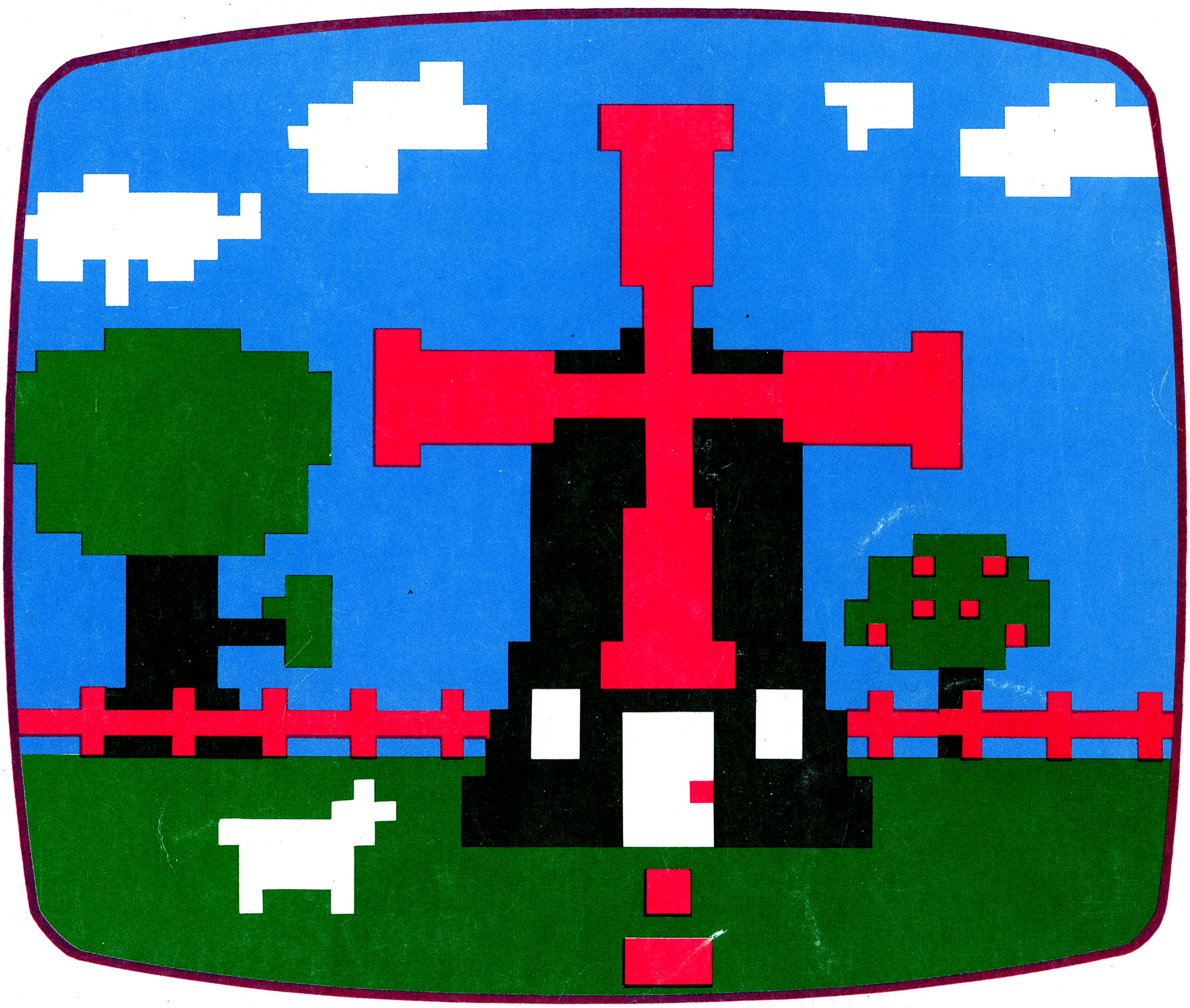
Introduction to Low Resolution GRAPHICS

How to draw lines, create shapes, animate figures, prepare charts for business or pleasure.



SCELBI Publications

$2 –

Introduction to Low Resolution GRAPHICS

By Nat Wadsworth

SCELBI Publications
20 Hurlbut Street, Elmwood, CT 06110

IMPORTANT NOTICE

The information in this manual has been carefully reviewed and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies or for the success or failure of various applications to which the information contained herein might be applied.

Foreword

There are lots of small computers available today that are capable of displaying information graphically in at least a low resolution mode. This means that information can be summarized by a computer and placed in a visual format that is entertaining to people.

Alas, while this capability is provided, it appears that few people are using it. This is a shame. Why isn't the average user of a small personal computer capitalizing on this power? I hope it is because they have simply not been introduced to the easily understood techniques that may be used to produce graphic displays on their machines.

The purpose of this publication is to get users started utilizing low resolution graphics as a means to liven up the interface between people and computing machines. Only the simplest of techniques are presented here to get across the fundamentals. Once mastered, the enthusiastic initiate can call on his or her own artistic talents to further the craft. Indeed, much of the fun of computer graphics is that the personal tastes and preferences of the individual programmer can be expressed on the video screen.

Right now there are thousands of individuals dabbling in the area of creating programs that utilize low resolution graphics. We shall really start to make progress when there are hundreds of thousands of people who are comfortable with the art.

I urge you to get started now. To enjoy the thrill of being a pioneer in an exciting area of the application of small computers serving individual people. An area where the creative talents of individuals can do much to advance the art as a whole. Low resolution graphics capability has much to offer. Use it for all it is worth.

Nat Wadsworth

September, 1979

ACKNOWLEDGEMENT

I would like to thank all the people at SCELBI Publications for their continued dedication to excellence in an area of publishing that is most demanding. Their technical and production people are most helpful in working with me to get my manuscripts and programs accurately reproduced in book form.

Julie MacGregor at SCELBI must receive special accolades for her tireless devotion to getting this book into production in a very short amount of time.

Contents

Introduction

Learning By Doing

Many small computer systems sold today have at least a limited form of graphics capability. The Commodore PET, the Radio Shack TRS-80 and the APPLE II, for instance, are all able to at least display or "plot" at a designated point in a display matrix. (Some of these units, with appropriate software, can provide much more complex types of graphics capability.)

Because a machine is capable of doing something doesn't mean that it is going to do it! People have to know several things before a personal computer is going to utilize its graphics capabilities effectively. They have to know how the machine does it and they have to tell the machine to do it! It seems that we are in the stage of personal computer use where not too many people understand how to utilize a small system's graphics capability. I hope to change that a little bit through this publication!

"Because a machine is capable of doing something, doesn't mean that it is going to do it!"

The discussion that follows will be aimed at showing users how a computer, equipped with what is commonly referred to as "low resolution" graphics capability, can be programmed to provide interesting and entertaining displays. This will be done here by developing a specific "game" program in a step-by-step manner that is fashioned around simple graphics effects.

What do I mean by "low resolution" graphics? I mean any system (such as an APPLE II, Commodore PET or Radio Shack TRS-80) that is capable of controlling the display on a television, video monitor or other type of cathode-ray-tube device so that it causes a "point" in a matrix to be on or off or light up with one color or another. To be "low resolution" it is assumed that the number of points in the matrix is on the order of a few thousand points or less. For instance, an APPLE-II system when operating in the normal low resolution graphics mode has a display matrix that consists of 40 points in the X direction and 40 in the Y direction for a total of 1600 points on the screen.

What type of game will we be developing in this publication? Well, it is a game of football. An interactive game of football that pits a person against the well designed capriciousness of a computer. A game of football that entertains the player with graphic action, and yes, one that can include sound effects, too!

"... you will have the knowledge to allow you to be 'graphics boss' of your own machine."

More important than the fact that we will develop a " football" game in this publication is the fact that you will learn how to form images, then make them move and, if desired, even emit sounds. With this knowledge you will be in a position to go off on your own. Then you can create your own animated version of football or parcheesi. Or, you can create cartoons. Or, if your line is of a more serious nature, you can chart and draw business graphs or represent chemical structural diagrams or draw simple electronic schematics. In other words, you will have the knowledge to allow you to be "graphics boss" of your own machine. That is what you really want anyhow, right? Then keep reading on.

Chapter 1

Getting Started

Let's All Start on the Same Foot

There is the old story of the army sergeant who was reviewing and attempting to discipline a group of new recruits. He had them all lined up for inspection. "*Attention!*" he bellowed. His greenhorns nervously stood their straightest. "Now everybody raise their right foot!" he shouted in preparation for a boot inspection. The sergeant looked down the long line of raised legs, then his expression changed to one of disbelief. "All right." he roared, "Who is the wise guy that has *both* of his feet up!?"

We don't want that kind of situation here. So, let's start by making sure that we all understand the basic organization of a display matrix and the kinds of commands or directives we can use to control the contents of the matrix.

Figure 1 illustrates a hypothetical matrix of a display that I will use for initial discussion purposes. You should note that the drawing presents a grid composed of 16 rows and 16 columns of squares. There are thus 256 squares in the grid or matrix. In order to talk about individual squares within this grid it is necessary to designate some reference point and correlate our discussions to this point.

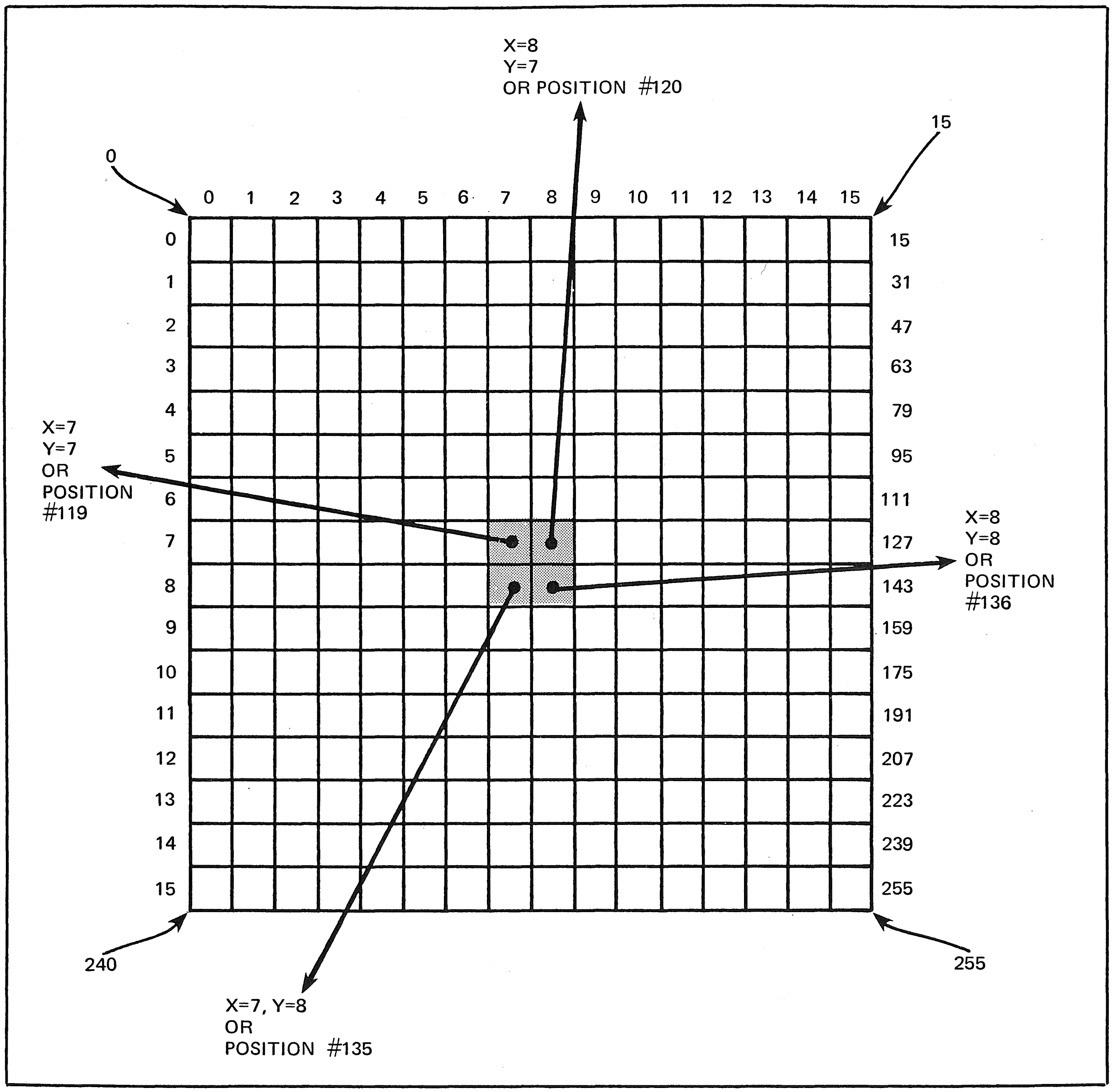
A method that has done very well for mathematicians for many years is to call one side of the grid the "X" side and an *adjacent* side the "Y" side. A point along each such designated side can be specified as the zero reference point and all other points along that side referred to that point.

Now a likely place from common experience to use as a zero reference point for the grid in Figure 1 would be at the lower left-hand corner of the matrix. Squares could then be designated as being to the right of the zero point along the X side or axis and towards the top from the zero point along the Y side.

You will note please, however, that I have not labeled the diagram in such a fashion. Rather, the illustration shows a zero reference point at the *top* left-hand corner of the diagram. While this still allows us to reference points along the X axis to the right of the starting point, note that along the Y axis we will be going *down* (instead of up) from the reference point.

Now why, you may be asking, would I try to confuse you right at the start like that? I am not trying to confuse you. I am, alas, trying to show you how most of the designers of popular cathode-ray-tube (abbreviated CRT hereafter) displays like to refer to the operation of their creations. In the typical system, the electron beam

Figure 1



sweeps lines from left to right. Each line is placed below the previous one. Thus, it is convenient to start the reference point at what may seem an unconventional location on the screen. But, that is how it is, so to stay consistent in this field, that is how I will present things in our discussions.

"In the typical system, the electron beam sweeps lines from left to right."

To summarize, points on a CRT matrix can be referenced from the *upper left-hand corner* along an "X" axis that increases to the right and a "Y" axis that increases as we go *down*.

Thus, in the diagram, we could define the shaded squares in the center of the matrix to be residing at the four positions: X =7, Y = 7; X = 8, Y = 7; X = 7, Y = 8 and X = 8, Y = 8.

Aha! But that is not the only way that we could reference positions of squares within the grid. Another way would be to assign each square in the grid a number. Some popular small computer systems do just this. In the example, I have indicated that the squares in the top row could be designated as boxes 0 through 15. Boxes in the row beneath it could be referred to by the numbers 16 through 31. In the next lower row they would be numbered 32 through 47. The bottom row in the diagram would have numbers 240 through 255.

Oh yes, please note in the illustration that I have shown the first position in a column or row as being position zero. I presume that anyone reading this publication is familiar with the custom of computer users to assign the reference zero to the first address of memory, etc. Thus, I won't do anything more here than to caution you to watch out and *think* when we are talking about items versus referencing their positions. Remember, the *fifteenth* item in a line will be referenced as being in the *fourteenth* cell because the *first* item will be residing in cell number *zero!* It gets to be tricky stuff sometimes, but again, it is the convention established among computer users.

Plot It or Poke It

Remember here that Figure 1 refers to a hypothetical display matrix. Most systems will have a considerably larger display grid with which to work.

Suppose all the points in the diagram of Figure 1 were "turned off" or "not illuminated" by the computer and we wanted to have the four shaded squares or "points" on our display lit? The computer language of some systems would allow us to do that by making statements something along the following lines:

PLOT 7,7

PLOT 8,7
PLOT 7,8
PLOT 8,8

Or, in general by using a "PLOT X,Y" directive where X and Y represent distances along a corresponding axis.

Some systems, such as an APPLE-II, will allow us to designate the color (or, in black and white displays, the intensity) of points by preceding PLOT statements with a color-designating statement, such as "COLOR=Z", where the variable Z represents an allowable number that specifies a certain color. Thus, with an APPLE-II system, one could get a blue, orange, green and a yellow square at the four shaded positions in Figure 1 by issuing directives along the lines of

"Some systems, such as an APPLE-II, will allow us to designate the color of points. . . ."

COLOR=6
PLOT 7,7
COLOR=9
PLOT 8,7
COLOR=12
PLOT 7,8
COLOR=13
PLOT 8,8

Other systems might require that the illumination of a spot be indicated by referencing the position's number. Thus, for the example, one might need to designate instructions such as

ON 119,120,135,136

Still other systems make things just a tad more complicated. Sometimes it is necessary to directly place data into a specific memory location. The types of displays we are dealing with in this discussion are driven out of sections of memory reserved for such purposes. The CRT can be viewed as an image of this section of memory. Indeed, the hardware portion of such a system simply keeps scanning the corresponding display memory buffer area and decodes the data there to turn the display on or off at matching positions on the screen.

In such systems it is generally necessary to know the memory address at which the display buffer starts. From that point one adds a displacement value to reach the location of the byte that is to be activated. For instance, in a Commodore PET unit, a memory buffer starts at address 32768. To get the positions in our hypothetical display grid to light up in such a system, we would need to do something along the following lines: Add the display position number to the base memory address and place a specific data code into that memory location. What data code? The data code for the type of character or graphic symbol we want to see displayed! The PET has a choice of several hundred such symbols. How do you put the data into the memory location? One such way is to use a BASIC language "POKE" or equivalent directive.

"... in a Commodore PET unit, a memory buffer starts at address 32768."

So, if we wanted to fill the four shaded squares in our example diagram with a symbol, such as an asterisk (*), we would need to poke the code for an asterisk (42) into memory locations that were offset 119, 120, 135 and 136 units from the CRT buffer's base address of 32768. Thus, we would need to perform a series of directives such as

```
POKE 32887,42
POKE 32888,42
POKE 32903,42
POKE 32904,42
```

Remember, the grid in our example is only 16 by 16 units. Actually poking data into the memory addresses just calculated on a Commodore PET unit would not result in the four cells being adjacent, as the display matrix is larger than 16 by 16 units. It is, in fact, 40 (X axis) by 25 (Y axis) units. Given this information, as an exercise can you determine just where in the display the data would appear if the addresses used in this discussion were actually used?

Accessing the Screen

In this discussion, what I mean by "accessing the screen" is simply finding out the status of a display using the computer. (Obviously, *I* can find out the status of the display directly by looking at it! Trouble is, I wouldn't be able to manipulate what I saw as fast as I might like to. The computer can do so much better in that area, so why not let it?)

This capability is generally provided in the form of instructions that are sort of the inverse of a POKE or PLOT command. For instance, with a Radio Shack TRS-80 unit you can use a statement "POINT (X,Y)" to determine the display status of a cell. This directive will return a zero value if the corresponding position on the display is turned off (not illuminated). It returns a nonzero value (such as −1) if the spot is illuminated when the statement is executed.

Note that the POINT statement uses the X and Y axes as references.

This is similar to the method used on the APPLE-II. Here the statement "SCRN(X,Y)" returns a value from 0 to 15. Each value represents the color being displayed at the specified matrix position. The value 0 corresponds to black (or off as it is essentially "no color"). The other figures represent the various colors the APPLE-II is capable of generating.

"The opposite of a POKE statement is the 'PEEK(N)' directive."

The opposite of a POKE statement is the "PEEK(N)" directive. Here N stands for the value of an address in memory. The PEEK statement returns the contents of the memory address specified. In a system such as the Commodore PET, one can determine the status of a point in the display by peeking at an address in the display's memory buffer. Doing so will return a number corresponding to the character or symbol code being displayed. One or more of these character codes, such as "32" for a "space," represents a "no display" or unilluminated condition at that point on the CRT!

Chances are your computer system uses one of the types of directives mentioned (or something similar) to activate or deactivate a position on the screen. At least now we have a common language or shorthand from which to start a more detailed discussion of developing graphic displays.

Chapter 2

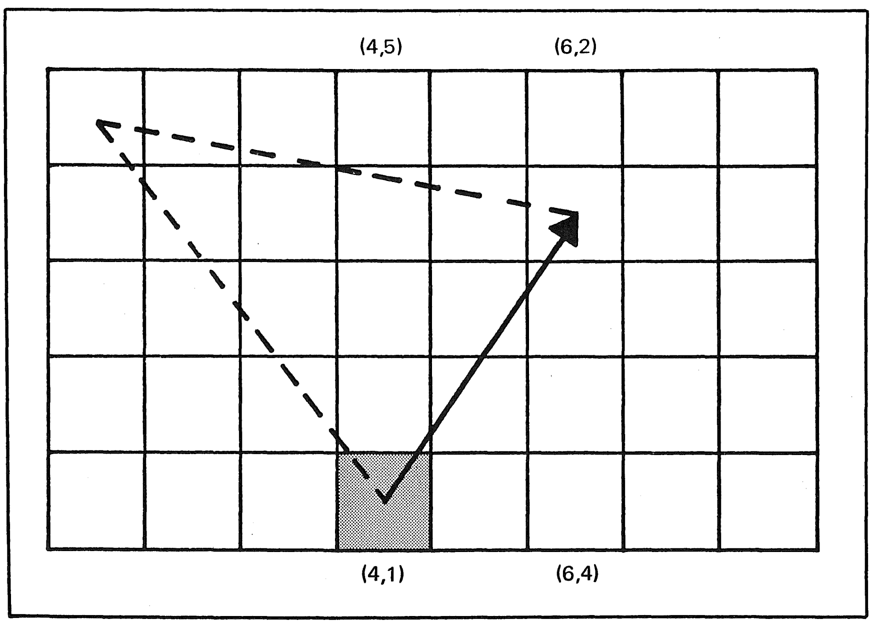
A Whole Chapter on Math

Math — How Much Is Really Necessary?

Nothing more complicated than high school algebra is needed in order to get started drawing graphic figures. That is all we shall use in this publication. Of course, if mathematics is your bag, you will undoubtably see how more sophisticated mathematical principles could be used to good advantage in various situations. And, indeed, if you want to graph or draw mathematical functions you will have to have an understanding of what it is you want to represent. It will be presumed in such cases, that the reader will be as prepared as one's interests leads him/her.

Now, as I have said, in order to do just about anything using graphics, it is necessary to call upon some basic high school algebra techniques.

For instance, the 8 by 5 block of squares shown in Figure 2 could be referenced using the standard cartesian coordinate



system. Suppose one wanted to move the shaded square shown as residing at (4,1), meaning X = 4 and Y = 1, so that it resided at the point (6,4) indicated by the arrow. To a viewer not familiar with the workings of a computer, it would seem that to reach the point (6,4) one would simply move 2 squares in the positive direction along the X axis and 3 squares up (positive), along the Y axis from the starting point at (4,1).

