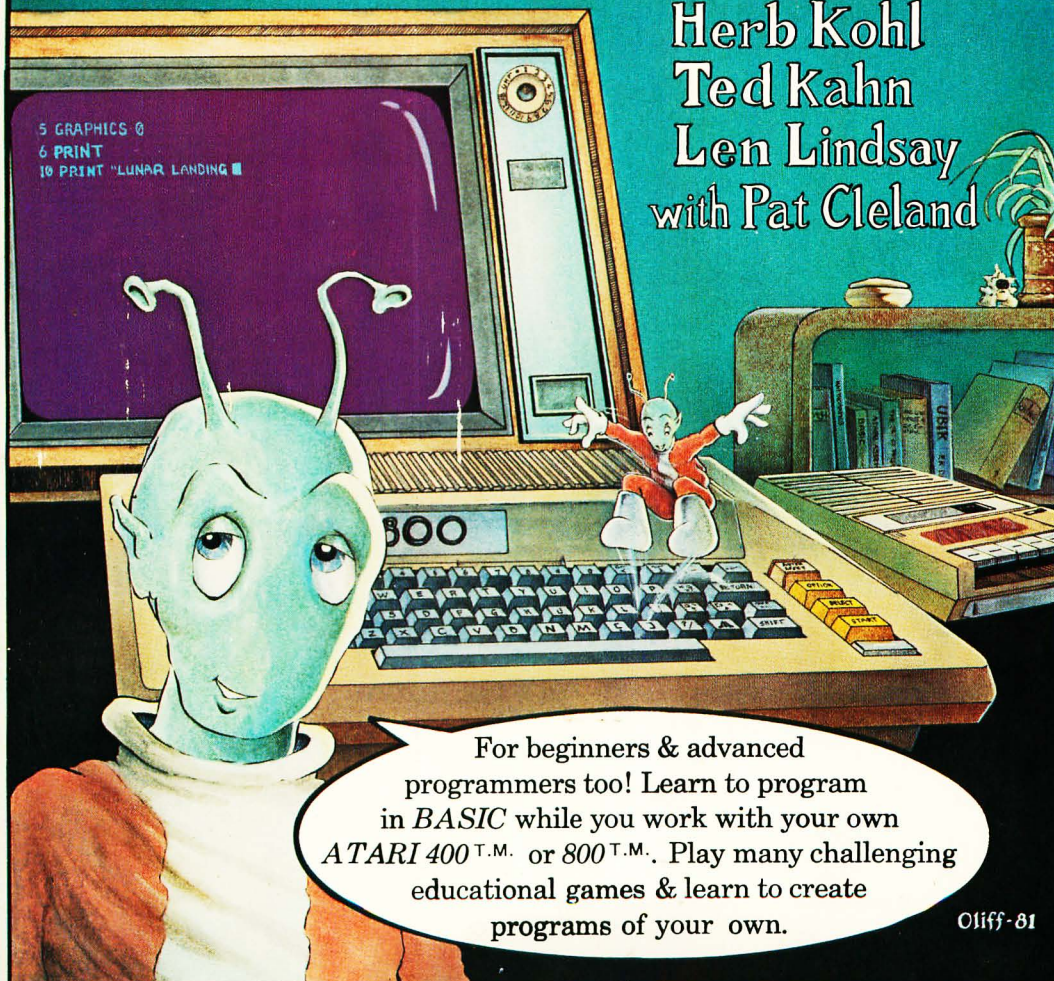


# ATARI® Games & Recreations

Herb Kohl  
Ted Kahn  
Len Lindsay  
with Pat Cleland



For beginners & advanced  
programmers too! Learn to program  
in *BASIC* while you work with your own  
*ATARI 400*™ or *800*™. Play many challenging  
educational games & learn to create  
programs of your own.

Oliff-81





# ATARI®

# Games and Recreations

Herb Kohl  
Ted Kahn  
Len Lindsay  
with Pat Cleland

Illustrated by Steve Oliff



Reston Publishing Company, Inc.  
*A Prentice-Hall Company*  
Reston, Virginia

Library of Congress Cataloging in Publication Data

Kohl, Herbert R.

Atari games and recreations.

Includes index.

1. Video games—Data processing. 2. Atari computer—Data processing. 3. Basic (Computer program language) I. Kahn, Ted M.

II. Lindsay, Len. III. Title.

GV1469.3.K63 794

82-7629

ISBN 0-8359-0298-6

AACR2

ISBN 0-8359-0242-0 (pbk.)

ATARI is a registered trademark of Atari, Inc. (a Warner Communications Company), used by permission.

© 1982 by Reston Publishing Company, Inc.

*A Prentice-Hall Company*

Reston, Virginia 22090

*All rights reserved. No part of this book  
may be reproduced, in any way or by any means,  
without permission in writing from the publisher.*

10 9 8 7 6 5 4 3

Printed in the United States of America



# Contents

## **PART I      LEARNING ABOUT THE ATARI HOME COMPUTERS THROUGH GAMES   1**

### **Chapter 1      Communicating With Your Computer   3**

Making Mistakes and Correcting Them, 13

The Elusive and Powerful RND(1), 22

Using RND(1): Some Loops and a Branch Will Make a Game, 26

The Game at Last: Guess a Number, 33

### **Chapter 2      Theme and Variations: An Introduction to the Fine Art of “Dressing Up”   37**

The Tortoise and the Hare at Last—Almost! 47

The Race: Versions 1, 2, and 3, 50

Dressing Up Game Programs, 64

Getting More Complex, 72

### **Chapter 3      Drill and Basic Skills   77**

Number Facts, 78

Spelling Drill and a Little Grammar: Using ‘String Variables,’ 94

BASIC Vocabulary Introduced in Part I, 104

<b>PART II</b>	<b>GAMES FOR THE ATARI HOME COMPUTER 107</b>
<b>Chapter 4</b>	<b>Number and Logic Games 109</b>
	Guess the Number—Four Variations, 111
	Getting to 100—Three Versions, 116
	NIM, 119
	Dice, 126
	Thrice Dice, 128
	Reverse, 132
	QWERT, 136
<b>Chapter 5</b>	<b>Word and Guessing Games 159</b>
	A Simple Word-Guessing Program, 161
	Meditation With Word Imagery, 163
	A Three-Letter Word Dictionary and Three-Letter Word Games, 165
	Using Your Dictionary, 169
	Three-Letter-Word Identifier for ATARI, 173
	Series, 175
	Secret Codes, 181
	Anagrams, 185
	Fractured Tales, 188
	Fortune Teller, 194
	The Oracle, 195
	Your First Computer Horoscope Program, 196
	Silly Questions, Silly Answers, 198
	Color Preference, 200
	Some Useful Subroutines for Word and Guessing Games, 217
<b>PART III</b>	<b>THE ATARI SPECIAL 219</b>
<b>Chapter 6</b>	<b>Graphics 221</b>
	Some Simple Plotting Games, 223
	Rectcent or Guess the Center, 226
	Animation, 229
	Moving Messages, 233
	Some Geometric Experiments, 236
	Straight Lines or There's More in Mathematics Than Meets the Eyes, 241
	High Resolution Drawing, 245
	Text and Character Graphics, 249
<b>Chapter 7</b>	<b>Sound and Music 253</b>
	Pitch, 256
	Voice, Distortion and Volume, 262
	Chord Builder, 265



Color and Sound Combinations, 268  
 Jazz—Combining Color and Sound, 270  
 The Terrors of War (Sights and Sounds), 273

## Chapter 8

### Color 275

Color Dressing for Number Guessing, 281  
 A Random Color Mosaic, 281  
 A 3-D Color Cube, 284  
 Triangle—A Simple Demonstration of Color Plotting, 285  
 Modern Art à la Josef Albers, 286  
 Using Paddle Controllers to Play With Color and Sound, 288

## Appendix I

Errors and Error Messages: Helping You “Debug” Your Program 289

## Appendix II

ATARI BASIC Cards—Reference Guide 297

## Appendix III

ATARI Pokes 313

## Appendix IV

Clocks, Time, and Timing Input 317

## Appendix V

A Scorekeeper Subroutine for More Than One Player 321

## Appendix VI

Screen Design Sheets for GRAPHICS Modes 0-8 323

## Appendix VII

ATASCII Character Codes 331

Index 335





# Introduction

There are many different ways to approach your ATARI® 400™ or ATARI 800™ Home Computer. You can play with it; create programs of your own; and explore the keyboard, graphics, sound and color control built into the machine. Another way is to copy programs that others have written for you and to play them.

In this book we will provide many games that you can copy and play on your ATARI computer. We will also show you ways to play with your computer, to make up your own programs and games, and to dress up your games so that they look good on the machine.

Dressing up games so that their presentation is interesting and so that some humor and character can be integrated into programs is just part of what might be called the personality of computing. Every computer has its own personality, and every person who spends time working with computers also develops a style and personality that characterizes his or her programs. There are usually many ways to write a given program and many different ways to display it on the screen. Given the flexibility of computers and programming you should allow your own style to emerge and not worry about copying others or making mistakes. Approaching a computer is like learning a new language. The mistakes you make will contribute to your ultimate mastery of the computer and its capabilities.

Part of this book is for beginners. Each chapter in Part I will contain




ATARI is a registered trademark of Atari, Inc. (a Warner Communications Company), used by permission.

an exposition of ideas and commands used in most of the programs presented. However, at the end of each chapter there are sophisticated programs that should interest experienced programmers. As you read through the book, we will explain many of those aspects of computer language that are used in the programs and exercises presented. We will also illustrate some basic concepts such as *looping* and *branching* that will enable you to develop complex programs. It is possible to use this book as a basic learning text as well as a games book. In fact, much of the material in the text can be used on other personal computers to master the central concepts and vocabulary of BASIC.

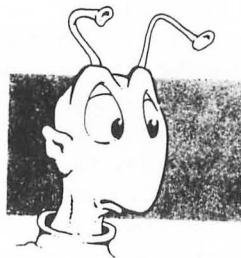
The ATARI computer is particularly suited to be approached through games, as it grew from games like "Pong <sup>TM</sup>" and provides you with the capacity to program complex games controlled by paddles and joysticks. You can buy packaged games programs for your computer and paddles and joysticks, as well. You can also, with practice, program games of your own that use joysticks and paddles and are as challenging as those commercially available.



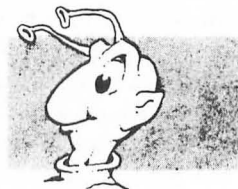
Here is a picture of a joystick. You might have noticed that there are two puzzled creatures sitting on it, wondering whether to play "Star Raiders <sup>TM</sup>" or make up their own games. 

With the assistance of Steve Oliff who has worked for *MARVEL* and other comic book publishers on many projects, we have conjured this parent and child (we don't know their sexes nor do we have the slightest idea what sexes exist in the part of space they come from). According to Steve, the parent's name is Ta\*ri and the child is Ta\*ri, Junior.

Contrary to common assumption about the sophistication and advanced technological state of extraterrestrial beings, Ta\*ri and Junior



Ta\*ri



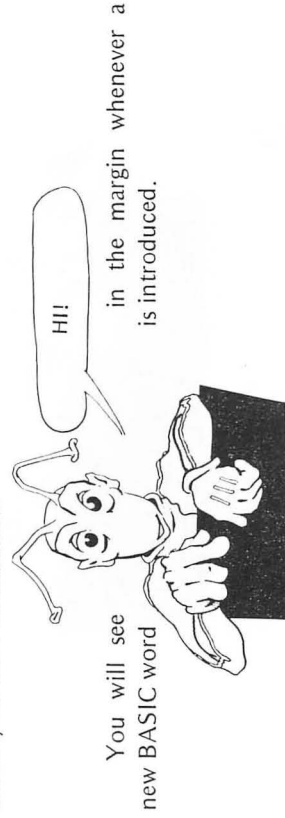
Ta\*ri Junior



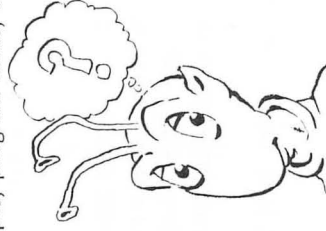
Star Raiders and Pong are registered trademarks of Atari, Inc.

know less about computers than you do. Delta Drama 2, the galaxy they come from, is more advanced than ours in some ways. Teleporting themselves through space and time has never been a problem. The Dramarians have spent all their energy on the development of internal powers of communication and are just beginning to discover computers. Ta\*ri Junior and her dad decided to teleport to Earth and explore the world of computers.

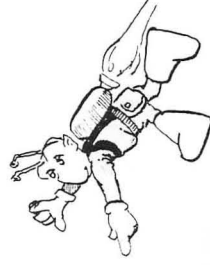
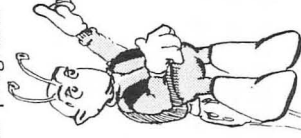
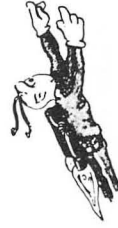
Throughout the book we will use them to call your attention to new concepts, critical points in programs, explanations of programs and to places where it would be easy for you to vary programs we present and create your own innovations.



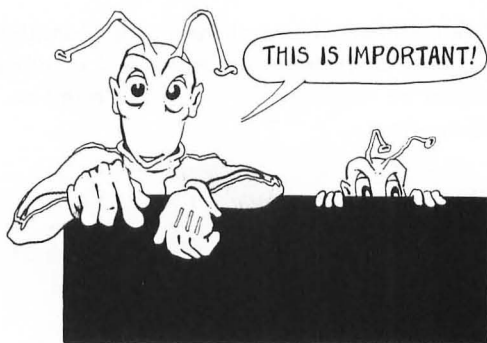
Explanations of concepts, program ideas, or computer structure will be indicated by:



There are some parts of programs that are more critical than others. It is often difficult for people learning about programming to separate these points from directions, comments, or inessential elaborations of programs. Ta\*ri Junior will indicate critical points in programs in the following ways:



In addition to critical points, there are places in computer programs where one has to be extra careful not to make mistakes. Sometimes you have to introduce new data or refer back and forth to different parts of a program. It is easy at these and other points to make mistakes. We'll indicate where you have to *watch out for mistakes* this way:



There are also many places where you might want to change a program or introduce a new program idea. Good places for this kind of innovation and creativity will be indicated by Ta\*ri.

Places where concepts and words in BASIC are reviewed will be pointed out in this way:



We hope that Ta\*ri and Junior will help you navigate your way through our programs and make it possible for you to program comfortably and creatively.

Here are some additional pointers to help you:

There are several special keys on the keyboard. Some examples are [RETURN], [SHIFT], [BREAK], and [CLEAR]. The special keys will be written in upper-case letters and enclosed in brackets.

Sometimes it is important to leave a space as a part of your program; to do this just press the space bar on the keyboard. A small solid square ■ is used in the program listings in Chapter One to indicate places where spaces are needed within a program. By Chapter Two you will be able to spot needed spaces without this reminder.

Ta\*ri's boot is used to reference footnotes at the bottom of a page. Since the asterisk is used in programming to indicate multiplication, Ta\*ri's boot eliminates unnecessary confusion.



Different types of games are presented in this book. In Part I, we'll present a number of introductory games as well as the fabulous race between the tortoise and the hare. We'll also include a section on games that can be used to help your children learn the basic skills of arithmetic and spelling. You can set up a spelling test or create some math drills, and you can easily teach your child to do the same.

In Part II we will also present language guessing games with an emphasis on games that are slightly silly and fun to play as well as some number games. These games provide ways to use your computer as a party toy for children and adults. Included are fortune telling games, riddles, jokes, and mystery games.

Part III will deal with the special graphics, sound, and color aspects of the Atari Home Computer. It will show you how to draw graphs, do simple animated cartoons, compose songs, add sound and color to your games, and mix all of the different modes of the computer to create multimedia games and performances.

The appendices of the book have a number of useful charts, flash cards, and graph-paper designs that should be useful when you try to play with your ATARI computer. The error dictionary will help you deal with programming errors that the computer can pick up. It is important as you begin to program for yourself to understand that errors are so common (even among pros) that the machine has programs built into its logic that tell you when you make common errors and also give you indications of how to correct them.

