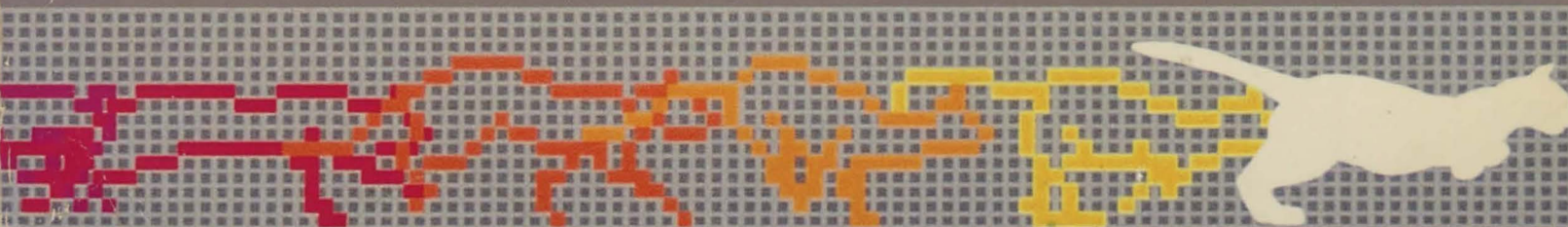
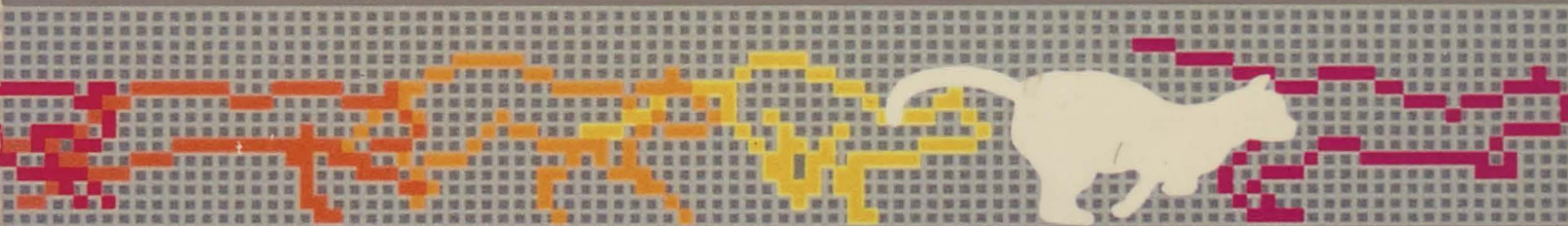
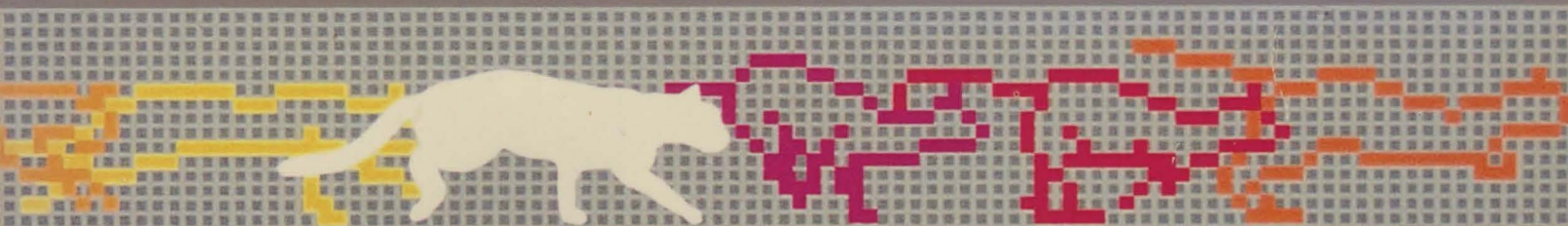
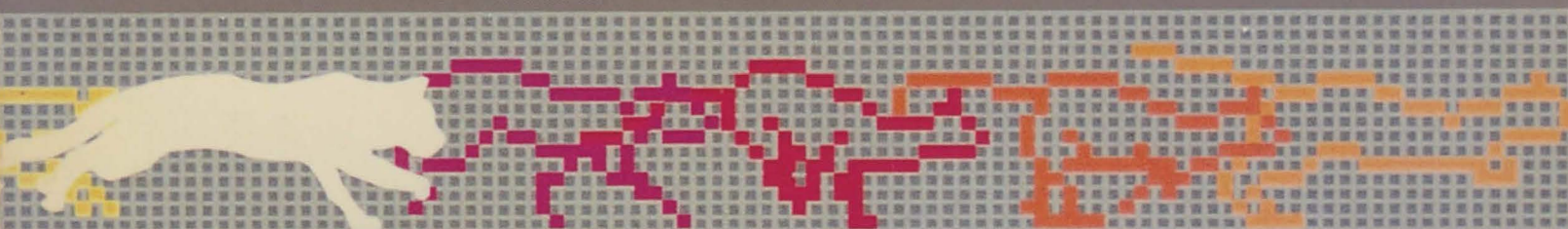


Create Your Own

# Games Computers Play

Keith Reid-Green







Michael Yurech

Create Your Own

---

# Games Computers Play

---

Kevin S. Reid-Green

UNIVERSITY  
CENTRAL PRESS





---

Create Your Own

---

# Games Computers Play

---

Keith S. Reid-Green

**digital**<sup>TM</sup>  
DIGITAL PRESS

Copyright © 1984 by Digital Equipment Corporation.

All rights reserved. Reproduction of this book, in whole or in part, is strictly prohibited. For copy information contact: Digital Press, Educational Services, 12 Crosby Drive, Bedford, Mass. 01730.

Printed in the U.S.A.  
10 9 8 7 6 5 4 3 2 1

Documentation Number: EY-00025-DP  
ISBN: 0-932376-29-0

#### Library of Congress Cataloging in Publication Data

Reid-Green, Keith.

Create your own games computers play.

1. Computer games. 2. Games—Data processing.

I. Title.

GV1469.2.R44 1984 794.8'2 83-21014

ISBN 0-932376-29-0

#### TRADEMARKS

Apple is a registered trademark of Apple Computer, Inc.

CADDs is a registered trademark of Computervision, Inc.

The DIGITAL logo, DECSYSTEM-20, and Professional are trademarks of Digital Equipment Corporation.

Othello is a trademark of Gabriel Industries, Inc.

Tektronix is a registered trademark of Tektronix, Inc.

#### CREDITS

Part 1: Courtesy the Computer Museum.

Part 2: Courtesy the Computer Museum

Part 3: Courtesy the Computer Museum

Chapter One: Arthur Grace. Stock, Boston

Chapter Two: Anestis Diakopoulos, Stock, Boston

Chapter Three: Interior of the casino at Bally's Park Place Casino Hotel

Chapter Four: © 1982 Walt Disney Productions

Chapter Five: Courtesy National Aeronautics and Space Administration

Chapter Six: Courtesy Suffolk Downs Photofinish.

Chapter Seven: The Bettmann Archive

Chapter Eight: © 1982 Walt Disney Productions

Chapter Nine: Phyllis Graber Jensen. Stock, Boston

Chapter Ten: Courtesy of Trustees of Boston Public Library

Chapter Eleven: Courtesy the Rodin Museum, Philadelphia. Gift of Jules E. Mastbaum.

Page 57: Courtesy The International Museum of Photography at George Eastman House

Page 103: Courtesy The International Museum of Photography at George Eastman House

Copyeditor: Mary Skousgaard

Proofreaders: Lynn Hamilton and Mary Skousgaard

Art: Keith Reid-Green

Designer: Diane Jaroch

Typesetter: York Graphic Services

Printer: Kingsport Press

Cover Art: Janis Keating, Computer Images, Digital Equipment Corp.



---

## **Preface**

When I ask my students in computer graphics to do a project, many of them want to develop a computer game. That's fine with me—computer games can be just as exacting as "serious" projects. The next question is usually, "Where can I find a book that tells me how to do a game?" Since I couldn't answer that question, I had to write that book myself.

Computer games are not the same as arcade games. Most arcade games are finely tuned combinations of hardware and software that run only one program. On the other hand, a personal computer is a general-purpose machine that allows you to enjoy many different programs. You can write computer games if you have enough skill, knowledge, and patience. This book may add to your skill and knowledge—the patience part is up to you.

The color photographs were taken from a Chromatics high-resolution terminal driven by a DEC-2020 computer. Line drawings throughout the book were made on a Computervision CADD54 system and output to a CalComp 960 plotter. Programs were written and tested on a Tektronix 4054 intelligent terminal. Thanks are due to Brian Astle and Dawn Kleinfeld for permission to reproduce their work, and to Stanley Reid-Green, who suggested the title. Errors in English are the responsibility of Marcia Reid-Green. Technical errors are due to Dennis Quardt. The good stuff is mine.





---

## Contents

---

### 1

Part One  
**Fundamental Computer  
Know-How**

#### **Getting Started**

<i>How to Use This Book</i>	4
<i>The Subject: Games and Sports</i>	5
<i>The Computer Display Screen</i>	5
<i>Character Graphics</i>	6
<i>Bit Mapping</i>	6
<i>Requirements for Good Programs</i>	6
<i>First Steps</i>	7
<i>Coding Conventions Used in Examples</i>	8

---

### 2

#### **Software Tool Kit**

<i>Windowing and Viewporting</i>	14
<i>Clipping</i>	16
<i>Lines</i>	16
<i>Drawing</i>	20
<i>Scaling</i>	21
<i>Translating</i>	21
<i>Rotating</i>	22
<i>Arcs</i>	22
<i>Filling</i>	23
<i>Things to Do</i>	29

---

### 3

#### **Randomness**

<i>Random Numbers</i>	34
<i>Goaltender Game</i>	36
<i>Refinements</i>	44
<i>Things to Do</i>	44

---

**4**

---

**Realism and Animation**

Realism	54
Perspective	55
Details	56
Animation	56
Target Practice Game	57
Refinements	63

---

**5**

---

*Part Two***Some Familiar Games****Ballistic Trajectory Games**

Gravity	68
Vectors	68
Dropping a Ball	69
Bouncing Ball	70
Cannon Shooting Game	73
Refinements	79

---

**6**

---

**Racetrack Games**

Designing the Track	82
Adding Horses	85
Wagering	88
Race Commentary	97
Designing Racehorses	101
Refinements	107

---

**7**

---

**Maze and Fantasy Games**

Simple Mazes	110
Fantasy Games	110
Defining a Maze Numerically	111
Generating a Maze	112
Solving or Traversing a Maze	115
How to Move the Maze Runner	125



<i>The Maze Runner's View of the Maze</i>	127
<i>Perspective Maze</i>	136
<i>Refinements</i>	143

---

**8**

---

**Outer Space Games**

<i>Newton's Laws</i>	146
<i>Space Race Game</i>	147
<i>Applying Gravity</i>	147
<i>Detecting Collisions</i>	150
<i>Controlling the Spacecraft</i>	151
<i>Refinements</i>	155

---

**9**

---

**Instructional Games**

<i>Artificial Intelligence</i>	158
<i>Concentration Game</i>	159
<i>Memory Drills</i>	164
<i>Refinements</i>	165

---

**10**

---

**Part Three**  
**Finishing Touches****Computer-Aided Design**

<i>Digitizing</i>	172
<i>Text Definition</i>	173
<i>Text Sequence</i>	174
<i>Font Table Storage and Retrieval</i>	175
<i>Horizontal Letters</i>	177
<i>Slanted Letters (Italics)</i>	178
<i>Character Rotation</i>	179
<i>Sequence Angle</i>	181
<i>Filled (Shaded) Polygons</i>	182
<i>Nontext Fonts</i>	186
<i>Hard Copies</i>	186
<i>Picture Maker</i>	187
<i>Insertions</i>	187

Deletions	189
Moves	191
Copies	191
Codes	191

## **11**

---

### ***Ideas for Hobbyist Graphics and Games Programs***

Hobbyist Programs	194
Games	196

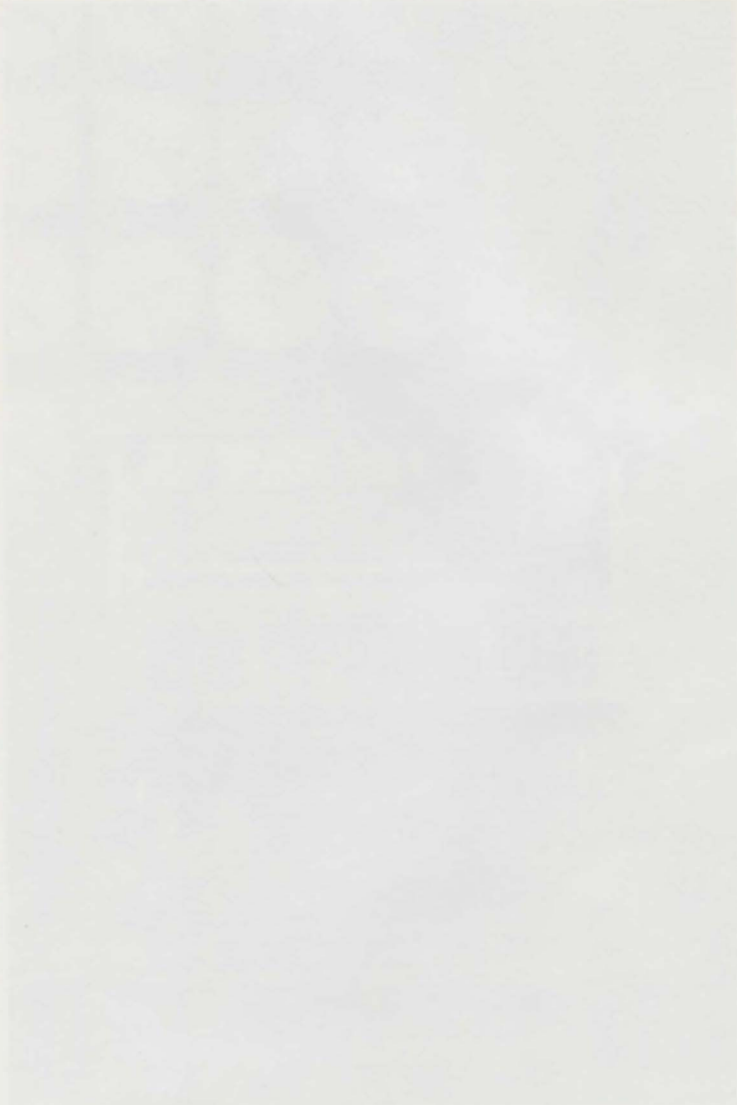
Appendix	201
Notes and Derivations	209
Bibliography	241

---

Create Your Own

---

# **Games Computers Play**

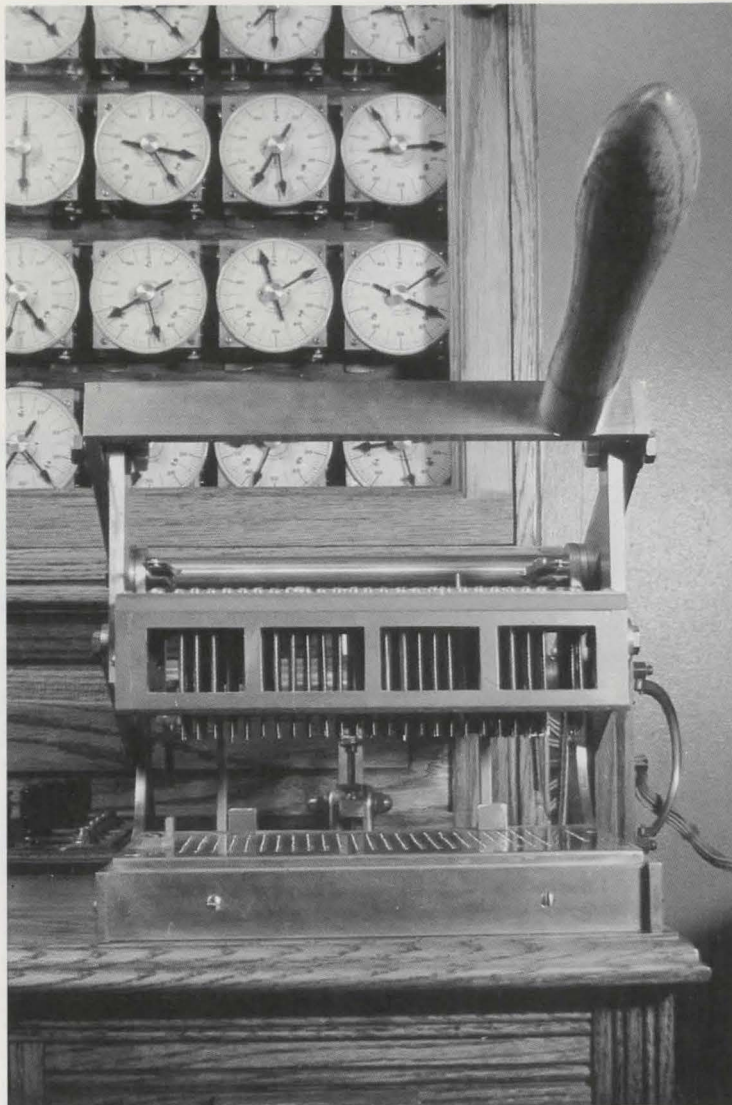






---

## ***Fundamental Computer Know-How***







---

## ***Getting Started***



If you bought *Create Your Own Games Computers Play* or got it as a gift, you probably own or use a personal computer. You are likely to have played and enjoyed arcade games but they're expensive. You may own several game disks or cassettes but you've found they're getting boring. Obviously, it's time to expand your enjoyment of computing by writing your own game programs.

The first aim of this book is to help you write a program for yourself, a game you will like and have fun with. The best way to do this is to start with something simple and then build on it as your skills increase.

Sooner or later, you'll reach the point where you want to show your work to friends. That is the time to improve the code to make it easier to use and to pay more attention to the visual effects, the graphics.

---

### **How to Use This Book**

There are three parts to this book. Part One describes the fundamentals needed for writing games. Part Two consists of several "no-frills" descriptions of game types, using concepts drawn from trigonometry, geometry, or physical laws. Finally, Part Three provides ideas for more games and a way to improve their appearance.

Please do not be in too big a rush to get started on the games themselves, because you will need the information found in the first chapters to understand the way the games are coded and described. Even if you are already a good programmer, you should look over Part One first. For instance, a description of the coding conventions used in all examples is found in Chapter 1.

If your computer doesn't have a good set of software tools for graphics, work through the *Things to Do* section in Chapter 2. You'll also find that the sub-routines throughout the book will be useful whenever you are doing a graphics application.

The games early in the book, such as Goaltender in Chapter 3, are very simple. The level of difficulty will increase later in the book as your skills have a chance to develop.

In Part Two, each chapter covers a specific type of game. Each game chapter emphasizes a different idea or technique. The chapter on *Ballistic Trajectories Games* illustrates the importance of realism while teaching the application of physical principles to computer games. *Racetrack Games* introduces polar coordinates and rudimentary animation.

The *Maze and Fantasy Games* chapter incorporates perspective projection, and *Outer Space Games* applies vectors. Finally, *Instructional Games* introduces artificial intelligence. Students who feel the need to justify writing a computer game may like the fact that each of these games allows them to learn something about mathematics, physics, or computer science.

Part Three contains ways to simplify the development of pictures and text used in games. It also has some ideas for projects that go beyond those found in this book.

Program codes have been carefully debugged but are not intended to be copied verbatim. The code is there only to support the written description of the program and to provide additional detail if necessary. You are urged to concentrate on the descriptive code when going through a program.

You may notice that the language describing the programs is a little more



formal than the rest of the book. Formal language is more precise and, therefore, less subject to misinterpretation.

Although you need some mathematics to develop almost any kind of computer game, this book does not go very deeply into theory. Since readers' backgrounds will vary, the math sections, including the *Appendix* and the *Notes*, will be of interest to some, while others will use them only for occasional reference.

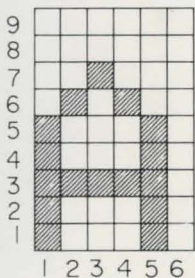
### ***The Subject: Games and Sports***

In the English language we distinguish between games and sports by classifying games as generally sedentary in nature, while sports are principally physical. Games played with equipment such as cards, dice, or a board are not actually improved by turning them into computer programs. However, computerizing them provides an opponent for a solitary player.

As a consequence of the attributes of computers, the best simulations of sporting events deal with the cerebral aspects of the sport, where the user of the computer is not acting as a player but rather, is managing the participants. A manager in a computer baseball game exercises options such as pinch hitting and having a base runner attempt to steal, but the computer controls play of the game by sampling from the statistics of the simulated players. Coordination is the principal physical skill that a computer can measure, because most peripheral devices available for playing games cannot measure strength or endurance.

Computer games as a category may consist not only of sports and games but also of hobbies and pastimes, thereby including entertainments in which the user participates only by watching. Generally speaking, such non-participatory computer games do not hold the user's interest, although it is possible to think of some that are a lot more fun than others. A computerized kaleidoscope is not nearly so intriguing as an animated cartoon in which the characters perform a randomly varied script, as in the scene from "Fencing Fools," described and illustrated in Chapter 11.

Good computer games take advantage of the machine's attributes. A computer can monitor a large set of rules at high speed, perform calculations, vary the game to fit the skill of the players, allow controlled random variation, display complex and moving images, and accept various input information. The computer is much better than people at most of these tasks, so it is also necessary to temper the computer's ability in order not to make it an overpowering opponent (See Chapter 9, *Instructional Games*).



The letter A in a  $6 \times 9$  array of pixels

### ***The Computer Display Screen***

Personal computers may be divided into two broad classes depending on the way their screens display graphic information. Both types process text in about the same way—each character is made up of a rectangular array of pixels ("pixel" is short for "picture element," the smallest spot that the computer can display on the screen). For more information, see the *Appendix, The Display Screen*.

A character in a typical computer may be made up of a  $6 \times 9$  array of pixels. The letter A could be depicted as in the adjacent illustration. Column 6 and rows 8 and 9 are blank to allow for spaces between characters and for spaces between the rows of characters.

### Character Graphics

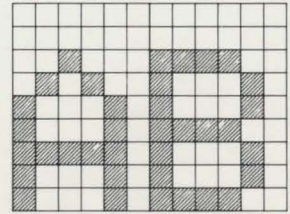
In character graphics machines, pictures are made by redefining standard characters, such as "A," "\*", "3," and so on. To display a square on a character graphics screen, a programmer might elect to redefine two characters, say A and B, by changing the pixels so that A looks like the left-hand half of a square and B looks like the right-hand half.

The command

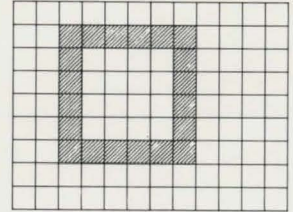
```
PRINT "AB"
```

will now display a square, not AB.

There are major restrictions to the quality of the display that can be generated on a character graphics computer. For example, if the programmer wants to rotate the square or change its size, it is necessary to redefine the characters or to have another pair of characters previously defined to look like the rotated square. Using character graphics, it is not possible to define all rotations of a square because there aren't enough characters.



*Pixel array of the letters A and B*



*Pixel array of the letters A and B redefined as a square*

### Bit Mapping

A reasonable representation of three-dimensional images requires a "bit-mapped" display with high resolution. A bit-mapped display is one in which each pixel is represented in the computer's memory. The resolution of the display is dependent on the density of pixels—the more pixels per square inch, the higher the resolution. Since each pixel requires a well-defined amount of memory, the cost of a computer is proportional to the quality of its resolution.

Because of the characteristics of the human eye, it is necessary that a computer display be rewritten or "refreshed" at least 30 times per second in order to eliminate flickering. As resolution increases, and with it the amount of memory needed, the computer must be fast enough to read out all the screen buffer, or bit map, at least 30 times per second.

The primary advantage of a bit-mapped display is that each pixel can be changed independent of any others, whereas in the character graphics display described above, pixels are changed in one-character groups of 54. The ability to change individual pixels is very important in the production of realistic graphics.

### Requirements for Good Programs

The first rule of any computer program, including a game, is that it be complete. Ignoring statistically improbable events can only lead to results that are disappointing to the user. It is unlikely, for example, that five-of-a-kind will be thrown on any roll of five dice. If, however, the programmer of a dice game omits this possibility due to laziness or ignorance, the game is incomplete. Suppose that 1296 copies of this program are sold. There is a 50 percent chance that one of those customers will discover the omission on the first roll of the dice.

Good programs are made from

- (1) A sound algorithm (definition of the program)
- (2) Completely debugged code (every step in the program has been tried out)
- (3) Thoroughly edited input (illegal or invalid data are rejected—see below)



First, a complete and accurate algorithm is essential. Without it, steps 2 and 3 won't help.

Next, debug your program. One of the best ways to debug code is to test all paths through the program so that every instruction is exercised. Then ask a friend or colleague to find ways to make the program fail.

Finally, the program should respond to any input in a nonfailing way. If a program asks a user, "What is your name?" and the user replies, "3," the computer can proceed without error by referring to the user by the name "3." However, if an income tax preparation program asks, "How many dependents?" and the user replies, "K," the program must detect that the input was not numeric and repeat or simplify the request: "How many dependents? Please enter a number not less than zero."

A good program will prompt the user—that is, explain the nature of the anticipated input: "How many dependents (0 to 20)?"

Then, after the input is accepted, it will be tested for validity and, if invalid, the request will be repeated:

```
REPEAT
  PRINT "How many dependents (0 to 20)?"
  INPUT D
UNTIL D >= 0 AND D <= 20
```

which is the same as the BASIC statements

```
100 PRINT "How many dependents (0 to 20)?"
110 INPUT D
120 IF D < 0 OR D > 20 THEN 100
```

---

### **First Steps**

It is a serious mistake to begin a computer program by writing code. The first steps should be to design the application and make notes of some kind that define the overall idea.

Computer games, for example, consist of three fundamental sections—setting the initial conditions, playing the game, and determining a winner. Initial conditions include establishing the number of players, apportioning the props among the players (13 cards each for a hand of Hearts, for instance), and initiating play.

During play, the algorithm or set of rules defining the game must be followed. When people play games, the rules are either well-known to all participants, in which case there is no question about when a rule has been broken (failure to follow suit in Hearts, for example), or it is necessary to provide impartial referees to pass judgment on events that might be evaluated differently by opposing players. In some cases referees are needed to maintain order among the players.

In any computer game, the computer must referee. Therefore, it is essential that the game algorithm cover every contingency, including cheating, and disallow any illegal move made by the players.

At the end of a game, the computer program must indicate that the game is over, who the winner is, and any other appropriate information, such as the

score, the player's won-lost record, and so on. The opportunity to play again should be presented either with the same contestants or new ones.

---

### ***Coding Conventions Used in Examples***

Most personal computers are programmed in BASIC, although there is movement toward use of languages like PASCAL that have better structure and allow the programmer to conform to modern programming practice. The programming examples in this book conform as much as possible to structured coding practice and machine independence. To avoid code that depends on the characteristics of an individual machine, some subroutines are explained but not shown in the examples. To compromise between BASIC and PASCAL, a form of structured BASIC is used in which PASCAL-like structures and top-down code are combined with standard BASIC.

Except for the four structured statements described below, the coding conforms to the rules for Tektronix BASIC. There are four graphics statements that require explanation:

**PAGE** This instruction clears the display screen and moves the cursor to the upper left-hand corner of the screen.

**HOME** The cursor is moved to the upper left-hand corner of the screen.

**MOVE** The instruction **MOVE X,Y** moves the cursor to the point X,Y.

**DRAW** The instruction **DRAW X,Y** draws a line from the existing position of the cursor to the point X,Y and moves the cursor to X,Y.

So the program

```
PAGE
MOVE 10,10
DRAW 20,10
DRAW 20,20
DRAW 10,20
DRAW 10,10
```

will erase the screen and then draw a square of 10 units on a side.

The **LET** statement may be written with or without the word **LET**.

```
LET A = 2
```

is the same as

```
A = 2
```

The variable **PI** is the same as the constant 3.14159 . . . .

Tektronix BASIC supports the operators **MAX** and **MIN**. The statement

```
A = 3 MAX B
```

will set the value of **A** as the greater of 3 and the value of **B**.

Logical variables are also supported. The statement

```
IF F THEN 300
```

will cause a branch to statement 300 if **F** is "true"—in this code, "true" means the value of **F** is not zero. The **NOT** operator may be used to invert the meaning



of a logical variable:

```
A = NOT(A)
```

sets A to 1 if A was 0; otherwise A is set to zero.

The value of a true expression is 1, of a false expression, 0. In the statement

```
A = (L < 5)
```

if L is less than 5, then A becomes 1; otherwise A becomes 0.

Logical variables can reduce the amount of code—many personal computers suffer from small memories, so code minimization is often important. Consider this problem: if A = 3, then add 2 to B. The code can be

```
100 IF A <> 3 THEN 120
110 B = B + 2
```

or

```
100 B = B + (A = 3)*2
```

The four structured statements are

```
IF condition THEN expression-list ELSE expression-list ;
WHILE condition DO expression-list ;
REPEAT expression-list UNTIL condition
CASE expression OF case: expression-list ; ...;
```

The IF statement has two forms: IF...THEN or IF...THEN...ELSE. In the first form, the code

```
100 IF A = 3 THEN
110   B = 5
120   C = 0
130 ;
```

will cause lines 110 and 120 to be executed only if A is 3 when line 100 is executed. In BASIC, these statements are converted to

```
100 REM IF A = 3 THEN
101 IF NOT(A = 3) THEN 130
110 B = 5
120 C = 0
130 REM ;
```

As an example of IF...THEN...ELSE, look at the following:

```
100 IF A = 3 THEN
110   B = 5
120   C = 0
130 ELSE
140   B = 4
150   C = -1
160 ;
```

If A = 3, B will become 5 and C, 0. If A is not equal to 3, then B becomes 4 and C, -1, because the structured statements are converted to the following BASIC code:

```
100 REM IF A = 3 THEN
101 IF NOT(A = 3) THEN 130
110 B = 5
120 C = 0
129 GO TO 160
130 REM ELSE
140 B = 4
150 C = -1
160 REM ;
```

The WHILE...DO statement will loop through all the statements below the WHILE...DO to the terminating statement (;) until the condition stated in the WHILE...DO is no longer true. The code

```
100 J = 0
110 WHILE J < 3 DO
120   PRINT J
130   J = J + 1
140 ;
```

will cause the numbers

```
0
1
2
```

to print out. This code, in BASIC, is

```
100 J = 0
110 REM WHILE J < 3 DO
111 IF NOT(J < 3) THEN 140
120 PRINT J
130 J = J + 1
139 GO TO 110
140 REM ;
```

Code for the REPEAT...UNTIL is similar, but the condition test is done at the bottom of the loop:

```
100 J = 0
110 REPEAT
120   PRINT J
130   J = J + 1
140 UNTIL J = 3
```

This code does the same thing as the WHILE...DO above. In BASIC, it becomes

```
100 J = 0
110 REM REPEAT
120 PRINT J
130 J = J + 1
139 IF NOT(J = 3) THEN 110
140 REM UNTIL J = 3
```

The CASE statement allows various pieces of code to be executed depending on the value of an expression:

```
100 CASE J - 3 OF
110   0:
```



```

120     K = 1
130     ;
140 1:
150     K = 2
160     ;
170 2:
180     K = 4
190     L = 0
200     ;
210 ;

```

If the expression  $J - 3$  is equal to 0, then statement 120 will be executed. If  $J - 3 = 1$ , statement 150 will be executed. If  $J - 3 = 2$ , statements 180 and 190 will be executed. No code will be executed between statements 100 and 210 if  $J - 3$  is equal to any other value. Note that a semicolon ends each individual case, and a terminating semicolon closes the CASE statement.

The example above converts to the following BASIC code:

```

100 REM CASE J - 3 OF
110 REM 0:
111 IF J - 3 <> 0 THEN 130
120 K = 1
130 REM ;
140 REM 1:
141 IF J - 3 <> 1 THEN 160
150 K = 2
160 REM ;
170 REM 2:
171 IF J - 3 <> 2 THEN 200
180 K = 4
190 L = 0
200 REM ;
210 REM ;

```

Complex structures can be built up using structured statements within other structures. Whenever multiple statements are terminated by semicolons, these pairings occur with the earliest semicolon applying to the latest statement. For instance:

```

100 CASE J OF
110 1:
120     IF K = 3 THEN
130         A = 5
140         B = 4
150     ;
160     PRINT B
170 ;
180 3:
190     PRINT C
200 ;
210 ;

```

Remarks may be inserted on the same line as an instruction by using '!' as a separator. Thus

```

100 READ B      ! Load the B-array

```

the same as

```
100 REM Load the B-array  
110 READ B
```

*Don't copy the programs.* They won't work on your machine because the language, although similar to ordinary BASIC, has been deliberately modified to provide structured programming capability. Use the algorithms and the code as a guide. They are not aimed at a specific type of computer. In general, it is possible to implement them on most bit-mapped personal computers, and in some cases on character machines, but some reductions are necessary for those with small memories, slow processors, or low graphics resolution.

The algorithms used throughout the book are built by degrees from the simplest examples. Users of small-memory microcomputers should develop the first algorithms and improve them as hardware permits. Owners of larger machines will find that this same stepwise refinement is a good way to write and debug programs. Although the games tend to get more complex in later chapters, the initial algorithm is as simple as possible.

If your computer is going to play games well, it will be because of you. No matter what its scale, it can do just what it is programmed to do. Likewise, the algorithm by itself is only a skeleton of an idea. You may expand any algorithm to make it more interesting and easy to use by the addition of realistic details and attention to visual effects. Then the game as you imagined it will really appear on the screen.

---

## ***Software Tool Kit***





Most computers do not have a complete set of subroutines for all purposes. While some of the most widely used routines are available (trigonometric functions like sine and cosine, for example), it is usually necessary for you to develop a series of software tools to simplify program development. Otherwise, you spend a lot of time "reinventing the wheel" whenever a new program is written.

In this chapter, we are going to develop a tool kit for graphics. You will need to know how to scale things down so they fit on your screen, develop objects from points, lines, circles, and arcs, change the sizes, shapes, and orientations of predefined objects, and fill outlines with colors. Once you have mastered these tools, you can concentrate on developing games without getting bogged down in the details of how to draw the pictures.

At the end of the chapter is a section called *Things to Do*. It describes in detail the steps you might take to debug a subroutine for drawing a picture. If you prefer not to wait, go ahead and do the examples as you go along. Be sure to convert them so they will make sense on your machine, and write them as subroutines so they can be used in more than one place in your game.

### Windowing and Viewporting

It is natural to think of events in terms of their actual measurements; for example, a horse racetrack is one mile around. To display a track on a computer, you must scale it down to fit the dimensions of the display screen. The units in which a display is generated are not miles or any other "real world" units, but pixels. As seen in the illustrations, one computer may measure pixels from the lower left-hand corner of the screen, whereas another begins in the upper left-hand corner. One screen may have 768 rows of 1024 pixels, another 190 rows of 240 pixels.

Automatic conversion from real-world units ("windowing") to pixels tends to simplify programming and debugging, because real-world numbers are used in program variables.

To convert Y-values (vertical), four numbers are involved: the pixel Y-address of the bottom left-hand corner Y1S, the pixel Y-address of the top right-hand corner Y2S, the real-world units associated with the bottom left-hand corner of the screen Y1W, and the upper right-hand corner Y2W. To find a pixel Y-address YS from an arbitrary real-world number YW, it is necessary to set up a ratio between the two scales:

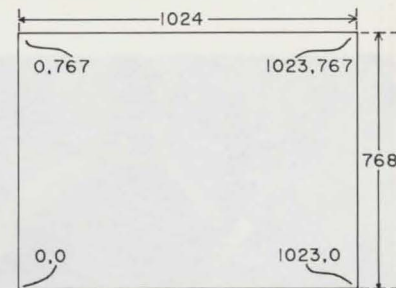
$$\frac{Y2W - Y1W}{Y2S - Y1S} = \frac{YW - Y1W}{YS - Y1S}$$

So to find YS,

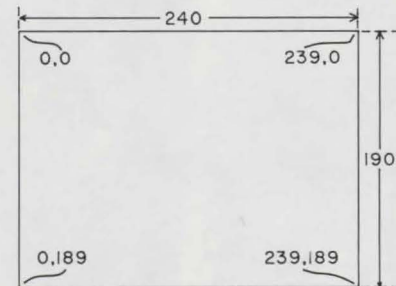
$$YS = \frac{Y2S - Y1S}{Y2W - Y1W} (YW - Y1W) + Y1S$$

The pixel X-addresses (horizontal) must be converted from real-world units in the same way. Then two functions FNX and FNY may be coded so that when an X,Y pair of real-world coordinates is given to these functions, they return a pixel address.

$$FNX(X) = \frac{X2S - X1S}{X2W - X1W} (X - X1W) + X1S$$

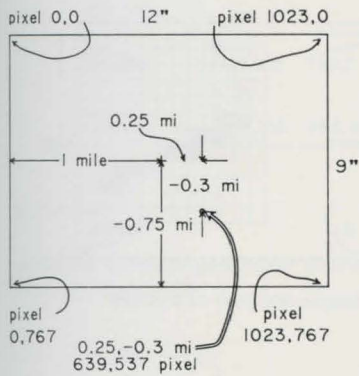


Computer screen, with 768 rows of 1024 pixels, measuring from lower left-hand corner

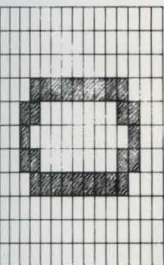
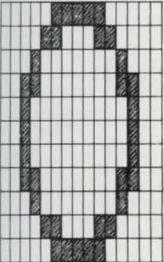
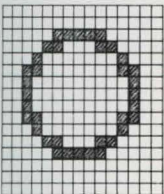


Computer screen, with 190 rows of 240 pixels, measuring from upper left-hand corner





A 9" x 12" computer screen showing pixel address of one edge of the grandstand



Aspect ratio of a circle with a radius of five pixels on a screen having square pixels. Aspect ratio of a circle with a radius of five pixels on a screen having pixels twice as tall as they are wide. Pixel aspect ratio compensated for by use of windowing.

and

$$FNY(Y) = \frac{Y2S - Y1S}{Y2W - Y1W} (Y - Y1W) + Y1S$$

For example, a racetrack is to be displayed such that the center of the oval is in the middle of the screen. Because the screen is not square and scale must be preserved, the extents in the horizontal will be different from those in the vertical.

If in the X-direction (horizontal) a distance of one mile is to be seen on either side of the center of the track, 0.75 miles will be visible above and below the center in the Y-direction. If one edge of the grandstand is 0.25 miles east and 0.3 miles south of the center of the track, what is the pixel address of this point? (See the illustration.)

If the center of the track is at 0,0 miles, the screen shows the real world from -1 to +1 miles west to east (X-direction) and -0.75 to +0.75 miles south to north (Y-direction).

Using the variable names in FNX and FNY above

Variable	Quantity	Units	Variable	Quantity	Units
X1S	0	pixels	Y1S	767	pixels
X2S	1023	pixels	Y2S	0	pixels
X1W	-1	miles	Y1W	-0.75	miles
X2W	1	miles	Y2W	0.75	miles
X	0.25	miles	Y	-0.3	miles

$$\begin{aligned} \text{so } FNX &= \frac{1023 - 0}{1 - (-1)} (0.25 - (-1)) + 0 \\ &= \frac{1023}{2} (1.25) = 639.375 \end{aligned}$$

$$\begin{aligned} \text{and } FNY &= \frac{0 - 767}{0.75 - (-0.75)} (-0.3 - (-0.75)) + 767 \\ &= \frac{-767}{1.5} (0.45) + 767 = 536.9 \end{aligned}$$

Since pixel addresses must be integers (whole numbers), it is clear that the functions should be

$$\begin{aligned} FNX(X) &= \text{INT}((X2S - X1S)/(X2W - X1W)*(X - X1W) + X1S + .5) \\ FNY(Y) &= \text{INT}((Y2S - Y1S)/(Y2W - Y1W)*(Y - Y1W) + Y1S + .5) \end{aligned}$$

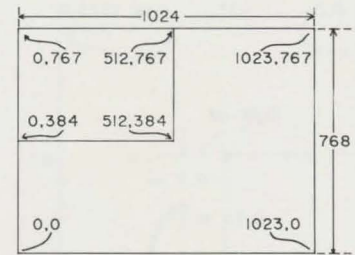
which yields the pixel address 639, 537.

There is another definite advantage to working in real-world units. On some display screens pixels are square (as in the first illustration here), but most pixels are taller than they are wide (as in the second illustration). Thus, for example, if a circle with a radius of five pixels is drawn using pixels taller than they are wide, the result will be an ellipse whose aspect ratio is the same as the aspect ratio of a pixel. Using real-world units, the circle will be circular, although nothing can be done to alter the pixel shape.

Frequently an application requires that parts of the display screen be used for different functions. One section may contain graphics, another text. Each section is referred to as a "viewport." The extents of the viewport are usually expressed in terms of pixel addresses, so four numbers are required to define a viewport.

A viewport in the upper left quarter of the screen can be described by the pixel addresses of the two opposite corners 0, 384 and 512, 767, as seen in the viewport diagram. To scale real-world units to this viewport, the variables X1S, Y1S, X2S, and Y2S become

X1S = 0      X2S = 512  
Y1S = 384    Y2S = 767



Viewporting part of a screen

### Clipping

If, as above, these screen coordinates represent  $-1$  to  $+1$  miles horizontally and  $-0.75$  to  $+0.75$  miles vertically, it is possible to express values such as  $2, -1$  that are outside the viewport but on the screen. If a line is drawn from  $0, 0$  to  $2, -1$ , it will exceed the boundaries of the viewport and should be "clipped"—only the part inside the viewport should be displayed.

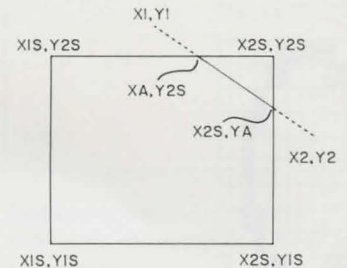
The test to see if an individual pixel  $x, y$  is outside the viewport is

IF  $x < X1S$  or  $x > X2S$  or  $y < Y1S$  or  $y > Y2S$  THEN outside

Simple. All that is necessary is to display only those pixels that fail the "outside" test. However, clipping every pixel is too slow, because the conversion from world units is done even on pixels that are to be clipped. It is much faster to clip a line in its world coordinates and to convert to pixels only the portion within the viewport.

All lines fall into one of three categories: entirely within the viewport, entirely outside, or partly within. Notice in the clipping illustration that a line is not necessarily entirely outside the viewport merely because its endpoints are outside the viewport.

The visible portion of this line is the segment from  $XA, Y2S$  to  $X2S, YA$ , where  $XA$  and  $YA$  are as of yet unknown. All lines may be clipped by examination with the four lines defining the viewport and by clipping off any parts that pass through each of the four lines. This is done by replacing, in the above case, the point  $X1, Y1$  with  $XA, Y2S$  and  $X2, Y2$  with  $X2S, YA$ . (The process for determining the intersection of two lines is discussed in the *Cartesian Coordinates* section of the Appendix.)



Clipping endpoints outside the viewport

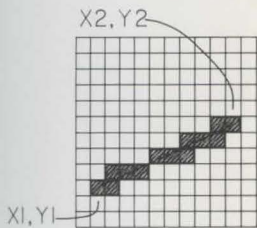
### Lines

Most personal computers allow the user to draw a line on the display screen by specifying the endpoints of the line. For example, to draw a line from  $X1, Y1$  to  $X2, Y2$  it may be necessary to write

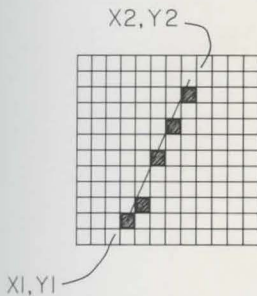
```
MOVE X1, Y1
DRAW X2, Y2
```

or simply

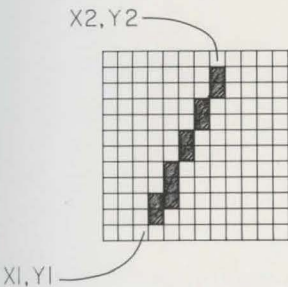




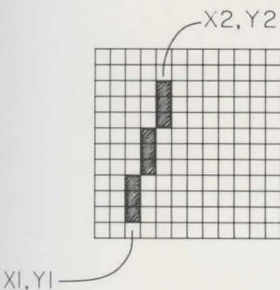
Automatic computer generation of line from  $X_1, Y_1$  to  $X_2, Y_2$



Beginning of slow line-drawing algorithm



Completed slow line-drawing algorithm



Faster line-drawing algorithm

DRAW  $X_1, Y_1$  TO  $X_2, Y_2$

The computer automatically finds which pixels lie on the theoretical line and illuminates them, as shown here. However, software that generates lines automatically is not always available, especially for microprocessors or very inexpensive computers.

All display processes are ultimately reduced to illuminating individual pixels, including the drawing of a straight line. As we shall see, parametric curves such as circles and ellipses are usually reduced to short straight lines, which means that a lot of lines will be converted to pixels during the generation of a display. Consequently, the conversion should be fast.

Suppose a line is to be drawn between two world-coordinate points  $V, W$  and  $X, Y$ . After windowing and clipping to the viewport boundaries, these points are converted to the pixel addresses  $X_1, Y_1$  and  $X_2, Y_2$ . The accompanying drawings illustrate the following discussion.

A valid algorithm to find the pixels on a nonvertical line consists of illuminating the pixel at the intersection of the line with each of the vertical lines that pass through pixels between  $X_1$  and  $X_2$ .

To complete the algorithm, if any pair of consecutive pixels is not connected, it is filled (illuminated) vertically. This is a slow algorithm because the equation for the intersection of two lines must be solved for each pixel between  $X_1$  and  $X_2$ .

A much better algorithm uses the ratio of horizontal to vertical distance between the endpoints. If  $Y_2 - Y_1$  is three times as big as  $X_2 - X_1$ , then it is clear that for every horizontal step, three vertical steps must be filled.

Here is the code for this faster line-drawing algorithm:

### Line Program Code

	Sequence Number
$X = X_1$	330
$Y = Y_1$	340
GOSUB 500	350
$F = 0$	360
REPEAT	370
$F1 = F + \text{ABS}(X_2 - X_1)$	380
$F2 = F - \text{ABS}(Y_2 - Y_1)$	390
IF $\text{ABS}(F1) \geq \text{ABS}(F2)$ THEN	400
$F = F2$	410
$X = X + \text{SGN}(X_2 - X_1)$	420
ELSE	430
$F = F1$	440
$Y = Y + \text{SGN}(Y_2 - Y_1)$	450
;	460
GOSUB 500	470
UNTIL $X = X_2$ AND $Y = Y_2$	480

Compare the algorithmic description below with the code above by matching the sequence numbers. Notice the simplicity of the code—no multiplication or division is necessary and only the very fast ABS and SGN functions are used. In

fact,  $ABS(X2 - X1)$  and  $SGN(X2 - X1)$  might be assigned to variables before the REPEAT (the same could be done for  $Y2 - Y1$ ), thereby further reducing the necessary computations.

<b>Line Code Description</b>	<b>Sequence Number</b>
X and Y are set to the beginning point on the line X1,Y1.	330
Subroutine 500 fills the pixel whose address is X,Y. F is used to establish the ratio of the number of address increments in X with those in Y.	330
F1 is set to F plus the horizontal distance between the endpoints of the line.	380
F2 is set to F minus the vertical distance between the endpoints of the line.	390
If the X-ratio is currently not less than the Y-ratio, F is set to the Y-ratio and a step of -1, 0, or +1 is added to the X-address, depending on whether $X2 < X1$ , $X2 = X1$ , or $X2 > X1$ , respectively.	400
If the X-ratio is currently less than the Y-ratio, F is set to the X-ratio and a step of -1, 0, or +1 is added to the Y-address, depending on whether $Y2 < Y1$ , $Y2 = Y1$ , or $Y2 > Y1$ , respectively.	430
The pixel whose address is X,Y is filled.	470
The process is continued until the end of the line is reached.	480

An elegant but not very fast line drawing algorithm may be defined recursively. (A recursive procedure is one that calls itself, similar to a BASIC subroutine beginning at statement 100 that contains a GOSUB 100 within the subroutine.)

Suppose a green line is to be drawn from X1,Y1 to X2,Y2. A recursive line-drawing algorithm is

```

LINE(X1,Y1,X2,Y2):
  IF X1 = X2 AND Y1 = Y2 THEN
    PIXEL(X1,Y1) = GREEN
    RETURN
  ELSE
    LINE(X1,Y1,(X2 + X1)/2,(Y2 + Y1)/2)
    LINE((X2 + X1)/2,(Y2 + Y1)/2,X2,Y2)
  ;

```

This procedure is called LINE and requires as inputs the pixel addresses of the endpoints. If the endpoints of the line are on the same pixel, color it green and end the procedure. Otherwise, call the procedure with the endpoints of the two halves of the line.

A programmer using BASIC cannot normally use recursion but instead must write a program that builds a stack of the pertinent data (in this case the endpoints of the line) and repeatedly refers to the stack until it is empty.

It is necessary to establish how big the stack must be. Since the stack is increased each time the line is halved, the stack depth depends on the number of times the longest possible line might be halved before the endpoints lie on the same pixel. This in turn depends on the size of the display. If the screen contains 1024 by 1024 pixels, it is possible to halve 1024 to get 512, halve 512 to get 256, and so on, 10 times before reaching 1. In other words,  $1024 = 2^{10}$ . The stack must be ten deep, and since the values X1,Y1,X2,Y2 are to be retained, the stack array S can be defined as S(10,4).



Because of roundoff and the halving process, extra points will be generated. They can be omitted by comparing a generated point with the previously displayed point and displaying it only if their pixel addresses (X or Y depending on the slope of the line) differ. Study the recursive line-drawing code and then its explanation:

### Recursive Line Program Code

	Sequence Number
P = 1	290
F = 0	300
X4 = -999	310
Y4 = -999	320
S(1,1) = X1	330
S(1,2) = Y1	340
S(1,3) = X2	350
S(1,4) = Y2	360
G = 0	370
IF ABS(X2 - X1) < ABS(Y2 - Y1) THEN	371
G = 1	372
;	373
REPEAT	380
V = S(P,1)	390
W = S(P,2)	400
X = S(P,3)	410
Y = S(P,4)	420
P = P - 1	430
IF ABS(V - X) MAX ABS(W - Y) < 0.5 THEN	440
X3 = INT(V)	450
Y3 = INT(W)	460
IF F = 0 OR NOT(X3 = X4 AND NOT(G) OR (Y3 = Y4 AND G)) THEN	470
F = 1	480
X4 = X3	490
Y4 = Y3	500
GOSUB 660	510
;	520
ELSE	530
P = P + 2	540
S(P,1) = V	550
S(P,2) = W	560
S(P,3) = (V + X)/2	570
S(P,4) = (W + Y)/2	580
S(P - 1,1) = S(P,3)	590
S(P - 1,2) = S(P,4)	600
S(P - 1,3) = X	610
S(P - 1,4) = Y	620
;	630
UNTIL P = 0	640
!	Keep going until the stack is empty

### Recursive Line Code Description

	Sequence Number
Set stack counter P to 1.	290
Set "pixel processed" flag F to 0.	300
Set X4,Y4, the address of the last pixel processed, to an impossible ad-	310

dress. This is not necessary in most versions of BASIC—however, in line 470 a comparison with X3 and Y3 is made. Even though the first comparison is not meaningful since  $F = 0$ , in one or two BASIC interpreters the values for X4 and Y4 must have been preset in order to avoid an 'undefined' error. (In other BASICs, an undefined variable is presumed equal to zero.)

Put the endpoints of the line on the stack.	330
Set G to reflect the slope of the line. If $Y2 - Y1$ is greater in magnitude than $X2 - X1$ , then $G = 1$ ; otherwise $G = 0$ .	370
Set V,W,X,Y to the most recently stacked endpoints.	390
Reduce the stack counter by 1.	430
If the endpoints are at the same pixel address, set X3,Y3 to that address.	440
If either no pixel has been filled or this point has not been filled previously, set the "pixel processed" flag to 1, set X4,Y4 to the address of the last pixel processed, and fill that pixel (subroutine 660).	470
If the test in line 470 failed, halve the line and put the resulting two sets of endpoints on the stack. Add 2 to the stack counter.	530
Continue until the stack is empty.	640

## Drawing

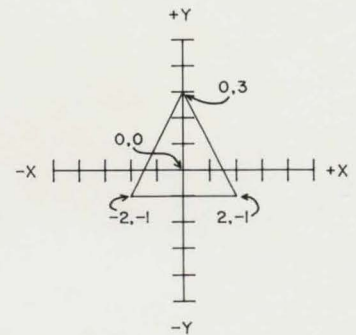
Let us define "drawing" as beginning at a point  $X0,Y0$ , making a straight line from  $X0,Y0$  to  $X1,Y1$ , making another straight line from  $X1,Y1$  to  $X2,Y2$ , and so on, until a specified number of X,Y pairs has been used.

For instance, to define a triangle, four pairs of points are needed:  $-2,-1$   $2,-1$   $0,3$  and  $-2,-1$  again, as illustrated in the drawing. If these points are put into a DATA statement after the number 4 to define how many pairs there are, a simple routine to draw the figure can be constructed.

```
DATA 4,-2,-1,2,-1,0,3,-2,-1
READ J          ! Read '4'
READ X8,Y8      ! Read -2,-1
MOVE X8,Y8      ! Move to -2,-1
WHILE J >= 2 DO ! Repeat thru next ; until J < 2
  READ X8,Y8    ! Read next point
  DRAW X8,Y8    ! Draw from previous point to this point
  J = J - 1     ! Reduce J
;
```

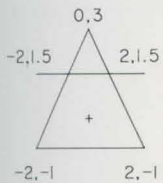
If more than one draw sequence is necessary, the program must recognize the end of each group of lines. Let us say that any number less than -999 or less signals "quit" when read into J. Thus

```
DATA 4,-2,-1,2,-1,0,3,-2,-1
DATA -9999
REPEAT
  READ J
  IF J > -999 THEN
    READ X8,Y8
    MOVE X8,Y8
    WHILE J >= 2 DO
```



Four pairs of points needed to define a triangle:  $-2, -1$   $2, -1$   $0, 3$   $-2, -1$





*Drawing needing more than one draw sequence*

```

READ X8,Y8
DRAW X8,Y8
J = J - 1
;
UNTIL J < -999

```

These data statements describe the accompanying figure, which has two draw sequences:

```

DATA 4,-2,-1,2,-1,0,3,-2,-1
DATA 2,-2,1.5,2,1.5
DATA -9999

```

### Scaling

Having defined a figure's dimensions, it may be necessary to redraw it in a different size. This process, scaling, is done by multiplying each X,Y pair by a scaling factor S9. If S9 = 1, the figure will remain the same. A factor of 0.5 will reduce each dimension by half. For instance, a figure 4 units wide by 4 units tall will become, after a scaling factor of 0.5, 2 units wide by 2 units tall.

To allow for scaling, the program becomes

```

REPEAT
  READ J
  IF J > -999 THEN
    READ X8,Y8
    MOVE X8*S9,Y8*S9
    WHILE J >= 2 DO
      READ X8,Y8
      DRAW X8*S9,Y8*S9
      J = J - 1
    ;
  ;
UNTIL J < -999

```

Scaling can cause an unexpected side effect if a figure is not defined around the origin (the point 0,0). For example, in the accompanying illustrations, a triangle is shown before and after scaling by a factor of 0.5.

Not only have the triangle's dimensions halved, but so has its distance from the origin.



*Triangle before scaling*



*Triangle after scaling by a factor of 0.5*

### Translating

A set of X,Y pairs may be moved so that the origin of the set (the pair 0,0) moves to the point X9,Y9 by adding X9 to each of the X-values and Y9 to each of the Y-values in the set. This must be done after scaling otherwise X9 and Y9 will also be scaled. The above MOVE and DRAW commands become

```

MOVE X8*S9 + X9,Y8*S9 + Y9
DRAW X8*S9 + X9,Y8*S9 + Y9

```



## Rotating

A set of X,Y pairs may be rotated around the local origin by computing a rotated pair X7,Y7 as follows:

$$X7 = X8 \cdot \cos(\theta) - Y8 \cdot \sin(\theta)$$

$$Y7 = X8 \cdot \sin(\theta) + Y8 \cdot \cos(\theta)$$

where  $\theta$  is the counterclockwise angle of rotation in radians. (If T is the angle in degrees,  $\theta = T \cdot 0.0174533$ .) Rotation may be done before or after scaling but must be done before translation because rotation is always about the origin.

A program to:

Rotate through T degrees

Scale by S9

Translate to X9,Y9

Draw a figure

using the data statements defined above, is

```
CO = COS(T*0.0174533)
SO = SIN(T*0.0174533)
REPEAT
  READ J
  IF J > -999 THEN
    READ X8,Y8
    MOVE (X8*CO - Y8*SO)*S9 + X9, (X8*SO + Y8*CO)*S9 + Y9
    WHILE J >= 2 DO
      READ X8,Y8
      DRAW (X8*CO - Y8*SO)*S9 + X9, (X8*SO + Y8*CO)*S9 + Y9
      J = J - 1
    ;
  ;
UNTIL J < -999
```

## Arcs

Although many figures may be drawn using only straight lines, the quality of most figures is improved by the addition of circular or elliptical arcs. The general elliptical arc may be described fully by the coordinates of its center, its horizontal and vertical radii, and the starting and ending angles of the arc as measured from the center. Similarly a circular arc may be described, where the radius of the circle is used in place of the horizontal and vertical radii of the ellipse.

To draw the arc, first set

```
X0,Y0 = center of arc
M1 = horizontal axis length
M2 = vertical axis length
A1 = starting angle, converted to radians
A2 = ending angle, converted to radians
```

The cursor is moved to the first point on the elliptical arc, at

$$X0 + M1 \cdot \cos(I), Y0 + M2 \cdot \sin(I)$$

and then short line segments are drawn in one-degree increments between A1 and A2. The program must therefore increase or decrease the angle by one-degree increments depending on whether A1 is less than or greater than A2.

The program code and description for drawing an ellipse are as follows:

### ***Ellipse Program Code***

	Sequence Number
I = A1	150
MOVE X0 + M1*COS(I),Y0 + M2*SIN(I)	160
IF A1 < A2 THEN	170
REPEAT	180
I = I + PI/180	190
IF I > A2 THEN	200
I = A2	210
;	220
DRAW X0+M1*COS(I),Y0+M2*SIN(I)	230
UNTIL I = A2	240
ELSE	250
REPEAT	260
I = I - PI/180	270
IF I < A2 THEN	280
I = A2	290
;	300
DRAW X0 + M1*COS(I),Y0 + M2*SIN(I)	310
UNTIL I = A2	320
;	330

### ***Ellipse Code Description***

	Sequence Number
Having set X0, Y0, M1, M2, A1, and A2, set I equal to A1.	150
Move the cursor to the start of the elliptical arc.	160
If A1 is less than A2, execute the code between lines 180 and 240.	170
Increase I by $\pi/180$ radians, or 1 degree.	190
If I is greater than A2, set I equal to A2.	200
Draw this arc segment.	230
If A1 was greater than A2 in line 170, execute the code between lines 260 and 330.	250
Decrease I by $\pi/180$ radians, or 1 degree.	270
If I is less than A2, set I equal to A2.	280
Draw this arc segment.	310
If I is not equal to A2, continue at line 260.	320

### ***Filling***

Having developed the capability to draw outlines of figures, the next step is to color segments of figures in order to add realism. It is relatively easy to color rectangles—given the opposite corners X1,Y1 and X2,Y2 an algorithm is:



```

I = Y1
REPEAT
  I = I + 0.01*SGN(Y2 - Y1)
  MOVE X1,Y1 + I
  DRAW X2,Y1 + I
UNTIL I >= Y2

```

A better one would replace 0.01 with a number which guaranteed that exactly one pixel row at a time would be filled by the MOVE-DRAW commands. To fill a circle or circular arc, a similar procedure can be followed by drawing concentric circles, reducing the radius by approximately one pixel each time.

Irregular polygons are more difficult to fill. One possible procedure entails examining a horizontal line that passes through the polygon. When the line reaches the first line on the polygon, start filling. Continue to fill until the second line on the polygon is encountered, start again on the third line, and so forth.

If this is done for every pixel row between the biggest and smallest Y-values, the polygon will be filled. The algorithm requires a table of the nodes (end-points) of each line, sorted to simplify determining whether a pixel is inside or outside the polygon.

To set the window for your machine, use the windowing functions described in the Windowing and Viewpointing section of this chapter, with the numerical values required by your computer. Then change all MOVE and DRAW commands to use these functions. For example,

```
MOVE FNX(X(1)),FNY(Y(1))
```

260

and so forth.

In the following code for filling a polygon, the screen has 100 rows of 130 pixels:

### **Shade a Polygon Program Code**

	Sequence Number
DATA 6,65,50,85,45,80,20,70,30,60,20,50,40 ! Define polygon	130
READ N ! Read number of nodes	140
DELETE X,Y,T ! Define arrays to contain N nodes	150
DIM X(N + 1),Y(N + 1),T(N,4)	160
Y1 = 9999 ! Initialize	170
Y2 = -9999	180
I = 0	190
REPEAT	200
I = I + 1	210
READ X(I),Y(I) ! Get a node	220
Y1 = Y1 MIN Y(I) ! Find the smallest Y	230
Y2 = Y2 MAX Y(I) ! and the largest Y so far	240
IF I = 1 THEN	250
MOVE X(1),Y(1) ! Move to first node	260
ELSE	270
DRAW X(I),Y(I) ! draw to this node	280
;	290
UNTIL I = N	300
X(N + 1) = X(1) ! Duplicate first node	310
Y(N + 1) = Y(1) ! as last node to close polygon	320



```

DRAW X(1),Y(1)          ! and draw closure          330
I = 0                    340
REPEAT                    350
  I = I + 1              360
  IF X(I)<=X(I+1) THEN ! Save each line with smallest x first 362
    T(I,1) = X(I)        364
    T(I,2) = Y(I)        366
    T(I,3) = X(I + 1)    368
    T(I,4) = Y(I + 1)    370
  ELSE                    372
    T(I,1) = X(I + 1)    374
    T(I,2) = Y(I + 1)    376
    T(I,3) = X(I)        378
    T(I,4) = Y(I)        380
  ;                        382
UNTIL I = N              410
REPEAT                    420
  C = 0                  ! Sort T on smaller X of each line 430
  I = 1                  440
  REPEAT                  450
    I = I + 1            460
    IF T(I - 1,1) > T(I,1) THEN ! If a pair of lines is 470
      S = 0              ! out of sequence, 480
      REPEAT              490
        S = S + 1        ! switch the pair of lines 500
        C = T(I - 1,S)  510
        T(I - 1,S) = T(I,S) 520
        T(I,S) = C      530
      UNTIL S = 4        540
    ;                    550
  UNTIL I = N            ! Test all pairs 560
UNTIL C = 0              ! Do it again if a pair was switched 570
S = Y1                   ! Begin at Ymin 580
REPEAT                    590
  S = S + 0.233          ! Move up 0.233 units 600
  C = 0                  610
  I = 0                  620
  REPEAT                  630
    I = I + 1            640
    IF S >= T(I,2) MIN T(I,4) AND S <= T(I,2) MAX T(I,4) THEN 650
      X1 = T(I,1)        660
      IF T(I,1) <> T(I,3) THEN 670
        X1=(S-T(I,2))/(T(I,4)-T(I,2))*(T(I,3)-T(I,1))+T(I,1) 680
      ;                  690
    IF C=0 THEN          ! If horizontal line crosses a polygon line, 700

```

```

      MOVE X1,S      ! move or draw                      710
    ELSE            ! depending on whether inside or outside 720
      DRAW X1,S      ! the polygon                      730
    ;                                                       740
    C = NOT(C)       ! Change "lines crossed" flag        750
  ;                                                         760
  UNTIL I = N        ! Process all polygon lines          770
UNTIL S >= Y2        ! End when at Ymax                  780

```

### Shade A Polygon Code Description

Sequence  
Number

On a screen defined as 0 to 130 in X and 0 to 100 in Y, define 6 nodes at 65,50 85,45 80,20 70,30 60,20 and 50,40.

Read the number of nodes. 140

Define arrays X and Y to hold the coordinates of all the nodes, repeating the first node as the last node to close the polygon. Define array T to hold the X,Y values of the endpoints of each line. 160

Set Y1, which will become the smallest Y-value Ymin, to a large number. Set Y2, which will become the largest Y-value Ymax, to a small number. 170

Put the coordinates of each node into the X and Y arrays and collect Ymin and Ymax. Draw the polygon. 200

Put the coordinates of the first node into the (N + 1)st node and draw to the first node to close the polygon. 310

Put pairs of nodes into T such that the node with the smallest X-value is entered first. 350

Sort the T array so that the line with the smallest X-value is first, and the other lines follow in sequence by ascending X. 420

Move up the polygon from Ymin to Ymax in steps of 0.233. 580

Set C = 0 to indicate that an even number of polygon lines have been looked at. 610

If the horizontal line intersects a polygon line, then compute X1, the X-intersection of the lines. 650

If this line is an even count (C = 0), then move to the intersection. 700

If it is an odd count, draw to the intersection, filling from the previous intersection. 720

Change C to reflect that another line has been counted. 750

Process all polygon lines. 770

Process until S has reached Ymax. 780

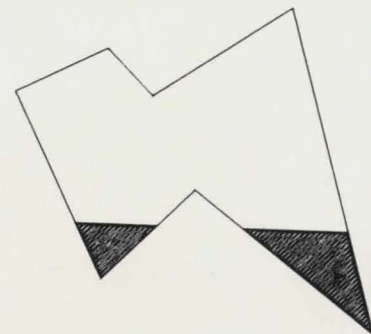
The above code will result in a polygon shaded as illustrated here.

If the programmer has access to the display memory (that is, access to pixel addresses), an easy way to implement a filling algorithm is to define it recursively. Given an arbitrary polygon drawn in white on a black background, a pixel address X,Y known to be inside the polygon and the polygon to be colored red, a recursive algorithm is

```

FILL(X,Y):
  IF PIXEL(X,Y) = BLACK THEN

```



Polygon in the process of being shaded

```

RETURN
ELSE PIXEL(X,Y) = RED
FILL(X - 1,Y)
FILL(X + 1,Y)
FILL(X,Y - 1)
FILL(X,Y + 1)
;

```

This procedure is called FILL. If the pixel being examined is black, quit. Otherwise, color it red and examine the pixels above, below, to the left, and to the right of the pixel at X,Y.

### Fill Program Code

	Sequence Number
REPEAT	520
IF P(X,Y) <> 0 THEN	! If the pixel at X,Y is not 0, 530
X = INT(S(C)/1000)	! get another pixel address 540
Y = S(C) - X*1000	! from the stack 550
C = C - 1	! and reduce the stack counter 560
ELSE	! If the pixel at X,Y is = 0, 570
P(X,Y) = 4	! set it to 4, 580
GOSUB 880	! process the color change, 590
S(C+1)=(X-1)*1000+Y	! and stack the addresses of 620
S(C+2)=(X+1)*1000+Y	! the four pixels above, below, 660
S(C + 3) = X*1000 + Y - 1	! left, and right of X,Y 700
S(C + 4) = X*1000 + Y + 1	740
C = C + 4	! increase the stack counter 750
;	760
UNTIL C = 0	! Repeat until the stack is empty 770

### Fill Code Description

	Sequence Number
If the pixel at X,Y is not white, get an address from the stack and reduce the stack counter by 1.	530
If P(X,Y) is white, set it to red. Depending on hardware characteristics, it may be necessary to redraw the screen explicitly. This is done by Subroutine 880.	570
Put the address X - 1,Y on the stack.	620
Put the address X + 1,Y on the stack.	660
Put the address X,Y - 1 on the stack.	700
Put the address X,Y + 1 on the stack.	740
Add 4 to the stack counter.	750
If the stack is not empty, continue.	770

Pixels are accessed from an array P, and X and Y are set to a pixel address inside a previously drawn polygon. An array S that will serve as a stack has been defined. The stack counter C has been set to zero. In P, the number 0 indicates that the pixel is colored white, 7 is for black, and 4, red. To conserve memory, X and Y will be saved on the stack in the form  $X*1000 + Y$ . (If the number of pixels



in the Y-direction is more than 1000, an appropriately larger multiplier must be chosen.)

This program will fill the polygon shown here, beginning at  $P(X,Y)$ , but will not fill the central island because the black pixels will act as a barrier in the same way as the perimeter of the polygon.

Given this polygon, the stack for this algorithm has to hold as many as 308 addresses at once. Since there are only 244 pixels to be filled, there is obviously something very inefficient about the algorithm.

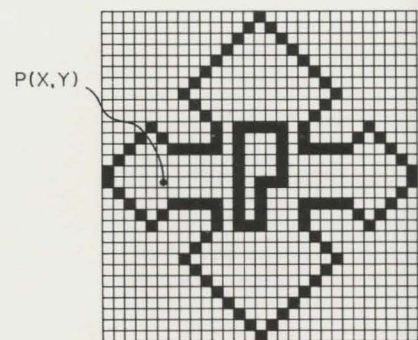
Look at the sequence of three illustrations of pixels being filled. Whenever a pixel  $P(X,Y)$  is found to be white, it is colored red and the four pixels next to its edges are put on the stack, as shown in the first drawing.

In the second drawing, two pixels have been colored red and there are seven pixels on the stack, including a red one. The last pixel address put on the stack is the first to be removed for processing. Therefore, the seventh pixel stacked is now removed from the stack and processed, which causes another red pixel to be put on the stack (see third drawing). Even though red pixels from the stack are not reprocessed (since they are not white), time and memory are wasted. An obvious refinement is, therefore, not to stack red pixels. This improvement reduces the stack size to a maximum of 132 pixels.

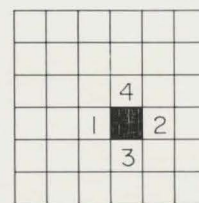
Refer to the partially shaded polygon while studying the following code and description for recursive filling.

### Recursive Fill Program Code

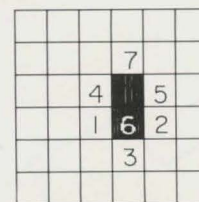
	Sequence Number
REPEAT	520
IF $P(X,Y) <> 0$ THEN	530
$X = \text{INT}(S(C)/1000)$	540
$Y = S(C) - X*1000$	550
$C = C - 1$	560
ELSE	570
$P(X,Y) = 4$	580
GOSUB 880	590
IF $P(X-1,Y) = 0$ THEN      ! Only stack the pixel to the left	600
$C = C + 1$ ! of X,Y if it is white	610
$S(C) = (X-1)*1000 + Y$	620
;	630
IF $P(X+1,Y)=0$ THEN              ! Only stack the pixel to the right	640
$C = C + 1$ ! of X,Y if it is white	650
$S(C) = (X+1)*1000 + Y$	660
;	670
IF $P(X,Y-1) = 0$ THEN              ! Only stack the pixel below	680
$C = C + 1$ ! X,Y if it is white	690
$S(C) = X*1000 + Y - 1$	700
;	710
IF $P(X,Y+1) = 0$ THEN              ! Only stack the pixel above	720
$C = C + 1$ ! X,Y if it is white	730
$S(C) = X*1000 + Y + 1$	740
;	750
;	760
UNTIL $C = 0$	770



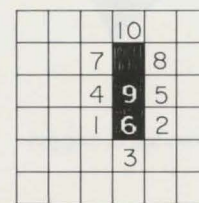
A polygon to be shaded by the recursive fill algorithm



Shading first pixel



Shading second pixel



Shading third pixel

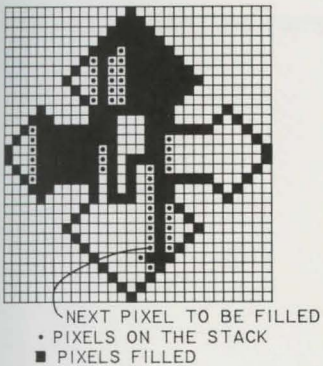
The improved recursive procedure is

```
FILL(X,Y):
  IF PIXEL(X,Y) = WHITE THEN
    RETURN
  ELSE PIXEL(X,Y) = RED
    IF PIXEL(X - 1,Y) = WHITE THEN
      FILL(X - 1,Y)
    ;
    IF PIXEL(X + 1,Y) = WHITE THEN
      FILL(X + 1,Y)
    ;
    IF PIXEL(X,Y - 1) = WHITE THEN
      FILL(X,Y - 1)
    ;
    IF PIXEL(X,Y + 1) = WHITE THEN
      FILL(X,Y + 1)
    ;
  ;
```

And the program to implement it is

### Recursive Fill Code Description

	Sequence Number
If the pixel at X,Y is not white, get a new pixel address from the stack and decrease the stack counter.	530
If P(X,Y) is white, color it red, and	570
If the pixel at X - 1,Y is white, put its address on the stack and add 1 to the stack counter.	600
If the pixel at X + 1,Y is white, put its address on the stack and add 1 to the stack counter.	640
If the pixel at X,Y - 1 is white, put its address on the stack and add 1 to the stack counter.	680
If the pixel at X,Y + 1 is white, put its address on the stack and add 1 to the stack counter.	720
Continue until the stack is empty.	770



Next pixel to be filled, ● pixels on the stack, ■ pixels filled

### Things to Do

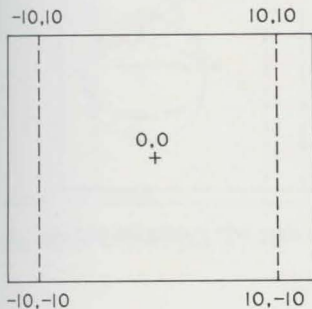
Before proceeding to the next chapter, prepare the functions and subroutines suggested here, which can be used as tools to simplify display of graphics. Refer back to the relevant sections if you need to refresh your memory.

### Windowing

Prepare a pair of windowing functions FNX and FNY that convert your computer's screen into the workspace illustrated by the first of the accompanying drawings.

Notice that the required window is square, whereas the ratio of length to width of your screen is probably about 13 to 10. Test the functions by drawing a line on the screen from -10,-10 to 10,10 using the equivalent of

```
MOVE FNX(-10),FNY(-10)
DRAW FNX(10),FNY(10)
```



Defining desired workspace



**Drawing**

Write the drawing subroutine so it uses FNX and FNY by modifying the MOVE and DRAW statements to

```
MOVE FNX(X8),FNY(Y8)
```

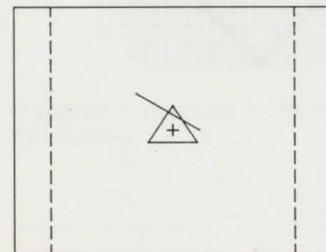
and

```
DRAW FNX(X8),FNY(Y8)
```

and by adding a RETURN statement at the end. Renumber the subroutine to begin at 500. To test the subroutine, use

```
DATA 4,-2,-1,2,-1,0,3,-2,-1      100
DATA 2,-3,3,2,2,0                 110
DATA -9999                        120
RESTORE 100                       130
GOSUB 500 ! Draw a figure          140
END
```

The result should be a triangle centered on the local origin, with a line drawn through its top, as illustrated in the second drawing.



Correct result of picture-drawing exercise

**Scaling**

Improve the subroutine to include the scaling capability and test it again, using the same test program as above with the added statement

```
S9 = 2                             131
```

The picture produced by this test should be the same as the one above, except twice as big.

**Translating**

Change the MOVE and DRAW commands in the subroutine to allow for translation as well as scaling

```
MOVE FNX(X8*S9 + X9),FNY(Y8*S9 + Y9)
```

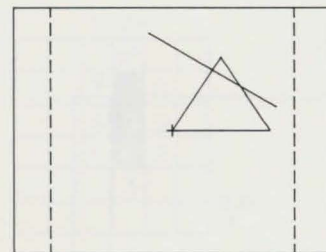
and

```
DRAW FNX(X8*S9 + X9),FNY(Y8*S9 + Y9)
```

Test again, using the same data statements, but using

```
RESTORE 100                        130
S9 = 2 ! Double scale              131
X9 = 2 ! Translate 2 units horiz.  132
Y9 = 1 ! Translate 1 unit vert.    133
GOSUB 500 ! Draw figure            140
```

The resulting picture should have the lower left-hand corner of the triangle in the middle of the screen, with the triangle scaled and translated as illustrated.