You, however, having read this far, know that to make this move using the typical low resolution graphing capabilities of the type we have been discussing, we have to direct the machine to do something slightly different! Namely, the computer must reference the move along the Y axis for the example to be *negative* in direction!

How is that?

"The computer . . . 'thinks' that going down along the Y axis is a positive direction."

(Remember, the computer would start its scan from the top left-hand square (instead of the bottom left-hand square). The computer also "thinks" that going down along the Y axis is a positive direction. Accordingly, the shaded square to the computer is referenced by the computer as (6,2). To go from the shaded square to the location pointed to by the arrow, the computer will calculate a move of 2 squares in the positive direction along the X axis. It will view the move along the Y axis as being 3 units in the *negative* direction! Study the diagram and review the concepts carefully here until you are sure you understand the translation! (Note in this example that the computer begins at (1,1) and not at (0,0) as would normally be the case. This reference point was chosen to simplify this illustration.)

Thus, for instance, if the computer had already displayed the shaded point in the diagram, one could have it display the location pointed to by the arrow using a statement on an APPLE-II system such as:

```
PLOT X+2,Y−3
```

Note that X and Y in this statement refer to the coordinates of the last point (the shaded square in the example) displayed by the computer.

If one was interfacing with a user who wished to give directions for the move in reference to the standard cartesian starting point (bottom right square), one would only have to have the computer make the simple translation: Change all positive moves along the Y axis to negative and vice versa. Thus, when the operator said, "Move

3 units in the positive direction along the Y axis," the computer would translate that to mean, "Move 3 units (still up here!) in the negative direction."

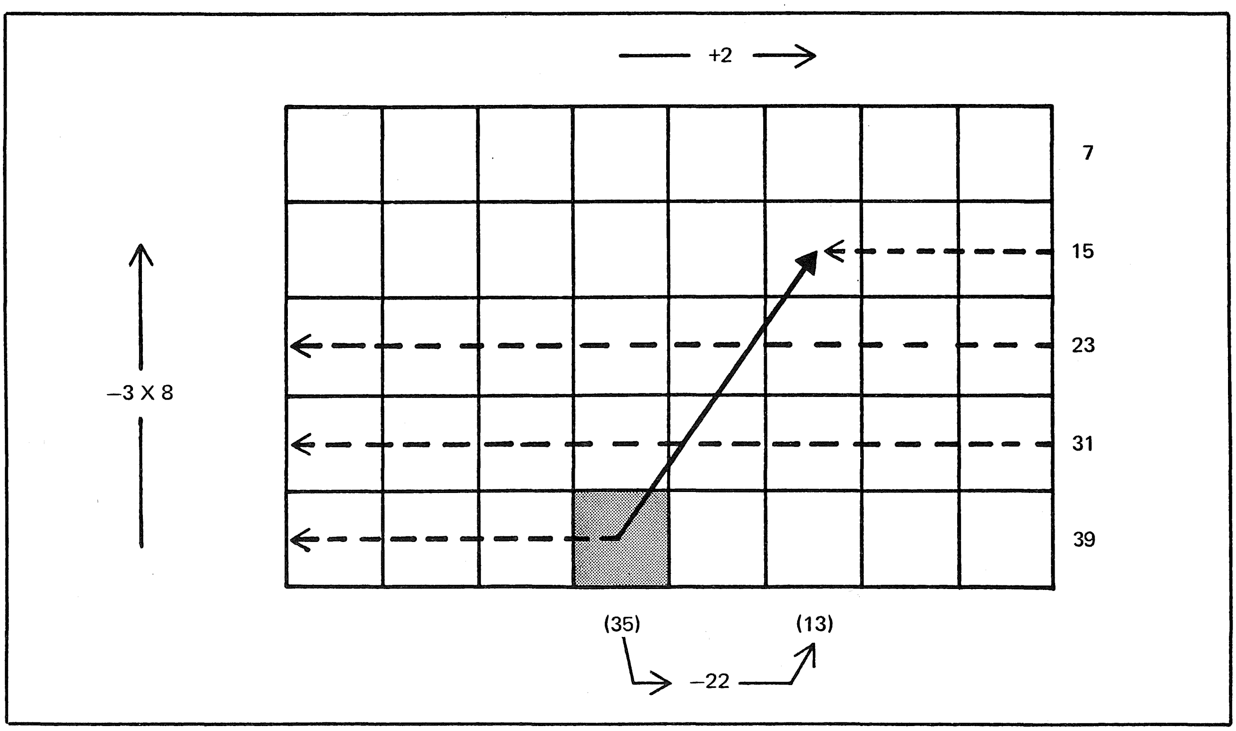
As discussed earlier, some computer graphing systems do not allow positions to be simply identified by defining X and Y coordinates. Some units assign a number to each square. The number of a square becomes a function of the number of squares in a row in such cases. Figure 3 illustrates the same type of move being made as has been discussed for Figure 2. That is, the location pointed to by the arrow is 2 units to the right of and 3 units above the shaded square. In a system that assigned numbers to the squares (beginning in the top left-hand corner at square "zero"), the diagram depicts the shaded square as being number 35. The arrow points to the square assigned number 13.

". . . some computer graphing systems do not allow positions to be simply identified by defining X and Y coordinates."

To translate the move from cartesian coordinates in the example, one would need to proceed in the following manner.

First, determine the number of rows (up or down) between the starting and ending point. In this example, three rows separate the two locations. The number of rows that separate the points will become a multiplier value.

Next, it is necessary to determine a sign, positive or negative, for the "row multiplier." In keeping with our computer-oriented



convention, to move up on the screen means a minus direction, as it is moving back towards the reference point (at the upper left-hand corner of the screen). Thus, in the example of Figure 3, the sign of the row multiplier will be minus or negative.

Now the number of columns in a row is multiplied by the "row multiplier." In this example (–3 X 8) = –24.

Finally, this value (–24 here) is augmented by the offset in rows between the two points. Note that here also the offset is a "signed" (positive or negative) value. For the example, it is positive if to the right of the starting point and negative if to the left. If the starting and ending squares are in the same column, then the offset is zero. Figure 3 has an offset value of +2. Adding the offset value (+2) to the row multiplier (–24) yields a result of (–22). As illustrated in the diagram, taking 22 from 35 yields 13.

". . . the offset is a 'signed' (positive or negative) value."

Thus, in a PET or similar computer where a display is controlled by the contents of a display buffer in memory, assuming the display buffer was organized only as a 5 by 8 matrix, the shaded square in the diagram would be illuminated by a command such as POKE BASE+35,42. The arrow on the diagram would be illuminated by a directive such as POKE BASE+35–22,42 (or POKE BASE+13,42). Again, for the sake of clarity, Figure 3 has assumed a starting point at (1,1) instead of the usual (0,0).

Be sure and study the discussion of Figures 2 and 3 before proceeding further in this manual. It is crucial for further understanding that these fundamental concepts be understood.

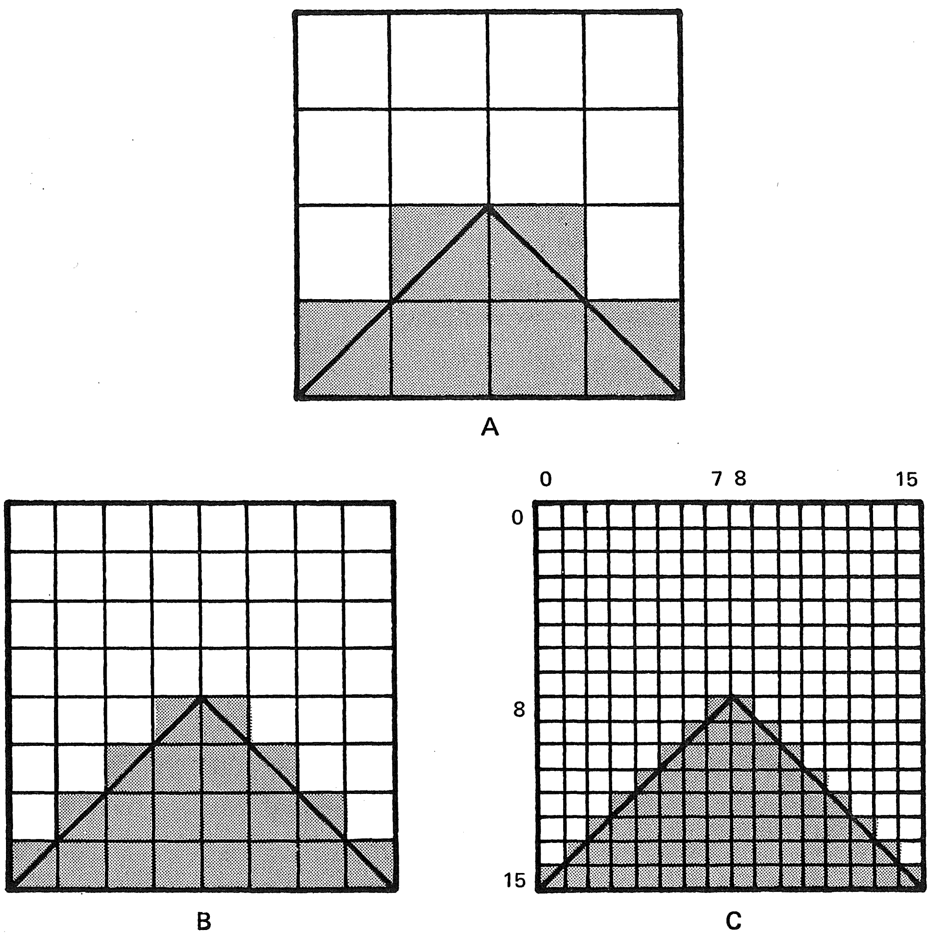
Chapter 3

Drawing Simple Shapes

Getting Started

We can draw pictures of simple objects by putting a number of points together. Figures 4 and 5 illustrate a number of ways in which we could form the representation of a triangle. The diagrams in those

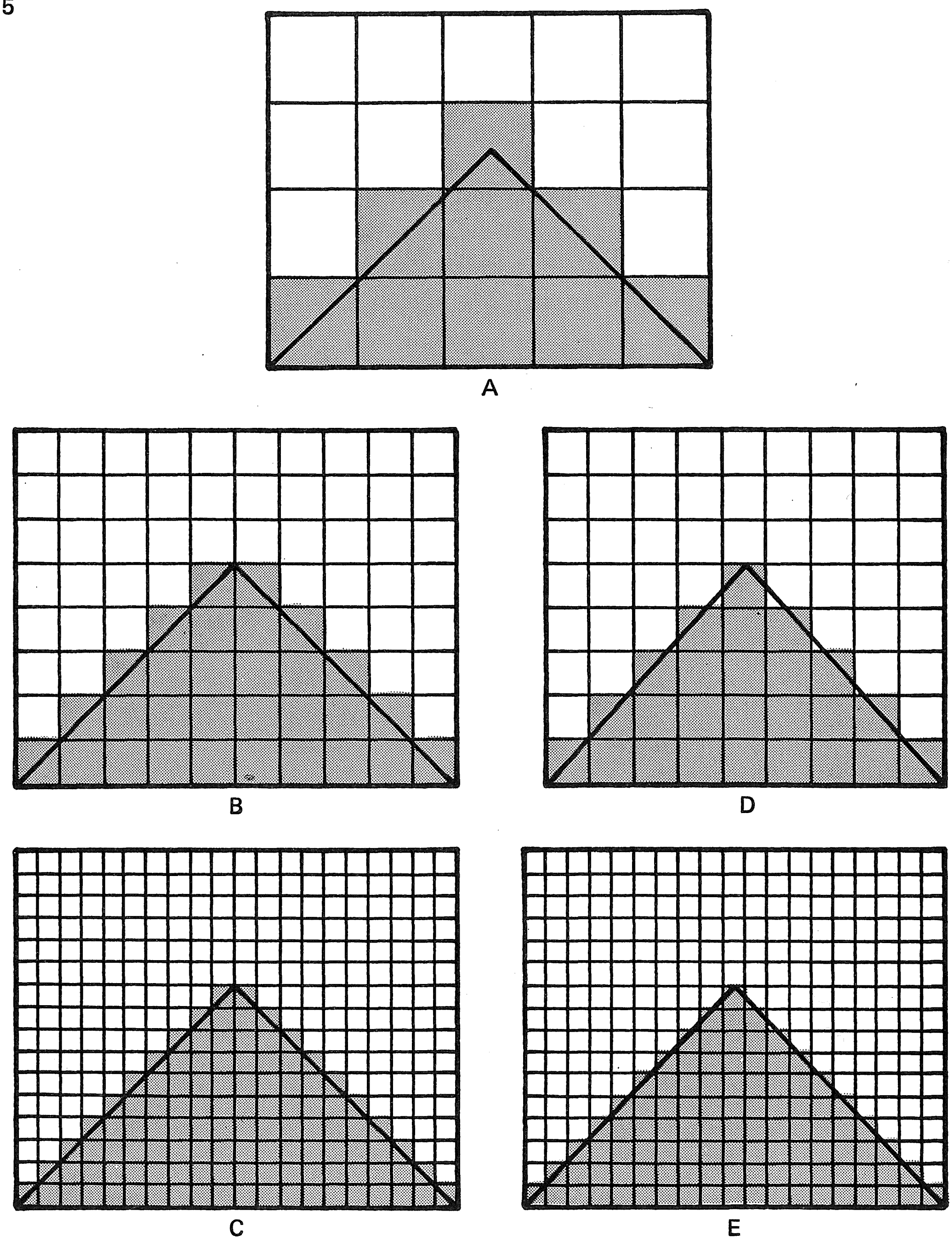
Figure 4



figures also highlight some subtle points about creating low resolution pictures.

Want to get started? OK, if you have an APPLE-II system, try executing the following directives in BASIC language

Figure 5



```
GR
COLOR=13
PLOT 2,3
PLOT 3,3
PLOT 1,4
PLOT 2,4
PLOT 3,4
PLOT 4,4
```

Issuing these directives will result in the crude representation of the triangle shown in Figure 4A to be drawn on the screen!

Don't have an APPLE-II system? Then try this on a Radio Shack TRS-80

```
CLS
SET (2,3)
SET (3,3)
SET (1,4)
SET (2,4)
SET (3,4)
SET (4,4)
```

On a Commodore PET you would need to do something like

```
(Strike key to clear the screen)
A=32768
POKE A+81,
POKE A+82,
POKE A+120,
POKE A+121,
POKE A+122,
POKE A+123,
```

If your system is like any one of these, you can issue the corresponding statements to obtain similar results. Do you see what is being done? The reference point for these diagrams is taken as the top left-hand corner. That point is designated as being at X=0 and Y=0 on the screen. Figures 4A and 5A represent "magnified" views of a crude triangle being constructed from just a few illuminated squares.

Do you notice a difference between Figures 4A and 5A? Sure you do. Figure 4A uses an even number of squares in both the vertical and horizontal directions. Figure 5A draws the triangle using an odd number of sectors in both directions. Do you think one looks better than the other? Why not figure out the statements needed to draw Figure 5A on your display and compare the two versions? The two drawings are provided to illustrate a simple point. Some pictures can be drawn better if an odd number of sectors are used rather than an even number, or vice versa. Remember this when you start creating pictures on your own. If you can't get the desired shape or effect with one attempt, try redrawing the diagram using one more or less sector in one or both dimensions. It's simple, but it sure can work wonders at times.

"Some pictures can be drawn better if an odd number of sectors are used rather than an even number."

If we increase the number of points we illuminate, our object will get larger. It may also appear to get "smoother" in appearance. Compare Figure 4A with 4B or 5A to 5B. Try drawing these figures on your screen by expanding the concept used to draw Figure 4A. You might want to note an interesting phenomena by examining Figures 5A and 5B. When the number of squares in Figure 5A is doubled to that shown in 5B, the number of sectors in the X direction switches to an even value! If you want to keep the aesthetic value of using an odd number of squares in the X direction, you would have to settle for slightly more or less than a doubling in the X dimension as illustrated by Figure 5D.

If you try drawing Figure 4B or 5B on your screen, you will soon learn firsthand the effects of doubling both dimensions. The number of points that must be plotted is more than doubled! It soon becomes apparent that a better method than individually specifying all the points to be drawn is desirable.

'If we increase the number of points we illuminate, our object will get larger."

Figure 4C shows a triangle that is four times larger than in 4A. Do you really want to specify all the points, on a one-by-one basis, that need to be illuminated in order to create that triangle? Not likely! It is time to call on your computer and BASIC language to do some of the work for you. Listing 1 shows one way of drawing the triangle in Figure 4C using a series of BASIC statements grouped as a subroutine. The first version in Listing 1 is for an APPLE-II system. Statement line number 5 calls the subroutine that commences at line 10.

Similar listings are included to draw the same type of figure on a Radio Shack TRS-80 and a Commodore PET. Note that the listing for the PET requires a conversion from X and Y coordinates to the linear addressing scheme utilized by that system's graphics. The conversion used in the POKE statements is A+X+Y*40. A is defined as the starting address of the display buffer. This simple

Listing 1

```
1 GR: COLOR=13
4 BASEX=0:BASEY=0
5 GOSUB 10
6 END
10 Y=BASEY+8
20 FOR X=BASEX+7 TO BASEX+8: PLOT
   X,Y: NEXT X
30 Y=Y+1: FOR X=BASEX+6 TO BASEX+
   9: PLOT X,Y: NEXT X
40 Y=Y+1: FOR X=BASEX+5 TO BASEX+
   10: PLOT X,Y: NEXT X
50 Y=Y+1: FOR X=BASEX+4 TO BASEX+
   11: PLOT X,Y: NEXT X
60 Y=Y+1: FOR X=BASEX+3 TO BASEX+
   12: PLOT X,Y: NEXT X
70 Y=Y+1: FOR X=BASEX+2 TO BASEX+
   13: PLOT X,Y: NEXT X
80 Y=Y+1: FOR X=BASEX+1 TO BASEX+
   14: PLOT X,Y: NEXT X
90 Y=Y+1: FOR X=BASEX TO BASEX+
   15: PLOT X,Y: NEXT X
100 RETURN
```

```
1 CLS
4 B1=0:B2=0
5 GOSUB 10
6 END
10 Y=B2+8
20 FOR X=B1+7 TO B1+8:SET (X,Y):NEXT X
30 Y=Y+1:FOR X=B1+6 TO B1+9:SET (X,Y):NEXT X
40 Y=Y+1:FOR X=B1+5 TO B1+10:SET (X,Y):NEXT X
50 Y=Y+1:FOR X=B1+4 TO B1+11:SET (X,Y):NEXT X
60 Y=Y+1:FOR X=B1+3 TO B1+12:SET (X,Y):NEXT X
70 Y=Y+1:FOR X=B1+2 TO B1+13:SET (X,Y):NEXT X
80 Y=Y+1:FOR X=B1+1 TO B1+14:SET (X,Y):NEXT X
90 Y=Y+1:FOR X=B1 TO B1+15:SET (X,Y):NEXT X
100 RETURN
```

```
1 (Statement to clear screen)
4 A=32768:B1=0:B2=0
5 GOSUB 10
6 END
10 Y=B2+8
20 FOR X=B1+7 TO B1+8:POKE A+X+Y*40,102:NEXT X
30 Y=Y+1:FOR X=B1+6 TO B1+9:POKE A+X+Y*40,102:NEXT X
40 Y=Y+1:FOR X=B1+5 TO B1+10:POKE A+X+Y*40,102:NEXT X
50 Y=Y+1:FOR X=B1+4 TO B1+11:POKE A+X+Y*40,102:NEXT X
60 Y=Y+1:FOR X=B1+3 TO B1+12:POKE A+X+Y*40,102:NEXT X
70 Y=Y+1:FOR X=B1+2 TO B1+13:POKE A+X+Y*40,102:NEXT X
80 Y=Y+1:FOR X=B1+1 TO B1+14:POKE A+X+Y*40,102:NEXT X
90 Y=Y+1:FOR X=B1 TO B1+15:POKE A+X+Y*40,102:NEXT X
100 RETURN
```

procedure enables us to handle PET graphics in a manner similar to those used on the APPLE-II and TRS-80 systems where we use the PLOT or SET statements.

Note that Listing 1 operates by first setting the Y coordinate value. It then uses a FOR--NEXT loop to turn on all the desired points along the X axis for the current value of Y. You see, it does save quite a few individual PLOT directives over the method suggested for drawing the diagram in Figure 4A!

Make Your Triangle Multiply

The triangle produced by the subroutine exhibited in Listing 1 can be positioned just about anywhere you want it on the CRT screen. How? By merely initializing the value of the reference point variables BASEX and BASEY (or B1 and B2 in the PET and TRS-80 versions). Listing 2 shows a "calling sequence" that will cause the triangle to be drawn a number of times on the screen. (The triangles will overlap a bit on their sides. You can modify the program so that they are completely separated if you like. I just happened to like the pattern they made when they slightly overlap!) Listing 2 is specifically for an APPLE-II system. However, with just a few minor modifications (such as using the appropriate statement type to

Listing 2

```
  1 GR : COLOR=13
  3 FOR BASEX=0 TO 24 STEP 12
  4 FOR BASEY=0 TO 24 STEP 8
  6 GOSUB 10
  7 NEXT BASEY
  8 NEXT BASEX
  9 END
 10 Y=BASEY+8
 20 FOR X=BASEX+7 TO BASEX+8: PLOT
    X,Y: NEXT X
 30 Y=Y+1: FOR X=BASEX+6 TO BASEX+
    9: PLOT X,Y: NEXT X
 40 Y=Y+1: FOR X=BASEX+5 TO BASEX+
    10: PLOT X,Y: NEXT X
 50 Y=Y+1: FOR X=BASEX+4 TO BASEX+
    11: PLOT X,Y: NEXT X
 60 Y=Y+1: FOR X=BASEX+3 TO BASEX+
    12: PLOT X,Y: NEXT X
 70 Y=Y+1: FOR X=BASEX+2 TO BASEX+
    13: PLOT X,Y: NEXT X
 80 Y=Y+1: FOR X=BASEX+1 TO BASEX+
    14: PLOT X,Y: NEXT X
 90 Y=Y+1: FOR X=BASEX TO BASEX+
    15: PLOT X,Y: NEXT X
100 RETURN
```

clear the display) the same essential calling sequence can be used for other types of computers.

And What About Erasing?

Now the calling sequence in Listing 2 is nice if you want to draw a whole bunch of triangles and leave them on the screen. But suppose you just want to have the triangle change its position. That is, for it to disappear from one part of the screen and appear in another place. Well, in that case you had better "erase" the old triangle. Right?