In order to play the games in this book, you will need either an ATARI 400™ or an ATARI 800™ home computer, equipped with the power supply transformer box and television interface box (which are standard equipment for Atari home computers) and an ATARI BASIC programming language cartridge. Any standard black and white or color television set may be used with your ATARI Home Computer; however, many of the games in this book make use of color features of your ATARI Home Computer and will *look* better on a color set. Some of the games in

this book require a minimum of 16K of RAM (Random Access Memory); however, most of the games may be programmed with only 8K of memory. It is also recommended that you have a copy of the *Atari 400/800 BASIC Reference Manual*, as well as a copy of the *Atari BASIC* book.

In order to save the games you program so that you can replay them in the future, you will also need either an Atari 410™ Program Recorder (which uses standard tape cassettes) or an Atari 810™ Disk Drive with either the 810 Master Diskette or 810 Master Diskette II. (The Atari 810 uses standard 5 1/4" floppy diskettes.)

For a few of the games in this book, you will also need either Atari CX40 Joystick Controllers or Atari CX30 Paddle Controllers. These controllers add a great deal to the "playability" of many games and are a wise investment for building up your system's entertainment value.

It is not necessary to read this book sequentially. If you have experience with computers, you can start anywhere and use those chapters that challenge you. If you are a newcomer to the machine, it is best to begin with Part I and then dip in and out of the book and explore the various types of games the computer can run. We urge you to play as many games as you can, as they will familiarize you with the machine. However, we hope that even more than playing on your ATARI computer, you will play *with it* and take power over it. Ultimately, *you* are the intelligence of your computer. The more you know and the more you are willing to experiment, the more interesting things you can do with your ATARI Home Computer. The computer can be useful, fun, and even compassionate if *you* make it so!

# Acknowledgments

The authors would like to thank the following people, who made this book possible: Nikki Hardin, our editor at Reston; Bob Albrecht and the people at DYMAX; and Barry Vincent, who helped in editing some of the programs. We would like to express special thanks to Dale Disharoon, who tested all of the games for us and did most of the program editing and formatting in the book; and to Steve Oliff, who drew the magnificent artwork of Ta\*ri and friends. Many thanks also to Judy and Frona for their patience; and to Rhianon, a beautiful four-year-old who did some play testing for Len Lindsay. Last, but not least, thanks to Peter Rosenthal and the Home Computer Division of Atari, Inc. for supplying ATARI 800™ Home Computers to the authors while their work was in progress.





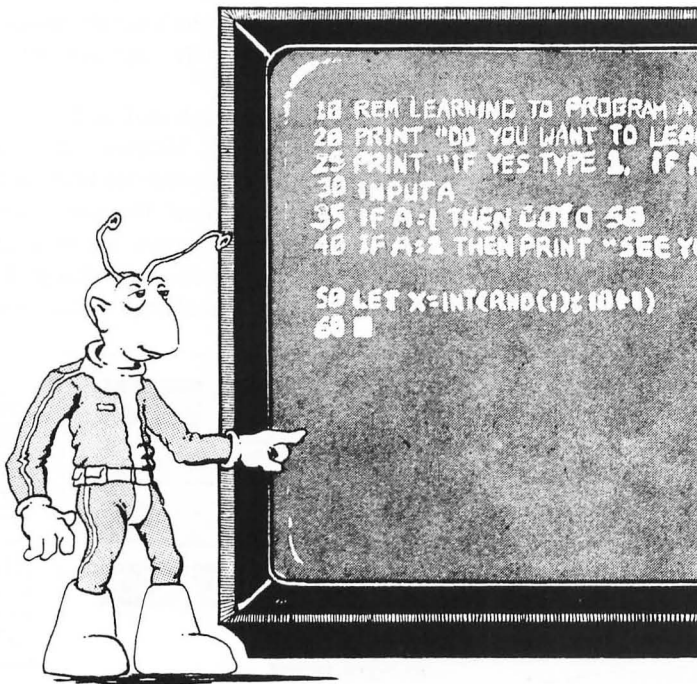
Part I

# Learning About the ATARI Home Computers Through Games



# Chapter 1

## Communicating With Your Computer





The first thing to remember about your computer is that it takes orders and follows directions. You tell it what to do. If your instructions are interesting, it can do things that would be very difficult for you to do with a paper and pencil. If your instructions are faulty, it will likely have a nervous breakdown and commit errors that you will have to correct. There are people who dream of building a self-correcting computer, but that is only a dream at present.


In order to give the computer directions, you and the machine have to communicate in a common language. One common language of this computer and the people that use the machines is called *BASIC*. Learning ATARI BASIC is no more difficult than learning a few new words in a foreign language or mastering a vocabulary or spelling list in school. In this book, we will not be using all of the words in ATARI BASIC. In fact, we will be using fewer than thirty words. The power of the computer is that with a very small vocabulary it can do many sophisticated and complex things. When you have mastered the vocabulary introduced here, you will find yourself beginning to talk and think BASIC. Mastering the vocabulary is a way of learning how the computer functions, and as you understand this, you will be able to take on increasingly complex programming problems.

Your ATARI Home Computer is a flexible system which may be expanded to include several parts. The most important one is the keyboard and console unit, which includes the computer's central processing unit (CPU) and memory. You communicate with your ATARI computer through this keyboard and console unit.

The keyboard is much like a typewriter keyboard and has many features that are similar to ordinary typewriters. However, there are a number of special features that are built into the computer keyboard and we will introduce you to many of them throughout the course of this book. One special feature is that numbers and letters are never interchangeable. With a regular typewriter, you can freely interchange 1 (the number) and l (the letter) or 0 and o. This interchange would confuse



the computer. Remember, *for most computer programs, numbers and letters are two totally separate realms.*

Now to BASIC. The first two words to learn are RUN and LIST.  RUN instructs the computer to do the program that is in its *memory*. The program could be a game or a tax computation schedule or an animated cartoon. The current program in the computer's memory will execute when you type RUN and press the [RETURN] key. If you turn on the machine now, type RUN, and press the [RETURN] key, nothing will happen because there is nothing in the program memory when you start out. You have to put something in the memory.

When you type LIST and press the [RETURN] key, the computer will print on your TV screen the contents of its program memory. If you type LIST now you will not see anything, since you have not put anything in the memory. The TV will simply say READY, indicating that you can begin to program whenever you are ready.

In order to put a program into the memory of your ATARI computer, you have to give each line you write a numerical name or label. That label is called a *line number*. You can start with any line number you like (as long as it's an integer from 0-32767). As you continue to write your program, you may have line numbers out of order; however, your computer will reorganize your program so that lines are all in numerical order. Remember: *No matter what order you enter the lines of your program, the computer will order all the line numbers from the smallest to the largest and will follow the program in that order.*

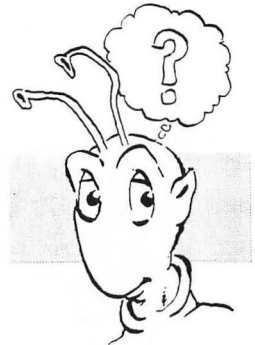
For example, if your lines were entered in this order:

```
100
50
10
250
```

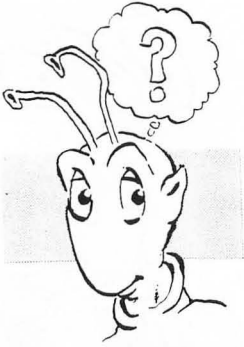
They would be reorganized and followed in this order:

```
10
50
100
250
```

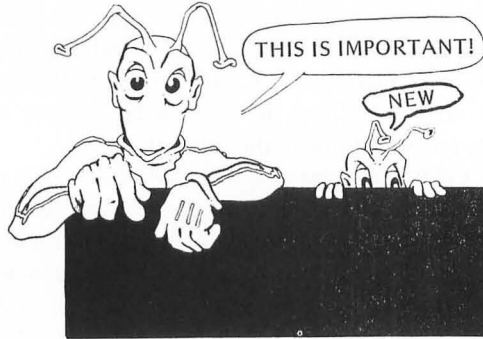
If you do not use a line number for a BASIC command, your ATARI computer will try to perform the BASIC command *immediately*, and the command will not be stored in the program memory. For example, the commands RUN and LIST are not preceded by numbers; they are "executed" as soon as you finish the line by pressing the [RETURN] key. The



All ATARI BASIC commands are always written in upper-case letters.



fact that you can try out most ATARI BASIC statements in this *immediate mode* is a convenient way to learn new BASIC commands. It is also very useful in tracking down programming errors, as we will see later. However, if you want your computer to remember a BASIC program (even if it's only one line long), you must use line numbers for each line.



Once you put a program into the memory of your computer it will remain there unless: 1) you turn the computer off; 2) you type the BASIC command NEW and then press the [RETURN] key.

It is important to remember these two facts. People who are new to personal computers sometimes forget that turning off the machine wipes out the memory. If you are in the middle of a program and want to save it, use your ATARI 410 Program Recorder or ATARI 810 Disk Drive. (See the ATARI Operator's Manuals for instructions.)

Finally, remember that by using the NEW command you can clear the memory of your machine any time you want and write a new program.

PRINT is a command that tells the computer to print something on the TV screen. It's almost as simple as that. If you want the machine to print your name, type PRINT and put your name in quotes after the command. Then press [RETURN].



PRINT "YOUR NAME" [RETURN]

Use quotes in the PRINT command when you have something you want the machine to reproduce. Try to have the computer PRINT this sentence: I KNOW HOW TO PRINT THIS SENTENCE.

After typing: PRINT "I KNOW HOW TO PRINT THIS SENTENCE" remember to press the [RETURN] key. The computer will execute your command after [RETURN] is pressed. Did you remember the quotation marks?

The PRINT command works in a different way if you leave out the quotes. When there are no quotes, the PRINT instruction tells the computer to act like a calculator. PRINT "3+3" will cause the machine to print 3+3. However, PRINT 3+3 will cause the machine to print the result, 6.

See what happens when you tell the machine to PRINT the following:

```
PRINT "23+45" [RETURN]
PRINT 23+45 [RETURN]
PRINT 45*45 [RETURN]
PRINT "45*45" [RETURN]
```



If you now try to PRINT words or letters and forget the quotes, all you will get is 0 (not o) as the computer assumes a value of 0 unless told otherwise.

Try this:

```
PRINT CAT [RETURN]
PRINT "CAT" [RETURN]
```

Now tell the machine to PRINT "I AM JUST A MACHINE" and press [RETURN]. You will get your sentence. What happens if you instruct the machine to RUN?

Nothing! That is because you have not given a name (line number) to the line with the PRINT instruction, and so it did not go into the machine's memory. Here is the instruction with a line number added:



```
10 PRINT "I AM NOTHING BUT A MACHINE"
```

Now, if you press [RETURN], type RUN, and then press [RETURN] again, the machine will follow your instructions. In fact, each time you type RUN and press [RETURN] the computer will print the line. *It remembers your instructions and follows your directions.* That is the essence of *programming*.

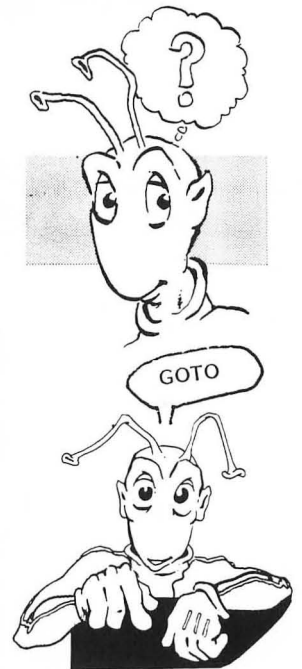
If you now type LIST and press [RETURN], the one-line program you have just written will be printed on the screen. *Remember, RUN makes the program work, and LIST shows you what the program is.*

Now for a new word and a new concept. The word is GOTO. GOTO instructs the computer to *go to* a specific line in the program and do what the line says and then continue with the next line. GOTO can set up a *loop*, one of the most powerful things that can be done with a computer. Look at the following program:

```
10 PRINT "I AM NOTHING BUT A MACHINE"
20 GOTO 10
```

This simple program says to print the sentence "I AM NOTHING BUT A MACHINE" and then go on to line 20. But line 20 says go back to line 10 and do that again.

The program loops back because of the GOTO instruction. In fact,





RUN the program and what happens? The sentence is repeated over and over and over and over. Unless you stop the program it will run until your TV or computer burns out. You have created an *infinite loop* which you can break simply by pressing the [BREAK] key on your computer keyboard. If you want to restart the program from where it left off, type CONT (for continue).

Now let's make this simple loop a bit fancier. By adding a *semicolon* (;) to the end of the PRINT instruction, your ATARI computer will print each occurrence of the sentence alongside the previous one instead of going to the next line each time. This way you will fill up the whole screen; however, the sentences will be run together like this:



I AM NOTHING BUT A MACHINE I AM NOTHING  
BUT A MACHINE I AM NOTHING BUT A MACHI  
NE I AM NOTHING BUT A MACHINE I AM NOTHI

If you add a space after the last word of your sentence and place the quotes like this:

```
10 PRINT "I AM NOTHING BUT A MACHINE ";
```

You will get more readable spacing and the sentence will be repeated.

I AM NOTHING BUT A MACHINE I AM NOTHIN  
G BUT A MACHINE I AM NOTHING BUT A MAC  
HINE I AM NOTHING BUT A MACHINE I AM N



The reason for this is that anything inside the quotes (including blank spaces) will be reproduced by the computer. If you ever are depressed and feel that no one knows your name, here is a program to try in order to fill up your screen with name recognition:

```
10 PRINT "YOUR NAME ";  
20 GOTO 10
```



Now here is a more complex loop: Read the program and try to figure out what it will do on the screen before typing it into the machine and RUNNING it. It is a good idea to get into the habit of visualizing your programs and guessing what they will look like. Later on you will find this habit useful in correcting complicated programs that don't do exactly what you want them to do.



With ATARI BASIC, your ATARI Home Computer is initially set up to display 38 characters on each line before automatically moving to the next line on the screen.



```
10 PRINT "I DON'T LIKE SPINACH■";
20 PRINT "BUT POPEYE DOES■";
30 GOTO 10
```

Here is one more simple change to try. Replace the semicolon (;) at the end of each line with a comma (,). Then run the program. Your program should look like this:

```
10 PRINT "I DON'T LIKE SPINACH■",
20 PRINT "BUT POPEYE DOES■",
30 GOTO 10
```

Like the semicolon, the comma is a spacing device. It moves to the next tab stop on your ATARI computer before printing the next line. Try using a semicolon on one line and a comma on the next line. How does this change what you see on the screen?



So far you should be familiar with the following concepts: BASIC words, special keys and delimiters (the symbols such as quotes which tell the computer how to execute a command).

## Concepts

**Program**—A series of instructions put in the computer's memory that determines what the computer is to do.

**Line Number**—A positive integer (less than 32768) that begins any "logical line" of a program. These numbers instruct the computer to store the information on that line in the memory. No matter what order you type line numbers in your program, the computer puts them in order from smallest to largest. When a program is executed, the computer begins at the lowest line number and runs through the instructions in increasing order.

**Logical line**—What you type from the first digit of the line number until you press the [RETURN] key. This may appear on the screen as two or more "visual lines," if it is more than 38 characters long.

**Loop**—A loop exists in a program when the computer is instructed at one line to go back to a previous line and execute a command or sequence of commands. An infinite loop occurs when a command sequence is executed again and again indefinitely with no instruction in the program to stop the loop at a particular point.

## BASIC Vocabulary

- RUN**—An instruction to execute a program in the computer's memory.
- LIST**—An instruction to list all the program lines in the computer's memory.
- READY**—This word appears on the screen to indicate that the computer is ready for you to enter, LIST, or RUN a program.
- PRINT**—This command functions in two ways: 1) Without quotes, PRINT followed by an arithmetic expression turns the computer into a calculator. Upon pressing the [RETURN] key the answer to your arithmetic problem will be printed. 2) PRINT followed by anything in quotes instructs the computer to print whatever is contained within the quotes.
- NEW**—This command followed by a press of the [RETURN] key clears the memory of the computer, makes it new.
- GOTO**—This instruction at one line instructs the computer to execute the line number after the GOTO command next. Thus, GOTO 100 means the computer executes line 100 next and continues from there. Using GOTO the computer can be instructed to skip backwards and forwards in a program. This:

```
10 PRINT "X"
20 PRINT 2+2
30 GOTO 10
```

instructs the computer to skip back and execute line 10.

