Color the Shapes

by Sol Guber

Structured programming is a way of thinking. It divides the parts of a program into smaller and smaller parts, and then, when the parts are very tiny (and obvious), starts to write the program. This kind of thinking is the basis of any FORTH program, first the top-down, then the bottom-up. First, look at the big picture, then keep looking at the littler and littler picture. Finally, from the details, build up the big picture again. Action! uses the same thinking to make up programs.

The game.

Color the Shapes is a game which was written using a top-down, then bottom-up type of programming.

First, let's go over the game briefly. It's a competitive coloring game for either one or two persons. The object in the one-player game is to color in all the shapes on the board with any of the four colors that are shown on the bottom of the screen.

The only rule is that shapes with a side in common cannot have the same color. If all they have is a corner in common, then they may share a color. If you try to fill in a spot with a color that cannot be used there, you'll hear a double beep, and a message will be shown on the screen.

To make the game more fun, there's an option for the computer to fill up to five shapes at random with a random color. The object of the two-person game is to be the last person to color a shape. That person is the winner.

Since there's no way for the computer to determine if there are any more legal moves, I've included an option to quit. This is done by moving the cursor to the Q on the bottom of the screen.

Figure 1 shows a sample board that will need to be colored in. Each time the game is played, a new board will be generated. The letters in the various shapes are used later in the description of the data structures.

Figure 2 shows a game in progress. The bottom

Color the Shapes *continued*

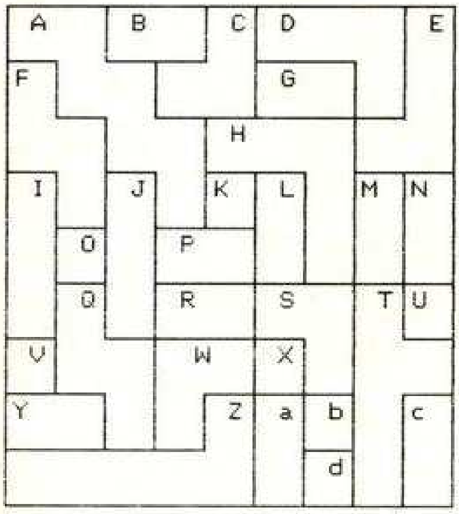


Figure 1.

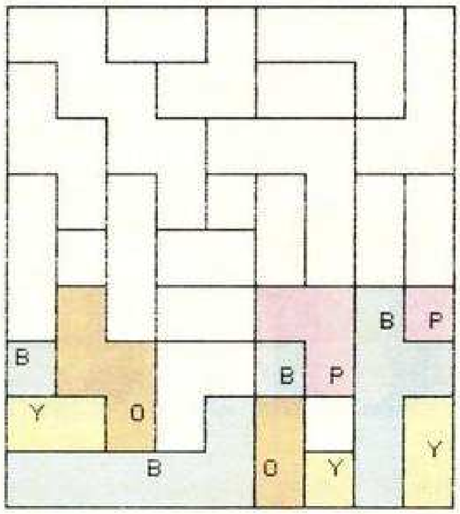


Figure 2.

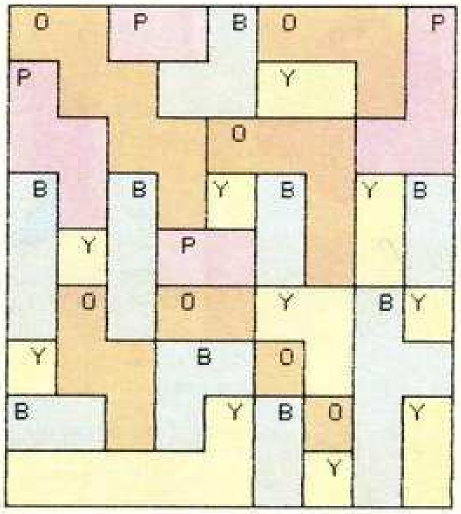


Figure 3.

of the board has been filled in with various colors. Figure 3 shows a completed game.

The cursor is a star that is shown on the screen. **Color the Shapes** can be played with either a **KoalaPad** or a joystick. A question will be asked after the entering of the players' names, to determine if this will be the joystick version.