That is simple enough to do if you only want to have the single triangle somewhere on the screen at any one time. Listing 3 illustrates a calling sequence that will do the job. The difference between it and Listing 2 is that it has a statement to clear the screen prior to drawing another triangle. (Again, the listing is specifically for an APPLE-II unit. You will need to make minor statement changes for other systems.)

Listing 3 will serve fine if all you need to display on the screen is the one item drawn by the subroutine. Suppose, however, that you will have other items on the screen at the same time that you desire to move the triangle about? Unless you plan to redraw the entire screen, you sure don't want to use a "clear screen" statement

Listing 3

```
  3 FOR BASEX=0 TO 24 STEP 12
  4 FOR BASEY=0 TO 24 STEP 8
  5 GR : COLOR=13
  6 GOSUB 10
  7 NEXT BASEY
  8 NEXT BASEX
  9 END
 10 Y=BASEY+8
 20 FOR X=BASEX+7 TO BASEX+8: PLOT
    X,Y: NEXT X
 30 Y=Y+1: FOR X=BASEX+6 TO BASEX+
    9: PLOT X,Y: NEXT X
 40 Y=Y+1: FOR X=BASEX+5 TO BASEX+
    10: PLOT X,Y: NEXT X
 50 Y=Y+1: FOR X=BASEX+4 TO BASEX+
    11: PLOT X,Y: NEXT X
 60 Y=Y+1: FOR X=BASEX+3 TO BASEX+
    12: PLOT X,Y: NEXT X
 70 Y=Y+1: FOR X=BASEX+2 TO BASEX+
    13: PLOT X,Y: NEXT X
 80 Y=Y+1: FOR X=BASEX+1 TO BASEX+
    14: PLOT X,Y: NEXT X
 90 Y=Y+1: FOR X=BASEX TO BASEX+
    15: PLOT X,Y: NEXT X
100 RETURN
```

to get rid of the old triangle. Nope, all you want to do is erase the old triangle. So, you execute a routine just like the one for drawing a triangle, only now you turn the display off at those points.

Such a procedure is a snap on the APPLE-II. Listing 4 shows that all one has to do is change the calling sequence so that it reexecutes the subroutine with COLOR=0, which effectively extinguishes the old triangle. (What else could you do with the APPLE-II? You could reexecute the subroutine with COLOR set to some other value, so that previous positions of the triangle are displayed in a color different than its current position!)

"... to get rid of the old triangle ... change the calling sequence so that it reexecutes the subroutine with COLOR=0 ..."

If one set up the drawing subroutine for a PET so that the portion of the POKE statement that designates the code to be inserted was a variable name, then a similar type of calling sequence would work there. (i.e., If a POKE statement in the subroutine appeared as POKE A+X+Y*40,Z, then a statement in the calling sequence could alter the variable Z between a displaying and nondisplaying code. For instance, setting Z to the value 32 would effectively blank out the triangle if the drawing subroutine was reexecuted.)

The situation would be a little more complicated with a TRS-80 or similar system. One would need to actually create a second subroutine. This would be identical to the one shown for the TRS-80 in Listing 1 except that the SET (X,Y) directives would be replaced with the RESET (X,Y) command. One would then have the calling sequence alternately call the two subroutines: one to draw the figure using the SET statements, the other to eliminate it through the use of the RESET statements.

Listing 4

```
  1 GR
  2 FOR BASEX=0 TO 24 STEP 12
  3 FOR BASEY=0 TO 24 STEP 8
  4 COLOR=13: GOSUB 10
  5 COLOR=0: GOSUB 10
  6 NEXT BASEY
  7 NEXT BASEX
  8 END
 10 Y=BASEY+8
 20 FOR X=BASEX+7 TO BASEX+8: PLOT
    X,Y: NEXT X
 30 Y=Y+1: FOR X=BASEX+6 TO BASEX+
    9: PLOT X,Y: NEXT X
 40 Y=Y+1: FOR X=BASEX+5 TO BASEX+
    10: PLOT X,Y: NEXT X
 50 Y=Y+1: FOR X=BASEX+4 TO BASEX+
    11: PLOT X,Y: NEXT X
 60 Y=Y+1: FOR X=BASEX+3 TO BASEX+
    12: PLOT X,Y: NEXT X
 70 Y=Y+1: FOR X=BASEX+2 TO BASEX+
    13: PLOT X,Y: NEXT X
 80 Y=Y+1: FOR X=BASEX+1 TO BASEX+
    14: PLOT X,Y: NEXT X
 90 Y=Y+1: FOR X=BASEX TO BASEX+
    15: PLOT X,Y: NEXT X
100 RETURN
```

Chapter 4

Drawing Lines

It's Time to Draw the Line

A few readers might wonder why I didn't discuss the drawing of a line before talking about something like triangles. After all, what could be simpler than drawing a line? Lots of things, it turns out! Drawing a line by computer, yes a plain old straight line, is not quite so simple as it might appear at first glance.

Oh yes, it is not difficult to draw a perfectly vertical or a perfectly horizontal line on a screen. In fact, you already know how to do that. We drew some straight horizontal lines when we drew the triangle! The procedure for creating a horizontal line is simply to set Y to the value on which the line is to reside, then invoke a statement such as

```
FOR X=0 TO 39 STEP 1: PLOT X,Y:NEXT X
or
FOR X=0 TO 39 STEP 1:POKE A+X+Y*40,Z:NEXT X
```

"... what could be simpler than drawing a line? Lots of things, it turns out!"

where A=32768 for a PET system and Z is the code for the graphic symbol to be displayed.

Similarly, to draw a vertical line you can set X to a fixed value and then vary Y over the desired range of the line that is to be drawn.

It turns out, however, that the cases of a perfectly vertical or perfectly horizontal line are somewhat unique. It is a little bit harder to draw a line using a computer when the end points are not on the same X or Y coordinate.

To take a look at the situation, why don't you load in the program shown in Listing 5 into your machine?

Please note that from here on out in this publication, listings will be shown for the APPLE-II system. I'll assume you will make

Listing 5

```
1  GR : COLOR= 13
4 X1 = 0:X2 = 39
5 Y1 = 0:Y2 = 39
6  GOSUB 10
9  END
10  FOR X = X1 TO X2
20 Y =  INT (((Y2 - Y1) / (X2 - X
     1)) * X)
30  PLOT X,Y
40  NEXT X
50  FOR Y = Y1 TO Y2
60 X =  INT (Y * (X2 - X1) / (Y2 -
     Y1))
70  PLOT X,Y
80  NEXT Y
90  RETURN
```

minor changes if necessary in order for these programs to run on your system. If you are running a Radio Shack TRS-80 (Level II), this generally means substituting the SET (X,Y) or RESET (X,Y) for PLOT X,Y directives and using the appropriate "clear the screen" directive to replace the GR (GRaphics) command used on the APPLE-II. If you have a Commodore PET unit, then you will want to substitute the now familiar POKE A+X+Y*40,Z directive in place of PLOT X,Y. A is equal to 32768 for a PET in the POKE formula and Z represents whatever graphics code you want displayed. The code 32 may be used if you want the display turned off at a point.

Once you have Listing 5 loaded, modify it slightly by inserting a statement line numbered 45 that reads as follows: 45 RETURN. This little change will enable you to see something of interest related to the current discussion. Figure 6 also applies to this discussion.

Suppose we wanted to have the computer draw a line on our display screen from position 0,0 to position 39,39. How would we go about giving it directions to do such a task?

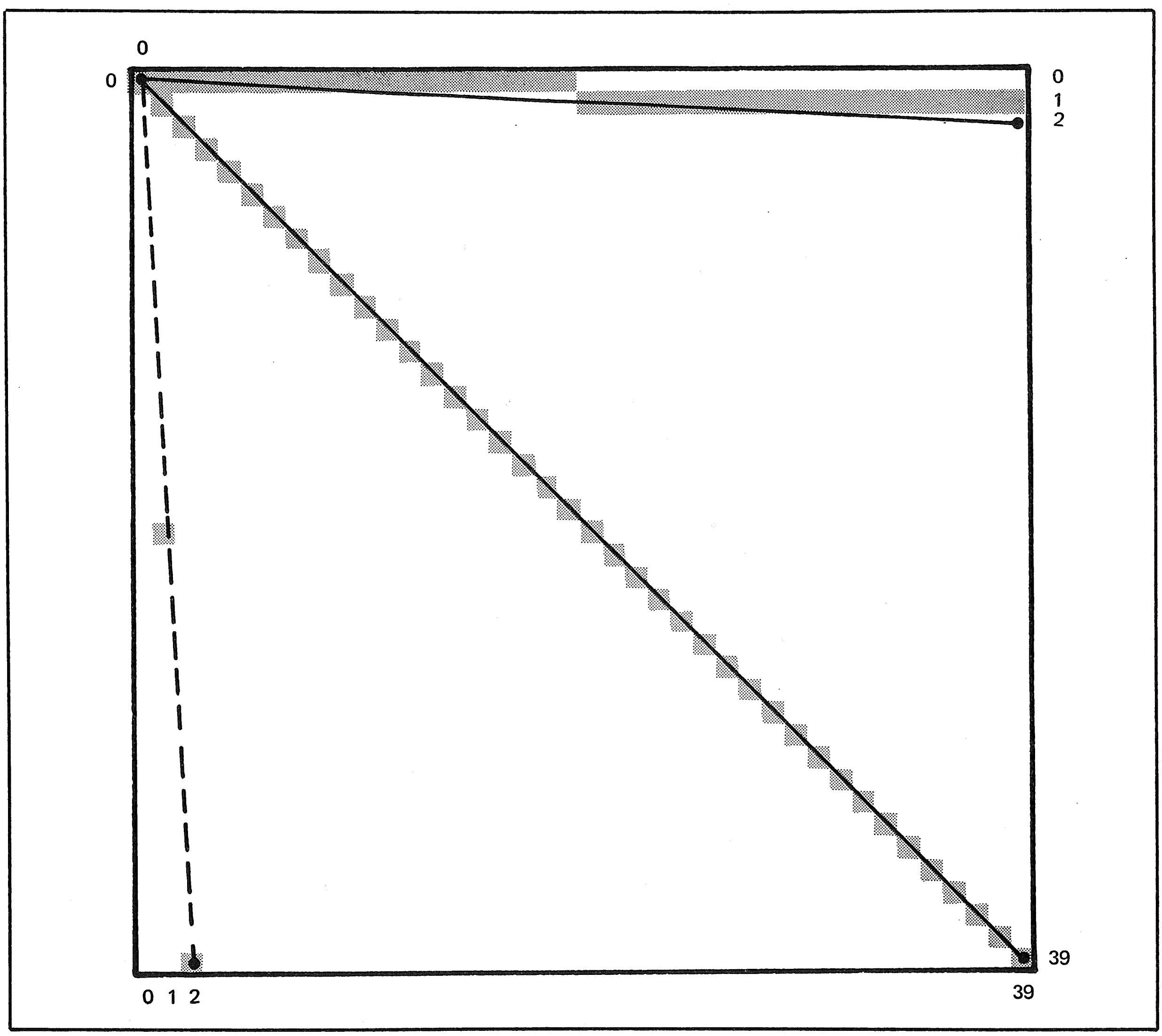
The first part of Listing 5 gives one possible way. It is based on an old high school algebra formula for the equation of a straight line in cartesian coordinates. Remember it?

$$Y = mX + b$$

The variable m in the formula stands for the slope of the line and b is the Y axis offset value. For the time being, we can forget about b as we shall initially restrict our discussion to lines that originate at 0,0. In such cases there is no Y axis offset.

Now the slope m is simply the change in units along the Y axis over the change in units along the X axis between two points on the line. What two points on the line? Why the starting and ending points of the line as far as we are concerned! So, if a line starts at X1,Y1 and ends at X2,Y2, then the slope can be equated to (Y2 – Y1)/(X2 – X1). Or, in other words, once the end points (or any two points, but I shall be using end points in my examples) have been defined, then points along the Y axis are those defined by multiplying the value of X at that location times (Y2 – Y1)/(X2 – X1). Line 20 in Listing 5 uses precisely that relationship to calculate values of Y along the line. Only integer values are used because we can only plot locations at integral points on the CRT screen.

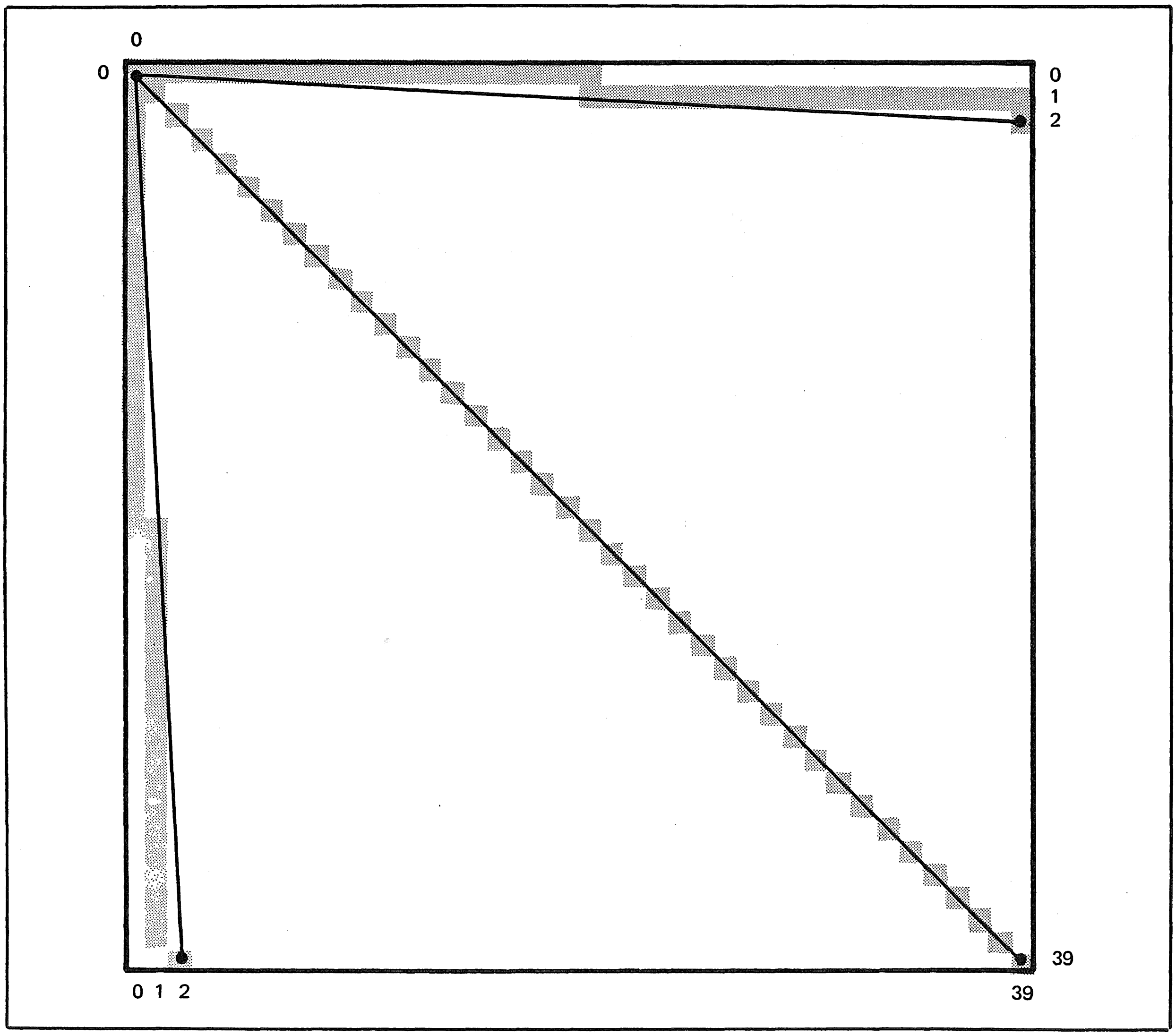
Figure 6



If you execute the program in Listing 5, with a RETURN statement inserted at line 45, it will draw what appears to be a nice diagonal line. That could lead you to think that the program works just fine. However, if you were to change line 4 of the calling sequence to X1=0:X2=2, you might be a little disappointed with the "line" drawn. As the dotted line in Figure 6 illustrates, you would only see a few points displayed along the line! That is hardly what you could call "drawing a line."

Restoring line 4 to its original value, X1=0:X2=39, and then changing line 5 to read Y1=0,Y2=2 would yield the nearly horizontal line shown in Figure 6. That line is not exactly perfect. For one thing, the end point of the line does not get displayed by the

Figure 7

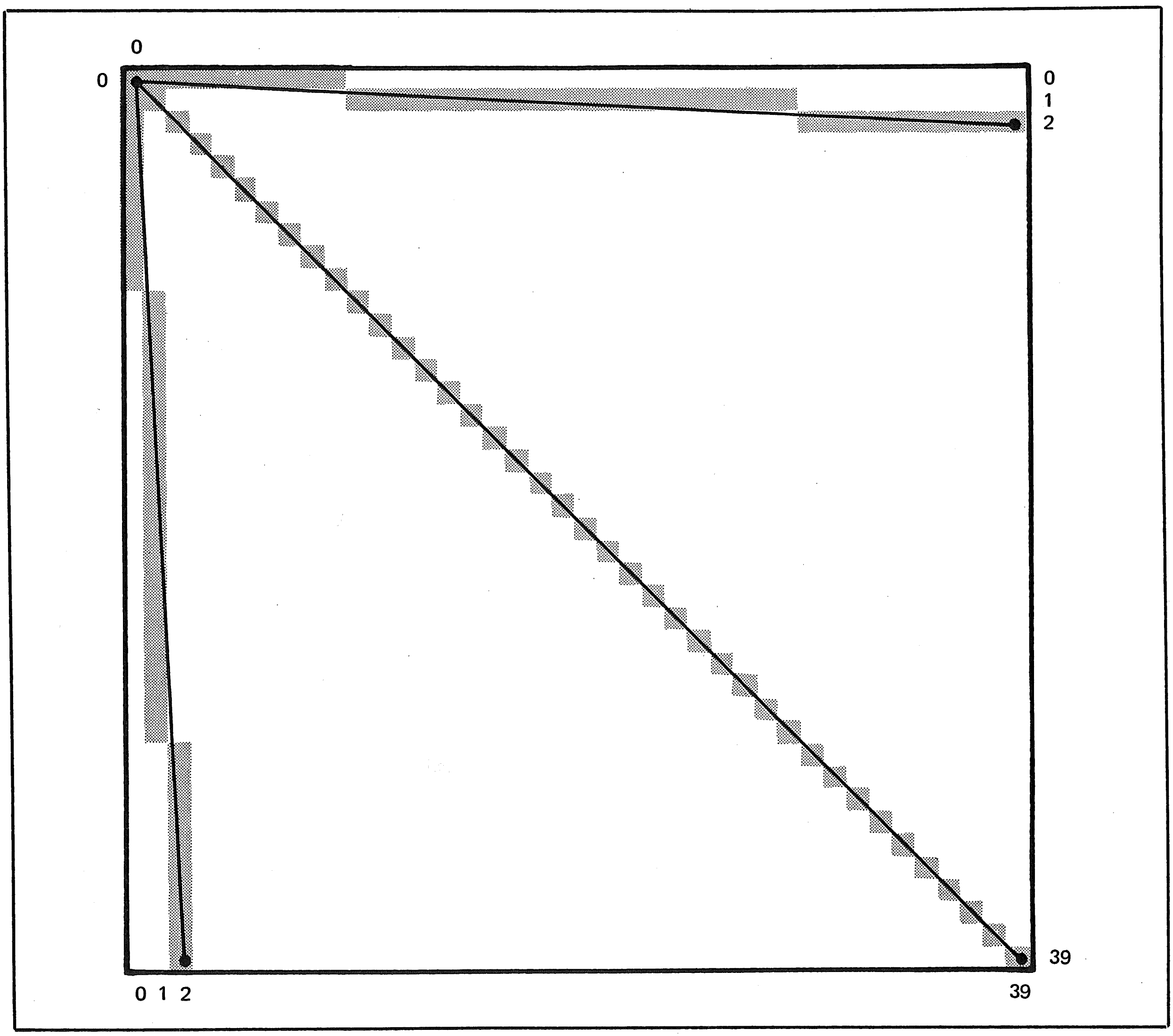


routine!

The reason we do not get a very good line drawn is because, with a RETURN statement at line 45, the program only calculates and plots points along the Y axis at discrete values of X. When X only goes from 0 to 2, you will only get a few points displayed, regardless of how far the line goes in the Y direction. We can improve the situation somewhat by removing the RETURN statement at line 45. Now the program will effectively fill in the gaps between points because it will also plot points along the X axis for discrete values of Y. Figure 7 illustrates the improvement one obtains when the entire program in Listing 5 is utilized.

Well, the lines in Figure 7 might be pretty good, considering

Figure 8



that they do show the end points of the line as well as a pretty rough approximation of the path that the line takes. However, to some people they may appear somewhat less than perfect. What seems to be the problem?

The problem is an anomaly of using digital computer techniques. A point along the line does not get plotted until a discrete value is reached. Thus, for the line that runs from X1=0 to X2=39 along the top of Figure 7, the line is plotted along Y=0 until Y reaches the value 1. It is plotted at Y=1 until Y=2, etc. Y reaches 2 just at the point that the line ends. This causes the line to appear somewhat lopsided or weighted towards the lower values of X.

A "smoother" line can be drawn by slightly modifying the program of Listing 5 so that it appears as shown in Listing 6. Compare lines 20 and 60 in those two listings. The simple technique of rounding off values to the next higher coordinate, by adding 0.5 to the product of the slope and the opposite axis' value, results in the improvement shown in Figure 8. Figure 8 is about the best you are going to be able to do when drawing straight lines with a low resolution display!

We aren't done with the matter of drawing straight lines yet! The program is Listing 6 is only for special cases of lines that start at the coordinate X=0,Y=0. It also will not handle the cases of a perfectly vertical or horizontal line. (Can you see why?)

What we really want is a general procedure for drawing a straight line starting and ending anywhere on a display. To do this, we need to add in the offset (b) part of our general line equation Y = mX +b. We also need to make a few tests so that our computer can handle the special cases when X or Y does not change value (thus

Listing 6

```
1  GR : COLOR= 13
4 X1 = 0:X2 = 2
5 Y1 = 0:Y2 = 39
6  GOSUB 10
9  END
10  FOR X = X1 TO X2
20 Y =  INT ((((Y2 - Y1) / (X2 -
    X1)) * X) + .5)
30  PLOT X,Y
40  NEXT X
50  FOR Y = Y1 TO Y2
60 X =  INT ((Y * (X2 - X1) / (Y2
    - Y1)) + .5)
70  PLOT X,Y
80  NEXT Y
90  RETURN
```

resulting in a delta value of zero in the divisor of the slope variable in the equation).