After scaling and translating picture

**Rotating**

Add the rotation capability to the subroutine by including the lines



```

C0 = COS(T*0.0174533) !  $\pi/180 = 0.0174533$ 
S0 = SIN(T*0.0174533)

```

and by changing the MOVE and DRAW commands to

```
MOVE FNX((X8*C0 - Y8*S0)*S9 + X9),FNY((X8*S0 + Y8*C0)*S9 + Y9)
```

and

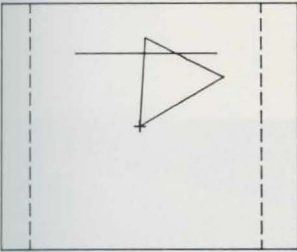
```
DRAW FNX((X8*C0 - Y8*S0)*S9 + X9),FNY((X8*S0 + Y8*C0)*S9 + Y9)
```

Test the subroutine with

```

RESTORE 100                                130
S9 = 2                                      131
X9 = 2                                      132
Y9 = 1                                      133
T = 30 ! Rotate 30 degrees                 134
GOSUB 500                                  140

```



After scaling, translating, and rotating picture

The scaled, translated, and rotated picture is shown in the fourth illustration. The line cutting through the triangle should now be very close to horizontal.

The completed subroutine may be used throughout the programs in this book. Another possibility is to save copies of the subroutine for unmodified figures (SIMPLE DRAW) and for figures to be translated (TRANSLATE A FIGURE, above), as well as a copy of the complete subroutine (ROTATE A FIGURE) because the latter will do some unnecessary work on figures that don't need rotation.

### Completing the Subroutine

Add FNX and FNY to the ELLIPSE routine and put a RETURN statement at the end. Renumber the subroutine to begin at 1000.

Test the subroutine, which generates a circle in the center of the screen, as follows:

```

X0 = 0 ! Center at 0,0                    100
Y0 = 0                                     110
M1 = 5 ! Circle of radius 5               120
M2 = 5                                     130
A1 = 0 ! Begin at 0 degrees               140
A2 = 2*PI ! End at 2 $\pi$  radians (360 degrees) 150
GOSUB 1000                                160

```

Now add the statements

```

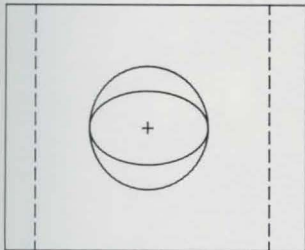
M2 = 3                                     170
GOSUB 1000                                180

```

to draw an ellipse inside the circle. The result is shown in the accompanying illustration.

Change A1 and A2 and observe the results.

Change line 100 to X0 = 1 and observe the results.



Drawing an ellipse inside a circle



## Randomness





Accuracy is the most valuable characteristic of a digital computer. Given a program and a known set of inputs, it is essential that a computer program invariably produce the same output.

In the development of a computer game, however, randomness is an important element. A player will quickly tire of a game unless it includes marginally unpredictable events within its rules. There is no point to playing a racing game if the order of finish is always the same. Action games involving the skill of one or more players have a built-in randomness, but if the computer participates in any way, its actions must be variable to make the game worthwhile.

---

### **Random Numbers**

The usual means of introducing variability into computer programs is through random numbers. Ways to produce what are called pseudo-random numbers must be available not only to the programmer of games but also to statisticians, researchers, operations research analysts, and anybody who needs to use the computer for simulation.

Pseudo-random numbers are so called because, while a sequence of these numbers conforms to most of the required tests for randomness, the sequence is always repeatable from the same set of initial conditions.

Most personal computers have a built-in RND function that supplies an unknown number between 0 and 1. In some applications, a random but repeatable sequence is required, as in the generation of a computer maze that must be the same every time the game is played. The Power Residue method generates such a sequence. Two numbers are used—a "seed" and a "multiplier." The multiplier should have half as many digits as the seed, so to select sizes for these numbers it is necessary to know how big a number your computer can handle without losing any digits.

Suppose  $N$  digits are allowed for the multiplier. The seed will have  $2N$  digits and the computer must be capable of retaining a number  $3N$  digits long without loss of accuracy. The subroutine will multiply the seed by the multiplier, yielding a number having  $3N$  digits. The rightmost, or least significant,  $2N$  digits become both the random number and the next seed.

In the case where a computer can retain a 12-digit number, a random number algorithm is as follows.

Having set the starting seed  $S$  to an appropriate 8-digit number and the multiplier  $M$  to an appropriate 4-digit number, execute the following routine to get a random fraction:

$$\begin{aligned}T &= S * M \\S &= T - \text{INT}(T/10^{\uparrow 8}) * 10^{\uparrow 8} \\R &= S/10^{\uparrow 8}\end{aligned}$$

$T$  becomes the 12-digit product of the seed and the multiplier. Then the new seed for the next random number is taken from the least-significant eight digits of the product. Finally, the random number is developed by converting the seed to an 8-digit decimal fraction.

Care must be taken to select the seed and multiplier in a way that avoids propagation of zeros. Choosing the seed 16777216 and the multiplier 3125 is clearly not a good idea:

Seed	Multiplier	Product	Truncated
16777216	3125	52428800000	28800000
28800000	3125	90000000000	00000000
0	3125	0	0

It is important to select values that cannot propagate zeros and will not generate sequences that repeat the same set of numbers over and over. The nature of this method is such that a repeating sequence is inevitable eventually, but that is unimportant as long as the number of random values developed before the sequence repeats is large enough.

The following program uses a seed  $S$  of 87654321 and a multiplier  $M$  of 2303

```

S = 87654321
M = 2303
I = 0
REPEAT
  I = I + 1
  T = S*M
  S = T - INT(T/10↑8)*10↑8
  R = S/10↑8
  PRINT R, " H"; ! H is control-H, or backspace
UNTIL I = 100

```

and generates as its first 100 numbers the values shown:

0.67901263	0.76608689	0.29810767	0.54196401
0.14311503	0.59391409	0.78414927	0.89576881
0.95556943	0.67639729	0.74295887	0.03427761
0.94133583	0.89641649	0.44717647	0.84741041
0.58617423	0.95925169	0.15664207	0.74668721
0.62064463	0.34458289	0.57439567	0.83322801
0.92410703	0.21849009	0.18267727	0.70575281
0.34872143	0.10545329	0.85892687	0.10858161
0.06344783	0.12035249	0.17178447	0.61963441
0.01804623	0.56046769	0.75709007	0.57843121
0.12707663	0.65747889	0.17388367	0.45409201
0.73389903	0.28946609	0.64040527	0.85333681
0.23467343	0.45290929	0.05009487	0.36848561
0.62235983	0.29468849	0.66759247	0.46545841
0.95071823	0.50408369	0.90473807	0.61177521
0.91830863	0.86477489	0.57657167	0.84455601
0.01249103	0.76684209	0.03733327	0.97852081
0.53342543	0.47876529	0.59646287	0.65398961
0.13807183	0.97942449	0.61460047	0.42488241
0.50419023	0.15009969	0.67958607	0.08671921
0.71434063	0.12647089	0.26245967	0.44462001
0.95988303	0.61061809	0.25346127	0.72130481
0.16497743	0.94302129	0.77803087	0.80509361
0.13058383	0.73456049	0.69280847	0.53790641
0.79846223	0.85851569	0.16163407	0.24326321



Multiplying any pair of numbers in the manner described above will cause the rightmost digit to cycle through a predictable sequence. Reading across the above list of numbers, you can see that as long as multiplying continues, the lowest digit will repeat the 3,9,7,1 pattern. Although this digit is cycling every four multiplications, the second lowest digit's cycle is longer:

6,8,6,0,0,0,2,8,4,2,8,6,8,4,4,4,2,6,0,2

Obviously it is not random, since it consists entirely of even numbers. Similar examination of the next digit to the left reveals that its cycle is even longer and that all the digits 0 through 9 occur: this digit is more "random" than the second lowest digit. In fact, each sequence gets more random, with the digit to the right of the decimal point having a very long cycle.

Therefore, it pays to choose digits from the high-order end of the number or to use the random number as a fraction, as above, by putting the decimal point on the left-hand end. Values for the seed and multiplier should end in 1,3,7 or 9 to avoid propagating 0s or 5s.

If this method is used to generate numbers with a random starting value, it is necessary to find a way to choose a potentially different seed each time the program is used. (The multiplier can be a constant.) One way to get a varying seed is to ask the user to provide the date, such as 12/31/83, and the time of day, say 11:30. Concatenate month, day, hour and minute to get 12311130. Examine the units digit. While it is not 1,3,7, or 9, add 1, getting 12311131 in this case. Use this value for the starting seed.

```
PRINT "Enter numeric month, day, and year (mm/dd/yy): ";
INPUT M,D,Y
PRINT "Enter time of day (hh:mm): ";
INPUT H,Y
S = ((M*100 + D)*100 + H)*100 + Y
M = S - INT(S/10)*10
IF M = INT(M/2)*2 THEN
  S = S + 1
  IF M = 5 THEN
    S = S + 2
```

Random fractions may be employed in applications requiring whole numbers. Because the random number is always greater than 0 and less than 1, the expression

```
INT(N*R)
```

where R is the random number and N is an integer, will yield a whole number between 0 and N - 1. To roll dice, for example,  $\text{INT}(6*R)+1$  gives a number between 1 and 6.

---

### Goaltender Game

Random numbers are put to good use in Goaltender, a game in which the computer shoots "pucks" and you try to stop them with a "hockey stick." Random



variables are used to control the starting point and direction of the puck. If the screen extents are assumed to be 0 to 130 in X and 0 to 100 in Y, goal posts may be drawn by the Ellipse subroutine described in Chapter 2, as circles of radius 2 at 30,5 and 100,5. The hockey stick may be drawn as a rectangle 7 units long and 2 units deep so that its top edge is on the line  $Y = 11$ .

The stick must move between  $X = 30$  and  $X = 100$  to protect the goal. If your computer has a game paddle, use it to control the stick by reading a value that represents the rotational position of the game paddle and use it as the center of the stick X2.

In the following game, Subroutine 2000 will be used for reading the game paddle and drawing the stick. Part of Subroutine 2000 depends on how your machine reads the game paddle (or other similar input device). Let's say your paddle gives values between 0 and 255 when you use the function PDL(1). Subroutine 2000 will be

### **Hockey Stick Controller Program Code**

	Sequence Number
X2 = PDL(1)	2000
X2 > 126.5 THEN	2010
X2 = 126.5	2020
;	2030
IF X2 < 3.5 THEN	2040
X2 = 3.5	
	2050
;	2060
MOVE X2 - 3.5,11	2070
DRAW X2 + 3.5,11	2080
DRAW X2 + 3.5,9	2090
DRAW X2 - 3.5,9	2100
DRAW X2 - 3.5,11	2110
RETURN	2120

### **Hockey Stick Controller Code Description**

	Sequence Number
The paddle value is read into X2. Since paddle values can exceed screen X-coordinates, X2 must be examined. The right-hand edge of the hockey stick will be off the screen when it is greater than 130, or when X2, the center of the stick, is greater than 126.5. Likewise, the left-hand edge of the stick will be off the screen whenever X2 is less than 3.5. So X2 is set to 126.5 if it was greater than 126.5 and to 3.5 if it was less than 3.5. Steps 2000 through 2060 can be replaced by	2000

$X2 = \text{PDL}(1) \text{ MAX } 3.5 \text{ MIN } 126.5$

in computers supporting the MAX and MIN functions.

The upper left-hand corner of the stick is on the line $Y = 11$ and 3.5 units to the left of (less than) X2. Move to that corner and draw the stick.	2070
--	------

In most computers, whenever a new stick position is read, the previous stick must be erased. Suppose a reserved word COLOR is used by your computer to define the color in which lines are to be drawn and that color 0 is black and color 1, white. Subroutine 2000 then becomes

**Hockey Stick Controller Revised Program Code**

	Sequence Number
COLOR = 0	2000
MOVE T2 - 3.5,11	2010
DRAW T2 + 3.5,11	2020
DRAW T2 + 3.5,9	2030
DRAW T2 - 3.5,9	2040
DRAW T2 - 3.5,11	2050
COLOR = 1	2060
X2 = PDL(1) MAX 3.5 MIN 126.5	2070
MOVE X2 - 3.5,11	2080
DRAW X2 + 3.5,11	2090
DRAW X2 + 3.5,9	2100
DRAW C2 - 3.5,9	2110
DRAW X2 - 3.5,11	2120
T2 = X2	2130
RETURN	2140

Notice that the first time the subroutine is used, T2 has not been assigned a value. After the first time, T2 is set on line 2130. To avoid problems with undefined variables, T2 can be set to a valid value, say 65, at the beginning of the program.

To test Subroutine 2000, add it to the following program. Also, in order to draw the goal posts and the puck, add the ELLIPSE subroutine, renumbered to start at 700.

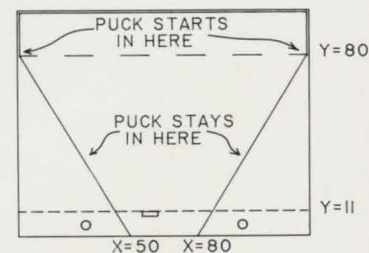
**Test Hockey Stick Program Code**

	Sequence Number
X0 = 30	100
Y0 = 5	110
M1 = 2	120
M2 = 2	130
A1 = 0	140
A2 = 2*PI	150
GOSUB 700	160
X0 = 100	170
GOSUB 700	180
REPEAT	190
GOSUB 2000	200
UNTIL X0 <> X0	210

The program draws goal posts at 30,5 and 100,5 and repeatedly calls Subroutine 2000. Twist the game paddle knob back and forth as far as you can. The hockey stick should move on a horizontal line from one edge of the screen to the other.

To keep the game simple, we might decide that all shots will be "on goal"—none will hit a goal post nor miss the goal mouth. If all shots originate above  $Y = 80$  and travel on a straight line, values of  $X$  for the goal end of the shot may be determined so that all shots are on goal, as shown in the Goaltender diagram.

So the coordinate values of the point from which the shot is to be taken,  $X5, Y5$ , must be chosen from the ranges  $0 \leq X5 \leq 130$  and  $80 \leq Y5 \leq 100$ . If the function  $RND(-2)$  returns a value between 0 and 1, it is clear that  $130 * RND(-2)$  will have values between  $130 * 0$  and  $130 * 1$ , so



Determining that all shots will be "on goal"



$$X5 = 130 * \text{RND}(-2)$$

For Y5, the minimum value must be 80. Since the difference between 80 and the maximum value, 100, is 20,

$$Y5 = 80 + 20 * \text{RND}(-2)$$

The point at the other end of the line on which the puck travels, X1,Y1, is at Y1 = 0 and  $50 \leq X1 \leq 80$ , so

$$X1 = 50 + 30 * \text{RND}(-2)$$

and Y1 = 0

To make the puck move along the line between X5,Y5 and X1,Y1, we will successively draw a new puck at a distance D along the line away from the previous position. The current puck location will be X3,Y3. The first puck position is therefore

$$X3 = X5$$

$$Y3 = Y5$$

If we assume that D = 3, where is the next puck position? In general (referring to the accompanying diagram),  $\lambda$  is the angle whose tangent is  $\Delta X / \Delta Y$ , but the angle to be used here is  $\theta$ , the angle whose tangent is  $\Delta Y / \Delta X$  ( $\theta = \arctan \Delta Y / \Delta X$ ),  $\Delta Y = Y5 - Y1$ , and  $\Delta X = X5 - X1$ . Y5 - Y1 will always be greater than zero, but X5 - X1 may be less than, greater than, or equal to zero. Since the puck must be made to travel toward X1,Y1, a positive value for  $\theta$  can be computed from

$$\theta = \arctan((Y5 - Y1) / \text{ABS}(X5 - X1))$$

So the next value of Y3 can be computed:

$$Y3 = Y3 + D * \sin(\theta)$$

If X3 is computed from

$$X3 = X3 + D * \cos(\theta)$$

the puck will always travel from left to right. If  $X1 < X5$ , it should travel from right to left, or

$$X3 = X3 - D * \cos(\theta)$$

Therefore, we could write

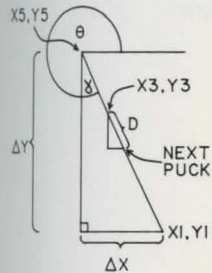
```
IF X1 >= X5 THEN
  X3 = X3 + D * COS(theta)
ELSE
  X3 = X3 - D * COS(theta)
;
```

or simply

$$X3 = X3 + D * \cos(\theta) * \text{SGN}(X1 - X5)$$

In the determination of the value of  $\theta$  there is a chance for error if  $X5 = X1$ , because division by zero is undefined in

$$\theta = \text{ATN}((Y5 - Y1) / \text{ABS}(X5 - X1))$$



Determining next puck position



If we treat  $X5 = X1$  as a special case, in which the puck is traveling straight down, then

$$\begin{aligned} X3 &= X5 \\ Y3 &= Y3 + D \end{aligned}$$

The puck travels downward until a goal is scored or a save is made. Since the puck will always go in the goal unless saved, a goal can be inferred from a "no save" condition. To determine a save, we have to see if the center of the puck is horizontally within 3.5 units of the center of the stick when the puck reaches  $Y = 11$ . Although it is not difficult to determine the exact value of  $X3$  when  $Y = 11$ , a good approximation is to test for this condition (see illustration):

Is  $ABS(X3 - X2) < 3.5$ ?

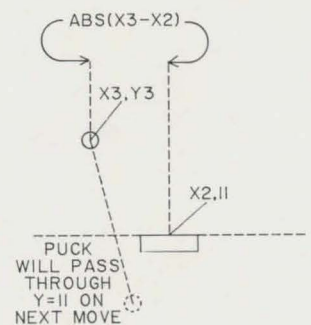
A better approximation is to check if either the last puck above  $Y = 11$  or the first one below  $Y = 11$  is on the stick. If so, it's a save; if not, a goal.

For realism's sake it is a good idea to show the puck bouncing off the stick if a save is made. That's easy to do—after a save, reverse the direction of  $Y$ . Instead of subtracting  $D \cdot \sin(\theta)$  from  $Y3$ , add it. Continue to display the puck until it passes through  $Y = 50$  or goes off the side of the screen.

A minimum version of Goaltender can now be coded. The proper way to convert this game for use on your machine is first to understand how the algorithm works. Then you may want to use the code as a guide in writing your own version. Remember, though, that you must accommodate the differences between your machine and the example. For instance, most versions of BASIC require an explicit definition of  $PI$ :

$PI = 3.14159...$

Also, you *must* add `DEF FNX` and `DEF FNY` statements as described in Chapter 2. An example of the use of these statements is shown at the end of this chapter in versions of Goaltender that run on various personal computers.



Determining a goal

## Goaltender Program Code

	Sequence Number
REPEAT	100
$X0 = 30$	110
$Y0 = 5$	120
$M1 = 2$	130
$M2 = 2$	140
$A1 = 0$	150
$A2 = 2 \cdot PI$	160
GOSUB 700	170
! Draw left goal post	180
GOSUB 700	190
! Draw right goal post	200
$X5 = 130 \cdot RND(-2)$	210
$Y5 = 80 + 20 \cdot RND(-2)$	220
$X3 = X5$	230
$Y3 = Y5$	240
$X1 = 50 + 30 \cdot RND(-2)$	250
$Y1 = 0$	260
$D = 3$	270
$M1 = 1$	280
! Set values for ELLIPSE subroutine	
! Generate random starting point X5,Y5	
! Start the puck at X5,Y5	
! Generate random ending point X1,Y1	
! Set puck travel increment	
! Reset ellipse radii	

```

M2 = 1          ! to puck size          280
S = -1          ! Set "not saved" flag   290
REPEAT          300
  GOSUB 2000     ! Read paddle and move hockey stick 310
  IF X5 = X1 THEN ! If puck is traveling straight down, 320
    Y3 = Y3 + D*S ! change Y-location          330
  ELSE          340
    T = ATN((Y5 - Y1)/ABS(X1 - X5)) ! Otherwise,          350
    X3 = X3 + D*COS(T)*SGN(X1 - X5) ! change X          360
    Y3 = Y3 + D*SIN(T)*S           ! and Y of puck        370
  ;            380
  IF Y3=>11 and Y3+D*SIN(T)*S<11 THEN ! If puck at stick line 390
    IF ABS(X3-X2)<3.5 OR ABS(X3+D*COS(T)*SGN(X1-X5))<3.5 THEN 400
      S = 1 ! set "saved" flag if puck hits stick 410
    ;            420
  ;            430
  X0 = X3 ! Put puck coordinates in ellipse 440
  Y0 = Y3 ! subroutine variables, and      450
  GOSUB 700 ! draw puck                    460
  UNTIL Y3 < 2 OR X3 < 0 OR X3 > 130 OR (Y3 > 50 AND S = 1) 470
  PAGE ! If finished, clear screen          480
  UNTIL X0 <> X0 ! and play again           490

```

### Goaltender Code Description

Sequence  
Number

After drawing the goal posts, randomly generate the endpoints of the line along which the puck will travel. 190

Set the puck travel increment D and prepare radii M1, M2 to draw the puck. 260

Set "not saved" flag S = -1. 290

Move the hockey stick to the position read from the game paddle. 310

Change the Y-location of the puck if puck travel is vertical. 320

Change X and Y if puck travel is not vertical. 340

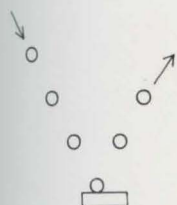
If the puck is crossing the line along which the stick travels, or if it will cross on the next move, determine if the X-value of the center of the puck in either position is within 3.5 units of the center of the stick. 390

If so, set S, the "saved" flag. Notice on lines 330 and 370 that since S has been changed from -1 to +1, the Y-direction of the puck will reverse. Since the X direction is not changed, the puck will appear to bounce off the stick (see illustration). 410

Prepare the ELLIPSE Subroutine for drawing the puck, and draw it. 440

Continue until a goal (Y3 < 2) or the puck has left the side of the screen (X3 < 0 OR X3 > 130) or the puck has passed Y = 50 after a save. 370

Clear screen and play again. 480



Bouncing the puck off the hockey stick



Some small but important changes are required before this game will run fast enough to be worth playing. The Ellipse Subroutine approximates circles and ellipses by drawing a short line every degree around the arc—360 lines are needed for a complete circle. For this program, change lines 740 and 820 to

```
I = I + PI/9                                740
and I = I - PI/9                             820
```

Now a puck will be drawn with straight lines every 20 degrees, eighteen lines per puck. If necessary, reduce the divisor even further, to 4.5 for nine lines or to 3 for six lines per puck. If the program is still too slow, replace lines 440–460 with code that draws a six-sided puck directly:

```
MOVE X3 + 1, Y3                                440
DRAW X3 + .5, Y3 + .866                        443
DRAW X3 - .5, Y3 + .866                        447
DRAW X3 - 1, Y3                                450
DRAW X3 - .5, Y3 - .866                        453
DRAW X3 + .5, Y3 - .866                        457
DRAW X3 + 1, Y3                                460
```

Having written and debugged the above algorithm for your machine, you may now consider one or two improvements. First of all, let's add some code for scorekeeping. Secondly, the angle  $\theta$  is being recomputed every time the puck moves. The value of  $\theta$ , and consequently  $\text{SIN}(\theta)$  and  $\text{COS}(\theta)$ , can be determined before beginning the shot. Finally, there is no need to make a special case of the vertical shot if  $\theta$  is set to 0 in this case. The "cleaned-up" algorithm is

<b>Goaltender Revised Code</b>	Sequence Number
GO = 0	100
SO = 0	110
REPEAT	120
X0 = 30	130
Y0 = 5	140
M1 = 2	150
M2 = 2	160
A1 = 0	170
A2 = 2*PI	180
GOSUB 700	190
X0 = 100	200
GOSUB 700	210
X5 = 130*RND(-2)	220
Y5 = 80 + 20*RND(-2)	230
X3 = X5	240
Y3 = Y5	250
X1 = 50 + 30*RND(-2)	260
D = 3	270
T = 0	280
! Preset $\theta = 0$	
IF X5 <> X1 THEN	290
! If line not vertical,	
T = ATN(Y5/ABS(X5 - X1))	300
! find $\theta$	
;	310



C1 = D*COS(T)	! Compute X and	320
S1 = D*SIN(T)	! Y increments	330
M1 = 1		340
M2 = 1		350
S = -1		360
REPEAT		370
GOSUB 2000	! Since $\theta$ is defined for a	380
X3 = X3 + C1*SGN(X1 - X5)	! vertical line, test is	390
Y3 = Y3 + S1*S	! not necessary	400
IF Y3 => 11 AND Y3 + S1*S < 11 THEN		410
IF ABS(X3-X2)<3.5 OR ABS(X3+C1*SGN(X5-X1)-X2)<3.5 THEN		420
S = 1		430
;		440
;		450
X0 = X3		460
Y0 = Y3		470
GOSUB 700		480
UNTIL Y3 < 2 OR X3 < 0 OR X3 > 130 OR (Y3 > 50 AND S = 1)		490
PAGE		500
IF Y3 < 2 THEN	! If puck passed the goal line,	510
GO = GO + 1	! increase "goals"	520
ELSE	! If not,	530
S0 = S0 + 1	! increase "saves"	540
;		550
PRINT GO;" GOALS"	! Print score	560
PRINT S0;" SAVES"		570
UNTIL X0 <> X0	! Play again	580

**Goltender Revised Code Description**

Let G0 be the number of goals, S0 the number of saves. Set them to zero.	Sequence Number 100
Draw goal posts.	130
Compute random variables.	220
Preset $\theta$ to zero. If the shot line is not vertical, compute $\theta$ . Notice that the variable Y1, which is always zero, has been omitted.	280
Compute C1 and S1, the X and Y puck increments.	320
Set puck radii and "not saved" flag.	340
Begin shot.	370
Add puck increments to puck position.	390
Test for a goal.	410
Display puck.	460
Continue until goal or save.	490
Clear screen.	500
If the puck crossed the goal line, add 1 to goals. If not, add 1 to saves.	510
Print number of goals and saves.	560
Shoot again.	580

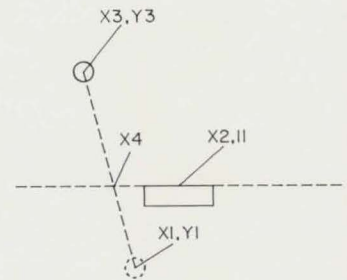
### Refinements

- Determine accurately if the puck hits the stick (see illustration).  
The value of  $X4$  is the X-coordinate of the center of the puck and may be calculated by the proportion:

$$X4 = \frac{X1 - X3}{Y3 - Y1} (Y3 - 11) + X3$$

If  $ABS(X4 - X2) < 3.5$ , the puck hits the stick.

- Vary the speed of the shot by making  $D$  a random variable between 3 and 6.
- When a goal is scored, flash a green light at the top center of the screen and sound a buzzer. (In most computers, a tone can be sounded by printing "Control-G" characters.)



Determining if puck hits or misses stick

### Things to Do

Convert Goaltender to run on your own machine. Some examples follow.

#### IBM Personal Computer Version Of Goaltender

In converting Goaltender for use on a specific computer, it is helpful to incorporate any special features of that computer's version of the programming language. The IBM personal computer has a command called CIRCLE, so that the ELLIPSE/CIRCLE routine (Subroutine 700 in the code above) can be omitted and CIRCLE commands used instead.

The IBM random number generator is invoked by using the keyword RND to get a random number between 0 and 1. However, the sequence of numbers produced by successive RNDs is always the same unless a RANDOMIZE(n) is used at the beginning of the program. This will change the random number sequence based on the value of  $n$ .  $N$  should be different each time the program is run. Fortunately, there is a clock in the IBM computer that may be read through the keyword TIME\$, giving hours, minutes, and seconds—for example, 13:25:30 is 1:25PM and 30 seconds. Part of the time value may be used in the RANDOMIZE statement, preferably seconds or seconds times minutes.

Since the IBM personal computer does not have a game paddle, use its arrow (direction) keys to move the hockey stick.

Despite changes to the code, the algorithm is the same as before. To convert the  $X$  and  $Y$  units used in the original Goaltender to fit the IBM screen, remember that in the original (see Chapter 2 section, *Windowing and Viewporting*)

```
X1W = 0
X2W = 130
Y1W = 0
Y2W = 100
```

whereas the IBM screen coordinates are

```
X1S = 0
X2S = 319
Y1S = 199
Y2S = 0
```

That is, while the original version was coded for a screen measuring 0 to 130 units in X, 0 to 100 units in Y, reading from the bottom left-hand corner, the IBM screen is 0 to 319 units in X, 0 to 199 units in Y, reading from the top left-hand corner. Using the windowing transformation

$$\text{FNX}(X) = \frac{X2S - X1S}{X2W - X1W} (X - X1W) + X1S$$

$$\text{FNY}(Y) = \frac{Y2S - Y1S}{Y2W - Y1W} (Y - Y1W) + Y1S$$

$$\text{or } \text{FNX}(X) = \frac{319 - 0}{130 - 0} (X - 0) + 0$$

$$\text{FNY}(Y) = \frac{0 - 199}{100 - 0} (Y - 0) + 199$$

which simplifies to

$$\text{FNX}(X) = \frac{319}{130} X$$

$$\text{FNY}(Y) = \frac{-199}{100} Y + 199$$

### **Goaltender for IBM Personal Computer Program Code**

```

100 DEF FNX(X) = X*319/130 ! Define windowing function
110 DEF FNY(Y) = Y*(-199)/100 + 199
120 DEFSTR A ! Establish data types
130 DEFINT I - N
140 DEFSNG B - H, 0 - Z
150 A$ = ""
160 SCREEN 1 ! Set medium resolution
170 I = VAL(RIGHT$(TIMES$,2)) ! Get seconds from time of day
180 RANDOMIZE(I) ! Randomize the random number generator
190 G0 = 0: S0 = 0: X2 = 65 ! Set score and hockey stick location
200 WHILE 1 = 1
210 CIRCLE (INT(FNX(30)),INT(FNY(5))),5,2 ! Draw goal posts
220 CIRCLE (INT(FNX(100)),INT(FNY(5))),5,2
230 X5 = 130*RND ! Establish line of shot
240 Y5 = 80 + 20*RND
250 X3 = X5
260 Y3 = Y5
270 X1 = 50 + 30*RND
280 D = 3
290 T = 0
300 IF X5 <> X1 THEN T = ATN(Y5/ABS(X5 - X1))
310 C1 = D*COS(T)
320 S1 = D*SIN(T)
330 S = -1
340 WHILE Y3 > 2 AND X3 > 0 AND X3 < 130 AND (Y3 < 50 OR S = -1)
350 LINE (FNX(X2-3.5),FNY(11))-(FNX(X2+3.5),FNY(9)),1,B !Draw stick
360 X3 = X3 + C1*SGN(X1 - X5) ! Compute location of puck
370 Y3 = Y3 + S1*S

```



```

380 IF Y3 >= 11 AND Y3 + S1*S < 11 AND (ABS(X3 - X2) < 3.5 OR
    ABS(X3 + C1*SGN(X5 - X1) - X2) < 3.5) THEN S = 1 ! Test for save
390 CIRCLE(X,Y),4,0 ! Erase puck
400 A$ = INKEY$ ! Has a key been pressed?
410 IF LEN(A$) <> 0 THEN GOSUB 2000 ! If so, move stick
420 X = FNX(X3): Y = FNY(Y3)
430 CIRCLE(X,Y),4,1 ! Draw puck
440 WEND
450 CLS ! Clear screen
460 IF Y3 < 2 THEN G0 = G0 + 1 ELSE S0 = S0 + 1 ! Add to score
470 LOCATE 1,1: PRINT G0; "GOALS" ! Display score at top left
480 PRINT S0; "SAVES"
490 WEND ! Play again

2000 LINE(FNX(X2-3.5),FNY(11))-(FNX(X2+3.5),FNY(9)),0,B !Erase stick
2010 IF A$ = "6" THEN X2 = X2 + 3 ! Test for arrow key press
2020 IF A$ = "4" THEN X2 = X2 - 3 ! and move stick
2030 IF RIGHT$(A$,1) = "M" THEN X2 = X2 + 1
2040 IF RIGHT$(A$,1) = "K" THEN X2 = X2 - 1
2050 A$ = ""
2060 RETURN

```

### Goaltender for IBM Personal Computer Code Description

Sequence  
Number

The windowing transformation is set up.	100
All variables beginning with the letter A are to be character strings.	
All variables beginning with I,J,K,L,M, or N are to be integers (whole numbers).	
All variables beginning with B through H and O through Z are to be "single precision" variables (capable of about 8 digits of precision).	140
The string variable A\$ is set to "null."	150
The screen is set to medium resolution (320 x 200).	160
I is set to the rightmost 2 digits of the time of day.	170
The random number generator is randomized.	180
Goals (G0) and saves (S0) are set to zero, and the hockey stick is centered in the goal (X2). Note that more than one statement may be entered on a line if the statements are separated by colons.	190
The sequence of instructions between 200 and 490 will continue as long as 1 = 1, that is, "forever." The WHILE...WEND construct is similar to the WHILE...DO in the examples.	200
Goal posts are drawn at 30,5 and 100,5 having a radius of 5 pixels in color 2.	210
The line along which the puck will travel is computed.	230
The variable S is set to indicate that the puck has not been saved.	330
Statements 340 through 440 are executed until a goal has been scored or the puck has rebounded after a save.	340
The rectangle defining the hockey stick is drawn.	350
The location where the puck will pass through the line of the stick is computed. If it will do so immediately, S is set to 1 if the stick will intercept the puck.	380
The puck is erased.	390

If a key has been pressed, it is entered into A\$, and Subroutine 2000 is called.	400
The next location of the puck is transformed into an IBM screen location X,Y.	420
The puck is redrawn at X,Y.	430
The sequence 340 – 440 is continued until satisfied.	440
The screen is cleared.	450
If a goal was scored, G0 is increased by 1. If not, S0 is increased by 1.	460
Numbers of goals and saves are printed.	490
The sequence 200 – 490 is continued.	490
SUBROUTINE 2000:	
The hockey stick is erased.	2000
If the last key pressed was a shift→, the stick is moved 3 units to the right;	2010
if a shift←, 3 to the left;	
if an unshifted →, 1 to the right;	
if an unshifted ←, 1 to the left.	
Clear A\$.	2050
Return to the main program.	2060

### **Apple II Version Of Goaltender**

In the Apple computer, one of the high-resolution graphics modes allows pictures on most of the screen with three lines of text at the bottom. This is where the goals and saves will be printed.

The graphics part of the screen is 0 to 279 units in X and 0 to 159 in Y, reading from the top left-hand corner:

```
X1S = 0
X2S = 279
Y1S = 159
Y2S = 0
```

To get a random number between 0 and 1, RND(5) is used.

Game paddles, available for the Apple, are used in this version. The function PDL(1) returns a value between 0 and 255, depending on the position of the rotary knob on paddle 1. If we use PDL(1)/2 to position the hockey stick, we will be able to put its center anywhere between 0 and 127.5. Since the stick is 7 units wide and the screen is defined to be from 0 to 130 in X, the entire range of the screen can be covered by the stick.

Unlike the IBM personal computer version of BASIC, the Apple version does not have a CIRCLE command, so we will use the ELLIPSE Subroutine to draw circles.

### **Goaltender For Apple II Program Code**

Sequence  
Number

```
G0=0: S0=0 X2=65                      ! Set score and hockey stick location 90
```



```

PRINT: PRINT: PRINT: PRINT ! Clear text from bottom of screen 95
HGR: HCOLOR=3 ! Set high resolution and color white 100
DEF FN(X) = X/130*279 ! Define windowing functions 110
DEF FNY(Y) = Y/100(-159) + 159 120
X0 = 30: Y0 = 5: M1 = 2: M2 = 2: A1 = 0 130
PI = 3.14159 ! Define  $\pi$  140
A2 = 2*PI 150
GOSUB 700 ! Draw goal posts 160
X0 = 100 170
GOSUB 700 180
X5 = 130*RND(5): Y5 = 80 + 20*RND(5) ! Establish line of shot 190
X3 = X5: Y3 = Y5 200
X1 = 50 + 30*RND(5): Y1 = 0 210
D = 3: M1 = 1: M2 = 1: S = -1 220
GOSUB 2000 ! Draw stick 230
IF X5 = X1 THEN Y3 = Y3 + D*S 240
IF X5 <> X1 THEN T = ATN((Y5 - Y1)/ABS(X1 - X5)): 250
X3=X3+D*COS(T)*SGN(X1-X5): Y3=Y3+D*SIN(T)*S ! Locate puck
IF Y3 >= 11 AND Y3 + D*SIN(T)*S < 11 THEN 260
IF ABS(D*COS(T)*SGN(X1 - X5)) < 3.5 THEN S = 1 ! Test for save
X0 = X3: Y0 = Y3 270
GOSUB 700 ! Draw puck: note that previous puck hasn't been erased 280
IF NOT (Y3<2 OR X3<0 OR X3>130 OR (Y3>50 AND S=1)) THEN 230 290
IF Y3 < 2 THEN GO = GO + 1 ! Add to score 300
IF Y3 >= 2 THEN SO = SO + 1 310
PRINT GO: "GOALS" ! Display score at bottom 320
PRINT SO: "SAVES" ! of screen 330
PRINT 340
GOTO 100 ! Play again 350

I = A1 ! Ellipse subroutine 700
HPLLOT FN(X0 + M1*COS(I)),FNY(Y0 + M2*SIN(I)) 710
IF A1 >= A2 THEN 800 720
I = I + PI/9 730
IF I > A2 THEN I = A2 740
HPLLOT TO FN(X0 + M1*COS(I)),FNY(Y0 + M2*SIN(I)) 750
IF I <> A2 THEN 730 760
RETURN 770
I = I - PI/9 800
IF I < A2 THEN I = A2 810
HPLLOT TO FN(X0 + M1*COS(I)),FNY(Y0 + M2*SIN(I)) 820
IF I <> A2 THEN 800 830
RETURN 840

HCOLOR = 0 ! Erase stick 2000
HPLLOT FN(X2 - 3.5),FNY(11) TO FN(X2 + 3.5),FNY(11)
TO FN(X2 + 3.5),FNY(9) TO FN(X2 - 3.5),FNY(9)
TO FN(X2 - 3.5),FNY(11) 2010
X2 = PDL(1)/2: HCOLOR = 3 ! Locate and draw stick 2020
HPLLOT FN(X2 - 3.5),FNY(11) TO FN(X2 + 3.5),FNY(11)
TO FN(X2 + 3.5),FNY(9) TO FN(X2 - 3.5),FNY(9)
TO FN(X2 - 3.5),FNY(11) 2030
RETURN 2040

```



**Goaltender for Apple II Code Description**

	Sequence Number
Scores are zeroed and the hockey stick location is set.	90
Text lines are erased.	95
The windowing transformation is set up.	100
Variables to draw the goal posts are set and the posts drawn.	130
The line along which the puck will travel is computed.	190
Subroutine 2000 reads the position of the rotary knob on game paddle 1 and moves the stick.	230
The location where the puck will pass through the line of the stick is computed. If it does so on this turn, S is set to 1.	250
The puck is drawn in its new location.	280
If a goal has been scored or a save made, the score is changed and displayed at the bottom of the screen.	300
The game continues with another shot.	350

**DEC Professional Version of Goaltender**

The Professional has arrow keys, but they return the same values whether the Shift key is up or down. Consequently, to convert the Goaltender algorithm, other keys must be used to move the hockey stick, for example, the letter keys *a* and *d*. With the Shift key up, keys *a* and *d* will cause the stick to move one unit to the left or right. With the Shift key down, the stick will move three units.

Multiple BASIC statements may appear on the same line, if separated by the backslash character. PRO/ BASIC supports user function definitions, but not the SGN function. To simulate SGN, it is therefore necessary to write a function FNSGN that returns -1 whenever the function value is less than zero, 0 when equal to 0, and +1 when greater than 0.

Viewport and window commands are available on the Professional, as is the random number function RND, which returns a value between 0 and 1. The PLOT ARC command permits a user to draw arcs or circles without resorting to SIN and COS functions.

**Goaltender for DEC Professional Program Code**

```

100 SET VIEWPORT 0,1,0,.625           ! Define viewport and window
110 SET WINDOW 0,130,0,100
120 DEF FNSGN(PARM)=INT(PARM/(ABS(PARM)+(PARM = 0)))!Simulate SGN
150 A$ = ""
160 CLEAR                             ! Clear screen
180 RANDOMIZE ! Randomize the random number generator
190 GO=0\ S0=0\ X2=65 ! Set score and hockey stick location
210 SET POSITION (30,10)\ PLOT ARC (30,5,360) ! Draw goal posts
220 SET POSITION (100,10)\ PLOT ARC (100,5,360)
230 X5 = 130*RND                       ! Establish line of shot
240 Y5 = 80 + (20*RND)
250 X3 = X5
260 Y3 = Y5

```

```

270 X1 = 50 + (30*RND)
280 D = 3
290 T = 0
300 IF X5 <> X1 THEN T = ATN(Y5/ABS(X5 - X1))
310 C1 = D*COS(T)
320 S1 = D*SIN(T)
330 S = -1
340 IF Y3<=2 OR X3<=0 OR X3>=130 OR (Y3>=50 AND S<>-1) THEN GOTO 450
350 PLOT (X2-3.5,11),(X2-3.5,9),(X2+3.5,9),(X2+3.5,11),(X2-3.5,11)
360 X3 = X3 + C1*FNSGN(X1 - X5)          ! Draw stick (line 350) and
370 Y3 = Y3 + S1*S                      ! compute location of puck
375 TMP = ABS(X3 + C1*FNSGN(X1 - X5) - X2)! Test for save (line 380)
380 IF Y3>=11 AND Y3+S1*S<11 AND (ABS(X3-X2)<3.5 OR TMP<3.5) THEN S=1
385 SET WRITING MODE 8                  ! Set erase mode
390 SET POSITION (X,Y+ 4)\ PLOT ARC (X,Y,360) ! Erase puck
395 SET WRITING MODE 4                  ! Set draw mode
400 CALL INKEY(A$)                      ! Has a key been pressed?
410 IF LEN(A$) <> 0 THEN GOSUB 2000      ! If so, move stick
420 X = X3\ Y = Y3
430 SET POSITION (X,Y + 4)\ PLOT ARC (X,Y,360) ! Draw puck
440 GOTO 340
450 CLEAR                               ! Clear screen
460 IF Y3 < 2 THEN GO = GO + 1 ELSE SO = SO + 1 ! Add to score
470 SET POSITION (1,1)\ PRINT GO;" Goals" ! Display score at top left
480 PRINT SO;" Saves"
490 GOTO 150                            ! Play again
2000 SET WRITING MODE 8                  ! Erase stick
2001 PLOT (X2-3.5,11),(X2-3.5,9),(X2+3.5,9),(X2+3.5,11),(X2-3.5,11)
2005 SET WRITING MODE 4
2010 IF A$ = "D" THEN X2 = X2 + 3        ! Test for key press
2020 IF A$ = "A" THEN X2 = X2 - 3        ! and move stick
2030 IF A$ = "d" THEN X2 = X2 + 1
2040 IF A$ = "a" THEN X2 = X2 - 1
2050 A$ = ""
2060 RETURN

```

### Goaltender for DEC Professional Code Description

	Sequence Number
Define a viewport consisting of the whole screen.	100
Define the range of the screen as 0 to 130 in X, 0 to 100 in Y.	110
Define the SGN function.	120
Initialize variables and clear the screen.	150
Draw the goal posts.	210
Compute the line of travel of the puck.	230
Set S to indicate that the puck has not been saved.	330
Statements 340 through 440 are executed until a goal has been scored or the puck has rebounded after a save.	340
The hockey stick is drawn.	350
A test is made to see if the puck is about to hit the stick or pass over the goal line.	380
The computer is prepared to draw in the erase mode.	385

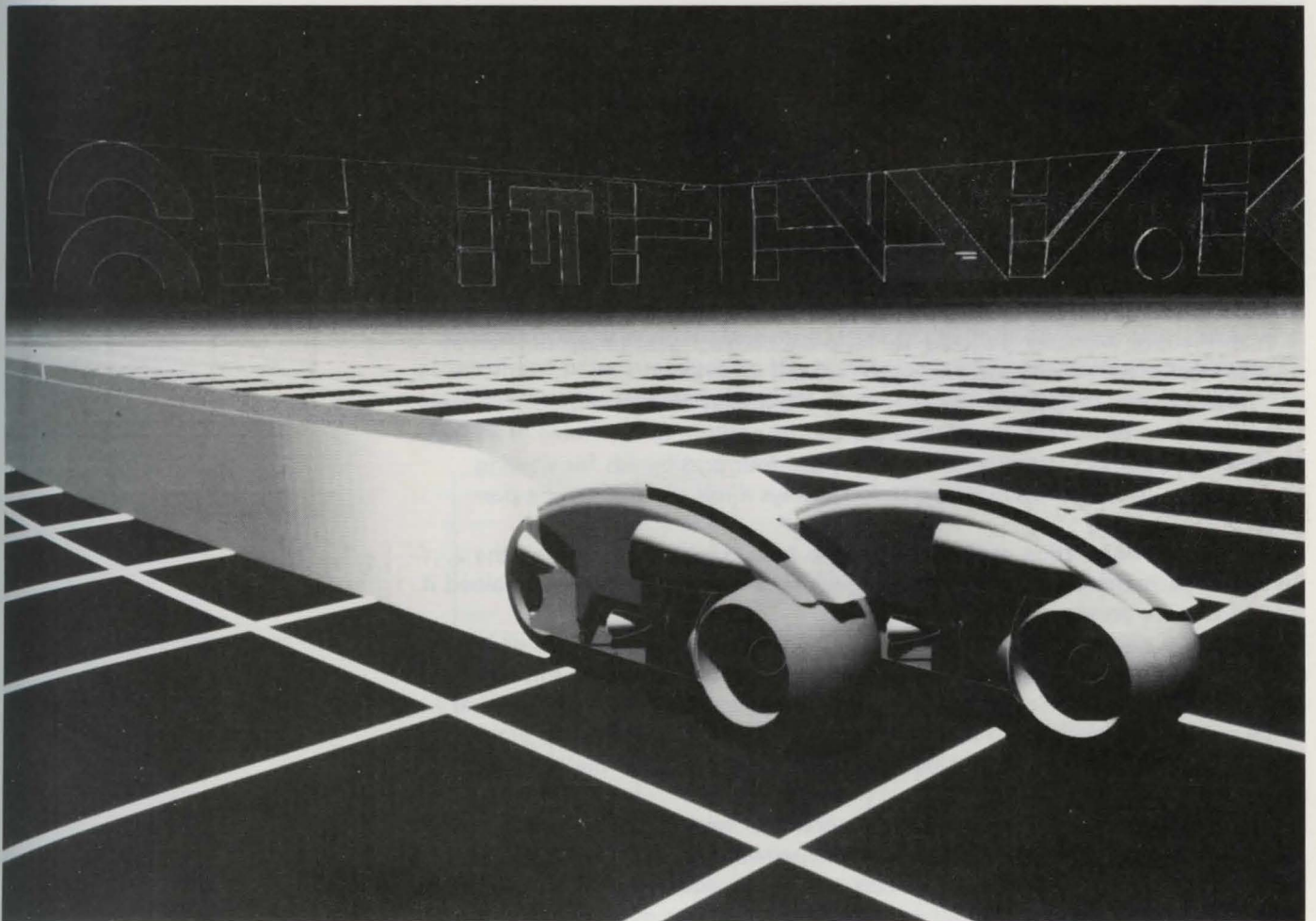
The puck is erased.	390
The visible mode is reinstated.	395
If a key has been pressed, it is entered into A\$, and subroutine 2000 is called.	400
The next location of the puck is determined and the puck is drawn.	420
The sequence 340–440 is continued until satisfied.	440
The screen is cleared, and either “Another goal scored” or “Another save” is tallied and displayed.	450
The sequence 150–490 is continued.	490
SUBROUTINE 2000:	
The hockey stick is erased.	2000
If the last key pressed was a shift-D, the stick is moved 3 units to the right.	2010
If a shift-A, three units to the left.	2020
If an unshifted d, one unit to the right.	2030
If an unshifted a, one unit to the left.	2040
Clear A\$.	2050
Return to the main program.	2060





---

## ***Realism and Animation***



Properly done graphics can make a good game great. Realism is sometimes conveyed by a static picture, but more often by accurate animation, which requires that a well-timed sequence of static scenes pass rapidly across the display screen.

If you have seen computerized cannons that shoot along a short, arched trajectory instead of a longer, more natural trajectory, or if you've seen a supposedly human figure made from a few squares, you will agree that realistic graphics are an important part of a computer game.

In this chapter, you will be introduced to some of the techniques that give a computer game polish and a finished appearance. Simple games like dice and Target Practice illustrate the use of realism, perspective, and animation.

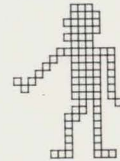
The pictures of a running cat that were first published in 1899 may seem out of place in a book on modern computer games, but they are fine studies of motion. Exactly the same process is used to animate any figure. Careful analysis of the object in motion leads to development of a rapidly processed sequence of "stills."



*Short, unrealistic trajectory*



*Long, more realistic trajectory*



*Supposedly human figure*

## Realism

Dice and cards, although too static to be well-suited to computer games, make a good starting point for a discussion of computer graphics technique because they are widely understood. Many major card and dice games have been written for computers, and as programming exercises they are interesting examples of how realism can improve a very simple game. It isn't even necessary to have a graphics output device to show spots rather than a number between 1 and 6 on the faces of the dice.

On a display screen, dice should appear three-dimensional and they should fall in a random way. It is much more attractive to give the appearance of dice rolled out on a green baize table top than simply to show one face of each die with all the dice in a line. It should be the programmer's goal to combine an attractive display with a reasonably straightforward program.

Three random factors may be applied to the display of dice without much effort—the value of the die, its orientation, and its position on the table. If all dice are displayed in this same orientation, there are four possibilities for viewing each number. The illustration shows the four ways a roll of "six" can be portrayed.

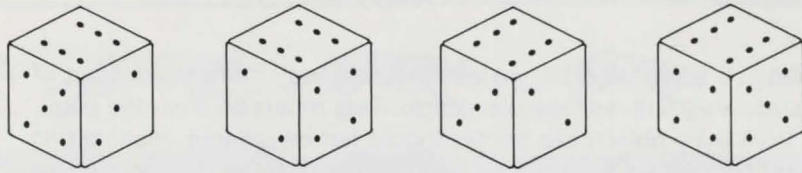
Notice the orientation of the spots on the faces displaying the 3 and the 2. Because it recreates the way the spots appear on a real die, it must be retained if realism is to be preserved.

Since it is necessary to be able to read the spots, dice should fall in such a way that they do not overlap or interfere with each other, but they still should be displayed randomly. Suppose each die is allotted a position within a vertical stripe on the tablecloth, as in the accompanying illustration. A random number may be used to place each die somewhere on the stripe to give the appearance of a real roll.

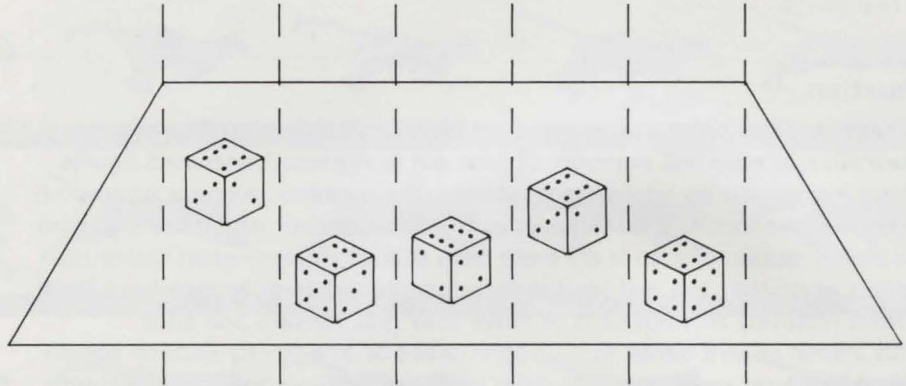
In designing a computer game, the programmer must strive to achieve the highest realism possible within the framework of hardware capability and without undue repetition. A complex and lengthy animation sequence may be fascinating to watch once or twice but should not occur time and time again in the course



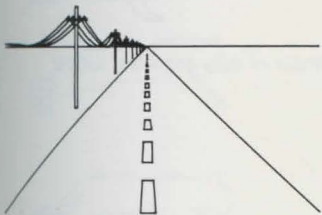
Four ways a roll of "six" can be portrayed



Simulating realism in a roll of dice



of a game. Players should not be made to wait for more than a few seconds for the computer to finish a display sequence.

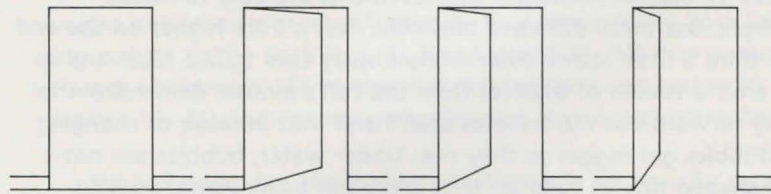


Realistic perspective

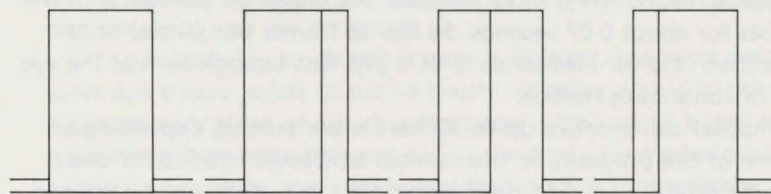
### Perspective

One of the most obvious omissions in graphics is a lack of perspective. A door should open as in the first set of doors shown here, not as in the second set, unless it is a deliberate attempt to simulate a sliding door.

Objects should appear bigger the closer they are to a viewer, as in the row of telephone poles stretching to the horizon. It is also important to control the apparent distance between the poles. A method for doing so is described in Chapter 7.



Realistic sequence of a door opening



Unrealistic sequence of a door opening

## Details

Attention to detail is essential. Most pictures look “wrong” if one pixel is out of place or inappropriately colored. Specular (mirror-like) reflection from the pupil of the eye or from shiny objects like bottles should not be ignored. Highlighting and shading, not to mention the shadows cast by objects in the display, are part of a real image.

## Animation

Early motion-picture cartoons, many of which are still shown on Saturday-morning television, are perfect examples of how not to represent animated figures. Running is obviously an intermittent motion—the runner accelerates when pushing off with one foot and slows down as the opposite foot contacts the ground. The running motion of men is different from that of women—men rotate their shoulders when they run and tend to lean forward, whereas women keep their shoulders relatively still, but tend to move their hips forward and back.

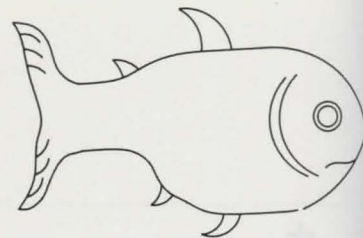
No animal or bird moves at a uniform speed or in a strictly uniform pattern either. When a cat runs, its entire upper body profile flows smoothly through a sinusoidal motion that is not so apparent in a running dog. Flying birds lose altitude when their wings are in the upward stroke. In fact, birds are difficult to animate properly because different species fly in very different ways. Big birds are easiest; they tend to soar a lot.

Fish may be easiest of all because they don’t have many moving parts. The fish shown here may be divided into three parts for animation: the head, the body, and the tail. To make the fish move forward, move the tail and body sections forward one increment without moving the head. Next, move the tail forward another increment without moving the body or head. The fish will appear to be two increments shorter. Next, move the head forward two increments and the body one while keeping the tail in place. (An increment may be one or two pixels.) Motion should be deliberate and slow for a big fish, fast and lively for a little fish.

Obviously, this is minimal animation. Improvements are easy to make, though. For example, use three different tails—the first a little higher on the end of the body, the third a little lower. Alternately display tails 1,2,3,2,1,2,3, and so on. Periodically, emit a stream of bubbles from the fish’s mouth. Remember that the fish is moving forward but the bubbles aren’t and that because of changing water pressure, bubbles get bigger as they rise. Under water, bubbles are not spherical but are shaped like an eyeglass lens, although many viewers would probably expect to see spherical bubbles.

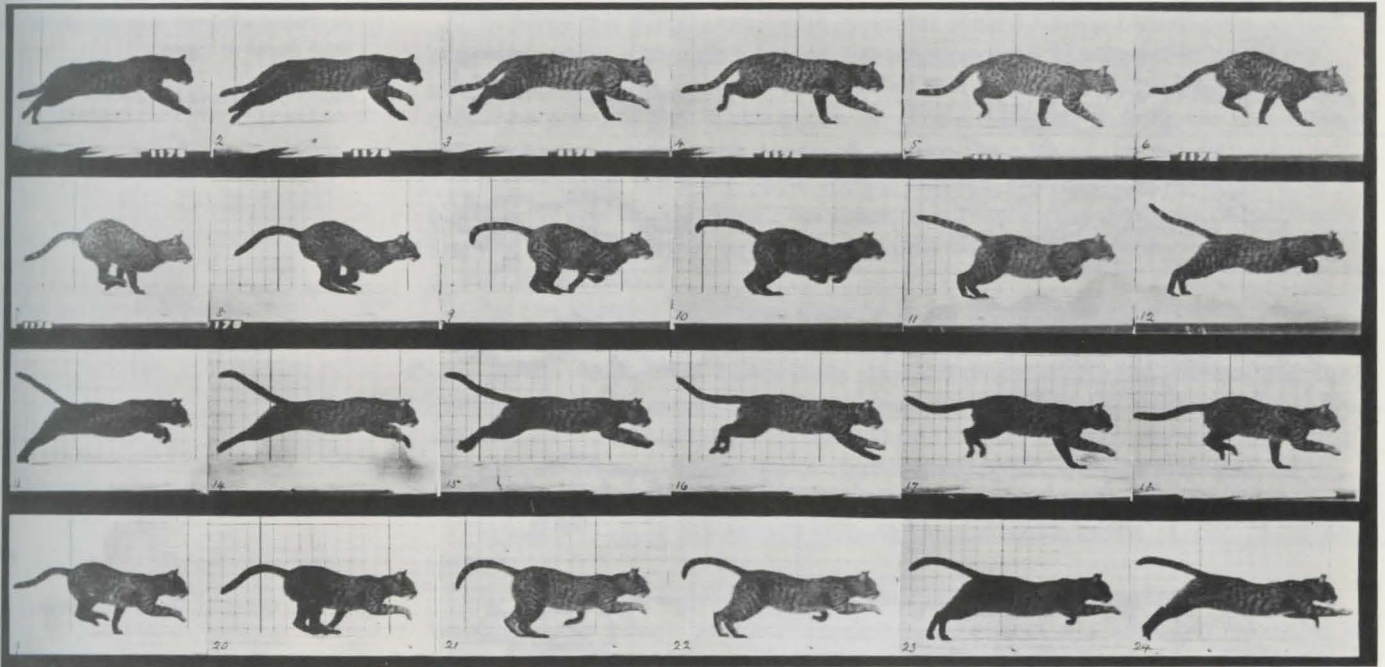
A running animal requires more than the three frames per cycle needed to animate the fish because its motion is more complex. An image on the retina of the human eye persists for about 0.07 seconds. So the 16 frames per second once used as the projection rate for motion pictures is just fast enough to fool the eye into the illusion of continuous motion.

A typical computer can produce up to 30 frames per second, depending on the execution time of the program, so the number of frames needed for one motion cycle can be established. One running cycle for a human may be defined

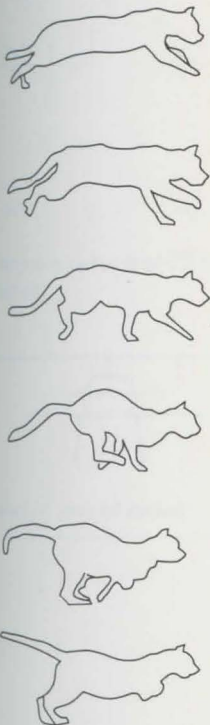


*Example of easy animation subject*





Original Muybridge running cat photograph



Outlines of selected Muybridge frames

as beginning when his right foot touches the ground and ending two paces later, when his right foot reaches the same position.

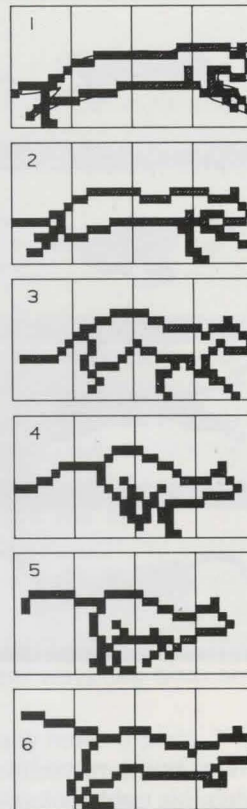
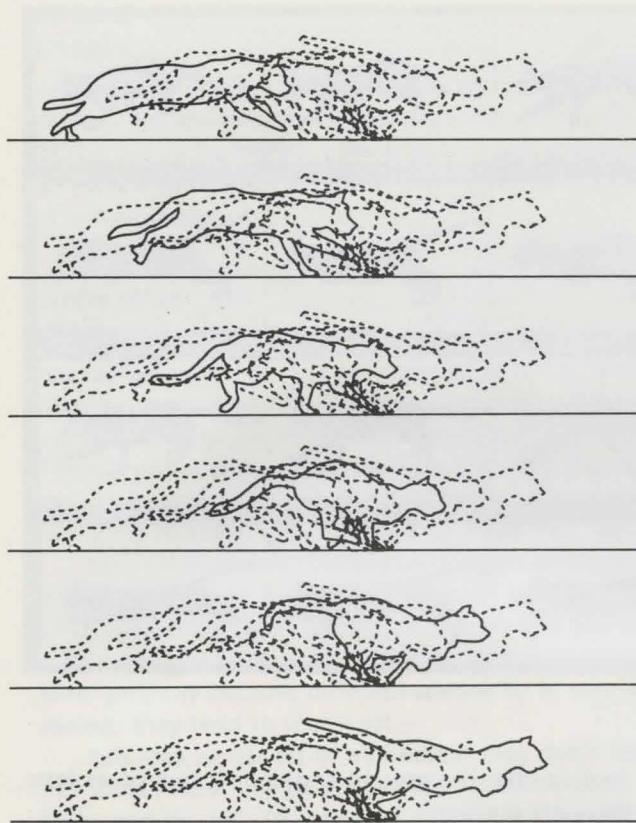
If a cat executes three running cycles per second and the program can produce 18 frames per second, it is necessary to define a running cycle in 6 frames. The motion of a running cat can be analyzed from slow-motion movies or a sequence of pictures, such as the beautiful sequence photographed by Eadward Muybridge over eighty years ago. These photographs, taken by 24 cameras electrically timed to shoot at short intervals, provide 12 images per cycle.

In frames 1 and 2, the cat is doing a desultory leap, followed by bringing its weight onto the front legs (frames 3–5), gathering the back legs (6–7), transferring weight to the back legs (8), and lunging (9–12) into a much better leap, (frames 13 through 16). Notice that the cat does not gather its legs very well (frames 18–21) and consequently the next leap is not going to be very impressive.

When Muybridge photographed animals in motion, he expected the pictures to be of interest to artists, so he used a rectangular wire grid as a background to provide a means of determining the scale of the picture. As it turns out, the grid can be used to "digitize" the picture—to convert it into a numerical representation for input into the computer.

The easiest way to do this is with a digitizer, but most personal computer users don't have access to such a luxury. Another way is to put a sheet containing a picture on a piece of square-ruled paper with carbon paper in between and trace the picture onto the grid. Using the grid to provide coordinates, choose important points on the picture and find their X,Y coordinates, which can then be used with a standard routine such as DRAW A TRANSLATED FIGURE.





*Animated sequence with feet aligned from frame to frame*

*Running sequence at very low resolution*

In the next picture, the cat was first digitized; then wherever a continuous smooth curve was present, a curve was fit to the points. There are 15 curves on this cat, plus a straight line at the base of one foot and a line and circular arc describing the ear. To draw the cat using straight lines, it may be advisable to digitize a few more points.

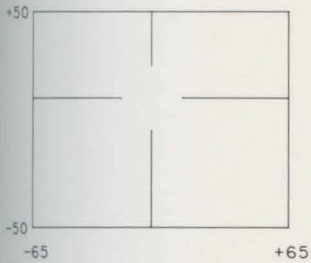
You may create the basis for a running sequence by making digitized outlines of selected frames from the Muybridge photographs. The example shown here used frames 1, 3, 5, 7, 9, and 11.

When the cat is set in motion on the screen, it is essential to align the pictures so that the positions of those feet contacting the ground do not vary from one frame to the next, as shown in the second sequence of running cat drawings.

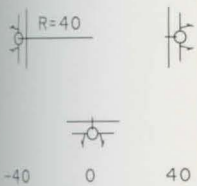
Notice that this technique is applicable to very low resolution—the resulting cat is clearly recognizable.

### **Target Practice Game**

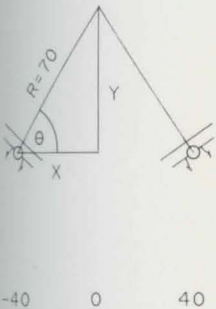
Target Practice is a simple game that makes use of animation as well as the rotation, translation, and scaling subroutines described in Chapter 2. Imagine yourself as the pilot of a World War I biplane in the air over France. In front of you is an enemy plane, banking from side to side in a vain effort to escape your twin machine guns. When you fire at the right moment, the enemy will bank into the stream of bullets and be shot down.



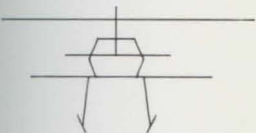
Crosshair gunsight in Target Practice game



Pendulum radius  $R$  of 40 for swing of aircraft



Pendulum radius  $R$  of 70 for swing of aircraft



Head on view of aircraft

To make this game work, your computer should have a button on a joystick or game paddle, although a key on the keyboard can also be used. When the button is pressed, it causes execution of a subroutine that sets a "shooting" flag only if no previous shot is still in the air. This is necessary to keep the player from firing continuously until the enemy strays into the path of the bullets.

Two features make the game appear realistic. First, the enemy airplane, as it banks from side to side, must slow down as it reaches the extremes of its pendulum-like motion. Second, bullets must appear to get smaller and closer together as they move away.

We can imagine the enemy airplane to be swinging like a pendulum from a point above the center of the screen. If FNX and FNY are set up to define a screen from  $-65$  to  $+65$  horizontally and  $-50$  to  $+50$  vertically, and a crosshair gunsight is shown, as illustrated here, then the center of the sight is at  $0,10$ . The enemy airplane can be made to swing from  $-40$  horizontally, through the point  $0,10$  to  $+40$  along a circular arc. A "pendulum" radius  $R$  must, therefore, be at least 40 to allow the aircraft to swing this far to either side.

In order to cause the aircraft to bank less steeply, a radius of, say, 70 can be chosen.

To compute the pendulum's location in  $X$ , a sine function can be used. Letting the angle  $\alpha$  begin at 0 and increase by 5 degrees at a time, the  $X$ -location is found by the equation:

$$X = 40 \sin(\alpha)$$

Since  $R$  is known, the  $Y$  value can be computed from Pythagoras' theorem:

$$Y^2 + X^2 = R^2$$

or

$$Y = \text{SQR}(R^2 - X^2);$$

The angle at which the aircraft is banked is  $\theta$ , the angle whose tangent is  $Y/X$ .

A simple biplane can be drawn using

```
DATA 2,-10,2.5,10,2.5
DATA 2,0,3,0,0
DATA 2,-4,0,4,0
DATA 6,2,-1.5,2.5,-0.5,1.5,1,-1.5,1,-2.5,-0.5,-2,-1.5
DATA 2,-7,-1.5,7,-1.5
DATA 2,-2.5,-1.5,-3.5,-5
DATA 2,-3.75,-4.5,-3.25,-5.5
DATA 2,2.5,-1.5,3.5,-5
DATA 2,3.75,-4.5,3.25,-5.5
DATA -9999
```

The following program will draw the crosshairs and show the biplane banking from side to side.

### Target Practice Program Code

PAGE	120
MOVE FNX(-65),FNY(10) ! Draw crosshairs	130
DRAW FNX(-15),FNY(10)	140



MOVE FNX(15),FNY(10)	150
DRAW FNX(65),FNY(10)	160
MOVE FNX(0),FNY(-50)	170
DRAW FNX(0),FNY(-5)	180
MOVE FNX(0),FNY(25)	190
DRAW FNX(0),FNY(50)	200
R = 70           ! Initialize	210
G = 0	220
F = 0	230
L = 0	240
REPEAT	250
X = 40*SIN(G)           ! Compute position of target	260
Y = 70 - SQR(R↑2 - M↑2)	270
RESTORE 660           ! Prepare X9, Y9, S9, and T	280
X9 = X               ! for drawing target	290
Y9 = Y + 10	300
S9 = 0.5	310
IF X = 0 THEN	320
T = 0	330
ELSE	340
T = ATN(Y/X)*180/PI	350
;	360
GOSUB 760           ! Draw target using ROTATE A FIGURE	370
G = G + 5*PI/180   ! Add 5 degrees to bank angle	380
UNTIL B = 1	390

**Target Practice Code Description**

	Sequence Number
Clear the screen and draw crosshair sight.	120
Set R, the banking radius of the enemy aircraft, to 70.	210
Set G, F, and L to zero. G is an angle that will be used to swing the enemy aircraft from side to side. F is a "guns firing" flag. Guns are firing when F = 1. L is a scaling factor for bullets. (F and L are not used yet.)	220
Compute the location of the enemy aircraft.	260
Prepare to read the definition of the aircraft.	280
Set translation values X9 and Y9. Since the lowest Y-value will be zero, add 10 to cause the pendulum arc to pass through the center of the crosshairs at 0,10.	290
Set drawing scale S9 at 0.5.	310
In computing the angle $\theta$ (the variable T), the value Y/X is undefined where X = 0. But at X = 0, T should be 0, so test for X = 0 and if so, set T = 0. If not, compute T as the angle whose tangent is Y/X radians, multiplied by $180/\pi$ to convert T to degrees.	320
Call ROTATE A FIGURE.	370
Add 5 degrees to G.	380
Loop again through statements 250-390.	390

If your computer is the bit-map type, you will notice that instead of drawing a single airplane moving from side to side, the program is drawing airplanes at every step along the pendulum. To eliminate this difficulty it will be necessary to



erase the aircraft before redrawing it in a new position. One way to erase is to draw the airplane in black pixels—that is, select the color "black," set appropriate values of X9, Y9, S9, and T and call ROTATE A FIGURE after line 250.

The way to shoot at the enemy airplane also depends on the characteristics of your computer. The variable L will be used to establish the location of fired bullets, so when L = 0, no bullets are flying. Add after line 250 the equivalent of

```
IF button pushed THEN
  GOSUB 1000
;
```

Subroutine 1000 should be

```
IF L = 0 THEN
  F = 1
  L = 2
;
RETURN
```

1010  
1020  
1030  
1040

This code says, in effect, "If the firing button has been pushed and bullets are not flying, set F, the 'bullets flying' flag and L, the bullet location factor." Now, the code can be written to display six bullets and to test for a hit. (See Chapter 7 and the *Appendix* for a discussion of perspective.)

### Target Practice Revised Program Code

Sequence  
Number

```
REPEAT
  B = 0
  PAGE
  MOVE FNX(-65),FNY(10)
  DRAW FNX(-15),FNY(10)
  MOVE FNX(15),FNY(10)
  DRAW FNX(65),FNY(10)
  MOVE FNX(0),FNY(-50)
  DRAW FNX(0),FNY(-5)
  MOVE FNX(0),FNY(25)
  DRAW FNX(0),FNY(50)
  R = 70
  G = 0
  F = 0
  L = 0
REPEAT
  X = 40*SIN(G)
  Y = 70 - SQR(R^2 - X^2)
  RESTORE 660
  X9 = X
  Y9 = Y + 10
  S9 = 0.5
  IF X = 0 THEN
    T = 0
  ELSE
    T = ATN(Y/X)*180/PI
  ;
  GOSUB 760
  G = G + 5*PI/180
```

100  
110  
120  
130  
140  
150  
160  
170  
180  
190  
200  
210  
220  
230  
240  
250  
260  
270  
280  
290  
300  
310  
320  
330  
340  
350  
360  
370  
380

! Turn off "target hit" flag  
! Draw crosshairs  
! Initialize  
! Compute position of target  
! Prepare X9, Y9, S9, and T  
! for drawing target  
! Draw target using ROTATE A FIGURE  
! Add 5 degrees to bank angle

```

IF F <> 0 THEN      ! If bullets are flying,          390
  L = L/2           ! compute their Y-location        400
  RESTORE 590       ! and prepare variables           410
  T = 0             ! for drawing bullets             420
  S9 = L            430
  X9 = 0            440
  Y9 = 10 - 60*L    450
  GOSUB 760         ! Draw bullets                   460

  IF Y = 0 AND L > 0.015 AND L < 0.03 THEN ! Test if bullets 470
    PRINT "BANG..." ! hit target                   480
    B = 1          490
  ;               500

  IF L < 1/64 THEN ! If bullets reached horizon,      510
    F = 0          ! reset flags                     520
    L = 0          530
  ;               540
;               550

UNTIL B = 1         ! Continue until target is hit    560
UNTIL B <> B         ! Start again when target is hit  570
END                580

DATA 2,30,0,26.25,7.5 ! Bullets                     590
DATA 2,22.5,15,20.625,18.75 600
DATA 2,18.75,22.5,16.875,26.25 610
DATA 2,-30,0,-26.25,7.5     620
DATA 2,-22.5,15,-20.625,18.75 630
DATA 2,-18.75,22.5,-16.875,26.25 640
DATA -9999                 650

DATA 2,-10,2.5,10,2.5      ! Target                 660
DATA 2,0,3,0,0             670
DATA 2,-4,0,4,0            680
DATA 6,2,-1.5,2.5,-0.5,1.5,1,-1.5,1,-2.5,-0.5,-2,-1.5 690
DATA 2,-7,-1.5,7,-1.5     700
DATA 2,-2.5,-1.5,-3.5,-5  710
DATA 2,-3.75,-4.5,-3.25,-5.5 720
DATA 2,2.5,-1.5,3.5,-5     730
DATA 2,3.75,-4.5,3.25,-5.5 740
DATA -9999                 750

C0 = COS(T*0.0174533) ! Rotate, translate, and scale a figure 760
S0 = SIN(T*0.0174533) 770
REPEAT                  780
  READ J                790

  IF J >-999 THEN       800
    READ X8,Y8          810
    MOVE FNX((X8*C0 -   820
Y8*S0)*S9 + X9),FNY((X8*S0 + Y8*C0)*S9 + Y9
    REPEAT              830
      READ X8,Y8
      DRAW FNX((X8*C0 -   860
Y8*S0)*S9 + X9),FNY((X8*S0 + Y8*C0)*S9 + Y9)
      J = J - 1         870
    UNTIL J = 1        880

```

UNTIL J < -999	890
RETURN	900
	910

### Target Practice Revised Code Description

Crosshairs are drawn as before.	Sequence Number 120
The enemy aircraft's position is computed and the aircraft is drawn.	260
The "bullets flying" flag is tested.	390
Bullet location factor L is halved.	400
The scale of the drawn bullets and their location on the screen are determined by L.	430
Bullets are drawn. Notice that L, which was set to 2 in Subroutine 1000, is halved each time Statement 400 is executed, so L takes on the values 1, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, and so on.	460
For the bullets to hit the target, the Y-location of the aircraft must be Y = 0 when the bullets reach it, which is when L = 0.015625. Because numerical precision in computers varies, a better test for bullet location is when L is at least 0.015 but less than 0.03.	470
If the bullets hit the enemy aircraft, print "BANG..." and set "target hit" flag B.	480
If L is less than 0.015625 (or 1/64), the bullets missed. Turn off the "bullets flying" flag F and reset the bullet location factor L to 0.	510
If B is not 1, the enemy aircraft has not yet been hit. Continue to process from line 250.	560
If the aircraft was hit, begin again at line 100.	570

### Refinements

- Instead of printing "BANG..." when the enemy aircraft is hit, show it "going down in flames" by reducing Y by 5 units at a time until Y = 0. This is easy to do: change line 270 to

$Y = 70 - \text{SQR}(R \uparrow 2 - X \uparrow 2) - 5 * B$  270

Delete line 480 and change line 560 to

UNTIL Y = 0 ! Continue until plane goes down 560

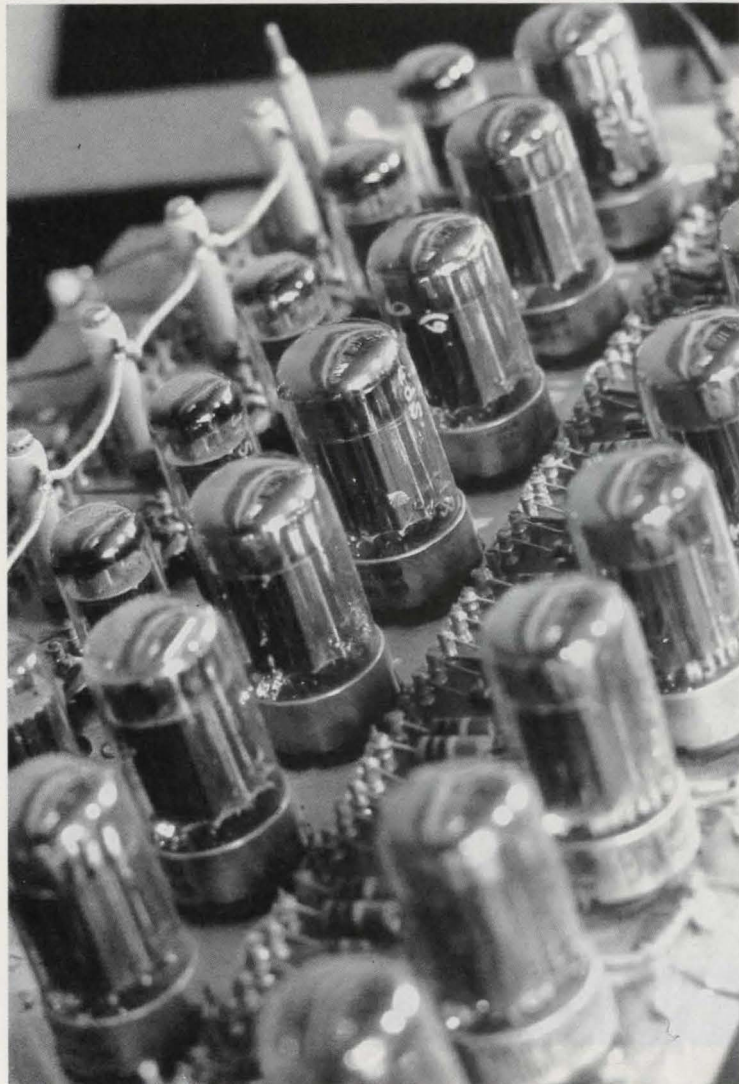
- Add a trail of smoke as the plane goes down.
- Erase the previous set of bullets before drawing the next set.
- Randomly vary the extents of the pendulum swing of the enemy aircraft.
- Keep score. Display hits and misses at the top of the screen, for example:

3 HITS  
2 MISSES





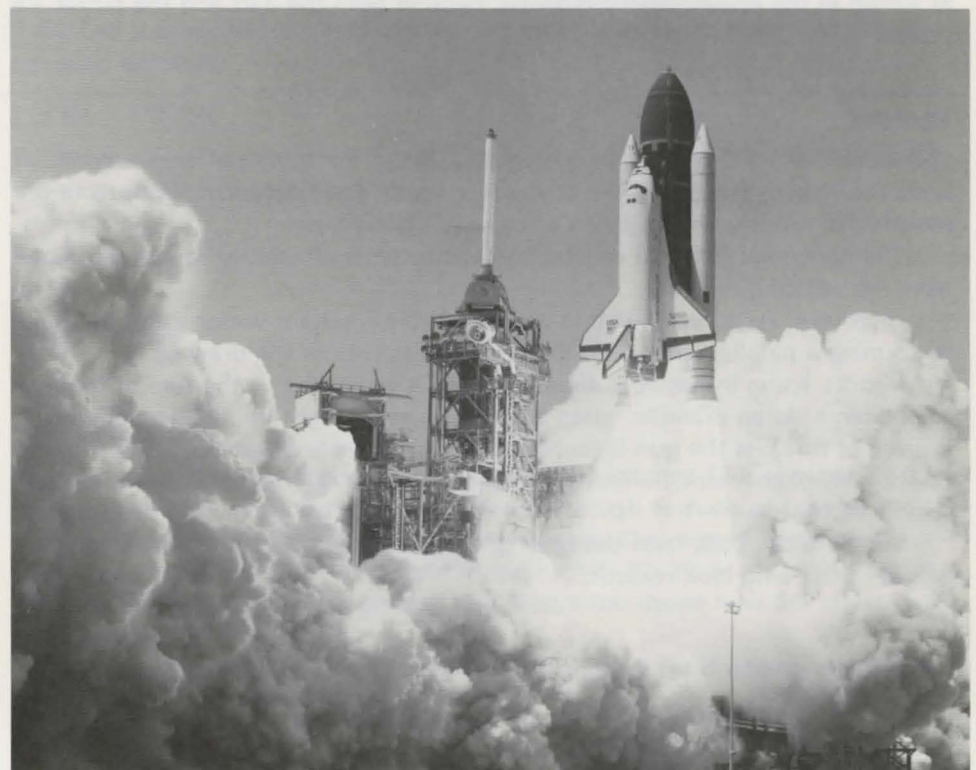
## ***Some Familiar Games***







## Ballistic Trajectory Games



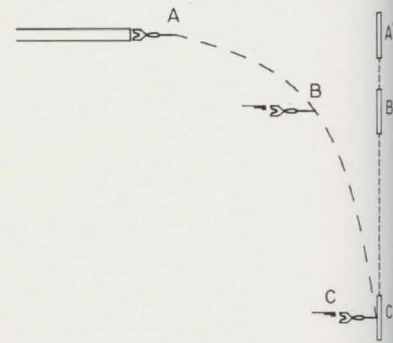
Target practice, in one form or another, is the basis for many games. Over the centuries, people have thrown rocks and spears and fired arrows and bullets at targets, sometimes to practice for hunting or war and sometimes just for the fun of it. Many modern games are obviously derived from target practice—darts, for example, and just about anything played with a ball.

