ø THEN GOTO EraseButton<
  IF ID(ID)<>ID THEN  <
    ID(ID)=ID<
    IF ID>numbut THEN numbut=ID<
    TitLe$(ID)=" "+TitLe$+" "<
    X1(ID)=X:Y1(ID)=Y  <
    Type(ID)=Type<
  END IF<
  IF Type(ID)<1 THEN GOTO EraseButton<
  State(ID)=State<
  ON Type(ID) GOTO One, Two, Three<
 <
One:<
  X2(ID)=X1(ID)+LEN(TitLe$(ID))*1Ø+1:Y2(ID)=Y1(ID)+1
1<
  LINE(X1(ID),Y1(ID))-(X2(ID),Y2(ID)),Ø,bf<
  LINE(X1(ID),Y1(ID))-(X2(ID),Y2(ID)),1,b<
  IF State=2 THEN<
    LINE(X1(ID),Y1(ID))-(X2(ID),Y2(ID)),1,bf    <
    CALL SetDrMd&(rastport&,5)    <
  END IF  <
  CALL Move&(rastport&,X1(ID)+1,Y1(ID)+8)<
  PRINT TitLe$(ID):CALL SetDrMd&(rastport&,1)<
EXIT SUB<
<
Two:<
  X2(ID)=X1(ID)+2Ø:Y2(ID)=Y1(ID)+1Ø<
  LINE (X1(ID),Y1(ID))-(X2(ID),Y2(ID)),1+(State=1)<
  LINE (X1(ID),Y2(ID))-(X2(ID),Y1(ID)),1+(State=1)<
  LINE (X1(ID),Y1(ID))-(X2(ID),Y2(ID)),1,b    <
  CALL Move&(rastport&,X1(ID)+23,Y1(ID)+8)<
  PRINT TitLe$(ID)<
EXIT SUB<
<
Three:<
  X2(ID)=X1(ID)+12:Y2(ID)=Y1(ID)+12<
  Xc=X1(ID)+5:Yc=Y1(ID)+5<
  CIRCLE (Xc,Yc),12<
  IF State=2 THEN CIRCLE (Xc,Yc),6<
  PAINT (Xc,Yc),1+(State=1)<
  CALL Move&(rastport&,X1(ID)+19,Y1(ID)+8)<
  PRINT TitLe$(ID)←
EXIT SUB←
←
CheckButton:←
  Xm=MOUSE(1):Ym=MOUSE(2)←
  Checkloop:←
    ID=ID+1←
    IF ID>numbut THEN←
      ID=Ø←
      EXIT SUB←
    END IF←
    IF ID(ID)=Ø THEN GOTO Checkloop←
    IF (Xm>X1(ID) AND Xm<X2(ID) AND Ym>Y1(ID) AND Ym
<Y2(ID)) THEN ←
      State=1-(State(ID)=1)←
      GOTO Skipped←
    END IF   ←
    GOTO Checkloop←
←
EraseButton:←
  ID(ID)=Ø:State(ID)=Ø←
  TitLe$(ID)=""←
  X1(ID)=Ø:Y1(ID)=Ø  ←
  Type(ID)=Ø←
  EXIT SUB←
    ←
END SUB←
  ←
  ←
  ←
```

**Program 2. BUTTON Demonstration**
Filename: BUTTON.DEMO

```
'BUTTON Demo←
' Copyright 1987, COMPUTE! Publications, Inc.←
' ALL rights reserved.←
←
DEFINT a-z ←
DIM voice(8),word$(9),State(6),TitLe$(6),X(6),Y(6),T
ype(6)←
FOR i=Ø TO 8:READ voice(i):NEXT←
DATA 11Ø,Ø,15Ø,1,222ØØ,64,1Ø,Ø,Ø←
FOR i=1 TO 9:READ word$(i):NEXT←
DATA one,two,three,four,five,six,seven,eight,nine←
LOCATE 1,1Ø:PRINT "Copyright 1987, COMPUTE! Publicat
ions, Inc."←
LOCATE 2,21:PRINT "All rights reserved."←
'Initialize buttons←
FOR ID=1 TO 6←
```