Listing 7 shows a general line-drawing algorithm that fills the bill. The line-drawing subroutine starting at line 5000 expects the starting and ending points of the line X1,Y1 and X2,Y2 to be set up before it is called.

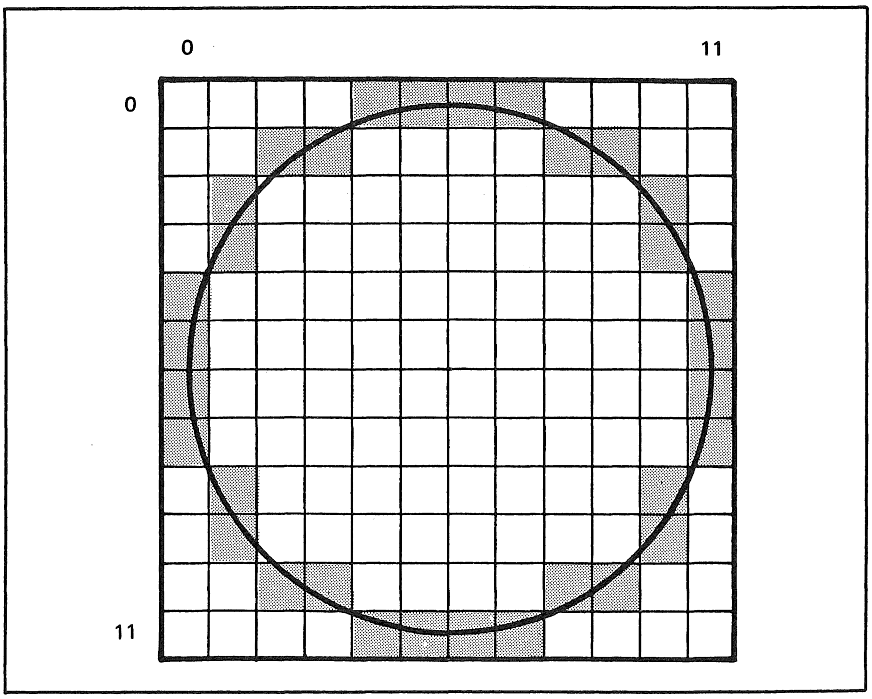
The calling sequence I have shown in Listing 7 will cause an APPLE-II system to draw lines of random length and direction with randomly varying colors. If you RUN it, your display screen will soon fill up with a continuously changing pattern. Systems that do not provide different colors can still be coaxed into interesting displays by alternately having the lines be drawn in white and black. This is easy to do with a PET by changing the POKE character each time the line drawing subroutine is called. With a TRS-80 you would need to create another line drawing subroutine that utilized the RESET (X,Y) statement. In any event, you can see how BASIC's RND (random) function can be used in connection with the line drawing subroutine to create random patterns.

Listing 7

```
1  GR : COLOR= 13
2 X1 =  INT ( RND (1) * 38):X2 =
      INT ( RND (1) * 38): IF X1 =
     X2 THEN 2
3 X1 =  INT ( RND (1) * 38):X2 =
      INT ( RND (1) * 38)
5 Y1 =  INT ( RND (1) * 38):Y2 =
      INT ( RND (1) * 38)
6  GOSUB 5000
7  COLOR=  RND (1) * 14 + 1
8  GOTO 2
9  END
5000  IF X2 > X1 THEN A = 1
5010  IF X2 < X1 THEN A =  - 1
5020  IF X2 = X1 THEN 5070
5030  FOR X = X1 TO X2 STEP A
5040 Y =  INT ((((Y2 - Y1) / (X2 -
     X1)) * (X - X1)) + .5) + Y1
5050  PLOT X,Y
5060  NEXT X
5070  IF Y2 > Y1 THEN B = 1
5080  IF Y2 < Y1 THEN B =  - 1
5090  IF Y2 = Y1 THEN 5140
5100  FOR Y = Y1 TO Y2 STEP B
5110 X =  INT (((Y - Y1) * (X2 -
     X1) / (Y2 - Y1)) + .5) + X1
5120  PLOT X,Y
5130  NEXT Y
5140  RETURN
```

Listing 8
```
200  GR : COLOR= 13
210 X = 5:Y = 5
220  GOSUB 6000
240  END
6000  PLOT X + 4,Y: PLOT X + 5,Y:
      PLOT X + 6,Y: PLOT X + 7,Y
6010  PLOT X + 2,Y + 1: PLOT X +
     3,Y + 1: PLOT X + 8,Y + 1: PLOT
     X + 9,Y + 1
6020  PLOT X + 1,Y + 2: PLOT X +
     10,Y + 2
6030  PLOT X + 1,Y + 3: PLOT X +
     10,Y + 3
6040  PLOT X,Y + 4: PLOT X + 11,Y
      + 4
6050  PLOT X,Y + 5: PLOT X + 11,Y
      + 5
6060  PLOT X,Y + 6: PLOT X + 11,Y
      + 6
6070  PLOT X,Y + 7: PLOT X + 11,Y
      + 7
6080  PLOT X + 1,Y + 8: PLOT X +
     10,Y + 8
6090  PLOT X + 1,Y + 9: PLOT X +
     10,Y + 9
6100  PLOT X + 2,Y + 10: PLOT X +
     3,Y + 10: PLOT X + 8,Y + 10:
      PLOT X + 9,Y + 10
6110  PLOT X + 4,Y + 11: PLOT X +
     5,Y + 11: PLOT X + 6,Y + 11:
      PLOT X + 7,Y + 11: RETURN
```

Figure 9



Avoid Going around in Circles

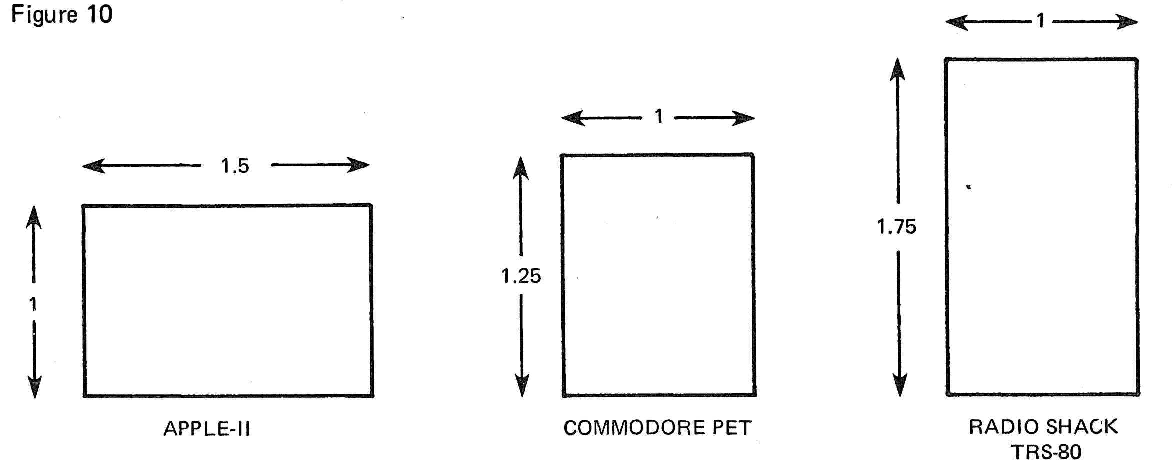
Just as drawing lines on a display is not as easy as one might initially think for the general case; drawing of circles can become quite complex for the general case. For that reason and another factor that I will discuss shortly, I recommend that you minimize your use of circular diagrams. Furthermore, when you find you really want to use a circle, I suggest you draw it using a point-by-point specification method such as that used to draw triangles that I presented earlier.

Figure 9 shows the points that could be illuminated to draw a circle. Listing 8 illustrates the method for drawing the circle. Notice that a subroutine is used to actually draw the diagram. The calling sequence allows the user to specify initial values for X and Y so that the circle may be positioned wherever desired on the screen.

Alas, if you were to load the program of Listing 8 into your computer system and try displaying the circle it draws, you might be a bit disappointed. The circle would quite likely look more like an ellipse than a perfectly round ring. Why? Because, unfortunately, most CRT displays today do not plot or illuminate a sector that is actually a square. The sectors are generally rectangular. To make matters worse, there does not appear to be any standardization amongst manufacturers.

For instance, Figure 10 illustrates the typical shapes of a low resolution sector for several popular systems. The numbers along the sides of each rectangle give the approximate ratio of the sides in the vertical and horizontal directions. You can promptly see that an APPLE-II's display illuminates a sector that is somewhat longer horizontally than vertically. On the other hand, a TRS-80's display

Figure 10



has sectors that are just the opposite. They are taller vertically than horizontally! The PET unit is somewhat like the Radio Shack TRS-80 except that the ratio is not as pronounced. For drawing circles, the PET unit has about the best symmetry, but it too is not perfectly balanced.

What to do if you really want a circle to look like a circle rather than an ellipse? You will need to use special graph paper that accurately represents the ratios of the display sectors on your particular system. Draw a circle on that graph, and then create the exact statements needed to draw the round circle on your display.

Chapter 5

A Graphics Library

Build Up Your Own Graphics Library

The whole key to really effectively applying graphics on your own personal machine is the building up of a "library" of routines that you can call on as desired. This library must be carefully organized so that it consists of subroutines that operate in such a manner that they can be positioned wherever desired on the screen. That is, they should start from a set reference point, such as the upper left-hand sector (0,0). They then may be positioned by having the calling sequence set up the appropriate offset values. This procedure has been introduced in this publication starting off with the triangles and was continued with the circle. This principle will be continued so you can at least start your library with some of the items presented here.

Remember, the key idea is to structure your subroutines so that they are able to be offset by a *base address*. When you first design an item, the base address can be zero. Later, when you want to position the drawing at some particular point on the screen, you have the calling sequence set the base value to the desired starting point of the drawing.

Creating a Library Means Drawing Pictures

You become the boss when it comes to building up your graphics library. You also become the artist! The strategy and the fundamental technique is simple.

You get some graphing paper.

Now be careful! I generally like to work with the "engineering" type of graphing paper you can buy from drafting paper supply houses. I usually work with the type that has 1/4-inch or 1/8-inch grids. But you can buy the paper with grids ranging from one inch down to 1/20 inch. By "be careful," I mean to keep in mind the fact that this kind of graphing paper has square grids. The individual sectors that are illuminated on your screen, as previously pointed out, are not likely to be exactly square. If you plot a square on your graph paper, you are going to get a rectangle if you illuminate the corresponding positions on your screen. If you plan a circle, you will get an ellipse.

I have found I can work pretty effectively using the regular engineering-type graphing paper. I just keep in mind the kind of distortion likely to occur on the screen and make adjustments if necessary.

However, if you are a perfectionist, or are going to get involved in fancy drawings or critical representations, you may want to construct your pictures on special graph paper. You can make your own by drawing grids that have the same horizontal to vertical ratios as that used on the display screen your system uses. If you have access to a duplicating or mimeographing machine, you can make up one master and then run off a bundle. You can do the same thing if there is an offset printer in your neighborhood. In fact many of the offset printing firms can actually make up pads of 50 or so sheets of your own personally created grids.

In any event, get a hold of some graphing paper that suits you. Then, lightly sketch the outline of the object you wish to represent on the screen, going along grid lines wherever possible. Then fill in the portion(s) to be represented as you see fit. In some cases you will just want to outline the object, such as was done for the circle illustrated in Figure 9. In other cases, such as the triangles shown earlier, you may want to fill in the entire object.

When the outline of your sketched object goes at an angle to the grid, you will have to make a judgment about illuminating a sector. A good rule of thumb is that if more than half the sector is "inside" the line, then illuminate it. However, sometimes you will have to use "artistic judgment." This is particularly true when you are drawing curves or dealing with angular and irregularly shaped objects, etc.

Don't be afraid to experiment and try different arrangements. If something doesn't seem to come out right, try some of the following alternatives:

1) Try just bordering the object instead of illuminating it solidly or vice versa.

2) Try reversing the background. That is, surround the drawing with illuminated points so that the object or its outline is portrayed by sectors that are not illuminated.

3) Try changing the number of sectors along one or both dimensions. Especially try going from an odd to an even value or vice

versa.

4) Reposition the item you are trying to represent on the grid or show it in a new perspective.

5) If you have color capability (such as on the APPLE-II) or special graphics symbols (such as on the PET), by all means try to capitalize on that capability. Alter the colors to enhance lines or change the graphics symbols used to accent a line or portion of a drawing.

Once you have your drawing represented on graphing paper, you are ready to construct your "general purpose" subroutine. By general purpose, I mean a subroutine constructed in such a way that it can be called upon, by setting up parameters, to draw the item at different locations on the screen. The easiest technique to use is the one illustrated when presenting the triangle and circle in this manual. You simply have the subroutine construct the drawing as though it was positioned initially at 0,0 (as the starting point). Then use variables that can be offset by the subroutine calling sequence. In Listing 8 variables X and Y are set by the calling sequence so that the circle can be drawn anywhere on the screen after initial values of X and Y are defined. In Listing 1, the triangle is offset along the X axis by the variable named BASEX and the Y axis offset is determined by variable BASEY, if desired, or simply by the variable Y.

"When a subroutine has been prepared in this manner, it can be used again and again . . ."

When a subroutine has been prepared in this manner, it can be used again and again in the same or different programs. The item represented by the subroutine can be placed wherever desired on the screen. As will be observed later, this technique also permits a programmer to animate pictures by rapidly changing the positions of drawings on the screen. This concept was introduced by the program of Listing 4 that causes a triangle to move about the screen.

Let's Stack Your Library with a Deck of Cards

Listing 9 shows a large group of subroutines (starting at line 9000) that may be placed in your library. They may be used to draw pictures of playing cards – from the Ace of Hearts to the Deuce of Spades. The first part of the listing illustrates just one way that the subroutines may be called. Lines 10 through 220 in the program will repeatedly deal two cards at random from a deck and cause them to be drawn on the display screen. Lines 20 and 40 set values of X and Y to position the starting point for the subroutine at line 9000 that draws an outline of a playing card.

Figure 11 illustrates how a playing card is built up. This is done by selecting any one of a group of smaller "picture blocks" and placing it in the proper position within the outline or border of a card

Listing 9

```
10  GR : COLOR= 13: REM   GRAPHIC
     S/COLOR STATEMENT FOR COLOR
     SYSTEMS
20 X = 2:Y = 7
30  GOSUB 9000
40 X = 22:Y = 7
50  GOSUB 9000
60 X = 4:Y = 15
70 C1 =  INT ( RND (1) * 52 + 1)
80 C2 =  INT ( RND (1) * 52 + 1)
90  IF C2 = C1 THEN 80
100 S =  INT (C1 / 13) + 1
110  GOSUB 2000
120 X = 4:Y = 9
130 N =  INT (C1 / 4) + 1
140  GOSUB 2100
150 X = 24:Y = 15
160 S =  INT (C2 / 13) + 1
170  GOSUB 2000
180 X = 24:Y = 9
190 N =  INT (C2 / 4) + 1
200  GOSUB 2100
210  FOR K = 1 TO 2000: NEXT K
220  GOTO 60
2000  IF S = 1 THEN  GOSUB 9200
2010  IF S = 2 THEN  GOSUB 9250
2020  IF S = 3 THEN  GOSUB 9300
2030  IF S = 4 THEN  GOSUB 9350
2040  RETURN
2100  IF N = 1 THEN  GOSUB 9500
2110  IF N = 2 THEN  GOSUB 9530
2120  IF N = 3 THEN  GOSUB 9560
2130  IF N = 4 THEN  GOSUB 9590
2140  IF N = 5 THEN  GOSUB 9620
2150  IF N = 6 THEN  GOSUB 9650
2160  IF N = 7 THEN  GOSUB 9680
2170  IF N = 8 THEN  GOSUB 9710
2180  IF N = 9 THEN  GOSUB 9740
2190  IF N = 10 THEN  GOSUB 9770
2200  IF N = 11 THEN  GOSUB 9800
2210  IF N = 12 THEN  GOSUB 9830
2220  IF N = 13 THEN  GOSUB 9860
2230  RETURN
9000  FOR I = 0 TO 15 STEP 1
9010 W = X + I: PLOT W,Y: PLOT W,
     Y + 25
9020  NEXT I
9030  FOR I = 1 TO 24 STEP 1
9040 Z = Y + I: PLOT X,Z: PLOT X +
     15,Z
9050  NEXT I
9060  RETURN
```

```
9200  REM   **HEART**  CALL SUIT
     CLEANUP
9210  GOSUB 9950
9220  COLOR= 11: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9222  PLOT X + 3,Y: PLOT X + 8,Y
9224  PLOT X + 2,Y + 1: PLOT X +
     3,Y + 1: PLOT X + 4,Y + 1: PLOT
     X + 7,Y + 1: PLOT X + 8,Y +
     1: PLOT X + 9,Y + 1
9226  FOR I = 1 TO 10 STEP 1:W =
     X + I: PLOT W,Y + 2: NEXT I
9228  FOR I = 1 TO 10 STEP 1:W =
     X + I: PLOT W,Y + 3: NEXT I
9230  FOR I = 1 TO 10 STEP 1:W =
     X + I: PLOT W,Y + 4: NEXT I
9232  FOR I = 2 TO 9 STEP 1:W = X
      + I: PLOT W,Y + 5: NEXT I
9234  FOR I = 2 TO 9 STEP 1:W = X
      + I: PLOT W,Y + 6: NEXT I
9236  FOR I = 3 TO 8 STEP 1:W = X
      + I: PLOT W,Y + 7: NEXT I
9238  FOR I = 4 TO 7 STEP 1:W = X
      + I: PLOT W,Y + 8: NEXT I
9240  PLOT X + 5,Y + 9: PLOT X +
     6,Y + 9
9245  RETURN
9250  REM   **DIAMOND**  CALL SUI
     T CLEANUP
9260  GOSUB 9950
9270  COLOR= 11: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9272  PLOT X + 5,Y
9274  PLOT X + 4,Y + 1: PLOT X +
     5,Y + 1: PLOT X + 6,Y + 1
9276  FOR I = 3 TO 7:W = X + I: PLOT
     W,Y + 2: NEXT I
9278  FOR I = 2 TO 8:W = X + I: PLOT
     W,Y + 3: NEXT I
9280  FOR I = 1 TO 9:W = X + I: PLOT
     W,Y + 4: NEXT I
9282  FOR I = 2 TO 8:W = X + I: PLOT
     W,Y + 5: NEXT I
9284  FOR I = 3 TO 7:W = X + I: PLOT
     W,Y + 6: NEXT I
9286  PLOT X + 4,Y + 7: PLOT X +
     5,Y + 7: PLOT X + 6,Y + 7
9288  PLOT X + 5,Y + 8
9290  RETURN
9300  REM   **CLUB**  CALL SUIT C
     LEANUP
9310  GOSUB 9950
```

```
9320  COLOR= 6: REM   COLOR STATE
     MENT FOR COLOR SYSTEMS
9322  FOR I = 4 TO 7:W = X + I: PLOT
     W,Y: NEXT I
9324  FOR I = 3 TO 8:W = X + I: PLOT
     W,Y + 1: NEXT I
9326  FOR I = 3 TO 8:W = X + I: PLOT
     W,Y + 2: NEXT I
9328  PLOT X + 1,Y + 3: PLOT X +
     2,Y + 3: FOR I = 4 TO 7:W =
     X + I: PLOT W,Y + 3: NEXT I:
      PLOT X + 9,Y + 3: PLOT X +
     10,Y + 3
9330  FOR I = 0 TO 11:W = X + I: PLOT
     W,Y + 4: NEXT I
9332  FOR I = 0 TO 11:W = X + I: PLOT
     W,Y + 5: NEXT I
9334  FOR I = 0 TO 11:W = X + I: PLOT
     W,Y + 6: NEXT I
9336  FOR I = 0 TO 11:W = X + I: PLOT
     W,Y + 7: NEXT I
9338  PLOT X + 1,Y + 8: PLOT X +
     2,Y + 8: PLOT X + 5,Y + 8: PLOT
     X + 6,Y + 8: PLOT X + 9,Y +
     8: PLOT X + 10,Y + 8
9340  PLOT X + 5,Y + 9: PLOT X +
     6,Y + 9
9345  RETURN
9350  REM   **SPADE**  CALL SUIT
     CLEANUP
9360  GOSUB 9950
9370  COLOR= 6: REM   COLOR STATE
     MENT FOR COLOR SYSTEMS
9372  PLOT X + 5,Y: PLOT X + 6,Y
9374  FOR I = 4 TO 7:W = X + I: PLOT
     W,Y + 1: NEXT I
9376  FOR I = 3 TO 8:W = X + I: PLOT
     W,Y + 2: NEXT I
9378  FOR I = 2 TO 9:W = X + I: PLOT
     W,Y + 3: NEXT I
9380  FOR I = 1 TO 10:W = X + I: PLOT
     W,Y + 4: NEXT I
9382  FOR I = 1 TO 10:W = X + I: PLOT
     W,Y + 5: NEXT I
9384  FOR I = 1 TO 10:W = X + I: PLOT
     W,Y + 6: NEXT I
9386  PLOT X + 2,Y + 7: PLOT X +
     3,Y + 7: PLOT X + 5,Y + 7: PLOT
     X + 6,Y + 7: PLOT X + 8,Y +
     7: PLOT X + 9,Y + 7
9388  PLOT X + 5,Y + 8: PLOT X +
     6,Y + 8
```

```
9390  PLOT X + 5,Y + 9: PLOT X +
     6,Y + 9
9395  RETURN
9500  REM   **ACE**  CALL CLEANUP
      ROUTINE
9505  GOSUB 9980
9510  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9512  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y: NEXT I
9514  PLOT X,Y + 1: PLOT X + 4,Y +
     1
9515  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9516  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 2: NEXT I
9518  PLOT X,Y + 3: PLOT X + 4,Y +
     3
9520  PLOT X,Y + 4: PLOT X + 4,Y +
     4
9525  RETURN
9530  REM   **TWO**  CALL CLEANUP
      ROUTINE
9535  GOSUB 9980
9540  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9542  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y: NEXT I
9544  PLOT X + 4,Y + 1
9546  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 2: NEXT I
9548  PLOT X,Y + 3
9550  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 4: NEXT I
9555  RETURN
9560  REM   **THREE**  CALL CLEAN
     UP ROUTINE
9565  GOSUB 9980
9570  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9572  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y: NEXT I
9574  PLOT X + 4,Y + 1
9576  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 2: NEXT I
9578  PLOT X + 4,Y + 3
9580  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 4: NEXT I
9585  RETURN
9590  REM   **FOUR**  CALL CLEANU
     P ROUTINE
9595  GOSUB 9980
```

```
9600  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9602  PLOT X,Y: PLOT X + 3,Y
9604  PLOT X,Y + 1: PLOT X + 3,Y +
     1
9606  PLOT X,Y + 2: PLOT X + 3,Y +
     2
9608  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 3: NEXT I
9610  PLOT X + 3,Y + 4
9615  RETURN
9620  REM   **FIVE**  CALL CLEANU
     P ROUTINE
9625  GOSUB 9980
9630  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9632  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y: NEXT I
9634  PLOT X,Y + 1
9636  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 2: NEXT I
9638  PLOT X + 4,Y + 3
9640  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 4: NEXT I
9645  RETURN
9650  REM   **SIX**  CALL CLEANUP
      ROUTINE
9655  GOSUB 9980
9660  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9662  PLOT X,Y
9664  PLOT X,Y + 1
9666  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 2: NEXT I
9668  PLOT X,Y + 3: PLOT X + 4,Y +
     3
9670  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 4: NEXT I
9675  RETURN
9680  REM   **SEVEN**  CALL CLEAN
     UP ROUTINE
9685  GOSUB 9980
9690  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9692  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y: NEXT I
9694  PLOT X + 3,Y + 1
9696  PLOT X + 2,Y + 2
9698  PLOT X + 1,Y + 3
9700  PLOT X,Y + 4
9705  RETURN
9710  REM   **EIGHT**  CALL CLEAN
```

```
     UP ROUTINE
9715  GOSUB 9980
9720  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9722  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y: NEXT I
9724  PLOT X,Y + 1: PLOT X + 4,Y +
     1
9726  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 2: NEXT I
9728  PLOT X,Y + 3: PLOT X + 4,Y +
     3
9730  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 4: NEXT I
9735  RETURN
9740  REM   **NINE**  CALL CLEANU
     P ROUTINE
9745  GOSUB 9980
9750  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9752  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y: NEXT I
9754  PLOT X,Y + 1: PLOT X + 4,Y +
     1
9756  FOR I = 0 TO 4:W = X + I: PLOT
     W,Y + 2: NEXT I
9758  PLOT X + 4,Y + 3
9760  PLOT X + 4,Y + 4
9765  RETURN
9770  REM   **TEN**  CALL CLEANUP
      ROUTINE
9775  GOSUB 9980
9780  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
9782  PLOT X,Y: PLOT X + 2,Y: PLOT
     X + 3,Y: PLOT X + 4,Y
9784  PLOT X,Y + 1: PLOT X + 2,Y +
     1: PLOT X + 4,Y + 1
9786  PLOT X,Y + 2: PLOT X + 2,Y +
     2: PLOT X + 4,Y + 2
9788  PLOT X,Y + 3: PLOT X + 2,Y +
     3: PLOT X + 4,Y + 3
9790  PLOT X,Y + 4: PLOT X + 2,Y +
     4: PLOT X + 3,Y + 4: PLOT X +
     4,Y + 4
9795  RETURN
9800  REM   **JACK**  CALL CLEANU
     P ROUTINE
9805  GOSUB 9980
9810  COLOR= 15: REM   COLOR STAT
     EMENT FOR COLOR SYSTEMS
```

```
9812  FOR I = 0 TO 4:W = X + I: PLOT
    W,Y: NEXT I
9814  PLOT X + 2,Y + 1
9816  PLOT X + 2,Y + 2
9818  PLOT X,Y + 3: PLOT X + 2,Y +
    3
9820  PLOT X,Y + 4: PLOT X + 1,Y +
    4: PLOT X + 2,Y + 4
9825  RETURN
9830  REM   **QUEEN**  CALL CLEAN
    UP ROUTINE
9835  GOSUB 9980
9840  COLOR= 15: REM   COLOR STAT
    EMENT FOR COLOR SYSTEMS
9842  FOR I = 0 TO 4:W = X + I: PLOT
    W,Y: NEXT I
9844  PLOT X,Y + 1: PLOT X + 4,Y +
    1
9846  PLOT X,Y + 2: PLOT X + 2,Y +
    2: PLOT X + 4,Y + 2
9848  PLOT X,Y + 3: PLOT X + 3,Y +
    3: PLOT X + 4,Y + 3
9850  FOR I = 0 TO 4:W = X + I: PLOT
    W,Y + 4: NEXT I
9855  RETURN
9860  REM   **KING**  CALL CLEANU
    P ROUTINE
9865  GOSUB 9980
9870  COLOR= 15: REM   COLOR STAT
    EMENT FOR COLOR SYSTEMS
9872  PLOT X,Y: PLOT X + 4,Y
9874  PLOT X,Y + 1: PLOT X + 3,Y +
    1
9876  PLOT X,Y + 2: PLOT X + 1,Y +
    2: PLOT X + 2,Y + 2
9878  PLOT X,Y + 3: PLOT X + 3,Y +
    3
9880  PLOT X,Y + 4: PLOT X + 4,Y +
    4
9885  RETURN
9950  COLOR= 0: REM   COLOR STATE
    MENT FOR COLOR SYSTEMS
9952  FOR I = 0 TO 9 STEP 1
9954  FOR J = 0 TO 11 STEP 1
9956  W = X + J:Z = Y + I: PLOT W,
    Z
9958  NEXT J
9960  NEXT I
9962  RETURN
9980  COLOR= 0: REM   COLOR STATE
    MENT FOR COLOR SYSTEMS
9982  FOR I = 0 TO 4 STEP 1
9984  FOR J = 0 TO 4 STEP 1
```

```
9986 W = X + J:Z = Y + I: PLOT W,
     Z
9988  NEXT J
9990  NEXT I
9992  RETURN
```