Lots of computer and arcade games also involve batting an object or shooting at things. To achieve a realistic effect, the programmer must take into account the physical laws that apply to objects in motion. A brief discussion of gravity and vectors is followed by examples using a dropped and bouncing ball. The chapter ends with a simple cannon shooting game program and some suggested refinements.

## Gravity

Except for the effect of friction and lift on a body as it flies through the air, all propelled objects that are not self-powered behave according to the laws that pertain to ballistic (free fall) trajectories. Expressed simply, an object in a vacuum is subject to the force of gravity in the "downward" direction. If it is also in motion in another direction, that motion component is not altered because of gravitational force.

There is a classical demonstration of this fact in which a dart is fired at a target. The speed of the dart can be varied, but the gun is always horizontal. When the trigger is pulled, the target begins to fall. The dart will always hit the target, as shown in the accompanying drawing, because both the dart and the target are subject to the same downward force.



Force of gravity equal on dart and falling target

## Vectors

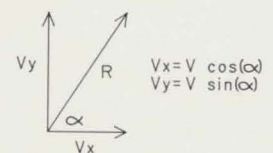
The motion of an object can be shown as a vector, a line whose length represents the velocity (speed) of the object and whose direction corresponds to the direction of motion, shown by an arrowhead on the line.

The horizontal ( $V_x$ ) and vertical ( $V_y$ ) components of velocity can be calculated from the adjacent diagram.

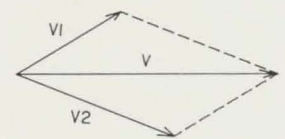
Given two vector components  $V_1$  and  $V_2$ , the resultant vector  $V$  can be found by forming a parallelogram.  $V$  is the diagonal, as seen in the drawing.

Vectors acting in opposite directions yield a vector that is the sum of the two components. As an example, when a bullet is shot straight up, the acceleration it is given as it leaves the gun is counteracted by the force of gravity acting straight down. (This phenomenon is illustrated here.) Thus if  $V_1 = 2$  and  $V_2 = -3$ , the resultant vector is  $V = -1$ .

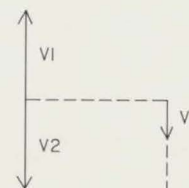
Vector calculations form the basis of ballistic trajectories and make computer games of this kind look realistic.



Calculating horizontal and vertical components of velocity



Calculating vector  $V$  from vectors  $V_1$  and  $V_2$



Typical result of opposing vectors. If  $V_2 > V_1$ , the resulting vector  $V = V_2 - V_1$

### Dropping a Ball

Vertical motion of an object may be described in terms of its height, the distance it has travelled, or its velocity. These equations all depend on time, which is continuous, not discrete. The only way it can be dealt with in a program, however, is in discrete intervals. Selection of a proper time interval produces realistic motion—the object will appear to fall neither too fast nor too slowly.

The equations of motion may be developed from a constant  $G$ , the force due to gravity, and from six variables:

$T$  Elapsed time

$Y$  Height of a falling object at time  $T$

$H_0$  Height at which the object started to fall

$V_0$  Velocity at which the object started to fall

$T_0$  Time at which the object reaches the ground

$T_1$  A time interval, say 0.1 seconds

On the earth,  $G$  is approximately 32 feet per second per second—that is, the force  $G$  increases the velocity of a falling object by about 32 feet per second every second.

When an object is dropped or thrown upward, it behaves according to an equation that gives the height of the object at any time after it is released:

$$Y = H_0 + V_0 * T - G * T^2 / 2$$

If the ball is dropped from a height  $H_0$ , the initial velocity  $V_0$  is zero, so the equation becomes

$$Y = H_0 - G * T^2 / 2$$

To find the time  $T_0$  at which the ball reaches the ground, it is necessary to find the time at which  $Y$  becomes zero. So

$$0 = H_0 - G * T_0^2 / 2$$

gives

$$T_0^2 = 2 * H_0 / G$$

or

$$T_0 = \text{SQRT}(2 * H_0 / G)$$

To show the ball falling on the screen, a short time interval  $T_1$  is chosen, say 0.1 seconds, and how much farther the ball drops through each equal time period will give the illusion of the ball's increasing velocity. The motion equation is evaluated for  $T = 0$ ,  $T = T_1$ ,  $T = 2 * T_1$ , and so forth, until  $T > T_0$  or  $Y \leq 0$ , at which time the ball has reached the ground. The illustration shows how the effect could be drawn.

Here are the code and description of the algorithm for dropping a ball from 80 feet.



**Dropping a Ball Program Code**

	Sequence Number
! CLEAR THE SCREEN	110
PAGE	120
! DRAW A LINE ACROSS THE SCREEN AT Y = 0	130
MOVE 0,0	140
DRAW 130,0	150
! ASSIGN INITIAL VALUES	160
HO = 80 ! Ho is the initial height of the ball	170
T = 0 ! T is elapsed time	180
T1 = 0.1 ! T1 is a time interval	190
VO = 0 ! VO is the initial velocity	200
G = 32 ! G is the force of gravity	210
X = 1 ! X is the object's horizontal position	220
REPEAT	240
Y = HO + VO*T - G*T <sup>2</sup> /2 ! Y is ball height at time T	250
! DISPLAY BALL AT X,Y	260
IF Y >= 0 THEN ! As long as ball is above ground,	270
GOSUB 500 ! erase last ball and draw this one	290
;	300
! INCREASE TIME BY ONE INTERVAL	310
T = T + T1 ! Increase elapsed time by one interval,	320
UNTIL Y < 0 ! continue until ball reaches ground	330
END	340

**Dropping a Ball Code Description**

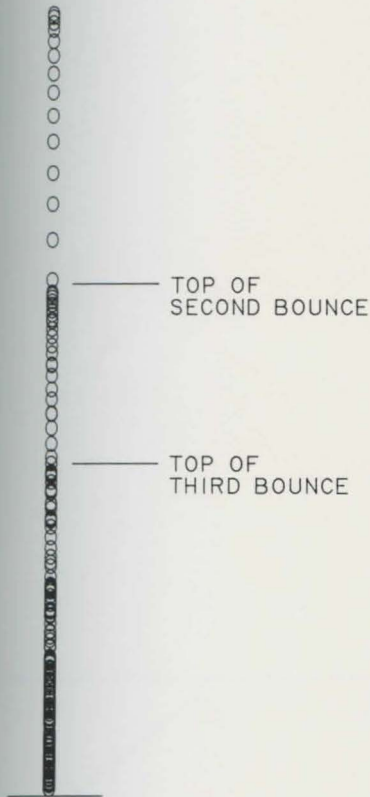
	Sequence Number
After clearing the screen, a horizontal line is drawn representing the ground.	140
Initial variables are set.	170
Height Y of the ball at time T is computed.	250
If the ball has not reached the ground, Subroutine 500 erases the last position of the ball and redraws it at Y.	270
Elapsed time is increased by one interval.	320
The program continues from line 240 until the ball reaches the ground.	330

**Bouncing Ball**

Examine the values of T and To at the end of the previous program,

T = 2.4  
T0 = 2.236

The ball actually reached the ground 0.164 seconds before the end of the last time interval. If the ball had bounced, it should next be displayed at the point X,Y where Y was the ball's position as of 0.164 seconds after it struck the ground. To do so, start T for the bounce at



The effect of the coefficient of restitution

$$T = T - T_0$$

so now  $T = 0.164$  and  $T_0 = 2.236$ , the time at which the ball first struck the ground.

The situation is different for the beginning of the second bounce.  $H_0$  is now zero and  $V_0$  is a function of the velocity of the ball when it struck the ground ( $V_{t0}$ ). Since a ball never rebounds with all its original velocity, we can say that  $V_0 = V_{t0} * R$ , where  $R$  is the coefficient of restitution ("bounciness"), a number between 0 and 1. If the rebound velocity were 80 percent of  $V_{t0}$ , then  $R$  would be  $80/100$ , or  $0.8$ .  $V_{t0}$  is found by multiplying  $G$  by the time taken for the ball to fall from the top of its last bounce, in this case  $T_0$ . So

$$\begin{aligned} V_{t0} &= G * T_0 \\ \text{and} \\ V_0 &= V_{t0} * R \\ \text{or} \\ V_0 &= G * T_0 * R \end{aligned}$$

There is also a difference in the use of  $T$  and  $T_0$  now that the ball is bouncing. Since  $T_0$  is the time from the top of the ball's trajectory, the total amount of time for one bounce is  $2 * T_0$ . So to find the initial position of the ball after the second bounce, you must use

$$T = T - 2 * T_0$$

It is now a minor matter to cause the bouncing ball to move across the screen. Ignoring friction, horizontal motion is a constant. To move the ball sideways, add a line after line 420:  $X = X + N * T_1$ . Define  $N$  after line 170 such that  $N$  is the horizontal velocity in feet per second, for example,  $N = 5$ . See the illustration of how this motion could be portrayed on a screen and then study the code that follows.

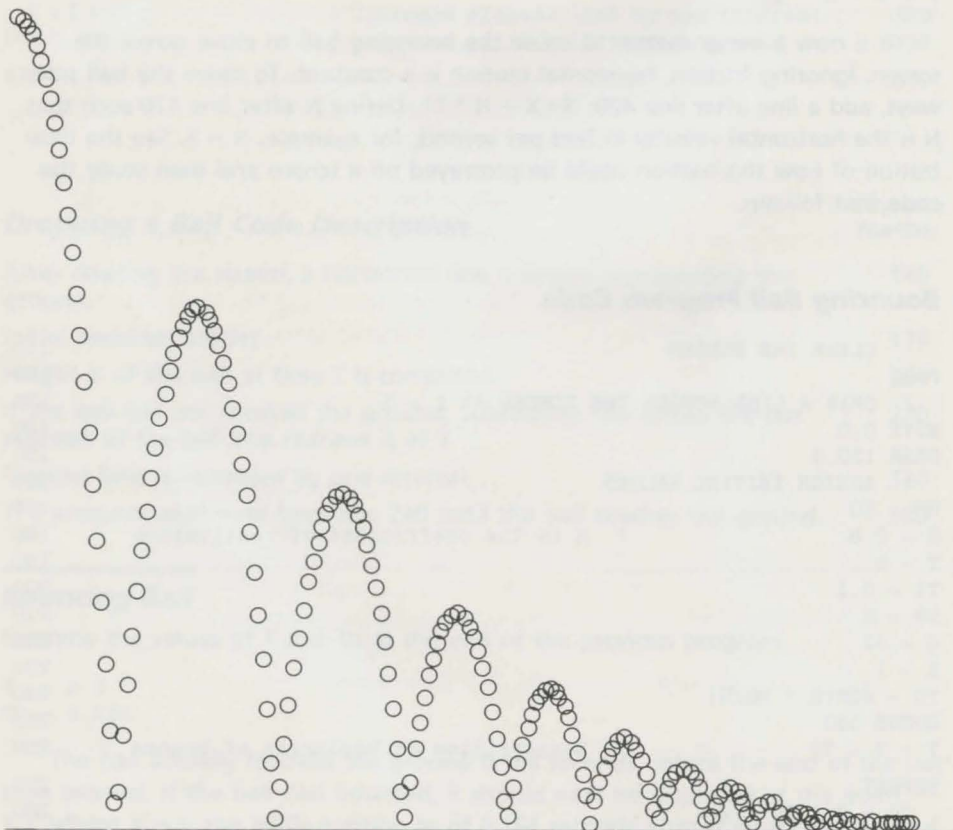
### Bouncing Ball Program Code

```

! CLEAR THE SCREEN                                     110
PAGE                                                    120
! DRAW A LINE ACROSS THE SCREEN AT Y = 0              130
MOVE 0,0                                                140
DRAW 130,0                                              150
! ASSIGN INITIAL VALUES                               160
H0 = 80                                                  170
R = 0.8                                                  180
! R is the coefficient of restitution
T = 0                                                    190
T1 = 0.1                                                200
V0 = 0                                                  210
G = 32                                                  220
X = 1                                                    230
T0 = SQR(2 * H0/G)                                     240
GOSUB 350                                               250
T = T - T0                                              260
! Restart time at beginning of bounce
REPEAT                                                  270
  V0 = G * T0 * R                                       280
  H0 = 0                                                290
  T0 = V0/G                                             300

```

GOSUB 350	310
T = T - 2 * T0	320
UNTIL V0 < 1	330
END	340
! FIND HEIGHT Y AT TIME T AND DISPLAY BALL	350
REPEAT	360
Y = H0 + V0 * T - G * T <sup>2</sup> /2	370
! DISPLAY BALL AT X,Y	380
IF Y >= 0 THEN	390
GOSUB 500 ! Erase last ball and draw new ball at X,Y	410
T = T + T1	420
;	430
! INCREASE TIME BY ONE INTERVAL	440
UNTIL Y < 0	460
RETURN	470

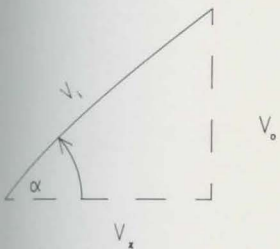


Portraying vertical velocity and horizontal motion of bouncing ball



**Bouncing Ball Code Description**

	Sequence Number
Having cleared the screen and drawn the ground line, initial values are assigned, including R, the coefficient of restitution.	110
Height at time T is computed in Subroutine 350, and the ball is displayed, until the ball reaches the ground.	250
Time is reset to the actual bounce time.	260
Initial upward velocity after the bounce $V_0$ is computed, initial height $H_0$ is set to 0, and the time at the bounce $T_0$ is set.	280
The next bounce is displayed by calling Subroutine 350.	310
The program computes another bounce if the velocity of the ball exceeds one foot per second.	330
<b>Subroutine 350</b>	
Height of the ball is computed.	370
If the ball is above the ground, the previous ball position is erased and the new ball is displayed.	410
Time is increased by one interval.	420
If the ball has not reached the ground, continue processing at line 370.	460
Otherwise, return to the main program.	470



Computing ballistic trajectory

**Cannon Shooting Game**

The rules of motion that apply to the bouncing ball may be applied very easily to a large number of games in which ballistic trajectories are needed. In general, a ball thrown at an angle or a bullet shot from a gun is not likely to be moving straight up or straight out. If a gun is fired at an angle, a component of the initial velocity  $V_i$  must be assigned to the constant horizontal motion  $V_x$  and the other component assigned to the initial vertical velocity  $V_0$ , as shown in the illustration.

Here,  $\alpha$  is the angle of the gun, and  $V_i$  is the known initial velocity of the projectile. If  $\alpha$  expressed in radians ( $2\pi$  radians = 360 degrees), then

$$V_0 = V_i * \sin(\alpha)$$

and

$$V_x = V_i * \cos(\alpha)$$

Since most people think in terms of degrees,  $\alpha$  may be entered in degrees and converted to radians by

$$\alpha = \alpha * 2\pi / 360$$

or

$$A = A * \pi / 180$$

or

$$A = A * 0.0174533$$

Imagine a simple game in which a gun is shown from the side of the screen and on the other side of the screen a blockhouse is drawn. The player enters an angle between 0 and 45 degrees. The barrel of the gun moves to the given angle and fires a shot. If the shot lands on the blockhouse, the game is over. If the shot misses, the player enters another angle and fires again until the blockhouse is hit.

**Cannon Shooting Game Program Code**Sequence  
Number

```

REPEAT                                                    100
  PAGE                ! Clear screen                    110
  S9 = 1              ! Set initial values              120
  V = 1000                                                    130
  T1 = 0.1                                                    140
  T = 0                                                       150
  G = 32                                                      160
  X1 = 124            ! Blockhouse location             170
  X2 = 130                                                    180
  Y1 = 0                                                      190
  Y2 = 6                                                      200
  MOVE 0,0           ! Draw level ground               210
  DRAW 130,0                                                 220
  RESTORE 970        ! Draw blockhouse                 230
  GOSUB 1040         ! using simple DRAW routine       240
  HOME              ! Move to top left corner of screen 250

REPEAT                                                    260
  PRINT 'ENTER ANGLE OF CANNON (0 TO 45):GG';           270
  INPUT A            ! Enter angle of cannon           280
  UNTIL A >= 0 AND A <= 45                               290

  X9 = 6.5           ! Set translation values          300
  Y9 = 2                                                     310
  RESTORE 910        ! Draw cannon barrel             320
  GOSUB 1190         ! using DRAW ROTATED FIGURE      330
  RESTORE 950        ! Draw gun body                  340
  GOSUB 1040         ! using simple DRAW              350
  X0 = 6.5           ! Draw cannon wheel             360
  Y0 = 2                                                     370
  M1 = 2                                                     380
  M2 = 2                                                     390
  A1 = 0                                                     400
  A2 = 2 * PI                                               410
  GOSUB 1360         ! using ELLIPSE routine          420
  M1 = 3             ! and draw shield                430
  M2 = 3                                                     440
  A1 = - 15 * 0.0174533                                     450
  A2 = 75 * 0.0174533                                       460
  GOSUB 1360                                                470
  RESTORE 930        ! Draw muzzle flash             480
  GOSUB 1190         ! using DRAW ROTATED FIGURE      490
  X0=6.5+5.5*COS(T*0.0174533)-SIN(T*0.0174533) ! Set    500
  H0=2+5.5*SIN(T*0.0174533)+COS(T*0.0174533) ! initial 510
  V0 = V * T1 * SIN(T * 0.0174533)             ! values for 520
  D = V * T1 * COS(T * 0.0174533)              ! shell travel 530

REPEAT                                                    540
  T0 = T0 + T1                ! Increase time          550
  Y = H0 + V0 * T0 - G * T02/2 ! and locate        560
  X = X0 + D * T0              ! the shell             570

  IF X < 128 AND Y > 0 THEN    580
    GOSUB 1010                ! Erase and redraw the shell if on screen 590
  ;                            600

```

```

UNTIL Y < 0 OR X >= 128 ! until at ground level or off screen 610
T8 = V0/G + SQR((2 * H0 + V0↑2/G)/G) ! Compute values 620
X = X0 + D * T8 ! to determine if 630
T8 = ((X2 + X1)/2 - X0)/D ! shell hit 640
H = H0 + V0 * T8 - G * T8↑2/2 ! blockhouse 650

IF X >= X1 AND ((Y1 <= H AND
Y2 >= H) OR (X <= X2)) THEN 660
    X9 = (X2 + X1)/2 ! If so, 670
    Y9 = 0 ! draw shell burst 680
    T = 0 690
    RESTORE 990 700
    GOSUB 1190 710
    MOVE 25,80 720
    PRINT "BANG!GGG" 730
ELSE 740
    IF X < X1 THEN ! If shell missed 750
        X9 = X ! but stayed on screen 760
        Y9 = 0 770
        RESTORE 990 ! draw shell burst 790
        GOSUB 1190 800
    ; 810
    MOVE 25,80 820
    PRINT "MISSED..." 830
    ; 840
    I = 1 850
    REPEAT 860
        I = I + 1 ! Wait a short time 870
    UNTIL I = 500 880
    UNTIL G < > G 890
    ! BARREL 900
    DATA 4,2,0,5.5,0,5.5,2,1,2,-9999 910
    ! FLASH 920
    DATA 7,12.5,3.75,16,5,15,3.5,18,3,14.5,2.5,15.5,1,12.5,2.25,-9999 930
    ! GUN BODY WITHOUT BARREL 940
    DATA 6,5,1,3,0,0,0,0,0.5,3,1.5,4.5,2.5,-9999 950
    ! HOUSE 960
    DATA 7,129,0,129,4,130,4,127,6,124,4,125,4,125,0,-9999 970
    ! BANG 980
    DATA 11,-1,0,-3,3.5,-1.5,2.5,-2,8,-1,5,0,9.5,0.5,4,1.5,6.5,1 990
    DATA 2,3,3,3,1,0,-9999 1000

```

Parts of the software tool kit described in Chapter 2 are useful here. The DRAW subroutine may be used to draw the blockhouse and the gun carriage. The DRAW A ROTATED FIGURE subroutine may be used for the gun barrel, the muzzle flash, and the explosion of the shell. The ELLIPSE subroutine can draw the cannon wheel and shield.



**Cannon Shooting Game Code Description**

To begin the program, the screen is cleared and constants are set:

S9 = 1      scaling value for DRAW A ROTATED FIGURE  
 V = 1000   muzzle velocity  
 T1 = 0.1   time interval  
 T0 = 0      start of trajectory time  
 G = 32      force due to gravity  
 X1 = 124   location of front of blockhouse  
 X2 = 130   location of back of blockhouse  
 Y1 = 0      location of blockhouse floor  
 Y2 = 6      highest point on blockhouse

X1, X2, Y1, and Y2 will be used later to determine if the blockhouse was hit.

The surface of the ground and the blockhouse may now be drawn. On square-ruled paper, draw a blockhouse six units long and six high, as shown in the accompanying drawing.

Beginning at X = 129, Y = 0 the data points are (sequence number 970):

X	Y
129	0
129	4
130	4
127	6
124	4
125	4
125	0

The gun body and muzzle flash are a little more complex. The drawings show the data points for the gun body and how the muzzle flash should look, using these muzzle flash data points (sequence number 930):

Muzzle X	Flash Y
12.5	3.75
16	5
15	3.5
18	3
14.5	2.5
15.5	1
12.5	2.25

The barrel and muzzle flash are defined as they would appear before rotation of the barrel, at  $\theta = 0$ . They are processed by the DRAW A ROTATED FIGURE subroutine. A shell burst is drawn at the point where the shell hits the ground or the blockhouse, also using DRAW A ROTATED FIGURE, because although the shell burst is not rotated, it is translated to the point where the shell hits as shown in the drawing.

The data points for the shell burst are (sequence numbers 990 – 1000):

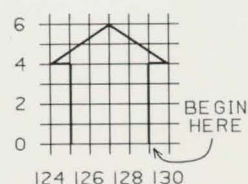
Sequence Number

110

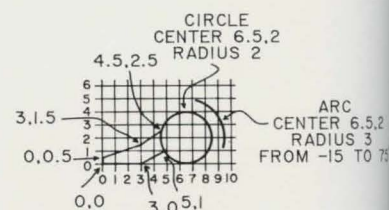
120

200

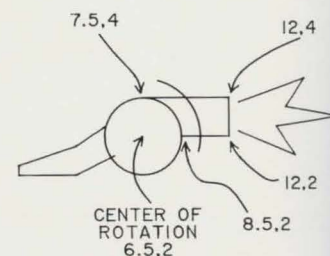
210



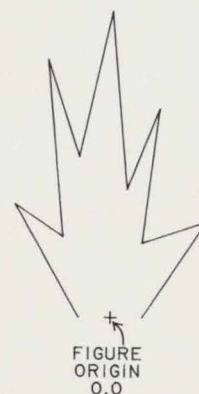
Data points for drawing a blockhouse



Data points for drawing side view of a gun



Data points for drawing gun with muzzle flash



Drawing of shell burst

Shell X	Burst Y
-1	0
-3	3.5
-1.5	2.5
-2	8
-1	5
0	9.5
0.5	4
1.5	6.5
1	2.3
3	3
1	0

The player is asked to

260

ENTER ANGLE OF CANNON (0 to 45)

The **G** characters in the program are Control-G, typed by holding the 'Ctrl' key while pressing G. This control character causes a bell or buzzer to ring momentarily on almost all terminals to get the user's attention.

270

When the player enters a number in the required range, it is saved at T, or  $\theta$  the rotation angle to be used by DRAW A ROTATED FIGURE (Subroutine 1190).

280

X9 and Y9 are set to the X,Y values of the center of rotation of the gun barrel. The barrel and gun body, wheel and shield are drawn, followed by the flash as the gun is fired.

300

Subroutine 1360 is the ELLIPSE routine.

420

Now the shell must be drawn and erased as it moves along the trajectory in intervals of T1 seconds. The trajectory begins at the gun mouth and ends when the shell reaches the ground.

Where is the mouth? The length of the barrel is 5.5 feet from the center of rotation and its thickness is 2.0 feet. If the barrel were horizontal, the center of the end of the barrel would be at  $6.5 + 5.5$ ,  $2.0 + 1.0$  since the center of rotation is at 6.5, 2.0. After rotation, the center of the end of the barrel is as shown in the illustration.

To find the resulting point, the center of rotation must be added (See Note 2 at the end of the book):

$$X0 = 6.5 + 5.5 * \cos(\theta) - \sin(\theta)$$

500

See Appendix, p. 201.

$$H0 = 2.0 + 5.5 * \sin(\theta) + \cos(\theta)$$

where X0 is the beginning point on the horizontal path of the shell and H0 is the beginning height. The initial vertical velocity V0 is

$$V0 = V * T1 * \sin(\theta)$$

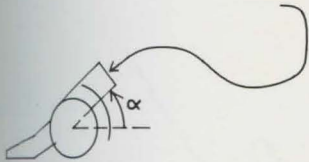
520

and the distance travelled horizontally in one time interval is

$$D = V * T1 * \cos(\theta)$$

530

Having established values for X0, H0, V0, and D, you can compute the trajectory. As in the bouncing ball problem, elapsed time T0 is increased by one time interval T1 and values for the X,Y position of the shell are computed:



Location of center of end of barrel



$$Y = H_0 + V_0 * T_0 - G * T_0^2 / 2$$

$$X = X_0 + D * T_0$$

560  
570

As soon as the shell in its newly computed position appears on the screen, the shell shown by the previous computation is erased and redrawn (Subroutine 1010) at its new position, and the trajectory is continued until the shell passes out of view to the right or returns to the ground.

580  
610

Now it is necessary to determine if the shell hit the blockhouse. (Remember that  $X_1$ ,  $X_2$  and  $Y_1$ ,  $Y_2$  were used to define the horizontal and vertical extents of the blockhouse.) If  $X$  is between  $X_1$  and  $X_2$  when  $Y$  becomes zero, the shell has hit the blockhouse. However, if the shell travelled past the house on a flat trajectory, maybe it was below  $Y_2$  as it went by. If so, it also hit the blockhouse. First, let's see if the shell actually landed between  $X_1$  and  $X_2$ .

The time for the shell to reach the top of its trajectory  $T_{top}$  is equal to  $V_0 / G$ . The time for the shell to come back down is longer, since it has to fall  $H_0$  feet more. The top of the trajectory  $H$  (see Note 3 at the end of the book) is at:

$$H = H_0 - V_0^2 / (2 * G)$$

From the bouncing ball problem, the time to fall from a height  $H$  is

$$T_{drop} = \text{SQR}(2 * H / G)$$

The total time of the trajectory is then

$$T_{top} + T_{drop} = V_0 / G + \text{SQR}(2 * H / G)$$

as seen in the trajectory drawing. Since

$$H = H_0 - V_0^2 / (2 * G)$$

the total time of the trajectory  $T_8$  is

$$T_8 = V_0 / G + \text{SQR}((2 * H_0 + V_0^2 / G) / G)$$

620

and where the shell landed is

$$X = X_0 + D * T_8$$

630

To determine how high the shell was when it passed over the center of the blockhouse (the average of  $X_1$  and  $X_2$ , or  $(X_2 + X_1) / 2$ ), the time may be calculated by substituting  $(X_1 + X_2) / 2$  for  $X$  in the formula  $X = X_0 + D * T$ :

$$(X_1 + X_2) / 2 = X_0 + D * T_{over}$$

or

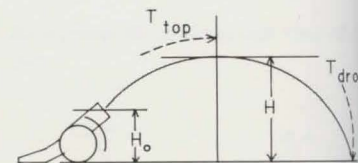
$$((X_1 + X_2) / 2) - X_0 / D = T_{over}$$

The height  $H$  of the shell when it is over the blockhouse is

$$H = H_0 + V_0 * T_{over} - G * T_{over}^2 / 2$$

650

So, to indicate a hit, either  $X$  must be between  $X_1$  and  $X_2$  or lower than  $Y_2$ . If the shell hit the blockhouse, a shell burst is displayed (Subroutine

660  
710

Calculating shell's trajectory time



1190) and the word "BANG!" is printed. If the shell missed, the shell burst is shown where  $Y = 0$  if  $X$  is on the screen. The word "Missed..." is printed.

After a short pause, the program starts again.

800

830

860

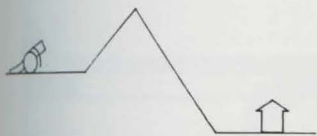
### Refinements

In this game, nothing is allowed to vary except the angle of the gun barrel. Consequently, having played it once, a user knows the correct angle and can hit the blockhouse on the first try. Some of the following suggestions are easy to implement and make the game more interesting.

- Randomly move the position of the blockhouse in the  $X$  direction so that  $X1$  may vary between 100 and 124. To do this, get a random number between 0 and 1 (using the RND function) and multiply it by 24. This yields a number between 0 and 24. Subtract it from  $X1$  and  $X2$  and alter the program so that the blockhouse is drawn by the TRANSLATE A FIGURE subroutine.
- Randomly place the gun and the blockhouse at different heights, as suggested in the illustration.
- Include an intervening hill of random size and location, as seen in the next illustration.
- Include the effect of a randomly strong wind, shown by an arrow whose length is proportional to the strength of the wind. At each time increment, add to  $X$  a factor proportional to the wind force  $W$ . To generate a random wind, first set  $W$  equal to a random number between 0 and 1. Draw the shaft of the arrow  $5*W$  in length. Then get a second random number  $R2$ . If  $R2 < .5$ , let  $W = -W$  and draw the arrowhead on the left-hand end of the line. If  $R2 \geq .5$ , draw the arrowhead on the right-hand end of the line.
- Allow the elevation of the gun to vary between  $-10$  and  $90$  degrees. If the combination of wind and barrel angle causes the shell to hit the cannon, do something spectacular—at least draw a big explosion and write "BOOM!!!" instead of "BANG!"
- Allow the player only three shots per game. If he misses with all three, he loses. Keep score from one game to the next. Give 5 points for a hit on the first shot, 3 on the second, 1 on the third. Deduct 2 points if all three shots miss.
- Change the target to a tank. Move the tank toward the cannon along the terrain. If the player cannot hit the tank by the time the tank reaches the top of the hill, make the tank aim its gun and fire at the cannon. The tank should never miss.



Placing the gun and blockhouse at different heights



Placing obstacle between gun and blockhouse



## Racetrack Games





Races are an ancient form of contest. Long before the original Olympic Games in Greece, people were competing in races and even training their domestic animals to race, inevitably for the purpose of wagering. People have ridden camels and ostriches in races, so it is certainly no surprise that many dogs and horses are bred for the express purpose of racing.

Racing is a form of what psychologists call displacement (in this case, a way to compete without going to war). The participants derive satisfaction from competing. The owners of the animals or vehicles used in racing enjoy developing superior racers. Spectators become a part of the contest by betting on the outcome. The same excitement can be generated by a realistic computer race game.

The design of an oval track and the movement of racing animals around it are most easily programmed through the use of polar coordinates. That is only the beginning of a complete horse race game, however, so discussions of betting and the announcer's commentary are included to add color and realism to the program. The final section of the chapter is devoted to the animation of the horses themselves, building on the introduction to the subject in Chapter 4.

### Designing the Track

A typical racetrack looks like two semicircles connected by straight sections, as shown in the first track diagram.

The next five track diagrams illustrate attempts to create a realistic racetrack using various methods.

It is tempting to break the track into four sections and move the horses or vehicles around each section independently. This requires relatively complicated logic:

```

θ1 = 270
θ2 = 90
WHILE Xhorse < X1 DO
    INCREASE Xhorse
    DRAW HORSE AT Xhorse,Yhorse
;
WHILE Xhorse > X1 DO
    INCREASE θ1
    DRAW HORSE AT X1 + Rhorse * cos(θ1), Y1 + Rhorse * sin(θ1)
;
Set Yhorse TO BACKSTRETCH Y, THAT IS -Y0
WHILE Xhorse > X2 DO
    DECREASE Xhorse
    DRAW HORSE AT Xhorse,Yhorse
;
WHILE Xhorse < X2 DO
    INCREASE θ2
    DRAW HORSE AT X2 + Rhorse * cos(θ2), Y2 + Rhorse * sin(θ2)
;
SET Yhorse TO Y0
WHILE Xhorse < X0 DO
    INCREASE Xhorse
    DRAW HORSE AT Xhorse,Yhorse
;

```

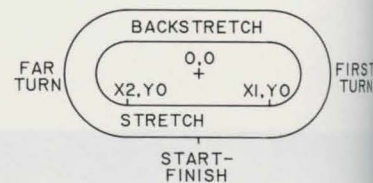


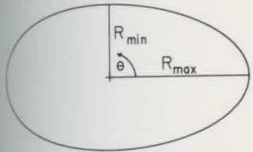
Diagram of a racetrack

whereas, if the track were circular,

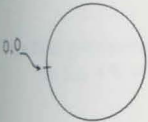
```

θ = 0
WHILE θ < 2π DO
  DRAW HORSE AT X0 + Rhorse * cos(θ), Y0 + Rhorse * sin(θ)
  INCREASE θ
;

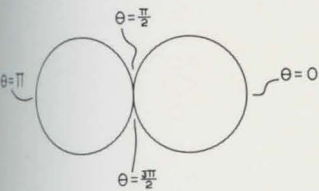
```



Attempt No. 1 to generate a racetrack diagram, using  $R$  and  $\theta$



Attempt No. 2 to generate a racetrack diagram, using  $X = X0 + \cos(\theta)$ ,  
 $Y = Y0 + \sin(\theta)\cos(\theta)$



Attempt No. 3, using  $R = \cos(\theta)$



Attempt No. 4, using  $R = 1 + \cos(\theta)$

If a function of  $R$  and  $\theta$  could be found for a typical track, then writing a program would be much easier. In order to generate a typical track using  $R$  and  $\theta$  the function must generate maximum values of  $R$  when  $\theta = 0$  and  $\theta = \pi$  and minimum values of  $R$  when  $\theta = \pi/2$  and  $3\pi/2$ , as seen in "Attempt No. 1."

This suggests a cosine function, since  $\cos(0) = 1$ ,  $\cos(\pi/2) = 0$ ,  $\cos(\pi) = -1$ , and  $\cos(3\pi/2) = 0$ .

Other attempts to improve the function, such as letting  $\theta = 0$  to  $2\pi$  in small increments and drawing  $X0 + R * \cos(\theta)$ ,  $Y0 + R * \sin(\theta)$  where  $R = \cos(\theta)$  yields "Attempt No. 2,"

which is not even as useful as letting  $R$  be a constant, since now the origin  $X0, Y0$  of the track is not in the center and the track is still not oval.

The function  $R = \cos^2(\theta)$  returns the origin to the middle of the track but the straightaways are pinched in ("Attempt No. 3"), because  $\cos^2(0) = 1$ ,  $\cos^2(\pi/2) = 0$ ,  $\cos^2(\pi) = 1$ ,  $\cos^2(3\pi/2) = 0$ , and  $\cos^2(2\pi) = 1$ .

```

X = 65
Y = 50
C = 5
T = 0
WHILE T <= 2 * PI DO
  R = COS(T)↑2 * C
  IF T = 0 THEN
    MOVE X + R * COS(T), Y + R * SIN(T)
  ;
  DRAW X + R * COS(T), Y + R * SIN(T)
  T = T + PI/72
;

```

What about  $R = 1 + \cos^2(\theta)$ ? Now we have

$\theta$	$\cos^2(\theta) + 1$
0	2
$\pi/2$	1
$\pi$	2
$3\pi/2$	1
$2\pi$	2

and the curve is shown in "Attempt No. 4."

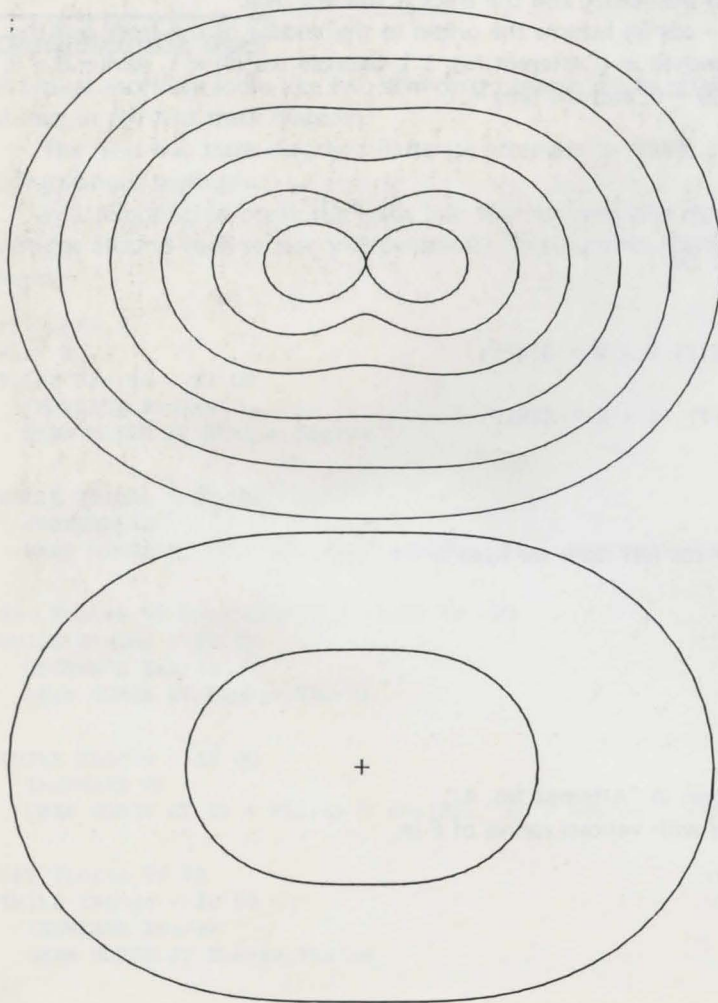
Experimentation with various values of  $P$  in

$$R = P + \cos^2(\theta)$$

shows that  $P = 2$  yields a good shape ("Attempt No. 5").

To generate a racetrack, the inside and outside edges of the track are drawn by using two values of  $C$ , as shown in the successful racetrack diagram.

```
X = 65
Y = 50
C = 10
WHILE C <= 20 DO
  T = 0
  WHILE T <= 2 * PI DO
    R = (2 + COS(T)↑2) * C
    IF T = 0 THEN
      MOVE X + R * COS(T), Y + R * SIN(T)
    ;
    DRAW X + R * COS(T), Y + R * SIN(T)
    T = T + PI/72
  ;
  C = C + 10
;
;
```



Attempt No. 5, experimenting with various values of  $P$  in  $R = P + \cos(\theta)$

Successful racetrack diagram, using two values of  $C$



### Adding Horses

Now that the track is ready, what about the horses? First of all, a race never begins on a turn, so the horses will begin and end at  $\theta = 270$  degrees—that is,  $3\pi/2$  or  $-\pi/2$ . Assuming that the horses start out evenly spaced and that the inside of the track is at  $C = 10$  and the outside is at  $C = 20$  in the equation,

$$R = (2 + \cos^2(\theta)) * C$$

then the space between the centers of two consecutive horses is

$$(20 - 10)/N$$

where  $N$  is the number of horses, say six.

In horse racing, usually the order of finish of the first three horses—win, place, and show—is important for parimutuel wagering. Also, the “odds” or pay-out on a bet will depend on the ratio of the number of people betting on the selected horse compared to the total betting pool. More about that later. Clearly a horse should perform to some extent according to his odds.

Let us arbitrarily decide that an average move for one horse should be  $\pi/72$  radians, about 2.5 degrees. Throughout the course of one race, a horse will be shown in about 144 positions. The horses will move according to random numbers until close to the end of the race, when the odds on each horse will begin to have an effect.

Using an array of six variables  $H(1)$  through  $H(6)$ , each horse’s angular position will be updated throughout the race, beginning at  $-\pi/2$  and being modified thus:

$$H(I) = H(I) + \pi/36 * \text{rnd}(-2)$$

where  $\text{rnd}(-2)$  is a random number between 0 and 1. Since the average such random number is 0.5, the average angular increment will be  $\pi/36 * 0.5$ , or  $\pi/72$ .

After one of the horses passes a certain point along the stretch, odds will be applied to all horses. So a horse racing at odds of 6 to 1 will move according to the formula

$$H(I) = H(I) + \pi/36 * \text{rnd}(-2) * \text{Odds}$$

where Odds in this case is 1/6. This means that the average move will now be only 1/6 of what it was before the odds were applied.

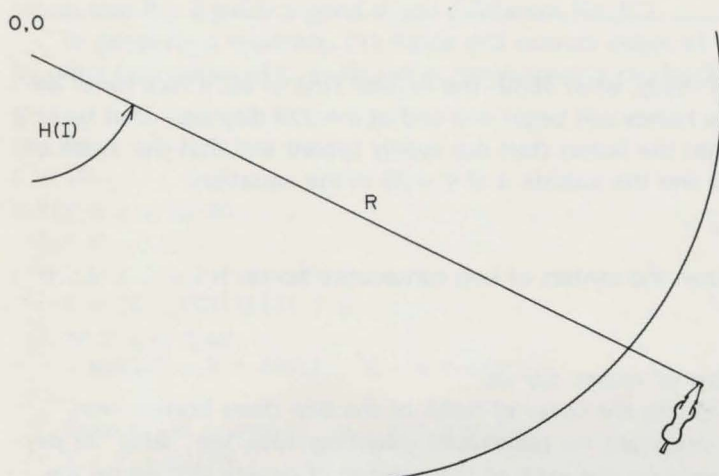
Odds will be selected for each horse from a table of allowable odds: 1 to 1 (even), 6 to 5, 5 to 4, 3 to 2, 2 to 1, 5 to 2, 3 to 1, 4 to 1, 5 to 1, 6 to 1, 7 to 1, 8 to 1, 9 to 1, 10 to 1, 15 to 1, 20 to 1, 30 to 1, 50 to 1, and 99 to 1.

Each horse will be displayed so that its nose touches the point

$$X + R * \cos(H(I)), Y + R * \sin(H(I))$$

and so that its spine is perpendicular to the line from its nose to the middle of the track, the point 0,0, as illustrated.

Horses can be modelled from a simple “above” view, as shown in the enlarged diagram.



Placement of a horse relative to the track

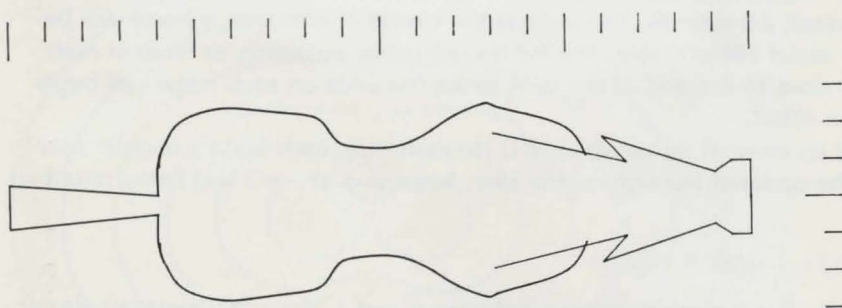


Diagram of a horse seen from above

The horses must continue to run around the track after they cross the finish line, at least until the third horse has finished. It would be highly unrealistic to stop or erase the winning horse while waiting to determine the "place" (second) horse and then the "show" (third) horse.

Given the goal of keeping note of the order of finish of the first three horses, the following algorithm will get them around the track:

### Simple Racetrack Program Code

	Sequence Number
X = 65 ! Assuming a screen size 0 to 130 in X, 0 to 100 in Y,	120
Y = 50 ! set X,Y to center of screen	130
DIM H(6),P(6),Q(6),W(3),S(19)	140
DATA 1,0.833,0.8,0.667,0.5,0.4,0.333,0.25,0.2,0.167,0.143,0.125	150
DATA 0.111,0.1,0.067,0.05,0.033,0.02,0.01	160
READ S ! Read odds table	170
H = -PI/2 ! Set horses at start	180
I = 1	190
REPEAT	200
Q(I)=3*I-2+INT(RND(-2)*4) ! Get random odds for each horse	210

```

P(I) = S(Q(I))                                     220
I = I + 1                                           230
UNTIL I > 6                                         240
C = 10                                              250
REPEAT                                              260
    MOVE X,Y - C * 2                                270
    T = -PI/2                                        280
    REPEAT                                          290
        R = (2 + COS(T)2) * C                        300
        DRAW X + R * COS(T), Y + R * SIN(T)         310
        T = T + PI/72                               320
    UNTIL T > 1.5 * PI                             330
    C = C + 10                                      340
UNTIL C > 20                                        350
F = 0      ! No horse has reached the handicap point 360
J = 0      ! No horse has finished                 370
WHILE J < 3 DO      ! Run horses until 3 have finished 380
    N=INT(RND(-2)*6+1) ! Select first horse to move on this turn 390
    M = N                                                400
    REPEAT                                          410
        I = M                                          420
        IF I > 6 THEN      ! If horse six just moved, 430
            I = I - 6      ! next to move is horse one. 440
        ;                                              450
        C = I * 1.6667 + 9.1667      ! Locate horse 460
        R = (2 +COS (H(I))2) * C      470
        GOSUB 1000      ! Draw horse 490
        IF H(I) > 1.4 * PI THEN      ! Set F to 1 if this horse 500
            F = 1      ! has passed the handicapping point 502
        ;                                              504
        IF H(I) >= 1.5 * PI THEN      ! Has this horse finished? 510
    CASE J OF
        0:      ! If so, is it first? 520
            W(1) = I      ! Save horse number as winner 540
            J = 1      550
            ; 560
        1:      ! Second? 570
            IF W(1) < > I THEN 580
                J = 2 590
                W(2) = I      ! Save "place" horse number 600
                ; 610
            ; 620
        2:      ! Third? 630
            IF W(1) < > I AND W(2) < > I THEN 640
                J = 3 650
                W(3) = I      ! Save "show" horse number 660
                ; 670
            ; 680
        ; 690

```



```

;
IF F = 0 THEN
    H(I) = H(I) + PI/36 * RND(-2)
ELSE
    H(I)=H(I)+PI/36*RND(-2)*P(I) ! Handicap all horses if F<>0
;
M = M + 1 ! Do next horse
UNTIL M > N + 5
;

```

700  
710  
712  
714  
716  
718  
720  
730  
740

### Simple Racetrack Code Description

Sequence  
Number

Establish X, Y as the middle of the track. Establish an odds table S. Set the initial angle of each horse H as  $\pi/2$ . This angle is the angle of the line from X,Y to the horse's nose. 120

Find at random an index to the odds table between table entries 1 and 4 for horse 1, 4 and 7 for horse 2, 7 and 10 for horse 3, etc. Save the index in the table Q and the odds in the table P. 210

Draw the inner and outer extents of the track. 250

Set F = 0 to indicate that no horses have reached the point at which the odds or handicapping will be applied. 360

Set J = 0 to indicate that no horses have crossed the finish line. 370

Run horses until three have finished. Select at random which horse moves first, moving the others sequentially thereafter until all six have moved. 380

For each horse, find the distance from the middle of the track to the horse's nose. 460

Draw the horse at  $X + * \cos(H(I))$ ,  $Y + * \sin(H(I))$  (Subroutine 1000). 490

If this horse has passed the handicapping point, set F to 1. 500

If this horse has passed the finish line, then 510

If no horses has finished, this horse is the winner: put its number in W(1) and show that one horse has finished. 520

If one other horse has finished, this horse is the "place" horse: put its number in W(2) and show that two horses have finished. 570

If two other horses have finished, this horse is the "show" horse: put its number in W(3) and show that three horses have finished. 630

Increase this horse's angle by  $\pi/36$  times a random fraction, times this horse's odds if F = 1 or times 1 if F = 0. Go to the next horse. 710

If six horses have already been processed and if less than three horses have finished, continue to process horses. 730

### Wagering

Now horses can be run around a track, but this game is not yet a reasonable simulation of a horse race. For example, at a real track people would be allowed to place bets, the announcer would "call" the race (describe the position of the horses as they ran), and at the end of the race a tote board would show the win,

place, and show horses along with their payout for a \$2 bet. Bettors holding paying tickets would cash them in and buy tickets for the next race based on the expected performance of a new group of horses.

A racing "card" or program usually calls for nine races. Assuming that a complete game is nine races, then, the preliminary information needed for the game is

How many horseplayers?

What is the name of each horseplayer?

Before each race, the odds on each horse must be shown to each player so bets may be placed. Each player is asked

How many bets? (1, 2 or 3)

Then for each bet,

BET 1: HOW MUCH? (\$2, 5, 10, 20, 50, 100) \$

WHICH HORSE? (1-6)

WIN, PLACE, or SHOW? (W, P, or S)

BET 2: and so on.

After all bets are placed, the track may be displayed along with the tote board, which will be used to show the numbers of the win, place, and show horses along with their payouts for a \$2 bet.

The horses are run randomly around the track until one horse passes the angle  $1.4\pi$ . Then, odds are applied to each horse until the order of finish has been established. As each of the first three horses passes the finish line, its payout for a \$2 bet is computed and displayed on the board. At a real racetrack, three betting pools are maintained, one for the winner, one for the place bets (shared between place tickets on the winner and the second horse), and one for the show bets (shared among show tickets on the first three horses). To cover expenses, taxes, and profit, the track takes 17 percent out of each pool, after which the other 83 percent is distributed to the ticket holders. For the place and show horses, distribution is proportional to the odds on the horses involved. The odds at a real track are, in fact, determined solely by the number of tickets on each horse in the pool:

$$\text{Odds} = (\text{Tickets on horse}) / (\text{Tickets in pool})$$

In this game, since the pool is too small to be realistic, the odds are established to reflect typical odds and the payout is to be computed from those odds:

$$\text{Winner} = \$ * 0.83 / \text{Odds} + \$$$

$$\text{Place} = \$ * 0.83 / \text{Odds} / 2 + \$$$

$$\text{Show} = \$ * 0.83 / \text{Odds} / 3 + \$$$

except that at a real racetrack the minimum payback must be 5 percent, or \$2.10 on a \$2.00 ticket. Otherwise, payment is rounded to the nearest 10 cents. For example, the payback on a \$2 ticket on the show horse is the \$2 bet plus 83 percent of \$2 (\$1.66), divided by the odds, divided by three. If the odds are 5 to 1, divide by 1/5 or 0.2. (Of course, the odds could be computed by dividing 1 into 5 instead of 5 into 1, in which case we would multiply by the odds.) The payout at 5 to 1 would be



$1.66 / .2 / 3 + 2$  or \$4.766666...

Rounded to the nearest dime, the payout is \$4.80. To get this result, use the formula

$\text{INT}((1.66/\text{ODDS}/3 + 2) * 10 + 0.5)/10$

which yields

$(1.66/0.2/3 + 2) * 10 = 47.666...$   
 $47.666... + 0.5 = 48.166...$   
 $\text{INT}(48.166...) = 48$   
 $48/10 = 4.80$

Unfortunately, the 0 in 4.80 will not be displayed by a PRINT statement. To get a trailing zero, it is necessary to convert the number to a string. Even then, the trailing zero will not appear without a little trickery. The number 4.805 will convert to the string "4.805." The last character of the string is then deleted, leaving "4.80." The value of the expression

$\text{INT}((1.66/\text{ODDS}/3 + 2) * 10 + 0.5)/10 + 0.005$

will always have three decimal places and may be used with the string manipulation described above to ensure two decimal places.

Here is a more realistic horserace:

<b>Horse Race Program Code</b>	<b>Sequence Number</b>
X = 65	120
Y = 50	130
DIM H(6),P(6),Q(6),W(3),S(19),L(10,9),P\$(200),B(10),A(10)	140
DIM (6,2)	150
DATA 1,0.833,0.8,0.667,0.5,0.4,0.333,0.25,0.2,0.167,0.143	160
DATA 0.125	165
DATA 0.111,0.1,0.067,0.05,0.033,0.02,0.01	170
P\$ = "" ! Empty the player name list	180
READ S ! Read odds	190
A = 0 ! Empty all wagering accounts	200
U = -1 ! Initialize commentator's flag	210
PAGE	220
PRINT "" , "COMPUTER PARK RACETRACKJJJ"	230
REPEAT	240
PRINT "HOW MANY HORSEPLAYERS? (1-10) GGG";	250
INPUT K ! Get number of players—at least one	260
UNTIL K >= 1 AND K <= 10 ! and not more than 10	270
I = 1	280
REPEAT	290
PRINT "NAME OF PLAYER ";I;"? G";	300
INPUT A\$ ! Get player names	310
GOSUB 6850	320
I = I + 1	330
UNTIL I > K	340
N = 1	350
REPEAT	360



```

H = -PI/2 ! Set start position of horses          370
I = 1                                              380
REPEAT                                           390
    Q(I) = 3 * I - 2 + INT(RND(-2) * 4) ! Select odds 400
    P(I) = S(Q(I))                               410
    I = I + 1                                     420
UNTIL I > 6                                       430

J = 1                                             440
B = 0                                             450

REPEAT                                           460
    PAGE                                          470
    PRINT "J", "COMPUTERPARK RACETRACK"          480
    PRINT " ", " ", " ", " ", " ", " "         490
    IO = N                                       500
    GOSUB 4270 ! Display race number and odds   510
    PRINT " RACE"JJ" ! on each horse           520
    PRINT " ", " ", " ", " ", " ", " "         530
    I = 1                                       540

    REPEAT                                       550
        PRINT " ", " ", " ", " ", " ", " "     560
    GOSUB 9000                                   570
    PRINT D$                                     580
    I = I + 1                                   1160
UNTIL I > 6                                       1170

GOSUB 6940 ! Get a player's name               1180

IF N > 1 THEN ! If not the first race,         1190
    PRINT "J";A$; ", YOU ARE "; ! show player's account 1200
    CASE SGN(A(J)) OF                          1210
        -1:                                    1220
            PRINT "DOWN $";ABS(A(J))           1230
            ;                                   1240
        0:                                     1250
            PRINT "EVEN"                       1260
            ;                                   1270
        1:                                     1280
            PRINT "UP $"                       1290
            ;                                   1300
            ;                                   1310
            ;                                   1320
        WHILE B(J) < 1 OR B(J) > 3 DO ! Let player elect to 1330
            PRINT "JHOW MANY BETS, "; ! make up to 3 bets 1340
            IF N = 1 THEN                       1350
                PRINT A$;                       1360
                ;                               1370
            PRINT "? (1-3) G";                 1380
            INPUT B(J)                         1390
            ;                                   1400
        E = 1                                 1410

```

```

        WHILE E < =B(J) DO ! Accept players' bets      1420
            PRINT "J";                                  1430
            IO = E                                       1440
            GOSUB 4270                                   1450
            PRINT " BET:"                               1460
            GOSUB 3000                                   1470
            E = E + 1                                    1480
        ;                                              1490

        J = J + 1                                       1500
        UNTIL J > K                                     1510

    PAGE          ! Display the track                  1520
    C = 10                                                1530
    REPEAT                                              1540
        MOVE X,Y - C * 2                                1550
        T = -PI/2                                        1560

        REPEAT                                          1570
            R = (2 + COS(T)2) * C                        1580
            DRAW X + R * COS(T), Y + R * SIN(T)        1590
            T = T + PI/72                               1600
        UNTIL T > 1.5 * PI                             1610

        C = C + 10                                      1620
    UNTIL C > 20                                         1630

    F = 0                                                1640
    MOVE 62,60      ! Draw tote board                  1650
    PRINT "$2 PAYS"                                     1660
    RESTORE 1680                                         1670

1680 DATA 5,0,9,3,9,3,0,0,0,0,9 ! Tote board          1680
DATA 2,0,6,3,6                                         1690
DATA 2,0,3,3,3                                         1700
DATA 7,41,6,5,6,5,9,41,9,41,0,29,0,29,9             1710
DATA 3,17,9,17,3,41,3                                 1720
DATA -9999 ! Translate tote board to 45,50 and draw it 1730
X9 = 45                                                1740
Y9 = 50                                                1750
GOSUB 4010                                             1760
I = 1                                                  1770
RESTORE 1790                                           1780

1790 DATA 50.2,56.2,62.2,56.2,74.2,56.2,62.2,53.2,74.2,53.2,74.2 1790
DATA 50.2                                              1795

    REPEAT                                              1800
        READ X8,Y8 ! Print $ signs on tote board      1810
        MOVE X8,Y8                                     1820
        PRINT "$";                                     1830
        I = I + 1                                       1840
    UNTIL I > 6                                         1850

    J = 0          ! Race the horses                   1860

    WHILE J < 3 DO                                     1870
        V = INT(RND(-2) * 6 + 1)                      1880
        M = V                                           1890

        REPEAT                                          1900
            I = M                                       1910

            IF I > 6 THEN                                1920
                I = I - 6                              1930
            ;                                           1940

```

```

C = I * 1.6667 + 9.1667      1950
R = (2 + COS(H(I))↑2) * C    1960
GOSUB 6970                    1970

IF H(I) > 1.4 * PI THEN      1980
    F = 1                    1982
;                              1984

IF H(I) >= 1.5 * PI THEN     1990

CASE J OF                    2000
    0:                        2010
        ! For the winner,
        W(1) = I              2020
        J = J + 1            2030
        MOVE 44.5,56.2       2040
        PRINT I; ! display horse number 2050

        G = 1.66 ! and $2 payout to win, 2060
        MOVE 49.2,56.2       2070
        GOSUB 4160           2080
        G = 0.83 ! place,    2090
        MOVE 61.2,56.2       2100
        GOSUB 4160           2110
        G = 0.5533 ! and show. 2120
        MOVE 73.2,56.2       2130
        GOSUB 4160           2140
    ;                          2150

    1:                        2160
        ! For the "place" horse,
        IF W(1) < > I THEN    2170
            W(2) = I          2180
            J = J + 1          2190
            MOVE 44.5,53.2     2200
            PRINT I; ! Display horse number and $2 place 2210
            G = 0.83 ! payout  2220
            MOVE 61.2,53.2     2230
            GOSUB 4160         2240
            G = 0.5533 ! and show. 2250
            MOVE 73.2,53.2     2260
            GOSUB 4160         2270
        ;                     2280
    ;                          2290

    2:                        2300
        ! For the "show" horse,
        IF W(1) < > I AND W(2) < > I THEN 2310
            W(3) = I          2320
            J = J + 1          2330
            MOVE 44.5,50.2     2340
            PRINT I; !Display horse number and $2 show 2350
            G = 0.5533 !payout. 2360
            MOVE 73.2,50.2     2370
            GOSUB 4160         2380
        ;                     2390
    ;                          2400
    ;                          2410
    ;                          2420

```



IF F = 0 THEN	! See Note 4 at the end of the book	2430
H(I) = H(I) + PI/36 * RND(-2)		2432
ELSE		2434
H(I)=H(I)+PI/36*RND(-2)*P(I)	! Apply odds if F<>0	2436
;		2438
M = M + 1		2440
GOSUB 5000	! Do racing commentary	2450
UNTIL M > V + 5		2460
;		2470
I = 1	! Pay bets for each player	2480
WHILE I <= K DO		2490
J = 1	! Each bet	2500
WHILE J <= B(I) DO		2510
M = 1		2520
WHILE M <= L(I,J + 3) DO	! Each qualifying horse	2530
IF L(I,J + 6) = W(M) THEN	! Compute payback	2540
V=INT((1.66/M*L(I,J)/P(W(M))+L(I,J)+0.05)*10)/10		2550
V = V MAX 1.05 * L(I,J)		2560
A(I) = A(I) + V		2570
;		2580
M = M + 1		2590
;		2600
J = J + 1		2610
;		2620
I = I + 1		2630
;		2640
MOVE 50,45		2650
IO = N		2660
GOSUB 4270	! Display race number	2670
PRINT "RACE OFFICIAL"	! and "race official"	2680
V = 1		2690
WHILE V < 1000 DO	! Pause	2700
V = V + 1		2710
;		2720
U = - 1	! Reset commentator's flag	2730
N = N + 1	! and do next race	2740
UNTIL N > 9	! until ninth is finished	2750
PAGE		2760
PRINT " ", "JCOMPUTER PARK RACETRACK JJ"		2770
PRINT "AT THE END OF TODAY'S RACING PROGRAM,"		2780
J = 1		2790
REPEAT	! Display final account	2800
GOSUB 6940	! balances of each player	2810
PRINT "J";A\$;" FINISHED ";		2820
CASE SGN(A(J)) OF		2830
-1:		2840
PRINT "DOWN \$";ABS(A(J))		2850
;		2860

```

0:                                     2870
    PRINT "EVEN"                       2880
;                                     2890
1:                                     2900
    PRINT "UP $";A(J)                 2910
;                                     2920
;                                     2930
    J = J + 1                         2940
UNTIL J > K                           2950
END                                    2960

```

Now to review the complete racing and wagering algorithm:

### **Horse Race Code Description**

	Sequence Number
Set X and Y to the coordinates of the center of the screen.	120
Define arrays.	140
Define all allowable odds.	160
Set the list of player names to "empty."	180
Read the odds table into the S array.	190
Set the balance of each player's wagering account A to zero.	200
Set commentator's flag U to -1.	210
Clear the screen and print title.	220
Ask for the number of players, at least one and not more than ten.	240
Get player names. (Subroutine 6850 puts player name A\$ into string array P\$.)	290
Set all horses at the starting angle $-\pi/2$ .	370
Randomly select the odds for all horses:	390

Horse	Is one of
1	even, 6/5, 5/4, 3/2
2	3/2, 2/1, 5/2, 3/1
3	3/1, 4/1, 5/1, 6/1
4	6/1, 7/1, 8/1, 9/1
5	9/1, 10/1, 15/1, 20/1
6	20/1, 30/1, 50/1, 99/1

Set number of bets per player, B, to zero.	450
Display the odds on each horse. (Subroutine 4270 gets race number.)	510
Subroutine 9000 gets the odds on a horse from a string containing all possible odds in groups of four characters each: "EVEN 6/5 5/4 3/2 ...50/1 99/1" using Q(I) to select the correct group, which is returned in D\$.	570
Get a player name into A\$ (using Subroutine 6940).	1180
If this is not the first race, display the status of the player's wagering account.	1190
Ask the player how many bets, B, he wants to make on this race (at least one and not more than three).	1330
Ask the player for his bet.	1420
Subroutine 4270 displays the word First, Second, or Third depending on	

the value of IO. Subroutine 3000 asks the player how much he is betting (\$2, 5, 10, 20, 50, or 100) to win, place, or show, and on which horse.

After each player has placed his bets, the race begins. The track is displayed.

The tote board appears in the middle of the track.

Subroutine 4010 is the DRAW A TRANSLATED FIGURE subroutine. The tote board is developed from the accompanying diagram.

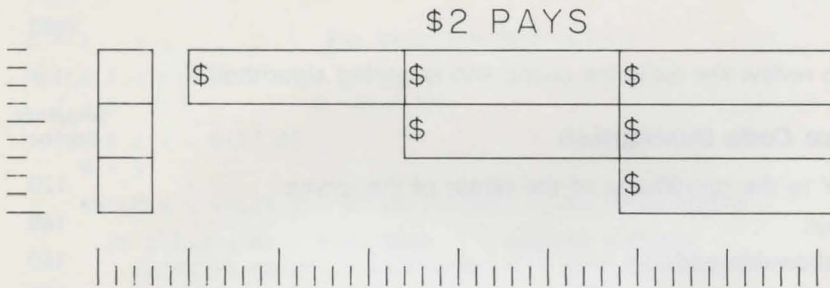


Diagram of racetrack toteboard

For each move, a horse is chosen at random to move first.

Subsequent horses go in ascending sequence until all have moved, for example, if 3 goes first, the order is 3,4,5,6,1,2.

Subroutine 6970 erases the horse in its existing position and redraws it after its move is computed.

The odds flag  $F$  is set to 1 if any horse has passed  $1.4\pi$ .

If the horse  $H(I)$  has passed the finish line, the event must be noted in the order-of-finish table  $W$  and the payout posted on the tote board.  $J$  is the number of horses that have already finished.

If  $J = 0$ , no horses have finished and so horse number  $I$  is the winner. The payout to win, place, and show for a \$2 ticket is displayed from the formula (Subroutine 4160).

$$\text{INT}(G * 2/P(I) + 2.05) * 10/10$$

where  $G = 2 * 0.83$ , or 1.66 for a win ticket

$G = 2 * 0.83/2$ , or 0.83 for a place

$$G = 2 * 0.83/3, \text{ or } .5533 \text{ for a show}$$

If the payout is less than \$2.10, then \$2.10 is paid. (Using the given odds table, there is no case where the payback on \$2 is less than \$2.10. Nevertheless, an error may occur if the odds table were to be changed and the minimum payback test is not included.) If  $J = 1$ , this horse is either already the winner or is the place horse. If it is not the winner, its payout to place and show are displayed.

If  $J = 2$  and this horse is neither the winner nor the place horse, then it is the show horse and its payout to show is displayed.

The new position of the horse is computed, including the odds if  $F = 1$ .

M is increased by one to prepare to process the next horse, and the racing commentary subroutine is called. When all horses have been processed, the test for three finishers is made (this is the end of the WHILE statement at 1870).



If three horses have not yet finished, the race continues.	2470
At the end of the race, for each player I, each bet J is paid or lost depending on whether the horse W(M) finished at least as high as the bet was placed. A win, place, or show ticket pays if the horse won, a place or show ticket pays if it placed, and only a show ticket pays if the horse came in third.	2480
Payout V is at least the wager plus 5%, added to the player's account A(I).	2540
In a real race, if there was a close finish or if a jockey claimed a foul, there would be an announcement including a caution to the bettors to retain all tickets until the race was ruled "Official." The ordinal number of the race (First, Second, etc.), printed in Subroutine 3550 from the value of IO, and the words "Race official" are displayed.	2650
After a pause of about three seconds, the race commentary flag U is reset and the next race begins.	2700
After nine races, the racing accounts of all players are displayed and the program ends. (Subroutine 6940 is used to get a player's name into A\$.)	2800

### **Race Commentary**

Race commentary would be very dull if the announcer merely listed the order in which the horses were running. Instead, the announcer moves rapidly through a rather stylized commentary that invariably begins with "They're off!" as the horses leave the starting gate. At regular intervals around the track he will preface his next list with the location of the horses. For a mile race these positions are

Passing the grandstand  
 Into the turn  
 Around the turn  
 Into the backstretch  
 Along the backstretch  
 Into the far turn  
 Around the far turn  
 Into the stretch  
 Down the stretch  
 Coming to the wire  
 At the finish

Usually the announcer continues with the name of the leading horse, how big its lead is, and sometimes its position on the track. For example, "Wimbush leads by half a length along the rail."

Leads are expressed by parts of a horse:

By a nose  
 By a neck  
 By half a length  
 By three quarters of a length  
 By a length  
 By a length and a half

and so on. Positions are usually

Along the rail  
 On the outside  
 Between horses (meaning there's a horse on either side)

It may be said that a horse

Leads  
 Is leading  
 Has the lead

Following horses may be announced by position, "Pride's Joy is third," merely "Pride's Joy next," or even "then Pride's Joy." Invariably the statement about the last horse is phrased something like "with Galloping Gizmo last" or "and Galloping Gizmo."

Assuming that one length is an angular difference of  $\pi/25$  between two horses, the racing order can be determined by sorting a table containing horse number and position, following which the leader's lead is determined by subtracting the second horse's position from the leader's position and examining the difference.

The commentary subroutine must list the commentary only at the start, finish, and ten specified intervals in between. Before each race begins, a flag U is set to -1. When the commentary routine is entered, U is examined. If  $U = -1$ , the commentator says, "They're off!" and U is set to 0. If U is at least 0 but less than 20, then U is tested again according to this table:

Horse's Position	and U	then Leader Is
$> -3\pi/8$	0	Passing the grandstand
$> -\pi/4$	2	Into the turn
$> 0$	4	Around the turn
$> 3\pi/8$	6	Into the backstretch
$> \pi/2$	8	Along the backstretch
$> 3\pi/4$	10	Into the far turn
$> \pi$	12	Around the far turn
$> 9\pi/8$	14	Into the stretch
$> 5\pi/4$	16	Down the stretch
$> 11\pi/8$	18	Coming to the wire

While  $U = 0$ , nothing will happen until the lead horse's angular position reaches or exceeds  $-3\pi/8$ ; then U is increased by 1. This triggers the appropriate leading remark, e.g., "Passing the grandstand." After the remark has been printed, U is increased by 1 once more. After leaving the subroutine, commentary will not resume until the lead horse reaches  $-\pi/4$ .

Meanwhile, it is necessary to establish the order in which the horses are running. An array Z having dimension (6,2) is filled with the angular position and number of each horse:  $Z(l,1) = H(l)$  and  $Z(l,2) = l$  for values of l from 1 to 6.

Now a flag is set to zero and successive pairs of  $Z(l,1)$  and  $Z(l-1,1)$  are compared, where l takes on the values 2 through 6. If  $Z(l,1)$  is greater than  $Z(l-1,1)$ , then  $Z(l,1)$  is interchanged with  $Z(l-1,1)$  and  $Z(l,2)$  with  $Z(l-1,2)$ , and the flag is set to a non-zero value. If the flag is not zero after  $Z(5,1)$  and  $Z(6,1)$  have been compared, then the process is repeated.

This will arrange the table so that the leading horse's position is in  $Z(1,1)$  and his number is in  $Z(1,2)$ , the second horse in  $Z(2,1)$  and  $Z(2,2)$ , and so forth.



The leading horse's number is printed, followed at random by "leads by," "is leading by," or "has the lead by." Then, depending on the following table, the amount by which he leads is printed:

$Z(1,1)-Z(2,1)$	Lead
$<\pi/144$	A nose
$<\pi/72$	A neck
$<\pi/50$	Half a length
$<\pi/33$	$\frac{3}{4}$ of a length
$<\pi/25$	A length
$<3\pi/50$	A length and a half

If  $Z(1,1) - Z(2,1)$  is greater than  $3\pi/50$ , the nearest whole number of lengths  $\text{INT}((Z(1,1) - Z(2,1))/(\pi/25) + 0.5)$  is printed.

For each of the next four horses, a random choice is made between

Followed by number —

Number — is next

Number — is second (or third, fourth, fifth as appropriate)

or Then number —

There is an 80 percent chance that nothing else will be said about these four horses. Twenty percent of the time, if one of these is horse number 1, the commentator will either say, "On the inside" or "Along the rail." If it is horse number 6, he will say, "On the outside," or if it is one of the other horses, "Between horses."

For the last horse, 80 percent of the time he will say, "And [or With] number 6 last [or sixth]." Twenty percent of the time he will say, "And [or With] number 6 running last [or sixth]." If the last horse is number 6, there is a 40 percent chance the commentator will say "On the outside."

Finally, when  $U = 20$  and  $J = 3$ , the commentator will announce the three winners. The following is a typical commentary:

THEY'RE OFF!!!

PASSING THE GRANDSTAND,  
NUMBER 4 LEADS BY HALF A LENGTH,  
NUMBER 6 SECOND,  
FOLLOWED BY NUMBER 1 ALONG THE RAIL,  
NUMBER 2 FOURTH,  
FOLLOWED BY NUMBER 3,  
AND NUMBER 5 RUNNING LAST.

INTO THE TURN,  
NUMBER 4 IS LEADING BY A NOSE,  
NUMBER 6 SECOND,  
NUMBER 2 IS THIRD,  
NUMBER 3 FOURTH,  
FOLLOWED BY NUMBER 1,  
AND NUMBER 5 RUNNING LAST.