In the **KoalaPad** version, the cursor is moved by pointing to the spot where you wish to move. The cursor will go there. In the joystick version, move the joystick in the direction that you wish to go, and the cursor will head that way.

The cursor's color is the same as the color that will be used to fill the shape. When the cursor is anywhere in the shape that you wish to fill, just press the trigger.

To change colors for the fill, move the cursor to any colored shape and press the trigger. This will give a beep, and the cursor will move to the bottom of the screen. By moving the cursor left or right, you move to the position of one of the other colors, or the Q. Press the trigger when the cursor is by the color you want. You can change colors as many times as you want.

Your turn will be over when you successfully fill in a shape. There's no way to lose a turn.

The structure.

Now that we know the basic outline of the game, what does this have to do with structured programming? That's easy to see by looking at the last PROC that was written.

It's just a long loop that does very logical things. It's made up of TITLE, PMGRAPHICS, SETUP, PMCLEAR, MAKEPM, GRID, SEARCH, CHECK_BOARD, INIT, NAME and, finally, the major DO OD loop. This loop just consists of two lines and a limit.

The first part of the PROC sets everything up and checks to see what's been done. The heart of the program can be explained very simply by the two functions TRIGGER and JOYSTICK.

TRIGGER checks to see if the trigger has been pressed. JOYSTICK checks to see if a move's been made. IF TRIGGER = 0 then COLOR_IN(SPOT). IF JOYSTICK = 1 then MOVE(). Do this until either the board is completed or QUIT = 1. How could any program be simpler?

This is the whole point of structuring programs: break everything down into easy-to-digest units that are logically simple. While the game's going on, the only two things to look at are the triggers and either the joystick or the **KoalaPad**.

How long should the program monitor these two things? Until the game is complete, or someone quits. Then what? Ask if another game is desired. If it is, play again; otherwise, finish. There's no need to monitor the keyboard, get data from the disk, or do anything else.

Structured programming uses the concepts of positive actions. Do an action until something happens or a flag is set or while a condition still occurs. It can be used in all parts of the program to make the programming easier and very logical. Let me go into some more details on how this type of thinking; the idea to *do* while or *until* is a very nice concept.

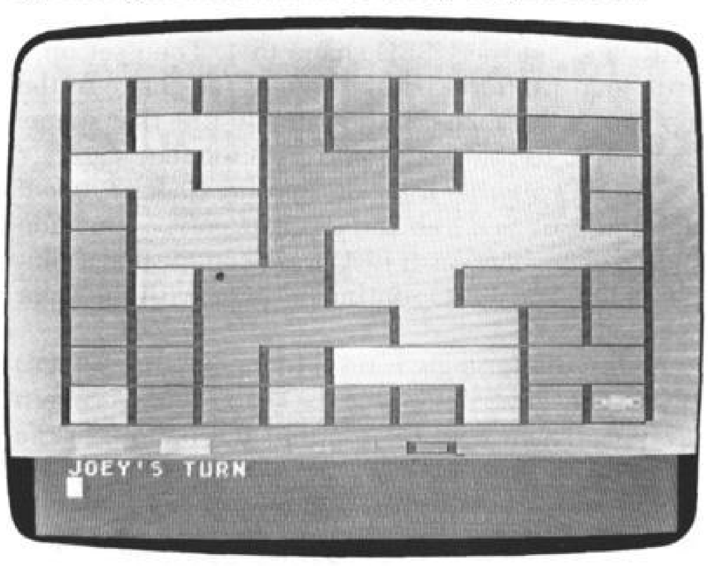
This simplicity is used in other parts of the program. Let us go through several of the other procedures and functions. If the trigger has been pressed, TRIGGER()=0 and we will C=COLOR_IN(SPOT). There are several options in that procedure. If the spot has a color there, B(SPOT)< >0, then what we want to do is change colors or quit. A loop is set up so that we continue to PICK_COLOR until a flag is returned to say that it is non-zero. A good pick has been done. If quit is one, then return. Otherwise move the cursor back to where it was and continue the turn. If the spot had not been colored in already, then we must check to see if it is a GOOD_COLOR. If the flag is returned as 0 then BEEP, print a message on the screen, BEEP again, and then RETURN. Finally, if it is a good move, then FILL_IN(SPOT), check to see if there are two players, and write the new name on the screen.