that has been drawn on the screen.

Each of the numerals, as well as the symbols for the Jack, Queen, King and Ace are shown at the top of Figure 11. A subroutine that creates each of these symbols is provided as part of the "card library." These subroutines start at line 9500. They are spaced 30 line numbers apart in Listing 9. Thus, the symbol for an ace can be drawn by calling the subroutine at line 9500; the symbol for a deuce, by calling line 9530; the symbol for a three, by calling line 9560, and so forth. Note that each of these subroutines starts by calling upon another subroutine that will clear the area in which the symbol is to be drawn. This is done so that a symbol is always drawn, so to speak, on a clean slate. Otherwise, we could get mixed up images if, for instance, the numeral seven was drawn on top of the digit six.

"... before a suit-drawing subroutine is called upon, the program must set initial values of X and Y ..."

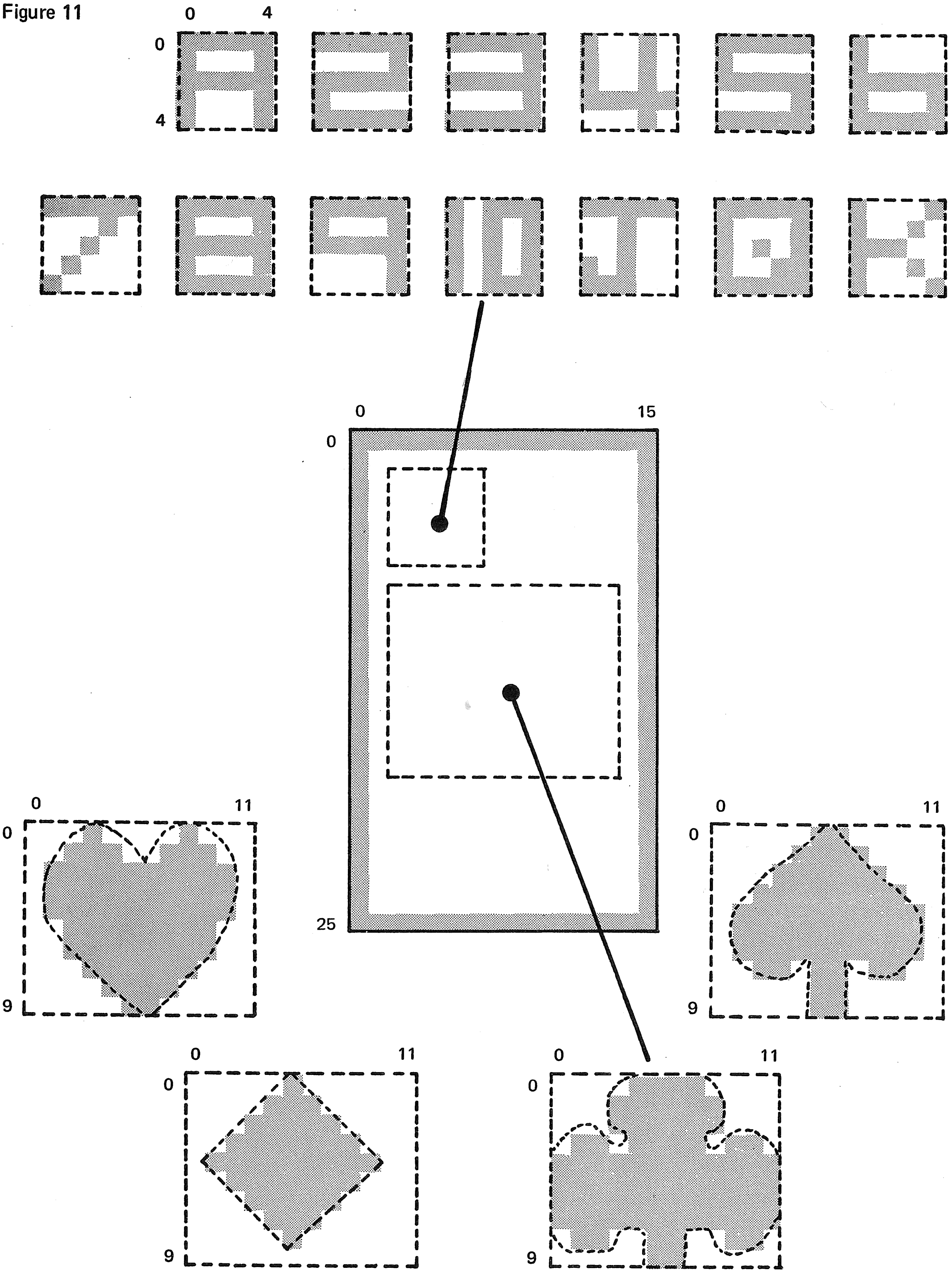
Now, before any of the card rank symbols are drawn by calling the desired subroutine, the values of X and Y must be defined as a starting reference point. The reference point for the subroutines is always the top left point of the block shown in the illustration. Lines 120 and 180 in Listing 9 are used to initialize the starting points for the card rank symbols. (Remember, there are two cards being drawn on the screen. Thus, there must be two reference points provided for the subroutine.)

The larger picture block shown in the illustration, in the center of the card, is used to draw the symbol for a card's suit. There are four standard card suits: hearts, diamonds, clubs and spades. Subroutines in Listing 9, starting at line number 9200, are capable of drawing the corresponding symbol for each of these suits. Note again, that each of these subroutines first calls on another subroutine to clear the area in which the suit symbol is to be drawn.

As is always the case, before a suit-drawing subroutine is called upon, the program must set initial values of X and Y to tell the program where to start drawing the illustration on the display screen. This is done by lines 60 and 150 in the example program.

The subroutine calling sequence in lines 10 through 220 is provided purely as an example. Normally, you would use the card drawing subroutines in connection with some type of card game that you were having the computer play, such as blackjack.

Figure 11



It will also be pointed out here that the subroutines shown in the example listing were created in a manner to maximize clarity of presentation. Each block is defined by specifying each sector that should be turned on (illuminated) all the way across each row. This was done on a row-by-row basis. Naturally, this method is very wasteful of memory. All of the subroutines could be considerably compressed by creating other subroutines that performed repetitive functions. By all means do such compression if memory is at a premium in your system. It is also possible to compress some of the subroutines by using different methods of specifying the sectors to be illuminated, such as capitalizing on nested subroutines.

TRS-80 users will probably want to double the number of horizontal points used to define a symbol. (By the way, if you want to sound like a real pro, you can refer to an individual low resolution point or sector as a *pixel*. For some reason the nomenclature reminds me of knitting so I rarely use it. However, some people who are really into graphics use it all the time, so remember it, at least you will know what they are talking about!) Don't forget also to substitute the SET statement in place of the PLOT statement shown in the listing for a TRS-80. You also won't need the COLOR statements shown in the listing on a TRS-80.

"... you can refer to an individual low resolution point or sector as a pixel."

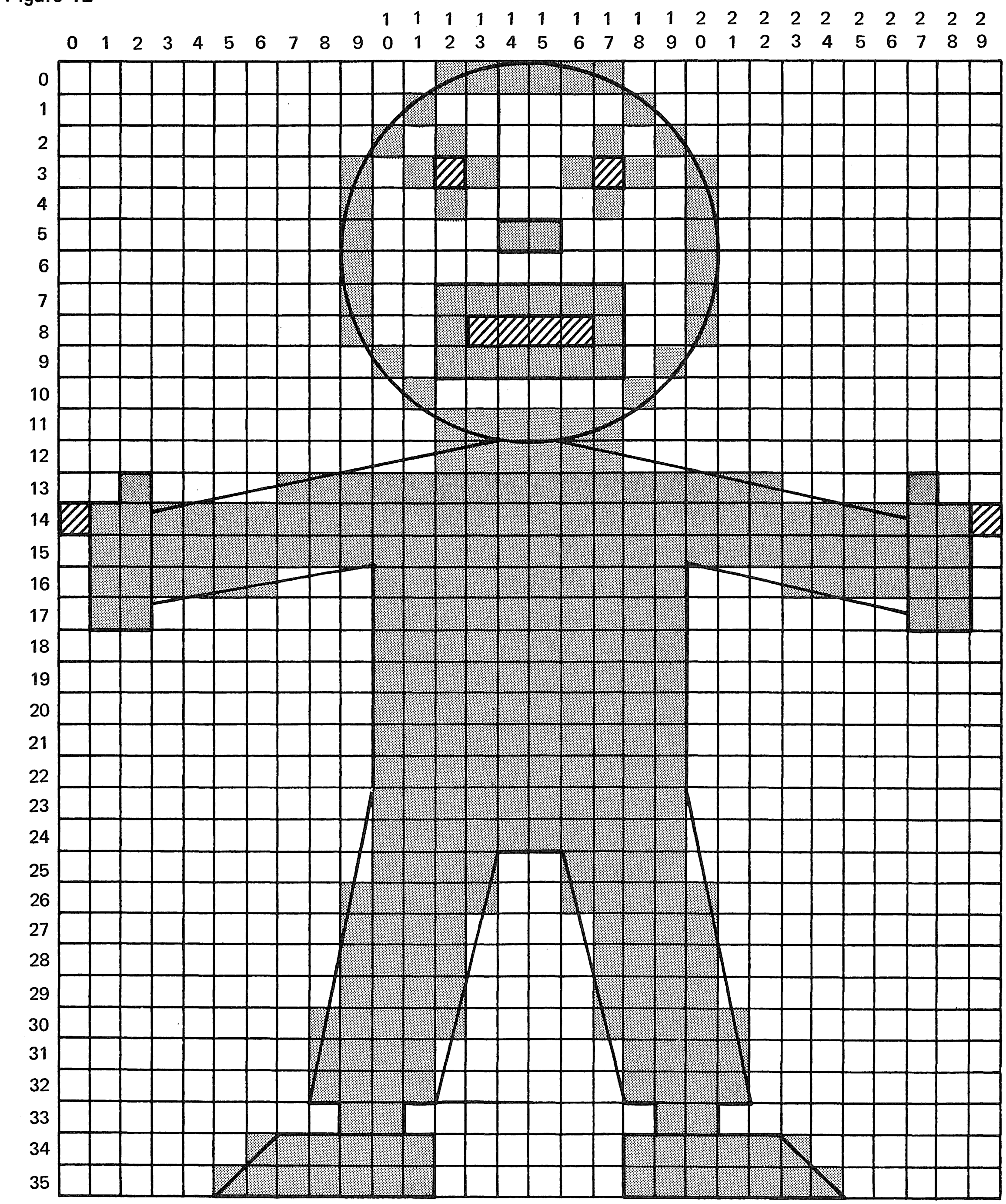
If you want to use the card drawing subroutines on a PET, things will be just a little more complicated as you now know. A statement shown in Listing 9 as PLOT X+1,Y+2 would translate to POKE A+(X+1)+(Y+2)*40,Z for the PET, where A is the starting address of the display buffer (32768) and Z is the code for the graphic symbol to be displayed. It is a bit more work to keep this formula straight as you convert the subroutines of Listing 9 for use on a PET, but it will go pretty smoothly once you get the knack of it. It is a good idea to check out each subroutine as it is entered to catch any boo-boos right away. Just initialize X and Y to some suitable values (such as 0,0 or 15,15 if you want things more towards the center of the screen) and call the subroutine you want to test. Use Figure 11 as a comparison to see that you get each card rank and suit symbol displayed correctly.

Once you have them checked out, be sure and save a couple of copies of the subroutines on your mass storage device. Then you will be all set to conjure up your own card playing games with genuine card drawing graphics. It makes a big difference in the impression your computer will make on friends when the card in play is actually drawn and displayed by the computer, rather than just having a printed message to the effect of "you drew the Ace of Clubs."

Are You Ready for Some Clowning Around?

Figure 12 shows the layout of a figure that, for lack of better words I have called a clown. I shall use the figure to introduce the subject of simple animation and a few other novelties. Note that Figure 12 shows several parts of the clown identified by a different type of

Figure 12



shading. These parts include the center part of the mouth, the eyes, and a "finger" on each hand. These parts of the character are not drawn by the main subroutines. They are filled in by small subroutines that will serve to animate the diagram.

Listing 10 provides the software for the clown. The main clown-drawing subroutine starts at line 9000. Because the clown takes up almost all the vertical space on the display of a typical low resolution system such as an APPLE-II, the subroutine does not provide for varying the reference point along the Y axis. (Indeed, if you want to draw the clown on a PET unit, you will have to be satisfied with the upper half of the figure! Note, however, that I did not attempt to animate anything below the waist, so you PET users will be able to save yourself some work here and still get all the benefits of the discussion.)

Listing 10

```
 10 GR
 15 GOSUB 9980
 20 COLOR=7
 30 GOSUB 9000
 40 Y=9:A=5
 50 FOR M=1 TO RND (5)
 60 GOSUB 9600
 70 S= RND (100 )+100:T=40: GOSUB
    9990
 80 GOSUB 9650
 90 S= RND (60 )+20:T=40: GOSUB
    9990
100 GOSUB 9600
110 S= RND (100 )+100:T=40: GOSUB
    9990
120 GOSUB 9650
130 S= RND (60 )+20:T=40: GOSUB
    9990
140 NEXT M
150 FOR I=1 TO 500: NEXT I
160 Y=4
170 GOSUB 9700
180 S=60:T=20: GOSUB 9990
190 GOSUB 9725
200 FOR I=1 TO 200: NEXT I
210 GOSUB 9750
220 GOSUB 9990
230 GOSUB 9775
240 FOR I=1 TO 500: NEXT I
250 Y=15
260 GOSUB 9800
270 S=240:T=240: GOSUB 9990
280 GOSUB 9825
290 FOR I=1 TO 500: NEXT I
```

```
 300 GOSUB 9850
 310 GOSUB 9990
 320 GOSUB 9875
 330 FOR I=1 TO RND (10000): NEXT
     I
 340 GOTO 40
9000 Y=1:A=5
9010 FOR X=A+12 TO A+17: PLOT X,
     Y: NEXT X
9020 Y=Y+1: PLOT A+11,Y: PLOT A+
     18,Y
9030 Y=Y+1: PLOT A+10,Y: PLOT A+
     12,Y: PLOT A+17,Y: PLOT A+19
     ,Y
9040 Y=Y+1: PLOT A+9,Y: PLOT A+11
     ,Y: PLOT A+13,Y: PLOT A+16,
     Y: PLOT A+18,Y: PLOT A+20,Y
9050 Y=Y+1: PLOT A+9,Y: PLOT A+12
     ,Y: PLOT A+17,Y: PLOT A+20,
     Y
9060 Y=Y+1: PLOT A+9,Y: PLOT A+14
     ,Y: PLOT A+15,Y: PLOT A+20,
     Y
9070 Y=Y+1: PLOT A+9,Y: PLOT A+20
     ,Y
9080 Y=Y+1: PLOT A+9,Y
9090 FOR X=A+12 TO A+17: PLOT X,
     Y: NEXT X
9100 PLOT A+20,Y
9110 Y=Y+1: PLOT A+9,Y: PLOT A+12
     ,Y: PLOT A+17,Y: PLOT A+20,
     Y
9120 Y=Y+1: PLOT A+10,Y
9130 FOR X=A+12 TO A+17: PLOT X,
     Y: NEXT X
9140 PLOT A+19,Y
9150 Y=Y+1: PLOT A+11,Y: PLOT A+
     18,Y
9160 Y=Y+1: FOR X=A+12 TO A+17: PLOT
     X,Y: NEXT X
9170 Y=Y+1: FOR X=A+12 TO A+17: PLOT
     X,Y: NEXT X
9180 Y=Y+1: PLOT A+2,Y
9190 FOR X=A+7 TO A+22: PLOT X,Y:
      NEXT X
9200 PLOT A+27,Y
9210 Y=Y+1: FOR X=A+1 TO A+28: PLOT
     X,Y: NEXT X
9220 Y=Y+1: FOR X=A+1 TO A+28: PLOT
     X,Y: NEXT X
9230 Y=Y+1: FOR X=A+1 TO A+6: PLOT
     X,Y: NEXT X
```

```
9240 FOR X=A+10 TO A+19: PLOT X,
     Y: NEXT X
9250 FOR X=A+23 TO A+28: PLOT X,
     Y: NEXT X
9260 Y=Y+1: PLOT A+1,Y: PLOT A+2
     ,Y
9270 FOR X=A+10 TO A+19: PLOT X,
     Y: NEXT X
9280 PLOT A+27,Y: PLOT A+28,Y
9290 FOR Y=Y+1 TO Y+7
9300 FOR X=A+10 TO A+19: PLOT X,
     Y: NEXT X
9310 NEXT Y
9320 FOR X=A+10 TO A+13: PLOT X,
     Y: NEXT X
9330 FOR X=A+16 TO A+19: PLOT X,
     Y: NEXT X
9340 Y=Y+1: FOR X=A+9 TO A+13: PLOT
     X,Y: NEXT X
9350 FOR X=A+16 TO A+20: PLOT X,
     Y: NEXT X
9360 Y=Y+1: FOR X=A+9 TO A+12: PLOT
     X,Y: NEXT X
9370 FOR X=A+17 TO A+20: PLOT X,
     Y: NEXT X
9380 Y=Y+1: FOR X=A+9 TO A+12: PLOT
     X,Y: NEXT X
9390 FOR X=A+17 TO A+20: PLOT X,
     Y: NEXT X
9400 Y=Y+1: FOR X=A+9 TO A+12: PLOT
     X,Y: NEXT X
9410 FOR X=A+17 TO A+20: PLOT X,
     Y: NEXT X
9420 Y=Y+1: FOR X=A+8 TO A+12: PLOT
     X,Y: NEXT X
9430 FOR X=A+17 TO A+21: PLOT X,
     Y: NEXT X
9440 Y=Y+1: FOR X=A+8 TO A+11: PLOT
     X,Y: NEXT X
9450 FOR X=A+18 TO A+21: PLOT X,
     Y: NEXT X
9460 Y=Y+1: FOR X=A+8 TO A+11: PLOT
     X,Y: NEXT X
9470 FOR X=A+18 TO A+21: PLOT X,
     Y: NEXT X
9480 Y=Y+1: PLOT A+9,Y: PLOT A+10
     ,Y: PLOT A+19,Y: PLOT A+20,
     Y
9490 Y=Y+1: FOR X=A+6 TO A+11: PLOT
     X,Y: NEXT X
9500 FOR X=A+18 TO A+23: PLOT X,
```

```
    Y: NEXT X
9510 Y=Y+1: FOR X=A+5 TO A+11: PLOT
     X,Y: NEXT X
9520 FOR X=A+18 TO A+24: PLOT X,
     Y: NEXT X
9530 RETURN
9600 COLOR=11: REM   COLOR STATEMENT
     FOR COLOR SYSTEMS
9610 FOR X=A+13 TO A+16: PLOT X,
     Y: NEXT X
9620 RETURN
9650 COLOR=0: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9660 FOR X=A+13 TO A+16: PLOT X,
     Y: NEXT X
9670 RETURN
9700 COLOR=6: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9705 PLOT A+12,Y
9710 RETURN
9725 COLOR=0: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9730 PLOT A+12,Y
9735 RETURN
9750 COLOR=6: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9755 PLOT A+17,Y
9760 RETURN
9775 COLOR=0: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9780 PLOT A+17,Y
9785 RETURN
9800 COLOR=9: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9805 PLOT A,Y
9810 RETURN
9815 PRINT S,T
9820 GOSUB 9990
9825 COLOR=0: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9830 PLOT A,Y
9835 RETURN
9840 NEXT S
9850 COLOR=9: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9855 PLOT A+29,Y
9860 RETURN
9875 COLOR=0: REM   COLOR STATEMENT F
     OR COLOR SYSTEMS
9880 PLOT A+29,Y
9885 RETURN
9980 POKE 2,173: POKE 3,48: POKE
```

```
     4,192: POKE 5,136: POKE 6,208
     : POKE 7,4: POKE 8,198: POKE
     9,1: POKE 10,240
9985 POKE 11,8: POKE 12,202: POKE
     13,208: POKE 14,246: POKE 15
     ,166: POKE 16,0: POKE 17,76
     : POKE 18,2: POKE 19,0: POKE
     20,96: RETURN
9990 POKE 0,S: POKE 1,T: CALL 2:
      RETURN
9999 END
```