AROUND THE TURN,  
NUMBER 6 HAS THE LEAD BY A LENGTH AND A HALF,  
NUMBER 4 IS NEXT,  
NUMBER 2 THIRD,  
NUMBER 1 IS FOURTH,



NUMBER 3 IS FIFTH,  
AND NUMBER 5 LAST.

INTO THE BACKSTRETCH,  
NUMBER 6 IS LEADING BY 2 LENGTHS,  
NUMBER 4 IS SECOND,  
THEN NUMBER 1,  
NUMBER 2 IS FOURTH,  
NUMBER 5 FIFTH,  
WITH NUMBER 3 SIXTH.

ALONG THE BACKSTRETCH,  
NUMBER 6 LEADS BY 3 LENGTHS,  
NUMBER 4 IS SECOND,  
FOLLOWED BY NUMBER 1 ALONG THE RAIL,  
THEN NUMBER 2 BETWEEN HORSES,  
NUMBER 5 IS FIFTH,  
AND NUMBER 3 LAST.

INTO THE FAR TURN,  
NUMBER 6 HAS THE LEAD BY 3 LENGTHS,  
NUMBER 2 SECOND,  
NUMBER 1 IS THIRD,  
FOLLOWED BY NUMBER 4,  
NUMBER 5 FIFTH,  
AND NUMBER 3 LAST.

AROUND THE FAR TURN,  
NUMBER 6 LEADS BY A LENGTH AND A HALF,  
NUMBER 2 IS SECOND,  
NUMBER 1 THIRD ALONG THE RAIL,  
NUMBER 4 IS FOURTH BETWEEN HORSES,  
NUMBER 5 IS FIFTH,  
AND NUMBER 3 LAST.

INTO THE STRETCH,  
NUMBER 6 LEADS BY A LENGTH AND A HALF,  
NUMBER 2 SECOND,  
NUMBER 1 THIRD,  
FOLLOWED BY NUMBER 4,  
THEN NUMBER 5,  
WITH NUMBER 3 RUNNING LAST.

DOWN THE STRETCH,  
NUMBER 6 LEADS BY 2 LENGTHS,  
NUMBER 2 IS SECOND,  
NUMBER 1 IS NEXT,  
NUMBER 4 IS FOURTH,  
NUMBER 5 IS FIFTH,  
AND NUMBER 3 SIXTH.

COMING TO THE WIRE,  
NUMBER 6 IS LEADING BY 2 LENGTHS,  
NUMBER 1 IS NEXT ON THE INSIDE,  
NUMBER 2 THIRD,  
NUMBER 5 FOURTH,  
NUMBER 4 IS FIFTH BETWEEN HORSES,  
AND NUMBER 3 LAST.

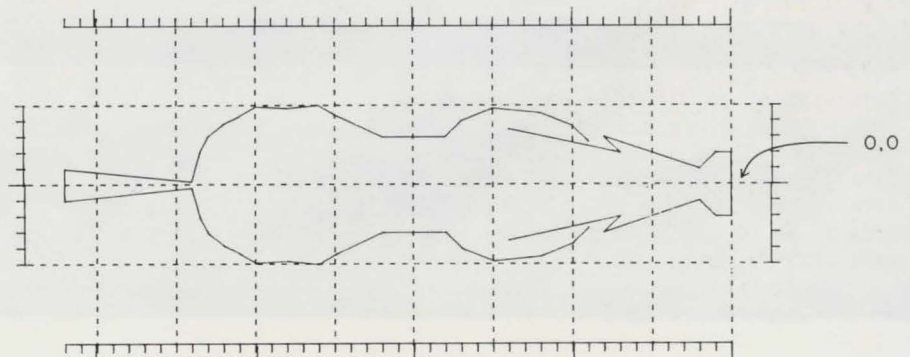
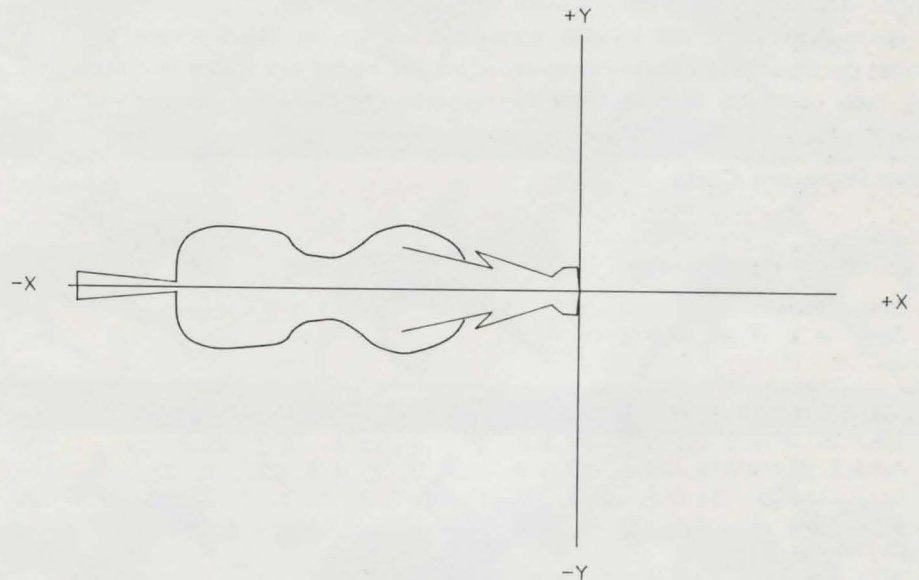
AND THE WINNER IS NUMBER 6.  
NUMBER 1 IS SECOND  
WITH NUMBER 2 THIRD

### Designing Racehorses

Since the racetrack is seen from above, it makes sense to show the horses in the same way. To achieve as much realism as possible, horses should be large enough to show details. This implies that a scaled-down version of the original design will be required. As horses go around the track it will be necessary to turn (rotate) them so they are running parallel to the rail.

The important point on the horse is the tip of his nose. As soon as this point reaches the finish line, the horse has officially finished the race. So the tip of his nose should be the local origin (the point  $X = 0$ ,  $Y = 0$ ) and the rest of the horse should be behind this point. Considering the horse as it starts the race, all points will be in the second and third quadrants of a rectangular coordinate system, as shown here.

Arbitrarily drawing the simplest possible horse on square-ruled paper yields the next figure.



Quadrant positioning of horse for start of race

Drawing layout for horse-drawing subroutine

This figure cannot be drawn in a continuous series of lines without retracing. It is suggested that the head should be drawn starting at the point  $-14, -3.5$  and ending at  $-14, 3.5$ . Then a new sequence, the body, should begin at  $-9, 2.7$ .

There is now enough information about what is needed in a horse-drawing subroutine. The routine must be capable of

- (1) Drawing an arbitrary number of continuous line sequences
- (2) Scaling: making the horse bigger or smaller
- (3) Translation: placing the horse such that its local origin is at a given X,Y location
- (4) Rotation: turning the entire figure about its local origin

After a jockey is added to the horse, the view from above is a reasonable representation of a galloping horse whose legs are gathered between strides, as shown by this frame of the famous Muybridge running horse sequence.

When the horse is striding, his legs will show from above, as you can see in the drawing, representing the horse extended in the gallop, as seen in another Muybridge frame.

His right foreleg is extended. This is called a right lead. Horses may run on a left or right lead and, in fact, change leads from time to time.

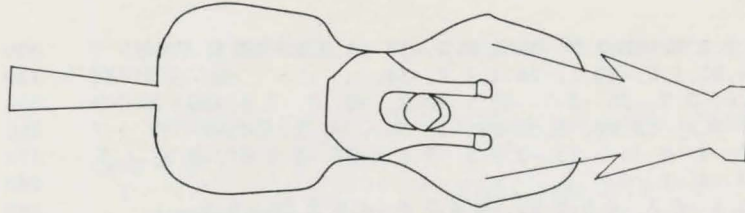
We might want to see a horse number on each horse. There is room for a number on the jockey's back; however, when the horses are scaled to fit the track, most computer terminals lack the resolution to make the number visible.

### Horse Program Code

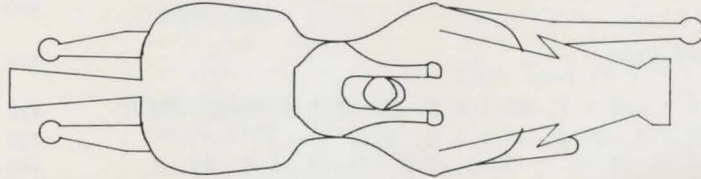
	Sequence Number
! HEAD	130
140 DATA 12,-14,-3.5,-7,-2,-8,-3,-2,-1,-1,-2,0,-2,0,2,-1,2,-2,1	140
DATA -8,3,-7,2,-14,3.5	150
! BODY	160
DATA 36,-9,2.7,-10,3.7,-12,4.5,-15,4.8,-17,4,-18,3,-22,3,-24	170
DATA 3.2,-25,4.4,-26,5,-28,4.8,-30,4.9,-31,4.3,-32,3.8,-33,3	180
DATA -33.5,2,-34,0.2,-42,1,-42,-1,-34,-0.2,-33.5,-2,-33,-3,-32	190
DATA -3.8,-31,-4.3,-30,-4.9,-28,-4.8,-26,-5,-25,-4.4,-24,-3.2	
DATA -22,-3,-18,-3	200
DATA -17,-4,-15,-4.8,-12,-4.5,-10,-3.7,-9,-2.7	210
! JOCKEY	220
DATA 31,-16,1,-13,1,-13,2,-16,2,-20,3,-22,3,-23,2.8,-24,2	230
DATA -24,1,0,-24,-2,-23,-2.8,-22,-3,-20,-3,-16,-2,-13,-2	240
DATA -13,-1,-16,-1,-16.4,-0.7,-16,0,-16.4,0.7,-16,1	250
DATA -15.3,0.5,-15.3,-0.5,-16,-1,-18,-1.2,-19,-1,-20,-0.5	260
DATA -20,0.5,-19,1,-18,1.2,-16,1	270
DATA 6,-17,-1,-17.5,-0.8,-18,-0.5,-18,0.5,-17.5,0.8,-17,1	280
DATA 7,-13,2,-12.5,2.2,-12,2,-11.9,1.5,-12,1,-12.5,0.9,-13,1	290
DATA 7,-13,-2,-12.5,-2.2,-12,-2,-11.9,-1.5,-12,-1,-12.5,-0.9	300
DATA -13,-1,-9999	310
! LEGS, RIGHT LEAD	320



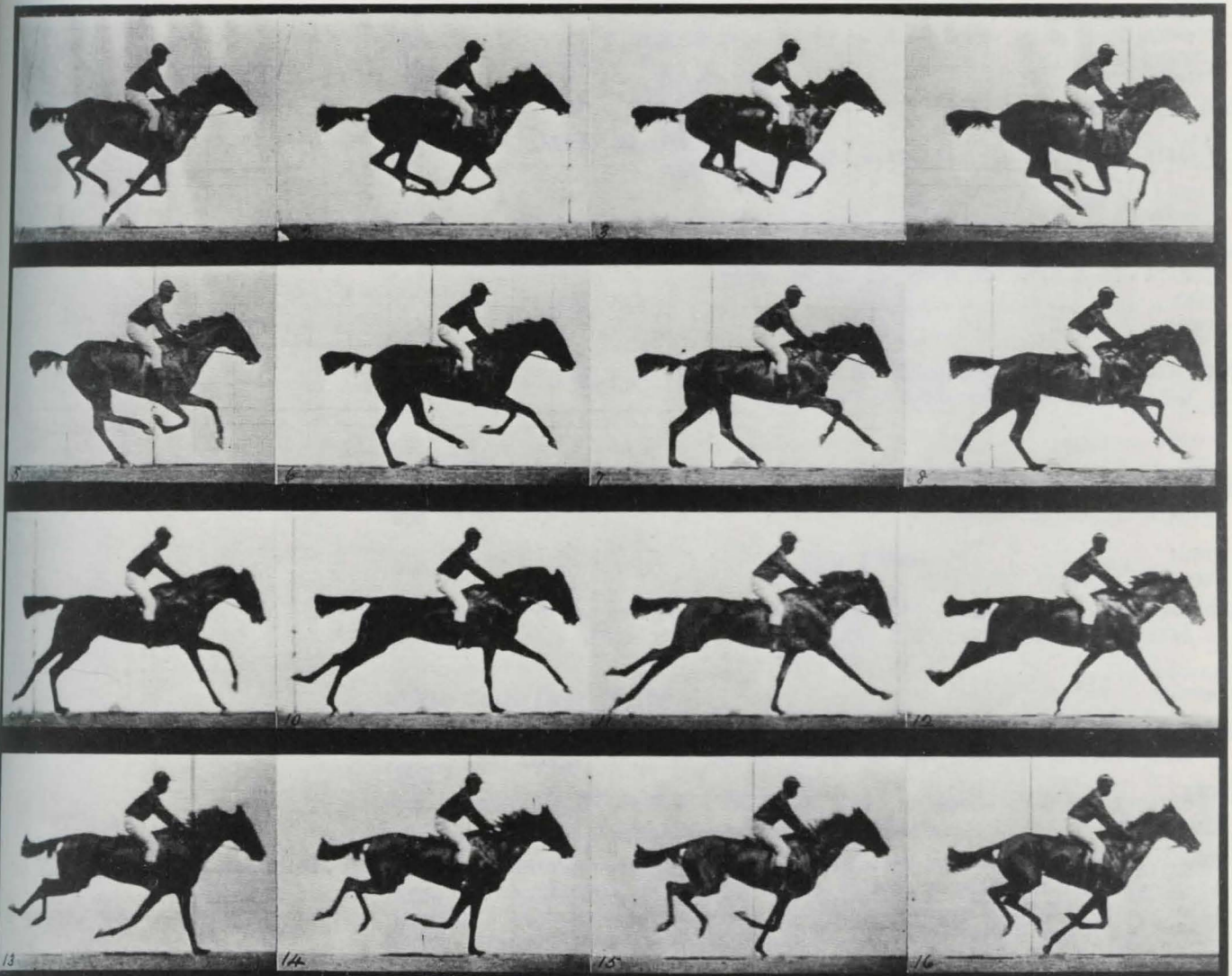
horse and jockey seen from above with  
legs gathered between strides



Striding horse and jockey seen from  
above



Muybridge photograph of jockey and  
striding horse



0, 3, -39.87	330
	340
5, -39, -3	350
4, -1	360
37, -2, 2, -1.5	370
	380
5, 2	390
	400
	410
9, 3, -39.87	420
	430
5, -39, -3	440
4, -1	450
3.87, 2, 2, 1.5	460
	470
2.25, -5, -2	480
	490
	500
	510
	520
-22, -0.75	530
	540
	550
0.75, -23	560
	570
	580
-21, -0.25	590
	600
	610
5, -0.75	620
99	630
	640
-22.75, 0.75	650
22, -0.5, -22,	660
	670
	680
	690
ch horse	710
	720
	730

```

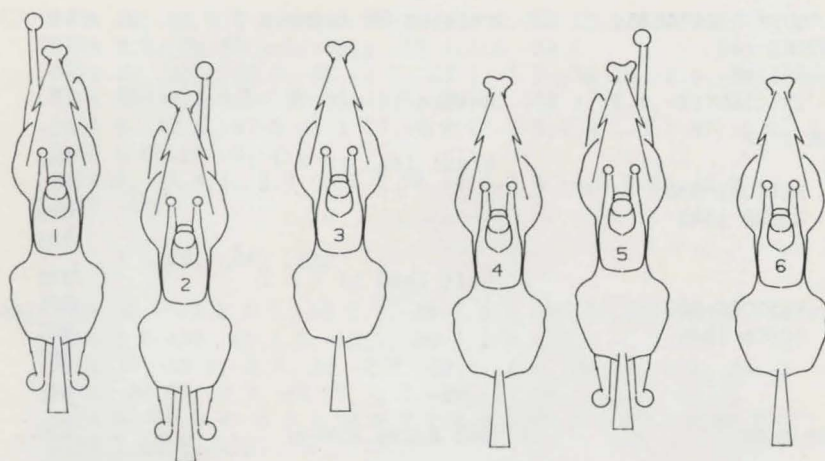
T = 90 * 0.0174533      ! rotated 90 degrees      740
RESTORE 140              750
GOSUB 1140               760
V = INT(RND(-2) * 3) + 1 ! Random integer V = 1, 2, or 3 770
CASE V OF                780
  1:                      ! Right lead if V = 1      790
    RESTORE 330           800
    GOSUB 1140            810
    ;                     820
  2:                      ! Left lead if V = 2      830
    RESTORE 420           840
    GOSUB 1140            850
    ;                     860
  ;                       870
CASE H OF                ! Get horse number        880
  1:                      890
    RESTORE 510           900
    ;                     910
  2:                      920
    RESTORE 530           930
    ;                     940
  3:                      950
    RESTORE 560           960
    ;                     970
  4:                      980
    RESTORE 590           990
    ;                    1000
  5:                    1010
    RESTORE 620          1020
    ;                    1030
  6:                    1040
    RESTORE 650          1050
    ;                    1060
  ;                      1070
GOSUB 1140              ! Draw number            1080
H = H + 1               1090
;                       1100
END                     1110

```

### Horse Code Description

	Sequence Number
Data describing the horse's head and body and the jockey are grouped, ending with -9999.	140
Legs for right and left leads are defined.	330
Numbers 1 through 6 are drawn on the jockeys' backs.	510
To test the horse-drawing program, six horses are drawn, with a little randomization to make them interesting to look at, as shown in the drawing of all six horses. Translation value X9 is between 23 and 27 for horse 1, between 38 and 42 for horse 2, etc. Y9 is between 50 and 70 for each horse. The scaling factor is 1, so the horses will be drawn "full	690





Example of variety possible in drawing horses

size" and the angle of rotation  $T$  is 90 degrees, which means that the horses will rotate counterclockwise 90 degrees about their noses.

The horse's head and body and the jockey are drawn. (Subroutine 1140 is the ROTATE, SCALE, AND TRANSLATE routine.) 750

A random value  $V = 1, 2$ , or  $3$  is computed. 770

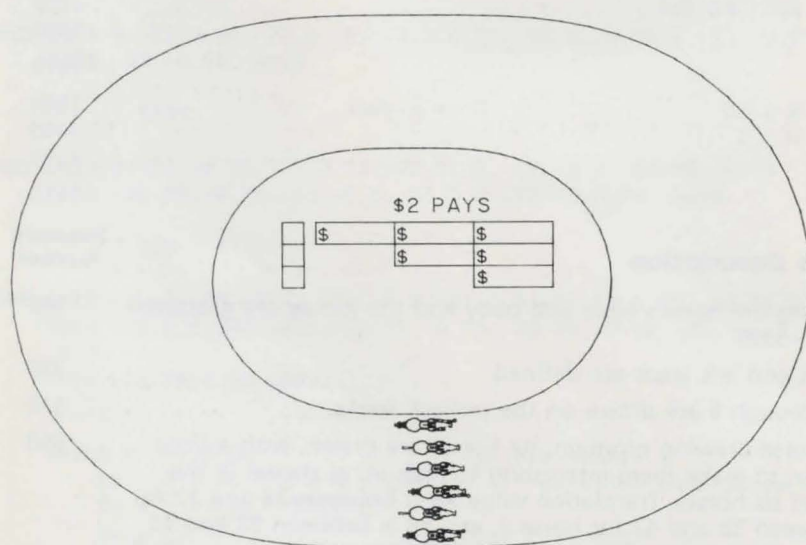
If  $V = 1$ , the horse will display a right lead. 790

If  $V = 2$ , the horse will display a left lead. 830

If  $V = 3$ , the legs will be gathered.

The numbers 1 through 6 are drawn on the jockeys' backs. 880

All put together, here's how the racetrack scene would look on your screen: track, tote board, horses, jockeys, and "announcer."



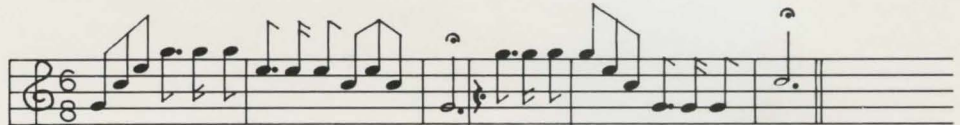
THEY'RE OFF!

The start of another race

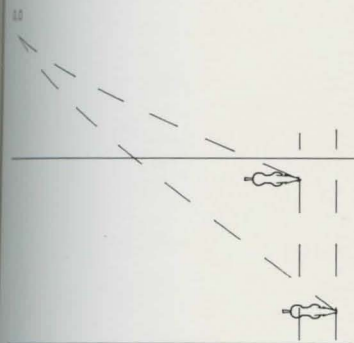
### Refinements

- Make the track as realistic as possible. Add red and white marking poles every  $\frac{1}{3}$  of a mile. Show a starting gate and a finish line. Color the infield green and the track brown. Randomly vary the colors of the horses so that most are chestnut but a few are gray or black. Jockey's pants are always white, but their shirts and caps are the colors of the racing stable to which the horse belongs.
- If your computer has a tone generator, play the "call to post" before every race. The music is provided here for you to use.

"Call to post" to be recreated with tone generator



- Randomly assign names to horses so that the announcer uses the horses' names instead of their numbers. If you can't come up with at least 54 racehorse names, look in the sports section of your newspaper.
- If the race is close, display a "photograph" of the finish, showing a side view of the horses with the winner's nose touching the finish line. In a photo finish, a vertical line appears on the photograph at the finish line and the camera is triggered as each horse passes this line. Be sure to show each horse's number on the saddle cloth behind his jockey's leg.
- Allow for the possibility of a thrown jockey or a lame horse roughly once every hundred races.
- Vary the lengths of different races from 6 furlongs ( $\frac{3}{4}$  mile) to a mile and  $\frac{1}{4}$ . Start the horses at the chosen length from the finish line. Do not move the finish line—it is always in front of the grandstand.
- At the start, around the turn, in the backstretch, and in the stretch, show the horses as they would be seen by spectators in the grandstand. At the finish, show the horses finishing from the point of view of a "railbird," a spectator who is standing along the rail at the finish line.
- The horses would be more realistically positioned on the track if they were not perpendicular to the line between their noses and the center of the track, but instead were parallel to a line tangent to the inside rail at the point where the aforementioned line intersects the rail. Make this correction.
- Because the horses are measured according to their angular position, a horse that is actually leading in the straightaways may not appear to be, as shown in the accompanying drawing. Correct for this problem.
- Give inside horses less distance to travel than outside horses. Let horses move to the rail to "save ground" and move out to pass slower horses. Avoid collisions.



Problem of misperception of which horse is leading





## ***Maze and Fantasy Games***



People have always had a fascination with the unknown. The feeling that “inside” information allows specialists to control mysterious powers persists to this day, along with the idea that supernatural beings sometimes appear to mortals and cause scary things to happen. Besides the excitement of exploring dark regions, events real and imagined outside the range of ordinary knowledge form the basis for many mystery stories, horror movies, and the so-called fantasy or adventure games. The setting for all such board and computer games is a maze.

In this chapter we will look at how mazes can be defined in computers and how to generate them. The explorer and computer-controlled fantasy game characters must be able to move about the maze, so a traversal algorithm is described. Then the use of “arrow” keys to control the explorer’s movements is explained.

Realistic-looking mazes add greatly to the fun of fantasy games. A section describing the maze runner’s view outlines a method for displaying walls using a one-point perspective projection technique. The *Refinements* section suggests improvements to the described algorithms.

### Simple Mazes

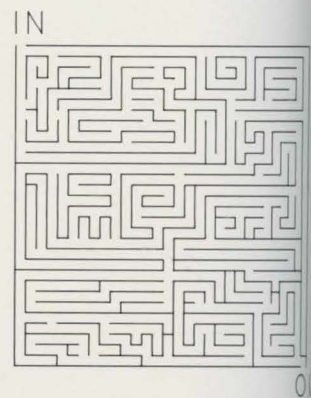
An ancient version of the maze, first played in Egypt, is Snakes and Ladders. A 10 x 10 board is used as a playing surface, on which squares are numbered from 1 to 100—1 to 10 reading from left to right on the bottom row, 11 to 20 from right to left on the next row, and so forth. Here and there on the board are snakes and ladders spanning two or more rows. Players move according to the spots on an ordinary die. When a turn ends at the foot of a ladder, the player moves upward to its top. If a turn ends at the tail of a snake, the player moves down to the head of the snake. The game ends when a player lands on square 100.

Snakes and Ladders has many features of a truly boring game—a complete absence of any player skill component, for example—but in a rudimentary way it is a maze game. A more common form of maze game is found in the entertainment section of many Sunday newspapers. This sort of maze is drawn as seen from above, with an entrance and an exit. The player uses a pencil to trace paths around the maze until he solves it. An example of this kind of maze is shown here.

This puzzle can be spoiled by the player’s looking ahead, “around corners,” as it were, which could not be done if he were in a three-dimensional maze such as the ones that used to be commissioned by European royalty as part of formal gardens. On the grounds of Hampton Court Palace, for instance, Henry VIII had a maze made of yew hedges eight feet high. If the king had included a treasure in the center of the maze and a few fierce monsters in the passageways, he would have invented one form of the fantasy adventure game.

### Fantasy Games

The setting for a fantasy computer game is, of course, a maze, a group of connected rooms or caverns that the player must explore in order to resolve a mystery—for example, where is the beautiful damsel? To do so, the player or maze



A maze game



runner has to develop ways to build up an ability to fight, bribe, or run away from a variety of terrifying beings while maintaining resources such as food and water so as not to starve to death. The usual way to gain strength or experience is to fight monsters, because they guard treasures or information that can be used to acquire food or weapons.

This type of game is well-suited to the computer because of its complexity and the game's need for controlled randomness in the location of hazards in the maze. Although some adventure games can be played on a board, the rules are extremely complex, often requiring a full-time referee. In computerized versions, the computer acts as referee, but the rules are not described to the player—they must be deduced.

### Defining a Maze Numerically

The simplest mazes are composed of a collection of square rooms, an example of which is shown here.

There are sixteen possible types of room, depending on how many walls there are to each room and what side each wall is on. The sixteen possibilities are illustrated here.

To represent the little maze shown above, a two-dimensional array  $M$  may be defined, having four columns and three rows. Then the number of each type of room could be put in  $M$ , as shown. This is not very satisfactory, because then to try to go south from  $M(I,J)$  it is necessary to perform the test

IF  $M(I,J) = 1$  OR  $M(I,J) = 2$  OR  $M(I,J) = 5$  OR  $M(I,J) = 6$  OR  $M(I,J) = 9$   
OR  $M(I,J) = 13$  OR  $M(I,J) = 14$  THEN  $J = J + 1$

A more efficient technique is to assign values to each wall and then add them up to get the room type. It doesn't work to choose

North = 1  
East = 2  
South = 3  
West = 4

because a room of type 3 could be composed of north and east walls or a south wall. Using powers of two will give a unique room type for each different room.

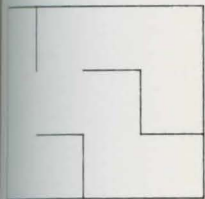
North = 1  
East = 2  
South = 4  
West = 8

The resulting room values are given in the Room Values chart.

To find out if it is possible to go in a given direction from the room  $M(I,J)$ , a simple computation is sufficient. For example, it is only possible to go north out of a room if there is no north wall, that is, if the room type is an even number.

IF  $M(I,J) + \text{INT}(M(I,J)/2)*2$  THEN  $J = J - 1$

This test looks at  $M(I,J)$  to see if it is an even number by comparing  $M(I,J)$  with two times the integer portion of half of  $M(I,J)$ . Since two times the integer



A simple maze

No walls	9	W & N
North wall	10	N, E & S
East wall	11	E, S & W
South wall	12	S, W & N
West wall	13	W, N & E
N & E	14	E & W
E & S	15	N & S
S & W	16	All walls

Sixteen types of square rooms, numbered 1 to 16

	I=	1	2	3	4
1		13	9	15	6
2		5	4	6	11
3		8	10	8	10

Two-dimensional array  $m$  indicating types of room in a simple maze, using system of rooms numbered 1 to 16



portion of half of any odd number is the even number immediately less than it, the test fails for any odd number.

On the other hand, if  $M(I,J)$  is even,  $\text{INT}(M(I,J)/2)*2$  is equal to  $M(I,J)$  and the test does not fail. In other words, for values 2 and 3, half the number is 1 and 1.5. In either case the INT of 1 or 1.5 is 1. Then multiplying by 2 gives the result 2 whether  $M(I,J)$  contains 2 or 3. In the first case the test is equal; in the second, not equal.

What is the test for no west wall? It is easily seen if the rooms are represented as binary numbers:

Decimal	Binary	W S E N
0	0000	0 0 0 0
1	0001	0 0 0 1
2	0010	0 0 1 0
3	0011	0 0 1 1
4	0100	0 1 0 0
5	0101	0 1 0 1
6	0110	0 1 1 0
7	0111	0 1 1 1
8	1000	1 0 0 0
9	1001	1 0 0 1
10	1010	1 0 1 0
11	1011	1 0 1 1
12	1100	1 1 0 0
13	1101	1 1 0 1
14	1110	1 1 1 0
15	1111	1 1 1 1

In the right-hand four columns, a zero denotes the absence of a wall in the indicated direction. To test if a move to the west is possible, it is only necessary to be sure that the room number is less than 8.

IF  $M(I,J) < 8$  THEN  $I = I - 1$

There is more than one way to test for an opening to the south:

IF  $M(I,J) < 4$  OR  $(M(I,J) > 7 \text{ AND } M(I,J) < 12)$  THEN  $J = J + 1$   
 or IF  $\text{INT}(M(I,J)/4) - 2*(M(I,J) > 7) = 0$  THEN  $J = J + 1$

The first test merely examines all the possibilities: if  $M(I,J)$  is 0, 1, 2, or 3 or 8, 9, 10, or 11, then it is one of the rooms without a south wall. The other test relies on the value of the truth of  $M(I,J) > 7$ . If  $M(I,J)$  is greater than 7, this expression has the value 1; otherwise it is zero. The expression  $\text{INT}(M(I,J)/4)$  equals zero if  $M(I,J)$  is 0, 1, 2, or 3; 1 if  $M(I,J)$  is 4, 5, 6, or 7; 2 if  $M(I,J)$  is 8, 9, 10, or 11; and 3 otherwise. So the whole expression equals zero only in the eight cases where the room has no south wall.

Finally, it is possible to move to the east as follows:

IF  $\text{INT}(M(I,J)/2) = \text{INT}(M(I,J)/4)*2$  THEN  $I = I + 1$

### Generating a Maze

Small mazes are not hard to make. They may be drawn on square-ruled paper, encoded with numbers between 0 and 15, written in DATA statements, and read

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Room values of 0 to 15, using powers of 2

into an array. This method is prone to room-coding errors, and the resulting game is not much of a challenge to the programmer either. He already knows what the maze looks like.

For practical purposes, most games require mazes too big to be encoded by hand, so a generating algorithm is necessary. Very large mazes may be built, even in computers with small memories, by making them multileveled. This requires that one or more rooms on each level of the maze contain a stairway leading down to the next level and another leading up. Only the level currently being explored is kept in memory. When the explorer moves to another level, the maze generator creates the new level in the same array used to hold the previous one.

Although a maze may be of any shape, the most storage-efficient computer mazes are rectangular or square. A 25 x 25 maze will consist of 625 rooms—clearly the rooms along the outside edges should be bounded by walls to keep the maze runner from straying out of the defined maze. If an array  $M(25,25)$  is defined such that the room  $M(1,1)$  is in the bottom left-hand corner and north is upward, then the following rules must be observed:

For  $X = 1, 2, \dots, 25$ , and for  $Y = 1, 2, \dots, 25$   
 All  $M(X,1)$  must have a south wall  
 All  $M(X,25)$  must have a north wall  
 All  $M(1,Y)$  must have a west wall  
 All  $M(25,Y)$  must have an east wall

The total number of walls inside the maze, the wall density, may be controlled by a probability function. Except for outside edges, if the probability of a wall is 0.5, then the expected distribution of the sixteen possible types of room is as follows: Out of every sixteen rooms, one will have no walls, four will have one, six will have two, four will have three, and one will have four walls, on the average.

This means that roughly one room out of every sixteen will be inaccessible. It may be useful to decide that no room will be composed of four walls, but this still does not mean that all rooms will be accessible, because there can be groups of rooms totally enclosed by walls, as illustrated here by a group of two rooms and a group of four.

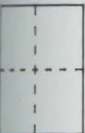
Depending on how the maze is to be used, it may not be essential that all rooms can be entered. If it is essential, the tedious process of checking that each room can be reached from any given room, say  $M(1,1)$ , can be done with the maze traversal algorithm explained in the next section.

The important question is whether the game calls for a new unknown maze every time the program is used or whether once defined, the maze does not change. In the latter case, mazes may be generated using different starting values for the random number generator and the wall probabilities until an acceptable maze is generated.

Here is a program code to generate and display a maze, followed by its description.

### Maze Generator Program Code

	Sequence Number
$X0 = 10$	! Number of columns 120
$Y0 = 10$	! Number of rows 130
$DIM M(X0,Y0)$	140



Enclosed groups of rooms in a maze



```

P = 0.25                ! Probability of a wall                150
M = 0                   ! Initialize maze                      160
Y = 1                   ! Y = 1, the bottom row                170
WHILE Y <= YO DO        ! For all Y,                          180
  X=1                   ! set X=1, the leftmost column          190
  WHILE X <= XO DO      ! For all X,                          200
    IF Y = YO THEN      ! If Y = the top row,                  210
      M(X,Y) = M(X,Y) + 1 ! there must be a north wall          220
    ;                                                            230
    IF X=XO THEN ! If X= the right-hand column,                240
      M(X,Y) = M(X,Y) + 2 ! there must be an east wall          250
    ;                                                            260
    IF Y = 1 THEN       ! If Y = the bottom row,              270
      M(X,Y) = M(X,Y) + 4 ! there must be a south wall          280
    ;                                                            290
    IF X = 1 THEN       ! If X = the left-hand column,         300
      M(X,Y) = M(X,Y) + 8 ! there must be a west wall          310
    ;                                                            320
    IF X>1 THEN ! West wall next room=east wall this room      330
      M(X,Y)=M(X,Y)+(INT(M(X-1,Y)/2)-INT(M(X-1,Y)/4)*2)*8      340
    ;                                                            350
    IF Y>1 THEN ! South wall this room=north wall lower room   360
      M(X,Y)=M(X,Y)+(M(X,Y-1)-INT(M(X,Y-1)/2)*2)*4             370
    ;                                                            380
    IF RND(1)<P AND Y<YO THEN ! Give this room a north          390
      M(X,Y) = M(X,Y) + 1 ! wall if random number < P          400
    ;                                                            410
    IF RND(1)<P AND X<XO THEN!Give room an east wall if         420
      M(X,Y)=M(X,Y)+2 ! random number < P and                   430
    ;                                                            440
    IF M(X,Y)<>INT(M(X,Y)/2)*2 THEN ! Draw plan view            450
      MOVE X*4+14,Y*4 ! of room                                460
      DRAW X*4+18,Y*4 ! Draw north wall if present             470
    ;                                                            480
    IF INT(M(X,Y)/2)<>INT(M(X,Y)/4)*2 THEN                      490
      MOVE X*4 + 18,Y*4                                         500
      DRAW X*4+18,Y*4-4 ! Draw east wall if present            510
    ;                                                            520
    IF Y = 1 THEN       ! Draw south wall if                    530
      MOVE X*4+14,Y*4-4 ! on bottom row                         540
      DRAW X *4+ 18,Y*4 - 4                                     550
    ;                                                            560
    IF X = 1 THEN       ! Draw west wall if                      570
      MOVE X*4+14,Y*4-4 ! on leftmost column                   580
      DRAW X*4 + 14,Y*4                                         590

```



;		600
X = X + 1	! Increase X	610
;	! Loop until X > number of columns	620
Y = Y + 1	! Increase Y	630
;	! Loop until Y > number of rows	640

### **Maze Generator Code Description**

	Sequence Number
X0 is set to the number of columns. Y0 is set to the number of rows.	120
M is defined to be the necessary size. P is defined to be the probability that an optional wall will occur.	140
The maze array M is cleared.	160
Y is set to indicate the bottom row.	170
For all values of Y	180
X is set to indicate the leftmost column.	190
For all values of X	200
The exterior walls are set. If Y = Y0, there must be a north wall. If X = X0, there must be an east wall. If Y = 1, there must be a south wall. If X=1, there must be a west wall. Note that in many versions of BASIC, lines 210–320 may be replaced by the single line	330
$M(X, Y) = M(X, Y) + (Y=Y0) + 2*(X=X0) + 4*(Y=1) + 8*(X=1)$	
If X is greater than 1, the west wall of M(X,Y) must be present if and only if M(X - 1,Y) has an east wall.	
If X is greater than 1, the west wall of M(X,Y) must be present if and only if M(X-1,Y) has an east wall.	330
If Y is greater than 1, the south wall of M(X,Y) must be present if and only if M(X,Y-1) has a north wall.	360
M(X,Y) may have a north wall if Y is less than Y0 (it must have a north wall if Y = Y0, see 240) and if a random fraction is less than P, and may have an east wall if X is less than X0 (see 210) and if another random fraction is less than P.	390
Having generated a room, draw it. If it has a north wall, draw it first. If it has an east wall, draw it.	450
If Y=1, draw a south wall. If X=1, draw a west wall.	530
Note that this is sufficient because if $X < 1$ and $Y < 1$ , the north wall of M(X,Y - 1) is the south wall of M(X,Y) and the east wall of M(X - 1,Y) is the west wall of M(X,Y).	

### **Solving or Traversing a Maze**

Now that we have generated a maze, we must provide a way to solve it. In the Sunday newspaper, solving means entering the maze at one point and moving to an exit. In fantasy games, instead of an exit, the maze may have a goal (such as a treasure) toward which the explorer is moving. In addition, the computer may take the role of a nasty monster trying to attack the player. On the other hand, it may be a creature guarding a secret needed by the explorer. It would then be

the creature's role to try to escape to a "safe" room. In any case, the following maze-solving techniques may be used to enable the participants to traverse a maze.

One method of traversing a maze is to maintain consistent contact with the walls on one side. In a maze of this type, as illustrated, the runner, facing south, keeps his left hand on the walls and follows the dotted-line path to the goal. Had the runner used his right hand, he would have arrived more quickly at the goal. In this maze the runner would have solved the maze one way or the other.

In the next maze shown, he couldn't have found the solution with his left hand on the wall. Obviously, some other technique must be tried.

Another way to solve a maze is to move randomly, marking the direction taken out of each room "with a piece of chalk." Each time a room is reentered, the old direction out is erased and the new one marked. Since motion is random, the new direction may be the same as the previous one. Rooms with two walls can be passed through. Rooms with three walls require that the runner turn around. Rooms with one wall require an equal probability of going in any of the three possible directions. A room without walls (type 0) requires an equal probability of going in any of the four possible directions.

Eventually, the runner will come across the goal, at which time he has solved the maze. Subsequently, to get to the goal he follows the arrows he has marked, because the last arrow drawn in every room is always correct. Of course, if there is more than one path to the goal, this method will not necessarily find the shortest.

The only way to find the shortest path is to be prepared to try all possible paths. Clearly, once a path of a given length has been found, any path under investigation may be ignored once it is found to be longer.

Look at this next maze. Is it possible to get from the lower left corner (column 1, row 1) to the goal (column 2, row 4)?

Start at column 1, row 1

Move north to column 1, row 2

Move north to column 1, row 3

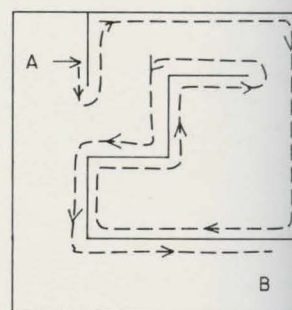
Move north to column 1, row 4

Move east to column 2, row 4

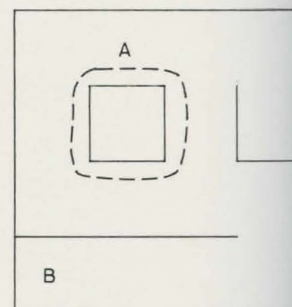
Given this top view of a small maze, the solution is easy. But imagine yourself standing in room C1,R1 (column 1, row 1) looking north. Your view would be something like that shown in the first perspective drawing here.

Suppose there is no lighting in the maze except the candle you are holding. The candle barely lights more than the room you are standing in. You can only see enough to go north, as you can imagine from looking at the second perspective drawing.

So you go north. Now you can see that you have a choice of continuing north or going east. Which do you choose? You have no information about which way to go, so you must plan to try both paths. This means that after you have exhausted one path by reaching a three-walled dead-end room, you must come back and try the other. However, if you elected to go east at C1,R2, you



Successful traversal of a maze "keeping one hand on the wall"



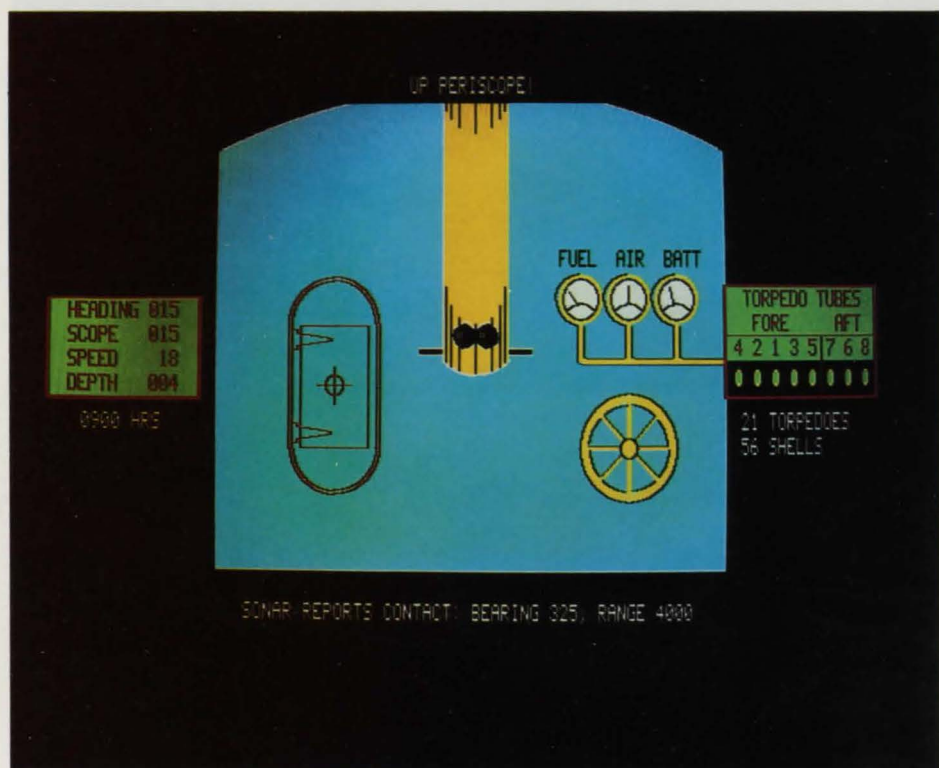
Unsuccessful traversal of a maze "keeping one hand on the wall"

		COLUMN			
		1	2	3	4
ROW	4		GOAL		
	3				
	2				
	1	START			

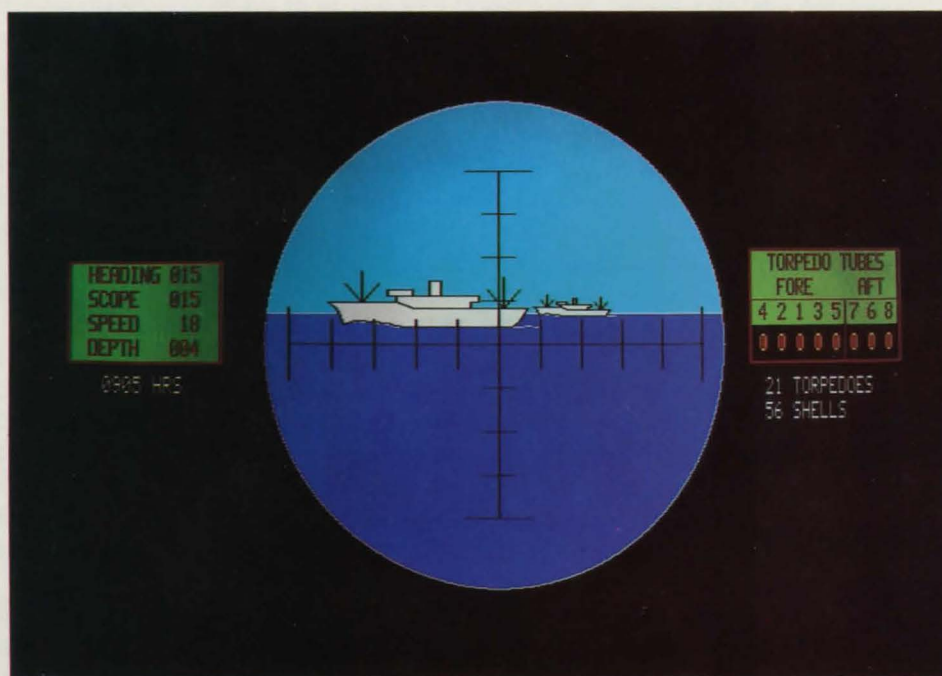
Maze in the form of a two-dimensional array



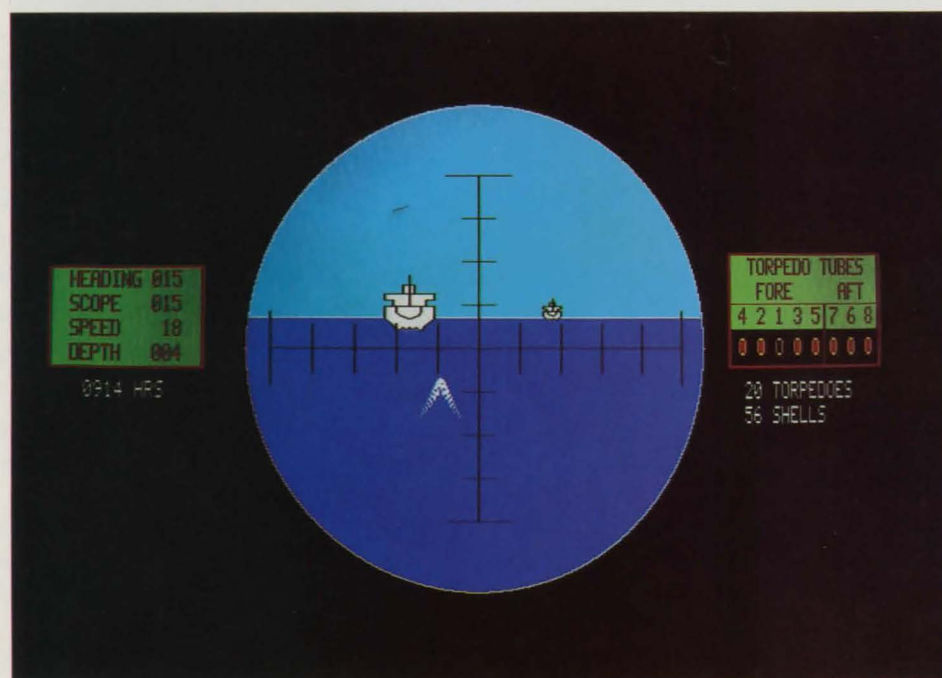
Scenes from a submarine game. Here is the player's view of the controls of the submerged sub. Notice how the periscope tube is made to appear cylindrical by the addition of a few vertical lines.





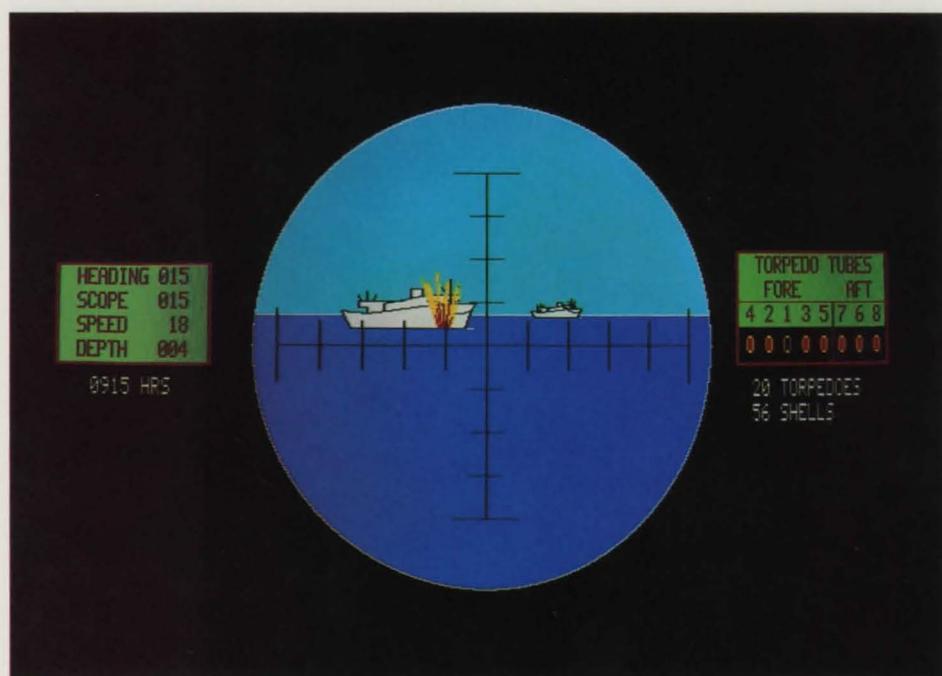


*A convoy has been sighted.*

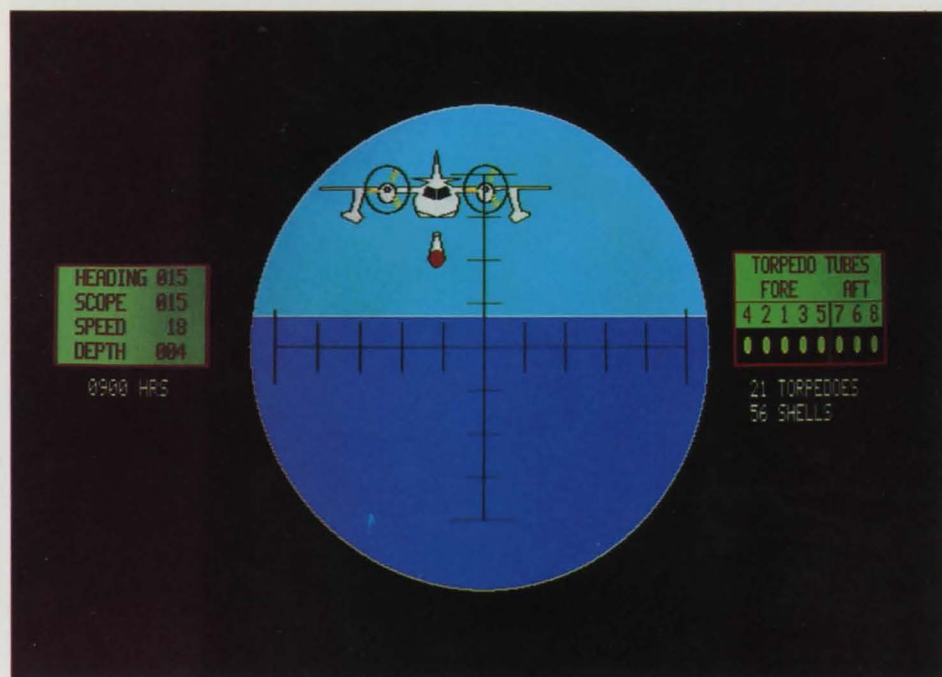


*Having positioned the sub behind the convoy, the commander has fired a torpedo from the center fore tube.*

Although the convoy is taking evasive action, the torpedo finds its mark.



A coastal command seaplane (or "flying boat," as it is called in the RAF) drops an aerial torpedo, which is seen through the periscope by the submarine commander.



# THE ENCHANTED MAZE



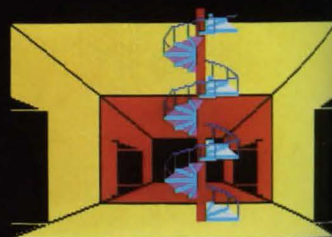
What is your name? > Zargon



© 1980, K. S. Reid-Green

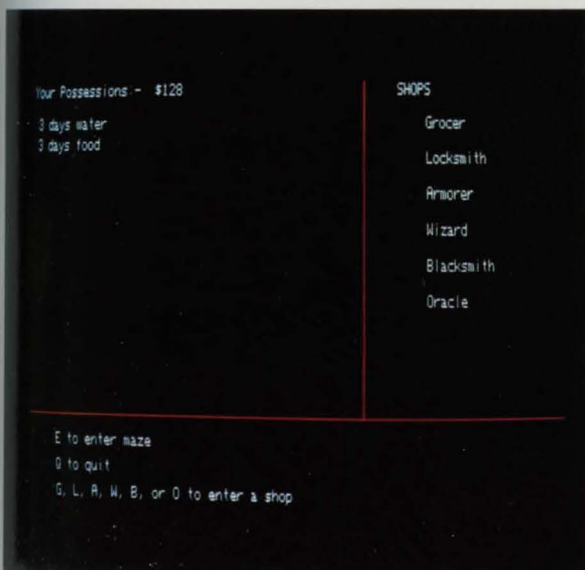
The title frame of the *Enchanted Maze* includes the monsters that inhabit it. Above the sign-in ("Zargon") is the dreaded and powerful Wanderer, who can move to intercept unwary maze explorers.

You are carrying      3 days food      experience 0.00  
                                  3 days water      \$128  
                                  strength 10.00

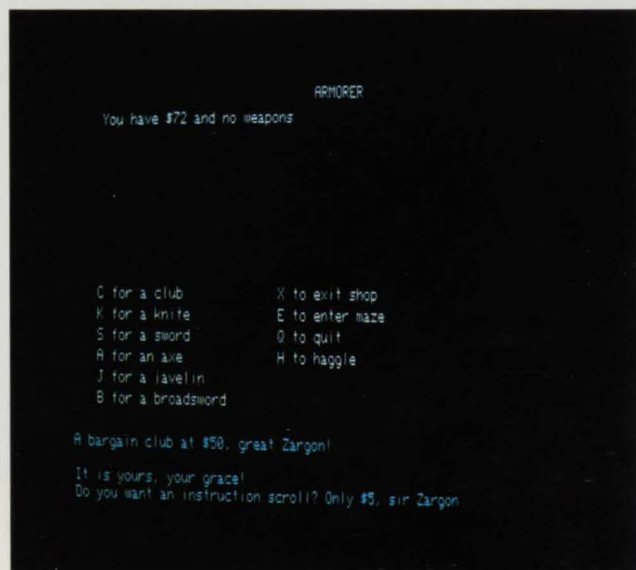


A new explorer is given three days' food and water, ten strength points, and money between \$50 and \$150. He can see one room ahead as there is no wall. The spiral staircase leads to a village where he can spend his money.

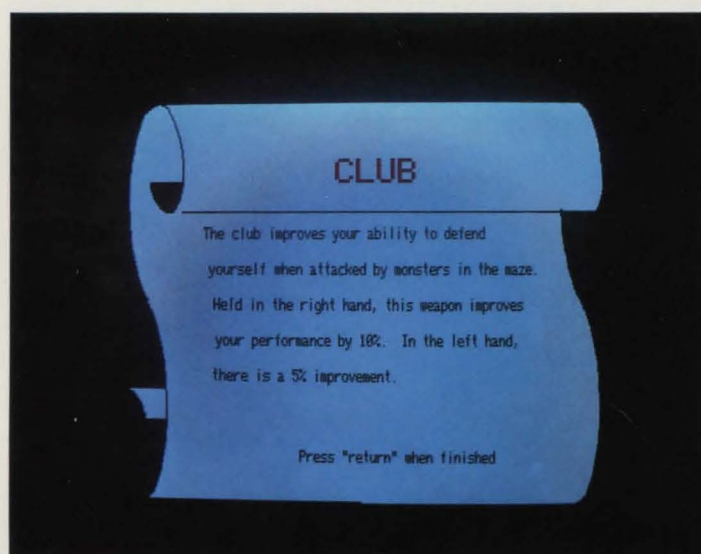




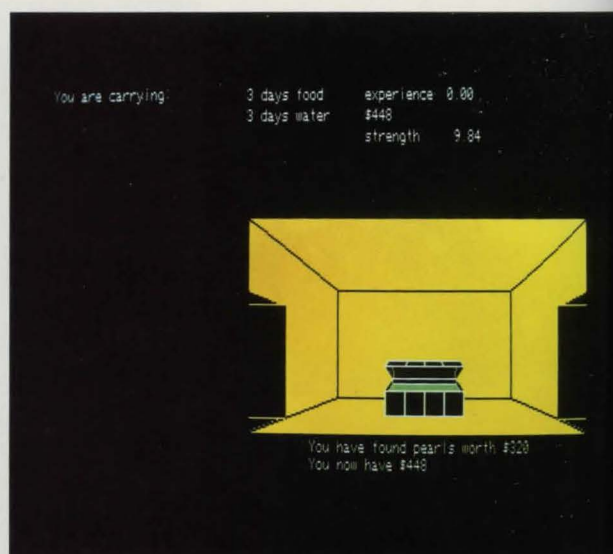
In the village, the explorer can choose to enter a shop, return to the maze, or quit the game.



This explorer has \$72 to spend in the armorer's shop. Having chosen to take a club for \$50 rather than haggle with the armorer, the explorer may buy an instruction scroll for an additional \$10.

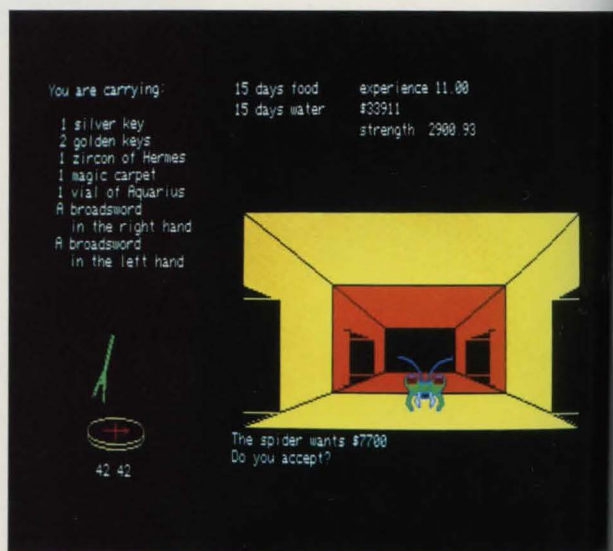


*This instruction scroll describes the benefits of owning a club.*

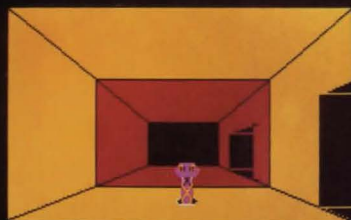


*The explorer has found an unguarded treasure.*

*A much more experienced explorer is carrying keys to get through doors, a zircon of Hermes that he can leave in the maze and later return to at will, a magic carpet that lets him move at once to the village, a vial of Aquarius to provide him with unlimited water, and two broadswords. He also has a divining rod that points to a water hole (no longer useful since he bought the vial of Aquarius) and a compass to show him direction and room numbers. He has just offered to bribe the spider.*



you are carrying: 3 days food experience 0.00  
 3 days water \$448  
 strength 9.86

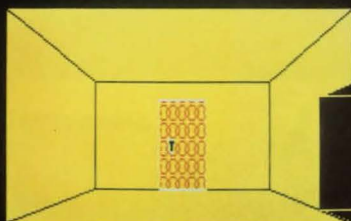


Fight, Bribe or Run from this snake?

Back in the maze, the explorer has encountered a snake. His options are to fight, bribe, or run.

you are carrying: 15 days food experience 22.00  
 15 days water \$33911  
 strength 2876.78

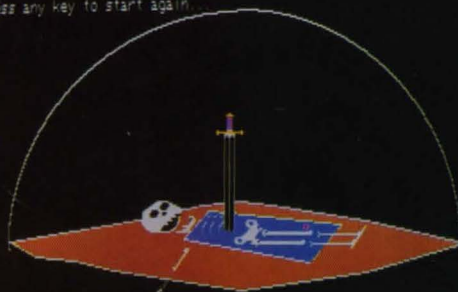
1 silver key  
 2 golden keys  
 1 zircon of Hermes  
 1 magic carpet  
 1 vial of Aquarius  
 A broadsword  
 in the right hand  
 A broadsword  
 in the left hand



44 43

The explorer is facing a door and must use a key to pass through it. There is a chance the key might break in the lock.

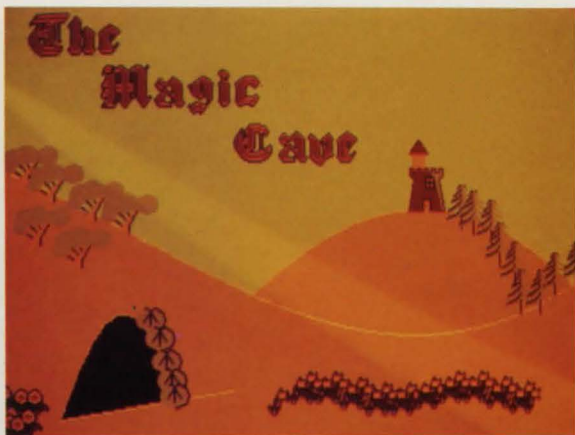
Press any key to start again.



At this very moment, a set of bones gradually rots alone in THE ENCHANTED MAZE.

This is the fate of an overzealous explorer. Maybe he was caught by the Wanderer or ran out of water. Various messages are composed to commemorate an explorer's demise, based on a random selection of clauses.

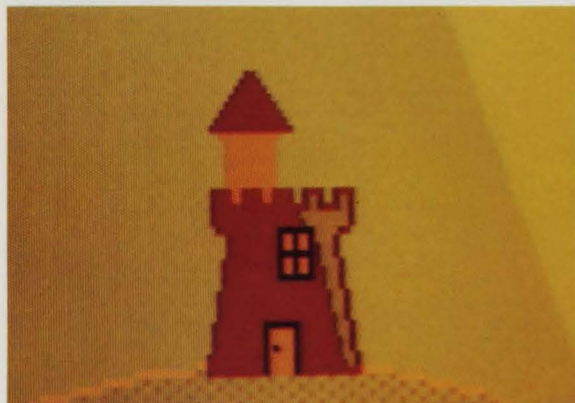




Unfinished opening display for Brian Astle's fantasy adventure game, "The Magic Cave." The final picture included a path to lead the eye from the tower to the cave, more randomness in the appearance of the flowers, and a more detailed cave.



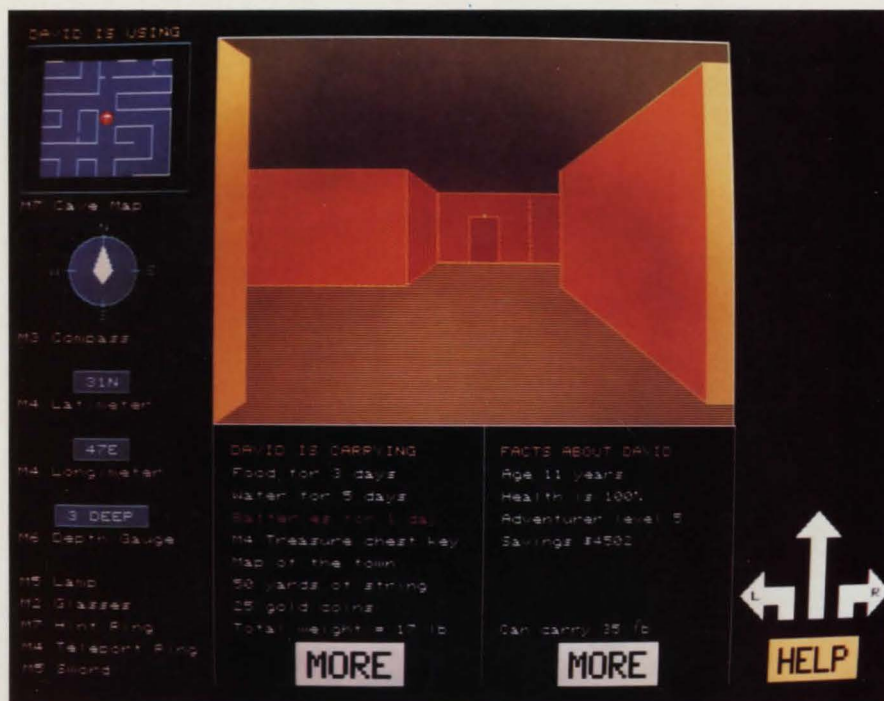
Attention to detail in the title includes a black outline around each letter.



Colors beyond the specifications of the display screen can be generated by alternating pixels of different colors. The grass below the tower is made up of green and cyan pixels.



Complete "Magic Cave" Opening



As the explorer moves around the maze, he encounters doors that can only be opened if he is carrying a key. Since there is a chance that a key may break, the explorer may become trapped.




From time to time, the explorer will encounter monsters that he may choose to fight. If he wins, he will find that most monsters are guarding a treasure chest.




After defeating a monster, the explorer sees a treasure chest that he may wish to open.

DAVID IS USING



M7 Cave Map



M3 Compass

34N

M4 Latimeter

47E

M4 Longimeter

3 DEEP

M6 Depth Gauge

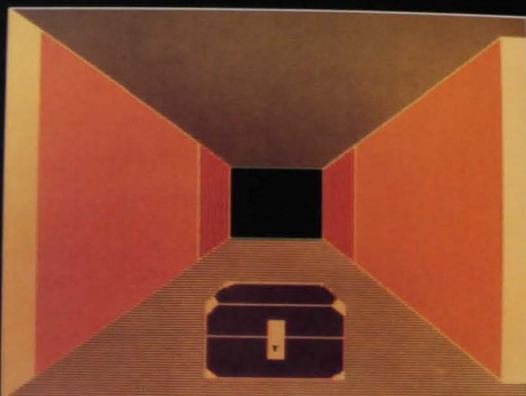
M5 Lamp

M2 Glasses

M7 Hint Ring

M4 Teleport Ring

M5 Sword



OPEN BOX

DAVID IS CARRYING

Food for 3 days

Water for 5 days

Batteries for 1 day

M4 Treasure chest key

Map of the town

50 yards of string

25 gold coins

Total weight = 17 lb

MORE

FACTS ABOUT DAVID

Age 11 years


Health is 100%

Adventurer level 5

Savings \$4502

Can carry 35 lb


MORE



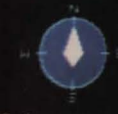
HELP

Most chests contain gold, silver, or precious gems that add to the explorer's ability to buy better weapons. Occasionally a chest may contain a weapon or a magic spell that the explorer can use against monsters. Rarely the chest may contain a danger, such as poisonous gas or another monster.

DAVID IS USING



M7 Cave Map



M3 Compass

34N

M4 Latimeter

47E

M4 Longimeter

3 DEEP

M6 Depth Gauge

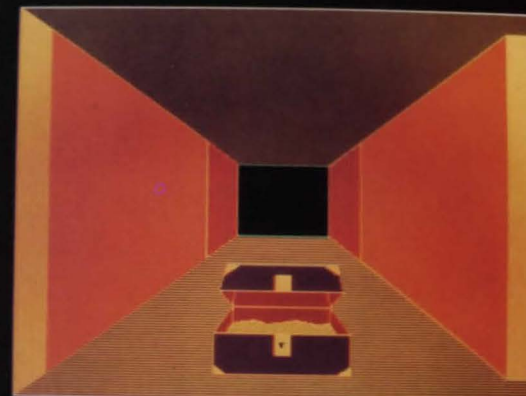
M5 Lamp

M2 Glasses

M7 Hint Ring

M4 Teleport Ring

M5 Sword



TAKE GOLD

SHUT BOX

DAVID IS CARRYING

Food for 3 days

Water for 5 days

Batteries for 1 day

M4 Treasure chest key

Map of the town

50 yards of string

25 gold coins

Total weight = 17 lb

MORE

FACTS ABOUT DAVID

Age 11 years


Health is 100%

Adventurer level 5

Savings \$4502

Can carry 35 lb

MORE

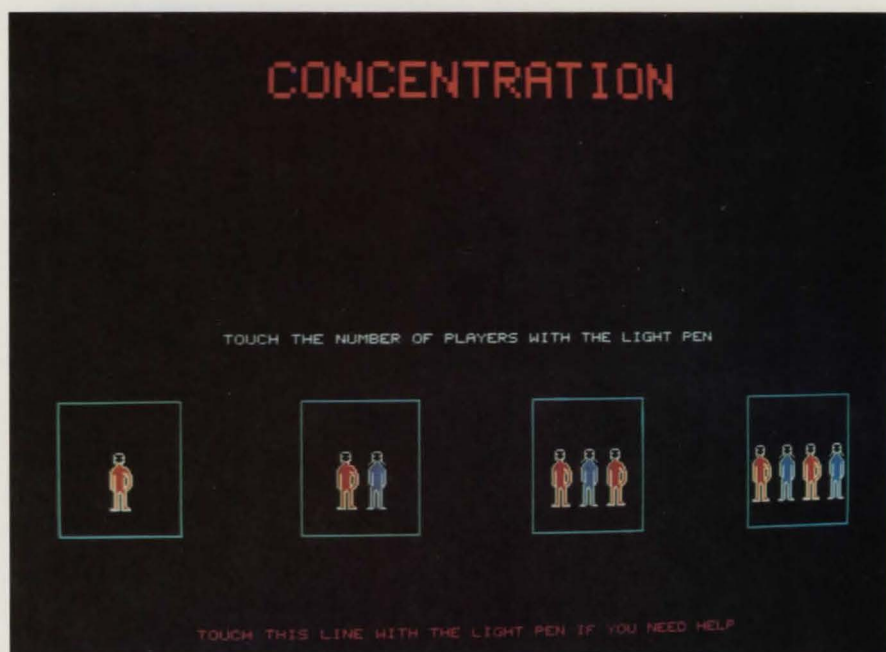


HELP

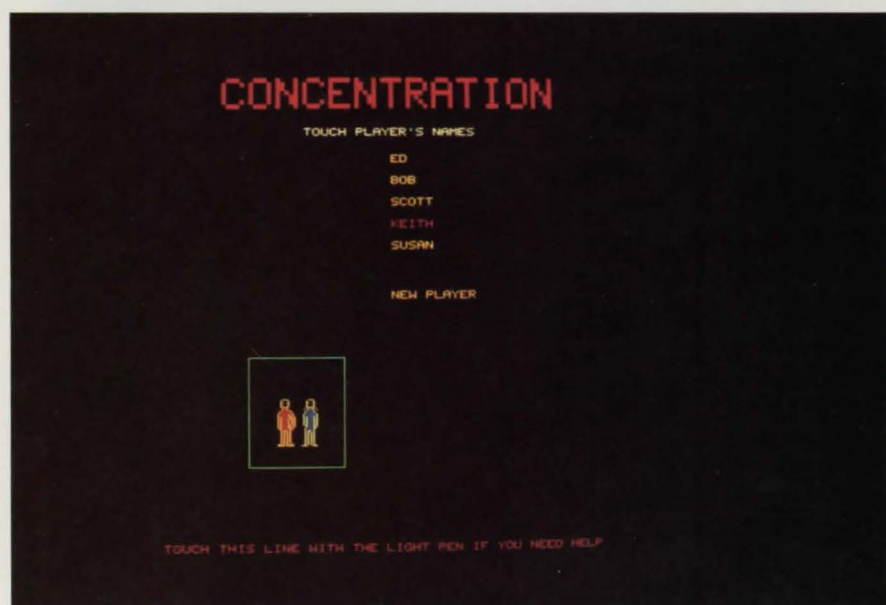
*The beginning moments of a Space Race game with the Earth and Moon at left and the spacecraft below the Earth on its way to Mars. The indicators show: 1) fuel usage, 2) the craft's heading and the fact that the main thruster is firing, 3) the velocity vector, and 4) the number of orbits completed.*



The opening frame of this Concentration game requests the number of players.

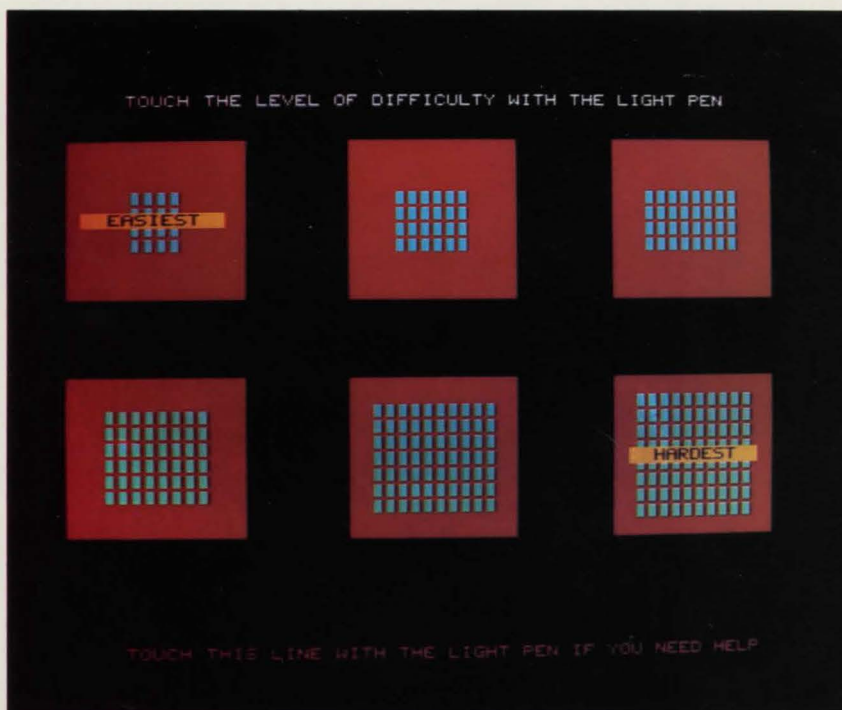


Two players will play Concentration. A list of previous players is displayed so that the opponents in this game may be identified.





Players decide on the level of difficulty of the game.



Having chosen the simplest game, Scott has been selected at random to play first.



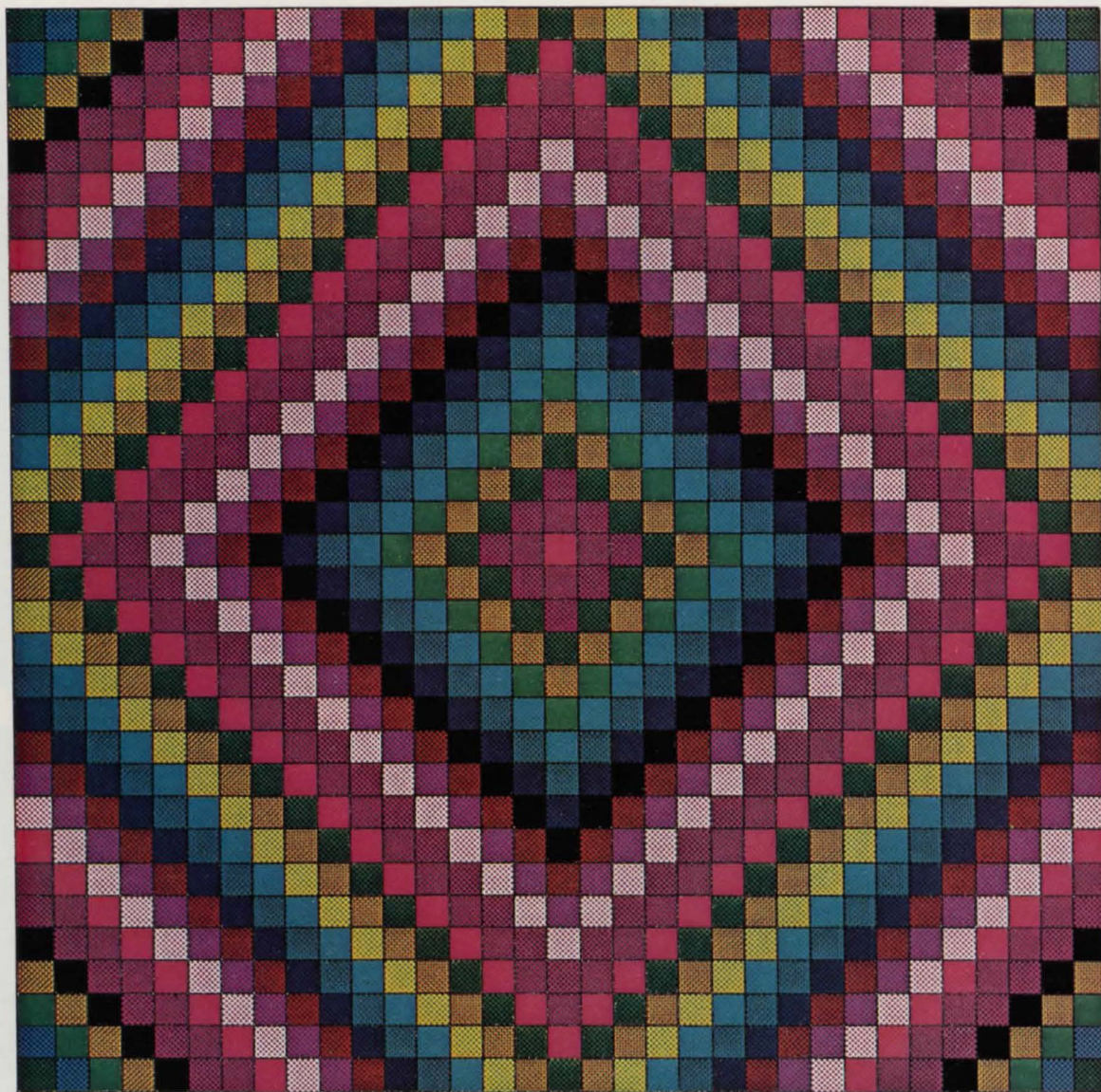
Part way through the game, Keith is ahead of Scott 2 to 1. Scott needs to find another "dog."



fireworks are displayed as Keith wins.