JOYSTICK is another example of a simple procedure that does only one thing; it checks to see if there has been any movement in the joystick or the **KoalaPad**. CFLAG is used to signal that the **KoalaPad** is to be used. If it's on, check the two locations in memory that store the value of the point on the pad that's being touched. If either point is less than five, then the pad is not being touched. RETURN a zero to show no movement. Next, calculate the X and Y position of the point that's being touched. If the movement is only slight, then RETURN a zero to again show no good movement. Otherwise, set the new X and Y positions to this point, and return a 1 to show success.

The other part of JOYSTICK() is used if play is with a joystick. First determine the value of the joystick. If it is 15, RETURN a zero to show no move. If the value is 11 and you can move left, then move left and return a 1 for success. If it is 7 and you can move right, then make the new position and return a success. Do this for up and down. If no move was possible, return a zero for unsuccessful move.

Both of these two procedures show how the logic was broken up into simple steps, each one of which was very obvious. There were other parts of the program that took judgement and thinking. They're not really a part of structured programming, but are necessary, anyway.

There's a lot of data stored about the screen. See Figure 1 for an example of an initial board. It's a nine-by-nine grid and can have many shapes in it.



Color the Shapes.

There are four data structures that were used to store information about the shapes. The first was an array called R. It is a simple one-to-one correspondence to the grid on the screen. The first value corresponds to the top square; the one below is R(11); and so forth.

To make some of the calculating easier, the array for R was made up to be ten squares by nine rows. R is filled with numbers corresponding to the shapes that are seen. Thus, the first shape (Figure 1) will put R(1)=1, R(2)=1, R(12)=1, R(13)=1, R(23)=1, etc.

The array B is a simple correspondence to array A. It just contains the color values of each square in the grid. The next array is GAR. Shape A corresponds to GAR(1), shape B corresponds to GAR(2), etc. The values in GAR tell how big the shapes are. The value is a two-digit number. The units digit is the row for the top of the shape, and the tens digit is the row for the bottom of the shape.

Thus, shape G is GAR(7) and has a value of 11. Shape M is GAR(13) and has a value of 43, and shape A gives GAR(1)=30. The final array is called USED. It corresponds to GAR and tells if each shape has been colored in. Every time a shape is filled with color, the corresponding shape in USED is given a

1. Thus the function COMPLETE, to determine if the game is over, just looks at each value in USED, and if there's a 1 in each spot, then all the shapes have been colored in.

Now that we have some information on how the data is stored, we can look at some of the other functions and see how simple they are to program.

Let's look at FILL_IN. First, we determine the number of the shape where we are from array R. Find the top and bottom rows of that shape from array GAR, and set the USED shape to 1. Then set up a little loop from the bottom row of the shape to the top row of the shape. If the value in R is that shape, then set B to that color, and FILLER that square.

FILLER's another little subroutine. Check to see if the right side is a line and the bottom is a line. You should change values if they are. Then, just do a simple PLOT, DRAWTO routine to fill in with the color selected.

A very similar logic is used in the function GOOD_COLOR. First, determine the shape you're on from array R. Then, find the top and the bottom of the shape from array GAR.

Start at the bottom row and check each square. If it's part of the same shape as the one that we're looking at, check all four squares around it to see if the color is present there. If it is, return a 0 to show failure. If everything's been checked, and no two colors will be touching, report a success (RETURN(1)).

Among the things that I haven't done is explain how some of the data is generated, or how the random shapes are made, but the logic in this part is also very straightforward and can be explored, if needed.

This game is a good example of two things. The first is that Action! makes structured programming very easy. The second is that, with good simple logic on the overall design of a program, it can be split into smaller and smaller parts. Each part can be further divided into parts that are easily programmed.

I hope you enjoy **Color the Shapes**. My daughter and I had fun inventing it. It's a good game of logic from which you can learn about programming. □

Sol Guber has been programming for his Atari 800 for five years now. The idea for this game came from his seven-year-old daughter Rebecca, to whom computers are a natural part of life.