- CONT**—See listing under the [BREAK] key for explanation.

## Special Keys

- [RETURN]**—The [RETURN] key is pressed when you are done with a line and want to enter it into memory and go on to the next line or when you have entered an instruction like LIST or RUN and want the computer to begin to execute your instructions.
- [BREAK]**—The [BREAK] key stops a program or the listing of a program as soon as you press it. If you want to continue the execution of a program, type the BASIC word CONT and then press [RETURN].

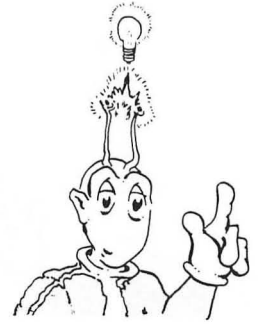
## Delimiters

- Quotation Marks**—" " are used with the PRINT command and instruct the computer to print whatever is between the quotes.

**Semicolon**—The ; indicates that if several things are to be printed they should follow directly after each other across the screen.

```
Without ; I AM THINKING
          I AM THINKING
          I AM THINKING
```

```
With ; I AM THINKING I AM THINKING I AM THINK
      ING I AM THINKING I AM THINKING I AM T
```

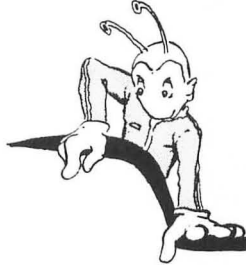


Try to create a number of programs of your own using the vocabulary and concepts already developed. Here are a few simple programs to get you started. Try to conceptualize the programs and *imagine* what they will do when RUN before you actually program and RUN them yourself. Then change the program, make your own variations, and play with the few simple words of BASIC we have introduced so far.

*Notice that small changes in a program can make major differences in what the computer does!*

Program 1A:

```
10 PRINT "I DON'T WANT TO■"
20 PRINT "STAY■"
30 GOTO 10
```



Program 1B:

```
10 PRINT "I DON'T WANT TO■"
20 PRINT "STAY■";
30 GOTO 10
```

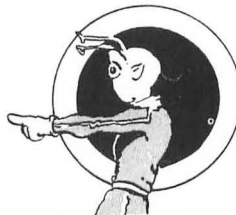


Program 2A:

```
10 PRINT "*** _ _ ***"
20 PRINT "** _ _ * _ _ *"
30 GOTO 10
```

Program 2B:

```
10 PRINT "*** _ _ ***",
20 PRINT "** _ _ * _ _ *",
30 GOTO 10
```





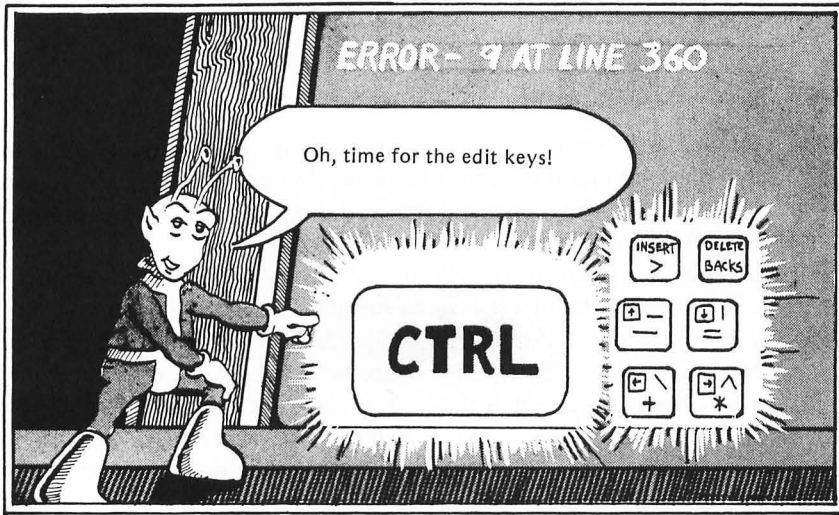
Experiment with simple GOTO loops using numbers, symbols, and spaces. Whenever you want to study the pattern you have created, press the [BREAK] key and the pattern will be frozen on the screen. Or see how to use [CTRL]-1 on page 00).

Here is a slightly more complicated pattern to study:

Program 3:

```
10 PRINT "1111111";  
20 PRINT "0000000";  
30 PRINT "@@@@@@@";  
40 GOTO 10
```

Make your own, and try to imagine a pattern and see if you can reproduce it on the screen.



## MAKING MISTAKES AND CORRECTING THEM

Don't be afraid of making mistakes as you learn to program your ATARI Home Computer. Experiment with different combinations of commands and numbers and letters and symbols. The more you experiment, the more familiar you will find the keyboard and the easier programming will become.

Even the most advanced programmers make mistakes. Sometimes the mistakes are simple, like typing TRINT instead of PRINT or GOSO instead of GOTO. Sometimes they are more complex and involve giving the computer commands it cannot execute such as:

```
10 PRINT "HELP"
20 GOTO 15
```

(The computer cannot do this, since there is no line 15.)

A number of special features have been built into your ATARI Home Computer and ATARI BASIC to help you understand the mistakes you make and to correct them. One feature is an *error detector* that shows you where you have made an error and in some cases tells you the kind of error you've made. The other feature is the *editing function*, which makes it easy to correct these errors.

In order to understand these two features of the ATARI Home Computer, let's make some deliberate errors and see what happens. Start with a simple typing error and tell the machine to:

```
PRINT "TOO BAD"
```



Then press the [RETURN] key. The machine will then write:

```
ERROR- PRINT □TOO BAD"
```

Notice that there is a square surrounding the first quote. This indicates that the machine cannot do what you commanded, and so you should look over your command for some simple error (in this case typing PRINT instead of PRINT).

Now *clear the screen* and let's make a slightly more complex error. To clear the screen without erasing the memory as you do when you type NEW and press [RETURN], press the [CLEAR] key while holding down either [SHIFT] key. The *cursor* will appear on the upper left-hand corner of your screen. The cursor is the little square that always tells you where the computer will print the next letter or symbol you type.

Try this program with a built-in error. (First be sure you have cleared the memory by typing NEW.)

```
10 PRINT "ANOTHER ERROR"
20 GOTO 11
```

Remember to press the [RETURN] key after each line. Try to RUN the program by typing RUN and pressing [RETURN] once again. The computer will be unable to do what you instructed and will print out two things. First it will print:

```
ANOTHER ERROR
```

executing line 10. Then it will print:

```
ERROR- 12 AT LINE 20
```



Don't panic! ERROR-12 AT LINE 20 means that at line 20 you referred to a nonexistent line in your program. *ERROR 12*, as you can see by looking at the ERROR messages listed in your Atari Basic Reference Manual, is a clue to you that the line your GOTO command referred to could not be found in your program. If you could correct the program to read:

```
10 PRINT "ANOTHER ERROR"
20 GOTO 10
```

the error would disappear, and the program would run. All the *Error Messages* help you figure out what has gone wrong. In Appendix I you will find information on "Errors and Error Messages: Helping You 'Debug' Your Program." We've listed and explained the most common Error

Messages, and in your *ATARI BASIC Reference Manual* there are explanations for others.

If you ever feel bad about making an error or two, just think about the fact that over 50 Error Messages were built into your computer; no matter how experienced a programmer is, one always expects to make mistakes and can use a little help.

Let's go back to the program with ERROR-12 AT LINE 20. Press the [SHIFT] [CLEAR] keys to clear the screen and then list the program again by typing LIST and pressing the [RETURN] key. The screen should now read:

```
10 PRINT "ANOTHER ERROR"
20 GOTO 11
```

Now take a look at the keyboard of your computer. There are a number of special keys that allow you to change a program, add or delete things from the middle of a program, and correct errors. These are called the *editing functions* and are controlled by the key just above the left [SHIFT] key. It is labeled [CTRL] and stands for *control* key.



The control key is used with other keys on the ATARI keyboard to move the cursor (the square on the TV screen that indicates where the computer will print the next letter or symbol you type). Six special keys that are used with the [CTRL] key for editing are indicated in black and white and are all on the right side of the keyboard.

Notice the four arrows on the special keys located at the right of rows two and three on the computer keyboard. The arrows indicate the direction they move the cursor. Try this experiment. Press down and hold the

[CTRL] key and at the same time press the [→] key. The cursor will move to the right. Now press [CTRL] and [↑]. The cursor will move up. After experimenting for a while with the four arrows and the [CTRL] key, bring the cursor to the second 1 in 11 of your program:

```
10 PRINT "ANOTHER ERROR"
20 GOTO 11
```



Now all you have to do is type a 0 and press [RETURN] and your program will be corrected to read:

```
10 PRINT "ANOTHER ERROR"
20 GOTO 10
```



Clear the screen by pressing [SHIFT] [CLEAR] and then LIST and RUN the program. You will find the error eliminated.

One of the special features of your ATARI keyboard is that each of the keys has a "rapid repeat" feature. If you press and hold down the X key, for example, you will get repeating Xs all across the screen. This feature may be used to speed up editing with the cursor control functions as well. For example, try the following:

Press and hold down the [CTRL] key, and with your other hand, also press and *hold* the [→] key. This is a quick way to relocate the cursor on the same line. Now, do the same thing with the [↑] key. This is a shortcut for relocation of the cursor key on a different line.

There are two more special keys that can be used with [CTRL] for editing your program. They are the [INSERT] and [DELETE] keys found on the upper right-hand side of the keyboard. These keys do just what they imply—insert and delete spaces or symbols from your programs.


Starting with the corrected program:

```
10 PRINT "A NOTHER ERROR"
20 GOTO 10
```

Bring the cursor to the A in ANOTHER as illustrated above. Now holding the [CTRL] key down, press the [INSERT] key four times. You will have added four spaces after the cursor. You can now type whatever you like in these spaces and change your program. Here is one possibility:

```
10 PRINT "NOT ANOTHER ERROR"
20 GOTO 10
```



After you have added something to your program, let's illustrate the use of the [DELETE] key.  You can eliminate what you have added by bringing the cursor to the N in NOT and pressing both the [CTRL] and [DELETE] keys at the same time. Each time you press [CTRL] [DELETE] one letter will be swallowed, and the rest of the line will move one space to the left.

Here are some exercises in editing that should help you become comfortable with [CTRL] and its related keys.

Clear the memory of your ATARI computer, and type in the following program:

```
10 PRINT "I AM TIRED"
20 GOTO 10
```

Now change the phrases "I AM TIRED" to the following phrases:

```
"I AM NOT TIRED"
"I AM NOT TRYING"
"I AM TRYING"
"IT IS TRYING"
"IT IS NOT TIRING"
```

Clear the memory once again, and type in this program:

```
10 PRINT "TA*RI";
20 PRINT "TARI■"
30 GOTO 10
```

Run the program. Now LIST the program, and experiment with editing it in the following ways.

Change line 10 in these ways:

```
10 PRINT "***+**■";
10 PRINT "**_**■";
10 PRINT "+*+*_*+*■";
```

Change line 20 and combine with some version of line 10:

```
20 PRINT "STARWAR■";
20 PRINT "STAR-WAR■";
20 PRINT "STAR-STARE■";
```



In case you hadn't noticed, the [DELETE] or [BACKSPACE] key can be used without the [CTRL] function as an eraser when you are writing your original program. Pressing this key erases the previous character. Holding down the key will erase a number of characters.



Change line 30 in combination with some of the above changes to:

```
30 GOTO 20
30 PRINT "THE END"
```

There is another editing function which adds and deletes lines instead of spaces. To see how this works type the following into your computer and bring the cursor to rest on the 2 in line 20 (you should be able to do this with the editing functions already described):

```
10 PRINT "*STAR*";
20 PRINT "+STAR+";
30 GOTO 10
```



Then press [SHIFT] and [DELETE] at the same time. Line 20 will disappear altogether. You have done *line editing* instead of letter, symbol, or space editing. If now you press [SHIFT] and [INSERT] you've inserted a line where you can add a new line 20. Thus:

```
[SHIFT] [DELETE] deletes a line
[SHIFT] [INSERT] adds a blank line
```

In this program delete line 30:

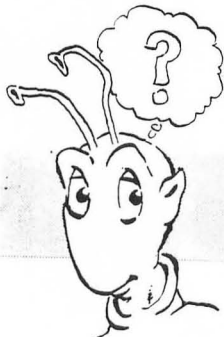
```
10 PRINT "+*";
20 PRINT "??";
30 PRINT "NOYESNO";
40 GOTO 10
```

Now insert a new line 30 that reads:

```
30 PRINT "SURPRISE . . I'M BACK!";
```

There is also another way to delete a line of your program. Simply type the number of the line you wish to delete and press [RETURN].

```
30 [RETURN]
```



And, now, for an editing trick that will save you a lot of time in some of your future programs. Introducing: "The Amazing Multiple Line Trick." Type NEW and press [RETURN], and then enter the following one-line program:

```
10 PRINT "THIS IS MY ONLY LINE" [RETURN]
```

Now, using your cursor control keys ([CTRL], [↑], [←], etc.), move the cursor until it sits over the 1 in 10 as shown here:

```
10 PRINT "THIS IS MY ONLY LINE"
```

Type a 2 and press [RETURN]. Do this again, except overwrite the 2 with a 3; repeat with a 4 over the 3, etc. After three or four of these, LIST the program. What has happened? The computer has made multiple copies of the same line but each has a different line number!

Your program will now look like this:

```
10 PRINT "THIS IS MY ONLY LINE"
20 PRINT "THIS IS MY ONLY LINE"
30 PRINT "THIS IS MY ONLY LINE"
40 PRINT "THIS IS MY ONLY LINE"
```

If you RUN this program, it will print this message once for each line number. In the example above, we got four identical statements. When you begin writing some more complex programs, this feature will save you a great deal of typing, since you can duplicate a BASIC statement anywhere in your program by just "editing" its line number! Always be sure to press the [RETURN] key, or the new line will not be properly stored in the program memory.

To insert a new line, simply give it a number which falls between the two line numbers where you want to place it; thus to insert a line between 20 and 30 in this program:

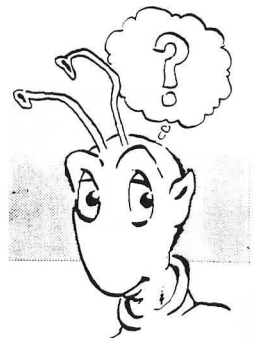
```
10 PRINT "WW";
20 PRINT "XX";
30 PRINT "ZZ";
40 GOTO 20
```


Simply type: 25 PRINT "YY";



Any other line number between 20 and 30 that was not already used as a line number would also do.

You may have noticed that all the programs in the book so far have line numbers that are multiples of 10. We didn't have to do it that way. They could be multiples of 5, or 100, or any number. These lines could also be numbered consecutively from 1 on. However it is always a good idea to leave some numbers open between lines just in case you might want to insert new lines later on.



Some ways to number a program: 

Program 1—leaves no room for new lines.

```
1 PRINT "X";
2 PRINT "Y";
3 PRINT "Z";
```

Program 2—will generally do.

```
10 PRINT "X";
20 PRINT "Y";
30 PRINT "Z";
```

Program 3—is a bit eccentric but will also do the same thing.

```
14 PRINT "X";
170 PRINT "Y";
176 PRINT "Z";
```



## Concepts

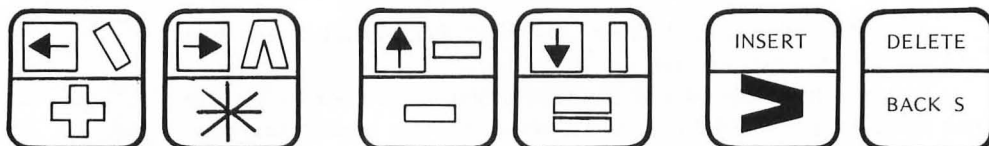


**Error Messages**—These are messages built into ATARI BASIC or into the logic of your computer. They flash on the screen when you've made a common mistake. For more information on errors and error messages, see Appendix I.