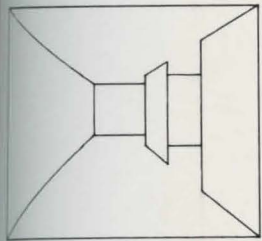




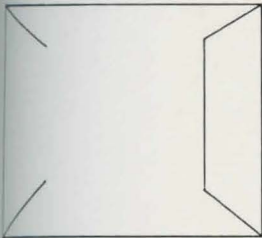


*Dawn Kleinfeld generated this quilt pattern from her hobbyist program. It uses random numbers to select compatible colors.*

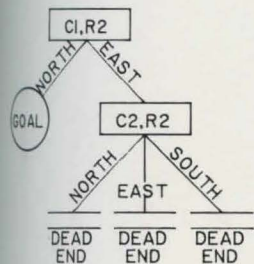




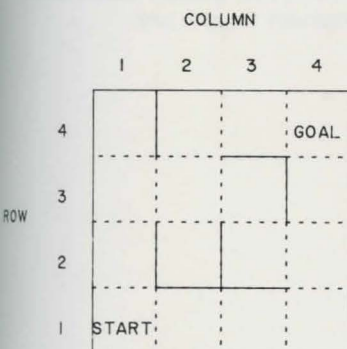
View as seen from room C1, R1 of  
maze looking north



View from same spot as if lit only by a  
candle



A tree of a simple maze with only one  
solution



A maze with more than one solution

are confronted with three more choices at C2,R2. If you keep careful notes as you make choices, you may try all the paths until you reach the goal.

So long as you don't have to make a choice, you don't need any notes. When you reach a dead end, you know that the path isn't any good, so you go back to your most recent note, cross it off, and pursue the next most recent.

Beginning at C1,R1 you move to C1,R2. Now you have a choice, so you write

AT C1,R2 GO NORTH  
AT C1,R2 GO EAST

and go east, so you delete that note. At C2,R2 you have a three-way choice, so you add to your notes

AT C1,R2 GO NORTH  
(AT C1,R2 GO EAST) (Note deleted)  
AT C2,R2 GO NORTH  
AT C2,R2 GO EAST  
AT C2,R2 GO SOUTH

You go south after deleting that note and end up at C4,R1, a dead end. So you go back to C2,R2, where you go east after crossing off that note. This path leads to the dead end at C4,R4. Go back to C2,R2, thence north to the dead end at C3,R4, crossing off the north note. Now the notes look like this:

AT C1,R2 GO NORTH  
(AT C1,R2 GO EAST)  
(AT C2,R2 GO NORTH)  
(AT C2,R2 GO EAST)  
(AT C2,R2 GO SOUTH)

Go back to C1,R2, cross off this note and go north, which leads to the goal.

Another way to keep notes which amounts to the same thing, is to draw a "tree" (a simple tree as shown here) and follow all the branches of the tree until the goal is found.

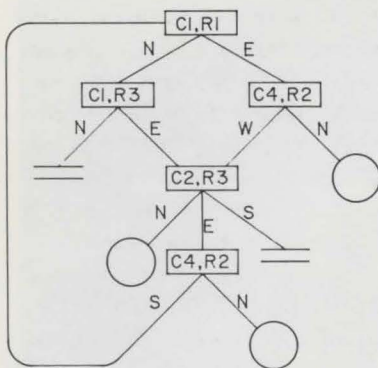
How does the tree look if a maze has more than one solution? Look at the next maze:

Its tree is more complex.

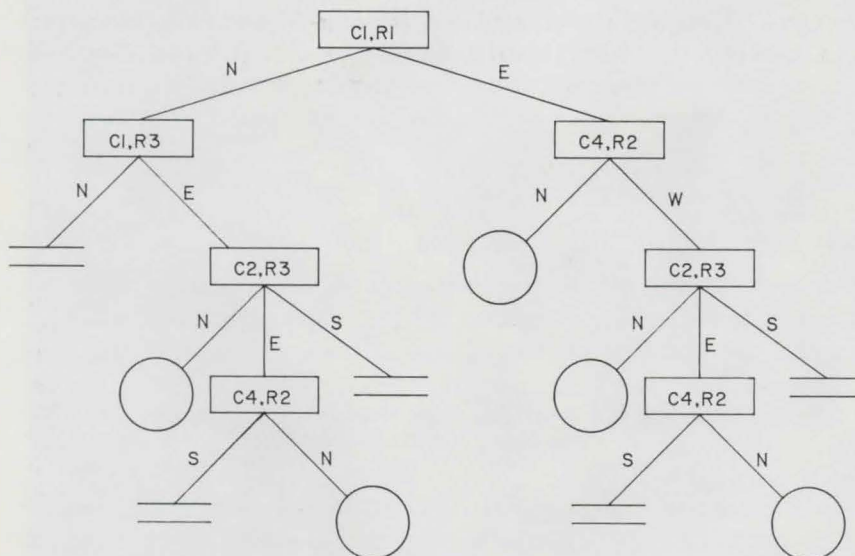
This is not a true tree because the maze runner can return to the starting point and can get to C2,R3 from two different paths. To make a tree out of it, it is reasonable to say that any path that returns to the start is the same as a dead end.

Also, instead of showing that two paths lead to the same C2,R3 node, the true tree is drawn "redundantly": that is, a given node is shown separately for each path that reaches it. Study the redundant tree shown here and note how the C2,R3 node is pictured twice—once as it is approached from the C1,R3 node and then as it is approached from the C4,R2 node.

In a redundant tree, all paths lead to the goal or to a dead end. Which is the shortest path to the goal? If a maze runner goes down all paths in the tree, counting rooms from the start each time, following the left branch first, he first comes to the goal by going north from C1,R1 to C1,R3, then east to C2,R3, then north to the goal, having passed through seven rooms.



A tree of a maze with more than one solution



Redundant tree, in which a node (C2,R3) reached from different paths is shown separately for each path



Now he knows after having followed a path for six rooms without reaching the goal that it is not going to be any shorter than the original path, so the next may be tried. If a shorter path is found, naturally that will be the one used for subsequent comparisons until the entire tree has been traversed. To keep from following redundant paths, we can say that if a room has already been entered, a path that has not reached the goal is being tried again.

The next program will test a maze to see if it can be solved. In order not to waste computer memory, some thought should be given to the maximum required size of the note pad S. If a room is type zero (no walls), the runner knows he need not retrace his steps, but he has three other choices. If all rooms were type zero, the maximum note pad would need space for three times the number of rooms in the maze, or  $3 * X0 * Y0$ .

The outside walls must be taken into account, however. In a maze having only outside walls, the note pad must hold information for the outside rooms plus  $3 * (X0 - 2) * (Y0 - 2)$  inside room notes. Around the outside edge, there is a maximum of two choices in every room except the corners, in which there is no choice. Along each of the north and south edges there are  $(X0 - 2)$  rooms with two choices and  $(Y0 - 2)$  along the east and west edges.

Two choices times two edges means that the note pad needs room for  $2 * (2 * (X0 - 2) + 2 * (Y0 - 2))$  notes for the edges plus  $3 * (X0 - 2) * (Y0 - 2)$  for the interior rooms. So S need not contain more than

$3 * (X0 - 2) * (Y0 - 2) + 2 * (2 * (X0 - 2) + 2 * (Y0 - 2))$   
or  $3 * X0 * Y0 - 2 * (X0 + Y0) - 4$  notes.

### **Solve a Maze Program Code**

```

DIM S(3*X0*Y0 - 2*(X0+Y0) - 4) ! Make space for notes           660
X9 = 10                        ! Use X9,Y9 to contain coordinates 670
Y9 = 10                        ! of maze goal                     680
X = 1                          ! X,Y are starting coordinates    690
Y = 1                          !                                 700
I = 0                          ! I = number of active notes      710
W = M(X,Y)                    720
IF W ≤ 0 THEN                  ! If starting room has no walls, leave note 730
  I = I + 1                    740
  S(I)=10000+100*X+Y ! Save direction, X and Y in one variable 750
;                               760
D = 4                          ! Set starting direction          770
IF W - INT(W/4)*4 = 2 THEN ! so that runner has back to 780
  D = D + 4                    ! a wall, if possible            790
;                               800
IF W = 4 OR W = 12 THEN       810
  D = D - 3                    820
;                               830
IF W = 8 THEN                  840
  D = D - 2                    850
;                               860
REPEAT                          ! Begin solution                870
  J = 0                        880

```



```

WHILE J < 4 DO      ! Leave notes to explore all openings      890
  K = J - 1          900
  IF J < 2 THEN      910
    K = K + 3*J + 5  920
  ;                  930
  IF INT(W/2↑(J + 1))*2 = INT(W/2↑J) AND D<>K THEN 940
    I = I + 1        950
    S(I) = 10000*2↑J + 100*X + Y 960
  ;                  970

  J = J + 1          980
;                    990
M(X,Y) = -1          ! Mark this room ''visited''             1000
REPEAT               1010
  IF I = 0 THEN      ! If no notes, maze has no solution      1020
    MOVE 0,90        1030
    "NO SOLUTIONGGG" 1040
  END                1050
  ELSE               ! Isolate direction and coordinates      1060
    D = INT(S(I)/10000) ! from latest note                    1070
    X = INT(S(I)/100)-2D*100 1080
    Y = S(I) - INT(S(I)/100)*100 1090
    I = I - 1        ! Delete note                            1100
  REPEAT             1110
    GOSUB 1180        ! Move and test for solution            1120
  UNTIL W<= 0 OR (W<>INT(W/3)*3 AND W<>5 AND W<>10) 1130
;                    1140
  UNTIL W<>7 AND W<>11 AND W<>13 AND W<>14 1150
UNTIL I<>I           ! Continue until solved                   1160
  ! MOVE AND TEST FOR SOLUTION 1180
GOSUB 1880           ! Display runner in maze                 1190
W = M(X,Y)           ! Get room data                          1200
M(X,Y) = -1          ! and flag "visited"                     1210
IF W<3 OR W=4 OR W=5 OR W=8 OR W=10 THEN ! See Note 5        1220
  IF D=2 THEN        ! If room has 0 or 1 wall or is a corridor, 1230
    X = X + 1        ! increase X if D = east                 1240
  ;                  1250
  IF D = 8 THEN      ! Decrease X if D = west                 1260
    X = X - 1        1270
  ;                  1280
  IF D = 1 THEN      ! Increase Y if D = north                1290
    Y = Y + 1        1300
  ;                  1310

```

```

IF D = 4 THEN          ! Decrease Y if D = south      1320
  Y = Y - 1           1330
;                      1340
;                      1350
IF W=INT(W/3)*3 AND W> 0 THEN ! If room has 2 adjacent walls, 1360
  IF (D=1 AND W=9) OR (D=4 AND W=12) THEN ! adjust D, X, and Y 1370
    X = X + 1         1380
  ;                  1390
  IF (D = 1 AND W = 3) OR (D = 4 AND W = 6) THEN      1400
    X = X - 1     1410
  ;              1420
  IF (D = 2 AND W = 6) OR (D = 8 AND W = 12) THEN    1430
    Y = Y + 1     1440
  ;              1450
  IF (D = 2 AND W = 3) OR (D = 8 AND W = 9) THEN      1460
    Y = Y - 1     1470
  ;              1480
  IF (D = 1 AND W = 3) OR (D = 4 AND W = 6) THEN      1490
    J = 8         1500
  ;              1510
  IF (D = 2 AND W = 3) OR (D = 8 AND W = 9) THEN      1520
    J = 4         1530
  ;              1540
  IF (D = 1 AND W = 9) OR (D= 4 AND W = 12) THEN      1550
    J = 2         1560
  ;              1570
  IF (D = 2 AND W = 6) OR (D = 8 AND W = 12) THEN    1580
    J = 1         1590
  ;              1600
  D = J          1610
;                1620
IF W = 7 THEN    ! If room has 3 walls,      1630
  X = X - 1      ! adjust D, X, and Y        1640
  D = 8          1650
;                1660
IF W = 13 THEN   1670
  X = X + 1      1680
  D = 2          1690
;                1700
IF W = 11 THEN   1710
  Y = Y - 1      1720
  D = 4          1730
;                1740
IF W = 14 THEN   1750
  Y = Y + 1      1760
  D = 1          1770
;                1780

```

```

IF X = X9 AND Y = Y9 THEN ! If solved, 1790
    GOSUB 1880 ! display maze runner 1800
                ! and end
MOVE 0,90 1810
PRINT "SOLVEDGGG" 1820
END 1830
; 1840
W = M(X,Y) ! Flag room and exit subroutine 1850
RETURN 1860

```

**Solve a Maze Code Description**Sequence  
Number

Make space for notes. 660

Let M(X9,Y9) be the maze goal. 670

Start the runner at a given X,Y. 690

Let I be the number of notes on the note pad. 710

It is necessary to resolve a special case at the beginning of the maze. When the maze runner starts, he must establish a valid direction in which to move. If he starts in a room without walls, a note about the path behind him is necessary; otherwise it may not get explored. If the runner starts with a wall behind him, no note is needed.

If a table of room types is built and the rooms are classified according to the wall with the smallest direction number, all odd-numbered rooms have a north wall.

Type	1	2	4	8
0				
1	x			
2		x		
3	x			
4			x	
5	x			
6		x		
7	x			
8				x
9	x			
10		x		
11	x			
12			x	
13	x			
14		x		
15	x			
LET	D = 4	D = 8	D = 1	D = 2

Rooms 2, 6, 10, and 14 have an east wall, rooms 4 and 12 a south wall, and room 8 a west wall. Most rooms have a north wall—for these the maze runner's starting direction D must be south. If D = 4, half the possibilities are resolved. Rooms 2, 6, 10, and 14 are such that the room number yields a remainder of 2 when divided by 4. For those rooms we want D to be 8. Therefore, lines 770–800, or

$$D = 4 + 4 * (W - \text{INT}(W/4) * 4 = 2)$$



takes care of all rooms except 0, 4, 8, and 12. For rooms 4 and 12, D should be 1. For room type 8, D should be 2. So lines 770–860, or

$$D = 4 + 4 * (W - \text{INT}(W/4) * 4 = 2) - 23 * ((W = 4) \text{ OR } (W = 12)) - 22 * (W = 8)$$

deals explicitly with every room type except 0. In the above expression, if  $W = 0$ , then D becomes 4. The path behind the maze runner in this case is the north path. So if the current room W is type 0, leave a note about exploring the north path. Then establish D so that  $D = 4$  if  $W = 0$ , or so that in every other case the runner starts out with his back to a wall. Now begin the solution process. 770

We have a maze runner standing in a room W facing a direction D, and we have taken care of any case where there can be an unexplored path behind him. Now it is necessary to look for openings to his left and right and in front of him. How can we tell not to bother with the side behind him (if he is facing north, then there is no need to look south, etc.)? We could say the following: 870

If W is an even number, there is no wall to the north.  
 So if D is not 4 (south), make a note.  
 If W/2 is an even number, there is no wall to the east.  
 So if D is not 8 (west), make a note.  
 If W/4 is an even number, there is no wall to the south.  
 So if D is not 1 (north), make a note.  
 If W/8 is an even number, there is no wall to the west.  
 So if D is not 2 (east), make a note.

The code for the above could be written as four separate tests. Or if a function of some variable J exists, where  $J = 0, 1, 2$ , and then 3, to test for W and also D, the code will be shorter. We know that

If W is an even number, then  $\text{INT}(W/2) * 2 = \text{INT}(W)$   
 If W/2 is an even number, then  $\text{INT}(W/4) * 2 = \text{INT}(W/2)$   
 If W/4 is an even number, then  $\text{INT}(W/8) * 2 = \text{INT}(W/4)$   
 If W/8 is an even number, then  $\text{INT}(W/16) * 2 = \text{INT}(W/8)$

The variables in the four equations above are the divisors 2, 4, 8, and 16 and on the right-hand side, 1 (implied) 2, 4, and 8. So we want  $\text{INT}(W/f1(J)) * 2 = \text{INT}(W/f2(J))$  for our test, where

J	f1(J)	f2(J)
0	2	1
1	4	2
2	8	4
3	16	8

Both functions double each time J increases by 1, which suggests powers of 2. In fact,  $f2(J) = 2 \uparrow J$  and  $f1(J) = 2 \uparrow (J + 1)$ . (Remember that any number raised to the zero power equals 1.) What about the D test? When  $J = 0$ , we want to compare D with 4, and so on, according to the following table:

J	K
0	4
1	8
2	1
3	2

where  $K$  is the number to compare with  $D$ . By inspecting the table we can see that when  $J = 2$  or  $3$ ,  $K = J - 1$ , and when  $J = 0$  or  $1$ ,  $K = J*4 + 4$ , which can be written

$$K = (J - 1)*(J > 1) + (J*4 + 4)*(J < 2)$$

Another way to look at this is to let  $K = J - 1$ , which is right for two cases, and apply a correction factor for the other two:

J	J-1	Error
0	-1	5
1	0	8
2	1	0
3	2	0

If  $J < 2$ , the error as a function of  $J$  is  $3*J + 5$ . So

$$K = J - 1 + (3*J + 5)*(J < 2)$$

Therefore, beginning with  $J = 0$ , and continuing as long as  $J < 4$ , we compute  $K$  for comparison with  $D$ . Then we can say that if there is an opening in this direction and it is not behind the maze runner, make a note. Notice that the direction saved with the note is  $f2(J)$ , that is,  $2 \uparrow J$ , as above, times 10000 to put the direction 1, 2, 4, or 8 in the fifth digit to the left of the decimal point in the note.  $J$  is increased by 1 and the sequence 890-990 is repeated until  $J = 4$ . Now, the room in the maze is set to -1 to show that it has been visited. At this point, the runner is standing in the maze ready to consult his notes. If there are none ( $I = 0$ ), all paths have been tried and there is no solution. If there is at least one note, the runner has to separate  $D$ ,  $X$ , and  $Y$  from the five-digit note and cross off the note. Then he has to move into the next room (Subroutine 1180), check to see if he has solved the maze, and if not, to make more notes if a decision is required in this new room. He may move through rooms with exactly two walls without making notes. When he gets to a dead end or encounters a room he has previously entered, then he must consult his notes again. This process is repeated until the solution is found or he has no more notes, in which case there is no solution.

When Subroutine 1180 is called, the maze runner must move into another room based on  $D$ , the direction he currently faces. How he moves depends of course on the type of room  $W$  as well as on  $D$ .

First, the position of the runner in the maze is displayed (Subroutine 1880).

Then  $W$  is set to the room type  $M(X,Y)$ , which is then set to -1 to show it has been entered. Now it is necessary to build another table to see how to move. It is clear that if  $W$  is a dead end ( $W = 7, 11, 13$ , or  $14$ ), there is only one way out of the room. If  $W$  has at most one wall,  $D$  will be adjusted as a consequence of making notes and then retrieving one. If  $W = 5$  or  $10$  the room is a corridor— $D$  will not change. Only the two-walled rooms  $W = 3, 6, 9$ , and  $12$  require modification of  $D$ .



If W	and D	Change X to	Y to	D to
3	2		$Y - 1$	4
3	1	$X - 1$		8
6	4	$X - 1$		8
6	2		$Y + 1$	1
9	1	$X + 1$		2
9	8		$Y - 1$	4
12	4	$X + 1$		2
12	8		$Y + 1$	1

So if W has at most one wall or is a corridor, increase X by 1 if D = 2, decrease X by 1 if D = 8, increase Y by 1 if D = 1, and decrease Y by 1 if D = 4. 1220

If W = 3, 6, 9, or 12, adjust X, Y, and D according to the above table. 1360

If W is a dead end, turn around. 1630

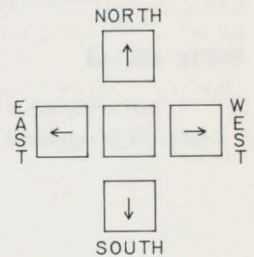
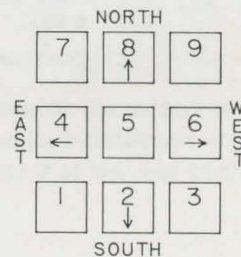
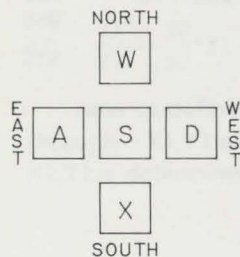
Test for a solution—if  $X = X9$  and  $Y = Y9$ , the runner is at the goal. Display the word SOLVED and end the program. Otherwise, set W to the type of the new room  $M(X,Y)$  and return to the main program.

1790  
1850

### How to Move the Maze Runner

It is much easier to move around a maze when the player is in control than for the computer to determine if a maze has a solution. A simple technique that does not require a joystick or similar controller may be developed from the assumption that four keys on the keyboard correspond to compass directions. It makes sense to select keys that are arranged in those compass directions relative to each other. One such set is the keys W, D, X, and A arranged around the S. They can be designated to stand for north, east, south, and west, respectively, as shown here.

Of course, the N, E, W, and S keys could be used, but they are not conveniently arranged. Obviously, if your keyboard has keys with arrows on them you will want to use them instead. Another possibility in the case of a keyboard with a numeric keypad is to use the numbers 8, 6, 2, and 4 for north, east, south, and west, respectively, as illustrated here.

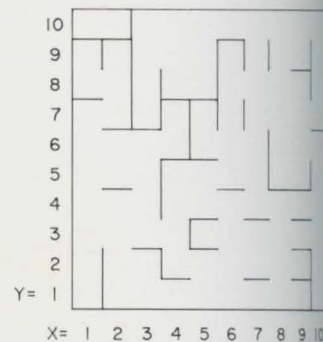




If possible, it is preferable not to require a "Return" after using each arrow key. It is essential to ignore any other key pressed. Also, the program must verify that the user is not trying to move through a wall. The following program moves the player through a 10 x 10 maze like the one shown here.

### Moving the Maze Runner Program Code

	Sequence Number
X9 = 10	500
Y9 = 10	510
X = 1	520
Y = 1	530
GOSUB 940	540
REPEAT	550
W = M(X,Y)	560
GOSUB 890	570
IF A\$="W" THEN ! Set J = 1 if key = north	580
J = 1	590
;	600
IF A\$="D" THEN ! Set J = 2 if key = east	610
J = 2	620
;	630
IF A\$="X" THEN ! Set J=4 if key = south	640
J=4	650
;	660
IF A\$="A" THEN ! Set J=8 if key = west	670
J = 8	680
;	690
IF INT(W/(J*2))*2 = INT(W/J) THEN ! If there is no wall in	700
IF A\$ = "W" THEN ! the indicated direction,	710
Y = Y + 1 ! move to next room	720
;	730
IF A\$ = "X" THEN	740
Y = Y - 1	750
;	760
IF A\$ = "D" THEN	770
X = X + 1	780
;	790
IF A\$="A" THEN	800
X=X-1	810
;	820
GOSUB 940	830
ELSE	840
PRINT "GGG"	850
;	860
UNTIL A\$<>A\$	870



A 10 x 10 maze

**Moving the Maze Runner Code Description**Sequence  
Number

The goal and starting position are set, and the maze runner is displayed (Subroutine 940).	500
The type of the current room is put in W, and the program waits for an arrow key to be pressed (Subroutine 890). The value of J is set to 1 if the player wants to move north, 2 if east, 4 if south, and 8 if west, based on A\$, which contains the value of the key.	550
If there is no wall in the J direction, X or Y is increased or decreased by 1 as appropriate, and the maze runner is displayed in the new room. If a wall exists in the direction of J, a warning bell is rung and the maze runner does not move. Steps 550 – 870 are repeated.	700
To continue only until the maze runner reaches the goal, line 870 should read UNTIL X = X9 AND Y = Y9.	850
	870

**The Maze Runner's View of the Maze**

We have touched only briefly on how the maze looks from the inside, concentrating instead on the plan view. To develop an interior view, it is necessary to consider the mechanics of perspective.

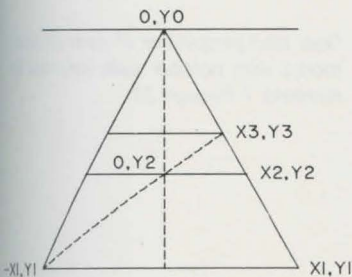
As everybody knows, objects appear to reduce in size as a function of distance. In fact, if it were possible to see infinitely far, an object at that distance would disappear. The point on the horizon at which the object disappears is called the "vanishing point." Between the viewer and the vanishing point, objects appear to get smaller and smaller the farther away they are. The horizontal line through the vanishing point is called the "horizon." No matter how far away, a point above eye level is above the horizon and a point below eye level is below the horizon.

To construct square rooms in a perspective view, it is necessary to determine how much foreshortening or shrinkage appears to occur as rooms get farther away. Renaissance painters knew of a technique to do this called "construction." Given one room, the next one may be constructed by drawing a line from the lower left corner of the first room through a point in the center of the far edge. The point of intersection of that line with the left-hand edge of the room, extended to the vanishing point, defines the upper right-hand corner of the next room. Use the illustration to understand the following application of this concept.

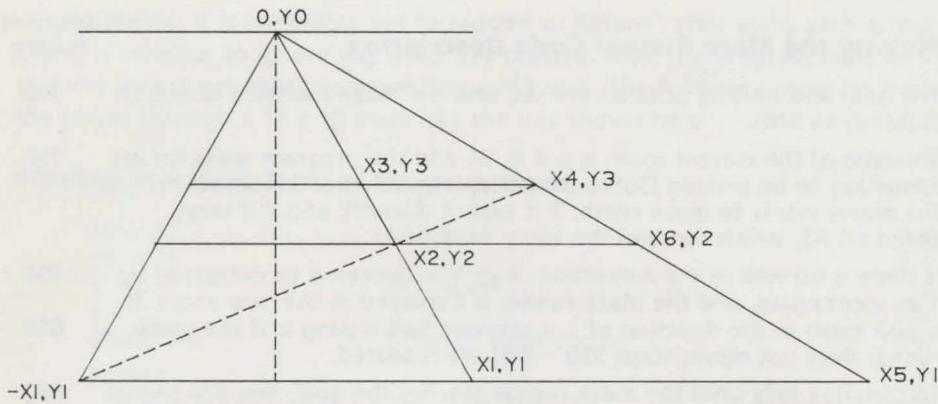
First, it is necessary to know the values of Y0, X1, Y1, and Y2 in this illustration. X2 can be computed, because it is the intersection of the line from the vanishing point through X1,Y1, with the horizontal line through Y2 (see Note 6 at the end of the book):

$$X2 = \frac{Y2 - Y0}{Y1 - Y0} * X1$$

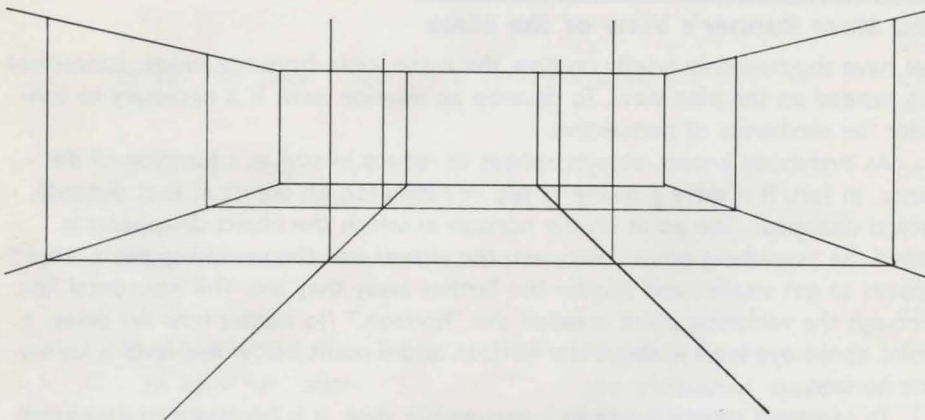
Now to define the far edge of the second room, a line is constructed through -X1,Y1 and 0,Y2 and its intersection with the line through 0,Y0 and X1,Y1 is determined. (See Note 7 at the end of the book.)



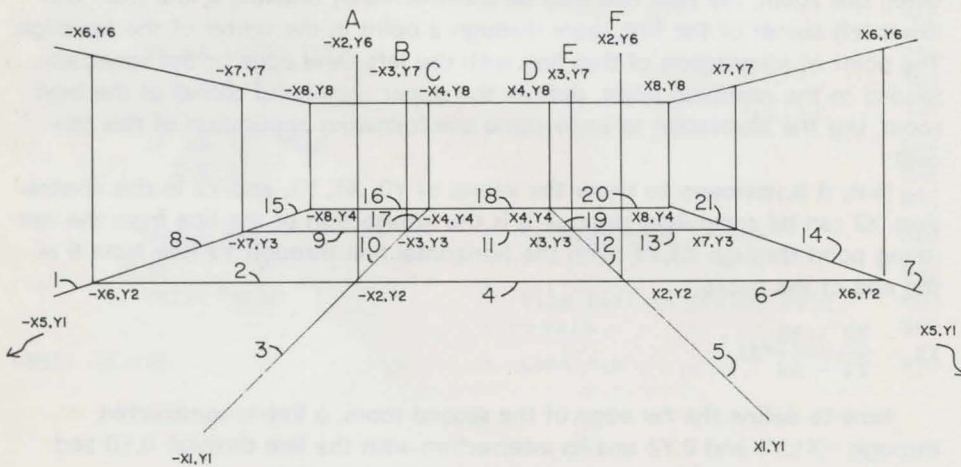
Using "construction" technique to generate perspective



Extending diagonal and horizontal lines to define corners of neighboring rooms



Floor plan of two rooms ahead plus one row of rooms to each side



Floor plan perspective of parts of nine rooms, with possible walls indicated by numbers 1 through 21



$$Y3 = 2*((Y2 - Y1) + (Y0 - Y1)) + Y1$$

$$\text{and } X3 = \frac{Y3 - Y1}{Y2 - Y1} * X1 - X1$$

The same process may be repeated using  $-X2$ ,  $Y2$ , and  $Y3$  to construct the floor of the next room, and so on.

The next program will construct a series of rooms, assuming that the dimensions of the viewing screen are  $-65$  to  $65$  in  $X$  and  $0$  to  $100$  in  $Y$ .

### Constructing a Series of Rooms Program Code

	Sequence Number
$Y1 = 10$	140
$Y2 = 40$	150
$Y0 = 60$	160
$X1 = 50$	170
MOVE $-X1, Y1$	180
DRAW $0, Y0$	190
DRAW $X1, Y1$	200
MOVE $0, Y0$	210
DRAW $0, Y1$	220
MOVE $-65, Y0$	230
DRAW $65, Y0$	240
REPEAT	250
$X2 = (Y2 - Y0)/(Y1 - Y0) * X1$	260
MOVE $-X2, Y2$	270
DRAW $X2, Y2$	280
$Y3 = 2/(1/(Y2 - Y1) + 1/(Y0 - Y1)) + Y1$	290
$X3 = (Y3 - Y1)/(Y2 - Y1) * X1 - X1$	300
MOVE $-X1, Y1$	310
DRAW $X3, Y3$	320
DRAW $-X3, Y3$	330
$X1 = X2$	340
$Y1 = Y2$	350
$X2 = X3$	360
$Y2 = Y3$	370
UNTIL $Y2 > Y0 - 1$	380

### Constructing a Series of Rooms Code Description

	Sequence Number
Set values for $X1$ , $Y0$ , $Y1$ , and $Y2$ .	140
Draw three lines to the vanishing point and draw the horizon.	180
Generate a series of rooms.	250
Compute $X2$ and draw the horizontal line through $Y2$ from $-X2$ to $X2$ .	260
Compute $Y3$ and $X3$ and draw the diagonal line used to make this computation. Draw the horizontal line from $-X3$ to $X3$ through $Y3$ .	290
Reset variables to compute the next room.	340
Continue computing rooms until close to the horizon.	380

Having developed the capability to lay out the floors of a series of rooms directly in front of the viewer, what about the rooms on either side? If diagonal

and horizontal lines are extended, their intersections define corners of neighboring rooms, as seen in the next perspective drawing.

In this drawing, the line through  $-X1, Y1$  and  $X2, Y2$  has been extended and so has the horizontal line through  $Y3$  to determine  $X4$  at their intersection. Then, other points such as  $X5$  and  $X6$  may be computed from the intersections of their respective horizontals with the line through  $0, Y0$  and  $X4, Y3$ .

$$X4 = \frac{Y3 - Y1}{Y2 - Y1} * (X2 + X1) - X1$$

$$\text{so } X5 = \frac{Y1 - Y0}{Y3 - Y0} * X4$$

$$\text{and } X6 = \frac{Y2 - Y0}{Y3 - Y0} * X4$$

The following program shows three rooms side by side, this time without drawing the construction diagonals and the vertical line through the vanishing point.

### **Constructing Three Rooms Side by Side Program Code**

```

Y1 = 10           ! Initialize                                140
Y2 = 40           150
Y0 = 60           160
X1 = 50           170
MOVE -X1,Y1       ! Draw sides of center rooms              180
DRAW 0,Y0          190
DRAW X1,Y1         200
MOVE -65,Y0        ! Draw horizon                            210
DRAW 65,Y0         220
REPEAT             230
  X2=(Y2-Y0)/(Y1-Y0)*X1 ! Compute and draw back of room      240
  MOVE -X2,Y2       250
  DRAW X2,Y2        260
  Y3 = 2/(1/(Y2 - Y1) + 1/(Y0 - Y1)) + Y1 ! Compute and draw 270
  X3 = (Y3 - Y1)/(Y2 - Y1)*X1 - X1           ! side rooms     280
  X4 = (Y3 - Y1)/(Y2 - Y1)*(X2 + X1) - X1     290
  X5 = (Y1 - Y0)/(Y3 - Y0)*X4                 300
  X6 = (Y2 - Y0)/(Y3 - Y0)*X4                 310
  MOVE X3,Y3          320
  DRAW -X3,Y3         330
  MOVE X5,Y1          340
  DRAW X6,Y2          350
  DRAW -X6,Y2         360
  DRAW -X5,Y1         370
  X1 = X2             ! Update                             380
  Y1 = Y2             390
  X2 = X3             400
  Y2 = Y3             410
UNTIL Y2 > Y0 - 1     ! until close to horizon              420

```



**Constructing Three Rooms Side by Side Code Description**Sequence  
Number

After setting initial values for the variables and drawing the side edges of the center rooms, X2 is computed and the far edge of the center room is drawn. All computations are done for the next center room and the side rooms. The second center room and the first side rooms are drawn. Variables are reset to move to the next set of rooms, and the program continues to draw rooms until close to the vanishing point.	140 230 270 320 380 420
---	--

With respect to constructing walls, let us say that in the maze there is only enough light to see two rooms ahead and one room on either side of the room in which the maze runner is standing. The accompanying floor plan is, therefore, of interest; what is shown are the edges of nine rooms, with walls around the outside edges of the farthest visible rooms and vertical lines at the other corners. Edges between walls and ceilings can be constructed in the same manner as the floor plan by extending lines from the vanishing point to an arbitrary point above the horizon.

In this construction, the ceiling is drawn to a point that is three-fourths as high above the horizon as the floor is below it. Since the number of visible rooms has been defined, the X,Y values of each corner as seen by the maze runner are fixed. You may compute them in advance and use them as constants rather than actually performing the construction as in the previous program.

The next problem is to hide parts of the drawing based on the existence of any combination of the twenty-one walls that may exist in the section that the maze runner can see.

In the next drawing, floor edges are numbered to simplify discussion:

X1 = 50	Y1 = 10
X2 = 20	Y2 = 40
X3 = 12.5	Y3 = 47.5
X4 = 9.0909	Y4 = 50.9091
X5 = 150	Y5 = 97.5
X6 = 60	Y6 = 75
X7 = 37.5	Y7 = 69.375
X8 = 27.2727	Y8 = 67.4318

The maze runner is standing in the room bounded by walls 3, 4, and 5. To determine what he can see to his left, it is necessary to imagine what is visible first if the closest edge has a wall, then the next closest, and so on. If his line of sight passes through edges 3, 2, and 8, adding a wall to 3 will prevent him from seeing edges 1, 2, or 8. Only part of edges 9 and 16 will be visible. The same parts of 9 and 16 will be visible if 3 has no wall but there is a wall on 2.

This suggests that what is important in the determination of visibility is the corner of a room. If either of the two walls that meet at a corner is present, the same part of the view is rendered invisible. This means that instead of considering twenty-one edges as criteria for invisibility, we could consider instead the six corners A, B, C, D, E, and F. For this example, however, we will consider the walls.

The most complicated case is typified by edge 16. If there is a wall on edge 9, only the part of wall 16 between uprights B and C is visible. If there is no wall on 9 but a wall on 17, wall 16 is visible from B to the wall's left edge. If 9 and 17 have no walls but there is a wall on 10, then wall 16 can be seen on either side of wall 10 but is obscured between uprights A and B. This suggests that a reason-



able way to process walls is in vertical slices—to the left of A, then between A and B, then between B and C, and so forth.

The following is an algorithm for the computer to follow in drawing the section to the left of A: If there is a wall on 3, draw it; everything else is invisible. If there is no wall on 3, then wall 1 is visible if there is a wall on 1. Then if wall 2 exists, it will obscure everything else. If not, wall 8 is visible. If wall 9 exists, the section to the left of A is drawn, omitting the vertical line on A. If 9 does not exist, then 15 is visible, and so is the section of 16 to the left of A.

Now the section between A and B is drawn. If 10 exists, everything else is invisible. If not, the section of 9 between A and B is visible. If 9 is a wall, then it is drawn including the upright B. If neither 10 nor 9 are walls, then 16 is visible between A and B.

Between B and C, if 17 is present, it is drawn; if not, the section of 16 is visible. Between C and D, if 18 is present, it is drawn. The other sections are dealt with in a similar manner.

Exploring a maze in this view rather than from above, the maze runner is unable to determine whether he is moving north, south, east, or west. Consequently, it is necessary to define the arrow keys differently. Instead of using them as compass directions, they should mean the following:

Press W to move forward  
D to turn right  
X to turn around  
A to turn left

Furthermore, it is very disorienting to turn and simultaneously move to another room. While W should allow the maze runner to move to another room, A, D, or X should only change the runner's viewing direction, leaving him in the same room as before. Depending on the direction the maze runner is facing, one of four views is seen. In each case, there are 21 possible walls, but they are in different rooms relative to the maze runner. In order to process them regardless of direction, a wall table may be defined, as shown here, in which the walls and rooms pertaining to each direction are shown.

The following program will examine the 21 pertinent walls, putting a one in V(N) if there is a wall on edge N, and a zero if not. Notice that in some cases, fewer than nine rooms are visible to the viewer. For example, if he is standing in M(1,1) facing north, there are no rooms to his left. The programmer must be careful not to refer to rooms M(X - 1,Y) because they are not defined.

LOOKING NORTH					LOOKING EAST					LOOKING SOUTH					LOOKING WEST				
I=2	16	18	20		J=0	1	8	15		I=0	7	5	↓	3	J=2	21	14	7	
	15	17	19	21		3	10	17			6	4	2	1		20	13	6	
I=1	9	11	13		J=1	→	11	18		I=1	4	12	10	9	J=1	18	11	4	←
	8	10	12	14		5	12	19			13	11		8		17	10		
I=0	2	4	6		J=2	7	13	20		I=2	21	19	17	15	J=0	6	9	2	3
	1	3	5	7			14	21			20	18	16			15	8	1	
J=0	J=1	J=2			I=0	I=1	I=2			J=2	J=1	J=0			I=2	I=1	I=0		

Wall table for maze runner

**Load Wall Table Program Code**

	Sequence Number
DIM V(21) ! Wall table	500
DEF FNN(W) = W-INT(W/2)*2 ! Define wall functions	510
DEF FNE(W) = INT(W/2)-INT(W/4)*2	520
DEF FNS(W) = INT(W/4)-INT(W/8)*2	530
DEF FNW(W) = INT(W/8)	540
X9 = 10 ! Initialize goal	550
Y9 = 10	560
X = 1 ! and starting position	570
Y = 1	580
GOSUB 1770 ! Display maze runner	590
D=2↑ INT(RND(-1)*4) ! Select a random direction D=1,2,4, or 8	600
REPEAT	610
V=1	620
I = 0	630
IF D = 1 THEN ! Get walls based on direction	640
REPEAT ! Facing north,	650
J = 0	660
IF X = 1 THEN	670
J = 1	680
;	690
REPEAT ! get west, north, and east walls	700
W = M(X - 1 + J, Y + I)	710
V(I*7 + J*2 + 1) = FNW(W)	720
V(I*7 + J*2 + 2) = FNN(W)	730
V(I*7 + J*2 + 3) = FNE(W)	740
J = J + 1	750
UNTIL J > 2 OR X + J >= X0	760
I = I + 1	770
UNTIL I > 2 OR Y + I > Y0	780
;	790
IF D = 2 THEN ! Facing east,	800
REPEAT	810
J = 0	820
IF Y = Y0 THEN	830
J = 1	840
;	850
REPEAT ! get north, east, and south walls	860
W = M(X + 1, Y + 1 - J)	870
V(I*7 + J*2 + 1) = FNN(W)	880
V(I*7 + J*2 + 2) = FNE(W)	890
V(I*7 + J*2 + 3) = FNS(W)	900
J = J + 1	910
UNTIL J > 2 OR Y - J < 0	920
I = I + 1	930
UNTIL I > 2 OR X + I > X0	940
;	950
IF D = 4 THEN ! Facing south,	960
REPEAT	970

```

J = 0                                     980
IF X = X0 THEN                           990
    J = 1                                1000
;                                         1010
REPEAT                                  ! get east, south, and west walls 1020
    W = M(X + 1 - J, Y - I)             1030
    V(I*7 + J*2 + 1) = FNE(W)           1040
    V(I*7 + J*2 + 2) = FNS(W)           1050
    V(I*7 + J*2 + 3) = FNW(W)           1060
    J = J + 1                           1070
UNTIL J > 2 OR X - J < 0                 1080
    I = I + 1                           1090
UNTIL I > 2 OR Y = I                     1100
;                                         1110
IF D = 8 THEN                            ! Facing west,                 1120
    REPEAT                               1130
        J = 0                           1140
        IF Y = 1 THEN                   1150
            J = 1                       1160
        ;                               1170
        REPEAT                          ! get south, west, and north walls 1180
            W = M(X - I, Y - 1 + J)     1190
            V(I*7 + J*2 + 1) = FNS(W)   1200
            V(I*7 + J*2 + 2) = FNW(W)   1210
            V(I*7 + J*2 + 3) = FNN(W)   1220
            J = J + 1                   1230
            UNTIL J > 2 OR Y + J > Y0    1240
            I = I + 1                   1250
            UNTIL I > 2 OR X = I         1260
        ;                               1270
        W = M(X, Y)                     1280
        GOSUB 1720                       ! Wait for a valid key press 1290
        IF A$ = "W" THEN                 ! Establish new direction 1300
            J = D                         1310
        ;                               1320
        IF A$ = "D" THEN                 1330
            J = 2*D                       1340
        ;                               1345
            IF D = 8 THEN                 1350
                J = J - 15                1360
            ;                             1370
        ;                               1380
        IF A$ = "X" THEN                 1390
            J = INT(D/4)                  1400
            IF D < 4 THEN                 1410
                J = J + 4*D               1420
            ;                             1430
        ;                               1440
        IF A$ = "A" THEN                 1450

```



```

J = INT(D/2) 1460
IF D = 1 THEN 1470
    J = J + 8 1480
; 1490
; 1500
D = J 1510
IF INT(W/D*2))*2 = INT(W/D) THEN 1520
    IF D = 1 THEN ! If there is no wall in the way, 1530
        Y = Y + 1 ! move in new direction 1540
    ; 1550
    IF D = 4 THEN 1560
        Y = Y - 1 1570
    ; 1580
    IF D = 2 THEN 1590
        X = X + 1 1600
    ; 1610
    IF D = 8 THEN 1620
        X = X - 1 1630
    ; 1640
    GOSUB 1770 ! Display runner 1650
ELSE 1660
    PRINT "GGG"; ! Ring bell if a wall is in the way 1670
; 1680
UNTIL X = X9 AND Y = Y9 ! Continue to goal 1690
END 1700

```

In this program, the wall table is used to move around the maze. Since perspective views of the maze are not developed, it is assumed that the arrow keys still mean "Move north," "Move east," and so on.

### **Load Wall Table Code Description**

Sequence  
Number

In order to reduce the amount of code, four functions are defined. 510  
When FNN(W) is invoked, its value is 1 if there is a wall on the north  
edge of room W and a zero if not. Similarly, FNE for an east wall, FNS  
for south, and FNW for a west.

After establishing the starting room M(X,Y) and the goal M(X9,Y9), Sub- 550  
routine 1770 is called to display the position of the runner in the maze. 590  
A random direction D is generated, and the wall table V is set to all 1's. 600

The index I, as seen in the four figures above, is set to zero. Then, de- 630  
pending on the direction D, the walls are brought to the wall table. 640

If the direction is north, the index J is set to zero unless X = 1, in which 650  
case J is set to 1. For each room, three edges are examined and the ap- 700  
propriate values are stored in V. After J exceeds 2 or the right-hand 760  
edge of the maze has been reached, I is increased until I exceeds 2 or 770  
the top of the maze has been reached. 780

A similar process is required for each of the other three directions, al- 800  
though only one of them is executed depending on the value of D.

The program waits for a key press (Subroutine 1720) after giving a beep signal that it is ready. The new direction is established based on the previous direction and the last key pressed. The following table shows the new direction required based on the existing value of D and A\$, the variable in which the key value is placed:

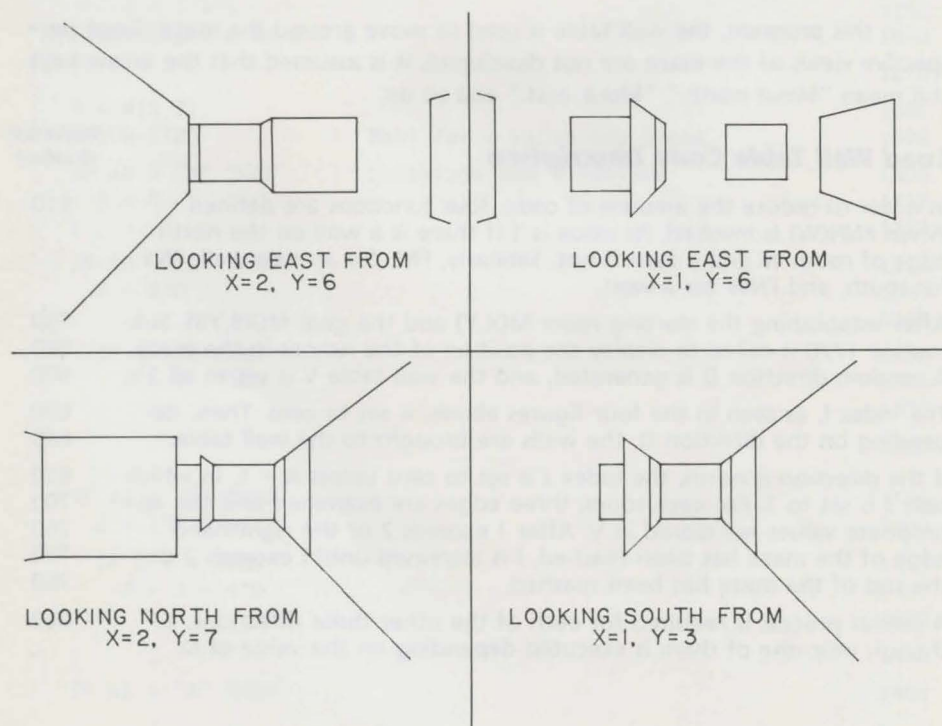
	W	D	X	A
1	1	2	4	8
2	2	4	8	1
4	4	8	1	2
8	8	1	2	4

The values of the room indices X and Y are modified based on the direction. If there is a wall in front of the maze runner, he cannot move forward—a sustained beep is sounded and the room is redisplayed. This process continues until the goal is attained.

### Perspective Maze

Now the view that a maze runner would see can be displayed. The arrow keys take on the meaning

- W Move forward if possible
- D Turn right without changing rooms
- X Turn around without changing rooms
- A Turn left without changing rooms



Looking east from X = 2, Y = 6

Looking east from X = 1, Y = 6

Looking north from X = 2, Y = 7

Looking south from X = 1, Y = 3

**Perspective Maze Program Code**

	Sequence Number
DIM V(21)	370
DEF FNN(W) = W - INT(W/2)*2	380
DEF FNE(W) = INT(W/2) - INT(W/4)*2	390
DEF FNS(W) = INT(W/4) - INT(W/8)*2	400
DEF FNW(W) = INT(W/8)	410
DATA 1,1,10,10	420
DATA 50,20,12.5,9.090909,150,60,37.5,27.272727	430
DATA 10,40,47.5,50.90909,97.5,75,69.375,67.431818	440
READ X,Y,X9,Y9,X1,X2,X3,X4,X5,X6,X7,X8,Y1,Y2,Y3,Y4, Y5,Y6,Y7,Y8	450
D = 2*INT(RND(-1)*4) ! Establish starting direction	460
REPEAT	470
V = 1	480
I = 0	490
REPEAT	500
J = 0	510
IF X = 1 AND D = 1 THEN	520
J = 1	530
;	540
IF X = X0 AND D = 4 THEN	550
J = 1	560
;	570
IF Y = 1 AND D = 8 THEN	580
J = 1	590
;	600
IF Y = Y0 AND D = 2 THEN	610
J = 1	620
;	630
REPEAT	640
K = X	650
L = Y	660
IF D = 1 THEN	670
K = K + J - 1	680
L = L + I	690
;	700
IF D = 4 THEN	710
K = K - (J - 1)	720
L = L - I	730
;	740
IF D = 2 THEN	750
K = K + I	760
L = L - (J - 1)	770
;	780
IF D = 8 THEN	790
K = K - I	800
L = L + J - 1	810
;	820
W = M(K,L)	830
N = I*7 + J*2	840
IF D = 1 THEN	850
V(N + 1) = FNW(W)	860



```

V(N + 2) = FNN(W) 870
V(N + 3) = FNE(W) 880
; 890
IF D = 2 THEN 900
    V(N + 1) = FNN(W) 910
    V(N + 2) = FNE(W) 920
    V(N + 3) = NS(W) 930
; 940
IF D = 4 THEN 950
    V(N + 1) = FNE(W) 960
    V(N + 2) = FNS(W) 970
    V(N + 3) = FNW(W) 980
; 990
IF D = 8 THEN 1000
    V(N + 1) = FNS(W) 1010
    V(N + 2) = FNW(W) 1020
    V(N + 3) = FNN(W) 1030
; 1040
J = J + 1 1050
N = 0 1060
IF (X + J = > XO AND D = 1) OR (Y + J = > YO AND
D = 8) THEN 1062
    N = 1 1064
; 1066
UNTIL J>2 OR N OR (Y<J AND D=2) OR (X<J AND D=4) 1070
I = I + 1 1080
N = 0 1082
IF (X+I=>XO AND D=2) OR (Y+I=>YO AND D=1) THEN 1084
    N = 1 1086
; 1088
UNTIL I>2 OR N OR (X=I AND D=8) OR (Y=I AND D=4) 1100
PAGE ! Clear screen 1110
IF V(3) THEN ! Draw from left to corner "A" 1120
    GOSUB 2760 1130
ELSE 1140
    IF V(1) THEN 1150
        GOSUB 2630 1160
    ; 1170
    IF V(2) THEN 1180
        GOSUB 2690 1190
    ELSE 1200
        IF V(8) THEN 1210
            GOSUB 2820 1220
        ; 1230
        IF V(9) THEN 1240
            GOSUB 2890 1250
        ELSE 1260

```

IF V(15) THEN	1270
GOSUB 2950	1280
;	1290
IF V(16) THEN	1300
GOSUB 3020	1310
;	1320
;	1330
;	1340
;	1350
IF V(4) THEN ! Draw "A" to "B"	1360
GOSUB 3110	1370
ELSE	1380
IF V(10) THEN	1390
GOSUB 3240	1400
ELSE	1410
IF V(9) THEN	1420
GOSUB 3180	1430
ELSE	1440
IF V(16) THEN	1450
GOSUB 3310	1460
;	1470
;	1480
;	1490
IF V(11) THEN ! Draw "B" to "C"	1500
GOSUB 3400	1510
ELSE	1520
IF V(17) THEN	1530
GOSUB 3530	1540
ELSE	1550
IF V(16) THEN	1560
GOSUB 3470	1570
;	1580
;	1590
IF V(18) THEN ! Draw "C" to "D"	1600
GOSUB 3600	1610
;	1620
IF V(19) THEN ! Draw "D" to "E"	1630
GOSUB 3700	1640
ELSE	1650
IF V(20) THEN	1660
GOSUB 3770	1670
;	1680
;	1690
;	1700
IF V(12) THEN ! Draw "E" to "F"	1710
GOSUB 3860	1720

ELSE	1730
IF V(13) THEN	1740
GOSUB 3930	1750
ELSE	1760
IF V(20) THEN	1770
GOSUB 3990	1780
;	1790
;	1800
;	1810
;	1820
IF V(5) THEN ! Draw "F" to corner	1830
GOSUB 4070	1840
ELSE	1850
IF V(7) THEN	1860
GOSUB 4200	1870
;	1880
IF V(6) THEN	1890
GOSUB 4130	1900
ELSE	1910
IF V(14) THEN	1920
GOSUB 4320	1930
;	1940
IF V(13) THEN	1950
GOSUB 4260	1960
ELSE	1970
IF V(21) THEN	1980
GOSUB 4450	1990
;	2000
IF V(20) THEN	2010
GOSUB 4390	2020
;	2030
;	2040
;	2050
;	2060
W = M(X,Y)	2070
GOSUB 2530 ! Wait for an arrow key	2080
J = 0	2090
IF A\$ = "W" THEN ! Compute new direction	2100
J = J + D	2110
;	2120
IF A\$ = "D" THEN	2130
J = J + 2*D	2140
IF D = 8 THEN	2150
J = J - 15	2160
;	2170
;	2180



IF A\$ = "X" THEN	2190
J = J + INT(D/4)	2200
IF D < 4 THEN	2210
J = J + 4*D	2220
;	2230
;	2240
IF A\$ = "A" THEN	2250
J = J + INT(D/2)	2260
IF D = 1 THEN	2270
J = J + 8	2280
;	2290
;	2300
D = J	2310
IF A\$ = "W" THEN	2320
IF INT(W/(D*2))*2 = INT(W/D) THEN ! If no wall, move	2330
IF D = 1 THEN	2340
Y = Y + 1	2350
;	2360
IF D = 4 THEN	2370
Y = Y - 1	2380
;	2390
IF D = 2 THEN	2400
X = X + 1	2410
;	2420
IF D = 8 THEN	2430
X = X - 1	2440
;	2450
ELSE ! If wall, ring bell	2460
PRINT "GGG";	2470
;	2480
;	2490
UNTIL X = X9 AND Y = Y9 ! Continue to goal	2500
END	2510

### Perspective Maze Code Description

	Sequence Number
After defining the maze as usual, but not drawing a top view, the wall table is defined.	370
The direction functions are defined, and the starting position of the runner, the position of the goal and values of room corners computed in the perspective floor layout are read into their symbolic names.	380 420 450
A random starting direction is chosen and the solution begins.	460
Note that loading the wall table, lines 500 through 1100, has been consolidated somewhat from the previous program (lines 640-1270).	500
The screen is cleared and the visible section between the left-hand edge and corner A is displayed.	1100
Then corner A to corner B.	1360
Then corner B to corner C.	1500
Then corner C to corner D.	1600

Then corner D to corner E. 1630

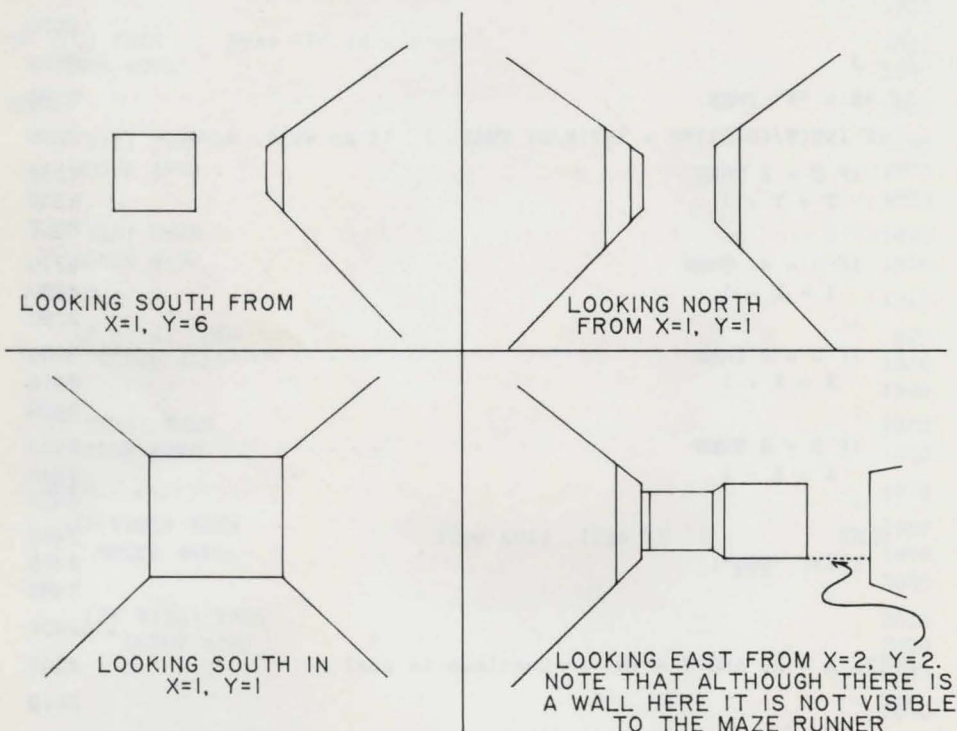
Then corner E to corner F. 1710

Then corner F to the right-hand edge. 1830

The computer waits for an arrow key (Subroutine 2530), then computes a new direction. If the "Move Ahead" key was pressed, the room indices X and Y are updated, unless moving ahead causes the runner to crash into a wall. 2080

The program continues until the goal is reached. 2500

Shown below are typical perspective views found in the 10 x 10 maze.

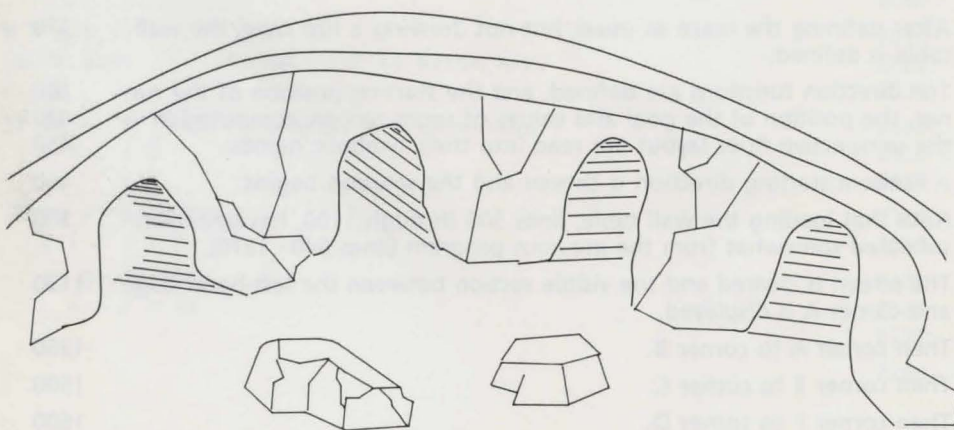


Looking south from X = 1, Y = 6

Looking north from X = 1, Y = 1

Looking south in X = 1, Y = 1

Looking east from X = 2, Y = 2. Note that although there is a wall here, it is not visible to the maze



Maze shown as caves instead of rooms

---

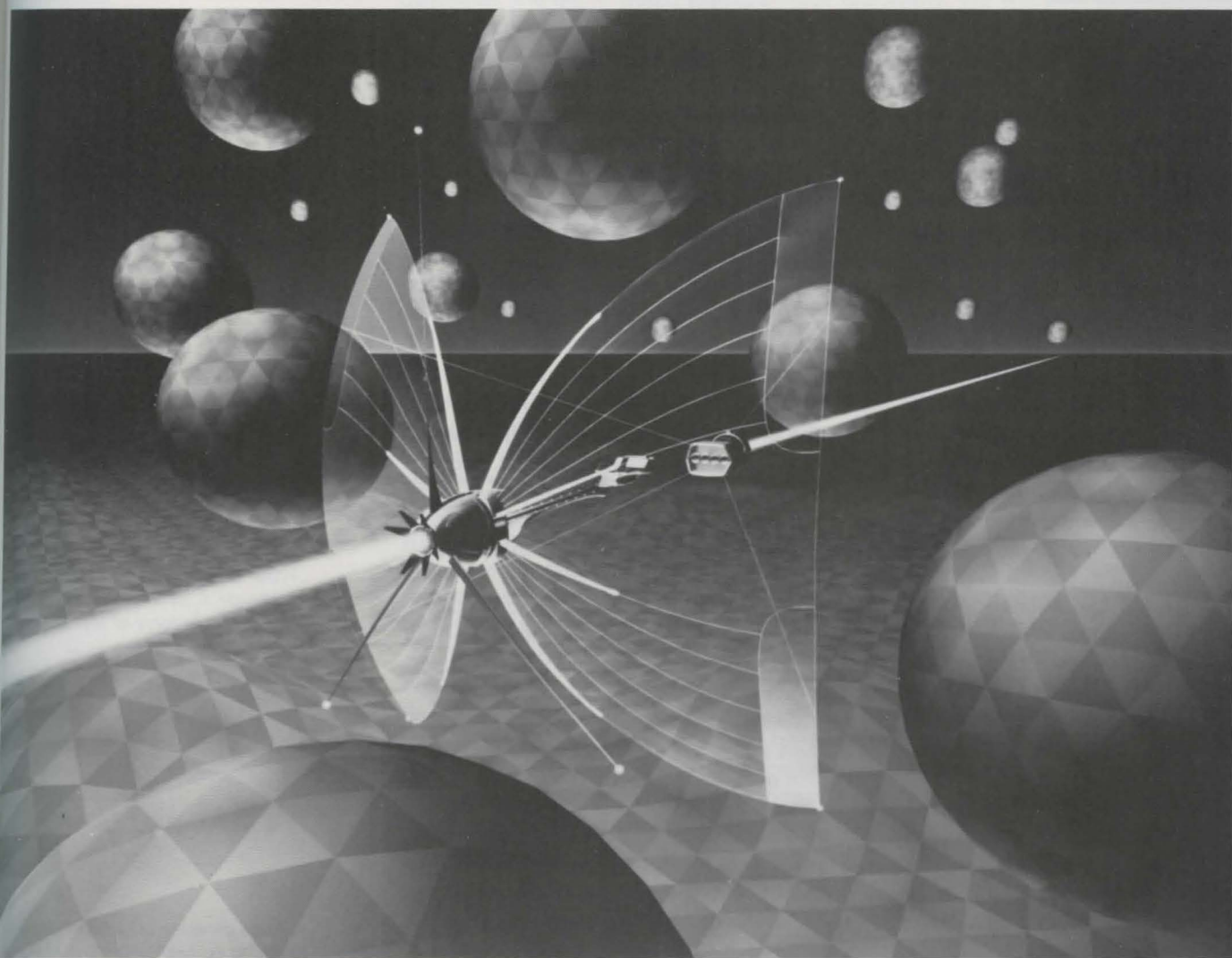
**Refinements**

- Color or shade walls to represent the fact that the only light in the maze is being carried by the maze runner.
- Extend the perspective view to show all visible walls, no matter how far away.
- Display the maze as a labyrinthine cave, as suggested in the accompanying drawing. Randomly show objects in some of the caves occasionally, such as bones, broken weapons, discarded implements, spider webs, bats hanging from the roof of the cave, and so on.
- Make a multilevel maze with ladders, ropes, or staircases leading from one level to another. Make sure that a ladder going down from one level goes up in the same place from the level below. Use the S key to take a ladder to a different level.
- In a multilevel maze, put a concealed hole in the floor of a different room on each level, so that when a runner enters that room he falls to an unknown place on an unspecified level. The destination room should be the same every time, but it will be the maze runner's task to establish where he is. When he falls, let him know that he has fallen.
- Give the maze runner a supply of food and water and provide water holes in fixed locations in the maze so he can replenish his supply. Randomly provide edible delicacies such as lizards, grasshoppers, and such, to replenish his food supply. If he runs out of food or water, he "dies" in the maze and has to begin again.
- Give him a certain amount of strength that he can use to fight creatures like beligerent lizards. As he fights them, his strength diminishes but his experience improves. Let him exchange experience for renewed strength, either by returning to the start of the maze or by entering special rooms.
- After he successfully fights monsters, provide for the possibility that the monster may have been guarding a treasure. Let the maze runner use the treasure to buy weapons that make it easier to fight monsters. Make the monsters harder to fight and the treasures more worthwhile the deeper the runner goes in the maze.





## Outer Space Games



Outer space has replaced Cowboys and Indians as the source for larger-than-life heroes whose superhuman feats conquer the frontier by exploration and combat with hostile natives. It has become unacceptable in recent years to buy toy guns for children but laser blasters are fine, presumably because malevolent aliens are acceptable as cannon fodder whereas malevolent humans are not. Certainly, several billion Space Invaders have been slaughtered since the invention of the arcade game and so far nobody seems to feel very guilty about it.

To achieve realistic outer space games, the programmer must have an understanding of Newton's Law of Universal Gravitation and his Laws of Motion. A brief discussion of these laws is followed by their application in a simple Space Race game. Obviously, vectors, as presented in the *Ballistic Trajectory* chapter, reappear here. The chapter ends with improvements to the game such as collisions and controlled flight.

### Newton's Laws

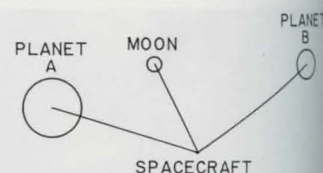
It is often said that there is no gravity in space. In fact, the reason for the feeling of weightlessness is that a body under no other acceleration forces is said to be in a state of "free fall"—falling under the influence of the Law of Universal Gravitation. This law, which states that all bodies attract each other according to their masses and to their distance apart, was recognized by Isaac Newton. It is popularly supposed that he came to an understanding of the Law of Universal Gravitation by watching an apple fall from a tree. Actually, he was studying the motion of the moon around the earth when he hit upon the law:

$$F = G \frac{m m'}{r^2}$$

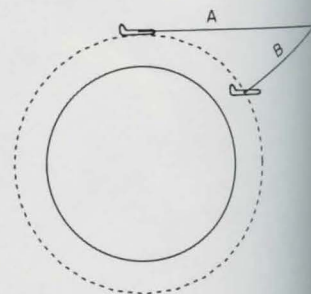
which means that the force  $F$  of attraction of two bodies is proportional to their masses  $m$  and  $m'$  divided by the square of the distance  $r$  between them. The constant of proportionality in this case is  $G$ , the gravitational constant. Thus, a spacecraft floating between two planets, one of which has a moon, is subject to gravitational pull along three directions, as illustrated here by a line from the ship to planet A, a line to its moon, and another to planet B. If all other bodies are extremely far away or extremely small, their influence will be so slight that it can be ignored.

Newton also defined his Laws of Motion, among which is the famous statement that a body in motion tends to remain in motion. That is, in the absence of any force, something moving in a given direction at a given speed will continue to do so. It is because of this law and the Law of Universal Gravitation that we can put satellites in orbit around the earth. As a satellite travels in a straight line, the gravitational attraction of the earth pulls it "down." If the linear speed of the satellite is just right for the altitude, the distance it gains from the earth will match the distance it falls due to gravity, and a circular orbit will be achieved, which is demonstrated in the first of three drawings dealing with orbits.

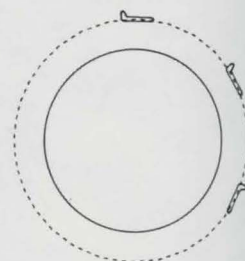
In fact, it is possible to put a satellite in a geosynchronous orbit, in which the satellite remains above a fixed point on the earth if the craft completes one orbit in 24 hours travelling from east to west. Since the earth revolves about its axis once every 24 hours, the satellite's rate of travel is sufficient to keep it in place.



Gravitational pull exerted on a spacecraft by three bodies due to law of universal gravitation

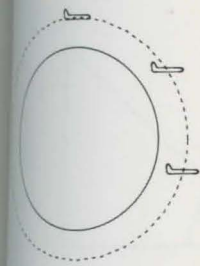


Circular orbit achieved by correct speed for altitude A combined with distance fallen owing to force of gravity B



Attitude generally but erroneously assumed to be held by orbiting spacecraft





altitude actually held by orbiting space-

craft

Notice, though, that because we are accustomed to thinking in terms of a flat earth, most people assume that an orbiting spacecraft would maintain a given aspect with respect to the earth's surface, as shown in the second orbit drawing. However, in accordance with Newton's law, there is no force acting to push the nose down, so it moves as in the third orbit drawing and would appear to a sequence of observers on the earth to have performed one backward somersault per orbit, even if the orbit is geosynchronous. Orbital purists can "correct" for this occurrence by applying a rotational velocity to the spacecraft sufficient to cause one *forward* somersault per orbit.

### Space Race Game

It is possible to apply the gravitation and motion formulas to a spacecraft game, but considerable modification of the numerical values involved is necessary; otherwise, the game would proceed much too slowly. We will also want to display objects on the screen that would be invisible if they were drawn in proportion to their true distance apart.

Consider a simple game called Space Race. On the screen will be displayed two planets, one of which has a moon orbiting it. Let us call the planet with the moon, earth, and the other planet Mars. (Mars actually has two moons.) A spacecraft is in space near earth. The object of the game is to pass alternately through the orbits of Mars and earth as often as possible. Each time an orbit is crossed, a point is scored. To score the next point, the ship must cross the orbit of the other planet.

The captain of the spacecraft has a certain amount of fuel that he can use to control the ship, but his object is clearly to use gravity as much as possible in order to conserve fuel. After he runs out of fuel, he may have achieved an orbit stable enough to gain several more points, so the game must continue until the spacecraft either crashes into one of the planets or the moon or until it is so far away that it is deemed lost in space.

### Applying Gravity

Newton's law says that two bodies are attracted to one another, but when one body is vastly bigger than the other, as in the case of earth and a spacecraft, it is reasonable to assume that the larger body does not move at all. So computations of motion with respect to gravity need only be done for the ship.

There are three forces acting on the craft: the gravitational pulls of the two planets and the moon. The forces are pulling along the lines between the center of gravity of the ship and of each of the three bodies. Depending on their relative directions, the forces may tend to reinforce or to cancel each other.

Since this game is being developed only in two dimensions, the resultant force or net force acting on the ship may be thought of as having an X and a Y component. If the X and Y component of each of the three forces is computed, the resultant force's components are

$$\begin{aligned} X(\text{resultant}) &= X(\text{earth}) + X(\text{Mars}) + X(\text{moon}) \\ \text{and } Y(\text{resultant}) &= Y(\text{earth}) + Y(\text{Mars}) + Y(\text{moon}) \end{aligned}$$

To find the components of each force it will be necessary to know the angle of each force vector. We know two points on the line of the vector, namely the center of the spacecraft  $X_0, Y_0$  and the center of the body  $X_n, Y_n$ . So the angle of the vector is  $\theta$ , shown in the illustration. (Also see Note 8 at the end of the book.)

$$\theta = (\text{ACS}(\frac{X_n - X_0}{r}))$$

IF  $Y_n < Y_0$  THEN

$$\theta = 360 - \theta$$

;

where  $r = \text{SQR}((X_n - X_0)^2 + (Y_n - Y_0)^2)$

Clearly,  $r$  can only be zero when the spacecraft is at the center of gravity of one of the bodies, which means that it has crashed, so this case should not occur. However, to avoid any unforeseen error, it is safe to say that if  $r = 0$ , then  $\theta = 0$ . To compute the angle of the force vector, then, the code is:

$$\theta = \text{SQR}((X_n - X_0)^2 + (Y_n - Y_0)^2)$$

IF  $\theta < > 0$  THEN

$$\theta = \text{ACS}((X_n - X_0) / \theta)$$

IF  $Y_n < Y_0$  THEN

$$\theta = 360 - \theta$$

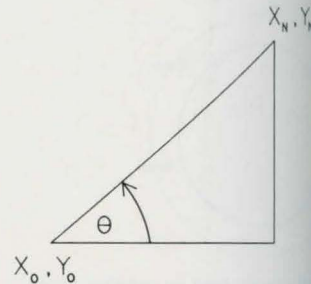
;

;

The following algorithm will allow a spacecraft to fall freely under the influence of three bodies until it is lost in space. The viewing surface ranges from -65 to 65 in X and from 0 to 100 in Y. Angles are expressed in degrees.

### Free Fall Program Code

	Sequence Number
DIM D0(2),D1(2),D2(2),D3(2),D(2),V(2)	170
DATA 1,-0.1,0,.000001,1,1000000,10000000,5000000	180
DATA -50,40,-50,50,50,80	190
DATA 1,4,2.5	200
READ V,A,G,M0,M1,M2,M3,D0,D2,D3,R1,R2,R3 ! Initialize	210
D = D2	220
R = R2	230
GOSUB 620 ! Draw earth	240
D = D3	250
R = R3	260
GOSUB 620 ! Draw Mars	270
REPEAT	280
GOSUB 720 ! Erase moon	290
D1(1) = D2(1) + 15*COS(A) ! Compute new position of moon	300
D1(2) = D2(2) + 15*SIN(A)	310
D = D1	320
R = R1	330
GOSUB 620 ! Redraw moon	340
GOSUB 680 ! Erase and redraw spacecraft	350
F1=G*(M0*M1)/SQR((D1(1)-D0(1))↑2+(D1(2)-D0(2))↑2)↑2!Compute	360
F2=G*(M0*M2)/SQR((D2(1)-D0(1))↑2+(D2(2)-D0(2))↑2)↑2 ! forces	370
F3=G*(M0*M3)/SQR((D3(1)-D0(1))↑2+(D3(2)-D0(2))↑2)↑2 ! & angles	380
A1=SQR((D1(1)-D0(1))↑2+(D1(2)-D0(2))↑2) ! of force vectors	390



Determining the angle of the force vector



```

IF A1 < > 0 THEN                                     400
  A1 = ACS((D1(1) - D0(1))/A1)                       410
  A1=360-(D1(2)<D0(2))+A1*((D1(2)=>D0(2))-(D1(2)<D0(2))) 420
;                                                     430
A2 = SQR((D2(1) - D0(1))2 + (D2(2) - D0(2))2)      440
IF A2 < > 0 THEN                                     450
  A2 = ACS((D2(1) - D0(1))/A2)                       460
  A2=360-(D2(2)<D0(2))+A2*((D2(2)=>D0(2))-(D2(2)<D0(2))) 470
;                                                     480
A3 = SQR((D3(1) - D0(1))2 + (D3(2) - D0(2))2)      490
IF A3 < > 0 THEN                                     500
  A3 = ACS((D3(1) - D0(1))/A3)                       510
  A3=360-(D3(2)<D0(2))+A3*((D3(2)=>D0(2))-(D3(2)<D0(2))) 520
;                                                     530
V(1)=V(1)+F1*COS(A1)+F2*COS(A2)+F3*COS(A3) ! Compute change in 540
V(2)=V(2)+F1*SIN(A1)+F2*SIN(A2)+F3*SIN(A3) ! X & Y components 550
D0(1) = D0(1) + V(1) ! of spacecraft velocity 560
D0(2) = D0(2) + V(2) 570
A = A + 1 ! Increase moon-earth angle by 1 degree 580
UNTIL ABS(D0(1)) > 200 OR D0(2) < -100 OR D0(2) > 200 590
END ! Continue until lost in space 600

```

### Free Fall Code Description

Sequence  
Number

Define D0 as a pair of values comprising the X,Y location of the ship, D1 the location of the moon, D2 the location of the earth, and D3 the location of Mars. Define D as a pair of numbers to be used when drawing one of the bodies, and V as the components of velocity of the spacecraft. 170

Read initial values for the velocity components of the spacecraft V so that  $V_x = 1$  and  $V_y = -.1$ , the initial angle of the moon relative to the earth  $A = 0$ , the gravitational constant  $G = 0.000001$ , the masses of the ship  $M_0 = 1$ , the moon  $M_1 = 1000000$ , the earth  $M_2 = 10000000$ , and Mars  $M_3 = 5000000$ , the initial locations of the ship  $D_0 = -50,40$ , the earth  $D_2 = -50,50$  and Mars  $D_3 = 50,80$ , and the radii of the moon  $R_1 = 1$ , the earth  $R_2 = 4$  and Mars  $R_3 = 2.5$ . 210

Let D = D2 and R = R2 to use Subroutine 620 to draw the earth. 220

Let D = D3 and R = R3 to use Subroutine 620 to draw Mars. 250

Begin to process the moving bodies. 280

Erase the moon (Subroutine 720). 290

Compute a new location of the moon relative to the earth and the angle between the earth and the moon, A. 300

Draw the moon in its new position. 340

Erase the spacecraft, then redraw it in its new position (Subroutine 680). 350



Compute F1 the gravitational force between the ship and the moon.	360
F2 the gravitational force between the ship and the earth.	370
F3 the gravitational force between the ship and Mars.	380
A1 the angle at which F1 is applied.	390
A2 the angle at which F2 is applied.	440
A3 the angle at which F3 is applied.	490

Assume that the resultant force is equal to the change in velocity (note from the formula  $F = ma$ , force equals mass times acceleration, that this is the case if  $m = 1$ . The mass of the spacecraft was chosen to be 1 in line 220), and compute the change to the X and Y components of the spacecraft's velocity.