Listing 1.
Action! listing.

```
; COLOR THE SHAPES
;
; by Rebecca Guber and Sol Guber

MODULE

BYTE ARRAY
  R(100),USED(60),PLAYER(20),B(90),
  CLS=704,A(10),GAR(60),
  INTER=[72 169 0 141 10 212 141 27
        208 104 64],
  TX=[0 252 0 4],TY=[248 0 8 0],
  TEST=[246 255 10 1],
  COLORS=[8 122 88 28 132 248 190
          14 190],
  STAR=[0 0 0 0 24 126 60 60 126 24
        0 0 0 0]

CARD SC1,YP1,YP,Y1
BYTE CFLAG,COL,PLAYNUM,COUNT,DX,DY,
     OLDX,OLDY,X,Y,TURN,QUIT

PROC SETUP()
CARD Z
Z=PEEKC(560)
POKE(Z+166,143)
POKEC(512,INTER)
POKE(54286,192)
POKE(87,10)
POKE(623,160)
FOR Z=0 TO 8 DO
  CLS(Z)=PEEK(COLORS+Z)
OD
RETURN

PROC BLOCK(BYTE I)
BYTE J
FOR J=152 TO 157 DO
  PLOT (I,J)
  DRAWTO(I+5,J)
OD
RETURN

PROC NEWDIR(BYTE A,B)
DX=0
DY=0
IF LOCATE(A+1,B)>0 THEN
  DX=1
ELSEIF LOCATE(A-1,B)>0 THEN
  DX=-1
ELSEIF LOCATE(A,B-1)>0 THEN
  DY=-1
ELSE
  DY=1
FI
RETURN

BYTE FUNC LINE(BYTE A,B)
BYTE Z,J
Z=LOCATE(A+1,B)
J=LOCATE(A-1,B)
Z==+J
J=LOCATE(A,B+1)
Z==+J
J=LOCATE(A,B-1)
Z==+J
IF Z>6 THEN
  RETURN(Z)
FI
NEWDIR(A,B)
RETURN(1)

PROC REMOVE(BYTE A,B)
DO
```

```
   PLOT(A,B)
   A==+DX
   B==+DY
   UNTIL LINE(A,B)<>1
OD
RETURN

PROC GRID()
BYTE I,X,Y,Z,XOLD,YOLD,Y1
COLOR=6
I=2
WHILE I<157 DO
  PLOT(3,I)
  DRAWTO(74,I)
  I==+16
OD
I=3
WHILE I<79 DO
  PLOT(I,2)
  DRAWTO(I,145)
  I==+8
OD
FOR I=2 TO 5 DO
  COLOR=I
  BLOCK((I-2)*10+5)
OD
COLOR=6
PLOT(45,153)
DRAWTO(50,153)
DRAWTO(50,157)
DRAWTO(45,157)
DRAWTO(45,153)
PLOT(51,158)
COLOR=0
FOR I=1 TO 40 DO
  DO
    X=RAND(8)*8+7
    Y=RAND(16)*8+10
    Y1=Y-10
    IF Y1/8=(Y1/16)*2 THEN
      X==+4
    FI
    UNTIL LOCATE(X,Y)<>0
  OD
  XOLD=X
  YOLD=Y
  IF Y1/8=(Y1/16)*2 THEN
    DX=0
    DY=-1
    REMOVE(X,Y)
    DX=0
    DY=1
    REMOVE(XOLD,YOLD)
  ELSE
    DY=0
    DX=-1
    REMOVE(X,Y)
    DY=0
    DX=1
    REMOVE(XOLD,YOLD)
  FI
OD
RETURN

PROC TITLE()
BYTE X,Y,C,K1,K2
CARD SC,J
SC1=PEEKC(88)
GRAPHICS(19)
SC=PEEKC(560)
FOR J=7 TO 9 DO
  POKE(SC+J,7)
OD
POKE(87,2)
COLOR=0
PLOT(0,1)
PRINTDE(6,"COLOR THE SHAPES")
PRINTDE(6," by rebecca guber")
PRINTDE(6," AND SOL GUBER")
POKE(87,3)
FOR J=1 TO 1000 DO
  FOR K2=1 TO 500 DO
  OD
  X=RAND(39)
  Y=RAND(12)+8
  C=RAND(255)
  SOUND(0,C,8,8)
  COLOR=RAND(4)
  PLOT(X,Y)
OD
SOUND(0,0,0,0)
RETURN

BYTE FUNC NEWSPOT(BYTE J,COUNT)
BYTE K,Y1,X1,Z,K1
R(J)==+128
Y1=((J-1)/10)*16+10
X1=((J-1) MOD 10)*8+7
FOR K=0 TO 3 DO
  Z=LOCATE(X1+TX(K),Y1+TY(K))
  K1=J+TEST(K)
  IF Z=0 AND R(K1)=0 THEN
    R(K1)=COUNT
    RETURN(K1)
  FI
OD
RETURN(0)