## Special Keys

**[SHIFT][CLEAR]**—Pressing the [CLEAR] key while holding down the [SHIFT] key clears the screen and puts the cursor in the upper left corner without erasing the memory.

**[CTRL]**—The control key functions with the following keys to provide you with the ability to edit your programs.



You may have noticed as you're reading that your TV changes color periodically. This is caused by a mechanism built into your ATARI Home Computer to protect your TV from burning out certain color phosphors as a result of having one image on the screen for too much time.

These keys move the *cursor*, the small square on the screen that indicates where the computer will enter the next thing you type.


[SHIFT] [INSERT]—Adds a line.

[SHIFT] [DELETE]—Deletes a line.

[10] [RETURN]—Deletes line 10 from your program. Typing a line number and pressing [RETURN] will delete that line of your program.

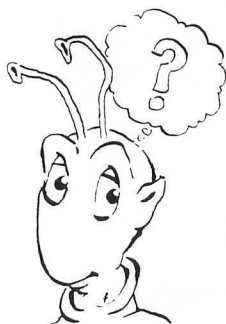


## THE ELUSIVE AND POWERFUL *RND(1)*

If you have ever played dice or roulette or bingo you already have an intuitive understanding of *RND(1)*.  If you have ever had to draw a name out of a hat or pick a card to see who goes first in a game, you have also encountered the idea of *RND(1)*. *RND(1)* is a random number function in ATARI BASIC that makes use of logic built into your ATARI Home Computer. It simply picks a number "at random," such as picking it out of a hat, spilling it out of a bingo cage, or choosing it from a deck of cards. Because the computer can command so much information and do things so quickly it can pick random numbers from a larger sample than you can do with cards or dice or bingo tokens. Instruct your computer to `PRINT RND(1)`. Remember, quotes are *not* used. What do you get? We got 0.2755226953, a decimal of 10 places. You will also get some 10-place decimal between 0 and 1. RUN the following program and you'll get a slew of random numbers:

```
10 PRINT RND(1)
20 GOTO 10
```

*Notice that this is just another version of the "name program" you did before.*

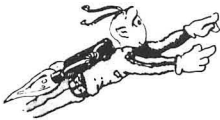


In ATARI BASIC you may use any number inside the parentheses for the *RND* function: *RND(0)*, *RND(1)*, *RND(12345)*, etc. All are equivalent. However, to make things simple and consistent, we will always use *RND(1)*.

If you [BREAK] this program and examine the numbers, you will find that they are all between 0 and 1. For many mathematical purposes these numbers are quite useful. However, for game purposes we need numbers that are more like those that we can get from dice and spinners and cards. We need whole numbers, which in ATARI BASIC language are obtained by using the *INT* function for *INT*egers. If you put *INT* in front of the *RND(1)* function, the decimal fraction is eliminated or "truncated," and only the whole number before the decimal will be printed. What do you think this program will do when you *RUN* it?

```
10 PRINT INT(RND(1))
20 GOTO 10
```

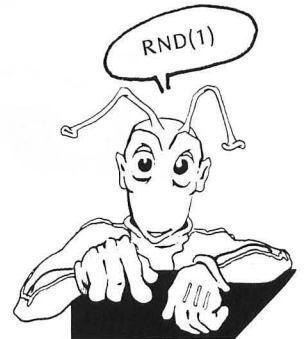
Parentheses are used, as in algebra, to mark off parts of formulas. They must always be paired.



The *INT* function will eliminate the decimal and *PRINT* the whole number, which in each instance here will be 0. For this reason we have to add a few more embellishments to the *INT(RND(1))* function in order to make it useful for games.

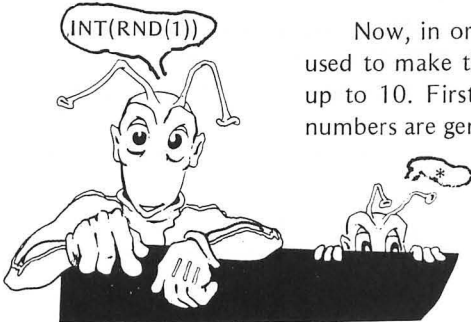
At this point it is important to understand what we are doing to the *RND(1)* function, as it is an example of the way in which complex programs are built out of simple parts. We begin with:

*RND(1)* which gives us a number between 0 and 1 to 10 decimal places.

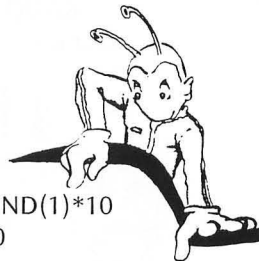


```
INT(RND(1))
```

Now, in order to go higher we introduce another number that will be used to make the original *RND(1)* greater than 1. Let's say we want to get up to 10. First we can *multiply* *RND(1)* by 10. Try this and see what numbers are generated by the program:

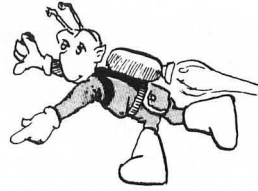


```
10 PRINT RND(1)*10
20 GOTO 10
```



If you *RUN* this program you will discover that you now have random numbers between 0 and 9.99999999. If you modify the program to get only integers, you will have a short program that looks like this:

```
10 PRINT INT(RND(1)*10)
20 GOTO 10
```



The careful use of parentheses is essential to avoid simple programming errors. Every left parenthesis has to have its corresponding right parenthesis. Check every statement that has several sets of parentheses to see that they are paired and placed where you want them.

This program will eliminate the decimal parts of all the numbers and PRINT only the integers between 0 and 9. If we want the integers between 1 and 10, we will have to add a number so our program would look like this:



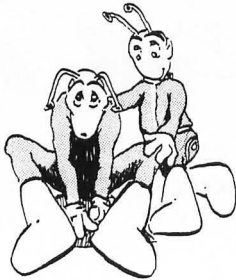
INT(RND(1)\*10)+1

```
10 PRINT INT(RND(1)*10)+1
20 GOTO 10
```



We now have a program that generates numbers between 1 and 10. If you wanted a program for throwing one die, you would write a RND program that would give you random numbers between 1 and 6. Could you write that program now? All it requires is some simple modifications of the program for numbers from 1 to 10.

If you have found it hard to follow this argument, don't give up. Here is a picture of what is happening. Always try to draw a picture of what you are doing with a program if you feel that you are getting lost. *A program can be diagrammed, and drawing is a powerful programmer's tool!*

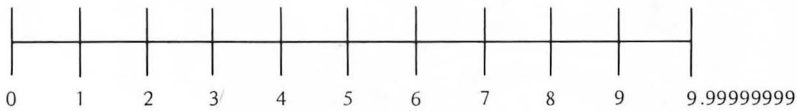


RND(1) gives numbers here to a maximum of 10 decimal places of accuracy.

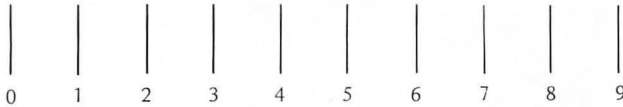




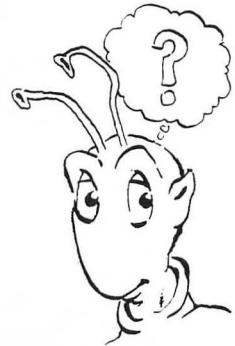
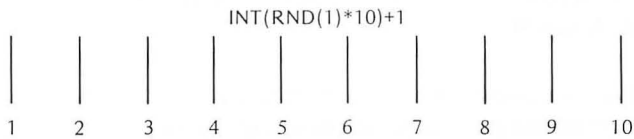
$RND(1)*10$  gives numbers between 0 and 9.99999999.

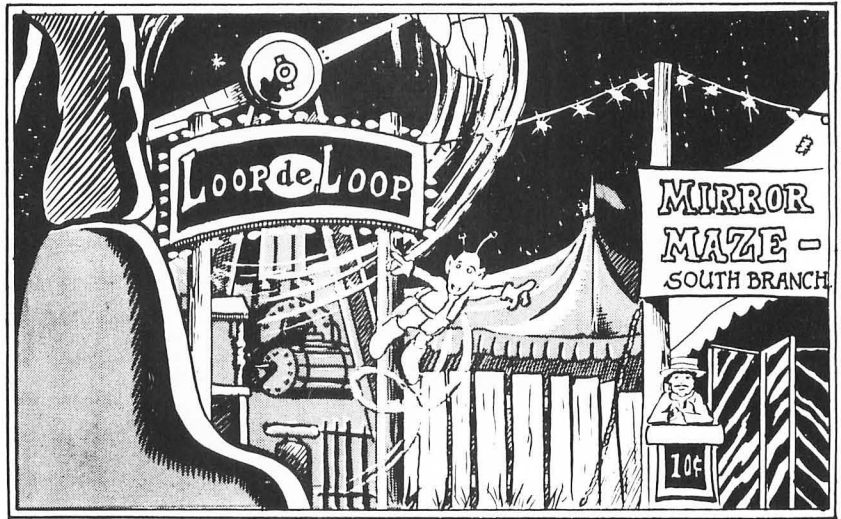


$INT(RND(1)*10)$  drops off the decimals and picks out the integers from 0 to 9. Notice that 10 is not picked out because  $RND(1)*10$  only goes to 9.99999999.



By adding one to the previous statement, we get the integers from 1 to 10:





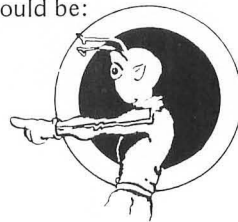
### USING RND(1): SOME LOOPS AND A BRANCH WILL MAKE A GAME

With a few additions to the vocabulary you will be able to create some interesting guessing games while mastering ATARI BASIC. The first addition is quite simple:

```
LET X =
```

X is a variable, that is, its value can vary according to what you want it to be. We will start with number variables and later on show you how to use string variables which can contain words and letters. Here is a simple example of deciding what X should be:

```
10 LET X = 10
20 PRINT X+30
```



What do you think the computer will do when you type RUN? It will print 40, as it added your X to 30. If you change X to the following numbers, what do you think the computer will print? X=90,23,2,34,3.007

Now, to get a little tricky and use what was learned in the section about RND, let X be a number chosen at random by the computer. We could ask X to be a number chosen at random between 1 and 6 (like tossing a die) by letting X be described as follows:

```
10 LET X=INT(RND(1)*6)+1
20 PRINT X
```

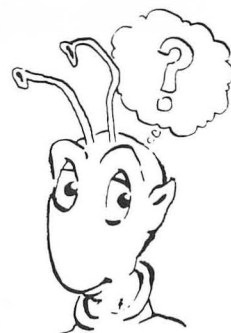
Try this program. The computer will choose a random integer between 1 and 6 and then print it. If we added the following line the computer would keep on choosing and printing random numbers between 1 and 6 as if it were tossing dice eternally: 30 GOTO 10

Try the following exercises before you go on: Have X be a random number between 1 and 20, 1 and 4, and 1 and 52. Write a program to PRINT the numbers chosen if the computer chooses only once. Then add a line so that the computer goes into an endless loop and keeps on choosing numbers at random.

LET X= can become a bore. Everytime you want to change X you have to write a new program just as you did above. In order to simplify things (and the computer is excellent at simplification once you get the hang of it), there is a statement in BASIC that lets you put a different number value for a variable into the computer whenever you choose. The key word is INPUT and it means just what it says: Put something in. That something, when it is a number, is usually designated by some variable name like X or Y or Z. Thus, INPUT X means that the computer will hold a space in its memory for X, the number you put in. Try the following miniprogram:

```
10 INPUT X
20 PRINT X+30
30 GOTO 10
```

This program differs from the LET X= program simply by replacing the LET statement by the INPUT statement. When you RUN the program a ? appears on the screen. You can now put any number into the program, say 2345, press [RETURN], and get the sum of that number plus 30. Notice that after the sum is given, the ? appears again. That is because at line 30 we told the computer to go back and give us another chance to put a new number into the machine. In this way you can do multiple adding (or any complex arithmetic operation) on any number of INPUTs without having to change the program. For example, if you want to set up a program for dividing a number by 5 and then multiplying the answer by 12, here is a simple way to do it:



```
10 INPUT X
20 PRINT X/5*12
30 GOTO 10
```



Line 20 will calculate in the following order: First calculate  $X/5$ ; then take this result and multiply it by 12. ATARI BASIC uses standard mathematical conventions, i.e., it does all multiplications and divisions first, from left to right, then additions and subtractions.



Try some numbers and see how easy it is to set up a program for continued calculation using one INPUT statement. Then stretch a bit and see if you can develop some programs using INT, RND(1), LET X=, and INPUT. Here's one program that uses all these new BASIC words. What do you think the program does? RUN it and see if you are right.

```
10 LET X=INT(RND(1)*6)+1
20 INPUT Y
30 PRINT X*Y
```

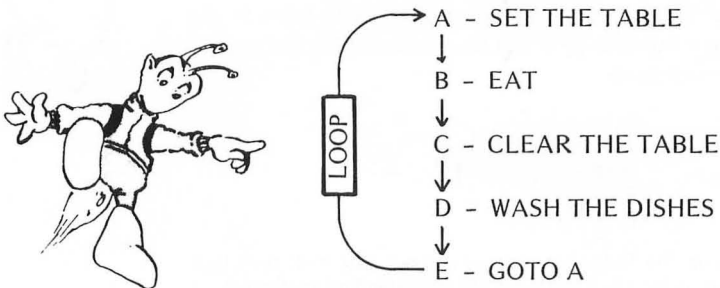
With one more concept we will be able to set up a program for a number guessing game. This might seem like a lot of work to develop a simple game, but remember that we are now almost halfway through the words you'll need to know in order to play, program, and modify most of the games in this book. The simple example belies the complexity that is embodied in the BASIC vocabulary already presented.

The new BASIC word (actually two words joined together to make a statement) is IF . . . THEN. This concept is understood by the computer in much the way you use it in everyday life. It tells the computer that IF something happens to be the case, THEN do something else, which may be to branch to another part of the program. IF it is not the case, THEN go on with business as usual.



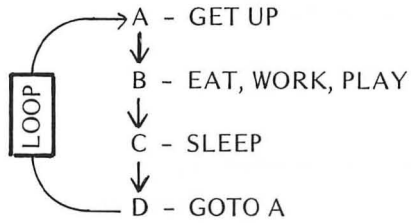
When you master branches and loops, you are ready to become a proficient and confident programmer. Here are some everyday loops and branches:

#### A Boring Loop



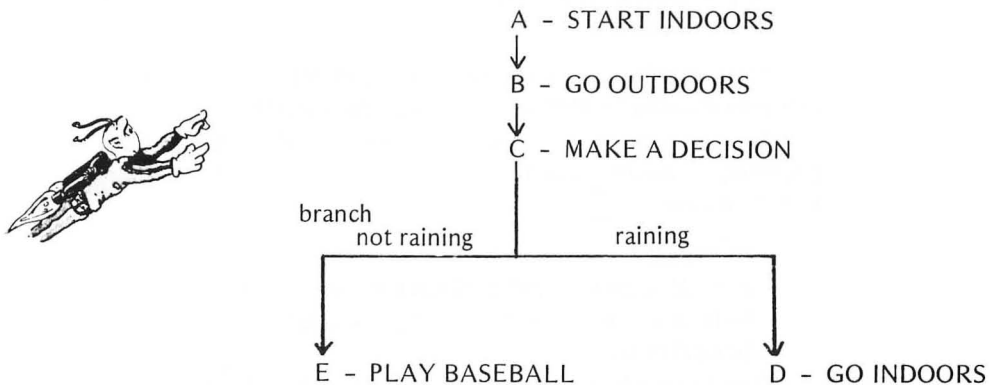
E is an *unconditional* branch and here it sets up a loop.