```
  READ State(ID),TitLe$(ID),X(ID),Y(ID),Type(ID)<
  BUTTON ID,State(ID),TitLe$(ID),X(ID),Y(ID),Type(ID
)<
NEXT <
DATA 1,Male,150,25,1<
DATA 2,Female,300,25,1<
DATA 1,Normal,150,50,2<
DATA 2,Fast,300,50,2<
DATA 1,Start,155,75,3<
DATA 2,Stop,305,75,3<
'Initialize voice parameters<
GOSUB Gender<
GOSUB Speed<
GOSUB Status<
count=1<
<
here:<
  WHILE MOUSE(0)<1<
    IF speak THEN<
      SAY TRANSLATE$(word$(count)),voice<
      count=count+1<
      IF count>9 THEN count=1<
    END IF<
  WEND:ID=0<
  BUTTON ID,State,"",0,0,0<
  IF ID=0 THEN GOTO here 'mouse pressed when not on
a button<
  State(ID)=State<
  'Reverse state of complementary button<
  IF (ID MOD 2) THEN ID2=ID+1 ELSE ID2=ID-1<
  State(ID2)=1-(State=1)<
  BUTTON ID2,State(ID2),"",0,0,0<
  'Change operating conditions according to button s
election<
  ON INT((ID/2)+.5) GOSUB Gender,Speed,Status<
  GOTO here <
END<
<
Gender:<
  IF State(1)=2 THEN voice(0)=110:voice(3)=0 ELSE vo
ice(0)=200:voice(3)=1<
  RETURN<
Speed:<
  IF State(3)=2 THEN voice(2)=150 ELSE voice(2)=250<
  RETURN<
Status:<
  speak=(State(5)=2)<
  RETURN<
  <
```

```
'Merge BUTTON subprogram here...←
←
 ←
  ←
  ←
```

# APPENDICES

# Appendix A: Menu Driver

*Brian Flynn*

*Here's a simple, straightforward file menu program for the Amiga. It's not only good for the programs in this book, but can be adapted to all your BASIC program disks. Works on any Amiga with Amiga Basic.*

The Amiga is without a doubt one of the most impressive personal computers on the market today. With a multitude of complex operations, it does almost anything needed. "Menu Driver" takes advantage of the speed, color, and mouse-selection features in allowing you to choose a BASIC application to run.

## How to Type In Menu Driver

Although you may use Menu Driver on as many disks as you want, you'll only have to type it in once. Type it in exactly as listed and save a copy to disk before you do anything else. After you have this generic copy of Menu Driver safely tucked away, you can begin to customize it for your own needs. When you're ready, simply load the program and make the changes needed.

## Make Your Own Modifications

There are four changes that need to be made to Menu Driver to completely customize it. The first is in the initial line of the program. Change this remark to reflect the name of the book, disk, chapter, and so forth, that you are handling with Menu Driver. This may seem trivial, but as your use of the program grows, it enables you to see at a glance what version you're looking at. Next, change the WINDOW statement in the SETSCREEN subroutine to reflect the title you want to appear in the window menu bar. For example, the line could be

```
WINDOW 2,"Card Games",,0,1
```

for this book's chapter on card games.

Now change the value of the variable N in the second line of the KEYVALUES subroutine to reflect the number of programs you want to handle. Typically, if you had ten programs you wanted to access, you would type N=10. Finally, at the end of the program, put a list of your programs into the DATA statements. The first should be the name that you want to appear onscreen as the title of the chapter. The name used in the WINDOW statement earlier would be fine. Each of the other DATA statements have two parts: the name of the program as you want it to appear on the menu and the actual filename on disk. For instance, if you had a program that drew a clock and gave the time, called Amiga Onscreen Clock, with a filename of AMCLOCK on the disk, your DATA would look like this:

```
DATA Amiga Onscreen Clock, AMCLOCK
```

Be sure that the actual filename is the second item in the DATA statement.

## Using Menu Driver with This Book

You should make a menu driver for each chapter in this book. Customize each version of Menu Driver using the instructions above. To load and run a program, load Amiga Basic and then load the menu driver for that chapter. Each chapter's menu driver will have an icon, so alternately you could just click on that. Using this method, you must make sure you always keep a copy of Amiga Basic in the directory that your programs are in. Once the menu driver is activated, simply click on the program you wish to run.

## Menu Driver

Filename: MENU DRIVER

*For instructions on entering this program, please refer to Appendix B, "COMPUTE!'s Guide to Typing In Amiga Programs."*