When the subroutine starting at line 9000 is executed, the clown will be drawn on the screen except for the center of the eyes, the middle of the mouth, and the index fingers on either hand. Note that you can vary the horizontal position of the figure by initializing the value of the variable A. You TRS-80 owners will probably want to double the number of sectors used along the horizontal axis to get a figure proportionally equivalent to that shown.

Animation Makes It Look Alive — Well, Almost!

Now the fun begins. The subroutine that starts at line 9600 causes the middle portion of the clown's mouth to be filled in. Another subroutine at line 9650 restores the mouth to the "open" position. By alternating the two subroutines, the clown can be made to appear as though it is opening and closing its mouth!

Similarly, a simple subroutine at line 9700 causes the left eye to close. The subroutine at 9725 causes the left eye to open back up. Execute the one at 9700, provide a slight delay, then execute the one at 9725 and the eye will appear to wink.

Subroutines at 9750 and 9775 can be invoked to cause the right eye to wink, too!

". . . by judiciously calling on the 'action' subroutines, you can have the clown open and shut its mouth, wink either or both eyes, and point to the right or the left."

You can call on the subroutine that starts at line 9800 to have a simulated index finger appear on the clown's left hand. The subroutine at line 9825 will make that finger disappear. Similarly, routines at 9850 and 9875 control the action of such a finger on the right-hand side of the finger.

So now, by judiciously calling on the "action" subroutines, you can have the clown open and shut its mouth, wink either or both eyes, and point to the right or the left. Can you think of ways to combine such a figure with a game or quiz to amuse people? I assure you, newcomers to computers get quite a kick out of seeing such a performance.

Now Add Some Sound

People expect something that opens and closes its mouth to make some noise. Well, you can synchronize some other simple subroutines with the animation subroutines just discussed so that the clown beeps and buzzes as it moves its mouth. It's good for a laugh from most people!

If you have an APPLE-II computer, you can use the noise-making subroutines shown in Listing 10. The subroutine starting at line 9980 sets up the basic sound generating program that is recommended in the *APPLE II Reference Manual*, published by APPLE Computer Incorporated (January 1978 edition) that comes with the basic APPLE-II machine. The subroutine at 9990 sets up the parameter values for frequency and duration of the tone that are passed to it and activates the sound unit.

Lines 10 through 340 in Listing 10 tie together all the various subroutines mentioned to provide a demonstration sequence of animation complete with sound effects. The use of some random numbers provides for a more interesting demonstration. This is done by varying the number of times that the clown opens and closes its mouth during each cycle. Random numbers also vary the pitch at which the clown emits buzzes and beeps. Study lines 10 through 340 so that you can see how the various subroutines are sequenced to provide the animation and synchronized noises. Remember, the sequence was chosen for demonstration purposes. Feel free to make your own variations. All I am trying to do here is illustrate concepts. Once you grasp them, you have the freedom to run off and create funny acts of your own!

They All Can Do It

Oh yes, almost any popular computer can make beeps and buzzes even if it does not have a circuit for that specific purpose built into it. If it is able to store data on an external magnetic tape device, you can probably jury-rig it to make entertaining burbles. You certainly can do this with a Radio Shack TRS-80 or Commodore PET. Just insert a short little tape-write subroutine in place of the noise generating subroutine I showed at line 9990 in Listing 10. If you want to get really fancy, create two or three such subroutines: one that writes, for instance, a series of ones to the tape unit; another that writes a series of zeros; and still another that writes alternating ones and zeros. You will then be able to select one of three different tones or buzzes.

Now refer to Figure 13. It shows a block diagram of how you can jury-rig your tape recorder system to hear the sounds you create with your tape-write subroutines. Unless you have a small speaker/

amplifier, you will have to be satisfied with listening to the sounds using an earphone. However, the electronic technician types among you will undoubtably have little difficulty hooking up a small speaker/amplifier so that your friends will be able to hear things through a speaker. (Frankly, you might be wise to keep the earphone arrangement available. For some reason, other members of your household might not appreciate your computer emitting burps and beeps at the wee hours of the night. Aren't those just the times when you want to run some of those fun-and-game programs that emit all those wierd noises?)

The arrangement shown in Figure 13 requires that you put the tape recorder in the "record" mode *without* having a tape cassette unit installed. Most cassette players have a "record lockout" switch that normally prevents the tape recorder from going into the record mode unless an "unprotected" cassette is installed in the machine. An unprotected cassette is one that has not had its recording tab knocked out. With the tab in place, the cassette will push against the record lockout switch when it is installed in the player.

Figure 13

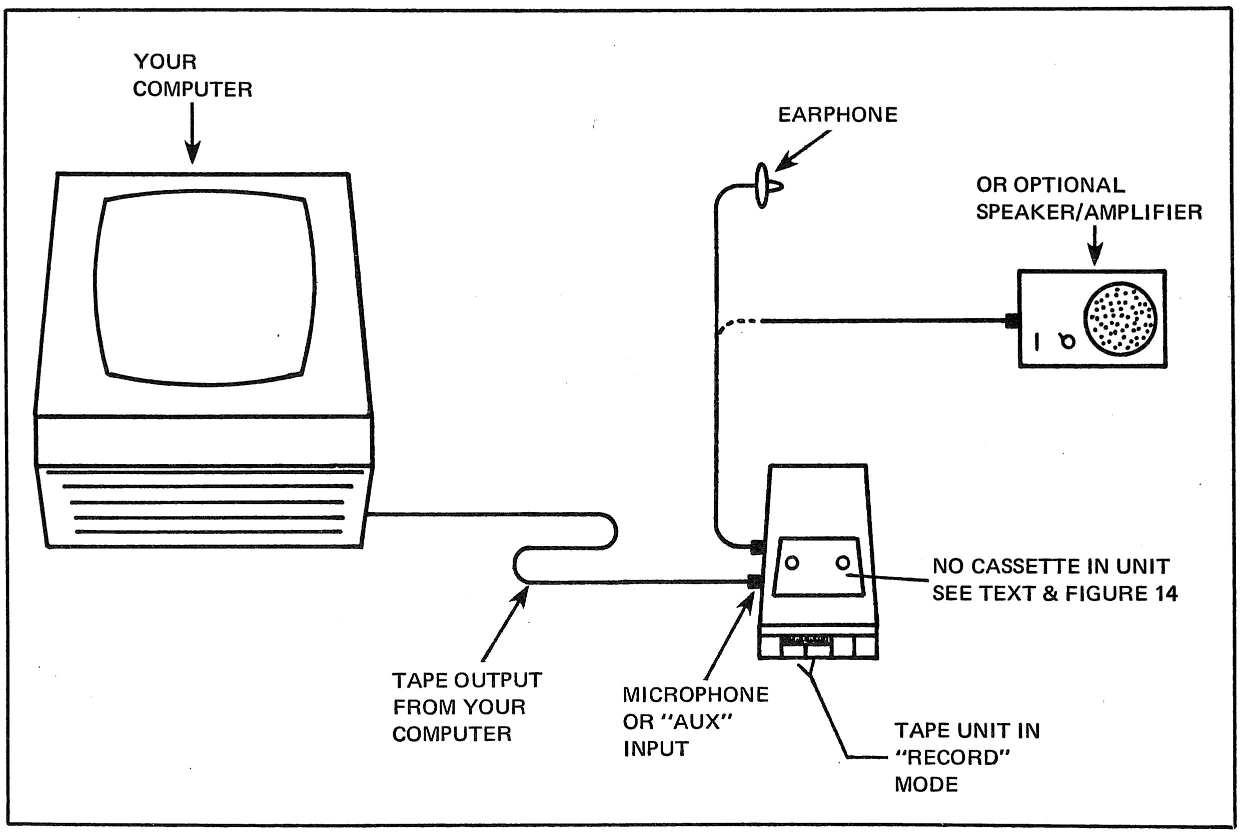


Figure 14 shows the typical location of the record lock-out switch on a tape cassette player. You can make the player think it has an unprotected cassette installed by taping the switch closed (as though a cassette were pushing against it) or using a plastic pen or similar object lightly wedged into the unit to keep the switch activated. With the switch activated, you should then be able to place the recorder in the normal record mode. When in this mode, most players will couple whatever is fed into the microphone or auxiliary input to the earphone jack so that monitoring can take place. It is this feature that you wish to take advantage of in order to hear what your computer sends out to the tape unit. Since the sounds are for entertainment only, there is no need to waste good tape by actually recording the nonsense sounds.

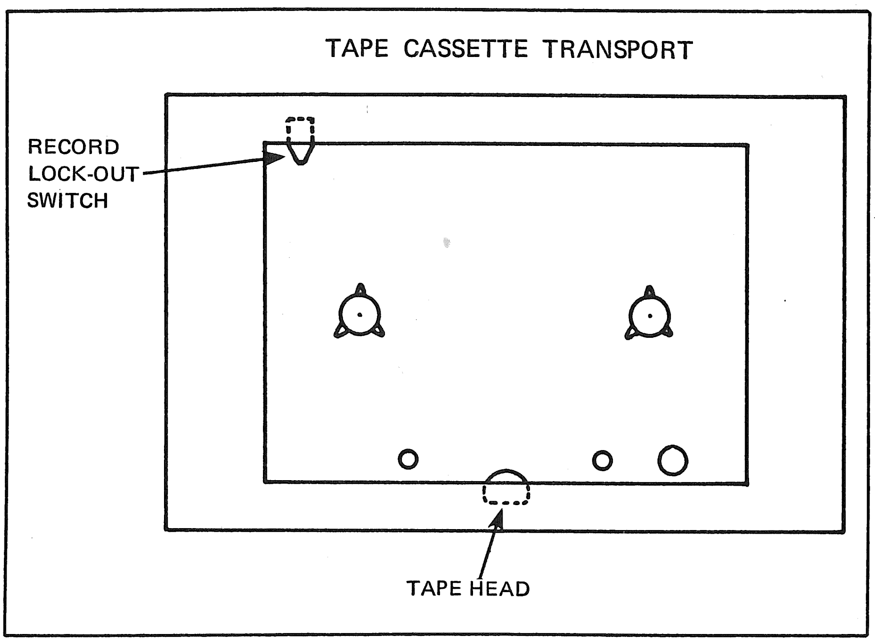
This simple arrangement provides a quick and inexpensive way for you to liven up animated performances as they appear on your display screen.

An Animated Game of Football

Now that you have a fundamental background in how to create and position graphic symbols, it is time to tie all that you have learned together as a final exercise. In doing so you will create a game that is graphically entertaining.

The game is called football. The object of the game is to guide a quarterback through a field of defenders to gain yardage. Gain enough yards and you score a goal. Or, in certain situations you can

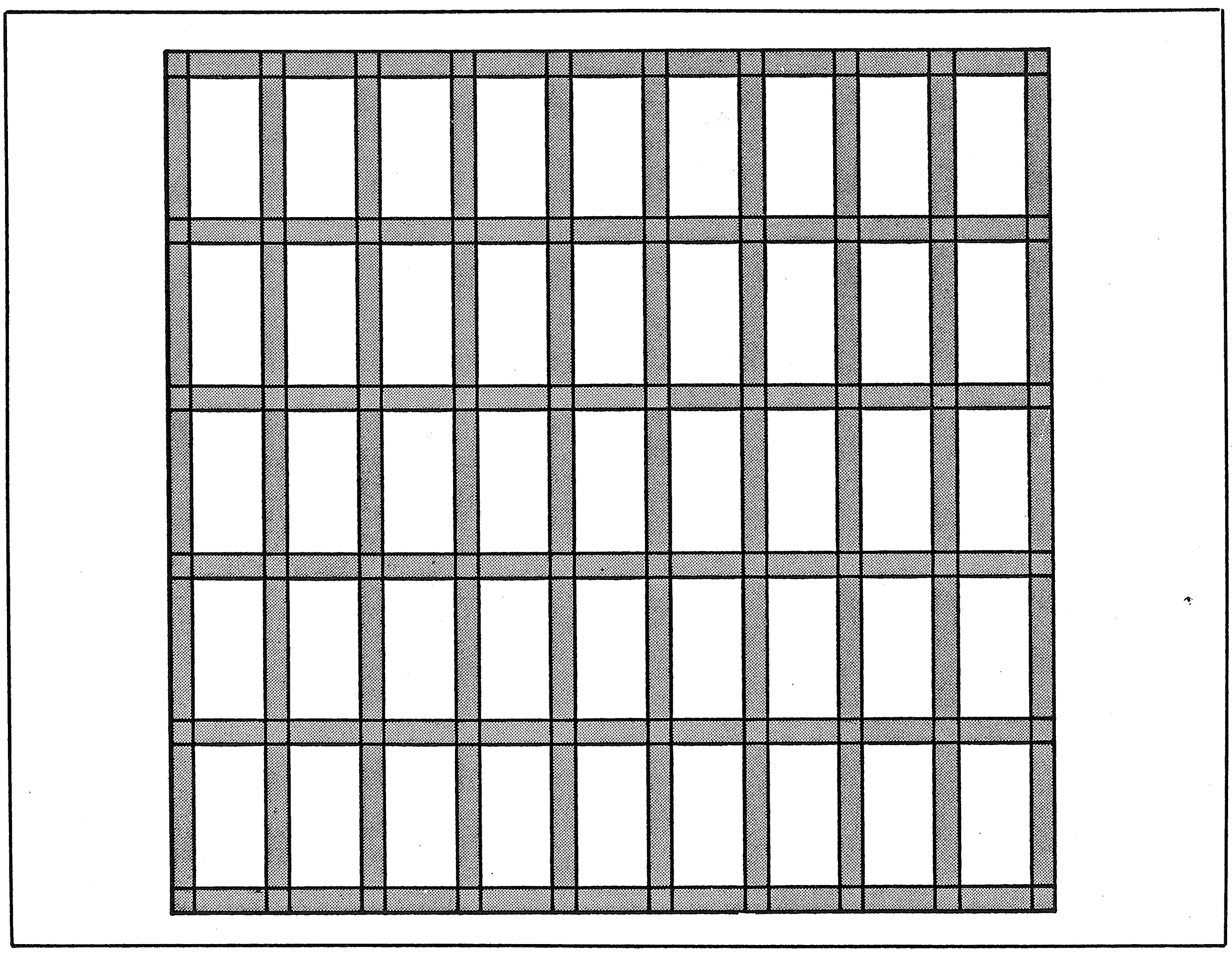
Figure 14



try to kick a field goal. Of course, all of the running and defending is done on your computer's video screen. But, you get to control the direction of the quarterback's movement. (The computer gets to control the defense!)

It should be remembered here that the version I am presenting is merely a guideline. The listing shown is for an APPLE-II system. However, by following the discussion and referring to the listing, you can gain the knowledge to modify the program to run on a PET, TRS-80 or other type of low resolution display system. Don't be afraid to try your own ideas. Change the graphic symbols if you like. Change the way the game is controlled or played if you want to! After all, the whole point of a computer is that it provides immense freedom of choice and creativity. Exercise some of that capability to suit your own tastes!

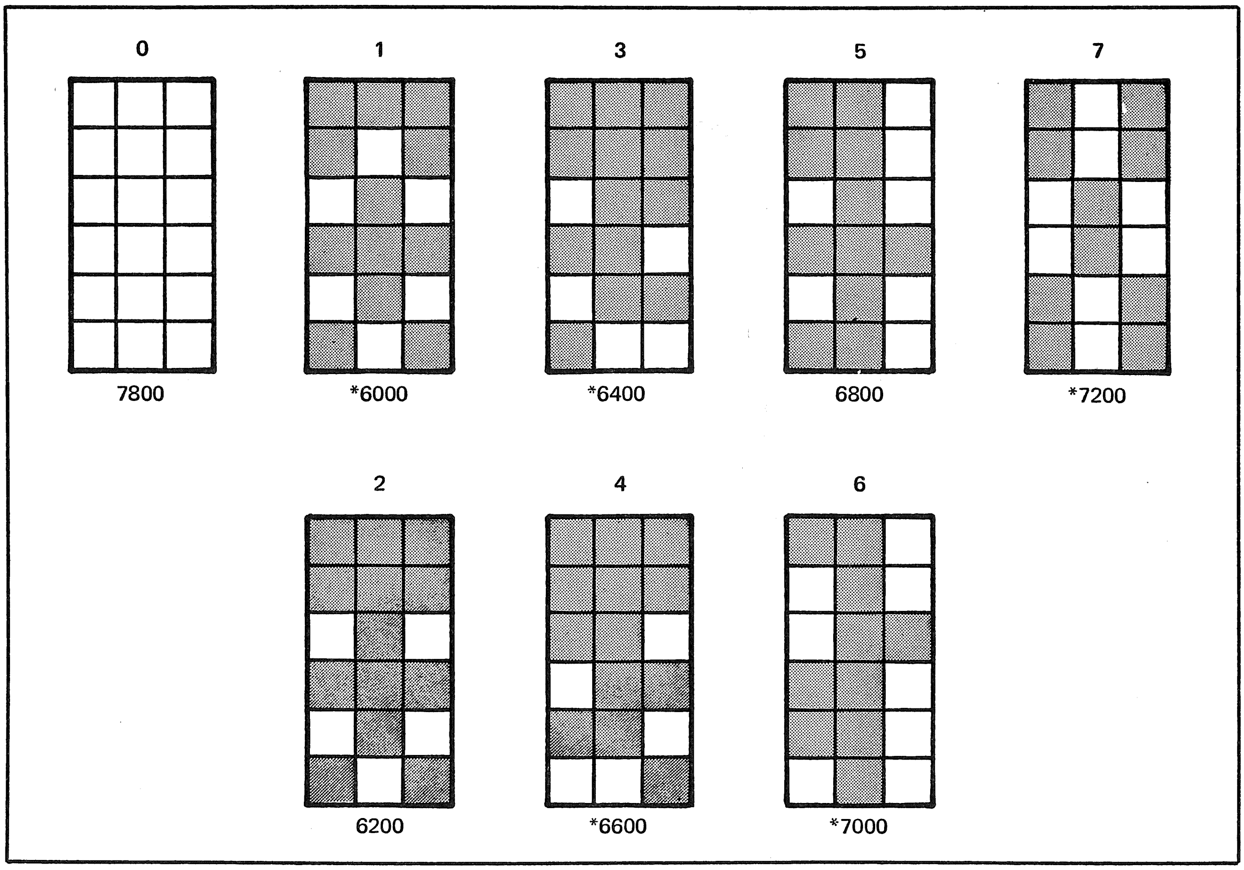
Figure 15



The game I am presenting takes place on a grid consisting of a 9 by 5 matrix as illustrated in Figure 15. Each box in the matrix is made up of 18 pixels arranged in a 3 by 6 fashion. Counting the grid lines, that are all one sector wide, the entire grid is defined by an area that is 37 units wide by 36 units high. This fits comfortably with some room to spare on an APPLE-II display. It will not fit in the vertical direction on a PET display. I suggest that you reduce the number of vertical boxes to three if you create this program for a PET system. On a TRS-80, you will probably want to increase the number of horizontal sectors to a box. If you do not, the animated characters are going to look rather skinny on the display!