BYTE FUNC OLDSPOT(BYTE J,COUNT)
BYTE K,K1
R(J)==-128
K=3
WHILE K<>255 DO
  K1=J+TEST(K)
  IF K1>0 AND K1<100 THEN
    IF R(K1)>128 THEN
      R(K1)==-128
      RETURN(K1)
    FI
  FI
  K==-1
OD
RETURN(0)

PROC FIND(BYTE J,COUNT)
BYTE K,K1
R(J)=COUNT
DO
  K=NEWSPOT(J,COUNT)
  IF K=0 THEN
    K1=OLDSPOT(J,COUNT)
    J=K1
  ELSE
    J=K
  FI
  UNTIL J=0
OD
RETURN

PROC SEARCH()
BYTE J,COUNT,K,K1
ZERO(R,100)
COUNT=1
FOR J=1 TO 89 DO
  IF R(J)=0 AND J MOD 10<>0 THEN
    FIND(J,COUNT)
    COUNT==+1
  FI
OD
FOR J=1 TO 89 DO
  IF R(J)>128 THEN
    R(J)==-128
  FI
OD
RETURN
```

```
; PMG.ACT FROM THE ACTION! TOOLKIT

INCLUDE "D1:PMG.ACT"


BYTE FUNC SIZE(BYTE K)
BYTE J
FOR J=K+1 TO K+9 DO
  IF R(J)=COUNT THEN
    RETURN(1)
  FI
OD
RETURN(0)


PROC CHECK_BOARD()
BYTE J,K
COUNT=1
FOR J=1 TO 99 DO
  IF J MOD 10 <>0 THEN
    WHILE R(J)<COUNT AND J<100 DO
      J==+1
    OD
    GAR(COUNT)=J/10
    K=(J/10)*10+10
    WHILE SIZE(K)=1 DO
      K==+10
    OD
    GAR(COUNT)==+(K-10)
    COUNT==+1
  FI
OD
COUNT==-1
RETURN

PROC SHIFT(BYTE X1)
BYTE Z,Z1
IF X1=140 THEN
  QUIT=1
  RETURN
FI
Z=(X1-60)/20+1
COL=Z+1
Z1=PEEK(705+Z)-6
POKE(705,Z1)
RETURN

PROC BEEP()
CARD Q
SOUND(0,220,10,10)
FOR Q=1 TO 25000 DO
OD
SOUND(0,0,0,0)
RETURN

BYTE FUNC PICK_COLOR()
BYTE S,TR,J,X1
CARD I1
FOR I1=OLDY TO 173 DO
  PMMOVE(1,X,I1)
OD
OLDY=173
PRINTE("PLEASE PICK A COLOR")
X1=60
PMHPOS(1)=60
IF CFLAG=1 THEN
  DO
    J=PEEK(624)
    IF J>5 THEN
      J=(J/50)*20+60
      PMHPOS(1)=J
    FI
    IF PEEK(636)=0 OR
       PEEK(637)=0 THEN
      BEEP()
      SHIFT(J)
      RETURN(1)
    FI
  OD
FI
DO
  DO
    S=STICK(0)
    TR=STRIG(0)
    IF TR=0 THEN
      BEEP()
      SHIFT(X1)
      RETURN(1)
    FI
    UNTIL S<>15
  OD
  IF S=7 THEN
    X1==+20
    IF X1=160 THEN
      X1=60
    FI
  FI
  IF S=11 THEN
    X1==-20
    IF X1=40 THEN
      X1=140
    FI
  FI
  PMHPOS(1)=X1
  FOR I1=1 TO 6000 DO
  OD
OD
RETURN(1)

BYTE FUNC GOOD_COLOR(BYTE SPOT,COL)
BYTE TOP,BOT,BLOCK,I
BLOCK=R(SPOT)
TOP=GAR(BLOCK)
BOT=(TOP MOD 10)*10
TOP=(TOP/10)*10
WHILE BOT<TOP+9 DO
  IF R(BOT)=BLOCK THEN
    FOR I=0 TO 3 DO
      IF B(BOT+TEST(I))=COL THEN
        RETURN(0)
      FI
    OD
  FI
  BOT==+1
OD
RETURN(1)

PROC FILLER(BYTE J)
BYTE X,Y,K,L,L1
L1=6
IF R(J)=R(J+1) THEN
  L1==+1
FI
L=14
IF R(J)=R(J+10) THEN
  L==+1
FI
X=(J/10)*16+3
Y=(J MOD 10)*8-4
FOR K=X TO X+L DO
  PLOT(Y,K)
  DRAWTO(Y+L1,K)
OD
RETURN

PROC FILL_IN(BYTE SPOT)
BYTE N,TOP,BOT,J
N=R(SPOT)
TOP=GAR(N)
BOT=TOP MOD 10
TOP=(TOP/10)*10
USED(N)=1
```