```
REM Put Chapter Title Here←
 GOSUB INITIALIZE←
 GOSUB MAIN.MENU←
 RUN TITLE.SHORT$(PICK)←
END←
←
INITIALIZE:←
 GOSUB SETSCREEN←
 GOSUB KEYVALUES←
 GOSUB SETMENUS←
 GOSUB SETCOLORS←
 GOSUB SHAPES←
RETURN←
 ←
SETSCREEN:←
 SCREEN 1,640,200,3,2←
 WINDOW 2,"Title",,0,1:REM Chapter Title←
 WIDTH 80←
 CLS←
RETURN←
←
KEYVALUES:←
 DEFINT A-Z←
 N = 1 : REM N is the number of programs to handle←
 DIM TITLE.LONG$(N),TITLE.SHORT$(N)←
 DIM CIRCLES(150)←
 CIRCLE.I(1) = 1: CIRCLE.I(2) = 75←
 READ CHAPTER$←
 FOR I=1 TO N←
  READ TITLE.LONG$(I),TITLE.SHORT$(I)←
 NEXT←
RETURN←
←
SETMENUS:←
 FOR I=2 TO 4←
  MENU I,0,0,""←
 NEXT←
 MENU 1,0,1,"STOP"←
 MENU 1,1,1," Go to BASIC"←
 MENU 1,2,1," Go to System"←
 MENU ON←
 ON MENU GOSUB GOODBYE←
RETURN←
←
GOODBYE:←
 WINDOW CLOSE 2: WINDOW 1: MENU RESET←
 SCREEN CLOSE 1←
 ITEM = MENU(1)←
 IF ITEM = 2 THEN SYSTEM←
 CLS←
 PRINT "Bye-Bye"←
 STOP←
RETURN←
←
SETCOLORS:←
 REM TAN, GREEN, & RED←
  PALETTE 4,.95,.7,.53←
  PALETTE 5,.14,.43,Ø←
  PALETTE 6,.93,.2,Ø←
RETURN←
←
SHAPES:←
 X=313: Y=8Ø: X1=X-7: X2=X+7: Y1=Y-3: Y2=Y+3←
 LINE(X1,Y1)-(X2,Y2),4,BF←
 FOR I=1 TO 2←
  K = 7-I←
  CIRCLE(X,Y),7,K: PAINT(X,Y),K←
  GET(X1,Y1)-(X2,Y2),CIRCLES(CIRCLE.I(I))←
 NEXT←
RETURN←
←
MAIN.MENU:←
 CLS←
 RTN$ = "OFF": PICK = 1←
 S$ = CHAPTER$: L = LEN(S$)←
 LINE(313-1Ø*L/2-15,15)-(313+1Ø*L/2+15,27),1,B←
 PAINT(313,2Ø),6,1←
 COLOR 1,6: LOCATE 3: PRINT PTAB(313-1Ø*L/2)S$←
 LINE(135,32)-(495,13Ø),2,B: PAINT(313,8Ø),4,2←
 COLOR 2,4←
 FOR I=1 TO N←
  IF I = PICK THEN INX = 2 ELSE INX = 1←
  CALL DRAW.CIRCLE(I,INX)←
  LOCATE I+5,3Ø: PRINT TITLE.LONG$(I)←
 NEXT←
 LINE(263,141)-(36Ø,153),2,B: PAINT(313,145),3,2←
 COLOR 2,3←
 LOCATE 19: PRINT PTAB(282)"Return"←
 COLOR 1,Ø←
 LOCATE 22,19: PRINT "Click Mouse on Choice,";←
 PRINT " then Click on Return"←
 GOSUB CHOOSE←
RETURN←
←
SUB DRAW.CIRCLE(R,INX) STATIC←
 SHARED CIRCLES(),CIRCLE.I()←
 Y = 8*R+32←
 PUT(182,Y),CIRCLES(CIRCLE.I(INX)),PSET←
END SUB←
←
CHOOSE:←
 GOSUB GURGLE←
 GOSUB CLICKIT←
 IF S$ = "" THEN GOSUB LOCATION←
 IF ASC(S$+" ") <> 13 AND RTN$ = "OFF" THEN←
  GOTO CHOOSE←
 END IF←
RETURN←
←
GURGLE:←
 FREQ = 300←
 FOR G=1 TO 5←
  FREQ = 500 - FREQ←
  SOUND FREQ,1,50←
 NEXT←
RETURN←
←
CLICKIT:←
 S$ = ""←
 WHILE MOUSE(0) = 0 AND S$ = ""←
  S$ = INKEY$←
 WEND←
  X = MOUSE(1)←
  Y = MOUSE(2)←
 WHILE MOUSE(0)<> 0: WEND: REM RESET←
RETURN←
←
LOCATION:←
 IF X>263 AND X<360 AND Y>141 AND Y<153 THEN←
  RTN$ = "ON"←
 ELSE←
  P = INT((Y-35)/9) + 1←
  IF X>170 AND X<210 AND P>0 AND P<= N THEN←
   CALL DRAW.CIRCLE(PICK,1)←
   CALL DRAW.CIRCLE(P,2)←
   PICK = P←
  END IF←
 END IF←
RETURN←
←
REM PROGRAMS←
 DATA Chapter Title Goes Here←
 DATA Program Title, FILENAME←
```

# Appendix B: COMPUTE!'s Guide to Typing In Amiga Programs

The precision of the Amiga is very evident once you start to type in a program. In order to get COMPUTE!'s Amiga listings to operate properly, you must type them in exactly as they appear. There is one exception to this rule.

## Entering the Listings

Examine a listing. At the end of each line, there is a left-arrow character. ***This delimeter should not be typed in as a part of the line.*** It's there just to let you know where the end of the line is. *When you are typing and you encounter the arrow, you should press the carriage return.*

This can be confusing, because the arrows don't always look as though they're at the end of a line. You may see something that looks like this:

```
MinuteDigital=CINT(RND)*72:HourDigital=INT(RND*12)*1
2←
```

Even though this appears to be two physical lines, it is actually only one logical line. You should keep typing until you get to the left-arrow. When you do, press the RETURN key. Your Amiga will display the line this way:

**MinuteDigital=CINT(RND)*72:HourDigital=INT(RND*12)*12**

The Amiga will not always display your lines the way they look in the book. This doesn't mean you've made an error. As long as you remember to place the carriage returns as indicated by the left-arrows, the line will function as it should if you haven't made any typing errors.

There's another thing to remember. Sometimes words and numbers are broken in unlikely places. You might see

```
IF MENU(Ø)=1 THEN ON MENU(1) GOSUB practice,hours,ha
lf,five,quit←
```

Again, this is one line that looks like two. Type it in, and the Amiga displays it this way:

```
If MENU(0)=1 THEN ON MENU(1) GOSUB practice,hours,half,five,quit
```

Keep in mind also that the Amiga handles line numbers in its own way. Typically, microcomputer BASIC line editors will insert lines in their proper numerical order automatically. Since the Amiga has an advanced editor that needs no line numbers, it will not. The Amiga handles line numbers as *labels*. For instance, if you were to type in lines 10, 20, and 30, and then go back and type in line 25, the Amiga would execute the lines in just that order, 10, 20, 30, and 25. Simply make sure you type the statements in the programs with line numbers in the exact order that they appear.

One more note: The Amiga is quite responsive to your keystrokes. A line accepted into the computer's memory can affect all the other lines in the listing. If you type in a variable in caps the first time and then mistakenly in lowercase later on, the Amiga line editor will change all the occurrences of that variable to lowercase. This makes it easy for you to get listings with all kinds of lower- and uppercase letters that don't match the book listing. In general, you'll get the best results if you type everything just the way you see it, except as noted above.

## Saving Your Files

Each listing has a suggested filename. Use this format to save your files:

**SAVE "FILENAME",A**

where FILENAME is the name of the file you are saving. The ,A causes your file to be saved in ASCII format. This is very important because it enables a file to be MERGED with another file.

## Creating .bmap Files

There are a number of .bmap files used in the programs in this book: graphics.bmap, diskfont.bmap, exec.bmap, and icon.bmap. If you use the 1.2 version of the Extras disk, you should use the following procedure to create these .bmap files.

1. From Amiga Basic, type

   **LOAD "Extras:BasicDemos/ConvertFD"**

   Run the program after it loads.
2. When the prompt *Name of .fd file to read>* comes up, type

   **Extras:FD1.2/graphics_lib.fd**
3. When the prompt *Name of .bmap file to produce>* appears, type

   **Libs:graphics.bmap**

   You'll get several warning messages, but these can be safely ignored.
4. When the program finishes, run it again using

   **Extras:FD1.2/diskfont_lib.fd**

   and

   **Libs:diskfont.bmap**

   respectively, to create the diskfont.bmap file. Run the program again, this time using

   **Extras:FD1.2/icon_lib.fd**

   and

   **Libs:icon.bmap**

   to create the icon.bmap file.

   To create the exec.bmap file run ConvertFD one more time, this time answering the prompts with

   **Extras:FD1.2/exec_lib.fd**

   and

   **Libs:exec.bmap**

# Appendix C: How to Use *COMPUTE!'s Second Book of Amiga Disk*

*COMPUTE!'s Second Book of Amiga Disk* is simple to use. There are a couple of things to keep in mind when using it.

First of all, *COMPUTE!'s Second Book of Amiga Disk* is not a startup disk. It does not contain any of the Amiga system programs. So in order to use it, boot your computer normally—either with Kickstart and then the Workbench, or with Workbench if you have an Amiga with Kickstart already in ROM.