540

Assume that an increment of velocity is equal to an increment of distance and increase the X and Y components of the location of the spacecraft accordingly.

560

Add 1 degree to the angle between the moon and the earth.

580

If the spacecraft is not lost in space, continue.

590

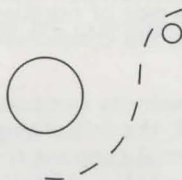
### Detecting Collisions

It is very easy to detect a collision between the spacecraft and one of the bodies since the ship is displayed as a dot. As soon as the dot is within one radius of the center of the body, it has crashed into that body. So in addition to the "lost in space" test, a "crashed" test is

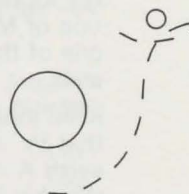
*Initial free-fall trajectory based on the constraints in the spacecraft program*



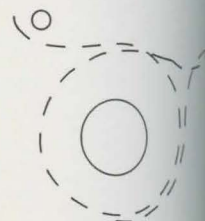
USING THE CONSTANTS FROM THE FREE FALL PROGRAM, THE STARTING POSITIONS OF THE EARTH, MOON AND SPACESHIP ARE SHOWN ABOVE. THE SHIP HAS AN INITIAL VELOCITY THAT WILL CAUSE IT TO MOVE TO THE RIGHT AND SLIGHTLY DOWNWARD.



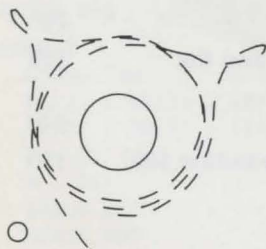
THE COMBINED GRAVITATIONAL PULL OF THE EARTH, MOON AND MARS DRAWS THE SPACESHIP PAST THE MOON.



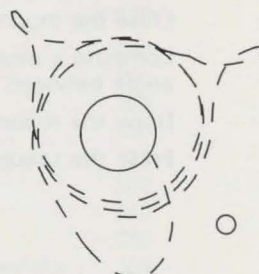
THE EARTH AND MOON ARE NOW PULLING THE SHIP IN THE SAME DIRECTION, SO THE ORBIT SWINGS BACK TOWARD THE EARTH AND THE SHIP GAINS SPEED RAPIDLY.



THE SHIP ORBITS THE EARTH ONCE AND IS THEN ATTRACTED BY THE MOON AGAIN



AFTER A SECOND ORBIT OF THE EARTH, THE SHIP PASSES BETWEEN THE EARTH AND THE MOON, THIS TIME FURTHER AWAY FROM THE MOON.



THE SPACESHIP IS SLOWED BY THE MOON, WHICH PASSES BY THE SHIP WHILE THE SHIP IS OUTSIDE THE MOON'S ORBIT. THE SPACESHIP CRASHES INTO THE EARTH.

```

IF R1 < SQR((D1(1) - D0(1))↑2 + (D1(2) - D0(2))↑2) OR
R2 < SQR((D2(1) - D0(1))↑2 + (D2(2) - D0(2))↑2) OR
R3 < SQR((D3(1) - D0(1))↑2 + (D3(2) - D0(2))↑2) THEN CRASHED

```

This test will be incorporated into the next program, which will include the capability to control the spacecraft.

Since the ship is shown only as a dot on the screen, it is not possible to tell which way the ship is pointing without additional data. One possibility is to allow the player to zoom in on the spacecraft when he wants to fire control rockets, but this then takes away his view of the planets and may cause a crash. A better idea is to provide some instruments at the bottom of the screen. The commander will need to see the attitude of the spacecraft (the direction it is pointing), its velocity vector (that is, the direction the ship is travelling), the amount of fuel remaining, and the number of orbits completed.

Realistically, as fuel is used up, the mass of the spacecraft should be decreased accordingly. To do so, it is necessary to decide what percentage of the total initial mass of the spacecraft was taken up by a full load of fuel and to reduce the spacecraft's mass proportionally as the fuel is consumed. Actually, this refinement would not be likely to have an appreciable effect on the performance of the game.

### Controlling the Spacecraft

Using the same screen units as before, -65 to 65 in X and 0 to 100 in Y, four squares of fifteen units on a side are drawn in the lower left-hand corner of the screen. In the middle two squares, circles of radius 5 are drawn. The squares are labelled "Fuel," "Attitude," "Velocity," and "Orbits." After each move of the spacecraft, these four indicators must be redrawn.

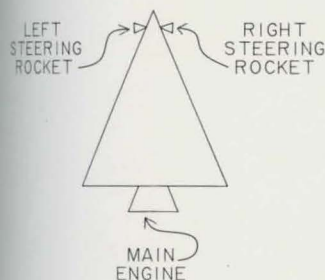
To maintain information about the spacecraft, several additional variables must be computed. Initially, the spacecraft must be controllable. Let us suppose it has three rocket engines: two steering rockets in the nose and a main thruster in the rear, as you see in the drawing of the spacecraft.

When a steering rocket is fired, it imparts a rotation to the spacecraft that will continue after the rocket is shut off. Therefore, two variables are necessary, the angle of attitude of the spacecraft and the angular rotation per second (or whatever time one move of the ship takes).

Whenever the right thruster is firing, the angular rotation B0 will be increased by 1 degree per second and decreased accordingly whenever the left thruster is firing. B0 will be added to the angle of attitude B during each time increment.

Each second, a steering rocket uses up one fuel unit and the main thruster uses three units. Fuel remaining, F, begins at 1000 and of course rockets are inoperable after  $F = 0$ . The variables L0 and R0 will be used to show that the left and right thrusters are firing—they will be 1 when firing and 0 otherwise. The variable T will show when the main thruster is firing. If  $T = 3$  when firing and zero otherwise, the fuel remaining can be computed every second by the formula:

$$F = F - T - L0 - R0$$



Spacecraft's three rocket engines



Since the game does not end when fuel is used up, it may be preferable to avoid reducing  $F$  below zero by the formula

$$F = (F - T - LO - RO) \text{ MAX } 0$$

$F$  can be displayed directly by printing its value in the left-hand box or showing it on a meter similar to an automobile fuel gauge.

The attitude of the spacecraft is shown by rotating a figure of the spacecraft through the angle  $B$  and drawing it in the "Attitude" box. As the drawing indicates, the figure can be quite simple.

Using the ROTATE A FIGURE subroutine, the numerical representation of the ship is

```
DATA 4,3,0,-2,2,-2,-2,3,0
DATA 4,-2,0.3,-3,0.5,-3,-0.5,-2,-0.3
DATA 4,2.5,0.2,2.6,0.4,2.4,0.4,2.5,0.2
DATA 4,2.5,-0.2,2.6,-0.4,2.4,-0.4,2.5,-0.2
DATA -9999
```

The program should include a flame emitting from any of the engines that are firing during the displayed time interval, because this is the only direct indication that a rocket is on or off. The captain could see that a rocket was firing by watching the "Fuel" indicator, but he would not know which steering rocket was on.

The "Velocity" indicator shows the direction in which the ship is travelling and how fast. A line is drawn from the center of the indicator, the point  $-22.5, 7.5$ , to the point  $V_x - 22.5, V_y + 7.5$ . If the length of this line is greater than 5, it will pass through the circle determining the size of the indicator, so to keep this from happening the line should be drawn only to the edge of the circle if it is greater than 5 units long:

```
MOVE -22.5,7.5
I = SQR(V(1)↑2 + V(2)↑2)
IF I < 5 THEN
  DRAW V(1) - 22.5, V(2) + 7.5
ELSE
  DRAW 5*V(1)/I - 22.5, 5*V(2)/I + 7.5
;
```

The number of orbits  $C$  is to be increased by one whenever the spacecraft crosses the orbit of one planet, having previously crossed the other planet's orbit, beginning with Mars. If a flag  $C0$  is set to zero to indicate that the next orbit to be crossed is that of Mars and to 1 if the earth's, then the test is

```
IF C0 = 0 AND D0(1) >= 50 THEN
  C0 = 1
  C = C + 1
ELSE
  IF C0 = 1 AND D0(1) <= -50 THEN
    C0 = 0
    C = C + 1
;
```

ATTITUDE



Attitude indicator on screen



To make a worthwhile game, the computer should have one or more ways to effect an interrupt; that is, the game progresses on its own until interrupted for a command to control the spacecraft. Three distinct interrupts are necessary, one for each of the three rockets. On many home computers it is possible to use a 'game paddle' for this function. A game paddle is a device with a rotary knob and a button. When the button is pushed, an interrupt occurs and the position of the rotary knob can be measured. If it is near the counterclockwise extreme it can control the left-turning rocket; near the other extreme, the right-turning rocket; and near the middle, the main thruster. It might be a good idea to paint indicator lines on the knob. Pushing the button toggles the rocket; that is, if the rocket were off, it would begin to fire, and if firing, it would shut off.

Here is the program for a controlled spacecraft.

### Controlled Spacecraft Program Code

	Sequence Number
DIM D0(2),D1(2),D2(2),D3(2),D(2),V(2)	170
DATA 1,-0.1,0,.000001,1,1000000,10000000,5000000	180
DATA -50,40,-50,50,50,80	190
DATA 1,4,2.5,0,0,0,0,0,0,1000	200
READ V,A,G,M0,M1,M2,M3,D0,D2,D3,R1,R2,R3,B0,B,L0,R0,T,C,CO,F	210
GOSUB 980 ! Draw instruments	220
D = D2	230
R = R2	240
GOSUB 860 ! Draw earth	250
D = D3	260
R = R3	270
GOSUB 860 ! Draw Mars	280
REPEAT	290
GOSUB 970 ! Erase moon	300
D1(1) = D2(1) + 15*COS(A)	310
D1(2) = D2(2) + 15*SIN(A)	320
D = D1	330
R = R1	340
GOSUB 860 ! Draw moon	350
GOSUB 930 ! Erase and draw spacecraft	360
F1=G*(M0*M1)/SQR((D1(1)-D0(1)) <sup>2</sup> +(D1(2)-D0(2)) <sup>2</sup> ) <sup>2</sup> ! Compute	370
F2=G*(M0*M2)/SQR((D2(1)-D0(1)) <sup>2</sup> +(D2(2)-D0(2)) <sup>2</sup> ) <sup>2</sup> ! forces	380
F3=G*(M0*M3)/SQR((D3(1)-D0(1)) <sup>2</sup> +(D3(2)-D0(2)) <sup>2</sup> ) <sup>2</sup> ! & angles	390
A1 = SQR((D1(1) - D0(1)) <sup>2</sup> + (D1(2) - D0(2)) <sup>2</sup> )	400
IF A1 <> 0 THEN	410
A1 = ACS((D1(1) - D0(1))/A1)	420
A1=360-(D1(2)<D0(2)) + A1*((D1(2)=>D0(2))-(D1(2)<D0(2)))	430
;	440
A2 = SQR((D2(1) - D0(1)) <sup>2</sup> + (D2(2) - D0(2)) <sup>2</sup> )	450
IF A2 <> 0 THEN	460
A2 = ACS((D2(1) - D0(1))/A2)	470
A2=360-(D2(2)<D0(2)) + A2*((D2(2)=>D0(2))-(D2(2)<D0(2)))	480
;	490

```

A3 = SQR((D3(1) - D0(1))2 + (D3(2) - D0(2))2)          500
IF A3 <> 0 THEN                                           510
  A3 = ACS((D3(1) - D0(1))/A3)                           520
  A3=360-(D3(2)<D0(2))+A3*((D3(2)=>D0(2))-(D3(2)<D0(2))) 530
;                                                         540

B0 = B0 + R0 - L0 ! Adjust rotation if steering rockets are on 550
B = B + B0          ! Change spacecraft attitude angle       560
V(1)=V(1)+F1*COS(A1)+F2*COS(A2)+F3*COS(A3)+SGN(T)*COS(B)/4 570
V(2)=V(2)+F1*SIN(A1)+F2*SIN(A2)+F3*SIN(A3)+SGN(T)*SIN(B)/4 580
D0(1) = D0(1) + V(1) ! Update spacecraft location           590
D0(2) = D0(2) + V(2)                                       600
F=F-T-L0-R0          ! Reduce fuel if thrusters firing      610
A = A + 1           ! Change moon's angle                   620
IF C0 = 0 AND D0(1) > =50 THEN ! Increase orbit count if    630
  C0 = 1              ! appropriate                          640
  C = C + 1           650
ELSE                  660
  IF C0 = 1 AND D0(1) < =-50 THEN 670
    C0 = 0              680
    C = C + 1           690
;                       700
;                       710
GOSUB 1240             ! Redraw instrument panel            720
K = 0                  ! Set "not crashed"                  730
IF R1 > SQR((D1(1) - D0(1))2 + (D1(2) - D0(2))2) THEN 740
  K = 1                ! Set "crashed" if hit moon          750
;                       760
IF R2 > SQR((D2(1) - D0(1))2 + (D2(2) - D0(2))2) THEN 770
  K = 1                ! Set "crashed" if hit earth         780
;                       790
IF R3 > SQR((D3(1) - D0(1))2 + (D3(2) - D0(2))2) THEN 800
  K = 1                ! Set "crashed" if hit Mars          810
;                       820
UNTIL K OR ABS(D0(1)) > 200 OR D0(2) < -100 OR D0(2) > 200 830
END                    ! Until crashed or lost              840

```

### Controlled Spacecraft Code Description

Sequence  
Number

Not shown is a routine that is activated by an interrupt. The selected rocket is toggled but is only turned on if the remaining fuel F is greater than zero.

The angular rotation of the spacecraft, B0, left steering rocket L0, right steering rocket R0, main thruster T, number of orbits C, and previous orbit C0 are all set to zero. Fuel remaining F is set to 1000. Free fall variables are set as before.	210
The instrument panel (Subroutine 980) and the planets are drawn (Subroutine 860).	220
The moon (Subroutine 860) and spacecraft (Subroutine 930) are drawn.	350
Forces and angles are computed.	370
Angular rotation B0 is adjusted if steering rockets are on.	550
Attitude of the spacecraft B is adjusted.	560
Vx and Vy are computed to include free fall and thrust vectors, assuming that if the main thruster T is firing, velocity is increased by 0.25.	570
The spacecraft's location is updated.	590
Fuel is reduced if thrusters are firing.	610
The moon's angular position is updated.	620
Number of orbits C is increased if appropriate.	630
The instrument panel is redrawn (Subroutine 1240).	720
If the spacecraft has not crashed and is not lost in space, the game	730

---

### **Refinements**

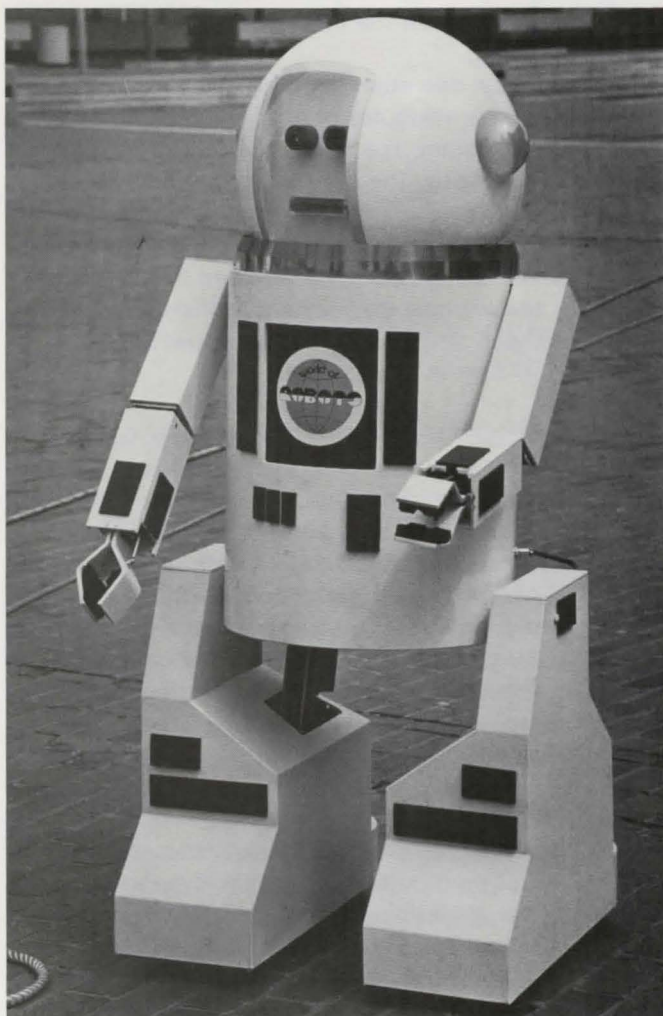
- Add random visible meteorites that destroy the spacecraft on collision.
- Require that the mission end when a spacecraft lands on one of the bodies. Landing should be considered successful only if the velocity on impact is small, say less than 1.
- Add instruments to the panel, for example, a display of B0.
- Expand the game to allow for two players. Include the possibility of a crash involving the two spacecraft. Allow a spacecraft to fire at the other at a cost of ten fuel units. If a hit occurs, randomly alter the velocity and attitude of the stricken ship.
- Improve the game's effects:
  - Randomly display stars in the background.
  - Add a roaring sound or a beep when rockets are firing.
  - Color the earth blue and white, the moon yellow, and Mars red; put canals on Mars and craters on the moon.
  - Make the instrument panel more like a real spacecraft panel.





---

## ***Instructional Games***



More than one home computer purchase has been justified by the argument that it will help the kids with their homework. Whether or not that turns out to be true, a great deal of effort has been spent on developing instructional software.

As a tutor, the computer has infinite patience. It can be tailored to the needs of the individual student and can introduce new material based on the student's learning rate.

Instructional software, especially for children, can be enlivened by presenting the material in a game. Our discussion of this aspect of *Games Computers Play* will begin with an introduction to artificial intelligence and then develop Concentration as an example of a learning game. Suggested memory drills and refinements complete the chapter.

### Artificial Intelligence

When a computer is made to exhibit what in humans would be called intelligent behavior, it is said to be using "artificial intelligence" (AI). Games that include the computer as one of the participants rely on the computer's artificial intelligence to make it play like a human being.

There has been a lot of philosophical discussion about intelligence and AI in the past fifty years. It is arguable that the attributes of artificial intelligence are not inherent in the computer and its program; the result resembles intelligent behavior because the programmer made it so. The same argument may be applied to trained animals. However, some psychologists teach that people are trained animals, too—trained by teachers, parents, friends, and enemies. In other words, people learn by experience. For our purposes, all we need is a computer program that learns by experience and we may say we have artificial intelligence. (Chapter 11 tells of a computer that was programmed to learn how to win at checkers by applying its experience in a series of games.)

### Concentration Game

One of the simplest uses of AI is to make the computer an interesting, but not overwhelming opponent in a game-playing situation. Consider the well-known game of Concentration. The original version is played with a deck of fifty-two playing cards. The cards are spread out on a table, face down, so that the back of each is fully visible. A player turns over a card and tries to match it by turning over another. If he finds a pair, he keeps them and tries again until he fails, whereupon he turns the two cards that were not a pair face down again. At the end of the game, the player who has uncovered the most pairs wins.

Concentration can be turned into a learning game with the aid of the computer. Objects to be matched can be animated figures, abstract objects, words, numbers, colors, or anything the computer can display.

Suppose there is a light pen or a touch-sensitive screen on the machine so that even children too young to use a keyboard can respond to the computer. A Concentration game in which a picture of an object is matched with the word for the object, as shown here, can be designed as a way to help teach reading.

In the one-player game, the computer must compete against the player in a nonfrustrating way. Obviously, the computer can be programmed to "peek" be-



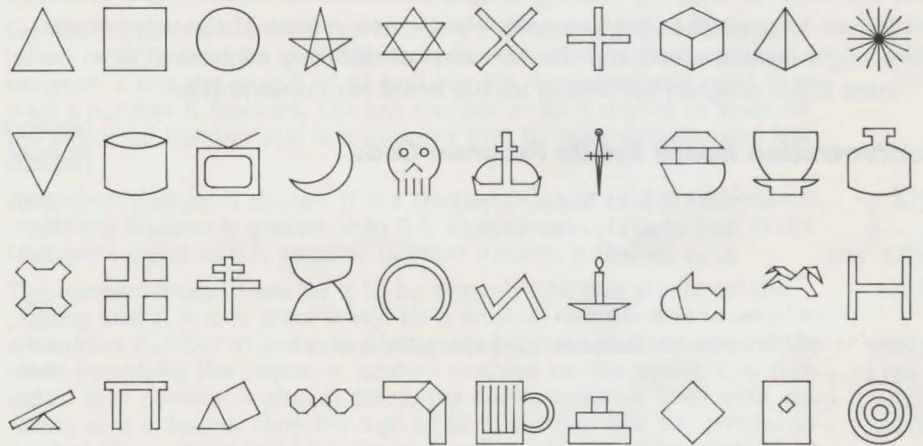
The game of Concentration as the fourth pair-match is attempted



hind the cards and never make a mistake, winning on its first turn. However, the program may be made to remember only those cards that have been turned and simply guess at a new card when no known pair exists. This version still gives the computer a considerable edge over most humans. What might be best would be a computer with a "fallible" memory that lets the player win about sixty percent of the games. Maybe this should be called "artificial stupidity"!

In Concentration, difficulty is due partly to the size of the board and partly to the appearance of the objects to be matched; it is harder to remember the location of abstract symbols than familiar objects such as animals. Thus, the simplest game might be played on a 4 x 4 board using objects familiar to small children. The next level of difficulty could be a 6 x 6 board in which the same objects have to be matched with words.

Abstract symbols that could make game  
of Concentration different



On the assumption that this game should be interesting to adults as well as children, the hardest level might be an 8 x 10 board that uses abstract symbols from a set such as the group of abstract symbols illustrated here.

At this level of difficulty, the one-player version should be played against a computer that never forgets cards that have been turned. This suggests that as the level of difficulty increases, the computer might become less and less forgetful, letting players at the lowest skill level win sixty percent of the time but decreasing the percentage as the skill level increases.

Suppose that after a player who identifies himself to the computer as Jim has played twelve games and won seven, his winning percentage is  $7/(7 + 5)$  or 0.583, that is, 58.3 percent. If Jim is to win sixty percent of the time, the computer may be given a  $60 - 58.3$  or 1.7 percent chance to "forget" a matched card. To do so, the program compares a random fraction with 0.017 and chooses at random a card other than the correct one if the random fraction is less than 0.017.

In general, the computer has the following probability of forgetting:

$$0.6 - \frac{\text{human wins}}{\text{games played}}$$

Clearly, when the human has won more than sixty percent of the games, the formula yields a negative number. The random fraction will always be greater than a negative number, so the computer will play with total recall until the human has once again won less than sixty percent of the games.

To widen the game's appeal, the number of players may be increased. When two or more humans are competing, the computer might or might not be a player.

In a version of Concentration using identifiable objects such as a kite or a ball, animation enlivens the game considerably. However, the duration of animation should be short—two or three seconds—and possibly occur only on the first appearance of the object. If sound accompanies the animation, it may be used in the game in which a word is matched with an object when either the word or the object is discovered. It is possible that using sound in this way may not produce the effect of teaching reading, since a dedicated nonreader might use only the sounds to remember the two cards. Hence, one version of the word-object game might include sound and the next level of difficulty might omit it.

Here is the program for setting up the board for Concentration:

<b>Concentration Board Set-Up Program Code</b>	<b>Sequence Number</b>
B1 = 0 ! Initialize board array	190
B2 = 0 ! and known cards array	200
GOSUB 690 ! Draw board	210
I = 1	220
REPEAT ! Put 30 numbers in selection array	230
B3(I) = I	240
I = I + 1	250
UNTIL I > 30	260
I = 1	270
REPEAT ! Select 18 objects	280
J = INT(RND(-2)*(31 - I)) + 1	290
K = B3(J)	300
B3(J) = B3(31 - I)	310
M = 0	320
WHILE M = 0 DO ! Randomly set M to +1 or -1	330
M = RND(-2) - 0.5	340
;	350
M = SGN(M)	360
J = 1	370
REPEAT ! Start at a random place	380
L = INT(RND(-2)*36) + 1 ! on the board	390
WHILE B1(L) <> 0 DO ! and find the next empty square	400
L = L + 1	410
IF L > 36 THEN	420
L = L - 36	430
;	440
;	450
B1(L) = K*M ! Save the object number and	460
M = -M ! do again—same number, opposite sign	470
J = J + 1	480



UNTIL J > 2	490
I = I + 1	500
UNTIL I > 8 ! for 18 pairs	510

### **Concentration Board Set-Up Code Description**

Two arrays B1 and B2 have been defined to contain thirty-six numbers each in preparation for a 6 x 6 game. B1 will contain the actual board and B2 the cards previously turned over. The computer will use B2 to determine its plays. B3 is an array of thirty numbers, because thirty different objects are available, even though only eighteen pairs can be selected for any one game. 190

Arrays B1 and B2 are set initially to zero and the game layout is drawn on the screen (Subroutine 690).

B3 is set to contain the numbers 1 through 30. 220

Eighteen different numbers representing eighteen objects will be selected random from B3 and placed on the board. A random number between 1 and the length of B3 (initially 30) is selected and used to extract a number K from B3. The last number in B3 is moved in place of the extracted number and it is assumed that B3 now contains one less number. 270  
290  
310

A random fraction is chosen. If the fraction is less than 0.5, M becomes -1. If the fraction is greater than 0.5, M becomes +1. Note that if the fraction is equal to 0.5, another random fraction is chosen. 320

The number chosen from B3 is to be entered into two places on the playing board. In one place it will be a positive number and in another a negative number in order to distinguish between the picture and the word describing the object. A random position on the board, L, is chosen. If that position is already filled, the next position is tried until an empty card is found. Then the sign of M is reversed and the process repeated. This is done eighteen times, that is, until each card contains an object and for every number representing a picture there is a corresponding number of the opposite sign representing the word for that picture. 370  
390  
470  
500

The computer determines whose turn it is and continues that turn until the player fails to find a match. For each match the player's score is increased. The computer must recognize the end of the game and terminate play. The easiest way to do so is to quit when the sum of the player's scores equals the number of pairs.

Whenever an object or word is disclosed, the B2 array is updated so the computer as a player can "remember" the object. After a match is made, the pair is removed from B1 and B2.

During the computer's turn, it must search B2 for known pairs and take them. (In this version, the computer is playing with total recall.) Then it selects a card that has not been turned and sees if it matches a card already known. If so, the computer takes the pair. Otherwise, it must select another unknown card. If it happens to match the previous card, the computer selects another unknown card and processes it as above, until a pair fails to match.

The program for playing Concentration is as follows.



**Playing Concentration Program Code**

	Sequence Number
H = 0	720
C = 0	730
REPEAT	740
REPEAT	750
GOSUB 1780	760
J = K	770
GOSUB 1780	780
IF ABS(B1(K)) = ABS(B1(J)) THEN	790
H = H + 1	800
B2(K) = 0	810
B2(J) = 0	820
B1(K) = 0	830
B1(J) = 0	840
GOSUB 1690	850
;	860
UNTIL ABS(B1(K)) <> ABS(B1(J)) OR H+C=18	870
!	875
B2(K) = B1(K)	880
B2(J) = B1(J)	890
GOSUB 1890	900
IF H + C < 18 THEN	910
I = 1	920
REPEAT	930
IF B2(I) <> 0 THEN	940
J = I + 1	950
REPEAT	960
IF ABS(B2(I)) = ABS(B2(J)) THEN	970
K = I	980
GOSUB 2010	990
K = J	1000
GOSUB 2010	1010
C = C + 1	1020
B1(I) = 0	1030
B1(J) = 0	1040
B2(I) = 0	1050
B2(J) = 0	1060
;	1070
J = J + 1	1080
UNTIL J > 36	1090
;	1100
I = I + 1	1110
UNTIL I > 35	1120
IF H + C < 18 THEN	1130
F = 1	1140
REPEAT	1150
L = INT(RND(-2)*36 + 1)	1160
IF B1(L)=0 OR B2(L)<>0 THEN	1170

```

REPEAT                                     1180
  L = L + 1                               1190
  IF L > 36 THEN                           1200
    L = L - 36                             1210
  ;                                         1220
  UNTIL B1(L) <> 0 AND B2(L) = 0           1230
;                                         1240
K = L                                     1250
GOSUB 2010 ! and display it               1260
I = 0                                    1270

REPEAT ! If there is a match among        1280
  I = I + 1 ! known cards,               1290
UNTIL ABS(B2(I)) = ABS(B1(L)) OR I = 36   1300
IF ABS(B2(I)) = ABS(B1(L)) THEN           1310
  K = I                                   1320
  GOSUB 2010 ! display the card,         1330
  C = C + 1 ! and take it.              1340
  B1(I) = 0                              1350
  B1(L) = 0                              1360
  B2(I) = 0                              1370
  B2(L) = 0                              1380
ELSE ! Otherwise, select another         1390
  B2(L) = B1(L) ! unknown card.          1400
  J = INT(RND(-2)*36 + 1)                1410
  IF B1(J) = 0 OR B2(J) <> 0 THEN         1420
    REPEAT                               1430
      J = J + 1                           1440
      IF J > 36 THEN                       1450
        J = J - 36                         1460
      ;                                   1470
      UNTIL B1(J) <> 0 AND B2(J) = 0       1480
    ;                                   1490
    B2(J) = B1(J)                         1500
    K = J                                 1510
    GOSUB 2010                            1520
    IF ABS(B1(J)) = ABS(B1(L)) THEN ! If cards match, 1530
      C = C + 1 ! take them               1540
      B1(J) = 0                           1550
      B1(L) = 0                           1560
      B2(J) = 0                           1570
      B2(L) = 0                           1580
    ELSE ! If not, flag "no match"        1590
      F = 0                               1600
    ;                                     1610
  ;                                     1620
  UNTIL F = 0 OR H + C = 18 ! Computer's turn ends 1630
; ! when all cards taken or no match      1640
;                                         1650
UNTIL H + C = 18 ! Game ends when all cards taken 1660

```

**Playing Concentration Code Description**

	Sequence Number
The human's score H and the computer's score C are set to zero.	720
Subroutine 1780 accepts a touch on the screen (or a light pen hit) and converts its location into a card location number K if the touch is in a valid place. The contents of the card are displayed. Two touches are accepted and the card numbers are placed in J and K.	750
If the cards match, the player's score is increased and the cards are removed from the board B1 and from the computer's memory B2.	790
Subroutine 1690 erases the two cards from the screen.	850
Lines 750 through 870 are repeated until the human fails to make a match.	870
The unmatched pair is placed in the computer's memory of the game and Subroutine 1890 erases the pictures or words, but not the cards.	880
If more cards remain, the computer searches its game memory B2 to see if it knows of any matches.	910
If it finds a match it takes the pair, adds 1 to its score, and removes the pair from the B1 and B2 arrays. If there are still more cards after all known pairs have been removed, the computer must select a previously unknown card.	970
A random integer between 1 and 36 is generated and the computer looks for a card that has not been taken and not been previously disclosed.	1130 1160
When such a card is found, it is displayed by Subroutine 2010 and B2 is examined to see if the computer knows the whereabouts of the matching card.	1230
If so, the computer takes the pair.	1310
If not, B2 is updated with the disclosed card and another new card is turned.	1390
If it matches the first card, the pair is taken and another card is disclosed.	1530
If they do not match, the computer's turn has ended.	1630
The game continues until all cards have been taken.	1660

**Memory Drills**

Games like Concentration can enliven study drills of many kinds, although using a computer for this purpose is not necessarily better than ordinary 3 x 5 cards with questions on one side and answers on the other. Additional use should be made of the computer as a patient tutor that remembers incorrect responses and repeats those questions. Given a computerized Concentration game, it is easy to expand the data base to include pictures and matching words for many kinds of drill.

Foreign language practice may readily include nouns (parts of the body, for instance) and action verbs (like "to run" or "to jump"). Conjugation ("he jumps," "you jump," and so on) may be more difficult. One way to approach conjugation drill would be to display identified figure groups representing "I," "he," "we," and so on, above the playing squares. Then, to animate "we jump," for example,



the "we" group would be shown in an animated jumping sequence in the picture square, to be matched with the words "we jump" in the appropriate language.

Most sciences may be subjects for instructional games. Beginning chemistry students may benefit from a chemistry apparatus drill—test tube, ring clamp, pipette, and so on. In geology, fossils may be represented if screen resolution is high enough. Arithmetic is another obvious possibility, in which the player must match " $2 + 2$ " with "4" or " $16/2$ " with "8." Shop classes are also candidates—auto parts, tools, machines, and so on.

Hobbies such as bird, tree, and flower identification may also be drill subjects (in identifying trees it is likely that the leaf should be displayed). Spelling drills are possible if the object is displayed and compared with several possible spellings of the word: A picture of a dog would be shown, followed by

1. Dug
2. Dogg
3. Doge
4. Dog

The player must then enter the number of the correctly spelled word.

### Refinements

- Include an opening dialogue asking for the number of players and their names. Look up each player's previous won-lost record against other players, including the computer, and display the record before the game begins. To do this, it will be necessary to maintain a disk or tape file of names and to update it every time a game is played.
- Display a running score for each player. Update it every time a pair is taken.
- When a player (e.g., Jim) wins, display "Jim wins!" and some skyrockets.
- Implement the artificial stupidity algorithm for games involving a beginner and the computer. Calculate

$$P = \frac{\text{games won}}{\text{total games}}$$

and cause the computer to "forget" by changing lines 970, 1310, and 1530 to

IF ABS(B2(I)) = ABS(B2(J)) AND P < RND(-2)	970
IF ABS(B2(I)) = ABS(B1(L)) AND P < RND(-2)	1310
IF ABS(B1(J)) = ABS(B1(L)) AND P < RND(-2)	1530

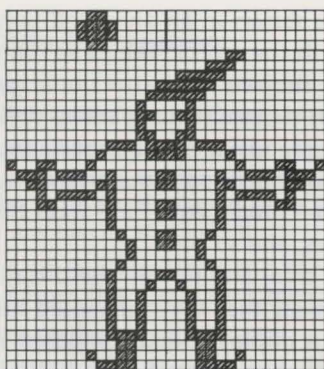
- Choose thirty objects that can be animated and cause them to be displayed through a short animated sequence. For example, here's a four-frame animation for a clown. To animate, display the pictures in the sequence 1,2,3,4,3,4,3,4.

Here's a three-frame sequence of a flag. To animate, display the pictures in the sequence 1,2,3,2,1,2,3,2,1,2,3.

And then a three-frame sequence of a clock. For animation, use the sequence 1,2,3,2,1,2,3,2,1,2,3.



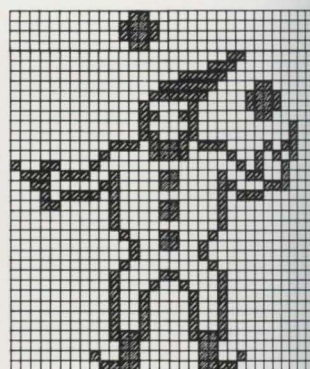
1



2

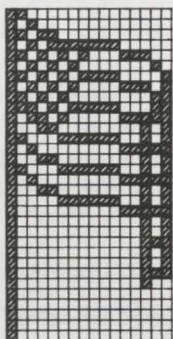
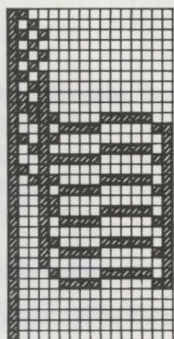
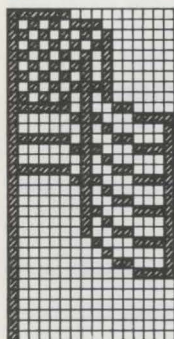


3

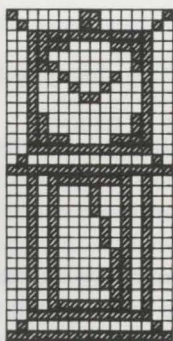
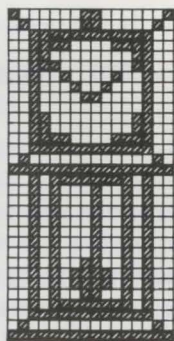
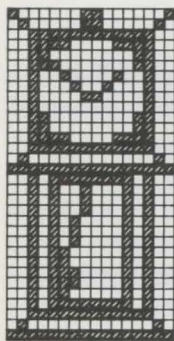


4

Animation sequence of a clown



Animation sequence of a flag



Animation sequence of a clock

Other possibilities include

Object	Animation
Ball	Three bounces
Kite	Oscillating tail
Dog	Wagging tail
Cat	From sitting to lying position
Frog	Jumping
Bike	Child pedalling
Fish	Swimming and ejecting bubbles
Tree	Falling leaf
Snake	Moving
Flower	Growing and blooming
Door	Opening
Box	Opening
Book	Opening
Rabbit	Hopping
Truck	Dumping
Crane	Raising bucket
Bird	Flying

- Add sound to each animated sequence.

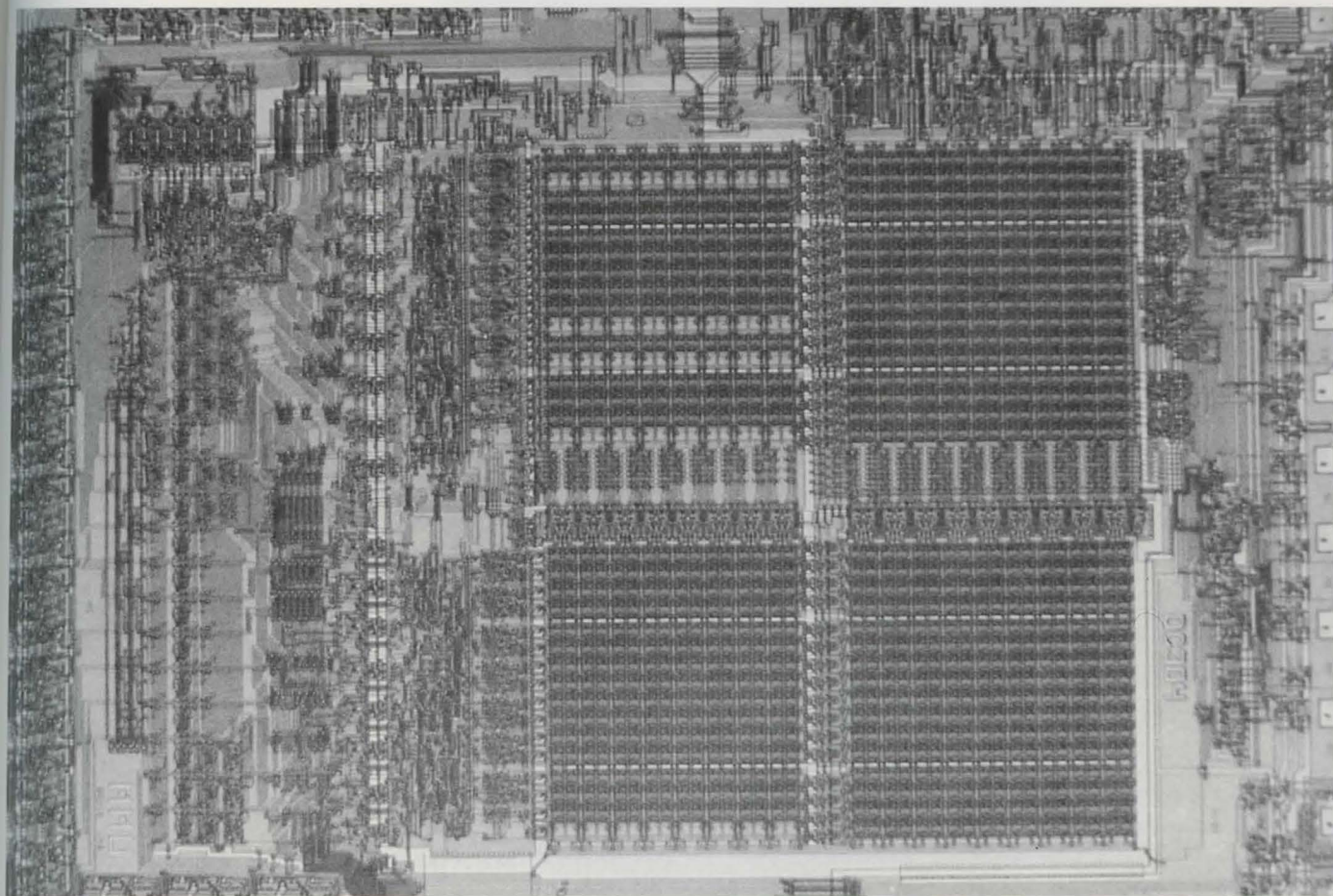




Part Three

---

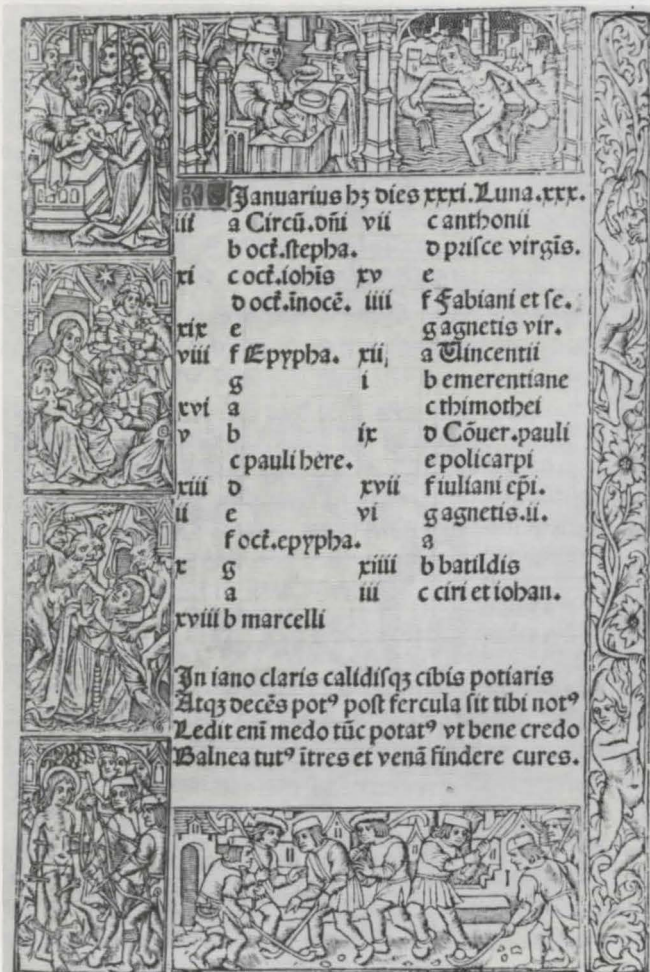
**Finishing Touches** *esign*







## Computer-Aided Design



As we have already discussed, the finishing details are what make a computer game look right. Good pictures and varied text give a polished, professional appearance but can be tedious to produce. However, techniques currently in use in the industrial world can come to the rescue.

Wide acceptance of computer graphics in industry has come through the development of computer-aided design (CAD). Automobiles and aircraft have been designed with the aid of computers for several years, not only because of cost reductions but also because of the other benefits that come from computer modelling—automatic drafting, fabrication, and simulation. A design flaw found by a computer can prevent a costly recall of an entire automobile model line or avoid a hazardous situation in an aircraft.

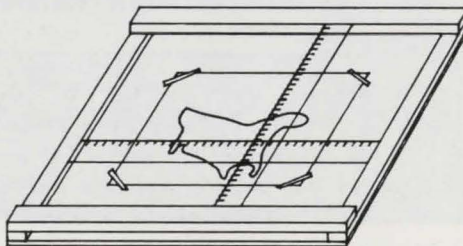
Computers can be used to design all kinds of other products such as dress patterns, house plans, electrical circuits, landscapes, and so on, all with potential benefits beyond the original intent. Given a computer model of a house, the machine can easily produce an accurate list of the components needed to build it. With the proper program and data base, a landscape plan can be simulated into the future to see how the property will look after the trees and bushes have grown. Even the colors of the flowers in April, May, and June can be evaluated without the expense and time needed to look at the real garden.

The basic elements of a CAD system are useful in generating the graphics for computer games. A CAD program was used to develop the opening frame for *The Magic Cave* shown in the color illustrations. A tree, ivy leaf, or flower was modelled one time. After the design was accepted, the program generated a subroutine that would replicate the tree, leaf, or flower in the location and scale specified in the subroutine calling sequence. In this way it was possible to pay attention to detail in the original design and then to get a lot of use out of the images by specifying only translation and scaling values.

## Digitizing

Since everything is expressed numerically inside a computer, the programmer cannot simply tell it to display a leaf or tree on the screen. The object must be defined precisely by mapping its outline—choosing significant points and assigning X,Y values to them. Usually, this is done with a digitizer, a machine that recognizes the position of a pointer or joystick and passes the coordinate values to the computer. An inexpensive semiautomatic digitizer is easy to build.

As shown in the illustration of a digitizer, sliding the two plastic rulers to a point on a drawing and reading the X and Y values where they cross will give a point value that can be entered into the computer and subsequently converted from inches into actual units (miles, feet, and so on) based on the scale of the drawing being digitized.



*An easy-to-build semiautomatic digitizer*



### Text Definition

Only a few types of geometric element are needed to represent a model. At a minimum, most models can be represented by points, lines, circular arcs, and circles. To make a readable engineering drawing, some text is also necessary.

A point can be represented by an X,Y pair of numbers. A line requires the X,Y values of the endpoints of the line. A circle is defined by its radius and the X,Y location of its center, and a circular arc also requires a starting and ending angle.

Customized text is much more complicated. In addition to the letters, numbers, and punctuation, such things as the height and width of each character, the angle of the text line, the slant of each character, and the character font are important. "Font" is the design of the set of characters, several of which are shown here, ranging from very simple to rather ornate. (See Font Tables at the end of the book.)

The easiest to implement is the Stick font, because all the characters are composed only of lines. The Leroy font requires circular arcs, Clearface uses elliptical arcs, and Gothic also requires that polygons and pairs of arcs be filled with ink. So it would seem that five different data structures are required for the four fonts: lines, circular arcs, elliptical arcs, filled polygons, and filled elliptical arcs. Since a circular arc is the same as an elliptical arc in which the major and minor radii are of the same length, only four types need to be defined.

Each character must be defined in a way that is independent of its ultimate size, because in one case the user may want characters 0.1 inches high for a normal page of text while in another he may be making a poster and want letters two inches high. It is reasonable to decide arbitrarily that each character will be defined on a grid and that a typical character will be six units wide and eight units high, as the letter shown defined on the grid here.

To define the letter in terms of the straight lines representing it, lines are drawn from 0,0 to 3,8 to 6,0. Then a line is drawn from 1,2.333 to 5,2.333. In many cases, connecting lines can be drawn, so the data structure to define a sequence of connected lines may consist of the data type code for a line, say the number 1, followed by the number of X,Y pairs defining the endpoints of the lines, as follows:

```
10 DATA "A"
20 DATA 1,3,0,0,3,8,6,0
30 DATA 1,2,1,2.333,5,2.333
40 DATA -1
```

Line 10 is used to define the fact that the following codes represent the capital letter A. On line 20, the first number specifies a line sequence. The second number specifies that three pairs of points comprise the sequence. The remaining six numbers are the actual pairs  $X_1,Y_1 = 0,0$ ,  $X_2,Y_2 = 3,8$ , and  $X_3,Y_3 = 6,0$ . The next number must be a data type code again, and it is. The first number on line 30 specifies another line sequence, this time with only two points,  $X_1,Y_1 = 1,2.333$  and  $X_2,Y_2 = 5,2.333$ . So the next number is again a data type, this time the number -1, which is the "end of character" code.

Data type 2 will represent an elliptical or circular arc. Definition of such an element requires the location of the arc's center,  $X_0,Y_0$ , the radius on a line par-

STICK, LEROY,  
CLEARFACE or  
GOTHIC

four fonts



Defining a letter on a  $6 \times 8$  grid



Arc defined by the structure 2, 3, 5, 2,  
3, 0, 90



allel to the x-axis  $R_x$ , the y-radius  $R_y$ , and the initial and final angles  $\alpha_1$  and  $\alpha_2$ . So data type 2 always consists of seven values—the structure

2,3,5,2,3,0,90

defines the arc shown here on a grid.

Data type 3 will be used to describe a filled, or solid, figure. Following the type code is the number of pairs of points defining the figure and then the pairs themselves. If the figure is a closed polygon, the last pair will be the same as the first. Thus, the structure

3,5,2,3,2,6,4,6,4,3,2,3

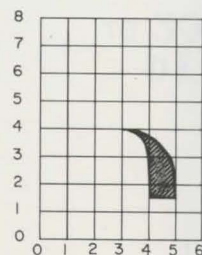
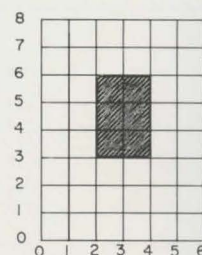
represents the closed polygon shown on a grid.

Data type 4 defines a pair of elliptical arcs, the space between which is filled. This structure always consists of nine numbers: the type,  $X_0, Y_0$ , the mutual center of the arcs,  $R_{x1}$  and  $R_{y1}$  of the inside arc,  $\alpha_1$  and  $\alpha_2$  and  $R_{x2}$  and  $R_{y2}$  of the outside arc. For example

4,3,1.5,1,2.5,0,90,2,2.5

defines the elliptical arc shown on a grid.

To simplify processing,  $R_{x1} \leq R_{x2}$ ,  $R_{y1} \leq R_{y2}$ , and  $\alpha_1 < \alpha_2$ . To draw a counter-clockwise arc from 270 degrees to 90 degrees,  $\alpha_1 = -90$ , not 270, and  $\alpha_2 = 90$ .



Elliptical arc defined by the structure 4, 3, 1.5, 1, 2.5, 0, 90, 2, 2.5

## Text Sequence

To generate a text sequence, the user will want to be able to specify the actual text, the font, the height, and the width of each character in the sequence, plus some other variables. For instance, as shown by the accompanying sequences, it may be useful to italicize (slant) the characters, or to display the sequence at an angle, or even to rotate each character about its center, or any combination of the above.

Since some characters are typically wider than others, it is a good idea to express in the data base the location of the center of each character. This not only allows rotation about the center of the defined character space but also specifies the relative width of the overall character. A W is much wider than an I, for example, while an A is somewhere in between, as demonstrated by the comparison here of these letters on the grids.

So it is necessary to add the center point of each character to the data base; thus the definition of the Stick A becomes:

```
10 DATA "A",3,4
20 DATA 1,3,0,0,3,8,6,0
30 DATA 1,2,1,2.333,5,2.333
40 DATA -1
```

ABC

Italicized sequence of characters

ABC

Angled sequence of characters

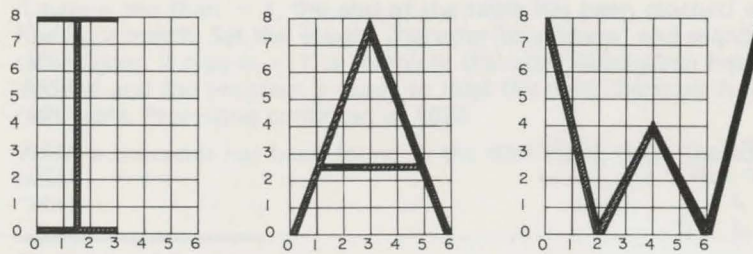
A B C

Individually rotated sequence of characters

A  
B  
C

Character string rotated 270°, individual characters rotated 90°

comparison of relative widths of characters of the same height



### Font Table Storage and Retrieval

A complete font table will consist of all the upper- and lower-case letters, ten digits, and punctuation characters. A unique data code will indicate the end of the table, in case a character is not found in the font. In this case, the character will be replaced by a "space." The end-of-table code used in the following examples is any number less than -1.

Note that the sample programs deal with a font as though it were expressed in the program in DATA statements. Since the four fonts together require that about 64,000 characters be represented in DATA statements, it is not acceptable to retain fonts in this way in most personal computers. Instead, the fonts should be written on an external storage medium such as disk and accessed from there.

Given a text string in B\$, how can the stroke table for each character be found in the font table? It is necessary to isolate one character at a time (into C\$ in the example) and progress through the table until the position of the character in the font table is found. If the stroke table is kept on disk, each character can be related to its disk address by a list in memory and thereby be accessed directly.

Several parameters must be known to process a found character:

- X,Y the location of the lower left-hand corner of the position on the screen of the first character in the input string B\$
- H the height of the character in inches
- W the width of the character in inches
- S the angle of slant of a character
- A the angle of the text sequence
- B the angle of rotation of each character about its center

Here is a program for looking up a character in a font table.

### Look Up Character in Font Table Program Code

```

RESTORE                ! Begin at the top of the font table      4000
REPEAT                 4010
  READ A$              ! Read a character                        4020
  IF A$ < > C$ THEN    ! If not the sought character,          4030
    READ N,N           ! skip over the values describing it    4040
    REPEAT              4050
      READ T            4060

```



CASE T OF	4070
1:	4080
READ C	4090
J = 1	4100
REPEAT	4110
READ N,N	4120
J = J + 1	4130
UNTIL J > C	4140
;	4150
2:	4160
READ N,N,N,N,N,N	4170
;	4180
3:	4190
READ C	4200
J = 1	4210
REPEAT	4220
READ N,N	4230
J = J + 1	4240
UNTIL J > C	4250
;	4260
4:	4270
READ N,N,N,N,N,N,N,N	4280
;	4290
;	4300
UNTIL T < 0     !   until entire table has been examined	4310
IF T < -1 THEN !   If character not found,	4320
C\$ = " "     !   replace it with a space	4330
RESTORE	4340
;	4350
;	4360
UNTIL A\$ = C\$     !   and try again	4370
RETURN	4380

### Look Up Character in Font Table Code Description

	Sequence Number
Begin at the top of the font table.	4000
Read a character from the font table.	4020
If it is not the character sought, read the X,Y values of the center of the character to skip over them.	4030
Then read the character type.	4060
If type = 1, read the number of point pairs into C and skip over C pairs of points.	4080
If type = 2, skip over six variables.	4160
If type = 3, read the number of point pairs into C and skip over C pairs of points.	4190
If type = 4, skip over eight variables.	4270
Continue to process lines 4050 through 4310 until type is a negative number.	4310



If type is less than - 1, the end of the table has been reached without finding a match. Set the sought character to a 'space' and search the table again. If type is - 1, a complete character description has been skipped and the program is ready to read the next character from the font table. Processing continues at 4020. 4320

When a character has been found in the font table, leave the subroutine. 4370

### Horizontal Letters

To keep things simple, let us begin by considering Leroy lettering on a horizontal line. Since you have to express character height and width in inches, you must decide what one unit on the character grid represents. The typical grid is eight units high and six units wide, so an arbitrary decision can be made that each grid unit is one-eighth of an inch.

To scale the characters to the specified size, it will be necessary to multiply X-grid units by W and Y-grid units by H and then divide by 8. After a character has been drawn, the X,Y values of the lower left-hand corner of the first character must be increased by the width of one character, plus something for space between characters. Since a character is six grid units wide, adding two more should be appropriate.

The following program can be used for horizontal lettering in fonts composed only of lines and arcs (types 1 and 2.)

### Horizontal Leroy Program Code

```

READ XO,YO          ! Read the character's center          5000
REPEAT                                     5010
  READ T              ! Read type                            5020
  CASE T OF                                     5030
    1:                ! Type = line: Read number of endpoints 5040
      READ C          5050
      J = 1           5060
      REPEAT          5070
        READ X1,Y1 ! Read an endpoint                      5080
        IF J = 1 THEN ! If first endpoint,                  5090
          MOVE X + X1*W/8,Y + Y1*H/8 ! move to it          5100
          ;                                                  5110
          DRAW X + X1*W/8,Y + Y1*H/8 ! Draw to endpoint     5120
          J = J + 1    5130
        UNTIL J > C   ! for all endpoints                    5140
      ;                                                  5150
    2:                ! Type = arc: Read center, X & Y radii, 5160
      READ X1,Y1,R1,R2,A1,A2 ! and initial & final angle    5170
      MOVE X+(X1+R1*COS(A1))*W/8,Y+(Y1+R2*SIN(A1))*H/8 ! Move 5180
      J = A1          ! to beginning of arc                 5190
  
```

```

REPEAT                                     ! Draw along arc           5200
  DRAW X + (X1 + R1*COS(J))*W/8,Y + (Y1 + R2*SIN(J))*H/8          5210
  J = J + 1                               ! in one-degree increments 5220
UNTIL J > A 2                             5230
;                                           5240
;                                           5250
UNTIL T < 0                               ! to end of character       5260
X = X + (2*X0 + 2)*W/8                    ! Move to next character   5270
RETURN                                     5280

```

### Horizontal Leroy Code Description

	Sequence Number
Read the character center into X0,Y0	5000
Read data type into T.	5020
If the type is 1, a line, read the number of endpoints C.	5040
Read a pair of grid coordinates X1,Y1.	5080
If this is the first pair, move to it after scaling; otherwise, draw to it.	5090
Repeat the process from 5080 for each pair of coordinates.	5140
If the type is 2, an arc, read the arc's center X1,Y1, the x and y radii R1 and R2, and the initial and final angle A1 and A2.	5160
Move to the beginning of the arc.	5180
Draw in one-degree increments to the end of the arc.	5200
Repeat the process from 5020 until there are no more entities defining this character.	5260
Locate the lower left-hand corner of the next character by adding 2 grid units to twice the x-center of the character and scaling the result.	5270

### Slanted Letters (Italics)

The addition of slanted or italic character capability expands the ways in which a single font can be displayed without requiring additions to the font data base and with only a minor extension of the code.

An italic character leans to the right. If the slant angle is 10 degrees, as shown in the accompanying diagram, a line defined as vertical must remain anchored at  $Y = 0$  but must lean 10 degrees to the right. So the x-value of a point must vary as a function of its y-value according to the formula

$$\begin{aligned}
 X &= X + \Delta x \\
 \text{or } X &= X + \Delta y * \tan(\sigma) \\
 X &= X + \Delta x, \text{ or } X = X + \Delta y * \tan(\sigma)
 \end{aligned}$$

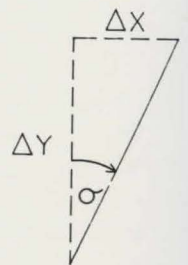
where  $\sigma$  is the slant angle. So each x-grid point becomes increased by  $y\text{-grid} * \tan(\sigma) * \text{height}/8$ , as shown on lines 5100, 5120, 5175, and 5205 of the following code for adding.

### Italics Added Program Code

```

READ X0,Y0                               5000
REPEAT                                    5010

```



Determining italic slant of a letter



```

READ T                                         5020
CASE T OF                                     5030
  1:                                           5040
    READ C                                     5050
    J = 1                                       5060
    REPEAT                                     5070
      READ X1,Y1                               5080
      IF J = 1 THEN ! Italic transformation for line included 5090
        MOVE X + X1*W/8 + Y1*H/8*TAN(S),Y + Y1*H/8 5100
      ;                                         5110
      DRAW X + X1*W/8 + Y1*H/8*TAN(S),Y + Y1*H/8 5120
      J = J + 1                                 5130
    UNTIL J > C                               5140
  ;                                           5150
  2: ! Italic transformation for arc included 5160
    READ X1,Y1,R1,R2,A1,A2                    5170
    X2 = (X1 + R1*COS(A1))*W/8 + (Y1+ R2*SIN(A1))*H/8*TAN(S) 5175
    MOVE X + X2,Y + (Y1 + R2*SIN(A1))*H/8      5180
    J = A1                                       5190
    REPEAT                                     5200
      X2 = (X1 + R1*COS(J))*W/8 + (Y1+R2*SIN(J))*H/8*TAN(S) 5205
      DRAW X + X2,Y + (Y1 + R2*SIN(J))*H/8      5210
      J = J + 1                                 5220
    UNTIL J > A2                               5230
  ;                                           5240
;                                           5250
UNTIL T < 0                                    5260
X = X + (2*X0 + 2)*W/8                        5270
RETURN                                         5280

```

### Character Rotation

Now rotation of characters about their centers can be added. The usual definition of rotation calls for counterclockwise movement, but the natural idea of rotation about a center, at least for right-handed people, is clockwise. Consequently, the expressed angle of rotation must be used with the opposite sign.

As shown in lines 5090–5120 of the following code, the center of the character is subtracted from the x,y value of the end of a line or a step around an arc, because rotation is done about the origin. Then the rotation formula is applied and the point is translated back to a point relative to the character's center. Since  $\cos(-B) = \cos(B)$  and  $\sin(-B) = -\sin(B)$ , lines 5110 and 5120 can also be written

$$\begin{aligned}
 X3 &= X2*\cos(B) + Y2*\sin(B) + X0*W/8 \\
 Y3 &= Y2*\cos(B) - X2*\sin(B) + Y0*H/8
 \end{aligned}$$

Notice that a great many redundant calculations are occurring. The values  $X0*W/8$  and  $Y0*H/8$  could be computed after line 5000 and thus be done only once per character rather than twice per stroke on straight lines and twice per degree on arcs. The values for  $\sin(-B)$ ,  $\cos(-B)$ ,  $\tan(S)$ ,  $H/8$ , and  $W/8$  remain constant for the entire character sequence and should be computed outside the sub-



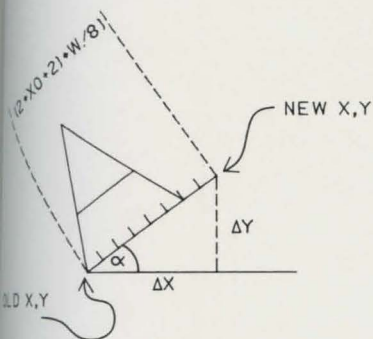
routine. The computations shown here in the code are not simplified in order to show all the details of each transformation.

### Character Rotation Added Program Code

```

READ X0,Y0                                5000
REPEAT                                     5010
  READ T                                   5020
  CASE T OF                               5030
    1:                                     5040
      READ C                               5050
      J = 1                                5060
      REPEAT ! Italics and rotation for line added 5070
        READ X1,Y1                         5080
        X2 = X1*W/8 + Y1*H/8*TAN(S) - X0*W/8 5090
        Y2 = Y1*H/8 - Y0*H/8               5100
        X3 = X2*COS(-B) - Y2*SIN(-B) + X0*W/8 5110
        Y3 = X2*SIN(-B) + Y2*COS(-B) + Y0*H/8 5120
        IF J = 1 THEN                       5130
          MOVE X + X3,Y + Y3                5140
          ;                                  5150
          DRAW X + X3,Y + Y3                5160
          J = J + 1                          5170
        UNTIL J > C                         5180
      ;                                     5190
    2: ! Italics and rotation for arc added 5200
      READ X1,Y1,R1,R2,A1,A2               5210
      X2=(X1+R1*COS(A1))*W/8+(Y1+R2*SIN(A1))*H/8*TAN(S)-X0*W/8 5220
      Y2 = (Y1 + R2*SIN(A1))*H/8 - Y0*H/8 5230
      X3 = X2*COS(-B) - Y2*SIN(-B) + X0*W/8 5240
      Y3 = X2*SIN(-B) + Y2*COS(-B) + Y0*H/8 5250
      MOVE X + X3,Y + Y3                   5260
      J = A1                               5270
      REPEAT                               5280
        X2=(X1+R1*COS(J))*W/8+(Y1+R2*SIN(J))*H/8*TAN(S)-X0*W/8 5290
        Y2 = (Y1 + R2*SIN(J))*H/8 - Y0*H/8 5300
        X3 = X2*COS(-B) - Y2*SIN(-B) + X0*W/8 5310
        Y3 = X2*SIN(-B) + Y2*COS(-B) + Y0*H/8 5320
        DRAW X + X3,Y + Y3                 5330
        J = J + 1                          5340
      UNTIL J > A2                         5350
    ;                                     5360
  ;                                     5370
UNTIL T < 0                                5380
X = X + (2*X0 + 2)*W/8                     5390
RETURN                                     5400
;                                         5410

```



Determining sequence angle of a character

### Sequence Angle

Having performed scaling, slanting, and character rotation, we may add the character sequence angle  $\alpha$ , as in lines 5130 and 5140 of the following code. In the previous transformations it was necessary only to modify the X part of the position of the lower left-hand corner of the character (line 5390 in the previous listing), because the character sequence was horizontal and therefore the Y-value did not change. Now, as shown in the accompanying drawing, the location of the next character will change in X and Y as follows (see lines 5450 and 5460):

$$x = (2 \cdot X_0 + 2) \cdot W/8 \cdot \cos(\alpha)$$

$$y = (2 \cdot X_0 + 2) \cdot H/8 \cdot \sin(\alpha)$$

It is worth noting that any transformation sequence, no matter how complex, can be expressed in a pair of equations. Consider lines 5090–5140. The right-hand sides of the expressions in lines 5090 and 5100 can be substituted into lines 5110 and 5120 wherever X2 and Y2 appear, thus eliminating lines 5090 and 5100.

Then the same process can be applied for X3 and Y3 in lines 5130 and 5140, eliminating lines 5110 and 5120. The BASIC language, in general, does not allow for statements longer than one line, although some personal computer versions will permit multiple-line BASIC statements.

### Sequence Angle Added Program Code

```

READ X0,Y0                                     5000
REPEAT                                         5010
  READ T                                       5020
  CASE T OF                                   5030
    1:                                         5040
      READ C                                   5050
      J = 1                                   5060
      REPEAT ! Italics, character angle, and sequence angle 5070
        READ X1,Y1 ! for line                 5080
        X2 = X1*W/8 + Y1*H/8*TAN(S) - X0*W/8 5090
        Y2 = Y1*H/8 - Y0*H/8                 5100
        X3 = X2*COS(-B) - Y2*SIN(-B) + X0*W/8 5110
        Y3 = X2*SIN(-B) + Y2*COS(-B) + Y0*H/8 5120
        X2 = X3*COS(A) - Y3*SIN(A)           5130
        Y2 = X3*SIN(A) + Y3*COS(A)           5140
        IF J = 1 THEN                         5150
          MOVE X + X2,Y + Y2                 5160
          ;                                   5170
          DRAW X + X2,Y + Y2                 5180
          J = J + 1                           5190
        UNTIL J > C                          5200
      ;                                       5210
    2: ! Italics, character angle, and sequence angle 5220
      READ X1,Y1,R1,R2,A1,A2 ! for arc        5230
      X2=(X1+R1*COS(A1))*W/8+(Y1+R2*SIN(A1))*H/8*TAN(S)-X0*W/8 5240

```

```

Y2 = (Y1 + R2*SIN(A1))*H/8 - Y0*H/8          5250
X3 = X2*COS(-B) - Y2*SIN(-B) + X0*W/8        5260
Y3 = X2*SIN(-B) + Y2*COS(-B) + Y0*H/8        5270
X2 = X3*COS(A) - Y3*SIN(A)                    5280
Y2 = X3*SIN(A) + Y3*COS(A)                    5290
MOVE X + X2,Y + Y2                            5300
J = A1                                         5310

REPEAT                                         5320
  X2=(X1+R1*COS(J))*W/8+(Y1+R2*SIN(J))*H/8*TAN(S)-X0*W/8 5330
  Y2 = (Y1 + R2*SIN(J))*H/8 - Y0*H/8          5340
  X3 = X2*COS(-B) - Y2*SIN(-B) + X0*W/8      5350
  Y3 = X2*SIN(-B) + Y2*COS(-B) + Y0*H/8      5360
  X2 = X3*COS(A) - Y3*SIN(A)                  5370
  Y2 = X3*SIN(A) + Y3*COS(A)                  5380
  DRAW X + X2,Y + Y2                          5390
  J = J + 1                                    5400
UNTIL J > A2                                  5410

;                                              5420

;                                              5430

UNTIL T < 0                                    5440

X = X + (2*X0 + 2)*W/8*COS(A)                 5450
Y = Y + (2*X0 + 2)*W/8*SIN(A)                 5460
RETURN                                         5470

```

### Filled (Shaded) Polygons

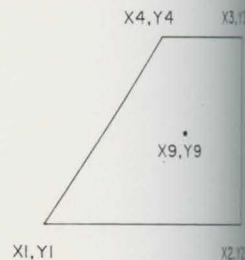
Now, filled polygons and arcs may be added to complete the character-generating routine. To draw a filled polygon, read all the defining points into an array, draw the polygon, then compute a new set of points inside the previous set, and redraw until the polygon is full. Consider the polygon shown here.

The point which is the "average" of the nodes of the polygon is  $X_9, Y_9$ , where  $X_9 = (X_1 + X_2 + X_3 + X_4)/4$  and  $Y_9 = (Y_1 + Y_2 + Y_3 + Y_4)/4$ , or in general, as shown in the formulas here.

$$X_9 = \frac{\sum_{i=1}^n X_i}{n} \quad Y_9 = \frac{\sum_{i=1}^n Y_i}{n}$$

This point is always inside a convex polygon. After drawing the original polygon, each node  $X, Y$  may be brought closer to  $X_9, Y_9$  by moving it part way along the line between  $X, Y$  and  $X_9, Y_9$ .

To draw solid arcs it is necessary to draw successive arcs, beginning with the arc of minimum radius and increasing the radius slightly until the larger arc has been drawn. In the following program, repeated code sequences have been made into subroutines.



Simple polygon to be filled



**Complete Character Generator Program Code**

```

READ X0,Y0                                         5000
REPEAT                                             5010
  READ T                                           5020
  CASE T OF                                       5030
    1:                                             5040
      READ C                                       5050
      J = 1                                       5060
      REPEAT                                       5070
        READ X1,Y1                               5080
        X2 = X1*W/8 + Y1*H/8*TAN(S) - X0*W/8     5090
        Y2 = Y1*H/8 - Y0*H/8                     5100
        GOSUB 6190                                5110
        IF J = 1 THEN                             5120
          MOVE X + X2,Y + Y2                       5130
          ;                                         5140
          DRAW X + X2,Y + Y2                       5150
          J = J + 1                                5160
        UNTIL J > C                               5170
      ;                                           5180
    2:                                             5190
      READ X1,Y1,R1,R2,A1,A2                     5200
      X2=(X1+R1*COS(A1))*W/8+(Y1+R2*SIN(A1))*H/8*TAN(S)-X0*W/8 5210
      Y2 = (Y1 + R2*SIN(A1))*H/8 - Y0*H/8         5220
      GOSUB 6190                                5230
      MOVE X + X2,Y + Y2                         5240
      J = A1                                       5250
      REPEAT                                       5260
        X2=(X1+R1*COS(J))*W/8+(Y1+R2*SIN(J))*H/8*TAN(S)-X0*W/8 5270
        Y2 = (Y1 + R2*SIN(J))*H/8 - Y0*H/8         5280
        GOSUB 6190                                5290
        DRAW X + X2,Y + Y2                       5300
        J = J + 1                                5310
      UNTIL J > A2                               5320
    ;                                           5330
    3:                                             5340
      ! Type = filled polygon                    5340
      DELETE F                                     5350
      READ N                                       5360
      ! Number of nodes                          5360
      DIM F(2,N)                                  5370
      I = 1                                       5380
      X9 = 0                                       5390
      Y9 = 0                                       5400
      REPEAT                                       5410
        READ F(1,I),F(2,I) ! Read a pair of points 5420
        X9 = X9 + F(1,I)/N ! and accumulate their 5430
        Y9 = Y9 + F(2,I)/N ! average               5440
        I = I + 1                                5450
      UNTIL I > N                                  5460
      X2 = F(1,1)*W/8 + F(2,1)*H/8*TAN(S) - X0*W/8 ! Transform 5470

```

```

Y2 = F(2,1)*H/8-Y0*H/8          ! first point          5480
GOSUB 6190                        5490
MOVE X + X2,Y + Y2              ! and move to it        5500
REPEAT                           5510
  L=(F(1,1)-X9)↑2+(F(2,1)-Y9)↑2 ! Compute distance      5520
  I = 2                          5530
  REPEAT                         5540
    L = L MAX (F(1,I) - X9)↑2 + (F(2,I) - Y9)↑2        5550
    X2 = F(1,I)*W/8 + F(2,I)*H/8*TAN(S) - X0*W/8      5560
    Y2 = F(2,I)*H/8 - Y0*H/8                          5570
    GOSUB 6190                                          5580
    DRAW X + X2,Y + Y2! Draw a polygon                 5590
    I = I + 1                                           5600
  UNTIL I > N                                           5610
  L = 1 - 0.1/SQR(L) ! Compute a step inward          5620
  I = 1                                                 5630
  REPEAT                                               5640
    F(1,I) = (F(1,I) - X9)*L + X9 ! Adjust polygon     5650
    F(2,I) = (F(2,I) - Y9)*L + Y9                     5660
    I = I + 1                                           5670
  UNTIL I > N                                           5680
UNTIL L < 0.1          ! until polygon filled          5690
;                                                         5700
4:                               ! Type = filled arc    5710
READ X9,Y9,X1,Y1,A1,A2,V2,W2    ! Read arc data       5720
IF ABS(V2 - X1) >= ABS(W2 - Y1) THEN ! Compute ratio   5730
  J = (W2 - Y1)/(V2 - X1 + (V2 = X1)) ! If Xchange≥     5740
  K = (V2-X1)/5 MIN 0.1          ! Ychange             5750
  REPEAT                           5760
    GOSUB 6010          ! Draw arc                     5770
    X1 = X1 + K         5780
    Y1 = Y1 + J*K       5790
    UNTIL X1 + K > V2    5800
  ELSE                          ! Compute ratio if      5810
    J=(V2-X1)/(W2-Y1+(W2=Y1)) ! Ychange > Xchange      5820
    K + (W2 - Y1)/5 MIN 0.1    5830
    REPEAT                   5840
      GOSUB 6010 ! Draw arc 5850
      X1 = X1 + J*K 5860
      Y1 = Y1 + K    5870
      UNTIL Y1 + K > W2 5880
    ;                  5890
  IF (X1 <> V2) OR (Y1 <> W2) THEN 5900
    X1 = V2          ! Draw final arc 5910
    Y1 = W2          5920
    GOSUB 6010       5930
  ;                  5940
;                  5950
;                  5960
UNTIL T < 0          ! until end of character          5970

```

```

X = X + (2*X0 + 2)*W/8*COS(A)          5980
Y = Y + (2*X0 + 2)*W/8*SIN(A)          5990
RETURN                                  6000

! Draw elliptical arc                    6010
X2=(X9+X1*COS(A1))*W/8+(Y9+Y1*SIN(A1))*H/8*TAN(S)-X0*W/8  6020
Y2 = (Y9 + Y1*SIN(A1))*H/8-Y0*H/8      6030
GOSUB 6190                              6040
MOVE X + X2,Y + Y2                      6050
I = A1                                  6060

REPEAT                                  6070
  X2=(X9+X1*COS(I))*W/8+(Y9+Y1*SIN(I))*H/8*TAN(S)-X0*W/8  6080
  Y2 = (Y9 + Y1*SIN(I))*H/8 + Y0*H/8    6090
  GOSUB 6190                              6100
  DRAW X + X2,Y + Y2                     6110
  I = I + 5                               6120
UNTIL I >= A2                             6130

X2=(X9+X1*COS(A2))*W/8+(Y9+Y1*SIN(A2))*H/8*TAN(S)-X0*W/8  6140
Y2 = (Y9 + Y1*SIN(A2))*H/8 - Y0*H/8     6150
GOSUB 6190                              6160
DRAW X + X2,Y + Y2                      6170
RETURN                                  6180

! Perform rotation                       6190
X3 = X2*COS(-B) - Y2*SIN(-B) + X0*W/8  6200
Y3 = X2*SIN(-B) + Y2*COS(-B) + Y0*H/8  6210
X2 = X3*COS(A) - Y3*SIN(A)              6220
Y2 = X3*SIN(A) + Y3*COS(A)              6230
RETURN                                  6240

```

### Complete Character Generator Code Description

	Sequence Number
Straight lines and simple arcs are dealt with as before.	5000
If the element is a filled polygon, the number of nodes N in the polygon is read, and an array F of N pairs is established.	5340
The average of these points will be computed as X9,Y9.	5390
Pairs of points are read and the average point is accumulated.	5410
The first point is transformed and a move is made to that point.	5470
The value L is to be the distance moved along the line from each point to the average point X9,Y9. As the polygon is drawn, L is made to be the square of the longest distance from a point to X9,Y9.	5520
L becomes a fraction of the distance from X9,Y9 to the most distant node.	5620
All nodes are moved closer to X9,Y9.	5640
Processing continues from 5510 until the polygon is filled.	5690
If the element is a filled arc, the arc's center X9,Y9, initial radii X1,Y1, starting and ending angles A1 and A2, and final radii V2,W2 are read.	5710
If the difference $\Delta x$ between the initial and final x-radii is greater than $\Delta y$ , the ratio $\Delta y/\Delta x$ is computed. Note the precaution to compute $\Delta y/1$ if $\Delta x = 0$ . Then K, the increment of radius change is computed.	5730



Subroutine 6010 draws the elliptical arc.	5770
The x-radius is increased by K and the y-radius by $K \cdot \Delta y / \Delta x$	5780
Arcs are drawn until the next x-radius will be greater than the final x-radius.	5800
A similar procedure is done for $\Delta y > \Delta x$ .	5810
If the final arc has not been drawn, it is drawn.	5900
The X and Y location of the next character is updated and the routine ends. Subroutine 6010 draws an elliptical arc:	5980
	6010
The initial point is transformed and moved to.	6020
Points along the arc are computed, transformed, and drawn to in 5-degree increments until the next increment will equal or exceed the ending angle A2.	6070
The point at the final angle is drawn to.	6140
Subroutine 6190 rotates X2,Y2 about the character's center, giving X3,Y3, which is rotated onto the angle of the character sequence, giving X2,Y2.	6190

### Nontext Fonts

Font tables need not be restricted to text. Many kinds of symbols are amenable to the same manipulations useful in the selection and placement of letters and numbers. Electrical symbols, for example, may be associated with letter codes, so that the input character R can cause a resistor symbol to be drawn. A complete set of electrical symbols is the basis for a program to draw schematic diagrams.