The Screen Cast The stars of our football game are tiny animated characters as depicted in Figure 16. There are seven basic configurations in which the players can appear plus a blank or "no character" symbol. The latter is needed to erase the previous position of a displayed player. Notice that some of the characters are drawn as pairs. Characters 1

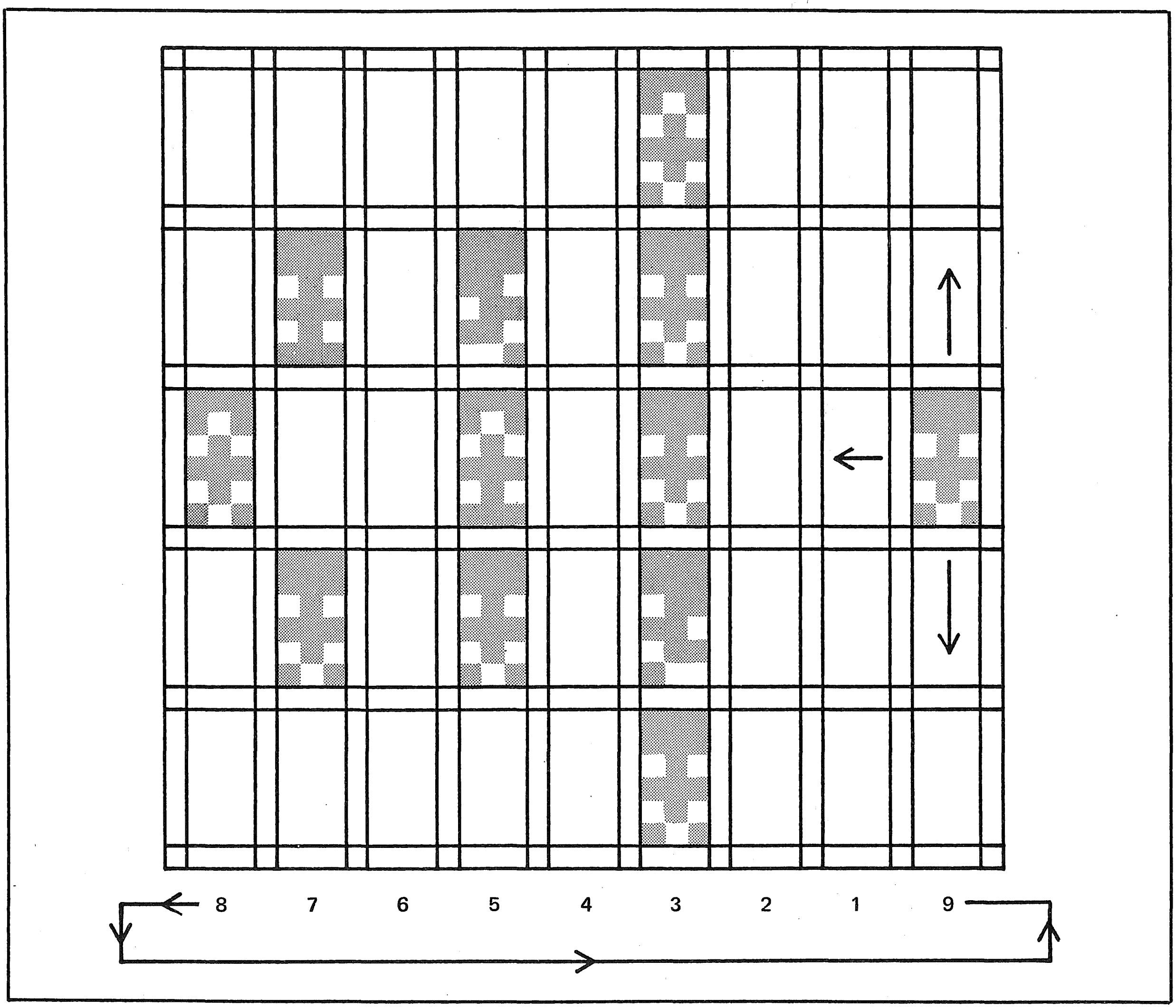
Figure 16



and 2 in Figure 16 will be paired together in an alternating fashion to represent a player that is opening and closing its mouth. Characters number three and four can be alternated to show an animated version of a player "running." You might not think so from seeing characters number 5 and 6 in Figure 16, but they can be alternately displayed to symbolize a player punting the football. Figure number 7 represents a player that has been flattened after being tackled! (Come on! Use your imagination. You have to use a little if you want to utilize low resolution graphics effectively. Don't judge the characters until you see them in action on a video screen!)

Figure 17 illustrates how the characters might typically appear in a "frozen frame" shot at the start of a game. For the APPLE-II version there are 11 defensemen lined up as shown. If you only have

Figure 17



a 9 by 3 box matrix, such as might be used on a PET, I would recommend you reduce the number of defensemen to something in the order of five or six. The defensemen are all automatically controlled by the computer. They will at all times advance upon and attempt to contact or "tackle" the quarterback. The quarterback is shown in Figure 17 all alone on the right end of the grid. Different colors are used to distinguish between the quarterback and the defensive players on an APPLE-II system. For black and white systems you can change the figure used to represent the quarterback. Or, on a system such as the PET, you can construct the quarterback using different graphic symbols than that used for the defensemen.

The arrows eminating from the quarterback in Figure 17 show the possible directions of movement afforded to the quarterback. The direction of movement is always under the control of a person serving as the coach of the offensive team. Note, however, that the quarterback can never run backwards. While evasive maneuvers can be made up and down on the screen (across the playing field), the loyal quarterback can only attempt to gain yardage, not fritter it away! (For most people, attempting to gain yardage will be enough of a challenge. It is not as easy as it might look at first glance.)

Each time the quarterback is able to advance a square without contacting or being contacted by a defenseman, a "yard" is gained. If the quarterback gets all the way across the grid, it is advanced, on the next move, back to the right side. This is depicted by the numbers shown at the bottom of the diagram in Figure 17. (These numbers and the arrows surrounding the quarterback are not produced on the display. They are provided for illustrative purposes only in Figure 17.)

Setting the Stage and the Cast

Listing 11 contains the program for the football game. The program is structured as a number of subroutines that are in turn tied together by several control routines.

The playing grid is drawn by the subroutine that starts at line 9000 in the listing. Nested FOR-NEXT loops are used to draw all the horizontal lines of the grid and then all of the vertical lines. The grid lines are drawn in white for an APPLE-II color system. On a black and white system, such as a TRS-80, you might not want to actually draw the playing grid. This is because a grid that is not distinguished by a different color from the graphic symbols may not be as pleasing to some as simply having an implied grid. Try it both ways, with and without a grid, to see which version you and your friends like better. Or, you can make the grid lines appear different by using a different kind of character, such as an asterisk. This is the technique to use on a PET. However, you can also use it to POKE a unique grid on the TRS-80 or similar systems. Again, don't be afraid to experiment. Take a look at some different methods; then use what *you* like best.

The cast of characters used in the game are defined in a series of subroutines starting at line 6000. The start of each character is indicated by the line number given below each in Figure 16. Notice also that some line numbers in Figure 16 are preceded by an asterisk. This indicates that some sound generation is associated with that configuration of the character.

Refer to Listing 11, line 6000 to study the development of a typical character. The first few lines in each subroutine have been left available to set the type of symbol one might want to use or define the color in a color system. Color definition for an APPLE-II has been done in line 6010 of the listing. The actual creation of character type number one (as designated in Figure 16) begins at line 6050.

Listing 11

```
1  GR
10 P = 80
20 Q = 10
30 V = 1
40 T = 900
50  GOSUB 1900
100 G = 1
200  IF G = 0 THEN G = 1: GOTO 22
     0
210  IF G = 1 THEN G = 0
220 D = 0:Q = 10
300  GOSUB 1900
1000  FOR D = 1 TO 4
1010  GOSUB 2000
1020 P = P - U: IF P <  = 0 THEN
     B = 7: GOTO 1400
1030 Q = Q - U: IF Q <  = 0 THEN
     D = 0:Q = 10: GOSUB 1700
1040  IF T <  = 0 THEN T = 0: GOTO
     1800
1060  IF D = 3 THEN  PRINT : PRINT
     "YOU NEED ";Q;" YARDS FOR A
     FIRST DOWN!": PRINT : PRINT
     "DO YOU WANT TO TRY A KICK";
     : INPUT W$: IF  LEFT$ (W$,1)
      = "Y" THEN 1500
1070  IF D < 4 THEN  GOSUB 1900
1080  NEXT D
1090 P = 100 - P: GOTO 200
```

```
1400  IF G = 0 THEN S2 = S2 + B
1410  IF G = 1 THEN S1 = S1 + B
1420 P = 80: GOTO 200
1500  GOSUB 3000
1530  FOR I = 1 TO 500
1540  GOSUB 7000
1550  FOR I = 1 TO 100
1560  GOSUB 6800
1570 U =  INT ( RND (1) * 55 + 1)

1580 P = P - U: IF P <  = 0 THEN
     I =  RND (1): IF I > 0.5 THEN
      PRINT : PRINT : PRINT "HURR
     AH! YOU KICKED A FIELD GOAL!
     ": PRINT :B = 3: GOTO 1400
1590  IF P <  = 0 THEN P = 80: GOTO
     200
1600 P = 100 - P: GOTO 200
1700  PRINT : PRINT "YOU PICKED U
     P A FIRST DOWN."
1710  PRINT : PRINT "CONGRATULATI
     ONS -- YOU BIG OX !!"
1720  FOR J = 1 TO 5000: NEXT J
1730  RETURN
1800 V = V + 1: IF V > 4 THEN V =
     4: GOTO 1840
1810 T = 900
1820  IF V = 3 THEN G = 1:P = 80:
     Q = 10:D = 0
1830  GOTO 300
1840  GR
1850  PRINT : PRINT : PRINT "IT'S
      ALL OVER - EXCEPT FOR THE S
     HOUTING!": FOR I = 1 TO 2000
     : NEXT I: GOSUB 1900: GR
1860  PRINT : PRINT : PRINT "WANT
      A NEW GAME";: INPUT W$: IF
      LEFT$ (W$,1) = "Y" THEN  GOTO
     10
1870  PRINT : PRINT : PRINT : PRINT
     : GOTO 9999
1900  PRINT " DOWN: ";D + 1;"  MA
     RKER: ";P;"  YARDS TO GO: ";
     Q
1910  PRINT "  TIME REMAINING: ";
     T;"   QUARTER: ";V
1920  PRINT "BLUE TEAM HAS: ";S1;
     "   YELLOW TEAM HAS: ";S2
1930  FOR J = 1 TO 2000: NEXT J
1940  RETURN
2000  FOR M = 0 TO 8
2010  FOR N = 0 TO 4
2020 A(M,N) = 0
2030  NEXT N
2040  NEXT M
2050 A(0,2) = 1
2060 A(1,1) = 1
2070 A(1,3) = 1
2080 A(3,1) = 1
2090 A(3,2) = 1
2100 A(3,3) = 1
2110 A(5,0) = 1
2120 A(5,1) = 1
2130 A(5,2) = 1
2140 A(5,3) = 1
2150 A(5,4) = 1
2200  GOSUB 9000
2210 X = 32:Y = 14: IF G = 0 THEN
     C = 13: GOTO 2220
2215 C = 2
2220  GOSUB 6200
2230  IF G = 0 THEN C = 2: GOTO 2
     240
2235 C = 13
2240  GOSUB 9200
2250 M = 0:N = 0:U = 0
2260 X = 32:Y = 14:X2 = 32:Y2 = 1
     4: IF G = 0 THEN C = 13: GOTO
     2270
2265 C = 2
2270  GOSUB 8000
2280 X = X2:Y = Y2: IF G = 0 THEN
     C = 13: GOTO 2290
2285 C = 2
2290  GOSUB 8000
2295  IF F = 1 THEN  GOTO 2330
2300  GOSUB 9500
2310  IF F = 0 THEN  GOTO 2280
2330 X = X2:Y = Y2
2340  GOSUB 7200
2350  RETURN
3000  FOR M = 0 TO 8
3010  FOR N = 0 TO 4
3020 A(M,N) = 0
3030  NEXT N
3040  NEXT M
3050 A(0,2) = 1
3060 A(1,1) = 1
3070 A(1,3) = 1
3080 A(3,1) = 1
3090 A(3,2) = 1
3100 A(3,3) = 1
3110 A(5,0) = 1
3120 A(5,1) = 1
3130 A(5,2) = 1
3140 A(5,3) = 1
```

```
3150 A(5,4) = 1
3200  GOSUB 9000
3210 X = 32:Y = 14: IF G = 0 THEN
     C = 13: GOTO 3220
3215 C = 2
3220  GOSUB 6800
3230  IF G = 0 THEN C = 2: GOTO 3
     240
3235 C = 13
3240  GOSUB 9200
3250 X = 32:Y = 14: IF G = 0 THEN
     C = 13: GOTO 3260
3255 C = 2
3260  RETURN
6000  REM    FIGURE TYPE # 1
6010  COLOR= C
6050  PLOT X + 1,Y + 1
6060  PLOT X + 2,Y + 1
6070  PLOT X + 3,Y + 1
6080  PLOT X + 1,Y + 2
6090  PLOT X + 3,Y + 2
6100  PLOT X + 2,Y + 3
6110  PLOT X + 1,Y + 4
6120  PLOT X + 2,Y + 4
6130  PLOT X + 3,Y + 4
6140  PLOT X + 2,Y + 5
6150  PLOT X + 1,Y + 6
6160  PLOT X + 3,Y + 6
6170  COLOR= 0
6172  PLOT X + 2,Y + 2
6174  PLOT X + 1,Y + 3
6176  PLOT X + 3,Y + 3
6178  PLOT X + 1,Y + 5
6180  PLOT X + 3,Y + 5
6182  PLOT X + 2,Y + 6
6184 S =  - 16336
6186  FOR I = 1 TO 20
6188 R =  PEEK (S) -  PEEK (S) -
      PEEK (S) -  PEEK (S) -  PEEK
     (S) -  PEEK (S)
6190  NEXT I
6199  RETURN
6200  REM    FIGURE TYPE # 2
6210  COLOR= C
6250  PLOT X + 1,Y + 1
6260  PLOT X + 2,Y + 1
6270  PLOT X + 3,Y + 1
6280  PLOT X + 1,Y + 2
6290  PLOT X + 2,Y + 2
6300  PLOT X + 3,Y + 2
6310  PLOT X + 2,Y + 3
6320  PLOT X + 1,Y + 4
6330  PLOT X + 2,Y + 4
6340  PLOT X + 3,Y + 4
6350  PLOT X + 2,Y + 5
6360  PLOT X + 1,Y + 6
6365  PLOT X + 3,Y + 6
6370  COLOR= 0
6374  PLOT X + 1,Y + 3
6376  PLOT X + 3,Y + 3
6378  PLOT X + 1,Y + 5
6380  PLOT X + 3,Y + 5
6382  PLOT X + 2,Y + 6
6399  RETURN
6400  REM    FIGURE TYPE # 3
6410  COLOR= C
6420 S =  - 16336
6425  FOR I = 1 TO 2
6430 R =  PEEK (S) -  PEEK (S) -
      PEEK (S)
6435  NEXT I
6450  PLOT X + 1,Y + 1
6455  PLOT X + 2,Y + 1
6460  PLOT X + 3,Y + 1
6465  PLOT X + 1,Y + 2
6470  PLOT X + 2,Y + 2
6475  PLOT X + 3,Y + 2
6480  PLOT X + 2,Y + 3
6485  PLOT X + 1,Y + 4
6490  PLOT X + 2,Y + 4
6495  PLOT X + 3,Y + 3
6500  PLOT X + 2,Y + 5
6505  PLOT X + 3,Y + 5
6510  PLOT X + 1,Y + 6
6520  COLOR= 0
6530  PLOT X + 1,Y + 3
6535  PLOT X + 3,Y + 4
6540  PLOT X + 1,Y + 5
6545  PLOT X + 2,Y + 6
6550  PLOT X + 3,Y + 6
6599  RETURN
6600  REM    FIGURE TYPE # 4
6610  COLOR= C
6620 S =  - 16336
6625  FOR I = 1 TO 2
6630 R =  PEEK (S) -  PEEK (S) -
      PEEK (S)
6635  NEXT I
6650  PLOT X + 1,Y + 1
6655  PLOT X + 2,Y + 1
6660  PLOT X + 3,Y + 1
6665  PLOT X + 1,Y + 2
6670  PLOT X + 2,Y + 2
6675  PLOT X + 3,Y + 2
```

```
6680  PLOT X + 2,Y + 3
6685  PLOT X + 1,Y + 3
6690  PLOT X + 2,Y + 4
6695  PLOT X + 3,Y + 4
6700  PLOT X + 1,Y + 5
6705  PLOT X + 2,Y + 5
6710  PLOT X + 3,Y + 6
6720  COLOR= 0
6730  PLOT X + 1,Y + 4
6735  PLOT X + 3,Y + 3
6740  PLOT X + 3,Y + 5
6745  PLOT X + 1,Y + 6
6750  PLOT X + 2,Y + 6
6799  RETURN
6800  REM    FIGURE TYPE #  5
6810  COLOR= C
6850  PLOT X + 1,Y + 1
6855  PLOT X + 2,Y + 1
6860  PLOT X + 1,Y + 2
6865  PLOT X + 2,Y + 2
6870  PLOT X + 2,Y + 3
6875  PLOT X + 1,Y + 4
6880  PLOT X + 2,Y + 4
6885  PLOT X + 3,Y + 4
6890  PLOT X + 2,Y + 5
6895  PLOT X + 1,Y + 6
6900  PLOT X + 2,Y + 6
6920  COLOR= 0
6930  PLOT X + 3,Y + 1
6935  PLOT X + 3,Y + 2
6940  PLOT X + 1,Y + 3
6945  PLOT X + 3,Y + 3
6950  PLOT X + 1,Y + 5
6955  PLOT X + 3,Y + 5
6960  PLOT X + 3,Y + 6
6999  RETURN
7000  REM    FIGURE TYPE #  6
7010  COLOR= C
7020 S =  - 16336
7025  FOR I = 1 TO 20:R =  PEEK (
     S): NEXT I
7050  PLOT X + 1,Y + 1
7055  PLOT X + 2,Y + 1
7060  PLOT X + 2,Y + 2
7065  PLOT X + 2,Y + 3
7070  PLOT X + 3,Y + 3
7075  PLOT X + 1,Y + 4
7080  PLOT X + 2,Y + 4
7085  PLOT X + 1,Y + 5
7090  PLOT X + 2,Y + 5
7095  PLOT X + 2,Y + 6
7100  COLOR= 0
7110  PLOT X + 3,Y + 1
7115  PLOT X + 1,Y + 2
7120  PLOT X + 3,Y + 2
7125  PLOT X + 1,Y + 3
7130  PLOT X + 3,Y + 4
7135  PLOT X + 3,Y + 5
7140  PLOT X + 1,Y + 6
7145  PLOT X + 3,Y + 6
7199  RETURN
7200  REM    FIGURE TYPE #  7
7210  COLOR= 11
7250  PLOT X + 1,Y + 1
7255  PLOT X + 3,Y + 1
7260  PLOT X + 1,Y + 2
7265  PLOT X + 3,Y + 2
7270  PLOT X + 2,Y + 3
7275  PLOT X + 2,Y + 4
7280  PLOT X + 1,Y + 5
7285  PLOT X + 3,Y + 5
7290  PLOT X + 1,Y + 6
7295  PLOT X + 3,Y + 6
7300  COLOR= 0
7305  PLOT X + 2,Y + 1
7310  PLOT X + 2,Y + 2
7315  PLOT X + 1,Y + 3
7320  PLOT X + 3,Y + 3
7325  PLOT X + 1,Y + 4
7330  PLOT X + 3,Y + 4
7335  PLOT X + 2,Y + 5
7340  PLOT X + 2,Y + 6
7350 S =  - 16336
7355  FOR I = 1 TO 150
7360 R =  PEEK (S)
7365  NEXT I
7370  RETURN
7800  REM    FIGURE TYPE #  0
7810  COLOR= 0
7850  FOR I = 1 TO 3
7860  FOR J = 1 TO 6
7870  PLOT X + I,Y + J
7880  NEXT J: NEXT I
7890  RETURN
8000 X1 = X2:Y1 = Y2
8010  IF  PDL (0) < 110 THEN Y =
     Y + 7
8020  IF  PDL (0) > 146 THEN Y =
     Y - 7
8030  IF Y < 0 THEN Y = 0
8040  IF Y > 28 THEN Y = 28
8050  FOR I = 1 TO 100
8060  IF  PEEK ( - 16287) > 127 THEN
     I = 100
```

```
8070  NEXT I
8080  IF  PEEK ( - 16287) > 127 THEN
     X = X - 4
8090  IF X > 32 THEN X = 32
8100  IF X < 0 THEN X = 32
8110  IF  PEEK ( - 16287) > 127 THEN
     8110
8120 T = T -  INT ( RND (1) * 5 +
     1)
8130  IF A(X / 4,Y / 7) <  > 0 THEN
     F = 1: RETURN
8140  IF X <  > X2 THEN U = U + 1

8150 X2 = X:Y2 = Y
8160 X = X1:Y = Y1
8170  GOSUB 7800
8180 X = X2:Y = Y2
8190  GOSUB 6200:F = 0: RETURN
9000  GR
9010  COLOR= 15
9020  FOR Y = 0 TO 35 STEP 7
9030  FOR X = 1 TO 36 STEP 1
9040  PLOT X,Y
9050  NEXT X
9060  NEXT Y
9070  FOR Y = 0 TO 35 STEP 1
9080  FOR X = 0 TO 36 STEP 4
9090  PLOT X,Y
9100  NEXT X
9110  NEXT Y
9120  RETURN
9200  FOR M = 0 TO 8
9210  FOR N = 0 TO 4
9220  IF A(M,N) = 0 THEN 9250
9230 X = M * 4:Y = N * 7
9235 X = M * 4:Y = N * 7
9240  GOSUB 6200
9250  NEXT N
9260  NEXT M
9270  RETURN
9500 F = 0: IF G = 0 THEN C = 2: GOTO
     9510
9505 C = 13
9510  FOR M = 0 TO 8: GOSUB 9550
9520  IF F = 1 THEN  RETURN
9530  NEXT M
9540 N = N + 1: IF N > 4 THEN N =
     0: RETURN
9550  IF A(M,N) = 0 THEN  RETURN

9560  IF  RND (1) > 0.2 THEN 9650
9570  IF X2 / 4 = M THEN 9650
9580  IF X2 / 4 < M THEN 9620
9590  IF (M + 1) * 4 = X2 AND N *
     7 = Y2 THEN F = 1: RETURN
9600  IF A(M + 1,N) <  > 0 THEN 9
     650
9610 A(M + 1,N) = 1:A(M,N) = 0:X =
     M * 4:Y = N * 7: GOSUB 7800:
      COLOR= C:X = (M + 1) * 4: GOSUB
     9800: RETURN
9620  IF (M - 1) * 4 = X2 AND N *
     7 = Y2 THEN F = 1: RETURN
9630  IF A(M - 1,N) <  > 0 THEN 9
     650
9640 A(M - 1,N) = 1:A(M,N) = 0:X =
     M * 4:Y = N * 7: GOSUB 7800:
      COLOR= C:X = (M - 1) * 4: GOSUB
     9800: RETURN
9650  IF  RND (1) > 0.4 THEN  RETURN

9660  IF Y2 / 7 = N THEN  RETURN

9670  IF Y2 / 7 < N THEN 9710
9680  IF M * 4 = X2 AND (N + 1) *
     7 = Y2 THEN F = 1: RETURN
9690  IF A(M,N + 1) <  > 0 THEN  RETURN
9700 A(M,N + 1) = 1:A(M,N) = 0:X =
     M * 4:Y = N * 7: GOSUB 7800:
      COLOR= C:Y = (N + 1) * 7: GOSUB
     9800: RETURN
9710  IF M * 4 = X2 AND (N - 1) *
     7 = Y2 THEN F = 1: RETURN
9720  IF A(M,N - 1) <  > 0 THEN  RETURN
9730 A(M,N - 1) = 1:A(M,N) = 0:X =
     M * 4:Y = N * 7: GOSUB 7800:
      COLOR= C:Y = (N - 1) * 7: GOSUB
     9800: RETURN
9800  IF  RND (1) > 0.2 THEN 9870

9810  FOR Z = 1 TO  RND (1) * 3 +
     1
9820  GOSUB 6400
9830  FOR J = 1 TO 50: NEXT J
9840  GOSUB 6600
9850  FOR J = 1 TO 50: NEXT J
9860  NEXT Z
9870  IF  RND (1) > 0.1 THEN 9940

9880  FOR Z = 1 TO  RND (1) * 3 +
     1
9890  GOSUB 6000
9900  FOR J = 1 TO 5: NEXT J
9910  GOSUB 6200
```

```
9920  FOR J = 1 TO 5: NEXT J
9930  NEXT Z
9940  GOSUB 6200: RETURN
9999  END
```

Note that lines 6050 through 6160 define every point within a box that is to be illuminated. Most important, too, is that line 6172 through 6182 define every point that is to be blanked! Failure to perform this step will result in problems when a character is repositioned on the screen and overlaid on top of a previous figure. (Line 6170 sets the color to zero for an APPLE-II which means black or "no color.") On a TRS-80, lines 6172 through 6182 would be RESET statements. On a PET, one would want to POKE blank characters in those positions.