Once the computer is booted, take a copy of Amiga Basic and place it onto *COMPUTE!'s Second Book of Amiga Disk.* There is room on the disk for one copy of Amiga Basic. The reason for this is simple. Whenever you try to run the COMPUTE! programs, the Amiga will try to find Amiga Basic in the same directory.

If you don't know how to copy programs from one disk to another, here's an easy way:

1. After the Amiga is up and running, put your disk with a copy of Amiga Basic on it in one drive, *COMPUTE!'s Second Book of Amiga Disk* in the other.
2. Click on the icon for your disk that has Amiga Basic on it.
3. Drag the Amiga Basic icon on top of the icon for *COMPUTE!'s Second Book of Amiga Disk.*

The Amiga will copy Amiga Basic from your original disk to *COMPUTE!'s Second Book of Amiga Disk.*

If you have one drive only, use this method:

1. After the Amiga is up and running, open up a CLI window by first clicking on the icon for your Workbench disk, and then clicking on the CLI icon.
2. At the prompt, type

   **COPY C/COPY to RAM:**

   and press RETURN, and the Amiga will copy the COPY program to RAM:.
3. Next, remove the Workbench disk and place the disk with your copy of Amiga Basic on it into the drive.
4. At the CLI prompt type

   **RAM:COPY DF0:AMIGABASIC#? to RAM:**

   and press RETURN, and the Amiga will copy Amiga Basic to RAM:.
5. Now place the *COMPUTE!'s Second Book of Amiga Disk* into the drive and type

   **RAM:COPY RAM:AMIGABASIC#? to DF0:**

   and press RETURN, and the Amiga will copy Amiga Basic to *COMPUTE!'s Second Book of Amiga Disk.*

Once you have a copy of Amiga Basic on *COMPUTE!'s Second Book of Amiga Disk,* you are ready to use it. The disk contains menu programs for each chapter. The filenames are these:

STRATEGYGAMES
CARDGAMES
ARCADEGAMES
FUNGAMES
APPLICATIONS
GRAPHICS
PROGRAMMING
INSIDEAMIGA

Open up *COMPUTE!'s Second Book of Amiga Disk* by clicking on its icon and then click on one of the filenames shown above. The menu programs are screen prompted. Just click the button for the program you wish to run.

# Index

From zapping your opponent's chess pieces with a laser to creating a ramdisk, the programs in *COMPUTE!'s Second Book of Amiga* offer a gateway to the inside of your Amiga that you've never walked through before.

Here are some of the best games, applications, programming tips, and graphics aids that have ever appeared in *COMPUTE!* magazine. You'll find the programs and articles are of the same high quality that you've always expected in COMPUTE! Books. And, best of all, they're ready to type in and use.

Here's a sample of what's inside:

- *Laser Chess*™, winner of COMPUTE!'s $10,000 programming contest, in which strategy and skill are used to defeat your opponent.
- "Rememory," where your powers of recall are stretched to the limit while you match a myriad of colorful patterns.
- "Climber 5," a sparkling arcade game that puts you on a skyscraper to retrieve wayward homeruns.
- "Canfield," with its turn-of-the-century atmosphere of honkytonks and bankrolls, where you fight the odds at solitaire.
- "Ramdisker," a program that demonstrates writing to RAM-based sequential files.
- And dozens of other programs that will instantly become a part of your Amiga library—ones you'll use over and over again.

Also included as a bonus is an all-purpose "Menu Driver" program that can be used on any of your Amiga disks for BASIC programs. With full instructions included, the "Menu Driver" allows easy access to BASIC programs, just by clicking on a button.

COMPUTE! Books remains the leader by providing readers with more in-depth and complete coverage of the Amiga than can be found anywhere else. From top to bottom, this book is no exeception. *COMPUTE!'s Second Book of Amiga* continues in the tradition of excellence you have come to expect from COMPUTE! Books.

$16.95

ISBN 0-87455-122-6

90000

9 780874 551228