(Action! listing continues on page 88)

Color the Shapes

continued from page 40

```
FOR J=BOT TO TOP+9 DO
  IF R(J)=N THEN
    B(J)=COLOR
    FILLER(J)
  FI
OD
RETURN

PROC INIT()
BYTE K,J,M,N,C
ZERO(PLAYER,20)
ZERO(B,99)
ZERO(USED,60)
PUT(125)
PRINTE("1 OR 2 PLAYERS?")
PLAYNUM=INPUTB()
PRINTE("WHAT IS YOUR NAME?")
INPUTS(A)
FOR K=1 TO A(0) DO
  PLAYER(K)=A(K)
OD
IF PLAYNUM=2 THEN
  PRINTE("NAME OF 2ND PLAYER?")
  INPUTS(A)
  FOR K=1 TO A(0) DO
    PLAYER(K+10)=A(K)
  OD
FI
PUT(125)
PRINT("USE A KOALA PAD (Y/N)?")
CFLAG=0
INPUTS(A)
IF A(1)='Y THEN
  CFLAG=1
FI
PRINTE("FILL SOME SHAPES IN?")
INPUTS(A)
IF A(1)<>'Y THEN
  RETURN
FI
PUT(125)
PRINTE("HOW MANY SHAPES, UP TO 5?")
J=INPUTB()
J==MOD 6
FOR K=1 TO J DO
  DO
    M=RAND(COUNT-1)+1
    UNTIL USED(M)=0
  OD
  N=M
  DO
    N==+1
    UNTIL R(N)=M
  OD
  DO
    C=RAND(4)+2
    UNTIL GOOD_COLOR(N,C)=1
  OD
  COLOR=C
  FILL_IN(N)
  USED(M)=1
OD
RETURN

BYTE FUNC SGN(BYTE I,J)
IF I=J THEN
  RETURN(0)
ELSEIF I>J THEN
  RETURN(-1)
FI
RETURN(1)

PROC MOVE()
BYTE Q,DEL
CARD K
IF OLDX<>X THEN
  Q=OLDX
  DEL=SGN(OLDX,X)
  WHILE Q<>X DO
    PMMOVE(1,Q,OLDY)
    Q==+DEL
  OD
  OLDX=X
  FOR K=1 TO 2000 DO
  OD
FI
IF OLDY<>Y THEN
  Q=OLDY
  DEL=SGN(OLDY,Y)
  WHILE Q<>Y DO
    PMMOVE(1,X,Q)
    Q==+DEL
  OD
  OLDY=Y
FI
RETURN

BYTE FUNC TRIGGER()
IF CFLAG=1 THEN
  IF PEEK(636)=0 OR PEEK(637)=0 THEN
    RETURN(0)
  FI
ELSE
  IF STRIG(0)=0 THEN
    RETURN(0)
  FI
FI
RETURN(1)

BYTE FUNC ABS(BYTE A,B)
IF A>B THEN
  RETURN(A-B)
FI
RETURN(B-A)

BYTE FUNC JOYSTICK()
BYTE P,X1
IF CFLAG=1 THEN
  X1=PEEK(624)
  Y1=PEEK(625)
  IF X1<5 OR Y1<5 THEN
    RETURN(0)
  FI
  X1=56+(X1/28)*16
  Y1=36+(Y1/28)*16
  IF ABS(X1,OLDX)<5 THEN
    RETURN(0)
  ELSEIF ABS(Y1,OLDY)<5 THEN
    RETURN(0)