## The Daily Round

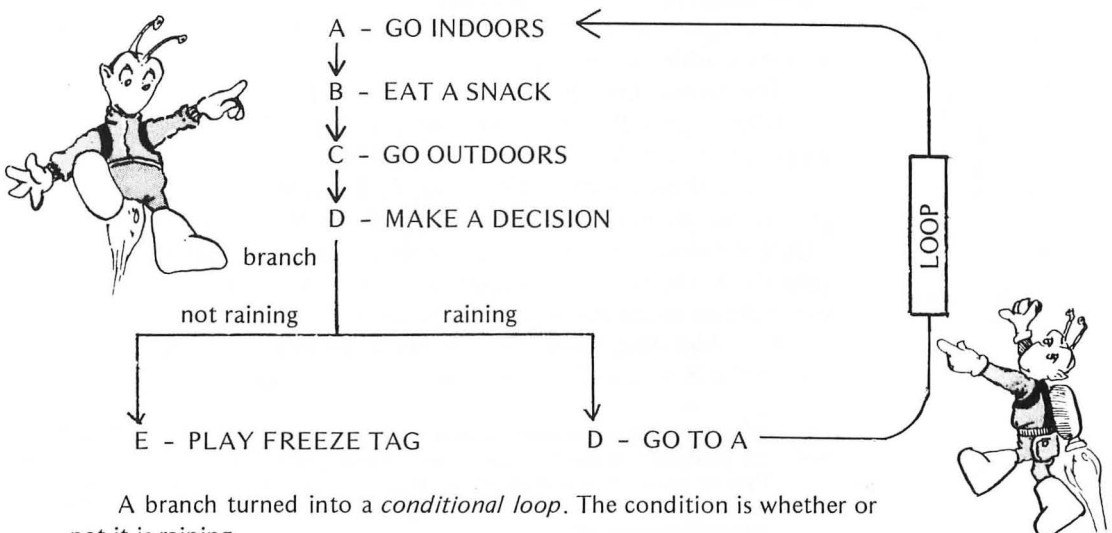


Here D is the unconditional branch which sets up a loop.

### A Wet Branch

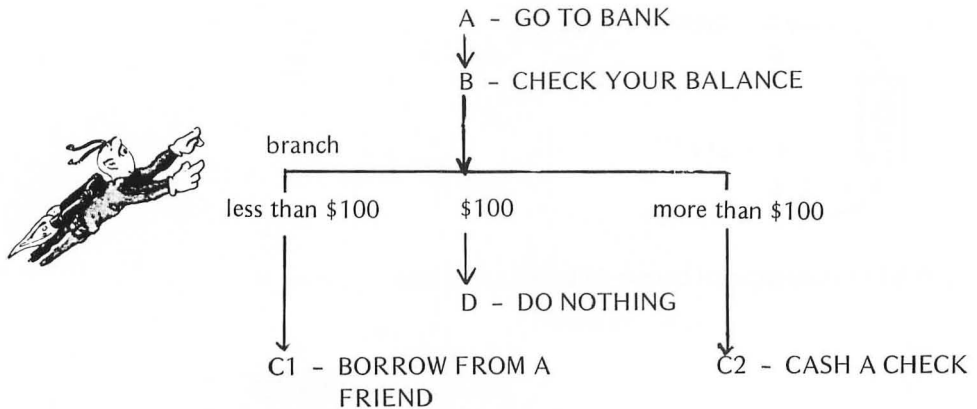


### Another Wet Branch



A branch turned into a *conditional loop*. The condition is whether or not it is raining.

## Two Conditional Branches and No Loop



After looking at these everyday examples of loops and branches, let's introduce an IF THEN branch to a simple INPUT program.

There is one more symbol we'll use:  $<>$  which means *not equal to* (literally, it means "less than or greater than") and is made by typing first  $<$  and then  $>$ .

```

10 INPUT X
20 IF X=6 THEN PRINT "THAT IS THE RIGHT NUMBER"
30 IF X <> 6 THEN PRINT "TRY AGAIN"
40 GOTO 10
  
```

Try to visualize what this program says. It will help you if you understand how we made up the games presented in the text and also help you to design programs on your own.

The first line, 10 INPUT X, says to put some number, any number, into the machine's memory.

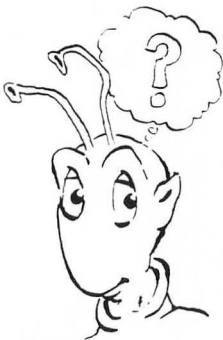
The second line, 20 IF X=6 THEN PRINT "THAT IS THE RIGHT NUMBER", says that if the number you put in is 6 then the machine should print "THAT IS THE RIGHT NUMBER".

If the chosen number (X) is not 6, the computer skips line 20 and goes on to the next step. That is one of the basic characteristics of IF THEN statements and one to think about when you begin to use them yourself. If the conditions of the statement are not met, the computer goes right on to the next line of the program.

The third line, 30 IF X  $<>$  6 THEN PRINT "TRY AGAIN", says that if the X you INPUT into the computer is not 6 then the machine



While we're at it, we might as well introduce  $<$  and  $>$ . They are used as they are in algebra to stand for less than ( $<$ ) and greater than ( $>$ ).  $< =$  means less than or equal to,  $> =$  is greater than or equal to, and  $<>$  means not equal to. Arithmetic symbols are: add +, subtract -, multiply \*, divide /, and exponentiation  $\wedge$ .



should print "TRY AGAIN" and return to the INPUT statement so that the guesser can try again.

Here is another example of an IF THEN branch that you should try to visualize. What will the computer do with this program? Put the program into the machine and see if it does what you expected:

```
10 PRINT "WHAT'S MY NUMBER?";
20 INPUT X
30 IF X+2=5 THEN PRINT "YOU GOT IT"
40 IF X+2 < > 5 THEN GOTO 10
```

IF you have been able to visualize this program THEN you are ready to use all of the language we have developed so far to program a number guessing game, to embellish it for your own pleasure. Before developing the game, let's review the concepts and vocabulary introduced in this part of the chapter.



### Review:

## Concepts

**Branching**—A program has branches at those lines where one of a number of different things can happen. Branches are represented by IF THEN statements.

## BASIC Vocabulary

**RND(1)**—This command generates a random number between 0 and 1 (not including 0 and 1 themselves).

**INT**—This function turns decimal numbers into integers in the following way: Each decimal number is reduced to the integer to the left of the decimal point. Thus 3 and 3.9 and 3.999999 will all be turned into 3 by the INT function. To see this, type PRINT INT(3) and press [RETURN]. The computer will then PRINT 3. If you type PRINT INT(3.999) and press [RETURN], the computer will also PRINT 3.

**INT(RND(1)\*any integer)**—This command generates integer random numbers between 0 and one less than the number you chose (not counting that number). To get the positive random numbers from 1 to your chosen number you change the command to INT(RND(1)\*any positive integer)+1. Thus INT(RND(1)\*9) will generate random numbers from 0 to 8, inclusive, and INT(RND(1)\*9)+1 will generate random numbers from 1 to 9, inclusive.

**LET X=**—This command assigns a number value to the variable named X.

**INPUT X**—This command allows the person using the computer to assign a different number value to X from the keyboard each time the program is RUN.

$X < Y$  X less than Y  
 $X \leq Y$  X less than or equal to Y  
 $X > Y$  X greater than Y  
 $X \geq Y$  X greater than or equal to Y  
 $X \neq Y$  X not equal to Y  
 $X + Y$  X plus Y  
 $X - Y$  X minus Y  
 $X * Y$  X multiplied by Y  
 $X / Y$  X divided by Y  
 $X \wedge Y$  X raised to the Y power

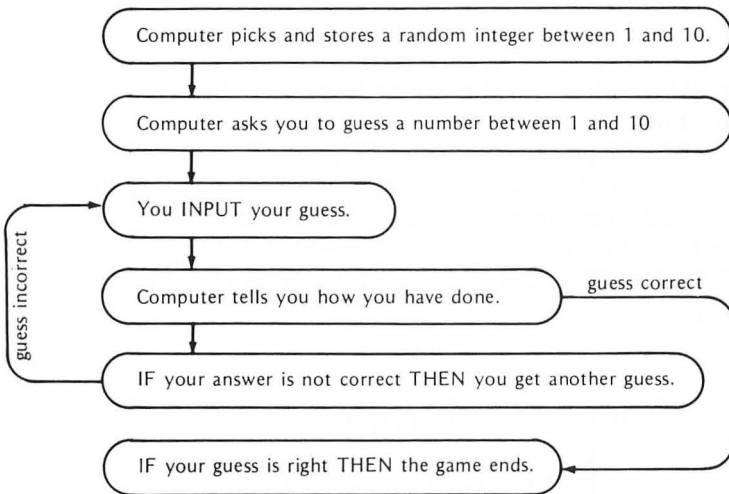
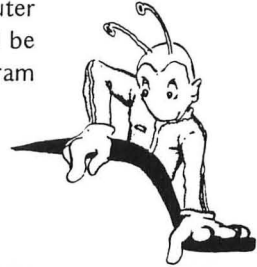
### Delimiters

( )—Parentheses are used, as in algebra, to mark off parts of formulas. They must always be paired.



## THE GAME AT LAST: GUESS A NUMBER


We will now program a simple number-guessing game. The computer will pick a number between 1 and 10 at random, store the number in its memory, and then ask you to guess the number. If you guess correctly, the computer will let you know. If you make a wrong guess, the computer will give you another chance. Thus the structure of the program will be as follows (it is a good idea to sketch out the structure of your program beforehand as a guide to your work):



In order to write this program in a language the machine understands, we start with the computer choosing a number between 1 and 10 at random. We do this using the RND(1) function.

```
10 LET X=INT(RND(1)*10)+1
```

Now the computer has chosen and stored a number. The next step is for the machine to ask you to guess. That can be done with a simple PRINT statement:

```
20 PRINT "I HAVE CHOSEN A NUMBER FROM 1 TO  
10. ■■TRY TO GUESS WHAT IT IS" 
```

Now you INPUT your guess. Notice that you have to use a variable letter other than X since X is already reserved for the number the computer chose. We'll use Y, but it could be any letter:



Whenever you see a line like this with more than 38 characters, keep on typing until the end of the logical line before pressing the [RETURN] key.

## 30 INPUT Y

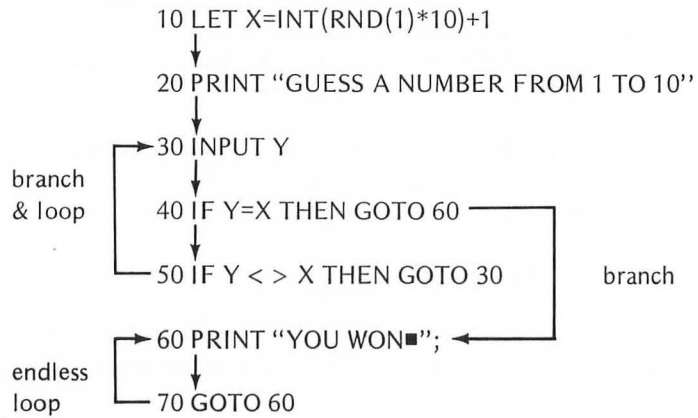
Next we have to set up some IF THEN branches and we'll be ready to play:

```
40 IF Y=X THEN PRINT "YOU GOT IT!■GOOD JOB."
50 IF Y < > X THEN GOTO 30
```

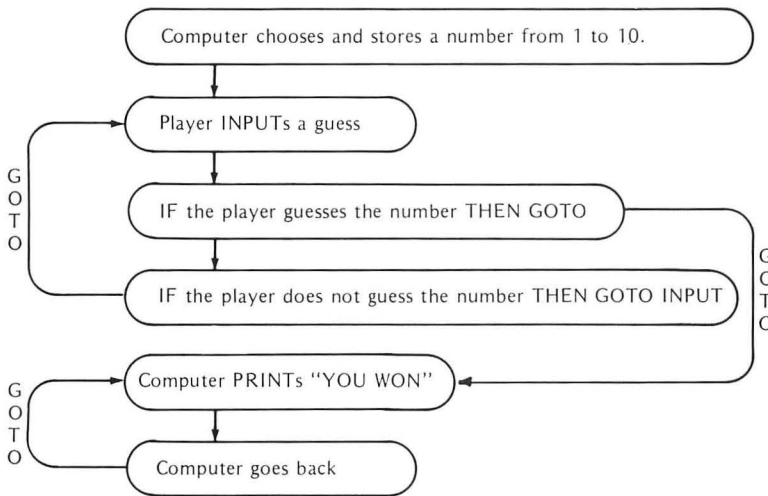
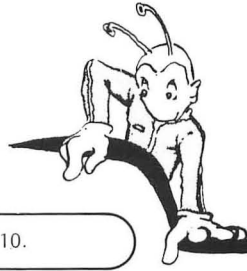


Line 50 instructs the machine to go back to the INPUT line and give the player another chance.

Now RUN the program and play the game. If you like, change the messages in the PRINT statements, add a new ending, or embellish the game in any way that strikes your fancy. For example, you might put in an endless loop that says "YOU WON", or make the computer choose a number from 1 to 25, etc. Here's one embellishment of the game using an endless loop.



This is how the program works:



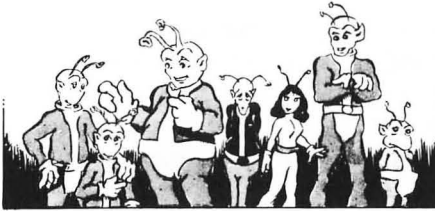
In this section on number games you'll find a number of embellishments of this simple game. You might want to look at them now or wait until you develop some more vocabulary and can complicate the game yourself.

See if you can write the program on your machine without looking at our program. Try to think through the steps required and recreate the game. If you can do that, you are ready to move on to more complex games and to some new BASIC vocabulary. If not, leaf ahead in the book, copy and play some of the games presented, and then try this chapter again some other time.



## Chapter 2

# Theme and Variations: An Introduction to the Fine Art of “Dressing Up”



The finest games, whether they are played on a computer or not, generally have evolved from a relatively small number of simple themes. For example, the ancient games of *GO* and Chess are games of capture, and *MONOPOLY*® is a game of control of money and property.

What makes each of these games unique is the way in which the theme has been “dressed up.” Each game places us in a play world which has its own personality and character. And it is this perennial combination of simplicity and elegance which attracts us and keeps us “hooked.” What is intriguing about fine games is that we may continue to find endless ways of playing them and modifying them, sometimes even changing the rules to create an entirely new game from an old structure.

We hope that this book will encourage you to change and modify games we present so that they become your own, and that you will create new games for your ATARI Home Computer that are novel and exciting to play. One way to get involved in this process is to watch how a very simple idea can grow and evolve. It's very much like watching a new house being built.

The theme for the game presented in this chapter is the classic race between the Tortoise and the Hare. The old, plodding, deliberate Tortoise is challenged to a race by the speedy Hare. The Tortoise is steady, regular, and sometimes painfully slow. The Hare is fast but erratic. It jumps all around, is impatient, and in the original Greek fable is foolish. The Hare

---

MONOPOLY® is a registered trademark of Parker Brothers, Inc.

runs too fast for itself and is defeated by the slow persistence of the Tortoise. In our computer equivalent of the race, the ending isn't so predictable. Each participant has a chance and the outcome differs from race to race. Before getting more specific about the Tortoise and the Hare, however, a few new parts of BASIC vocabulary have to be introduced.

The first new concept is that of a FOR-NEXT loop.

The first looping command we introduced was the GOTO command. In our example it set up loops like the following:

```
10 PRINT "I WISH I COULD DO MORE INTERESTING
   THINGS WITH MY ATARI COMPUTER";
20 GOTO 10
```

This GOTO instruction, without any way of stopping it, sets up an *infinite loop*. FOR-NEXT commands make it possible for you to control the exact number of times you want the computer to run through the loop. For example, suppose you wanted to PRINT "I'M JUST A MACHINE" exactly five times. Here are two ways of doing this using LET, IF-THEN, and GOTO statements:



```
10 LET X=0
20 LET X=X+1
30 PRINT "I'M JUST A MACHINE"
40 IF X<5 THEN GOTO 20
```

Or:

```
10 LET X=1
20 PRINT "I'M JUST A MACHINE"
30 LET X=X+1
40 IF X<=5 THEN GOTO 20
```

Now, here is how you can write this same program using a FOR-NEXT loop:

```
10 FOR X=1 TO 5
20 PRINT "I'M JUST A MACHINE"
30 NEXT X
```

Examine the program which uses the FOR-NEXT loop. Notice that line 10 tells the machine how many times to run through the loop. You can experiment with it; for example, change line 10 to:

```
10 FOR X=1 TO 10
```



Or:

```
10 FOR X=1 TO 1000
```

Line 20 is the PRINT command, telling the computer what to PRINT each time it goes through that loop. Line 30 is the crucial line; it closes the loop and is similar to a GOTO statement. The NEXT X command instructs the machine to go back to the beginning of the loop and do it again until the right number of loops are made.