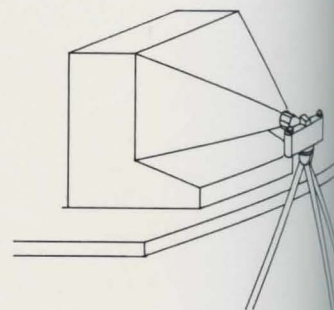
Similarly, mapping symbols can be used for mapping or charting. The length of a road section may be adjusted— independent of the distance between the lines—by changing the input value of the “width” modifier. To make posters or visual aids, symbols like circles and squares are useful, and flow diagrams need special characters. A font that includes trees, flowers, animals, clouds, and many other repeated objects can be used for pictures, titles, and so forth.

### Hard Copies

If you develop an original game, you may want to send a copy of some of the displays to friends, keep a permanent record of the screen images, or even use selected scenes to advertise it for sale. While standard text can be copied on a printer, a graphics output device is necessary to capture a hard, or permanent copy of a picture on the screen.

The least expensive graphics output device is an ordinary camera. It must be capable of focusing on the screen from 18 to 24 inches away and must have adjustable shutter speeds. Photographs taken at speeds faster than 1/30 of a second will show a black stripe on part of the screen due to the way in which the image is generated and refreshed. In 1/10 of a second the computer's screen is refreshed three times, so a shutter speed of 1/10 or 1/4 will eliminate the black stripe. It will be necessary to mount the camera on a stable base or tripod in order to avoid a blurred picture.

Unless the picture is taken in a dark room, it is very likely that reflections from the surface of the screen will detract from the quality of the picture. Be-



Using a hood to eliminate reflections and interference when photographing a computer screen

cause the brightness of the picture may reflect from mirrors or picture glass, a hood that fits on the computer is a very useful accessory to make, as illustrated here.

### **Picture Maker**

A font table is only part of a picture maker. In order to be most useful, the table should be controlled by an interactive program so the user can see the picture as it develops and be able to correct mistakes. This implies a data base that defines the elements of the picture.

In addition to the complex elements in font tables, it is useful to allow direct entry of simple entities such as points, lines, arcs, rectangles, and circles.

It would be nice to have a joystick for positioning a cursor (on-screen pointer) and a large set of programmable function keys for insertion, deletion, copying, and other functions needed in an interactive picture generator. However, such a program can be written for a computer with only a keyboard and a graphics screen. A storage medium such as tape or disk is also necessary in order to save the data that define a picture. The entire screen can be made available for the picture except for one line of text.

Arcs, circles, and rectangles may be hollow or filled with a specified color. Begin by setting the entire screen to a given color. After entities are placed on the screen, the program must be able to move, copy, or delete them and to change fonts as necessary. After deletion of an entity the picture will be incomplete—either an automatic REPAINT may occur or a REPAINT command may be invoked by the user. To work in areas where exact placement is required, a ZOOM command can enlarge a selected quarter of the screen so that each pixel is now seen as a 2 x 2 square. Another ZOOM will enlarge a newly selected screen quarter to 4 x 4, and so forth.

When the picture is complete, a SAVE command should allow the user to name the picture and save the data base associated with the name. Thereafter, a RESTORE command should ask for the name and reload the picture from the disk.

### **Insertions**

Two-character codes may be assigned for entry from the keyboard. If possible, the computer should accept two keystrokes without a Return and check to be sure they represent a valid code. If the numbers 0 to 7 are assigned (assuming an eight-color display) to the colors black, blue, green, cyan, red, magenta, yellow, and white, the code Bn, where "n" is one of the numbers 0 through 7, can be used to mean "set the entire screen (B = background) to the color n." Correspondingly, Cn can mean "use the color n on subsequent entities until another Cn command." The prefix I can be used for Insert, so that

Code	Means
IP	Insert point
IL	Insert line
IA	Insert arc
IC	Insert circle
IR	Insert rectangle
IT	Insert text



FA, FC, and FR can mean insert a filled arc, circle, or rectangle. The D prefix can be used for Delete, M for Move, and C for Copy. Thus, DC means DELETE A CIRCLE, although which circle to delete has not been specified in the command.

The ZOOM command Zn, where  $n = 1-9$ , means "ZOOM SCREEN SECTION n," where n equals one of the specified sections of the screen, as shown in the accompanying figures.

The command Z0 restores the screen to normal size.

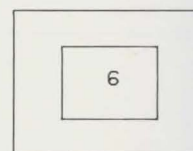
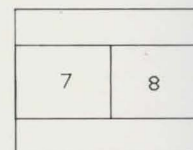
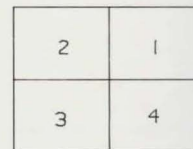
Insertion implies adding to the data base as well as to the picture. There are nine separate entity types:

Type	Code
Background	0
Point	1
Line	2
Arc	3
Circle	4
Rectangle	5
Filled Arc	6
Filled Circle	7
Filled Rectangle	8
Text	9

Data needed to describe each type are

Type	Data
Background	Code = 0,color
Point	Code = 1,color,X,Y of point
Line	Code = 2,color,X1,Y1,X2,Y2 (endpoints)
Arc	Code = 3,color,X0,Y0 (center), R (radius), $\alpha 1$ , $\alpha 2$ (starting and ending angle)
Circle	Code = 4,color,X0,Y0,R
Rectangle	Code = 5,color,X1,Y1,X2,Y2 (opposite corners)
Filled Arc	Code = 6,color,X0,Y0,R, $\alpha 1$ , $\alpha 2$
Filled Circle	Code = 7,color,X0,Y0,R
Filled Rectangle	Code = 8,color,X1,Y1,X2,Y2 (opposite corners)
Text	Code = 9,color,X,Y,f,H,W, $\sigma$ , $\alpha$ , $\beta$ ,Nc,N1

The text definition is the most complex. It requires that either the actual text be placed in a string (alphanumeric array) or that each text character be converted into a numeric code and stored following Nc as part of the numeric data base (in which case N1 is not needed). Text data base entries are as follows:



Screen sections designated in ZOOM command



Variable	Meaning
X,Y	Screen location of lower left corner of first character
f	Font number
H	Text height
W	Text width
$\sigma$	Slant angle
$\alpha$	Character sequence angle
$\beta$	Character rotation angle
Nc	Number of characters in this sequence
N1	Location of first character in string array

To insert a point, the program must recognize the IP command, then issue a message like "Move cursor to point." The operator then uses the arrow keys to move the cursor, pressing Return to signal that the point is to be placed at the cursor. Point code 1, color code, and the X,Y location of the cursor are placed at the end of the data base list and the pixel at X,Y is colored as specified.

A line is inserted similarly by accepting the command IL, moving the cursor to one end of the line, pressing Return, moving the cursor to the other end, and pressing Return again.

A circle is inserted by first entering the center X0,Y0 and then moving the cursor to a point on the circumference X1,Y1. The radius R is computed from

$$R = \text{SQR}((X1-X0)^2 + (Y1 - Y0)^2)$$

A rectangle may be drawn from any opposite pair of points X1,Y1 and X2,Y2:

```
MOVE X1,Y1
DRAW X2,Y1
DRAW X2,Y2
DRAW X1,Y2
DRAW X1,Y1
```

The easiest way to enter an arc is to specify with the cursor first the center, then a point on the circumference at the starting angle, then the point at the ending angle. Three points on the circumference—start, intermediate, and end—will also define an arc.

When text is to be inserted, the user must enter not only the actual text but also several parameters. Assuming that the commands F1 to F9 cause selection of a font, then f should be set to the appropriate font number automatically. The slant angle, sequence angle, and rotation angle must be entered by the user, and consequently the program must issue prompting messages for these inputs. The number of characters and their starting position in the string array can be computed automatically.

### Deletions

In order to effect a delete, move, or copy, the program must find the entity to be processed. The type (such as DC for DELETE CIRCLE) reduces the number of entities involved. The programmer has at least two ways to establish which is to be deleted. Each entity can be made to blink by alternately displaying it in black and white, say three times, after which the user responds with Y for Yes or N for

No, until all entities have been flashed or a Y has been pressed. A better method is to determine the distance from a controllable cursor to each selected entity (points, for example) and the closest deleted after all points have been checked.

Determining the distance from the cursor to a line is not so easy. If the shortest perpendicular distance from the cursor  $X_c, Y_c$  to each line is used, a situation such as shown here would result in the deletion of the line between  $X_1, Y_1$  and  $X_2, Y_2$  because the distance  $P_1$  to the extended line is shorter than  $P_2$ , although the cursor is closer to the line segment  $X_3, Y_3$  to  $X_4, Y_4$ . This problem may be eliminated by determining the  $X, Y$  value of the points  $Q_1$  and  $Q_2$ . If the coordinates  $X_q, Y_q$  of the point  $Q_1$  are such that

$$X_1 \leq X_q \leq X_2$$

$$\text{and } Y_1 \leq Y_q \leq Y_2$$

then the perpendicular distance is the shortest distance to the line segment. If not, then

$$d_1 = \text{SQR}((X_1 - X_c)^2 + (Y_1 - Y_c)^2)$$

$$\text{and } d_2 = \text{SQR}((X_2 - X_c)^2 + (Y_2 - Y_c)^2)$$

should be computed and the lesser of distances  $d_1$  and  $d_2$  used instead of  $P_1$ .

In the next drawing, the length of  $P$  (see Note 9 at the end of the book) is

$$P = \frac{((X_c - X_1)(Y_c + Y_1) + (X_2 - X_c)(Y_2 + Y_c) - (X_2 - X_1)(Y_2 + Y_1))}{\text{SQR}((X_2 - X_1)^2 + (Y_2 - Y_1)^2)}$$

The coordinates of the point  $X_q, Y_q$  can be found from

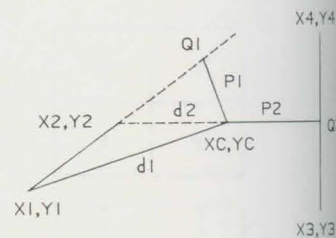
$$X_q = \frac{X_1 \frac{Y_2 - Y_1}{X_2 - X_1} + X_c \frac{X_2 - X_1}{Y_2 - Y_1} + Y_c - Y_1}{\frac{Y_2 - Y_1}{X_2 - X_1} + \frac{X_2 - X_1}{Y_2 - Y_1}}$$

$$\text{and } Y_q = \frac{X_1 - X_2}{Y_2 - Y_1}(X_q - X_c) + Y_c$$

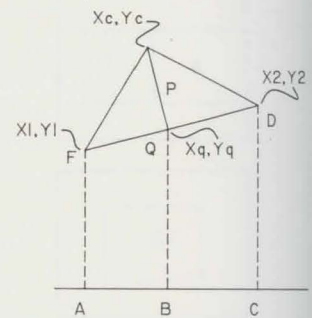
Text is the most difficult case of all. To determine which of several text strings is to be deleted, moved, or copied, think of a text sequence as a rectangle of the form shown here: so that the four corners are known. If the sequence is not horizontal, the coordinates of the corners are transformed by the rotation equations about  $X, Y$ . Is the cursor inside this rectangle? If so, the text sequence is the one to be processed.

If the cursor is inside the rectangle  $ABCD$ , positioned at the point  $Q$ , as shown in the illustration, then the sum of the areas of the four triangles  $AQD$ ,  $AQB$ ,  $BQC$ , and  $DQC$  is the same as the area of the rectangle. If not, as shown in the next illustration, the sum of the areas of the four triangles is greater than the area of the rectangle. To find the area of an arbitrary triangle, the trapezoidal method described above may be used. The area of a rectangle is, of course, the base times the height.

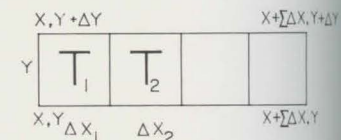
To delete a found entity, establish its location in the data base and remove it by flagging or by overlaying with valid entities.



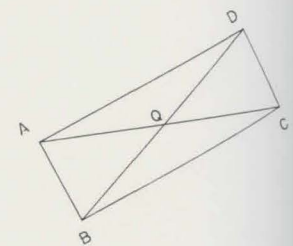
*Incorrect determination of the distance from the cursor to a line, since point  $Q_1$  is not on the line segment  $X_1, Y_1$  to  $X_2, Y_2$*



*Determining the length of  $P$ , the shortest distance from a cursor to line segment*



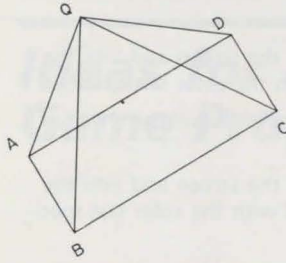
*Conception of text sequence showing the four corner coordinates*



*In locating a text sequence with a cursor at point  $Q$ , the cursor is inside the rectangle bounding the text only if the sum of the areas of the four triangles equals the area of the rectangle*



If the cursor is outside the bounding rectangle, the sum of the areas of the four triangles AQB, BQC, CQD, and ACD will be greater than the area of the rectangle ABCD.



### Moves

A MOVE is merely a translation from one point expressed by a cursor location to another. That is, if a MOVE is invoked and the entity is established relative to a point X,Y, then after the entity is determined, two values must be given by the user, a "Move from" X1,Y1 and a "Move to" X2,Y2. To accomplish the move, the entity should be erased by redrawing it in black, then a new location X',Y' is computed from

$$\begin{aligned} X' &= X + X2 - X1 \\ Y' &= Y + Y2 - Y1 \end{aligned}$$

X',Y' then replaces X,Y in the data base and the entity is redrawn at X',Y'.

### Copies

A COPY is identical to a MOVE except that the original entity is not erased. This means that a copy of the original data base entry must be generated with location X',Y' in place of X,Y.

### Codes

A possible set of codes and their functions is

Code	Name of Command	Function
Bn	SET BACKGROUND	The "Set Background" code colors the entire screen according to the number "n," where n = 0,1,2,...,7 to represent the colors black, blue, green, cyan, red, magenta, yellow, and white
Cn	SELECT COLOR	Following INSERT commands will use this color for the inserted entity
Ce	COPY ENTITY	"Copy Entity" selects an entity of type "e" already on the screen and makes another copy of it in a cursor-specified location and into the data base, where "e" = A(arc), C(circle), L(line), P(point), R(rectangle), or T(text)
De	DELETE ENTITY	"Delete Entity" selects an entity of type "e" and deletes it from the screen and from the data base
FA	INSERT FILLED ARC	The "FA" code inserts onto the screen and into the data base an arc of a circle that is filled with the color last specified



Code	Name of Command	Function
FC	INSERT FILLED CIRCLE	The "FC" code inserts onto the screen and into the data base a circle filled with the color last specified
Ff	SELECT FONT	Following INSERT TEXT commands will use the font numbered "f"
FR	INSERT FILLED RECTANGLE	The "FR" code inserts onto the screen and into the data base a rectangle filled with the color last specified
le	INSERT ENTITY	"Insert Entity" inserts onto the screen and into the data base an entity of type "e," colored with the color last specified
Me	MOVE ENTITY	An entity of type "e" already on the screen is moved to a new X,Y location. The data base values X,Y of the entity are updated.
RE	REPAINT	The screen is erased and redrawn from the information stored in the data base.
RI	RESTORE IMAGE	The data base for a previously saved image is found on a disk according to its name and drawn on the screen.
SI	SAVE IMAGE	The data base along with a picture name is written on the disk.
Zq	ZOOM	A specified quarter "q" of the screen is doubled in size. Z0, UNZOOM, restores the original image.

---

## ***Ideas for Hobbyist Graphics and Game Programs***





The best ideas for programs you can write will be those derived from your own experiences—hobbies, job, school subjects, and so forth. Obviously there is no need to be an astronaut in order to write an outer space game, but knowledge of physical laws makes that game realistic.

A fertile imagination also helps. Fantasy adventure games are not drawn from real life but from a combination of imagination and experience with maze games. Use of your own special skills should help you to produce a superior game. If you really like the finished product, you may want to try to sell it.

To make money by selling home-grown computer programs is tough, but some people have realized big profits doing so. An original, imaginative, useful program or an innovative game must be marketed aggressively in order for it to sell well. One way to try selling a program is to offer it to an established manufacturer (however, it's a good idea first to protect it by copyrighting). Another way is to offer copies of the program for sale at local computer stores, but unless an entire chain carries your product, sales will come in ones and twos. Books are available that pursue the subject of marketing in depth.

### Hobbyist Programs

Hobbies are good subjects for the personal computer programmer. Whether you write to sell or for fun, the important thing is to know the subject and to make the program interesting and easy to use. Just about any pastime or skill may be converted into material for a computer program.

Consider bird-watching, for example: Given enough patience and computer storage, information on local birds could be collected from the available literature in a form amenable to "decision tree" processing. When an unknown bird is glimpsed, the mystified birder could go to his computer, which would ask a series of questions that could be answered Yes, No, or Unsure (keys Y, N, and ?). An example of such a decision is shown here.

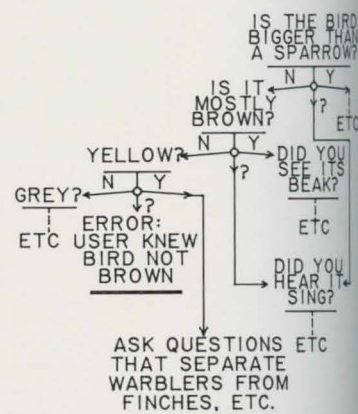
Graphical displays are useful for identification: The program could give the instruction "Select from the following the beak resembling your bird's beak" and then show the bird-watcher the choice of beaks pictured here.

Other features of the program should show characteristics of any possible bird, such as how it flies, how it walks or hops, its coloring, and its song if the computer has a good tone generator. The user should also be able to ask to see a particular bird by entering its name.

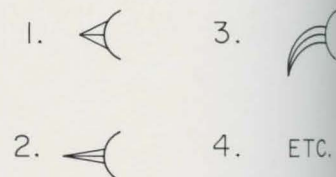
After a complete set of local birds is stored, migratory birds and "accidentals" could be added. (Accidentals are birds not common to the area but occasionally seen, for instance, after having been blown off course during storms.)

Hobbyist gardeners or landscape architects might write a program in which a site plan can be drawn, locating trees and bushes. The user would then be able to look at various views (from the street, from the living room window, and so on) and see how the site will look from year to year as the plants grow according to their known characteristics.

A vegetable garden planner would keep information from year to year on plant location and yield, warning the gardener of potential problems like "Too many kale crops in this bed" or "Tomatoes will shade carrots."



Decision tree for sample questions in a bird watching program



Display for choosing bird beak sighted by bird watcher



A house plan expediter would be useful to architects and builders. The computer would help design the house (and produce drawings if a plotter is attached), while at the same time computing the materials needed to build the house.

An interesting program that most people could use from time to time would be a trip planner. Such a program would require a lot of data, although the data base could be built up gradually, beginning with major roads within a few miles of the user's house. The program would have to be provided with speed limits on each road section, traffic lights (and their settings, ideally, although a good guess as to duration of green and red should do), patterns of traffic—trucks, peak times, and even where school buses and crossing guards may interrupt progress.

Given these data, the fastest route from one point to another could be computed by using a technique similar to the maze solution algorithm discussed in Chapter 7. Personal route preferences might also be considered, giving extra weight to chosen routes.

There is potential for a game called *The Mad Commuter* that might use a map of Manhattan—Long Island, Los Angeles, or any other famous traffic disaster area. Players would select routes, choose to take risks like speeding or passing on the right, and try to reach a given goal before the other players. Of course, they should be subject to policemen and traffic tickets if they decide to violate rules.

The accuracy of some of the above ideas in birding, gardening, and driving depends on the time of year, including potential weather conditions, time of day, day of the week, and so on, so a set of subroutines with that information would be an important tool. For example, the constellations in your location could be displayed only if the time of year and time of day (or night, in this case) are available to the program.

Needlework and weaving are wide-open hobby areas as far as computer-aided design is concerned. In weaving, patterns are generated by passing individual threads through eyes in the centers of metal wires, called "heddles," held in a frame. Thus a simple over-and-under weave is created by alternately threading heddles in two frames, which are interchanged by foot treadles as a shuttle carries a thread back and forth. More complicated looms have four frames and six or eight foot pedals to raise combinations of heddles, thereby producing very complex patterns. Programs have been written to allow testing of weaving patterns; they show the colors that result when the vertical (warp) threads that pass through the heddles interact with the horizontal (woof) threads from the shuttle. The color patterns depend on such things as the treadle sequence and which threads overlay the others.

A straightforward computer-aided design application is a furniture planner and arranger for interior decorators. The most useful program would allow the user to input a house plan and model each piece of furniture in three dimensions. The program should then be capable of displaying wire-frame models of each room seen from any aspect. Refinements include a perspective display of solid objects, in correct colors, to give a realistic idea of room layout.

One of the most impressive hobbyist applications is a music editor. The user enters musical notation on a staff and causes the computer to read the musical codes and generate the correct sounds. Good music programs currently require auxiliary devices to play the music well, but there are many computerized music machines available today. Before long, integrated digital stereo, television, and computer systems will be available that will permit programs that synthesize all kinds of sounds—music, voice, and special effects.

Physical phenomena are particularly amenable to computerization because for the most part they are very well defined. Optics—lens and prism systems—is relatively easy to simulate on a computer and can be useful to photographers and amateur astronomers. Personal computers are in frequent use to control such things as temperature and air conditioning in the home. A nice application for a gardener is the monitoring of climatic conditions to control greenhouse and cold frame environment and to compare the environments with crop yield. A small additional step is the monitoring of soil condition and temperature to report ideal planting times. Many such applications require the use of a histogram or bar graph program.

---

## **Games**

No book can cover all the possibilities for computer games. The following samples may suggest directions you would like to take in creating your own programs.

Simple entertainment games or cartoons can be interesting exercises in programming, although their appeal doesn't last very long. A very straightforward example of this type was used to sell early Apple computers. The program used random numbers to color small rectangles on the screen in a continually changing random pattern somewhat similar to a kaleidoscope.

I once wrote an animation demonstration called "Fencing Fools," in which an ornate castle with two towers and a balcony was displayed, as pictured here.

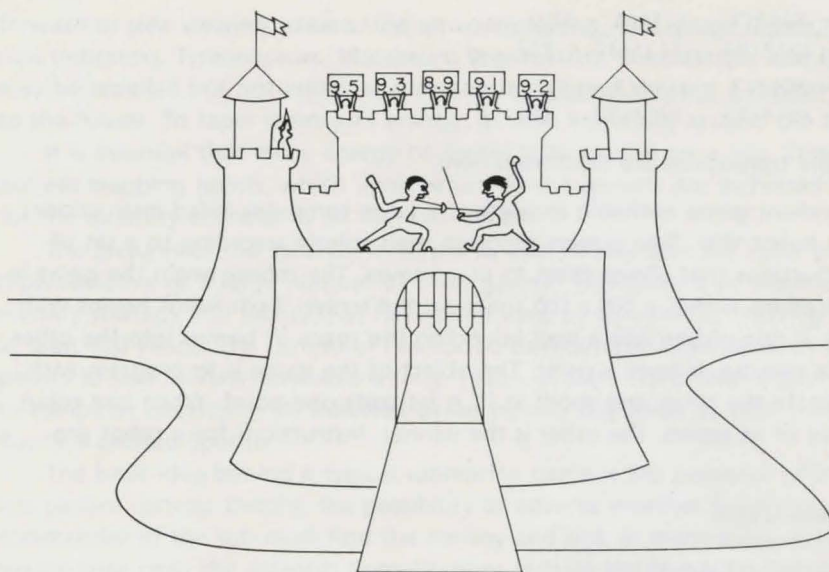
A fencer or swordsman stood on the balcony. Another appeared in the right-hand tower and jumped to the balcony. A damsel walked out of the door in the left-hand tower as the fencers approached each other. They then randomly lunged and parried until one of them was run through, whereupon the winner dragged the body to the side of the balcony and whistled (using the Control-G mentioned in Chapter 5). A monster rose from the moat and the winner threw the body of the loser to the monster as the damsel cried copious tears.

Watchers on the parapet held up score cards containing randomly selected numbers between 6.0 and 9.9 as if they were judging a diving contest. The damsel then turned and went back into the tower as the winner walked to the starting position at the left-hand side of the balcony, whereupon a new challenger appeared in the right-hand tower and the sequence was repeated.

Shooting gallery games are easy to do and fun to play. Using left and right arrow keys (discussed in Chapter 7), the player could move a gun along the bottom of the screen to aim at objects that progress across the screen. The number of bullets allotted to the player should be displayed and a rare, small target should allow the player to accrue extra bullets should he hit it.



Scene from a "Fencing Fools" animation



A very easy Get Those Aliens game could be done on any home computer with a reasonable graphics capability. Think of the screen as a view from the front of a spacecraft. A fixed crosshair sight would be imposed on the screen to enable the pilot to control the spacecraft and shoot at alien ships. If the alien were stationary, he would seem to move to the left as the pilot moved his spacecraft to the right (See Chapter 8).

If, in addition, the alien were moving, the game would be made more difficult. Using two random numbers  $Y_1$  and  $Y_2$  and two known values  $X_1$  and  $X_2$ , where  $X_1$  and  $X_2$  are opposite edges of the screen and  $Y_1$  and  $Y_2$  are random values between the bottom and top of the screen, an alien could be made to move along a random straight line across the screen by means of a loop that progresses from  $X_1$  to  $X_2$  in small increments.

Motion of the craft would complicate the trajectory, giving the alien a good chance to escape the boundaries of the screen before he could be shot down. The number of hits, misses, and escapes should be displayed.

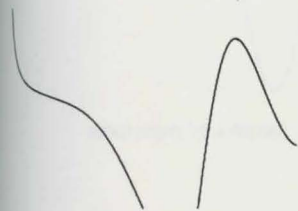
The alien's trajectory could be complicated by establishing a random curve, such as shown here, for the alien to follow. This type of curve requires four pairs of random numbers  $X_0, Y_0$ ;  $X_1, Y_1$ ;  $X_2, Y_2$ ; and  $X_3, Y_3$ .

The  $X$  and  $Y$  numbers would be any values that are within the screen boundaries, but the  $X$  values must be sorted into ascending sequence. The equation would then be evaluated from  $X_0$  to  $X_3$  or from  $X_3$  to  $X_0$ , the choice being made randomly, letting  $X$  begin at one end ( $X_0$  or  $X_3$ ) and progress to the other in small increments to find  $Y$  from the equation:

$$Y = A*Y_0 + B*Y_1 + C*Y_2 + D*Y_3$$

$$\text{where } A = \frac{(X - X_1)*(X - X_2)*(X - X_3)}{(X_0 - X_1)*(X_0 - X_2)*(X_0 - X_3)}$$

$$B = \frac{(X - X_0)*(X - X_2)*(X - X_3)}{(X_1 - X_0)*(X_1 - X_2)*(X_1 - X_3)}$$



Two samples of a random curve to be used to complicate a trajectory



$$C = \frac{(X - X_0)(X - X_1)(X - X_3)}{(X_2 - X_0)(X_2 - X_1)(X_2 - X_3)}$$

$$D = \frac{(X - X_0)(X - X_1)(X - X_2)}{(X_3 - X_0)(X_3 - X_1)(X_3 - X_2)}$$

Some sample trajectories are illustrated here.

An excellent game available to users of a large computer-aided instructional network is Robot War. Two players program their robots according to a set of simple instructions that allows them to plan moves. The robots begin the game in random locations within a 100 x 100 space on the screen. Each robot begins with five points. If one moves into a wall bounding the space or bumps into the other robot while moving, it loses a point. The object of the game is to program each robot to locate the other and shoot at it. A hit costs one point. When one robot loses all five of its points, the other is the winner. Instructions for a robot program include

```

Read my X-location
Read my Y-location
Read opponent's X-location
Read opponent's Y-location
Shoot at a given angle
Start moving 1,2, or 3 units in +X or +Y
Stop moving
Clear accumulator
Add to accumulator
Subtract from accumulator
Multiply accumulator
Store accumulator
Set constant value
Branch on zero, negative, or positive accumulator
Unconditional branch

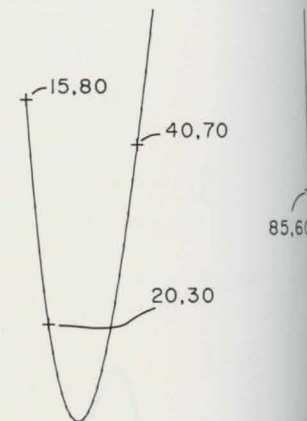
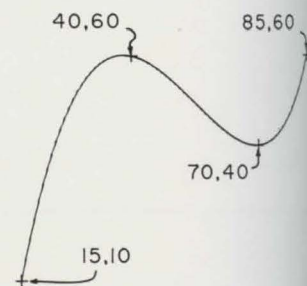
```

Robots alternate turns, processing one instruction at a time. A START MOVING instruction lets the robot continue moving while processing other instructions. Of course, by the time a robot has figured out the position of the other robot (which is difficult since there is no DIVIDE instruction), its opponent has probably moved.

This is a passive game in the sense that players do not control the game as it is being played. Its appeal is based on the ability of the player to predict a good strategy. There is also a frustration component, because it is always possible to design a robot that will defeat an opponent whose strategy is known from experience.

An interesting game that uses a form of maze exploration is Time Explorer, in which the player is the captain of a time-travelling capsule. The task before the player is to go back in time in search of fuel resources that were once on the surface of the earth but have since been buried by geologic phenomena. It is necessary that these resources be sought out because the earth has run out of low-cost energy.

The capsule must be guided through deserts, swamps, rivers, forests, mountains, and caves. Each environment contains various grades of fuel, as well as prehistoric creatures that attack the capsule and can damage sensors, such as the



Samples of trajectories

forward or side viewing screens, the air conditioning, fuel gauge, lights, or location indicators. Tyrannosaurs, Mosasaurs, Brontosaurus, Pterodactyls, and Hominids may be repelled but not killed, as a precaution against making a radical change to the future. To repel them uses energy, as does travelling around the terrain.

It is essential that more energy be found than is used on a trip. Travelling accrues mapping points, which upon return to the present are increased by points for the quantity and quality of fuel collected.

The pictures of the submarine, airplane, and convoy (see the color plates) are representative of a large number of "war games" (simulations of situations in military strategy and tactics that have been used by generals as training aids for at least 100 years). The computer has added considerable sophistication and complexity to real military simulations since 1950—in fact, World War II gave impetus to computer development, followed by an equally big boost in 1957 when the Russians orbited Sputnik.

The basic idea behind a typical submarine game is the presence of a more or less passive convoy. Despite the possibility of adverse weather conditions, the commander of the sub must find the convoy and sink as many ships as he can before they cross the Atlantic. Complications include the arrival of RAF Coastal Command seaplanes and Royal Navy destroyers that can sink the sub (British forces are used because the scenario is designed for 1940, when the U.S. was sending supplies but was not yet a participant.)

The commander has at his disposal 40 torpedoes and 100 rounds of ammunition for his deck gun. Clearly the submarine cannot use the deck gun while submerged but is more vulnerable to attack while surfaced. The seaplanes can attack a surfaced submarine, but destroyers using depth charges can locate and destroy a submerged sub by means of sonar and depth charges. Sonar beeps warn the commander of approaching destroyers. The submarine can reach friendly ports in Europe, Africa, and South America to repair damage, refuel, and take on ammunition.

Most games usually played on a board do not lend themselves to play between the computer and only one other player, because a computer can be programmed to win such games with ease. In 1957, an article appeared in the computer literature about a computer program that was programmed to play checkers according to the rules and to learn from experience.

The first game between the computer and a human was won, of course, by the human, because the computer had no strategy for winning. But each move was recorded by the computer by saving the position of the pieces on the board. When the computer lost the game, those moves were saved as losing moves and in subsequent games the computer would not repeat them unless forced.

After fourteen games, the human could no longer beat the computer. The same outcome can be expected in countless other games such as Tic-Tac-Toe, 3-D Tic-Tac-Toe, Othello™, Chinese Checkers, and so forth.

Chess is a notable exception, largely because it is too complex for computers to develop and evaluate a complete decision tree in the three minutes allowed between moves in tournament play. A reasonable start to a chess program is a chess problem solver. In the chess column of the Sunday newspaper there is usu-



ally a problem of the "white to move and mate in two" type. Such a problem can be solved by a computer by trying all possible combinations of two moves.

In the near future it is probable that manufacturers will develop a game computer with two screens back to back and two sets of controls. Games using a dual-screen machine will be much more interesting than one-screen games because it will be possible for the two players to be presented with different information. The Submarine Versus Convoy game, in which one player controls a submarine and the other controls aircraft and destroyers, would give the sub commander his view of the situation through a periscope while the surface commander would have a view from the bridge of his destroyer.

Most competitive games can be improved greatly by a two-screen computer or by connecting two computers so they can communicate with each other. Games requiring coordination would be much faster because the players no longer would need to take turns but could move simultaneously. A typical two-screen game would be an aerial dogfight, where each player sees a view on his screen as if he were looking out of his airplane cockpit. It might be the role of one player to locate and bomb targets on the ground while the other tries to intercept the bomber and shoot it down.

Given two computers connected over some distance by telephone lines, the concept of board games becomes more attractive. In this case, the computer can referee the game while displaying for each player his perspective view of the board. Currently, time-sharing systems with multiple terminals are available for commercial and school use. Such networks are open to the home via telephone lines but are too expensive for most personal computer users. As costs come down, individuals will be able to enjoy playing games against several, even unknown, opponents.

Further possibilities for computer games will depend to some extent on the directions taken by the designers of home computers. It is likely that improvements will be made in screen resolution and available colors, thus allowing for more attractive and detailed display. Improved sound and voice output is also on the way, along with larger memory. Additional storage would permit more sophistication in programs, including enhancements in artificial intelligence capability.

The addition of peripherals that can sense human states will open another whole realm of possibilities. The inputs into a polygraph, or lie detector, could be used by a computerized fortune teller or "psychiatrist." Voice input and complex sound output will add realism to most military simulations.

Still, the best computer peripheral is the human imagination. You can influence the direction of new computer games by applying your interests and skills to brand-new ideas for computer-based entertainment and by studying existing video and arcade games to evaluate their appeal. The field is very new; as computer capability and the base of hobbyist programmers continues to grow, better and better computer programs are the inevitable result.



## Appendix

## The Display Screen

A computer display screen is described in terms of its "resolution"—the number of discrete points (called picture elements or "pixels") that can be displayed. These points are arranged in a rectangular array that can be thought of in terms of rows and columns in the way a letter on a typewritten page appears on a certain line in a fixed position on the line. To describe how to find the letter on the page, it could be said that it is the seventh letter from the left on the eleventh line from the top—a pair of numbers could be used to describe its unique position.

Pixels on a screen are also described this way. The "address" or location of a pixel is given in terms of an X,Y pair, where X defines the pixel's horizontal location and Y its vertical. Manipulation of the drawings on a screen, therefore, is done by thinking of the screen as a manifestation of one of the coordinate systems used in analytic geometry. The best known of these is the rectangular, or Cartesian coordinate system, named after the French mathematician Rene Descartes.

## Cartesian Coordinates

In Cartesian coordinates, a point X,Y is located according to a pair of ruled axes, as shown in the illustration.

It is clear that a line can be described uniquely by two points X1,Y1 and X2,Y2. In geometry, a line is thought of as extending infinitely, but in computing it is more useful to think of the points X1,Y1 and X2,Y2 as the ends of a line segment.

There are several ways to express the equation of a line, including the two-point form:

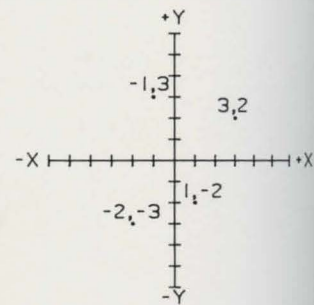
$$\frac{Y - Y_1}{Y_2 - Y_1} = \frac{X - X_1}{X_2 - X_1}$$

or  $Y = \frac{X - X_1}{X_2 - X_1}(Y_2 - Y_1) + Y_1$

Given a value X, its corresponding Y-value can be found by solving the right-hand side of the equation. Given a vertical line, X2 is equal to X1, so  $X_2 - X_1 = 0$ . Since the expression above requires division by  $X_2 - X_1$ , this equation is not useful for vertical lines. In this case it is only possible to find X in terms of Y:

$$X = \frac{Y - Y_1}{Y_2 - Y_1}(X_2 - X_1) + X_1$$

which will not work for horizontal lines.



Cartesian coordinates on a pair of ruled axes

## Circles

A circle drawn in Cartesian coordinates, like the one we see here, has the equation  $X^2 + Y^2 = R^2$  if its center is at the point 0,0, the "origin."

To find the X-value of a point on the circle given a Y-value,

$$X^2 = R^2 - Y^2$$

or  $X = \sqrt{R^2 - Y^2}$

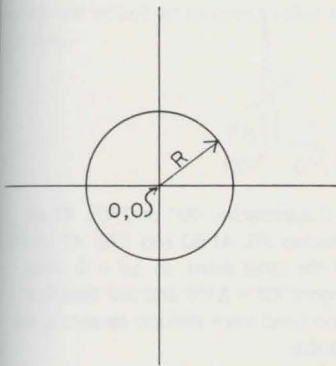
It is clear that for a given  $Y$  there are two values of  $X$ , namely,  $\sqrt{R^2 - Y^2}$  and  $-\sqrt{R^2 - Y^2}$ . Furthermore, if  $Y$  is greater than  $R$ , the expression  $R^2 - Y^2$  is less than zero. Square roots of negative numbers are undefined, so obviously the  $X$ -values of a circle exist only for  $Y$ -values in the range  $-R \leq Y \leq R$ .

If the circle's center is not at the origin but instead is at the point  $X_0, Y_0$ , the equation becomes

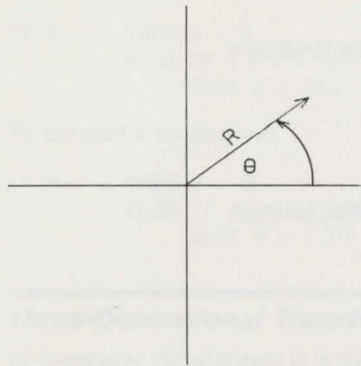
$$(X - X_0)^2 + (Y - Y_0)^2 = R^2$$

or  $X = X_0 \pm \sqrt{R^2 - (Y - Y_0)^2}$

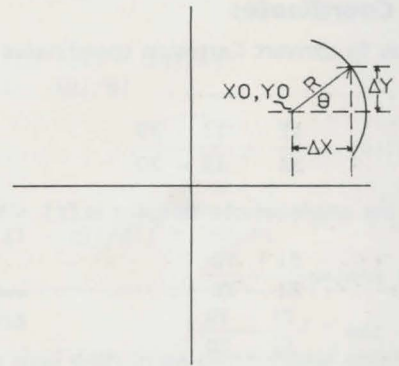
It is not natural to think of drawing a circle from top to bottom by evaluating the above equation. Using a compass, a circle is drawn by setting the point at the center, opening the compass to the radius  $R$  and sweeping it 360 degrees. This suggests the polar coordinate system, shown in the drawing, which is described in terms of  $R$  and  $\theta$ , a radius and an angle, rather than  $X$  and  $Y$ .



Drawing a circle



Polar coordinate method of determining points on a circle



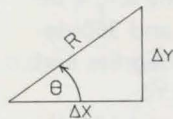
Describing a point on a circle in terms of Cartesian coordinates

A point on a circle at center  $X_0, Y_0$  can be described uniquely by a given  $R$  and  $\theta$  and can be transformed to Cartesian coordinates, as illustrated here.

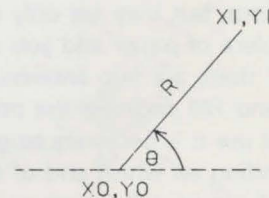
The equations are derived from the right-triangle relationships shown in the drawing of a right triangle, from which it is clear that  $\Delta Y = R \sin(\theta)$  and  $\Delta X = R \cos(\theta)$ , so the point where the line  $R$  touches the circumference of the circle is  $X_0 + \Delta X$ ,  $Y_0 + \Delta Y$  or, as illustrated below,

$$X_1 = X_0 + R \cos(\theta)$$

$$Y_1 = Y_0 + R \sin(\theta)$$



Relationships of sine, cosine, and tangents in a right triangle



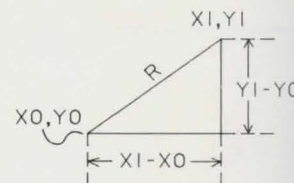
$X_1$  and  $Y_1$  can be found from  $X_0$ ,  $Y_0$ ,  $R$  and  $\theta$   $X_1 = X_0 + R \cos(\theta)$   $Y_1 = Y_0 + R \sin(\theta)$



Consider the equation of the line R having the endpoints  $X_0, Y_0$  and  $X_1, Y_1$ , illustrated below.

$$\Delta X = X_1 - X_0$$

and  $\Delta Y = Y_1 - Y_0$



The relationship between polar and Cartesian representation is seen here.  $\Delta X = X_1 - X_0$ ,  $\Delta Y = Y_1 - Y_0$  in Cartesian coordinates,  $X_1 = X_0 + \Delta X$ ,  $Y_1 = Y_0 + \Delta Y$  in polar coordinates  $X_1 = X_0 + R \cdot \cos(\theta)$ ,  $Y_1 = Y_0 + R \cdot \sin(\theta)$  so  $\Delta X = R \cdot \cos(\theta)$  and  $\Delta Y = R \cdot \sin(\theta)$

## Polar Coordinates

It is easy to convert Cartesian coordinates to polar by finding  $\theta$ :

$$\tan(\theta) = \frac{\Delta Y}{\Delta X} = \frac{Y_1 - Y_0}{X_1 - X_0}$$

so  $\theta$  is the angle whose tangent is  $(Y_1 - Y_0)/(X_1 - X_0)$ , written

$$\theta = \arctan \frac{Y_1 - Y_0}{X_1 - X_0}$$

or  $\theta = \tan^{-1} \frac{Y_1 - Y_0}{X_1 - X_0}$

As before, the arctangent function is not defined where  $X_1 = X_0$ , that is, at 90 degrees or 270 degrees, where as  $\theta$  approaches the angle,  $X_1$  approaches  $X_0$ , so that  $X_1 - X_0$  is zero at these points, illustrated below.

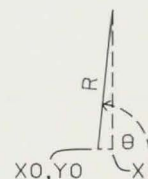
The arcsine and arccosine functions are defined for all values of  $\theta$ , because R is always greater than zero.

$$\theta = \arccos \frac{X_1 - X_0}{R}$$

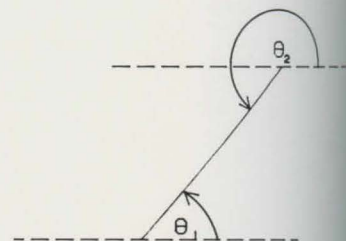
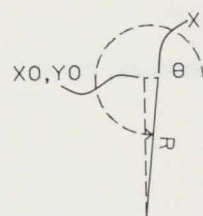
$$\theta = \arcsin \frac{Y_1 - Y_0}{R}$$

Most personal computers include the ACS (arccosine), ASN (arcsine), and ATN (arctangent) functions, but they are only defined for a "principal angle." If you draw a line on a piece of paper and you ask, "Relative to the horizontal, at what angle is this line?" there are two answers, as demonstrated here: one angle between 0 degrees and 180 degrees; the other between 180 degrees and 360 degrees. For practical use it is necessary to get an angle  $\theta$  between 0 degrees and 360 degrees depending on which end of the line is at the point  $X_0, Y_0$ .

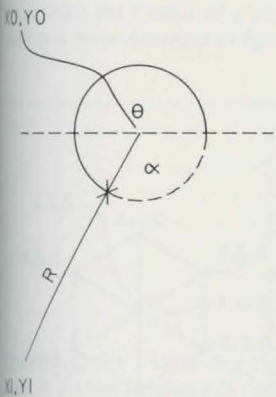
A typical ACS function will return an angle between 0 degrees and 180 degrees, an ASN between -90 degrees and 90 degrees, and an ATN between -45



As  $\theta$  approaches 90° or 270°,  $X_1$  approaches  $X_0$ . At 90 and 270,  $X_1$  and  $X_0$  are the same point, so  $\Delta X = 0$ . Since  $\tan(\theta) = \Delta Y / \Delta X$  and are therefore undefined since division by zero is not possible



Two answers to the question, "At what angle is this line relative to the horizontal?"



Whenever  $Y1 < Y0$ , that is,  $Y1$  is "below"  $Y0$ , then  $0 \leq \theta < 180$  the arccosine routine will return an incorrect value in this case.

degrees and 45 degrees, usually in radians, not degrees. (It is easy to convert from radians to degrees—there are  $2\pi$  radians in 360 degrees, so to convert, degrees = radians $\times 180/\pi$  where  $\pi = 3.14159\dots$ )

Something must be done about the principal angle phenomenon. If the ACS function returns an angle between 0 degrees and 180 degrees, the principal angle is correct unless  $Y1 < Y0$ .

When  $Y1 < Y0$ , for example as illustrated below, the angle returned by the ACS function will be  $\alpha$ , between 0 degrees and 180 degrees such that  $\theta = (360 - \alpha)$ .

To get  $\theta$  between 0 and 360 degrees, the algorithm is

```
IF Y1 >= Y0 THEN  $\theta = \text{ACS}((X1 - X0)/R)$ 
ELSE  $\theta = 2\text{PI} - \text{ACS}((X1 - X0)/R)$ 
```

which yields the value  $\theta$  in radians. To be perfectly safe, it may be advisable to avoid the division error if  $R = 0$  by using the algorithm

```
IF R = 0 THEN  $\theta = R$ 
ELSE IF Y1 >= Y0 THEN  $\theta = \text{ACS}((X1 - X0)/R)$ 
ELSE  $\theta = 2\text{PI} - \text{ACS}((X1 - X0)/R)$ 
```

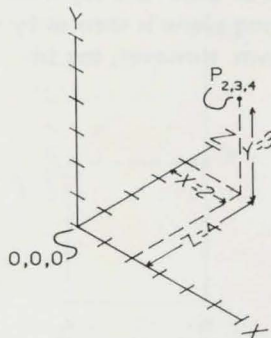
To convert  $\theta$  to degrees,

```
IF R = 0 THEN  $\theta = R$ 
ELSE IF Y1 >= Y0 THEN  $\theta = \text{ACS}((X1 - X0)/R) * 180/\text{PI}$ 
ELSE  $\theta = (2\text{PI} - \text{ACS}((X1 - X0)/R)) * 180/\text{PI}$ 
```

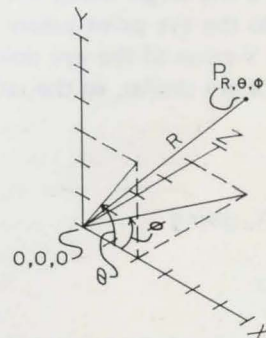
### Three-Dimensional Transformations

In computer simulations it is necessary to deal with three-dimensional objects and to project them onto a two-dimensional screen in a realistic way. A point may be represented in a three-dimensional Cartesian coordinate system by three numbers  $X$ ,  $Y$ , and  $Z$ , representing the point's position in space relative to the three axes. An example is given here, where  $P = 2,3,4$ .

In spherical coordinates, this point can be represented by a radius  $R$  and two angles  $\theta$  and  $\phi$ , as shown.



A point represented in the three-dimensional Cartesian coordinate system



Three-dimensional spherical coordinate system



The same point can be represented in cylindrical coordinates by the polar values in the X-Y plane  $R$  and  $\theta$  and the Z-value.

### Perspective

Eight points are needed to define a cube in three-dimensional space. A cube having a side of length two units, situated two units behind the X-Y plane consists of the following points:

X	Y	Z
0	0	2
2	0	2
2	2	2
0	2	2
0	0	4
2	0	4
2	2	4
0	2	4

Imagine a cube made out of wire. A piece of string can be attached to each point, or node, and stretched so that the strings are parallel to each other. If they are passed through holes in a piece of cardboard, the holes would represent a projection of the 3-D cube onto the 2-D board. Then if the appropriate holes are connected with pencil lines, an identifiable 2-D representation of a cube would appear. This is a parallel projection, which unlike a real-world view does not take into account that objects appear smaller as they get farther away from the viewer.

To make a perspective view of the cube, the strings must come together at a point called the "viewing point" or "eye point." It is necessary to find where each of the projection lines passes through the viewing plane, in this case the display screen of the computer.

Refer to the pair of drawings illustrating the comparison of parallel projection with perspective projection.

And then study the pair of diagrams illustrating the process of determining the values of X and Y, projected points on a screen.

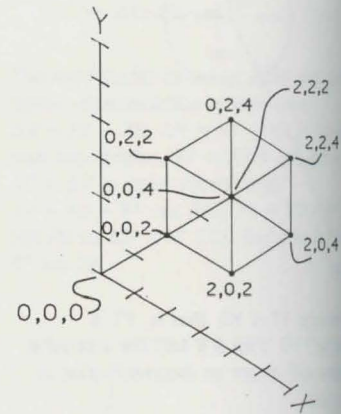
Looking toward the origin along the X-axis, the height at which the ray from the point  $P(x,y,z)$  to the eye point passes through the viewing plane is seen as  $E_y + Y$ , where  $E_y$  is the Y-value of the eye point and  $Y$  is unknown. However, the triangles EPA and ECB are similar, so the ratio

$$\frac{Ez}{Y} = \frac{Ez + Z_p}{Y_p - E_y}$$

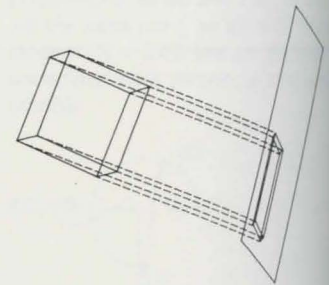
can be solved for  $Y$ , giving

$$Y = \frac{Y_p - E_y}{Ez + Z_p} * Ez$$

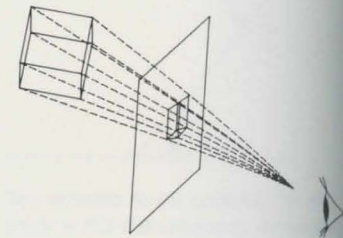
So the Y-coordinate of the projected point is  $Y + E_y$ . To find the X-coordinate, we can look down the Y-axis toward the origin and see where



Representing a cube in a three-dimensional coordinate system



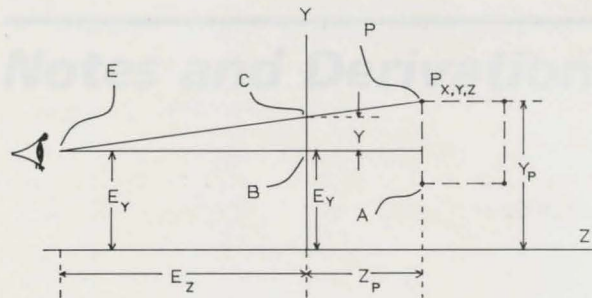
Parallel projection



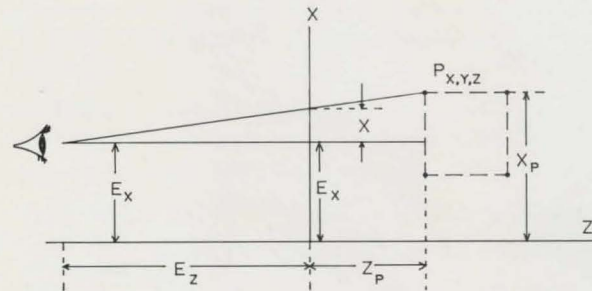
Perspective projection



Determining the Y-value of a projected point in a three-dimensional figure



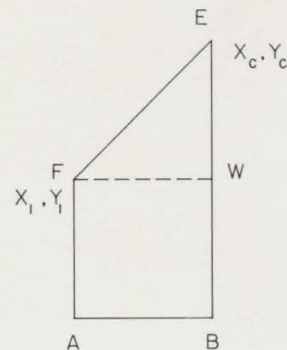
Determining the X-value of a projected point in a three-dimensional figure



$$X = \frac{X_p - E_z}{E_z + Z_p} * E_z$$

so the X-coordinate of the projected point on the screen is  $X + E_x$

In both the above diagrams, the values  $E_x$ ,  $E_y$ , and  $E_z$  are the respective X, Y, and Z values of the eye point and  $X_p$ ,  $Y_p$ , and  $Z_p$  are the X, Y, and Z values of the point  $P(x,y,z)$ . Performing this transformation on every point in a three-dimensional figure will yield a perspective view of the figure on the screen.



A trapezoid



## Notes and Derivations



### 1. From Chapter 2, Software Tools, Arcs Section

If logical variables are used, ellipses and circles may be drawn using a shorter subroutine:

ELLIPSE REVISED PROGRAM CODE	Sequence Number
I = A1	350
MOVE X0 + M1*COS(I), Y0 + M2*SIN(I)	360
REPEAT	370
I = I + (A1 < A2) - (A1 > A2)	380
IF (I > A2 AND A1 < A2) OR (I < A2 AND A1 > A2) THEN	390
I = A2	400
;	410
DRAW X0 + M1*COS(I), Y0 + M2*SIN(I)	420
UNTIL I = A2	430
ELLIPSE REVISED CODE DESCRIPTION	Sequence Number
Set I equal to A1.	350
Move to the beginning of the elliptical arc.	360
Add 1 to I if A1 is less than A2 and subtract 1 from I if A1 is greater than A2.	380
If I has exceeded A2 from either direction, set I equal to A2.	390
Draw an arc increment.	420
If I is not equal to A2, continue at line 370.	430

### 2. From Chapter 5, Ballistic Trajectory Games, Cannon Shooting Game Section

After rotation through the angle  $\alpha$ , the original point  $X', Y'$  becomes  $X, Y$  according to the rotation formulas:

$$X = X' * \cos(\alpha) - Y' * \sin(\alpha)$$

$$Y = X' * \sin(\alpha) + Y' * \cos(\alpha)$$

The coordinates of the center of the cannon mouth are 12,3 or 5.5,1 after subtracting the center of rotation.

$$\text{so } X = 5.5 * \cos(\alpha) - 1 * \sin(\alpha)$$

$$Y = 5.5 * \sin(\alpha) + 1 * \cos(\alpha)$$

To find the resulting point, the center of rotation must be added, giving

$$X0 = 6.5 + 5.5 * \cos(\alpha) - \sin(\alpha)$$

$$H0 = 2.0 + 5.5 * \sin(\alpha) + \cos(\alpha)$$

2

**Trace Program Code**

line if logical variables are  
 $P(I)*F + (F = 0))$ , the horse's  
 om number between 0 and  
 0) is false, giving it a value  
 ne horse. If  $F = 0$ , the  
 ply. The value of  
 ngular increment is

### Solve A Maze

space possible when logical  
programs where similar  
80 can be replaced by:

HEN

1220

1230

1240

1250

1260

1270

1280

1290

2)

\*(D=8)

1300

1310

1320

1330

1340

= 14)

1350

1360

### 6. From Chapter 7, Maze And Fantasy Games, The Maze Runner's View Of The Maze Section

To find  $X_2$ , we have to know the equation of the line through  $0, Y_0$  and  $X_1, Y_1$ . Given two points  $x_1, y_1$  and  $x_2, y_2$  the equation of the line through them is

$$\frac{Y - y_1}{y_2 - y_1} = \frac{X - x_1}{x_2 - x_1}$$

So the equation of the line through  $0, Y_0$  and  $X_1, Y_1$  is

$$\frac{Y - Y_0}{Y_1 - Y_0} = \frac{X - 0}{X_1 - 0}$$

Substituting  $Y_2$  for  $Y$  will give  $X_2$ :

$$\frac{Y_2 - Y_0}{Y_1 - Y_0} = \frac{X_2}{X_1} \quad \text{or} \quad X_2 = \frac{Y_2 - Y_0}{Y_1 - Y_0} * X_1$$

### 7. From Chapter 7, Maze And Fantasy Games, The Maze Runner's View Of The Maze Section

It is first necessary to obtain the equations of the two lines:

$$\frac{X + X_1}{X_1} = \frac{Y - Y_1}{Y_2 - Y_1}$$

$$\text{and} \quad \frac{X - X_1}{-X_1} = \frac{Y - Y_1}{Y_0 - Y_1}$$

Isolating  $X$  from each equation,

$$X = \frac{Y - Y_1}{Y_2 - Y_1} * X_1 - X_1$$

$$\text{and} \quad X = \frac{Y - Y_1}{Y_0 - Y_1} * (-X_1) + X_1$$

Now, since the left sides of these equations are equal to each other, so are the right sides. They may be combined into one equation with  $Y$  as the unknown. The point at which they are equal is of course the point at which the two lines intersect, so  $Y$  is in fact  $Y_3$ . After multiplying and separating terms,

$$\frac{X_1 * Y_3}{Y_2 - Y_1} - \frac{X_1 * Y_1}{Y_2 - Y_1} - X_1 = \frac{-X_1 * Y_3}{Y_0 - Y_1} + \frac{X_1 * Y_1}{Y_0 - Y_1} + X_1$$

Isolating  $Y_3$ ,

$$Y_3 * \frac{X_1}{Y_2 - Y_1} + \frac{X_1}{Y_0 - Y_1} = \frac{X_1 * Y_1}{Y_0 - Y_1} + 2 * X_1 + \frac{X_1 * Y_1}{Y_2 - Y_1}$$

and dividing,

$$Y_3 = \frac{2 * X_1 + X_1 * Y_1 * (1/Y_0 - 1/Y_1) + 1/(Y_2 - Y_1)}{X_1 * (1/(Y_2 - Y_1) + 1/(Y_0 - Y_1))}$$



$$\text{or } Y3 = \frac{2}{(1/(Y2 - Y1) + 1/(Y0 - Y1))} + Y1$$

$$\text{or } Y3 = 2*((Y2 - Y1) + (Y0 - Y1)) + Y1$$

$$\text{and } X3 = \frac{Y3 - Y1}{Y2 - Y1} * X1 - X1$$

### 8. From Chapter 8, Outer Space Games, Applying Gravity Section

The tangent of the angle  $\theta$  is  $(Yn - Y0)/(Xn - X0)$ . Most BASIC interpreters have an arctangent function ATN, but unfortunately these functions typically return a principal value, that is, a number between  $-45$  degrees and  $+45$  degrees, or sometimes between  $>-90$  degrees and  $<+90$  degrees. An angle between  $0$  degrees and  $360$  degrees, or between  $0$  and  $2\pi$  radians, is needed, defined for all values of  $X0$ ,  $Y0$ ,  $Xn$ , and  $Yn$ . The arccosine function is so defined, but most ACS routines return a value between  $0$  degrees and  $180$  degrees: that is, whenever  $Yn \geq Y0$  the principal angle is correct, but for  $Yn < Y0$  the angle is actually  $360$  degrees minus the ACS value.

To get the proper angle of the vector, then, the equation is

$$\theta = (\text{ACS}(\frac{Xn - X0}{r})) * ((Yn - Y0) \geq 0) + (360 - \text{ACS}(\frac{Xn - X0}{r})) * ((Yn - Y0) < 0)$$

$$\text{where } r = \text{SQR}((Xn - X0)^2 + (Yn - Y0)^2)$$

### 9. From Chapter 10, Computer-Aided Design, Deletions, Section

To find the length of  $P$  it is useful to remember that the area of a triangle is one half the base times the height. The base is the line  $FD$  and the height the line  $EQ$  whose length  $P$  is to be determined. The area of the triangle  $DEF$  can be calculated by adding the areas of the trapezoids  $ABEF$  and  $BCDE$  and subtracting the area of the trapezoid  $ACDF$ . Each trapezoid in this problem consists of a rectangle with a triangle on top: for example, the trapezoid  $ABEF$ , shown here. The line  $AB$  is  $Xc - X1$  in length,  $AF = Y1$  and  $WE = Yc - Y1$ , so its area is

$$(Yc - Y1)*(Xc - X1)/2 + (Xc - X1)*Y1$$

$$\text{or } (Xc - X1)*(Yc + Y1)/2$$

The area of  $BCDE$  is  $(X2 - Xc)*(Y2 + Yc)/2$  and of  $ACDF$ ,  $(X2 - X1)*(Y2 + Y1)/2$  so the area of  $DEF$  is

$$(Xc - X1)*(Yc + Y1)/2 + (X2 - Xc)*(Y2 + Yc)/2 - (X2 - X1)*(Y2 + Y1)/2$$

$$\text{or } ((Xc - X1)*(Yc + Y1) + (X2 - Xc)*(Y2 + Yc) - (X2 - X1)*(Y2 + Y1))/2$$

The length of the base of the triangle  $DEF$  is  $\text{SQR}((X2 - X1)^2 + (Y2 - Y1)^2)$ , so

$$P = \frac{((Xc - X1)*(Yc + Y1) + (X2 - Xc)*(Y2 + Yc) - (X2 - X1)*(Y2 + Y1))}{\text{SQR}((X2 - X1)^2 + (Y2 - Y1)^2)}$$

The coordinates  $X_q, Y_q$  of the point Q may be found by solving simultaneously the equations of the lines FD and EQ. For the line FD,

$$\frac{X_q - X_1}{X_2 - X_1} = \frac{Y_q - Y_1}{Y_2 - Y_1} \quad \text{or} \quad Y_q = \frac{X_q - X_1}{X_2 - X_1} (Y_2 - Y_1) + Y_1$$

Since EQ is perpendicular to FD, the slope of EQ is  $-1/M$  if  $M$  is the slope of FD. Rewriting the equation of FD into the form

$$\begin{aligned} Y_q &= \frac{X_q - X_1}{X_2 - X_1} (Y_2 - Y_1) + Y_1 \\ &= X_q \frac{Y_2 - Y_1}{X_2 - X_1} - X_1 \frac{Y_2 - Y_1}{X_2 - X_1} + Y_1 \end{aligned}$$

The slope  $M$  of FD is  $(Y_2 - Y_1)/(X_2 - X_1)$ , so the slope of EQ is  $(X_1 - X_2)/(Y_2 - Y_1)$  and the equation of EQ is

$$Y_q - Y_1 = \frac{X_1 - X_2}{Y_2 - Y_1} (X_q - X_c)$$

$$\text{or} \quad Y_q = \frac{X_1 - X_2}{Y_2 - Y_1} (X_q - X_c) + Y_1$$

so  $X_q$  may be found from the equation

$$\frac{X_1 - X_2}{Y_2 - Y_1} (X_q - X_c) + Y_c = \frac{X_q - X_1}{X_2 - X_1} (Y_2 - Y_1) + Y_1$$

to be

$$X_q = \frac{X_1 \frac{Y_2 - Y_1}{X_2 - X_1} + X_c \frac{X_2 - X_1}{Y_2 - Y_1} + Y_c - Y_1}{\frac{Y_2 - Y_1}{X_2 - X_1} + \frac{X_2 - X_1}{Y_2 - Y_1}}$$

and  $Y_q$  may be found from the equation of EQ by substituting the now-known value of  $X_q$ .