  FI
  X=X1
  Y=Y1
  RETURN(1)
FI
P=STICK(0)
IF P=15 THEN
  RETURN(0)
FI
IF P=11 AND OLDX>60 THEN
  X=OLDX-16
  RETURN(1)
ELSEIF P=7 AND OLDX<180 THEN
  X=OLDX+16
  RETURN(1)
ELSEIF P=14 AND OLDY>51 THEN
  Y=OLDY-16
  RETURN(1)
ELSEIF P=13 AND OLDY<152 THEN
  Y=OLDY+16
  RETURN(1)
FI
RETURN(0)

BYTE FUNC COMPLETE()
BYTE J
FOR J=1 TO COUNT-1 DO
```

```
   IF USED(J)=0 THEN
      RETURN(0)
   FI
OD
RETURN(1)

PROC NAME()
BYTE J
PUT(125)
FOR J=TURN*10+1 TO TURN*10+10 DO
   PUT(PLAYER(J))
   IF PLAYER(J+1)=0 THEN
      EXIT
   FI
OD
PRINTE("'S TURN")
RETURN

PROC COLOR_IN(BYTE SPOT)
BYTE K
CARD K1
IF B(SPOT)<>0 THEN
   DO
      UNTIL PICK_COLOR()<>0
   OD
   MOVE()
   IF QUIT=1 THEN
      RETURN
   FI
   X=OLDX
   Y=OLDY
   MOVE()
   RETURN
FI
IF GOOD_COLOR(SPOT,COL)=0 THEN
   BEEP()
   PRINT("YOU CANNOT USE THAT")
   PRINTE(" COLOR THERE")
   BEEP()
   RETURN
FI
COLOR=COL
FILL_IN(SPOT)
IF PLAYNUM=2 THEN
   TURN==! 1
FI
NAME()
FOR K1=1 TO 2000 DO
OD
RETURN

PROC SHAPES()
BYTE A,SPOT,J
DO
   TITLE()
   GRAPHICS(8)
   QUIT=0
   PMGRAPHICS(1)
   SETUP()
   POKE(705,22)
   POKE(623,160)
   PMCLEAR(1)
   MAKEPM(STAR,14,1,2,156,126)
   X=56
   Y=36
   OLDX=0
   OLDY=0
   MOVE()
   COLOR=3
   COL=3
   GRID()
   TURN=0
   SEARCH()
   CHECK_BOARD()
   INIT()
   NAME()
   DO
      IF TRIGGER()=0 THEN
         COLOR_IN(SPOT)
      FI
      IF JOYSTICK()=1 THEN
         MOVE()
      FI
      SPOT=(X-38)/16+10*(Y-36)/16
      UNTIL COMPLETE()=1 OR QUIT=1
   OD
   IF COMPLETE()=1 THEN
      FOR J=TURN*10+1 TO TURN*10+10 DO
         PUT(PLAYER(J))
         IF PLAYER(J+1)=0 THEN
            EXIT
         FI
      OD
      PRINTE(" IS THE WINNER")
   FI
   PRINTE("PLAY AGAIN?")
   A=INPUTB()
   UNTIL A='N
OD
RETURN
```

●