See what happens when you leave out the NEXT X instruction. RUN this on your ATARI:

```
10 FOR X=1 TO 10
20 PRINT "LOOP DE LOOP"
```

The machine goes through the loop once and then stops and waits for the next command. Now, add:

```
30 NEXT X
```

RUN the program again. What happened this time?

Here is a little practice with the FOR-NEXT loop:

- Write a program that will PRINT your name exactly 12 times.
- Write a program that will PRINT 3 random numbers between 1 and 6 (the equivalent of tossing a die 3 times).

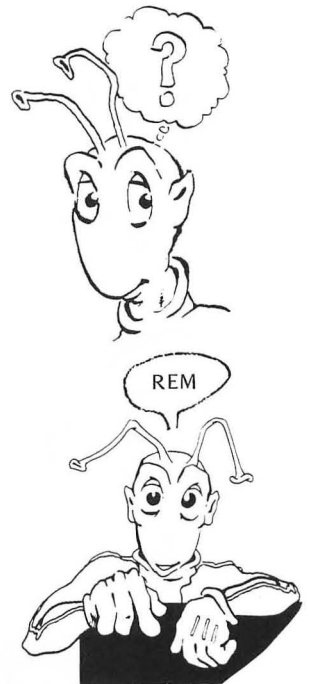
Another BASIC word that is useful when you write your own programs or try to understand other people's programs is REM.

REM means REMark. It really is not a part of your program but is a note to you, the programmer, to tell you what is in the program. It is difficult to look through a program and visualize what it is supposed to do without hints. Therefore, using REM statements is usually a good idea. In our number guessing game the REM statements might be used this way:

```
5 REM NUMBER GUESSING GAME FROM 1 TO 9
10 LET X=INT(RND(1)*9)+1
20 PRINT "I HAVE CHOSEN A NUMBER FROM 1 TO 10.
    TRY TO GUESS WHAT IT IS"
30 INPUT Y
40 IF Y=X THEN PRINT "YOU GOT IT!"
50 IF Y<>X THEN GOTO 30
```



Don't press the [RETURN] key until the end of each "logical line." When you run out of room on the screen, your ATARI Home Computer will just keep going on to the next line.



Here is another example of a REM statement:

```
5 REM PRINTING MY NAME MILLIONS OF TIMES
10 PRINT "MY NAME";
20 GOTO 10
```



The REM statement does not affect the program in any way, and you can write your REMs in any style that you find comfortable. Some programmers find BASIC grim and unimaginative, and so they write funny, eccentric REMs. Other REM statements are as austere as BASIC. It is a matter of style, of making the program and the machine your own.

There are four more commands that are needed to get on with the Tortoise and the Hare. These commands throw your ATARI Home Computer into GRAPHICS mode and let you draw on the screen. What this means is that the screen becomes like graph paper, and you can plot points on it just as you can plot points on graph paper with a pencil or pen. To get your TV screen into graph paper mode you must give the machines a GRAPHICS instruction. For the details of the nine GRAPHICS modes, your ATARI computer can use see Chapter Six. For our game the GRAPHICS 7 mode will be used. The command:

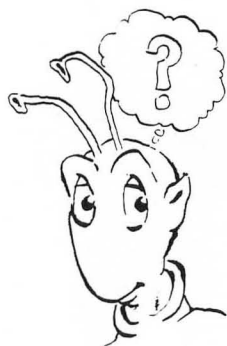
```
10 GRAPHICS 7
```

enables the computer to draw a graph that has 160 columns (numbered 0-159) and 80 rows (0-79). In addition, four lines of text can fit on the screen in the text window that appears at the bottom of your TV screen. Thus GRAPHICS 7 will set up your screen as illustrated in the following diagram.

The 0, 0 point of the graph is at the upper left-hand corner of your screen. This is very different from most graphs that one sees where the 0, 0 point is at the lower left.

The command 10 GRAPHICS 7 is not enough to get your ATARI computer to produce a graph, however. To produce an image the computer needs to have instructions about both color control and point location on the screen. Remember that your ATARI Home Computer takes over control of the TV screen. The screen does what the computer tells it to do. You have to give a COLOR command along with putting the screen into a GRAPHICS mode. In a while we'll show you how to be fancy with COLOR. For now we will assume a black-and-white screen. To get into GRAPHICS 7 and produce a black-and-white image, this is the instruction you type for the computer:

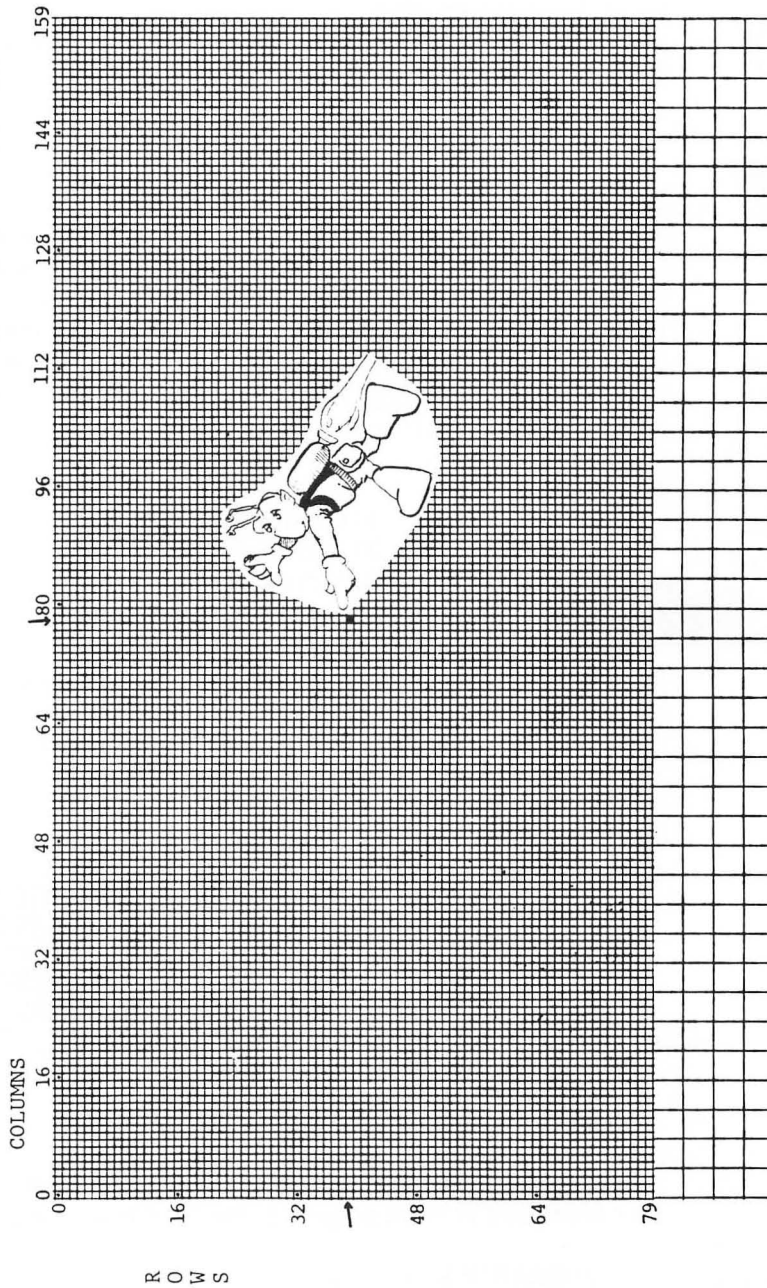
```
10 GRAPHICS 7
15 COLOR 1
```

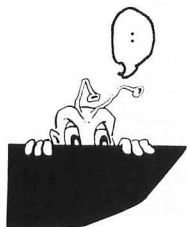




# Graphics Mode 6 or 7

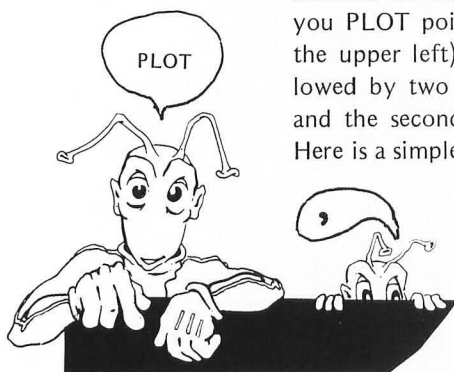
with Text Window





There is not much space taken up by these two lines and ATARI BASIC has a way of joining two short lines together. It uses the colon (:) to condense two or more BASIC command lines into one line. Thus lines 10 and 15 above can be rewritten as follows:

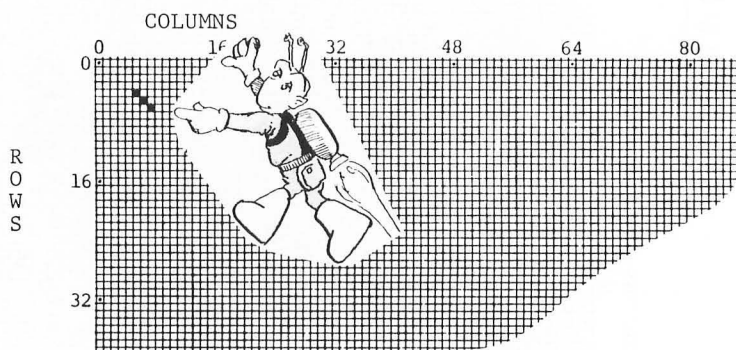
```
10 GRAPHICS 7:COLOR 1
15 [RETURN]
```



Once you have done that you are ready to PLOT points and draw lines on the screen. You can PLOT points on the screen in the same way you PLOT points on graph paper (remembering of course that 0, 0 is at the upper left). The BASIC word for plotting points is just PLOT, followed by two numbers. The first number is the column the point is in, and the second is the row. The numbers are separated by a comma (,). Here is a simple program and a graph which represents it:

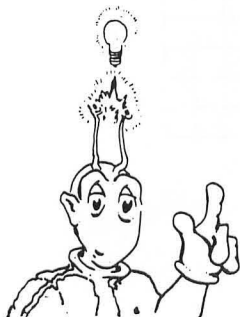
```
10 GRAPHICS 7: COLOR 1
20 PLOT 5,4
30 PLOT 6,5
40 PLOT 7,6
```

Remember that the number in front of the comma is the column number and the number after the comma is the row number. Also remember that in GRAPHICS 7 there are 160 columns (0-159) and 80 rows (0-79).



Now try to visualize where the following points will be plotted on your screen, and RUN the program to check out how close you were:

```
10 GRAPHICS 7:COLOR 1
20 PLOT 20,30
30 PLOT 120,10
40 PLOT 10,60
```



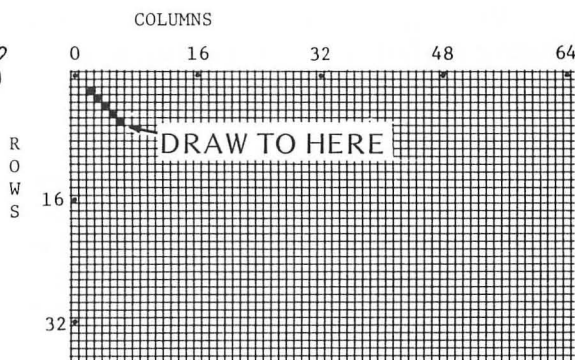
```
50 PLOT 50,60
60 PLOT 150,10
```

Try to PLOT some points yourself. A good exercise that can help you get familiar with the screen is to PLOT a point in your program and to mark that point on graph paper like the one shown above before RUNning your program. (See Appendix for blank GRAPHICS sheets to use.) After a while you'll develop a feel for the placement of points on the screen.

There is one final command that we need to consider before getting to the race. That command is DRAWTO. In ATARI BASIC, DRAWTO enables you to draw a line from any point on the screen to any other point on the screen.

Here is a simple DRAWTO program and its graph:

```
NEW
10 GRAPHICS 7:COLOR 1
20 PLOT 2,2
30 DRAWTO 6,6
```

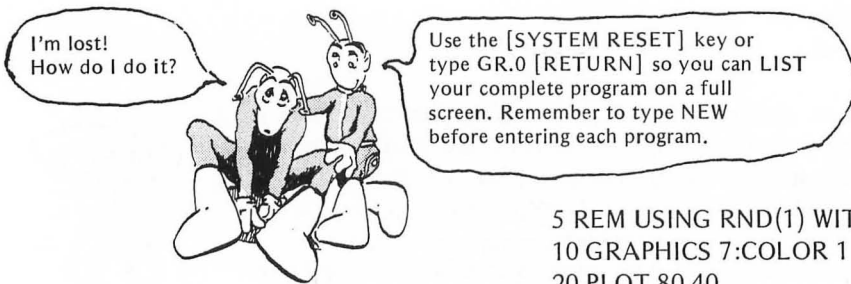



Try this. Then try the following programs on your ATARI Home Computer before RUNning the programs; study them and guess what they will look like on the screen:

```
10 GRAPHICS 7:COLOR 1
20 PLOT 150,79
30 DRAWTO 0,0
```

```
10 GRAPHICS 7:COLOR 1
20 PLOT 5,5
30 DRAWTO 10,10
40 PLOT 1,1
50 DRAWTO 6,7
```





```
5 REM USING RND(1) WITH GRAPHICS
10 GRAPHICS 7:COLOR 1
20 PLOT 80,40
30 DRAWTO RND(1)*100,RND(1)*50 
```

```
5 REM ILLUSTRATES HOW REPEATED USE OF
  RND(1) IS INTERESTING AND USEFUL FOR
  GAMES AND GRAPHICS
10 GRAPHICS 7:COLOR 1
20 PLOT 80,40
30 DRAWTO RND(1)*100,RND(1)*50
40 GOTO 20
```



**Review:** Remember the new vocabulary introduced here, and review these pages if you find yourself confused by our use of these new terms.

## Concepts

**Graphics modes**—The ATARI Home Computer has the capacity to turn the TV screen into a graph-paper-like sheet that you can draw on. There are nine different graphics modes in ATARI BASIC. The one we will use in this chapter is GRAPHICS 7 (see new BASIC vocabulary below). For further information turn to Chapter Six on ATARI Graphics.

## BASIC Vocabulary

**REM**—A BASIC statement which stands for REMark, reminder, remember, or whatever. In any case, it is ignored by the program and is there for your own use as comment or notes to yourself on what the program is doing or supposed to do.

**GRAPHICS 7**—This statement turns the screen into a graph with 160 columns (0-159) and 80 rows (0-79). At the bottom of the screen is a window that can display four lines of printed text.



Your ATARI Home Computer automatically PLOTs random numbers as integers.

**COLOR 1**—This statement tells the ATARI Home Computer to select a particular color for drawing or plotting points. With black-and-white TV screens, it provides a good contrast against the black graphics screen.

**FOR-NEXT**—A pair of statements that go together to set up a loop. The FOR statement opens up the loop and specifies how many times the loop will be repeated. The NEXT statement closes it each time and sends the program back to do the loop again until the specified number of repetitions have occurred. To set up a FOR-NEXT loop, each FOR statement in a program must be closed by a corresponding NEXT statement.