'Note that line 6050 through 6160 define every point within a box that is to be illuminated."

Again, I must point out that these characters serve as guidelines. For instance, on a TRS-80 you would probably want the characters to contain more sectors in the horizontal direction. Perhaps you would want each box in a grid to be 6 by 6 units. If such is the case, you will have to define more points in each box in order to define a character.

Please take careful note that each point making up a character is defined as an X *plus displacement* and Y *plus displacement* value. It is most important that you construct your characters in this manner in order that they may be moved and positioned anywhere desired on the grid, by initializing the variables X and Y. I trust by this point that my previous lessons on this essential point have reached their mark!

Lines 6184 through 6190 are statements to have an APPLE-II emit a short buzzing sound. You will want to create similar capability using appropriate statements for your system if you want your animated players to have sound effects. Initially, I suggest you try to create your sound effects to last about half a second. You can change the duration later to suit your auditory channels.

The other characters shown in Figure 16 are drawn by subroutines beginning at lines 6200, 6400, 6800, 7000 and 7200, plus the special blank box that is drawn by the subroutine starting at line 7800. All of these subroutines will be called on as needed to draw and animate the characters.

One other feature I will point out is that you may notice that some of the character-drawing routines insert the sound effects before the figure is drawn. Others have it after the character has been drawn. This arrangement has been derived from experience so as to closely synchronize the sounds with the action. As always, however, I urge you to experiment with your own ideas if you are not satisfied with what you see and hear.

Setting the Defense

Figure 17 illustrates one way the defense can be set each time a play is started. The positions of the defensemen are defined by a two-dimensional array (named "A," having elements M,N). The initial positions of the defensemen are set up in a sequence of instructions by a subroutine that starts at line 2000. The subroutine that actually examines the array and positions figures on the playing grid starts at line 9200.

A position in the matrix M,N is set to the value one if a defenseman is to be placed at the corresponding position on the playing grid. Note that the cells of the playing grid, corresponding to positions in the array, are numbered from left to right and top to bottom, per the convention generally established when dealing with CRT displays. Also note that the actual starting position of a box is obtained by multiplying the array position by a multiplication factor in each direction. The factor is four in the X direction and seven in the Y direction for the grid shown. This is because each box, *when the grid line associated with it is included,* takes four sectors in the X direction and seven in the Y direction.

When the defense is first set, all of the characters are drawn using the character defined by the subroutine that starts at line 6200. In other words, there is no initial animation of the characters. In initial tests of the program I did animate the defense as it was set. I found that the extra time it takes to produce the animation tended to slow the pace of a game to the point where it became annoying. The defensemen thus do not start their antics until play gets underway.

Placing the Defensemen in Motion

One of the major subroutines and probably the most involved one in the program begins at line 9500. The purpose of this subroutine is primarily to move the defensemen towards the quarterback in order to accomplish a "tackle." In conjunction with these maneuvers, the subroutine also serves to "animate" the football characters.

The fundamental operation of the defense-movements subroutine is as follows. The M,N array is scanned to locate the positions of the defensemen in the grid. Each time one is located, its position is compared to that of the quarterback. If it is not on the same X coordinate on the grid as the quarterback, then approximately 20 percent of the time (using the random function) a defenseman will advance along the X direction, always towards the quarterback. If a defenseman is on the same X coordinate as the quarterback or it is not advanced along the X axis, then there is a 40 percent chance that it will advance *towards the quarterback* along the Y axis. If either type of move, in the X or Y direction, results in a defenseman making contact with the quarterback, then a "tackle" has occurred. Only one row of grid positions is manipulated each time the subroutine is entered. This is because of the interplay that takes place under program control between the defense and the manual control of the quarterback by a player.

"Since animation of the characters slows down play somewhat, the level of animation has been kept low."

Interwoven into the defense-movements subroutine are subroutines to animate a small percentage of the defensemen. Thus, from time-to-time, a defenseman will be exercised to make it appear as through it is shifting its feet or "running." Also, from time-to-time a defense character has its mouth animated while it emits sounds that are meant to mimic a charging tackler! Since animation of the characters slows down play somewhat, the level of animation has been kept low. However, the percentage of characters that are animated is readily changed by altering the values used in lines 9800 and 9870 of the listing.

The speed at which animation takes place is controlled by delay loops at lines 9830 and 9850 when character "running" is being portrayed. Lines 9900 and 9920 control the rate at which the mouth of a character is opened and closed during animation.

Control of the Quarterback

The motion of the quarterback is under the control of a person playing the game of football. The subroutine that translates the directives of a player to move the quarterback on the screen begins at line 8000 in the listing.

For the version presented in the listing, a system paddle is used as an input device. A "paddle" on an APPLE-II system is a variable resistor that can be controlled by a player. Associated with a paddle is a "button." This button is used in the listed version of this program to cause the quarterback to advance along the grid in order to gain yardage. Advancement always takes place from right to left along the grid as indicated in Figure 17.

Line 8010 and 8020 test the position of the player paddle and either increase or decrease the offset value for the quarterback in order to change its position along the Y axis. Lines 8030 and 8040 are used to limit the Y range so that the figure does not get offset completely out of the playing grid.

Next, a loop is formed to provide a time "window" in which a player may push the paddle button to attempt to advance the quarterback along the X axis. If the button is pressed during this period, the quarterback can be advanced. Line 8110 ensures that only one advance directive will be accepted per entry to the subroutine. Lines 8090 and 8100 keep the quarterback within the boundaries of the playing grid. This is done by repositioning the quarterback on the right-hand side of the grid whenever it advances beyond the left-hand grid line.

"TRS-80 and PET users, who do not have paddle controls on their machines, will need to modify the quarterback moving subroutine . . ."

The balance of the subroutine calls on the figure erasing and drawing routines to reposition the quarterback when a move directive is received.

TRS-80 and PET users, who do not have paddle controls on their machines, will need to modify the quarterback moving subroutine to respond to a set of keyboard characters. Select one character to move the quarterback up, another to move it down, and yet another to have it move forward. Be sure to use the GET or INKEY$ statement as the input directive rather than an INPUT statement. The latter type of statement would cause the program to wait for an input each time the subroutine was used. What you really want is to just check and see if a particular character was inputted during the period, if not, the quarterback is not moved.

Putting It All Together

Once all the various graphics subroutines have been established, it is necessary to do some "housekeeping" work to keep track of the game in progress. Lines numbered 1 through 300 in the listing are used to initialize the program at the start of the game as well as between quarters. Additionally, lines 200 and 210 change the colors of the defensemen and the quarterback whenever a team scores or a fourth down has occurred. It is not necessary to change the colors or the figures if you do not have such capability. All you really need to do is make sure that the scorekeeping changes, to keep track of the team that is "quarterbacking."

The main calling sequence for the football program starts at line 1000 in the listing. It, in turn, calls on the subroutines already mentioned to perform graphics, plus some others that are shown with line numbers in the 1400 to 2000 range. These subroutines perform such functions as keeping the scoreboard updated, offering an option to permit kicking of the ball (punting) on a fourth down, and controlling the turnover of the ball when a quarter expires.

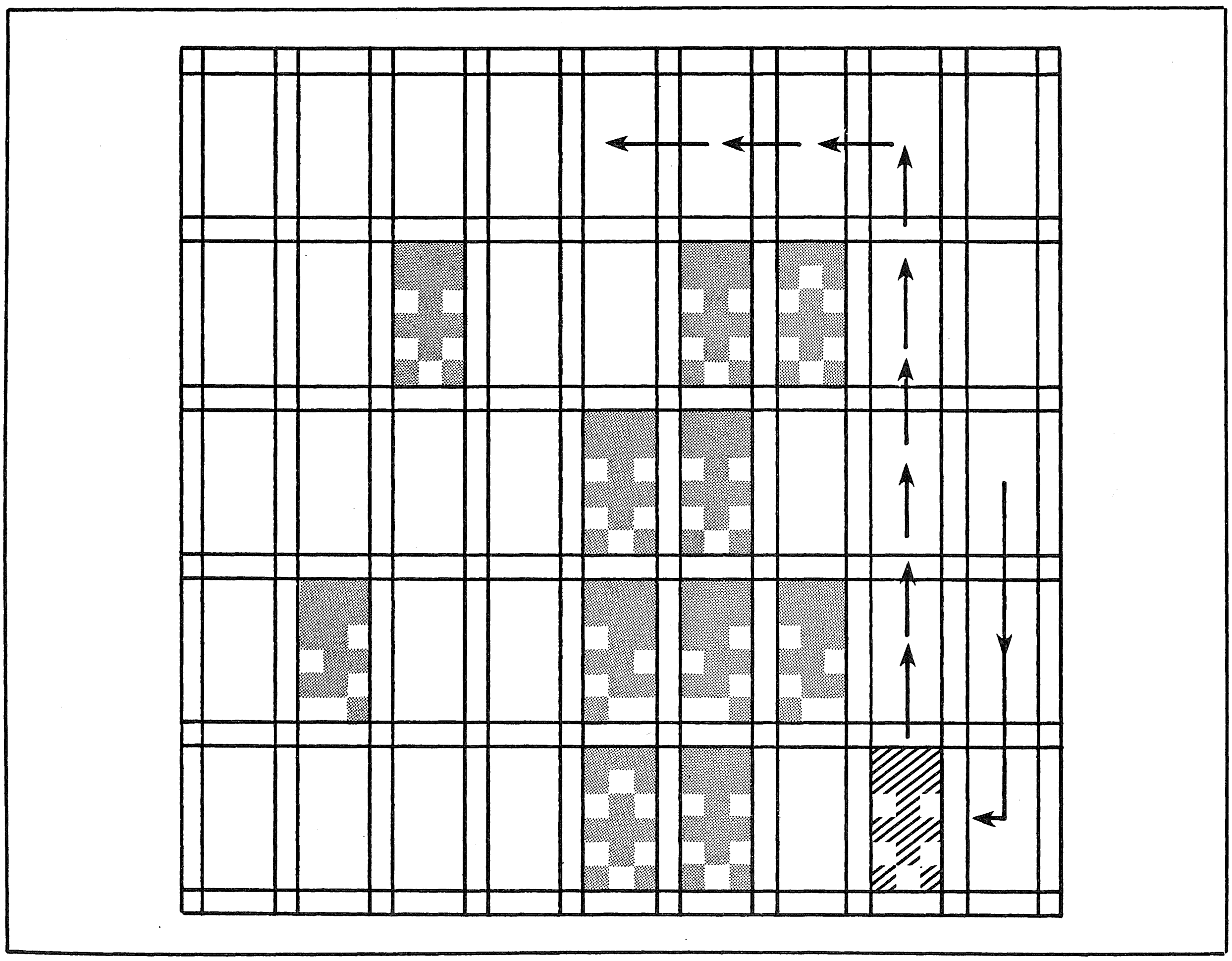
The scoreboard routines use a portion of the display screen that, on an APPLE-II, are reserved for text messages. For other types of systems you will most likely want to POKE scoreboard results beneath the playing grid. Or, you can present the scoreboard in between each new play in place of the playing grid.

Playing a Game By this time you probably have a pretty good idea of what is involved in actually playing a game of football. Ideally, two players take turns competing against one another. One captains the blue team, the other the yellow. (You can, of course, still enjoy the game by competing against yourself and playing for both teams.)

The game starts with the blue team defending. The yellow quarterback is then controlled by a player through a paddle and button (or keyboard keys in alternate versions). The object for the player in control is to advance the quarterback from the right side of the playing grid to the left while avoiding being "tackled" by a defenseman. A tackle occurs when any contact is made with the quarterback, whether as a result of the quarterback charging into a defenseman or a defenseman contacting the quarterback.

The defensemen are under computer control. They will con-

Figure 18



stantly proceed in the general direction of the quarterback. If the player controlling the quarterback does nothing, the quarterback will still be tackled. Figure 18 illustrates how the playing grid might appear when a play was in progress. The dotted arrows show a possible way for the quarterback to move in the situation shown in order to gain yards. Play ends when the quarterback is stopped because contact has been made with the defense. The number of yards gained per play is equal to the number of grid lines crossed.

After each play, the yardage required for a first down is calculated. If a first down has not been earned, the down is advanced. A player has four downs in which to gain 10 yards, thereby earning a new first down, or to score a goal.

In a fourth down situation, a player has the choice of attempting to gain yardage by running or advancing the ball by "kicking." If kicking is elected, a random number generator is used to compute a kick in the range of 1 to 55 yards. If the kick was long enough to reach the end zone, then a percentage of the kicks are scored as three-point field goals. A kick into the end zone results in the ball being brought back out to the 20-yard line (meaning the opposite team will have 80 yards to go to make a goal).

"Play always goes from right to left on the grid, regardless of which team is defending . . ."

When a score is made or four downs have been used, the opposite team defends.

Time is consumed during each play. At the end of each quarter the 900 second playing clock is reset. The yellow team always starts as quarterback to begin a game. At the start of the third quarter, the blue team starts on the offensive.

Scorekeeping is done after each play has been completed. The results are then automatically displayed on the "scoreboard." Play always goes from right to left on the grid, regardless of which team is defending, with the color of the characters being changed to show the team defending. Also note that the scoreboard always gives yardage to go in terms of the current quarterback's position relative to the goal.

To Help You Create Your Own

Remember, the object of this publication has been to give you the knowledge and insights necessary to create graphic applications on your own machine. Since each type of computer with low level resolution graphics capability is different, the actual implementation of the routines and programs I have described will generally have to be customized by *you* for your system. To help you do this for the football game described, I shall conclude this publication with a "map" of the source listing and a list of the key variable names used in the program. Try to apply what you have learned in this manual to get your own customized version of the program on line. Have fun, and happy animations!

Subroutine Map

1	Statements to initialize program at start, after quarters, after scoring, and when a first down is not achieved.
1000	Main calling sequence. Keeps track of downs. Calls on various subroutines to display graphics and conduct a game.
1400	Routine to update score and reset yard marker after a goal has been scored.
1500	Routine to kick ball if option taken on fourth down. Calls on graphics subroutines, randomly computes length of kick, and, if a field goal is made, goes to update score
1700	Routine to display message when a first down is obtained.
1800	Routine to keep track of quarters played. Resets clock for a new quarter. Changes teams at the half. Displays end of game message at conclusion of game.
1900	Routine to update and display scoreboard.
2000	Main graphics control routine. Sets defense at start of a play. Calls on defense-moves subroutine and quarterback control routine to develop play. Routine is exited when a "tackle" occurs.
3000	Graphics control routine for fourth down kick.
6000	Draws figure type 1 – front view, mouth closed.
6200	Draws figure type 2 – front view, mouth open, with sound.
6400	Draws figure type 3 – front view, running position A, with tapping sound.
6600	Draws figure type 4 – front view, running position B, with tapping sound.
6800	Draws figure type 5 – side view, kicking stance A.
7000	Draws figure type 6 – side view, kicking stance B, with sound.
7200	Draws figure type 7 – X = man tackled, with sound.
7800	Draws figure type 0 – blank to wipe out previous character displayed.
8000	Routine to move quarterback. Inputs directions from player. Keeps

quarterback within boundaries of the playing grid. Advances yardage counter if quarterback is advanced. Controls graphics to display new position. Advances timer. "Tackle" occurs if contact made with a defenseman.

9000	Routine to draw playing grid at start of each play.
9200	Draws defensemen in their initial positions at the start of play.
9500	Defensive moves routine. Selects defenseman using random functions to advance towards the quarterback. Controls movement and animation of the defensemen as they progress towards "tackling" the quarterback.

Variable Symbol	**Function**
A	Array (0 – 8 and 0 – 4) having 45 positions. An array element having the value 1 indicates a defenseman is at that position. The element is zero otherwise.
B	Temporary storage variable.
D	"Downs" counter.
F	Man tackled flag.
G	Team flag. 0 = blue defending, 1 = yellow defending.
I	Counter for temporary loops.
J	Counter for temporary loops.
M	X coordinates assigned to defense.
N	Y coordinates assigned to defense.
P	Current position marker.
Q	Yards required to obtain first down.
R	Assigned to sound generating routines.
S	Constant for sound generating routines.
S1	Blue team score.
S2	Yellow team score.
T	Time remaining in a quarter.
U	Yardage gained per play.
V	Current quarter in play.
W$	Keyboard input buffer.
X	Primary X coordinate.
X1	Temporary X coordinate.
X2	New X coordinate of quarterback.
Y	Primary Y coordinate.
Y1	Temporary Y coordinate.
Y2	New Y coordinate of quarterback.
Z	Counter for major loops.

Index

The Author

Introduction to Low Resolution Graphics is written by SCELBI's top staff author, Nat Wadsworth. He is truly a pioneer of the field having designed and developed the world's first "personal computing system," marketed by SCELBI in 1973-75. Nat wrote the Z80 Instruction Handbook, Z80 Gourmet Guide and Cookbook, and Machine Language Programming, among other books and has had articles published in the popular microcomputer magazines. One reason for his success and popularity is his unique ability to make even the most elusive concept seem simple!

Questionnaire for GRAPHICS Book

Do you have a Radio Shack TRS-80 or a Commodore PET computer? Are you unsure about creating your own version of the football game described in this book to run on your TRS-80 or PET? Well, why not first try your hand at it as a learning experience?

If you want something to compare your version with, drop this coupon in the mail. SCELBI will send you absolutely FREE* your choice of *one* of the following:

____ A.) Listing of "football" to run on a TRS-80 Level II.
____ B.) Listing of "football" to run on a Commodore PET.
____ C.) Listing of "windmill" scene in color for the APPLE-II.

(*Offer may be withdrawn without notice.)

Check your choice of *one* of the above. (Checking more than one will disqualify your request.) This coupon itself must be removed from the book and submitted to SCELBI. Photocopies of this form will *not* be accepted.

Would you like to see more instructive books similar to this one published by SCELBI? Help us determine your needs by providing the following information. It will only take a few minutes of your time to answer the questions. Your replies will help us continue to provide the kinds of publications *you* want!

Approximately how long have you been "involved" with computers?

Do you currently own a personal computer?
If so, what kind is it? (Manufacturer, CPU chip . . .)
How long have you had it?
How much memory does it have?
Do you have floppy disk capability?

Do you like to create or customize your own programs?
If so, what kind of languages do you frequently use?

Where do you use computers the most?

At home?	At school?
At work?	Other?

What types of applications do you use your computer for?

Business?

Education?

Home uses?

Recreation?

Other? (please list.)

Thank you for your assistance!

"Introduction to Low Resolution GRAPHICS"
Errata for page 67

```
9610  A(M + 1,N) = 1:A(M,N) = 0:X =
      M * 4:Y = N * 7: GOSUB 7800:
      COLOR= C:X = (M + 1) * 4: GOSUB
      9800: RETURN
9620  IF (M – 1) * 4 = X2 AND N *
      7 = Y2 THEN F = 1: RETURN
9630  IF A(M –  1,N) <> 0 THEN 9
      650
9640  A(M – 1,N) = 1:A(M,N) = 0:X =
      M * 4:Y = N * 7: GOSUB 7800:
      COLOR= C:X = (M – 1) * 4: GOSUB
      9800: RETURN
9650  IF RND (1) > 0.4 THEN RETURN
9660  IF Y2 / 7 = N THEN RETURN
9670  IF Y2 / 7 < N THEN 9710
9680  IF M * 4 = X2 AND (N + 1) *
      7 = Y2 THEN F = 1: RETURN
```

Now you can produce amazing computer graphics – even if you can't draw a straight line. Literally! Learn how to draw lines and shapes, make graphs, draw pictures and even do animation. The simple secrets of how to do all this are contained in SCELBI's new book "Introduction to Low Resolution Graphics."

Today's exciting personal and small business computing machines are generally provided with at least some kind of "low resolution" graphics capability. What is low resolution graphics? It is graphics presented on a point-by-point basis where the number of points is limited to about 8000 or less. The APPLE II by APPLE Computers, Inc., the Radio Shack TRS-80 and the Commodore PET all have low resolution graphics capability. So do many other kinds of microcomputers.

What can you do with low resolution graphics? All kinds of things. . . . If you know how! You can plot plain and simple or fancy and complex graphs to consolidate data, for business or pleasure purposes. But you can use the capability to improve the presentation and impact of almost anything you want your computer to tell people. It can be used to animate games or data, clarify and amplify educational materials, or just plain entertain people.

Introduction to Low Resolution GRAPHICS