---

## Font Tables

### Stick

REM	500
DATA " ",3,4	510
DATA -1,"A",3,4	520
DATA 1,3,0,0,3,8,6,0	530
DATA 1,2,1,2.333,5,2.333	540
DATA -1,"B",3,4	550
DATA 1,11,0,4,4,4,6,3,6,1,4,0,0,0,0,8,4,8,6,7,6,5,4,4	560
DATA -1,"C",3,4	570
DATA 1,8,6,6,4,8,2,8,0,6,0,2,2,0,4,0,6,2	580
DATA -1,"D",3,4	590
DATA 1,7,0,0,4,0,6,2,6,6,4,8,0,8,0,0	600
DATA -1,"E",3,4	610
DATA 1,4,6,0,0,0,0,8,6,8	620
DATA 1,2,0,4,4,4	630
DATA -1,"F",3,4	640
DATA 1,3,0,0,0,8,6,8	650
DATA 1,2,0,4,4,4	660
DATA -1,"G",3,4	670
DATA 1,10,6,6,4,8,2,8,0,6,0,2,2,0,4,0,6,2,6,4,4,4	680
DATA -1,"H",3,4	690
DATA 1,2,0,0,0,8	700
DATA 1,2,0,4,6,4	710
DATA 1,2,6,0,6,8	720
DATA -1,"I",1.5,4	730
DATA 1,2,0,0,3,0	740
DATA 1,2,1.5,0,1.5,8	750
DATA 1,2,0,8,3,8	760
DATA -1,"J",3,4	770
DATA 1,2,0,8,6,8	780
DATA 1,6,4,8,4,2,3,0,1,0,0,2,0,4	790
DATA -1,"K",3,4	800
DATA 1,2,0,0,0,8	810
DATA 1,2,0,3,6,8	820
DATA 1,2,1.167,4,6,0	830
DATA -1,"L",3,4	840
DATA 1,3,0,8,0,0,6,0	850
DATA -1,"M",3.5,4	860
DATA 1,5,0,0,0,8,3.5,0,7,8,7,0	870
DATA -1,"N",3,4	880
DATA 1,4,0,0,0,8,6,0,6,8	890
DATA -1,"O",3,4	900
DATA 1,9,2,0,4,0,6,2,6,6,4,8,2,8,0,6,0,2,2,0	910
DATA -1,"P",3,4	920
DATA 1,7,0,0,0,8,4,8,6,7,6,5,4,4,0,4	930
DATA -1,"Q",3,4	940
DATA 1,9,2,0,4,0,6,2,6,6,4,8,2,8,0,6,0,2,2,0	950
DATA 1,2,3,2,6,0	960
DATA -1,"R",3,4	970
DATA 1,7,0,0,0,8,4,8,6,7,6,5,4,4,0,4	980
DATA 1,2,3,4,6,0	990
DATA -1,"S",3,4	1000
DATA 1,10,0,2,2,0,4,0,6,2,4,4,2,4,0,6,2,8,4,8,6,6	1010



DATA -1, "T", 3, 4	1020
DATA 1, 2, 0, 8, 6, 8	1030
DATA 1, 2, 3, 0, 3, 8	1040
DATA -1, "U", 3, 4	1050
DATA 1, 6, 0, 8, 0, 2, 2, 0, 4, 0, 6, 2, 6, 8	1060
DATA -1, "V", 3, 4	1070
DATA 1, 3, 0, 8, 3, 0, 6, 8	1080
DATA -1, "W", 4, 4	1090
DATA 1, 5, 0, 8, 2, 0, 4, 8, 6, 0, 8, 8	1100
DATA -1, "X", 3, 4	1110
DATA 1, 2, 0, 0, 6, 8	1120
DATA 1, 2, 0, 8, 6, 0	1130
DATA -1, "Y", 3, 4	1140
DATA 1, 3, 0, 8, 3, 3, 6, 8	1150
DATA 1, 2, 3, 3, 3, 0	1160
DATA -1, "Z", 3, 4	1170
DATA 1, 4, 0, 8, 6, 8, 0, 0, 6, 0	1180
DATA -1, "0", 3, 4	1190
DATA 1, 9, 2, 0, 4, 0, 6, 2, 6, 6, 4, 8, 2, 8, 0, 6, 0, 2, 2, 0	1200
DATA -1, "1", 0, 25, 4	1210
DATA 1, 3, 0, 5, 0, 0, 5, 8, 0, 7	1220
DATA -1, "2", 3, 4	1230
DATA 1, 7, 0, 6, 2, 8, 4, 8, 6, 6, 0, 2, 0, 0, 6, 0	1240
DATA -1, "3", 3, 4	1250
DATA 1, 8, 0, 8, 6, 8, 3, 5, 6, 4, 6, 2, 4, 0, 2, 0, 0, 2	1260
DATA -1, "4", 3, 4	1270
DATA 1, 6, 4, 0, 4, 8, 3, 8, 0, 3, 0, 2, 6, 2	1280
DATA -1, "5", 3, 4	1290
DATA 1, 9, 6, 8, 0, 8, 0, 5, 4, 5, 6, 4, 6, 2, 4, 0, 2, 0, 0, 2	1300
DATA -1, "6", 3, 4	1310
DATA 1, 12, 6, 6, 4, 8, 2, 8, 0, 6, 0, 2, 2, 0, 4, 0, 6, 2, 6, 3, 4, 5, 2, 5, 0, 3	1320
DATA -1, "7", 3, 4	1330
DATA 1, 5, 0, 8, 6, 8, 6, 7, 0, 2, 0, 0	1340
DATA -1, "8", 3, 4	1350
DATA 1, 12, 2, 4, 4, 4, 6, 6, 4, 8, 2, 8, 0, 6, 2, 4, 0, 2, 2, 0, 4, 0, 6, 2, 4, 4	1360
DATA -1, "9", 3, 4	1370
DATA 1, 12, 0, 2, 2, 0, 4, 0, 6, 2, 6, 6, 4, 8, 2, 8, 0, 6, 0, 5, 2, 3, 4, 3, 6, 5	1380
DATA -1, "!", 3, 4	1390
DATA 1, 2, 1, 5, 8, 1, 5, 2, 5	1400
DATA 1, 2, 1, 5, 1, 1, 5, 0	1410
DATA -1, "''", 3, 4	1420
DATA 1, 4, 2, 6, 1, 5, 8, 2, 5, 8, 2, 6	1430
DATA 1, 4, 4, 6, 3, 5, 8, 4, 5, 8, 4, 6	1440
DATA -1, "#", 3, 4	1450
DATA 1, 2, 0, 2, 5, 5, 2, 5	1460
DATA 1, 2, 4, 1, 5, 7	1470
DATA 1, 2, 6, 5, 5, 1, 5, 5	1480
DATA 1, 2, 2, 7, 1, 1	1490
DATA -1, "\$", 3, 4	1500
DATA 1, 2, 2, 0, 2, 8	1510
DATA 1, 2, 4, 8, 4, 0	1520
DATA 1, 10, 1, 2, 2, 1, 4, 1, 5, 2, 5, 4, 4, 2, 4, 1, 5, 5, 2, 7, 4, 7, 5, 6	1530
DATA -1, "%", 3, 4	1540
DATA 1, 2, 1, 0, 5, 8	1550
DATA 1, 5, 1, 7, 2, 7, 2, 6, 1, 6, 1, 7	1560
DATA 1, 5, 4, 1, 5, 1, 5, 2, 4, 2, 4, 1	1570
DATA -1, "&", 3, 4	1580

DATA 1,9,6,0,0,6,2,8,3,6,0,3,0,1,1,0,3,0,6,3	1590
DATA -1,"'",3,4	1600
DATA 1,4,3,6,2.5,8,3.5,8,3,6	1610
DATA -1,"(",3,4	1620
DATA 1,6,4,8,3,7,2,5,2,3,3,1,4,0	1630
DATA -1,")",3,4	1640
DATA 1,6,2,8,3,7,4,5,4,3,3,1,2,0	1650
DATA -1,"*",3,4	1660
DATA 1,2,1,2,5,6	1670
DATA 1,2,3,7,3,1	1680
DATA 1,2,1,6,5,2	1690
DATA -1,":",3,4	1700
DATA 1,5,3,1.5,3.5,2,3,2.5,2.5,2,3,1.5	1710
DATA 1,5,3,4.5,3.5,5,3,5.5,2.5,5,3,4.5	1720
DATA -1,"=",3,4	1730
DATA 1,2,1,3,5,3	1740
DATA 1,2,1,5,5,5	1750
DATA -1,"-",3,4	1760
DATA 1,2,1,4,5,4	1770
DATA -1,"@",3,4	1780
DATA 1,10,4,0,4,3,2,3,2,0,4,0,6,2,6,4,4,6,2,6,0,4	1790
DATA -1,"+",3,4	1800
DATA 1,2,5,4,1,4	1810
DATA 1,2,3,6,3,2	1820
DATA -1,"]",3,4	1830
DATA 1,4,2,8,4,8,4,0,2,0	1840
DATA -1,";",3,4	1850
DATA 1,5,3,4.5,3.5,5,3,5.5,2.5,5,3,4.5	1860
DATA 1,5,2.5,1,3.5,2,3,2.5,2.5,2,3,1.5	1870
DATA -1,"",3,4	1880
DATA 1,5,2.5,-0.5,3.5,0.5,3,1,2.5,0.5,3,0	1890
DATA -1,".",3,4	1900
DATA 1,5,3,0,3.5,0.5,3,1,2.5,0.5,3,0	1910
DATA -1,"?",3,4	1920
DATA 1,2,3,0,3,1	1930
DATA 1,7,3,2.5,3,4,4,4,6,6,4,8,2,8,0,6	1940
DATA -1,"{",3,4	1950
DATA 1,9,4,8,3,7,5,2,6,2,4,5,1,4,2,3,5,2,2,3,0,5,4,0	1960
DATA -1,"}",3,4	1970
DATA 1,9,2,8,3,7,5,4,6,4,4,5,5,4,4,3,5,4,2,3,0,5,2,0	1980
DATA -1,"[",3,4	1990
DATA 1,4,4,8,2,8,2,0,4,0	2000
DATA -1,"^",3,4	2010
DATA 1,4,2,8,4,6,2.5,8,2,8	2020
DATA -1,"1",3,4	2030
DATA 1,2,3,0,3,8	2040
DATA -1,"-",3,4	2050
DATA 1,2,0,0,6,0	2060
DATA -1,"~",3,4	2070
DATA 1,3,1.5,5,3,8,4.5,5	2080
DATA 1,2,3,0,3,8	2090
DATA -1,"<",3,4	2100
DATA 1,3,5,7,1,4,5,2	2110
DATA -1,">",3,4	2120
DATA 1,3,1,7,5,4,1,2	2130
DATA -1,"/",3,4	2140
DATA 1,2,1,0,5,8	2150

DATA -1,"a",3,4	2160
DATA 1,3,0,0,3,8,6,0	2170
DATA 1,2,1,2.333,5,2.333	2180
DATA -1,"b",3,4	2190
DATA 1,11,0,4,4,4,6,3,6,1,4,0,0,0,0,8,4,8,6,7,6,5,4,4	2200
DATA -1,"c",3,4	2210
DATA 1,8,6,6,4,8,2,8,0,6,0,2,2,0,4,0,6,2	2220
DATA -1,"d",3,4	2230
DATA 1,7,0,0,4,0,6,2,6,6,4,8,0,8,0,0	2240
DATA -1,"e",3,4	2250
DATA 1,4,6,0,0,0,0,8,6,8	2260
DATA 1,2,0,4,4,4	2270
DATA -1,"f",3,4	2280
DATA 1,3,0,0,0,8,6,8	2290
DATA 1,2,0,4,4,4	2300
DATA -1,"g",3,4	2310
DATA 1,10,6,6,4,8,2,8,0,6,0,2,2,0,4,0,6,2,6,4,4,4	2320
DATA -1,"h",3,4	2330
DATA 1,2,0,0,0,8	2340
DATA 1,2,0,4,6,4	2350
DATA 1,2,6,0,6,8	2360
DATA -1,"i",1.5,4	2370
DATA 1,2,0,0,3,0	2380
DATA 1,2,1.5,0,1.5,8	2390
DATA 1,2,0,8,3,8	2400
DATA -1,"j",3,4	2410
DATA 1,2,0,8,6,8	2420
DATA 1,6,4,8,4,2,3,0,1,0,0,2,0,4	2430
DATA -1,"k",3,4	2440
DATA 1,2,0,0,0,8	2450
DATA 1,2,0,3,6,8	2460
DATA 1,2,1.167,4,6,0	2470
DATA -1,"l",3,4	2480
DATA 1,3,0,8,0,0,6,0	2490
DATA -1,"m",3.5,4	2500
DATA 1,5,0,0,0,8,3.5,0,7,8,7,0	2510
DATA -1,"n",3,4	2520
DATA 1,4,0,0,0,8,6,0,6,8	2530
DATA -1,"o",3,4	2540
DATA 1,9,2,0,4,0,6,2,6,6,4,8,2,8,0,6,0,2,2,0	2550
DATA -1,"p",3,4	2560
DATA 1,7,0,0,0,8,4,8,6,7,6,5,4,4,0,4	2570
DATA -1,"q",3,4	2580
DATA 1,9,2,0,4,0,6,2,6,6,4,8,2,8,0,6,0,2,2,0	2590
DATA 1,2,3,2,6,0	2600
DATA -1,"r",3,4	2610
DATA 1,9,2,0,4,0,6,2,6,6,4,8,2,8,0,6,0,2,2,0	2620
DATA 1,2,3,4,6,0	2630
DATA -1,"s",3,4	2640
DATA 1,10,0,2,2,0,4,0,6,2,4,4,2,4,0,6,2,8,4,8,6,6	2650
DATA -1,"t",3,4	2660
DATA 1,2,0,8,6,8	2670
DATA 1,2,3,0,3,8	2680
DATA -1,"u",3,4	2690
DATA 1,6,0,8,0,2,2,0,4,0,6,2,6,8	2700
DATA -1,"v",3,4	2710
DATA 1,3,0,8,3,0,6,8	2720
DATA -1,"w",4,4	2730



DATA 1,5,0,8,2,0,4,8,6,0,8,8	2740
DATA -1,"x",3,4	2750
DATA 1,2,0,0,6,8	2760
DATA 1,2,0,8,6,0	2770
DATA -1,"y",3,4	2780
DATA 1,3,0,8,3,3,6,8	2790
DATA 1,2,3,3,3,0	2800
DATA -1,"z",3,4	2810
DATA 1,4,0,8,6,8,0,0,6,0	2820
DATA -999	2830

**Leroy**

REM	500
DATA " ",3,4	510
DATA -1,"A",3,4	520
DATA 1,3,0,0,3,8,6,0	530
DATA 1,2,1,2.333,5,2.333	540
DATA -1,"B",3,4	550
DATA 1,4,4,0,0,0,0,8,4,8	560
DATA 1,2,0,4,4,4	570
DATA 2,4,2,2,2,-90,90	580
DATA 2,4,6,2,2,-90,90	590
DATA -1,"C",3,4	600
DATA 2,4,4,4,4,56,304	610
DATA -1,"D",3,4	620
DATA 1,4,4,8,0,8,0,0,4,0	630
DATA 2,4,4,2,4,-90,90	640
DATA -1,"E",3,4	650
DATA 1,4,6,0,0,0,0,8,6,8	660
DATA 1,2,0,4,4,4	670
DATA -1,"F",3,4	680
DATA 1,3,0,0,0,8,6,8	690
DATA 1,2,0,4,4,4	700
DATA -1,"G",3,4	710
DATA 2,4,4,4,4,56,304	720
DATA 1,3,4,4,6,4,6,0	730
DATA -1,"H",3,4	740
DATA 1,2,0,0,0,8	750
DATA 1,2,0,4,6,4	760
DATA 1,2,6,0,6,8	770
DATA -1,"I",1.5,4	780
DATA 1,2,0,0,3,0	790
DATA 1,2,1.5,0,1.5,8	800
DATA 1,2,0,8,3,8	810
DATA -1,"J",3,4	820
DATA 2,2,2,2,2,180,360	830
DATA 1,2,4,2,4,8	840
DATA 1,2,0,8,6,8	850
DATA -1,"K",3,4	860
DATA 1,2,0,0,0,8	870
DATA 1,2,0,3,6,8	880
DATA 1,2,1.167,4,6,0	890
DATA -1,"L",3,4	900
DATA 1,3,0,8,0,0,6,0	910
DATA -1,"M",3.5,4	920
DATA 1,5,0,0,0,8,3.5,0,7,8,7,0	930

DATA -1, "N", 3, 4	940
DATA 1, 4, 0, 0, 0, 8, 6, 0, 6, 8	950
DATA -1, "O", 3, 4	960
DATA 2, 3, 4, 3, 4, 0, 360	970
DATA -1, "P", 3, 4	980
DATA 1, 2, 0, 4, 4, 4	990
DATA 2, 4, 6, 2, 2, -90, 90	1000
DATA 1, 3, 4, 8, 0, 8, 0, 0	1010
DATA -1, "Q", 3, 4	1020
DATA 2, 3, 4, 3, 4, 0, 360	1030
DATA 2, 4, 1, 1, 1, 0, 90	1040
DATA 2, 6, 1, 1, 1, 180, 270	1050
DATA -1, "R", 3, 4	1060
DATA 1, 2, 0, 4, 4, 4	1070
DATA 2, 4, 6, 2, 2, -90, 90	1080
DATA 1, 3, 4, 8, 0, 8, 0, 0	1090
DATA 1, 2, 4, 4, 6, 0	1100
DATA -1, "S", 3, 4	1110
DATA 2, 3, 6, 3, 2, 18, 270	1120
DATA 2, 3, 2, 3, 2, -162, 90	1130
DATA -1, "T", 3, 4	1140
DATA 1, 2, 0, 8, 6, 8	1150
DATA 1, 2, 3, 0, 3, 8	1160
DATA -1, "U", 3, 4	1170
DATA 1, 2, 0, 8, 0, 3	1180
DATA 2, 3, 3, 3, 3, 180, 360	1190
DATA 1, 2, 6, 3, 6, 8	1200
DATA -1, "V", 3, 4	1210
DATA 1, 3, 0, 8, 3, 0, 6, 8	1220
DATA -1, "W", 4, 4	1230
DATA 1, 5, 0, 8, 2, 0, 4, 8, 6, 0, 8, 8	1240
DATA -1, "X", 3, 4	1250
DATA 1, 2, 0, 0, 6, 8	1260
DATA 1, 2, 0, 8, 6, 0	1270
DATA -1, "Y", 3, 4	1280
DATA 1, 3, 0, 8, 3, 3, 6, 8	1290
DATA 1, 2, 3, 3, 3, 0	1300
DATA -1, "Z", 3, 4	1310
DATA 1, 4, 0, 8, 6, 8, 0, 0, 6, 0	1320
DATA -1, "O", 3, 4	1330
DATA 2, 3, 4, 3, 4, 0, 360	1340
DATA -1, "1", 1.5, 4	1350
DATA 1, 2, 1.5, 0, 1.5, 8	1360
DATA -1, "2", 3, 4	1370
DATA 2, 3, 6, 3, 2, -90, 180	1380
DATA 2, 3, 0, 3, 4, 90, 180	1390
DATA 1, 2, 0, 0, 6, 0	1400
DATA -1, "3", 3, 4	1410
DATA 1, 3, 0, 8, 6, 8, 3, 5	1420
DATA 2, 3, 2.5, 3, 2.5, -180, 90	1430
DATA -1, "4", 3, 4	1440
DATA 1, 6, 4, 0, 4, 8, 3, 8, 0, 3, 0, 2, 6, 2	1450
DATA -1, "5", 3, 4	1460
DATA 1, 4, 6, 8, 0, 8, 0, 5, 3, 5	1470
DATA 2, "3", 2.5, 3, 2.5, -180, 90	1480
DATA -1, "6", 3, 4	1490
DATA 2, 3, 2.5, 3, 2.5, 0, 360	1500

DATA 2,3,2.5,3,5.5,60,180	1510
DATA -1,"7",3,4	1520
DATA 1,3,0,8,6,8,6,7	1530
DATA 2,6,0,4,7,90,180	1540
DATA -1,"8",3,4	1550
DATA 2,3,2,3,2,0,360	1560
DATA 2,3,6,3,2,0,360	1570
DATA -1,"9",3,4	1580
DATA 2,3,5.5,3,5.5,240,360	1590
DATA 2,3,5.5,3,2.5,0,360	1600
DATA -1,"!",1.5,4	1610
DATA 1,2,1.5,8,1.5,2.5	1620
DATA 1,2,1.5,1,1.5,0	1630
DATA -1,"\"",3,4	1640
DATA 1,4,2,6,1.5,8,2.5,8,2,6	1650
DATA 1,4,4,6,3.5,8,4.5,8,4,6	1660
DATA -1,"#",3,4	1670
DATA 1,2,0,2.5,5,2.5	1680
DATA 1,2,4,1,5,7	1690
DATA 1,2,6,5.5,1,5.5	1700
DATA 1,2,2,7,1,1	1710
DATA -1,"\$",3,4	1720
DATA 2,3,2.5,2,1.5,-180,90	1730
DATA 2,3,5.5,2,1.5,0,270	1740
DATA 1,2,2,0,2,8	1750
DATA 1,2,4,8,4,0	1760
DATA -1,"%",3,4	1770
DATA 1,2,1,0,5,8	1780
DATA 2,2,6,1,1,0,360	1790
DATA 2,4,2,1,1,0,360	1800
DATA -1,"&",3,4	1810
DATA 2,2,6,2,2,0,360	1820
DATA 2,2,2,2,2,90,270	1830
DATA 2,2,0,4,4,0,90	1840
DATA 2,2,4,4,4,270,360	1850
DATA -1,"'",3,4	1860
DATA 1,4,3,6,2.5,8,3.5,8,3,6	1870
DATA -1,"(",3,4	1880
DATA 2,4,4,1,4,90,270	1890
DATA 2,4,4,1.5,4,90,270	1900
DATA -1,")",3,4	1910
DATA 2,2,4,1,4,-90,90	1920
DATA 2,2,4,1.5,4,-90,90	1930
DATA -1,"*",3,4	1940
DATA 1,2,1,2,5,6	1950
DATA 1,2,3,7,3,1	1960
DATA 1,2,1,6,5,2	1970
DATA -1,":",3,4	1980
DATA 2,3,2,0.5,0.5,0,360	1990
DATA 2,3,5,0.5,0.5,0,360	2000
DATA -1,"=",3,4	2010
DATA 1,2,1,3,5,3	2020
DATA 1,2,1,5,5,5	2030
DATA -1,"-",3,4	2040
DATA 1,2,1,4,5,4	2050
DATA -1,"@",3,4	2060
DATA 1,2,1,2,5,2	2070
DATA 1,2,5,4,1,4	2080



DATA 1,2,3,6,3,2	2090
DATA -1,"+",3,4	2100
DATA 1,2,5,4,1,4	2110
DATA 1,2,3,6,3,2	2120
DATA -1," ",3,4	2130
DATA 2,3,7,1,1,0,360	2140
DATA -1,";",3,4	2150
DATA 2,3,5,0.5,0.5,0,360	2160
DATA 2,3,2,0.5,1.5,270,360	2170
DATA 2,3,2,0.5,0.5,0,270	2180
DATA 1,2,3,1.5,3,0.5	2190
DATA -1,"",2,4	2200
DATA 2,2,0.5,0.5,1.5,270,360	2210
DATA 2,2,0.5,0.5,0.5,0,270	2220
DATA 1,2,2,0,2,-1	2230
DATA -1,".",2,4	2240
DATA 2,2,0.5,0.5,0.5,0,360	2250
DATA -1,"?",3,4	2260
DATA 1,2,3,0,3,1	2270
DATA 1,2,3,2.5,3,4	2280
DATA 2,3,6,3,2,-90,180	2290
DATA -1,"a",2,4	2300
DATA 2,2.5,2.5,2.5,2.5,60,300	2310
DATA 1,2,4,0,4,5	2320
DATA -1,"b",2,4	2330
DATA 2,1.5,2.5,2.5,2.5,-120,120	2340
DATA 1,2,0,0,0,8	2350
DATA -1,"c",2,4	2360
DATA 2,2.5,2.5,2.5,2.5,60,300	2370
DATA -1,"d",2,4	2380
DATA 2,2.5,2.5,2.5,2.5,60,300	2390
DATA 1,2,4,0,4,8	2400
DATA -1,"e",2,4	2410
DATA 2,2,2.5,2.5,0,300	2420
DATA 1,2,0,2.5,4,2.5	2430
DATA -1,"f",2,4	2440
DATA 2,2.5,2.2,0,180	2450
DATA 1,2,0,5,0,0	2460
DATA 1,2,0,3,2,3	2470
DATA -1,"g",2,4	2480
DATA 2,2.5,2.5,2.5,2.5,60,300	2490
DATA 1,2,4,5,4,-1	2500
DATA 2,2,-1,2,2,180,360	2510
DATA -1,"h",2,4	2520
DATA 1,2,0,8,0,0	2530
DATA 2,2,3,2,2,0,180	2540
DATA 1,2,4,0,4,3	2550
DATA -1,"i",1,4	2560
DATA 1,2,1,0,1,5	2570
DATA 1,2,1,6.5,1,6.5	2580
DATA -1,"j",2,4	2590
DATA 2,2,0,2,2,180,360	2600
DATA 1,2,4,0,4,5	2610
DATA 1,2,4,6.5,4,6.5	2620
DATA -1,"k",2,4	2630
DATA 1,2,0,8,0,0	2640
DATA 1,2,0,2,4,5	2650
DATA 1,2,2,3.5,4,0	2660

DATA -1,"l",1,4	2670
DATA 1,2,1,0,1,8	2680
DATA -1,"m",3,4	2690
DATA 1,2,0,0,0,5	2700
DATA 2,1.5,3.5,1.5,1.5,0,180	2710
DATA 1,2,6,0,6,3.5	2720
DATA 2,4.5,3.5,1.5,1.5,0,180	2730
DATA 1,2,3,3.5,3,0	2740
DATA -1,"n",2,4	2750
DATA 1,2,4,0,4,3	2760
DATA 2,2,3,2,2,0,180	2770
DATA 1,2,0,5,0,0	2780
DATA -1,"o",2,4	2790
DATA 2,2,2.5,2,2.5,0,360	2800
DATA -1,"p",2,4	2810
DATA 2,1.5,2.5,2.5,2.5,-120,120	2820
DATA 1,2,0,5,0,-3	2830
DATA -1,"q",2,4	2840
DATA 2,2.5,2.5,2.5,2.5,60,300	2850
DATA 1,2,4,5,4,-3	2860
DATA -1,"r",2,4	2870
DATA 1,2,0,0,0,5	2880
DATA 2,2,3,2,2,0,180	2890
DATA -1,"s",2,4	2900
DATA 2,2,3.75,2,1.25,0,270	2910
DATA 2,2,1.25,2,1.25,-180,90	2920
DATA -1,"t",2,4	2930
DATA 1,2,0,7,0,2	2940
DATA 2,2,2,2,2,180,360	2950
DATA 1,2,0,5,4,5	2960
DATA -1,"u",2,4	2970
DATA 1,2,0,5,0,2	2980
DATA 2,2,2,2,2,180,360	2990
DATA 1,2,4,0,4,5	3000
DATA -1,"v",2,4	3010
DATA 1,3,0,5,2,0,4,5	3020
DATA -1,"w",3,4	3030
DATA 1,5,0,5,1.5,0,3,3,4.5,0,6,5	3040
DATA -1,"x",2,4	3050
DATA 1,2,0,5,4,0	3060
DATA 1,2,0,0,4,5	3070
DATA -1,"y",2,4	3080
DATA 1,2,0,5,0,2	3090
DATA 2,2,2,2,2,180,360	3100
DATA 1,2,4,5,4,-1	3110
DATA 2,2,-1,2,2,180,360	3120
DATA -1,"z",2,4	3130
DATA 1,4,0,5,4,5,0,0,4,0	3140
DATA -999	3150

**Clearface**

REM	500
DATA " ",3,4	510
DATA -1,"A",3,4	520
DATA 1,2,0,0,1,0	530
DATA 1,3,0.5,0,3,8,5,0	540
DATA 1,2,4.5,0,6,0	550

DATA 1,2,5.5,0,3.5,8	560
DATA 1,2,2.5,8,4,8	570
DATA 1,2,1.5,3,4.2,3	580
DATA -1,"B",3,4	590
DATA 1,2,0,0,4,0	600
DATA 1,2,0.5,0,0.5,8	610
DATA 1,2,0,8,3.5,8	620
DATA 1,2,1,8,1,0	630
DATA 1,2,1,4,4,4	640
DATA 2,3.5,6,2,2,-90,90	650
DATA 2,3.5,6,1.5,2,-90,90	660
DATA 2,4,2,2,2,-90,90	670
DATA 2,4,2,1.5,2,-90,90	680
DATA -1,"C",3,4	690
DATA 1,2,0,3,0,5	700
DATA 1,2,0.5,3,0.5,5	710
DATA 1,2,5,7,5,8	720
DATA 2,3,5,3,3,47,180	730
DATA 2,3,5,2.5,3,90,180	740
DATA 2,3,3,2.5,3,180,270	750
DATA 2,3,3,3,3,180,310	760
DATA -1,"D",3,4	770
DATA 1,2,0,0,3,0	780
DATA 1,2,0,8,3,8	790
DATA 1,2,0.5,0,0.5,8	800
DATA 1,2,1,8,1,0	810
DATA 2,3,4,3,4,-90,90	820
DATA 2,3,4,2.5,4,-90,90	830
DATA -1,"E",3,4	840
DATA 1,3,0,0,6,0,6,1.5	850
DATA 1,3,6,6.5,6,8,0,8	860
DATA 1,2,0.5,8,0.5,0	870
DATA 1,2,1,0,1,8	880
DATA 1,2,1,4,3.5,4	890
DATA 1,2,3.5,3.5,3.5,4.5	900
DATA 2,5.5,6.5,0.5,1.5,0,90	910
DATA 2,5.5,1.5,0.5,1.5,270,360	920
DATA -1,"F",3,4	930
DATA 1,3,6,6.5,6,8,0,8	940
DATA 1,2,0.5,8,0.5,0	950
DATA 1,2,1,0,1,8	960
DATA 1,2,1,4,3.5,4	970
DATA 1,2,3.5,3.5,3.5,4.5	980
DATA 2,5.5,6.5,0.5,1.5,0,90	990
DATA 1,2,0,0,1.5,0	1000
DATA -1,"G",3,4	1010
DATA 1,2,0,3,0,5	1020
DATA 1,2,0.5,3,0.5,5	1030
DATA 1,2,5,7,5,8	1040
DATA 2,3,5,3,3,47,180	1050
DATA 2,3,5,2.5,3,90,180	1060
DATA 2,3,3,2.5,3,180,270	1070
DATA 2,3,3,3,3,180,330	1080
DATA 1,5,5,4,6,4,6,0,5.5,0,5.5,4	1090
DATA -1,"H",3,4	1100
DATA 1,2,0,0,1.5,0	1110
DATA 1,2,0.5,0,0.5,8	1120
DATA 1,2,0,8,1.5,8	1130



DATA 1,2,1,8,1,0	1140
DATA 1,2,1,4,5,4	1150
DATA 1,2,4,5,0,6,0	1160
DATA 1,2,5,5,0,5,5,8	1170
DATA 1,2,6,8,4,5,8	1180
DATA 1,2,5,8,5,0	1190
DATA -1,"I",0.75,4	1200
DATA 1,2,0,0,1,5,0	1210
DATA 1,2,0,5,0,0,5,8	1220
DATA 1,2,0,8,1,5,8	1230
DATA 1,2,1,8,1,0	1240
DATA -1,"J",2.25,4	1250
DATA 1,2,3,8,4,5,8	1260
DATA 1,2,4,8,4,2	1270
DATA 1,2,3,5,8,3,5,0,8	1280
DATA 2,2,2,2,2,180,360	1290
DATA 2,0,5,2,0,5,0,5,0,360	1300
DATA -1,"K",3,4	1310
DATA 1,2,0,0,1,5,0	1320
DATA 1,2,0,5,0,0,5,8	1330
DATA 1,2,0,8,1,5,8	1340
DATA 1,2,1,8,1,0	1350
DATA 1,2,1,3,5,5,8	1360
DATA 1,2,6,8,5,8	1370
DATA 1,2,3,5,5,5,0	1380
DATA 1,2,6,0,4,5,0	1390
DATA 1,2,5,0,2,7,4,7	1400
DATA -1,"L",3,4	1410
DATA 1,2,0,0,1,5,0	1420
DATA 1,2,0,5,0,0,5,8	1430
DATA 1,2,0,8,1,5,8	1440
DATA 1,2,1,8,1,0	1450
DATA 1,3,0,0,6,0,6,1,5	1460
DATA 2,5,5,1,5,0,5,1,5,270,360	1470
DATA -1,"M",4.25,4	1480
DATA 1,2,0,0,1,0	1490
DATA 1,6,0,5,0,0,5,8,3,5,0,4,0,7,5,8,7,5,0	1500
DATA 1,2,7,0,8,5,0	1510
DATA 1,2,8,0,8,8	1520
DATA 1,2,8,5,8,7,5,8	1530
DATA 1,3,0,8,1,8,4,0	1540
DATA -1,"N",3,4	1550
DATA 1,2,0,0,1,0	1560
DATA 1,4,0,5,0,0,5,8,5,0,6,0	1570
DATA 1,2,5,5,0,5,5,8	1580
DATA 1,2,6,8,5,8	1590
DATA 1,3,0,8,1,8,5,5,0	1600
DATA -1,"O",3,4	1610
DATA 1,2,0,5,0,3	1620
DATA 2,3,3,3,3,180,360	1630
DATA 1,2,6,3,6,5	1640
DATA 2,3,5,3,3,0,180	1650
DATA 1,2,0,5,5,0,5,3	1660
DATA 2,3,3,2,5,3,180,360	1670
DATA 1,2,5,5,3,5,5,5	1680
DATA 2,3,5,2,5,3,0,180	1690
DATA 1,2,0,5,5,0,5,3	1700
DATA -1,"P",3,4	1710

DATA 1,2,0,0,1.5,0	1720
DATA 1,2,0.5,0,0.5,8	1730
DATA 1,2,0,8,4,8	1740
DATA 1,2,1,8,1,0	1750
DATA 1,2,1,4,4,4	1760
DATA 2,4,6,2,2,-90,90	1770
DATA 2,4,6,1.5,2,-90,90	1780
DATA -1,"Q",3,4	1790
DATA 1,2,0,5,0,3	1800
DATA 2,3,3,3,3,180,360	1810
DATA 1,2,6,3,6,5	1820
DATA 2,3,5,3,3,0,180	1830
DATA 1,2,0.5,5,0.5,3	1840
DATA 2,3,3,2.5,3,180,360	1850
DATA 1,2,5.5,3,5.5,5	1860
DATA 2,3,5,2.5,3,0,180	1870
DATA 1,2,0.5,5,0.5,3	1880
DATA 2,3,1,1,1,0,270	1890
DATA 2,3,1,0.5,1,0,270	1900
DATA 2,5,1,1.5,2,180,270	1910
DATA 2,5,1,1,2,180,270	1920
DATA -1,"R",3,4	1930
DATA 1,2,0,0,1.5,0	1940
DATA 1,2,0.5,0,0.5,8	1950
DATA 1,2,0,8,4,8	1960
DATA 1,2,1,8,1,0	1970
DATA 1,2,1,4,4,4	1980
DATA 2,4,6,2,2,-90,90	1990
DATA 2,4,6,1.5,2,-90,90	2000
DATA 2,4,2,1,2,0,90	2010
DATA 2,4,2,0.5,2,0,90	2020
DATA 2,6,2,1,2,180,270	2030
DATA 2,6,2,1.5,2,180,270	2040
DATA -1,"S",3,4	2050
DATA 1,3,0.33,1,0,1,0,2	2060
DATA 1,3,5.67,7,6,7,6,6	2070
DATA 2,3,6,3,2,0,270	2080
DATA 2,3,6,2.5,2,90,270	2090
DATA 2,3,2,3,2,-180,90	2100
DATA 2,3,2,2.5,2,-90,90	2110
DATA -1,"T",3,4	2120
DATA 1,4,0.6,5,0,8,6,8,6,6.5	2130
DATA 1,2,3.5,0,2.5,0	2140
DATA 1,2,2.75,0,2.75,8	2150
DATA 1,2,3.25,8,3.25,0	2160
DATA 2,1,6.5,1,1.5,90,180	2170
DATA 2,5,6.5,1,1.5,0,90	2180
DATA -1,"U",3,4	2190
DATA 1,2,0,8,1.5,8	2200
DATA 1,2,0.5,8,0.5,3	2210
DATA 1,2,1,3,1,8	2220
DATA 1,2,5,8,6,8	2230
DATA 1,2,5.5,8,5.5,3	2240
DATA 2,3,3,2.5,3,180,360	2250
DATA 2,3,3,2,3,180,270	2260
DATA -1,"V",3,4	2270
DATA 1,2,0,8,1.5,8	2280
DATA 1,3,0.5,8,3,0,5.5,8	2290

DATA 1,2,6,8,5,8	2300
DATA 1,2,1,8,3,3,1	2310
DATA -1,"W",4,4	2320
DATA 1,2,0,8,1,5,8	2330
DATA 1,4,0,5,8,2,5,0,4,5,8,6,3,1	2340
DATA 1,2,2,8,1,1,8	2350
DATA 1,2,4,8,5,8	2360
DATA 1,3,4,3,7,6,0,8,8	2370
DATA 1,2,8,5,8,7,8	2380
DATA -1,"X",3,4	2390
DATA 1,2,0,0,1,0	2400
DATA 1,2,0,5,0,5,5,8	2410
DATA 1,2,6,8,5,8	2420
DATA 1,2,1,5,8,0,8	2430
DATA 1,2,0,5,8,5,0	2440
DATA 1,2,4,5,0,6,0	2450
DATA 1,2,5,5,0,1,8	2460
DATA -1,"Y",3,4	2470
DATA 1,2,0,8,1,5,8	2480
DATA 1,3,0,5,8,2,75,4,2,75,0	2490
DATA 1,2,2,5,0,3,5,0	2500
DATA 1,3,3,25,0,3,25,4,1,8	2510
DATA 1,2,5,8,6,8	2520
DATA 1,2,5,5,8,3,25,4	2530
DATA -1,"Z",3,4	2540
DATA 1,4,0,6,5,0,8,6,8,0,5,0	2550
DATA 1,4,6,1,5,6,0,0,0,5,5,8	2560
DATA 2,0,5,6,5,0,5,1,5,90,180	2570
DATA 2,5,5,1,5,0,5,1,5,270,360	2580
DATA -1,"0",3,4	2590
DATA 1,2,0,5,0,3	2600
DATA 2,2,5,3,2,5,3,180,360	2610
DATA 1,2,5,3,5,5	2620
DATA 2,2,5,5,2,5,3,0,180	2630
DATA 1,2,0,5,5,0,5,3	2640
DATA 2,2,5,3,2,3,180,360	2650
DATA 1,2,4,5,3,4,5,5	2660
DATA 2,2,5,5,2,3,0,180	2670
DATA -1,"1",1,5,4	2680
DATA 1,4,0,6,1,8,1,5,8,1,5,0	2690
DATA 1,2,0,0,2,5,0	2700
DATA 1,2,1,0,1,8	2710
DATA -1,"2",3,4	2720
DATA 1,3,0,0,6,0,6,1,5	2730
DATA 2,5,5,1,5,0,5,1,5,270,360	2740
DATA 2,3,0,3,4,90,180	2750
DATA 2,3,0,2,5,4,90,180	2760
DATA 2,0,5,6,0,5,0,5,0,360	2770
DATA 2,3,6,3,2,-90,180	2780
DATA 2,3,6,2,5,2,-90,90	2790
DATA -1,"3",3,4	2800
DATA 2,0,5,6,0,5,0,5,0,360	2810
DATA 2,3,6,3,2,-90,180	2820
DATA 2,3,6,2,5,2,-90,90	2830
DATA 2,0,5,2,0,5,0,5,0,360	2840
DATA 2,3,2,3,2,-180,90	2850
DATA 2,3,2,2,5,2,-90,90	2860
DATA -1,"4",3,4	2870



DATA 1,2,2.5,0.5,0	2880
DATA 1,4,4,0,4,8,0,2.5,6,2.5	2890
DATA 1,2,3.5,0,3.5,7.4	2900
DATA -1,"5",3,4	2910
DATA 1,5,5,6.5,5,8,0,8,0,4,3,4	2920
DATA 1,2,0.5,4,0.5,8	2930
DATA 2,0.5,2,0.5,0.5,0,360	2940
DATA 2,3,2,3,2,-180,90	2950
DATA 2,3,2,2.5,2,-90,90	2960
DATA 2,4.5,6.5,0.5,1.5,0,90	2970
DATA -1,"6",3,4	2980
DATA 1,2,0,2,0,6	2990
DATA 1,2,0.5,6,0.5,3	3000
DATA 2,3,2,3,2,0,360	3010
DATA 2,3,2,2.5,2,0,360	3020
DATA 2,5.5,6,0.5,0.5,0,360	3030
DATA 2,3,6,3,2,0,180	3040
DATA 2,3,6,2.5,2,90,180	3050
DATA -1,"7",3,4	3060
DATA 1,2,0,0,1.5,0	3070
DATA 1,5,1,0,6,7.5,6,8,0,8,0,6.5	3080
DATA 1,2,6,8,0.5,0	3090
DATA 2,1,6.5,1,1.5,90,180	3100
DATA -1,"8",3,4	3110
DATA 2,3,6,2.5,2,0,360	3120
DATA 2,3,6,2,2,0,360	3130
DATA 2,3,2,3,2,0,360	3140
DATA 2,3,2,2.5,2,0,360	3150
DATA -1,"9",3,4	3160
DATA 2,0.5,2,0.5,0.5,0,360	3170
DATA 2,3,2,3,2,180,360	3180
DATA 1,2,6,2,6,6	3190
DATA 2,3,2,2.5,2,270,360	3200
DATA 1,2,5.5,2,5.5,6	3210
DATA 2,3,6,3,2,0,360	3220
DATA 2,3,6,2.5,2,0,360	3230
DATA -1,"(",3,4	3240
DATA 2,4,4,1,4,90,270	3250
DATA 2,4,4,1.5,4,90,270	3260
DATA -1,")",3,4	3270
DATA 2,2,4,1,4,-90,90	3280
DATA 2,2,4,1.5,4,-90,90	3290
DATA -1,":",3,4	3300
DATA 2,3,2,0.5,0.5,0,360	3310
DATA 2,3,5,0.5,0.5,0,360	3320
DATA -1,"=",3,4	3330
DATA 1,5,1,5.5,5.5,5.5,1,5,1,5.5	3340
DATA 1,5,1,2.5,5,2.5,5,3,1,3,1,2.5	3350
DATA -1,"-",3,4	3360
DATA 1,5,1,3.75,5,3.75,5,4.25,1,4.25,1,3.75	3370
DATA -1,"+",3,4	3380
DATA 1,5,1,3.75,5,3.75,5,4.25,1,4.25,1,3.75	3390
DATA 1,5,2.75,2,2.75,6,3.25,6,3.25,2,2.75,2	3400
DATA -1,";",3,4	3410
DATA 2,3,5,0.5,0.5,0,360	3420
DATA 2,3,2,0.5,1.5,270,360	3430
DATA 2,3,2,0.5,0.5,0,270	3440
DATA 1,2,3,1.5,3,0.5	3450

DATA -1,"",2,4	3460
DATA 2,2,0.5,0.5,1.5,270,360	3470
DATA 2,2,0.5,0.5,0.5,0,270	3480
DATA 1,2,2,0,2,-1	3490
DATA -1,".",2,4	3500
DATA 2,2,0.5,0.5,0.5,0,360	3510
DATA -1,"/",3,4	3520
DATA 1,5,1,0,1.5,0.5,8,4.5,8,1,0	3530
DATA -1,"a",2.75,4	3540
DATA 1,5,5,0.5,5.5,5.5,5.5,0.5,0	3550
DATA 2,3,3,3,3,47,313	3560
DATA 2,3,3,2.5,3,90,270	3570
DATA -1,"b",3,4	3580
DATA 1,5,0,8,1,8,1,0,0.5,0,0.5,8	3590
DATA 2,3,3,3,3,-133,133	3600
DATA 2,3,3,2.5,3,-90,90	3610
DATA -1,"c",2.5,4	3620
DATA 2,3,3,3,3,45,315	3630
DATA 2,3,3,2.5,3,90,270	3640
DATA -1,"d",2.75,4	3650
DATA 1,5,4.5,8,5.5,8,5.5,0.5,0.5,8	3660
DATA 2,3,3,3,3,47,313	3670
DATA 2,3,3,2.5,3,90,270	3680
DATA -1,"e",2.5,4	3690
DATA 1,2,0.5,3,5,3	3700
DATA 2,2.5,3,2.5,3,0,315	3710
DATA 2,2.5,3,2,3,0,270	3720
DATA -1,"f",1.75,4	3730
DATA 1,4,0.5,7,0.5,0,1,0,1,7	3740
DATA 1,2,3,7,3.5,7	3750
DATA 1,2,0,4,2,4	3760
DATA 2,2,7,1.5,1,0,180	3770
DATA 2,2,7,1,1,0,180	3780
DATA -1,"g",2.75,4	3790
DATA 1,2,0.5,-1,1,-1	3800
DATA 1,4,5,-1,5,5.5,5.5,5.5,5.5,-1	3810
DATA 2,3,3,3,3,47,313	3820
DATA 2,3,3,2.5,3,90,270	3830
DATA 2,3,-1,2.5,2,180,360	3840
DATA 2,3,-1,2,2,180,360	3850
DATA -1,"h",2.75,4	3860
DATA 1,5,0,8,1,8,1,0,0.5,0,0.5,8	3870
DATA 1,4,5,3,5,0.5,5,0.5,5,3	3880
DATA 2,3,3,2.5,3,0,180	3890
DATA 2,3,3,2,3,0,180	3900
DATA -1,"i",0.75,4	3910
DATA 1,5,0.5,7,1,7,1,7.5,0.5,7.5,0.5,7	3920
DATA 1,3,0,6,1,6,1,0	3930
DATA 1,3,0.5,6,0.5,0,1.5,0	3940
DATA -1,"j",1.75,4	3950
DATA 1,5,3.5,7,3,7,3,7.5,3.5,7.5,3.5,7	3960
DATA 1,3,2.5,6,3.5,6,3.5,-1	3970
DATA 1,2,3,6,3,-1	3980
DATA 1,2,0,-1,0.5,-1	3990
DATA 2,1.75,-1,1.25,2,180,360	4000
DATA 2,1.75,-1,1.75,2,180,360	4010
DATA -1,"k",2.75,4	4020
DATA 1,5,0,8,1,8,1,0,0.5,0,0.5,8	4030

DATA 1,2,1,3,5.5,6	4040
DATA 1,4,2.7,4,5.5,0,5,0,2.3,3.7	4050
DATA -1,"l",0.75,4	4060
DATA 1,3,0,8,1,8,1,0	4070
DATA 1,3,1.5,0,0.5,0,0.5,8	4080
DATA -1,"m",3.5,4	4090
DATA 1,5,0,6,1,6,1,0,0.5,0,0.5,6	4100
DATA 1,4,3.5,5,3.5,0,4,0,4,5	4110
DATA 1,4,6.5,5,6.5,0,7,0,7,5	4120
DATA 2,2.25,5,1.25,1,0,180	4130
DATA 2,2.5,5,1.5,1,0,180	4140
DATA 2,5.25,5,1.25,1,0,180	4150
DATA 2,5.5,5,1.5,1,0,180	4160
DATA -1,"n",2.75,4	4170
DATA 1,5,0,6,1,6,1,0,0.5,0,0.5,6	4180
DATA 1,4,5,5,5,0,5.5,0,5.5,5	4190
DATA 2,3,5,2,1,0,180	4200
DATA 2,3.25,5,2.25,1,0,180	4210
DATA -1,"o",2.75,4	4220
DATA 2,2.75,3,2.75,3,0,360	4230
DATA 2,2.75,3,2.25,3,0,360	4240
DATA -1,"p",2.75,4	4250
DATA 1,5,0,6,1,6,1,-3,0.5,-3,0.5,6	4260
DATA 2,2.5,3,3,3,-120,120	4270
DATA 2,2.5,3,2.5,3,-90,90	4280
DATA -1,"q",2.75,4	4290
DATA 1,5,5,5,5,5,5,5,5,5,-3,5,-3,5,5,5	4300
DATA 2,3,3,3,3,47,313	4310
DATA 2,3,3,2.5,3,90,270	4320
DATA -1,"r",2.75,4	4330
DATA 1,2,5,4,5.5,4	4340
DATA 1,5,0,6,1,6,1,0,0.5,0,0.5,6	4350
DATA 2,3,4,2,2,0,180	4360
DATA 2,3.25,4,2.25,2,0,180	4370
DATA -1,"s",2.75,4	4380
DATA 2,2.75,4.5,2.75,1.5,0,270	4390
DATA 2,2.75,1.5,2.75,1.5,-180,90	4400
DATA 2,2.75,4.5,2.25,1.5,0,270	4410
DATA 2,2.75,1.5,2.25,1.5,-180,90	4420
DATA 1,2,5,4.5,5.5,4.5	4430
DATA 1,2,0,1.5,0.5,1.5	4440
DATA -1,"t",2.25,4	4450
DATA 1,4,0.5,2,0.5,8,1,8,1,2	4460
DATA 1,2,4,2,4.5,2	4470
DATA 1,2,0,6,3,6	4480
DATA 2,2.5,2,1.5,2,180,360	4490
DATA 2,2.5,2,2,2,180,360	4500
DATA -1,"u",2.75,4	4510
DATA 1,2,0.5,2,0.5,6	4520
DATA 1,3,0,6,1,6,1,2	4530
DATA 1,4,5,2,5,6,5.5,6,5.5,2	4540
DATA 2,3,2,2.5,2,180,360	4550
DATA 2,3,2,2,2,180,360	4560
DATA -1,"v",2.75,4	4570
DATA 1,3,0,6,1,6,2.75,0	4580
DATA 1,4,5.5,6,2.75,0,2.25,0,0.5,6	4590
DATA -1,"w",3,4	4600
DATA 1,2,0.5,2,0.5,6	4610



DATA 1,3,0,6,1,6,1,2	4620
DATA 1,4,3.5,2,3.5,6,4,6,4,2	4630
DATA 1,4,6.5,2,6.5,6,7,6,7,2	4640
DATA 2,2.25,2,1.25,2,180,360	4650
DATA 2,2.25,2,1.75,2,180,360	4660
DATA 2,5.25,2,1.25,2,180,360	4670
DATA 2,5.25,2,1.75,2,180,360	4680
DATA -1,"x",2.75,4	4690
DATA 1,3,0,6,1,6,5,0	4700
DATA 1,3,5.5,0,4.5,0,0.5,6	4710
DATA 1,2,0,0,1,0	4720
DATA 1,2,0.5,0,5,6	4730
DATA 1,2,4.5,6,5.5,6	4740
DATA -1,"y",2.75,4	4750
DATA 1,2,0.5,2,0.5,6	4760
DATA 1,3,0,6,1,6,1,2	4770
DATA 1,4,5,2,5,6,5.5,6,5.5,-1	4780
DATA 1,2,5,-1,5,1.5	4790
DATA 1,2,0.5,-1,1,-1	4800
DATA 2,3,2,2.5,2,180,330	4810
DATA 2,3,2,2,2,180,360	4820
DATA 2,3,-1,2.5,2,180,360	4830
DATA 2,3,-1,2,2,180,360	4840
DATA -1,"z",2.75,4	4850
DATA 1,3,0,6,5.5,6,0.5,0	4860
DATA 1,3,5,6,0,0.5,5,0	4870
DATA -999	4880

**Gothic**

REM	500
DATA " ",3,4	510
DATA -1,"A",3,4	520
DATA 1,2,1,1,2,6	530
DATA 1,2,2,0,3,1.5	540
DATA 4,5.5,1.5,0.5,0.25,180,360,0.5,0.5	550
DATA 4,1.5,0,1.5,1.2,180,360,1.5,1.5	560
DATA 4,1,0,1,0.7,0,180,1,1.5	570
DATA 2,1.5,7,1.25,1,0,180	580
DATA 3,5,1.4,3,1.5,3.5,3.5,3.5,3,1.4,3	590
DATA 3,8,5,0,5.5,1,2.75,7,2.2,7.5,1.25,7.5,1.5,7,4.5,0.5,5,0	600
DATA 2,1,7,0.75,1,180,315	610
DATA 4,1.5,7,1.25,0.5,0,180,1.25,1	620
DATA -1,"B",3,4	630
DATA 2,0,8,5,1.6,270,330	640
DATA 4,4.25,6,0.75,1.25,-90,90,1.75,1.25	650
DATA 4,1,3,1,1,90,180,1,2	660
DATA 2,2,0,2,2,3,90,180	670
DATA 3,6,1,2,1.5,2.2,2,2,3,2,6.5,1,6.4,1,2	680
DATA 2,2,0,2,1.5,180,330	690
DATA 4,1.75,0,1.75,1,0,180,1.75,1.8	700
DATA 2,6,0,2.5,1,90,180	710
DATA 1,2,2.75,1,2.75,6.75	720
DATA 1,2,2,4.5,4.25,4.75	730
DATA 3,6,5,0.95,6,1,6,3,5.5,3.7,5,3,5,1	740
DATA 4,3,5,3,1.5,1.6,0,90,2.5,1.65	750
DATA -1,"C",3,4	760
DATA 1,2,5,1,5,6.5	770

DATA 1,2,1.5,1.25,4,2.5	780
DATA 1,2,1.6,5,8	790
DATA 2,3,4,3,4,270,360	800
DATA 3,5,3,2,4,2.5,4,7.5,3,7,3,2	810
DATA 4,3,4,2,4,90,270,3,4	820
DATA 4,5,5,8,0.5,1,180,360,0.5,1.5	830
DATA 4,5,6,7,3,0.5,0.75,180,270,1.5,0.75	840
DATA 3,5,4,7.5,4,7,5,2,7,5,8,4,7,5	850
DATA -1,"D",3,4	860
DATA 2,0,6.75,2.75,0.3,270,360	870
DATA 4,4,5.75,1,1,0,90,2,2	880
DATA 3,5,5,1,6,1,6,5.75,5,5.75,5,1	890
DATA 3,5,1,6.75,4,6.75,4,7.75,1,7.75,1,6.75	900
DATA 4,1,8,1,0.25,180,270,1,1.25	910
DATA 4,1,3,1,1,90,180,1,2	920
DATA 2,2,0,2,2,3,90,180	930
DATA 3,6,1,2,1.5,2,2,2,2,3,2,6.5,1,6.4,1,2	940
DATA 2,2,0,2,1.5,180,330	950
DATA 4,1,75,0,1,75,1,0,180,1,75,1,8	960
DATA 2,6,0,2.5,1,90,180	970
DATA 1,2,2.75,1,2,75,6.75	980
DATA -1,"E",3,4	990
DATA 1,2,5,1,5,6.5	1000
DATA 1,2,1.5,1.25,4,2.5	1010
DATA 1,2,1,6,5,8	1020
DATA 2,3,4,3,4,270,360	1030
DATA 3,5,3,2,4,2.5,4,7.5,3,7,3,2	1040
DATA 4,3,4,2,4,90,270,3,4	1050
DATA 4,5,5,8,0.5,1,180,360,0.5,1.5	1060
DATA 4,5,6,7,3,0.5,0.75,180,270,1.5,0.75	1070
DATA 3,5,4,7.5,4,7,5,2,7,5,8,4,7,5	1080
DATA 4,5,5,4,5,0.5,0.25,0,180,0.5,1	1090
DATA 3,6,5,5,4,5,8,4,4,6,4,5,5,5,4,8,5,3,4,5,5,5,4	1100
DATA -1,"F",3,4	1110
DATA 2,2,0,2,1.5,180,330	1120
DATA 4,1,0,1,1,90,180,1,2	1130
DATA 4,1,0,2,1,0,90,2,2	1140
DATA 2,4,5,0,1.5,1,90,180	1150
DATA 1,2,3,5,0.75,3,5,7	1160
DATA 3,5,3,5,4,4,4,4,5,5,3,5,5,3,5,4	1170
DATA 2,0,8,4,1.5,270,360	1180
DATA 3,5,1,2,2,2,5,2,6,7,1,6,5,1,2	1190
DATA 3,5,0,4,1,4,1,5,0.5,5,0,4	1200
DATA 4,4,75,8,0.75,1,180,360,0.75,1.5	1210
DATA 4,4,5,8,1.5,1,180,270,1.5,1.5	1220
DATA 3,5,3,7,3,7,7,3,4,8,4,5,7,3,7	1230
DATA -1,"G",3,4	1240
DATA 1,2,5,1,5,6.5	1250
DATA 1,2,1.5,1.25,4,2.5	1260
DATA 1,2,1,6,5,8	1270
DATA 2,3,4,3,4,270,311	1280
DATA 3,5,3,2,4,2.5,4,7.5,3,7,3,2	1290
DATA 4,3,4,2,4,90,270,3,4	1300
DATA 4,5,5,8,0.5,1,180,360,0.5,1.5	1310
DATA 4,5,6,7,3,0.5,0.75,180,270,1.5,0.75	1320
DATA 3,5,4,7.5,4,7,5,2,7,5,8,4,7,5	1330
DATA 2,5,8,1,3,5,270,360	1340
DATA 4,5,2.75,0.5,1,75,-90,90,1,1,75	1350

DATA -1,"L",2.5,4	1360
DATA 2,2,0,2,1.5,180,330	1370
DATA 2,1,0,1,2,90,180	1380
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	1390
DATA 1,3,4,1.5,3,0,3,7	1400
DATA 2,0,8,4,1.5,270,360	1410
DATA 3,5,1,2,2,2.5,2,6.7,1,6.5,1,2	1420
DATA 4,1,3,1,1,90,180,1,2	1430
DATA 4,4.5,8,0.5,0.5,180,360,0.5,1	1440
DATA 3,6,4,8,3.8,7.5,3.6,7.3,4,7.1,4.5,7,4,8	1450
DATA -1,"H",3,4	1460
DATA 2,2,0,2,1.5,180,330	1470
DATA 2,1,0,1,2,90,180	1480
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	1490
DATA 1,3,4,1.5,3,0,3,7	1500
DATA 2,0,8,4,1.5,270,360	1510
DATA 3,5,1,2,2,2.5,2,6.7,1,6.5,1,2	1520
DATA 4,1,3,1,1,90,180,1,2	1530
DATA 4,4.5,8,0.5,0.5,180,360,0.5,1	1540
DATA 3,6,4,8,3.8,7.5,3.6,7.3,4,7.1,4.5,7,4,8	1550
DATA 4,4,2,1,2,270,360,2,2	1560
DATA 3,5,5,2,6,2,6,5,5,5,5,2	1570
DATA 3,5,6,5,5,6,4,5,5,5,4,5,6,5	1580
DATA 1,2,2,4,5,5,6	1590
DATA -1,"K",3,4	1600
DATA 2,2,0,2,1.5,180,330	1610
DATA 2,1,0,1,2,90,180	1620
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	1630
DATA 1,3,4,1.5,3,0,3,7	1640
DATA 2,0,8,4,1.5,270,360	1650
DATA 3,5,1,2,2,2.5,2,6.7,1,6.5,1,2	1660
DATA 4,1,3,1,1,90,180,1,2	1670
DATA 4,4.5,8,0.5,0.5,180,360,0.5,1	1680
DATA 3,6,4,8,3.8,7.5,3.6,7.3,4,7.1,4.5,7,4,8	1690
DATA 1,2,5,0,6,1.5	1700
DATA 1,2,2,4,5,5,6	1710
DATA 3,5,4.5,1,5.5,1,5.5,0.75,5,0,4.5,1	1720
DATA 4,4,1,0.5,2.5,0,90,1.5,2.5	1730
DATA 4,4,4.5,1,1,-90,90,2,1	1740
DATA 4,5,4.5,1,0.5,0,90,1,1.5	1750
DATA 3,4,5,6,5,5,25,4,5,5,5,6	1760
DATA -1,"I",2.5,4	1770
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	1780
DATA 2,5,5,0,2.5,1.5,90,180	1790
DATA 3,6,5,1,45,5,6,3,4,5,5,4,1,2,4.5,1.33,5,1.45	1800
DATA 1,2,3,7,3,0	1810
DATA 4,3,2.5,1,1,90,180,1,2	1820
DATA 2,1.5,7,1.5,1,180,330	1830
DATA 4,1.5,7,1.5,0.5,0,180,1.5,1	1840
DATA 4,4,7,1,0.5,180,360,1,1	1850
DATA 3,5,2,7.5,3.5,6.2,4,6.5,2.5,7.75,2,7.5	1860
DATA -1,"J",2.5,4	1870
DATA 2,2.5,0,2.5,1.5,180,360	1880
DATA 4,2.5,0,1.5,1.5,270,360,2.5,1.5	1890
DATA 3,5,5,0,5,6,3,4,5,5,4,0,5,0	1900
DATA 1,2,3,7,3,0	1910
DATA 4,3,2.5,1,1,90,180,1,2	1920
DATA 2,1.5,7,1.5,1,180,330	1930



DATA 4,1.5,7,1.5,0.5,0,180,1.5,1	1940
DATA 4,4,7,1,0.5,180,360,1,1	1950
DATA 3,5,2,7.5,3.5,6.2,4,6.5,2.5,7.75,2,7.5	1960
DATA 4,1,0,1,1,0,180,1,1.5	1970
DATA -1,"N",3,4	1980
DATA 2,2,0,2,1.5,180,330	1990
DATA 2,1,0,1,2,90,180	2000
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	2010
DATA 1,2,3,0,3,6	2020
DATA 1,2,2,4.5,4,7.5	2030
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	2040
DATA 4,1.5,7,0.05,0.05,0,90,0.5,0.5	2050
DATA 4,0.5,6,0.5,0.5,0,90,1,0.5	2060
DATA 3,5,1,2,2,2.5,2,7,1,7,1,2	2070
DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	2080
DATA 3,6,4,7.5,5.6,5.5,6,4,6,3,4,6.5,4,7.5	2090
DATA 3,5,4,6,5,6,5,1.5,4,1.5,4,6	2100
DATA 3,5,5,0,5,6,1,4,5,2,3,75,1.25,5,0	2110
DATA 1,2,5,0,6,1,5	2120
DATA -1,"M",4,5,4	2130
DATA 2,2,0,2,1.5,180,330	2140
DATA 2,1,0,1,2,90,180	2150
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	2160
DATA 1,2,3,0,3,6	2170
DATA 1,2,2,4.5,4,7.5	2180
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	2190
DATA 4,1.5,7,0.05,0.05,0,90,0.5,0.5	2200
DATA 4,0.5,6,0.5,0.5,0,90,1,0.5	2210
DATA 3,5,1,2,2,2.5,2,7,1,7,1,2	2220
DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	2230
DATA 3,6,4,7.5,5.6,5.5,6,4,6,3,4,6.5,4,7.5	2240
DATA 3,5,4,6,5,6,5,1.5,4,1.5,4,6	2250
DATA 3,5,5,0,5,6,1,4,5,2,3,75,1.25,5,0	2260
DATA 1,3,5,0,6,1,5,6,6	2270
DATA 1,2,6,1,5,6,6	2280
DATA 3,6,7,7.5,8,6,5,8,6,7,6,6,4,6.5,7,7.5	2290
DATA 3,5,7,6,8,6,8,1.5,7,1.5,7,6	2300
DATA 3,5,8,0,8,6,1,7,5,2,6,75,1.25,8,0	2310
DATA 1,2,8,0,9,1,5	2320
DATA 1,2,5,4,5,7,7.5	2330
DATA 3,5,3,3,6,3,6,4,3,4,3,3	2340
DATA -1,"O",3,4	2350
DATA 4,3,4,1,75,4,90,360,3,4	2360
DATA 4,4,4,0,75,3,0,90,2,3	2370
DATA 1,2,1,5,5,4,7	2380
DATA 3,5,2,5,1.5,3,5,2,3,5,6,75,2,5,6,25,2,5,1.5	2390
DATA 1,2,1,5,1,3,5,2	2400
DATA -1,"Q",3,4	2410
DATA 4,3,4,1,75,4,90,360,3,4	2420
DATA 4,4,4,0,75,3,0,90,2,3	2430
DATA 1,2,1,5,5,4,7	2440
DATA 3,5,2,5,1.5,3,5,2,3,5,6,75,2,5,6,25,2,5,1.5	2450
DATA 4,4,5,-1,0.5,2,0,90,1,5,2	2460
DATA -1,"R",3,4	2470
DATA 2,2,0,2,1.5,180,330	2480
DATA 2,1,0,1,2,90,180	2490
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	2500
DATA 1,2,3,0,3,6,5	2510

DATA 1,2,2,5,4,8	2520
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	2530
DATA 4,1.5,7,0.05,0.05,0,90,0.5,0.5	2540
DATA 4,0.5,6,0.5,0.5,0,90,1,0.5	2550
DATA 3,5,1,2,2,2.5,2,7,1,7,1,2	2560
DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	2570
DATA 3,5,5,0,5.6,1,4.5,2,3.75,1.25,5,0	2580
DATA 1,2,5,0,6,1.5	2590
DATA 4,3,6,2,2,270,360,3,2	2600
DATA 4,3,5,6,1.5,1,0,90,2.5,1	2610
DATA 4,3,1.5,1,2.5,0,90,2,2.5	2620
DATA 4,4,6,1,2,0,90,2,2	2630
DATA 3,5,4,8,3,4,7,5,6.8,4.5,7.8,4,8	2640
DATA -1,"P",3,4	2650
DATA 2,2,0,2,1.5,180,330	2660
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	2670
DATA 1,2,3,0,3,6.5	2680
DATA 1,2,2,5.5,4.5,8	2690
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	2700
DATA 4,1.5,7,0.05,0.05,0,90,0.5,0.5	2710
DATA 4,0.5,6,0.5,0.5,0,90,1,0.5	2720
DATA 3,5,1,-2,2,-2,2,7,1,7,1,-2	2730
DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	2740
DATA 2,6,0,3,1.5,90,180	2750
DATA 3,5,4.5,1.3,5.5,1.5,5.5,7,4.5,6.5,4.5,1.3	2760
DATA 3,5,4.5,6.5,5.5,7,4.5,8,3.5,7,4.5,6	2770
DATA -1,"S",3,4	2780
DATA 2,2,0,2,1.5,180,330	2790
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	2800
DATA 1,2,3,0,3,8	2810
DATA 2,6,5,1,1,180,270	2820
DATA 2,3,5,2,1,270,360	2830
DATA 2,3,3,2,1,90,180	2840
DATA 2,0,3,1,1,0,90	2850
DATA 2,0,8,3,1.5,270,360	2860
DATA 2,6,0,3,1.5,90,180	2870
DATA 4,0,4.5,0.5,0.5,270,360,1.5,0.5	2880
DATA 3,5,0.5,4,0.5,6.5,1.5,6.7,1.5,3.8,0.5,4	2890
DATA 3,6,5.5,1.5,5.5,4,1.5,4.6,4.5,4.3,4.5,1.3,5.5,1.5	2900
DATA 4,0,3,1,1,0,60,1,1.5	2910
DATA 4,0,3,1,1,45,90,2,1	2920
DATA 4,3,3,2,1,140,180,2,1.5	2930
DATA 3,3,0.5,4.5,1,3,1,1.5,4.5	2940
DATA 4,3,5,2,1,330,360,2,2	2950
DATA 4,6,5,1,1,180,220,1,1.5	2960
DATA 4,4,8,1,1,180,360,1,1.5	2970
DATA -1,"T",3,4	2980
DATA 4,2,3.5,1,3,90,180,2,3	2990
DATA 4,3,3.5,2,3.5,180,270,3,3.5	3000
DATA 2,3,3.5,3,3.5,270,360	3010
DATA 4,2,6,2,1,90,180,2,2	3020
DATA 4,4,8,2,1,270,360,2,2	3030
DATA 1,2,2,0.5,2,7	3040
DATA 1,3,2,5,4,6,4,0.25	3050
DATA 3,5,2,1.5,3,2,3,5.5,2,5,2,1.5	3060
DATA 4,2,6,1,1,0,90,1,2	3070
DATA 4,4,8,1,1,180,270,1,2	3080



DATA -1,"U",3,4	3090
DATA 2,2,0,2,1.5,180,330	3100
DATA 2,1,0,1,2,90,180	3110
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	3120
DATA 1,2,3,0,3,6	3130
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	3140
DATA 4,1.5,7,0.05,0.05,0.90,0.5,0.5	3150
DATA 4,0.5,6,0.5,0.5,0.90,1,0.5	3160
DATA 3,5,1,2,2,2.5,2,7,1,7,1,2	3170
DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	3180
DATA 2,5,0,2,1,90,180	3190
DATA 3,6,4,0.9,4.5,1.5,1.5,7,4,7,4,1	3200
DATA 4,6,7,1,1,90,180,2,1	3210
DATA 1,2,5,0,6,1.5	3220
DATA 3,5,4,0.9,5,0.5,5,0.75,5,1.25,4,0.9	3230
DATA -1,"V",3,4	3240
DATA 2,2,0,2,1.5,180,330	3250
DATA 2,1,0,1,2,90,180	3260
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	3270
DATA 1,2,2,4.5,4,7.5	3280
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	3290
DATA 4,1.5,7,0.05,0.05,0.90,0.5,0.5	3300
DATA 4,0.5,6,0.5,0.5,0.90,1,0.5	3310
DATA 3,5,1,2,2,2.5,2,7,1,7,1,2	3320
DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	3330
DATA 2,6,0,3,1.5,90,180	3340
DATA 3,6,5.5,1.5,5,1.5,4.5,1.3,4.5,6.5,5.5,6.5,5.5,1.5	3350
DATA 3,5,5.5,6.5,4.3,8,3.7,7,4.5,5.8,5.5,6.5	3360
DATA -1,"W",4.5,4	3370
DATA 2,2,0,2,1.5,180,330	3380
DATA 2,1,0,1,2,90,180	3390
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	3400
DATA 1,2,3,0,3,6	3410
DATA 1,2,2,4.5,4,7.5	3420
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	3430
DATA 4,1.5,7,0.05,0.05,0.90,0.5,0.5	3440
DATA 4,0.5,6,0.5,0.5,0.90,1,0.5	3450
DATA 3,5,1,2,2,2.5,2,7,1,7,1,2	3460
DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	3470
DATA 2,6,0,3,1.5,120,180	3480
DATA 3,5,5.5,1.5,4.5,1.3,4.5,6.5,5.5,6.5,5.5,1.5	3490
DATA 3,5,5.5,6.5,4.3,8,3.7,7,4.5,5.8,5.5,6.5	3500
DATA 3,6,4.5,1.3,5.5,0.5,7,0.6,6,0.9,5.5,1.5,4.5,1.3	3510
DATA 2,8.5,0,3,1.5,90,180	3520
DATA 3,5,8.5,1.5,7.5,1.3,7.5,6.5,8.5,6.5,8.5,1.5	3530
DATA 3,5,8.5,6.5,7.3,8,6.7,7,7.5,5.8,8.5,6.5	3540
DATA 3,5,7.5,1.3,8.5,0.9,1,8.5,1.5,7.5,1.3	3550
DATA 1,2,5,4.5,7,7.5	3560
DATA -1,"Y",3,4	3570
DATA 2,2,0,2,1.5,180,330	3580
DATA 2,1,0,1,2,90,180	3590
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	3600
DATA 1,2,3,0,3,6	3610
DATA 1,2,2,4.5,4,7.5	3620
DATA 4,0.5,8,0.5,0.5,180,270,0.5,1.5	3630
DATA 4,1.5,7,0.05,0.05,0.90,0.5,0.5	3640
DATA 4,0.5,6,0.5,0.5,0.90,1,0.5	3650
DATA 3,5,1,2,2,2.5,2,7,1,7,1,2	3660



DATA 3,5,0.5,6.5,1.5,6.5,1.5,7.5,0.5,7.5,0.5,6.5	3670
DATA 2,6,0,3,1.5,120,180	3680
DATA 3,5,5.5,1.5,4.5,1.3,4.5,6.5,5.5,6.5,5.5,1.5	3690
DATA 3,5,5.5,6.5,4.3,8,3.7,7,4.5,5.8,5.5,6.5	3700
DATA 4,3.5,1.5,1,3,270,360,2,3	3710
DATA -1,"X",3,4	3720
DATA 2,2,0,2,1.5,180,330	3730
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	3740
DATA 1,2,5,0,6,1.5	3750
DATA 3,5,5,0,5.5,0.75,2,6.5,1,6.5,5,0	3760
DATA 1,2,2,1.3,5,7	3770
DATA 3,4,5,7,5,6,4.5,6,5,7	3780
DATA 4,5,8,1,1,270,360,1,2	3790
DATA 4,0,6.5,0.5,1,0,90,2,0.5	3800
DATA 3,5,1,3.5,4.5,3.5,5,4.5,1.5,4.5,1,3.5	3810
DATA -1,"Z",2.5,4	3820
DATA 2,2,0,2,1.5,180,330	3830
DATA 4,1.5,0,1.5,1,0,180,1.5,1.5	3840
DATA 2,5,0,2,1.5,90,180	3850
DATA 4,1,8,1,1,180,270,1,2	3860
DATA 3,5,1,6,3,6,3,7,1,7,1,6	3870
DATA 4,3,5,1,1,0,90,2,2	3880
DATA 4,3,5,1,1,270,360,2,1	3890
DATA 4,3,1.5,1,2.5,0,90,2,2.5	3900
DATA 3,4,4,1.5,4.5,1.5,4,1.3,4,1.5	3910
DATA 1,2,2,1.3,2,6	3920
DATA 1,2,1,4,3,4	3930
DATA -1,"-",3,4	3940
DATA 1,2,0.5,2.5,1,3.5	3950
DATA 1,2,5,4.5,5.5,5.5	3960
DATA 3,5,1,3.5,4.5,3.5,5,4.5,1.5,4.5,1,3.5	3970
DATA -1,"",3,4	3980
DATA 4,2.5,0,0.05,0.05,0,360,0.5,0.5	3990
DATA 4,2,0,0.5,1.5,270,360,1,1.5	4000
DATA -1,".",3,4	4010
DATA 4,2.5,0,0.05,0.05,0,360,0.5,0.5	4020
DATA -9999	4030

### Electrical Symbols

REM	500
DATA " ",3,4	510
DATA -1,".",3,4 ! DOT	520
DATA 4,0,0,0.05,0.05,0,360,0.22,0.22	530
DATA -1,"*",3,4 !'AND' GATE	540
DATA 1,4,6.4,-4,0,-4,0,4,6.4,4	550
DATA 2,6.4,0,4,4,-90,90	560
DATA -1,"+",3,4 !'OR' GATE	570
DATA 2,0,0,2,4,-90,90	580
DATA 1,2,0,4,3,4	590
DATA 1,2,0,-4,3,-4	600
DATA 2,3,-4,8,8,30,90	610
DATA 2,3,4,8,8,270,330	620
DATA -1,"<",>3,4 ! ARROWHEAD	630
DATA 3,3,1,0.5,0,0,1,-0.5	640
D"A"TA -1,A,3,4 ! ANTENNA	650
DATA 1,2,0,0,0,9	660

DATA 1,3,-2.5,9,0,6,2.5,9	670
DATA -1,"B",3,4 ! SQUARE BOX	680
DATA 1,5,-5,0,-5,10,5,10,5,0,-5,0	690
DATA -1,b,3,4 ! RECTANGULAR BOX	700
DATA 1,5,-5,0,-5,25,5,25,5,0,-5,0	710
DATA -1,"C",3,4 ! SIGNED CAPACITOR	720
DATA 1,2,0,0,0,-1.6	730
DATA 1,6,-1.6,-1.6,1.6,-1.6,1.6,-1.65,-1.6,-1.65,-1.6,-1.7,1.6,-1.7	740
DATA 1,2,0,-3.2,0,-4.8	750
DATA 2,0,-6.4,3.2,3.2,60,120	760
DATA 1,2,2,0,2,-0.8	770
DATA 1,2,1.6,-0.4,2.4,-0.4	780
DATA -1,"c",3,4 ! UNSIGNED CAPACITOR	790
DATA 1,2,0,0,0,-1.6	800
DATA 1,2,-1.6,-1.6,1.6,-1.6	810
DATA 1,2,0,-3.2,0,-4.8	820
DATA 2,0,-6.4,3.2,3.2,60,120	830
DATA -1,"D",3,4 ! DIODE	840
DATA 1,2,0,1.2,0,-1.2	850
DATA 3,4,0,0,2.4,1.2,2.4,-1.2,0,0	860
DATA -1,"d",3,4 ! ZENER DIODE	870
DATA 1,4,-0.5,1.7,0,1.2,0,-1.2,0.5,-1.7	880
DATA 3,4,0,0,2.4,1.2,2.4,-1.2,0,0	890
DATA -1,"F",3,4 ! FUZE	900
DATA 1,2,0,0,1,0	910
DATA 2,2.5,0,1.5,1.5,0,180	920
DATA 2,5.5,0,1.5,1.5,180,360	930
DATA 1,2,7,0,8,0	940
DATA -1,"f",3,4 ! FLIP-FLOP	950
DATA 1,5,-5,0,-5,20,5,20,5,0,-5,0	960
DATA 1,2,-5,3.2,-6.6,3.2	970
DATA 1,2,-5,16.8,-6.6,16.8	980
DATA 1,2,0,0,0,-1.6	990
DATA 1,2,0,20,0,21.6	1000
DATA 1,2,5,10,6.6,10	1010
DATA 2,-3,3.2,0.8,1,0,360	1020
DATA 1,2,-3,3,-2,2	1030
DATA 1,2,-2.2,4.5,-3.8,4.5	1040
DATA 2,-3,16.8,0.8,1,0,360	1050
DATA 1,2,-3,16.6,-2,15.6	1060
DATA 2,0,18.9,0.8,0.5,0,270	1070
DATA 2,0,17.9,0.8,0.5,-180,90	1080
DATA 1,3,-0.8,0.8,-0.8,2.8,0.3,2.8	1090
DATA 1,2,0.8,0.8,0.2,1.8	1100
DATA 1,2,-0.8,1.8,0.3,1.8	1110
DATA 2,0,3,2,3,0.5,0.5,-90,90	1120
DATA 2,1.8,10,0.8,1,45,315	1130
DATA 1,2,3,9,3,11	1140
DATA 1,3,4.6,11,3,10,4.6,9	1150
DATA -1,"G",3,4 ! GROUND	1160
DATA 2,3,3.2,0.8,1,0,360	1170
DATA 1,6,-1.6,-1.2,1.6,-1.2,1.6,-1.25,-1.6,-1.25,-1.6,-1.3,1.6,-1.3	1180
DATA 1,6,-0.85,-2.2,0.85,-2.2,0.85,-2.25,-0.85,-2.25,-0.85,-2.3	1190
DATA 0.85,-2.3	1200
DATA 1,6,-0.1,-3.2,0.1,-3.2,0.1,-3.25,-0.1,-3.25,-0.1,-3.3,0.1,-3.3	1210
DATA -1,"g",3,4 ! CHASSIS GROUND	1220



DATA 1,6,0,0,0,-1.2,-1.6,-1.2,0,-3.3,1.6,-1.2,0,-1.2	1230
DATA -1,"H",3,4 ! EARTH GROUND	1240
DATA 1,3,0,0,0,-1.2,-0.5,-2.2	1250
DATA 1,4,-2.1,-2.2,-1.6,-1.2,1.6,-1.2,1.1,-2.2	1260
DATA -1,"i",3,4 ! INTEGRATED CIRCUIT (SMALL)	1270
DATA 1,4,5.5,0,0,3.2,0,-3.2,5.5,0	1280
DATA -1,"I",3,4 ! INTEGRATED CIRCUIT (LARGE)	1290
DATA 1,4,11,0,0,6.4,0,-6.4,11,0	1300
DATA -1,"K",3,4 ! RELAY SOLENOID	1310
DATA 1,2,0,0,2.4,0	1320
DATA 1,5,2.4,0.8,2.4,-4,4.8,-4,4.8,0.8,2.4,0.8	1330
DATA 1,2,4.8,-3.7,-3	1340
DATA 2,4.8,-0.5,0.5,0.5,-90,90	1350
DATA 1,2,4.8,-1.2,4,-2	1360
DATA 2,2.4,-2.5,0.5,0.5,90,270	1370
DATA -1,"k",3,4 ! RELAY SWITCH	1380
DATA 2,0,0,0,2,0,2,0,360	1390
DATA 1,2,0,2,0,5,0	1400
DATA 1,3,7,-3,3.6,-3,3.6,-1.5	1410
DATA 3,3,3.3,-2,3.6,-1.5,3.9,-2	1420
DATA 1,3,7,1.5,3.6,1.5,3.6,0	1430
DATA 3,3,3.3,0.5,3.6,0,3.9,0.5	1440
DATA -1,"L",3,4 ! LAMP	1450
DATA 2,0,0.5,2,2,0,360	1460
DATA 1,2,-4,0,-1.3,0	1470
DATA 2,-1.3,1.6,1.6,1.6,270,360	1480
DATA 2,0,1.6,0.3,0.3,0,180	1490
DATA 2,1.3,1.6,1.6,1.6,180,270	1500
DATA 1,2,1.3,0,4,0	1510
DATA -1,"m",3,4 ! MU	1520
DATA 2,0,5,1,5,270,360	1530
DATA 2,2.5,5,1.5,5,180,360	1540
DATA 2,6,5,2,8,180,270	1550
DATA -1,"N",3,4 ! NPN TRANSISTOR (RIGHT EMITTER)	1560
DATA 4,0,0,3.9,3.9,0,360,4,4	1570
DATA 1,3,-4.8,1.6,-3.2,1.6,-1.6,-2	1580
DATA 1,6,-2.9,-2,2.9,-2,2.9,-1.9,-2.9,-1.9,-2.9,-1.95,2.9,-1.95	1590
DATA 1,3,4.8,1.6,3.2,1.6,1.6,-2	1600
DATA 1,2,0,-2,0,-6	1610
DATA 3,4,2,-0.4,2.65,0.4,2.5,-0.62,2,-0.4	1620
DATA -1,"n",3,4 ! NPN TRANSISTOR (LEFT EMITTER)	1630
DATA 4,0,0,3.9,3.9,0,360,4,4	1640
DATA 1,3,-4.8,1.6,-3.2,1.6,-1.6,-2	1650
DATA 1,6,-2.9,-2,2.9,-2,2.9,-1.9,-2.9,-1.9,-2.9,-1.95,2.9,-1.95	1660
DATA 1,3,4.8,1.6,3.2,1.6,1.6,-2	1670
DATA 1,2,0,-2,0,-6	1680
DATA 3,4,-2.63,0.4,-2.53,-0.46,-2.08,-0.3,-2.63,0.4	1690
DATA -1,"O",3,4 ! OPEN DOT (CENTERED)	1700
DATA 2,0,0,0,22,0,22,0,360	1710
DATA -1,"o",3,4 ! OPEN DOT (OFFSET)	1720
DATA 2,0,22,0,0,22,0,22,0,360	1730
DATA -1,"P",3,4 ! PNP TRANSISTOR (LEFT COLLECTOR)	1740
DATA 4,0,0,3.9,3.9,0,360,4,4	1750
DATA 1,3,-4.8,1.6,-3.2,1.6,-1.6,-2	1760
DATA 1,6,-2.9,-2,2.9,-2,2.9,-1.9,-2.9,-1.9,-2.9,-1.95,2.9,-1.95	1770
DATA 1,3,4.8,1.6,3.2,1.6,1.6,-2	1780
DATA 1,2,0,-2,0,-6	1790



DATA 3,4,-2.34,0.4,-2.2,-0.7,-2.9,0.15,-2.34,0.4	1800
DATA -1,"p",3,4 ! PNP TRANSISTOR (RIGHT COLLECTOR)	1810
DATA 4,0,0,3.9,3.9,0,360,4,4	1820
DATA 1,3,-4.8,1.6,-3.2,1.6,-1.6,-2	1830
DATA 1,6,-2.9,-2,2.9,-2,2.9,-1.9,-2.9,-1.9,-2.9,-1.95,2.9,-1.95	1840
DATA 1,3,4.8,1.6,3.2,1.6,1.6,-2	1850
DATA 1,2,0,-2,0,-6	1860
DATA 3,4,2.3,-0.4,2.95,0.28,2.4,0.5,2.3,-0.4	1870
DATA -1,"R",3,4 ! RESISTOR	1880
DATA 1,10,0,0,1,0,1.5,1,2.5,-1,3.5,1,4.5,-1,5.5,1,6.5,-1,7,0,8,0	1890
DATA -1,"T",3,4 ! TRANSFORMER COIL (LEFT HALF)	1900
DATA 2,0,1.25,1.25,1.25,-90,90	1910
DATA 2,0,3.75,1.25,1.25,-90,90	1920
DATA 2,0,6.25,1.25,1.25,-90,90	1930
DATA 2,0,8.75,1.25,1.25,-90,90	1940
DATA -1,"t",3,4 ! TRANSFORMER COIL (RIGHT HALF)	1950
DATA 2,0,8.75,1.25,1.25,90,270	1960
DATA 2,0,6.25,1.25,1.25,90,270	1970
DATA 2,0,3.75,1.25,1.25,90,270	1980
DATA 2,0,1.25,1.25,1.25,90,270	1990
DATA -1,"z",3,4 ! OMEGA	2000
DATA 1,3,0,0,2,0,2,1.2	2010
DATA 1,3,6,0,4,0,4,1.2	2020
DATA 2,3,4,3,3,-70,250	2030
DATA -9999	2040

---

## **Bibliography**

- Helms, Harry L. *The BASIC Book*. Peterborough, N.H.: Byte Books, © McGraw-Hill, 1983.
- Kindle, J. H. *Plane and Solid Analytical Geometry*. Schaum's Outline Series in Mathematics. New York: McGraw-Hill, 1950.
- Knuth, D. E. *Seminumerical Algorithms*. Reading, Mass.: Addison-Wesley Publishing Co., 1969.
- Muybridge, E. *Animals in Motion*. New York: Dover Publications, 1957.
- Muybridge, E. *The Human Figure in Motion*. New York: Dover Publications, 1955.
- Newman, W. M. and R. F. Sproull, *Principles of Interactive Computer Graphics*. 2d ed. New York: McGraw-Hill, 1979.
- Scheid, F. *Theory and Problems of Numerical Analysis*. Schaum's Outline Series. New York: McGraw-Hill, 1968.
- Sears, F. W. and M. W. Zemansky, *College Physics*. Cambridge, Mass.: Addison-Wesley, 1952.
- Tektronix, Inc. *Introduction to Graphics Programming in BASIC*. Manual 070-2059-01. Beaverton, Oregon: 1981.
- Tektronix, Inc. *Introduction to Programming in BASIC*. Manual 070-2058-01. Beaverton, Oregon, 1981.