Here is a simple example of a FOR-NEXT loop that goes around five times:

```
10 FOR X=1 TO 5
20 PRINT "HELLO"
30 NEXT X
```

If you RUN this simple program, the machine will print:

```
HELLO
HELLO
HELLO
HELLO
HELLO
```

**PLOT X, Y**—This statement places a point on the screen at the intersection of the X column and Y row. Notice that X and Y are separated by a comma. X and Y can take values from 0 through all the positive numbers so long as those numbers fall within the size of the numbers of rows and columns in the graphics mode you are using. For GRAPHICS 7, X can range from 0 to 159 and Y from 0 to 79. (Note: X and Y can be variables as well as numbers. For example, using a FOR-NEXT loop you can PLOT the following four points, 1,1 2,2 3,3 4,4.)

```
10 GRAPHICS 7:COLOR 1
20 FOR X=1 TO 4
30 PLOT X,X
40 NEXT X
```

**DRAWTO X,Y**—After PLOTting a point, you can draw a line from that point to any other permissible point on your screen by using the DRAWTO command. For example, this program will draw a diagonal line from the top left to the bottom right of your screen when RUN.

```

10 GRAPHICS 7:COLOR 1
20 PLOT 0,0
30 DRAWTO 159,79

```

### Special Keys

**[System Reset]**—restores the screen display to GRAPHICS 0 (mode 0), clears the screen and returns margins and other special variables to their “default” values. In general, if you just want to clear the screen or the text window in GRAPHICS modes, the proper operation is to press [SHIFT][CLEAR]. In most cases, pressing [SYSTEM RESET] will accomplish the same result, but it occasionally has unwanted side effects.

### Delimiters

**Colon**—The colon (:) is used to combine two or more ATARI BASIC statements into one statement with one line number. In this example:

```

10 GRAPHICS 7
20 COLOR 1

```

lines 10 and 20 can be rewritten as follows:

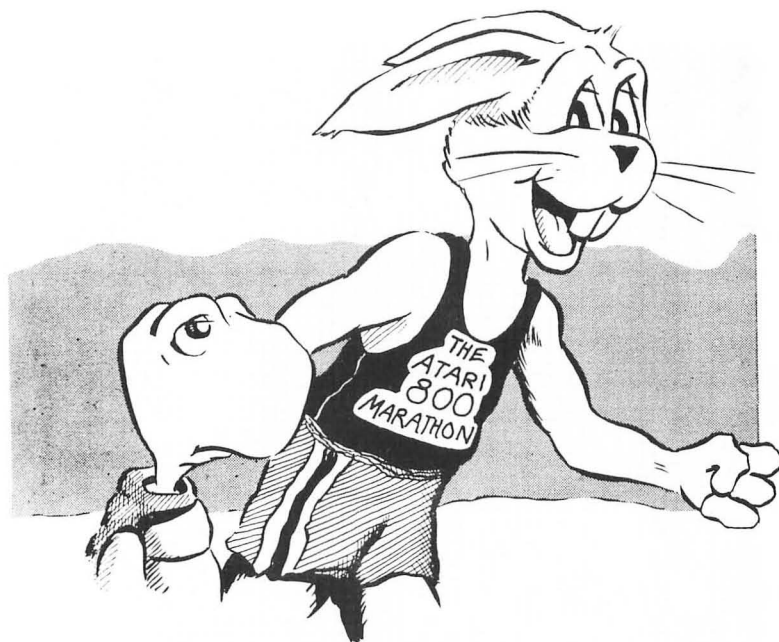
```

10 GRAPHICS:COLOR 1

```

Remember, whenever the logical line being typed in exceeds 38 characters, the ATARI Home Computer will automatically move on to the next line of the screen. (Lines in program listings that show two or more consecutive lines without a new line number are to be typed as one continuous line on your computer.)

**Comma**—The comma (,) is used to separate the variables or numbers assigned to the columns and rows used in the GRAPHICS mode.



## THE TORTOISE AND THE HARE AT LAST-ALMOST!

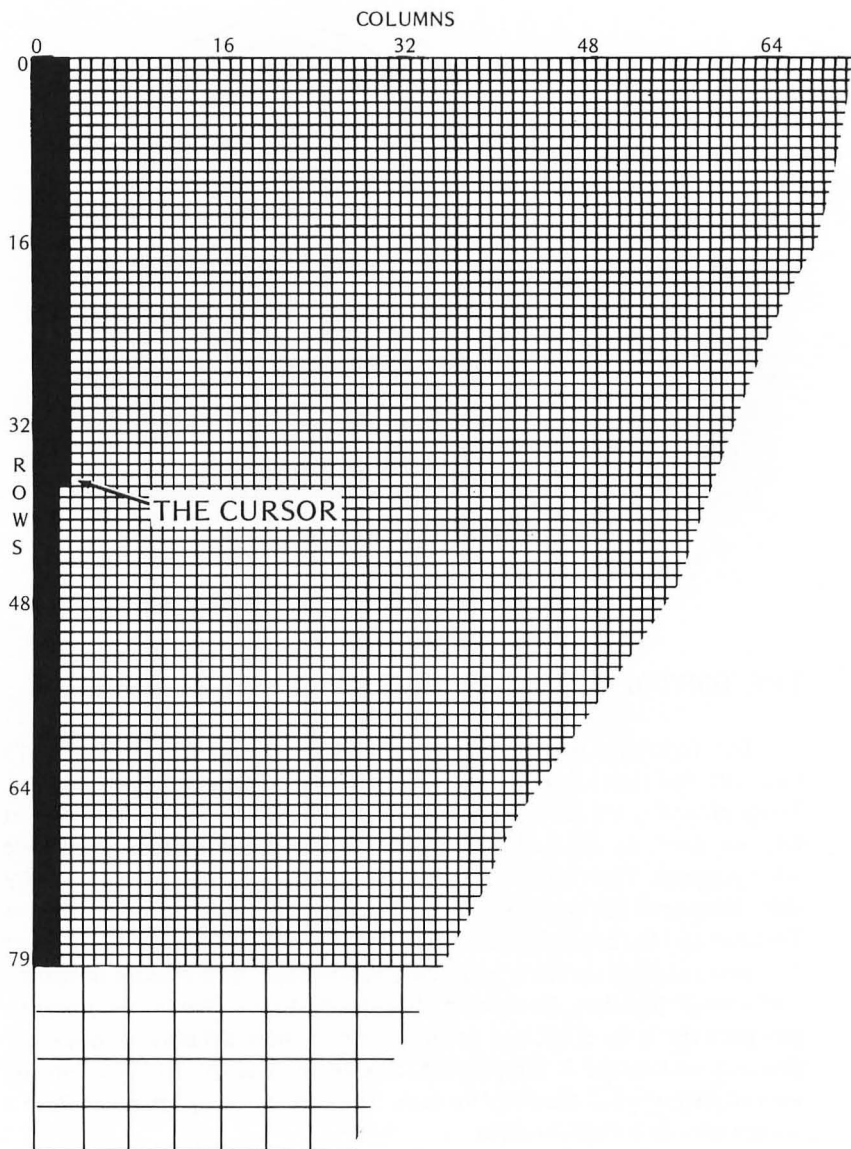
The following two programs are building blocks for the Tortoise's race with the Hare. Examine each one and try to guess what will happen. These programs are more complex than the ones you have encountered so far, but don't be alarmed if they confuse you. Try them out and notice what happens. Then look at the programs again and try to puzzle out why they happened the way they did. Why does one program represent the Tortoise and the other the Hare? Here are some hints that might be useful! The first program contains two FOR-NEXT loops. Think about how they could work together. Remember that the first loop (the X loop) begins, and then the Y loop follows. We'll explain in more detail later, but see if you can understand it for yourself. Remember that the REM statements are not part of what the program does. They are like program notes from a theater play or a music concert.



```

10 REM * THE TORTOISE FILLS THE SCREEN
15 REM *   STARTING FROM THE LEFT
20 GRAPHICS 7:COLOR 1
30 FOR X=0 TO 159
40 FOR Y=0 TO 79
50 PLOT X,Y
60 NEXT Y
70 NEXT X

```



This program will fill the entire screen, beginning at the upper left-hand corner and going down the 0 column until it reaches the bottom. Then it will begin at the top again and go down the 1 column. Here is a picture of what the screen would look like at the middle of column three.

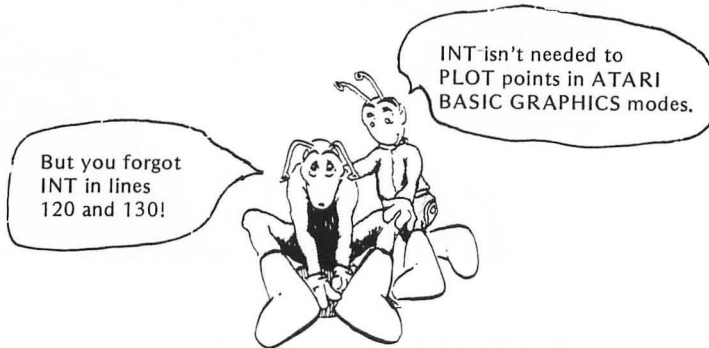
This program plods along, point by point, like the methodical Tortoise.



```

100 REM * THE HARE FILLS THE SCREEN
105 REM *   USING RANDOM POINTS
110 GRAPHICS 7:COLOR 1
120 PLOT RND(1)*159,RND(1)*79
130 DRAWTO RND(1)*159,RND(1)*79
140 GOTO 120

```

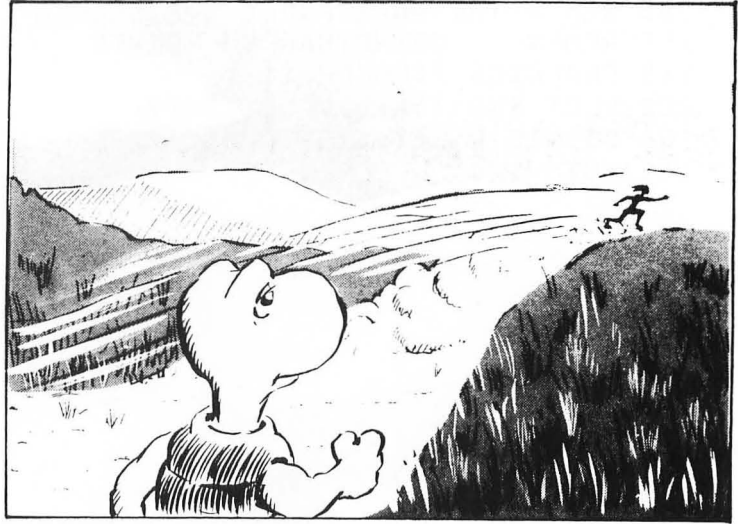


This program PLOTS a random point on the screen, picks another random point on the screen, and then draws a line from the first point to the second one. Using the GOTO loop at line 140, it then repeats the process all over again, filling the screen with random lines that dart about much like a Hare does.

The way we have set up these programs and given them line numbers makes it possible for you to run one after the other without typing NEW and clearing them from memory. If you type line numbers 10 through 70 and then RUN them, you will see the Tortoise training for the race first. Then, if you type lines 100 through 140 you will see the Hare going through its paces. 🐢



To stop either of these programs before they are finished, press the [BREAK] key. To continue, type CONT. The first program will eventually stop on its own, but the second program will continue indefinitely, even after the screen is filled, since there is no provision to stop it.



### THE RACE: VERSIONS 1, 2, AND 3

Now, for the race. We will take the two building blocks and combine them in a single program in which the Tortoise uses the left half of the screen and the Hare uses the right half. Thus all the points on the screen up to column 79 define the Tortoise's space, and all those from column 80 to 159 define the Hare's space. In addition, we will have to combine the moves of the Tortoise and Hare so that they run simultaneously. The way to do that is to put the Hare's moves inside of the loop that define the Tortoise's moves; so first the Tortoise will make one move, then the Hare will make one move, and then the loop will begin again.

Here is a program that sets up a race between the Tortoise and the Hare. It is the most difficult one you've encountered so far, so be patient with it.

Remember, we have changed the screen so that each creature only moves on one-half of the surface, and we've put the two creatures' moves inside the same loop.

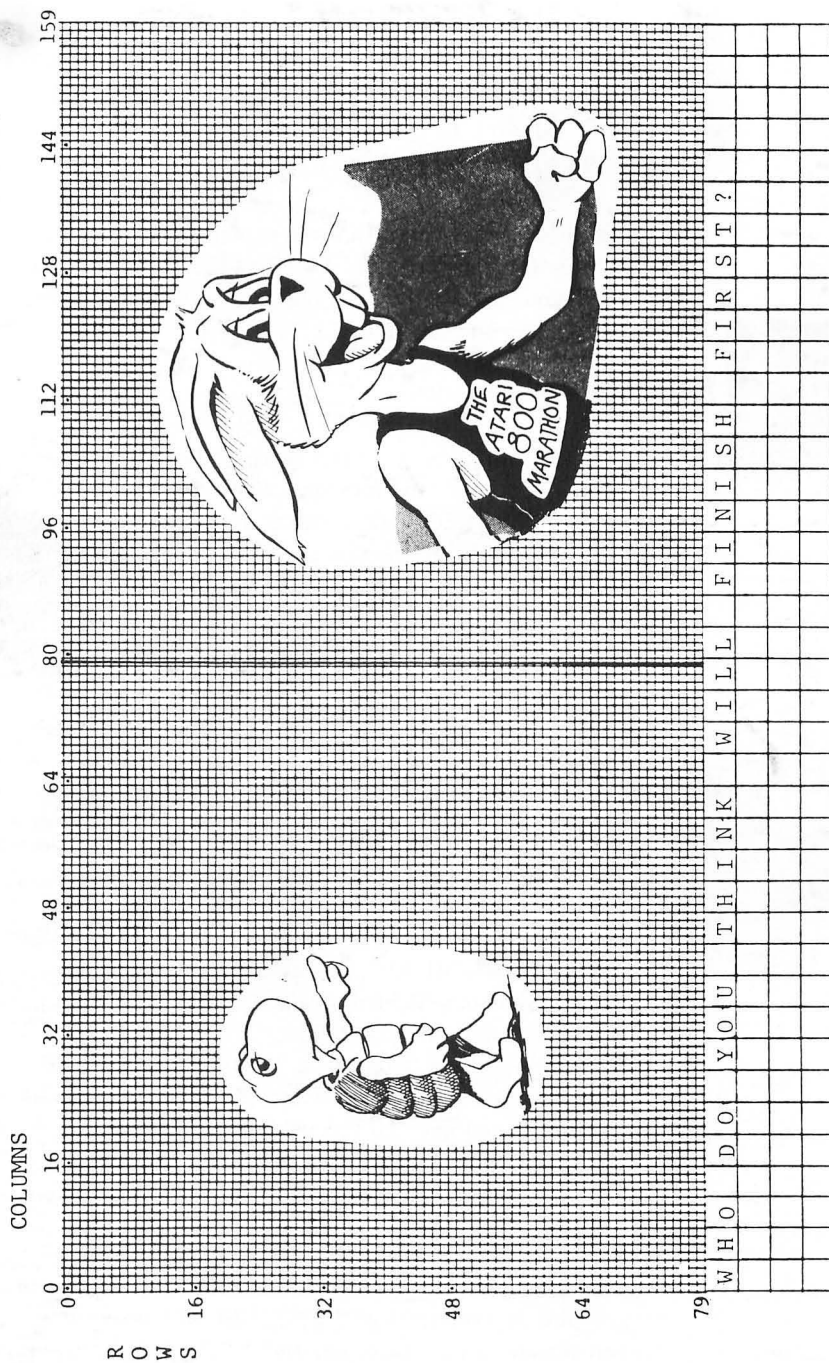
```

10 REM * THE TORTOISE & THE HARE
15 REM * VERSION 1
20 GRAPHICS 7:COLOR 1
25 PRINT "WHO DO YOU THINK WILL FINISH
   FIRST?"
30 FOR X=0 TO 79
40 FOR Y=0 TO 79
50 PLOT X,Y:REM * THIS IS THE TORTOISE

```

# Graphics Mode 6 or 7

with Text Window



**Notes:** Tortoise & Hare, Version 1

```

55 REM * AND HERE COMES THE HARE...
60 PLOT (RND(1)*79)+80,RND(1)*79
70 DRAWTO (RND(1)*79)+80,RND(1)*79
80 NEXT Y
90 NEXT X

```



Before we move on with some explanatory exercises, you may wish to save this program on your ATARI 410™ Program Recorder. To do this, load your recorder with a blank cassette tape, rewind to the beginning of the tape, and type:

CSAVE [RETURN]

Your ATARI computer will beep twice, indicating that you must depress *two* buttons on your tape recorder. When this happens, press down both the **PLAY** and **RECORD** buttons at the same time, and then press the [RETURN] key on the computer keyboard. When the word READY appears on the screen, your program has been recorded and saved for future use.

If you wish to reload the program at a later time, load this same tape into the recorder, rewind it, and type:

CLOAD [RETURN]



Your computer will then beep once, so press the **PLAY** button on the recorder and then press the [RETURN] key. Wait until the word READY appears on your screen; type LIST to see your program, and then RUN it.

To save a program on a diskette, using your ATARI 810™ Disk Drive, see the ATARI 810 Operator's Manual and the Disk Operating System (DOS) User's Manual, available from ATARI Home Computers, P.O. Box 427, Sunnyvale, CA 94086.

Now, let's go back and look at "The Tortoise & The Hare, Version 1" a little more closely. With the exception of line 25, we have almost the same program from lines 10 through 50 as we did previously on page 000. Line 50 will be done 6,400 times (80 times 80), plotting one point each time. The points are also plotted in a very rigid order: 0,0; 0,1; 0,2; . . . 0,79; 1,0; 1,1; . . . 1,79; etc.

Lines 60 and 70 are different from the previous program. They resemble lines 130 and 140 in the hare program on page 000, but we've changed the X values: (RND(1)\*79)+80 will produce a random value between 80 and 159. Therefore, this X value will always put a point on the right-hand side of the screen. This is what gives our program the effect of "running a race," since we are watching the slow, methodical Tortoise

(who is actually very fast for a tortoise!) filling up the left side of the screen, while the sporadic, frenzied Hare is randomly drawing lines, filling up the right-hand side. At first glance, it seems obvious which side is going to be filled first. But the initial advantage of the Hare is gradually eroded by the thoroughness of the Tortoise, who never misses a point!

Also note line 25. When you PRINT while in a GRAPHICS mode, the text is printed in the small "text window" at the bottom of the screen. This is an ideal spot for placing messages, and also for allowing you to INPUT values or PRINT information needed by the program. You could put whatever message you wanted in the PRINT statement at line 25 or leave it out entirely.

Note that we have two FOR-NEXT loops in our program. We call them "nested loops" because one loop is completely inside the other loop. The loop in which Y changes values is done over and over, until completion, for *each* value of X. Note also that for this reason, the statement NEXT Y must come before NEXT X.

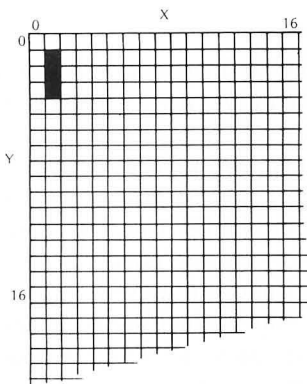
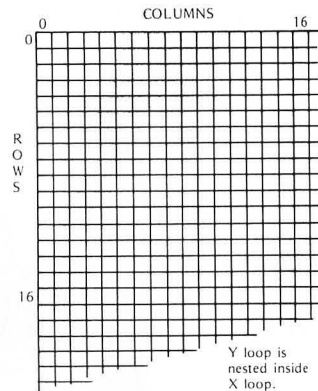
In order to understand the effect of nested loops, let's simplify the situation and pretend the screen is only 17 columns across by 17 rows down (each numbered from 0-16).

Now let's see what this program of nested loops will do:

```

10 GRAPHICS 7:COLOR 1: REM SIMPLIFIED
20 FOR X=1 TO 5
30   FOR Y=1 TO 3
40     PLOT X,Y
50   NEXT Y
60 NEXT X

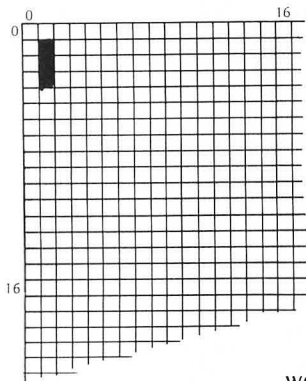
```



What this program says is to take the first value of X and then go to the Y loop and follow the commands in the Y loop. The Y loop goes around three times. So, for the first value of X (X=1), go through the Y loop as described in this chart and screen diagram:

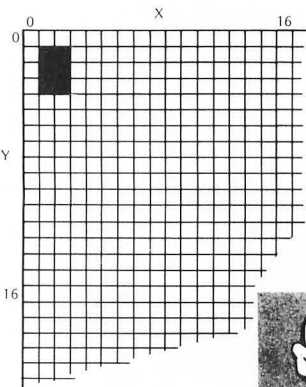
COLUMN X	ROW Y	Y LOOP SAYS PLOT X,Y
1	1	1,1
	2	1,2
	3	1,3

Now we are out of the Y loop and have to go to the next X. The next X takes us back into the Y loop with a new value of X:

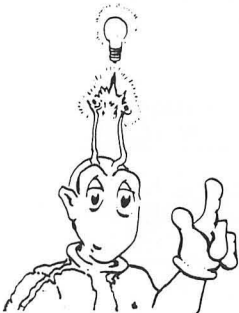
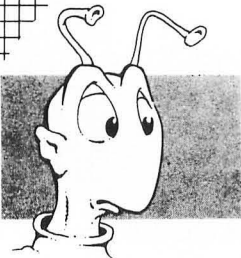
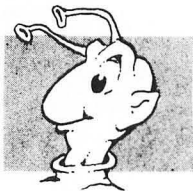


COLUMN X	ROW Y	Y LOOP SAYS PLOT X,Y
1	1	1,1
	2	1,2
	3	1,3
2	1	2,1
	2	2,2
	3	2,3

Complete the following chart and fill in the screen to show what it would look like when there are no more X values to graph and the program stops :



COLUMN X	ROW Y	Y LOOP SAYS PLOT X,Y
1	1	1,1
	2	1,2
	3	1,3
2	1	2,1
	2	2,2
	3	2,3
3	1	
4		
5		



Before going on, run the race a few times and share it with some friends. Will the Tortoise *always* win? How could we make the contest more intriguing? The answer to the first question is "No," but the odds are so overwhelming against the Hare winning that we probably wouldn't see a Hare victory for quite some time. However, we can make it appear like more of an even match by speeding our Tortoise up. Thus you could use this program with your friends as a friendly betting game. On first sight, most people will go for the Hare immediately! All we will do is to change lines 40 and 50 in Version 1 of the Tortoise and the Hare. Instead of having our Tortoise plot each and every point, we'll have it draw short line segments, very much like the Hare but in regular fashion.



```
40 FOR Y=0 TO 39
50 PLOT X,Y:DRAWTO X, Y+40
```

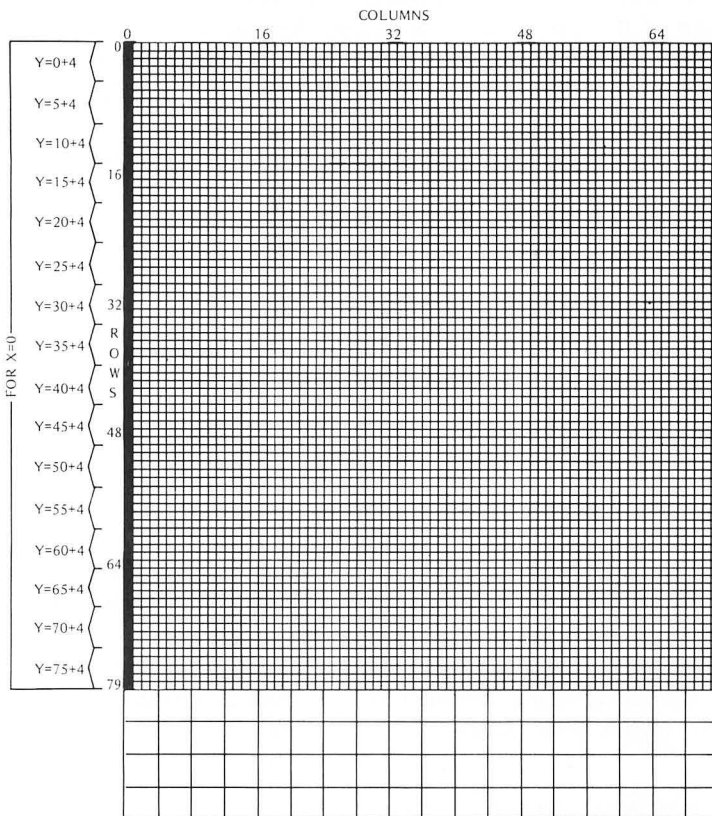
Now RUN this new version and see the difference. The Tortoise is still quite slow, but it will almost always come out the winner.

In order to speed up the Tortoise, we need to introduce a new part of the FOR-NEXT loop concept. This is the BASIC word STEP.

Change lines 40 and 50 so that they read as follows:

```
40 FOR Y=0 TO 75 STEP 5
50 PLOT X,Y:DRAWTO X,Y+4
```

What STEP does is change the rate at which Y increases each time through the loop. Instead of adding 1 to Y each time it gets to NEXT Y, this new program will add 5 to Y. Thus Y will take on the values 0,5,10,15,20,25, etc., all the way to 75.



The Tortoise will now be drawing short lines which don't overlap, thus saving a lot of time. It will still look like a sure win for the Hare, however, so the betting will probably stay the same. But if you play with lines 40 and 50 a little more, you can experiment until the visual contest looks about even. To start with, try this:



```
40 FOR Y=0 TO 70 STEP 10
50 PLOT X,Y:DRAWTO X,Y+9
```

We can also improve the visual appearance of the game by having messages print out during the game, at certain values of X. The listings on the next page show how to "dress this up" a little. It uses what could be called an "empty" FOR-NEXT loop. An empty loop creates a pause or delay in your program and might be called a "pause loop" or "time delay." Here is an example:



```
140 FOR WAIT=1 TO 800 : NEXT WAIT
```

Add this line to your program and RUN it. Notice the delay before the start of the race. You can use a pause loop to keep a message or image on the screen for a while or to slow down the operation of a program. Here are some examples to try and explore at a later time when we have finished with the race.

For example, what would this exasperatingly slow program do:

```
10 PRINT "HELLO";
20 FOR WAIT=1 TO 1500 : NEXT WAIT
30 GOTO 10
```

You can increase the speed of the same program by making the pause shorter—in this case by reducing 1500 to 50:

```
10 PRINT "HELLO";
20 FOR WAIT=1 TO 50 : NEXT WAIT
30 GOTO 10
```

Now, back to the Tortoise and the Hare. Notice that the next program listing looks different. It is actually an expansion of Version 1 with some added dressings.

This program listing has been specially reproduced to make it easy to read. We will be using this method of reproducing programs throughout the rest of this book. Some of the lines in the printout will be longer than



WAIT is just a long variable name, like X or Y or A.



the number of spaces on your screen. When this happens your ATARI Home Computer will split the line and continue it on the next line automatically. Most of the special function keys will be set off in brackets, and you will need to remember to press the [RETURN] key at the end of each logical line. Remember, each new logical line begins with a line number. Finally, the REM statements are optional. They are included to help you understand different parts of the program, but they are not necessary.



Whenever you see [CLEAR] in the listing of a program, this means:  
Press the [ESC] key, then hold down the [SHIFT] key and at the same time press the [CLEAR] key.

```

10 REM *          TORTOISE & THE HARE
20 REM *          VERSION 2
30 REM *          COPYRIGHT (C) 1980 BY
40 REM * TED M. KAHN & HERBERT R. KOHL
50 REM *
100 GRAPHICS 7:COLOR 1
120 PRINT "[CLEAR]A RACE: THE TORTOISE
    VS. THE HARE"
140 FOR WAIT=1 TO 800:NEXT WAIT
145 REM * THE ABOVE LOOP DOES NOTHING
    BUT DELAY BETWEEN PRINT STATEMENTS
180 PRINT "[CLEAR]ON YOUR MARKS, GET
    SET, GO!!!"
190 FOR WAIT=1 TO 500:NEXT WAIT
200 REM * THE RACE
210 FOR X=0 TO 79
215 REM * THE TORTOISE
220 FOR Y=0 TO 75 STEP 5
250 PLOT X,Y:DRAWTO X,Y+4
260 IF X=10 THEN PRINT "[CLEAR] PLACE
    YOUR BETS, FOLKS!"
262 IF X=40 THEN PRINT "[CLEAR]WANT TO
    CHANGE YOUR BETS?..."
264 IF X=60 THEN PRINT "[CLEAR] IT'S
    DOWN TO THE WIRE, FOLK..."

```





```

266 IF X=75 THEN PRINT "[CLEAR]AND THE
      WINNER IS...???"
268 IF X=79 THEN PRINT "[CLEAR] GUESS
      WHO!!!"
300 REM * THE HARE
310 PLOT (RND(1)*79)+80,RND(1)*79
320 DRAWTO (RND(1)*79)+80,RND(1)*79
330 NEXT Y
340 NEXT X
400 PRINT "[CLEAR] ONCE AGAIN, ORDER
      TRIUMPHS OVER CHAOS!"
500 FOR WAIT=1 TO 1000:NEXT WAIT

```

Lines 120, 180, 260, 262, 264, 266, 268, and 400 all contain PRINT statements with messages which will appear as comments in the text window when you RUN the program. When embedded in a PRINT statement [CLEAR], which means you first press down the [ESC] key and then hold down the [CTRL] key and at the same time press the [CLEAR] key, serves to clear the text window before the next comment appears.

Now, before moving on to the next program you may wish to save the last program on your ATARI 410™ Program Recorder or your ATARI 810™ Disk Drive.

Your patience and perserverance are rewarded! Here is Version 3 of the race between the Tortoise and the Hare. You may not immediately understand this program, but it will show you some of the things you can do with your ATARI Home Computer.

This version uses two colors and has a "title page." It also shows you an example of some of the things you will be able to do after reading the later sections of this book. Copy the program, RUN it, and add it to your collection.



Your ATARI Home Computer will display 24 lines of text on the screen at one time. Text is moved up and off of the screen when more than 24 lines appear in a program LISTing. To stop the program from scrolling off the top of the screen, press and hold down the [CTRL] key and at the same time type the number 1. To continue the LISTing, just press the [CTRL] key and type 1 again. Also, the program listings in this book conform to the standard margin settings on your ATARI Computer—that is, there are 38 characters per line on the screen. One printed line is the same as 1 line on the screen.



```

10 REM *      TORTOISE & THE HARE
20 REM *      VERSION 3
30 REM *      COPYRIGHT (C) 1980 BY
40 REM *      TED M. KAHN & HERBERT R. KOHL
50 REM *

```

```

100 GOSUB 1000:REM * ADD A TITLE PAGE
110 GOSUB 1100:REM * ADD SUSPENSE
120 GRAPHICS 7
125 POKE 752,1:REM * TURN CURSOR OFF
130 SETCOLOR 0,3,4:SETCOLOR 2,15,4
135 PRINT "WHO WILL FINISH FIRST?"

```



```

200 REM *
202 REM * THE RACE BEGINS...
204 REM *
210 FOR X=0 TO 79
220 FOR Y=0 TO 80 STEP 4
230 REM *
232 REM * THIS IS THE TORTOISE
234 REM *
240 PLOT X,Y:DRAWTO X,Y+3
250 REM *
252 REM * AND HERE IS THE HARE
254 REM *
270 COLOR 3
280 PLOT (RND(1)*79)+80,RND(1)*80
290 DRAWTO (RND(1)*79)+80,RND(1)*80
300 IF X=20 THEN GOSUB 1200
310 IF X=40 THEN GOSUB 1400
320 IF X=60 THEN GOSUB 1600
350 NEXT Y
360 NEXT X
400 GOSUB 2000:REM * WINNING MESSAGE
402 REM * NOTE THAT THIS PROGRAM
404 REM * ASSUMES THE TORTOISE WILL
406 REM * ALWAYS WIN! WILL IT?!
408 REM *
410 GOSUB 5000:REM * PAUSE
990 PRINT "THE END"
995 POKE 752,0:REM * TURN CURSOR ON
999 END :REM * END OF MAIN BODY OF
      PROGRAM AND BEGINNING OF SUBROUTINES

```



```

1000 REM * ADD A TITLE PAGE
1010 GRAPHICS 1
1015 POKE 752,1:REM (CURSOR OFF)
1020 POSITION 4,3
1022 PRINT #6;"THE TORTOISE"
1024 POSITION 8,5:PRINT #6;"AND"
1026 POSITION 6,7:PRINT #6;"THE HARE"
1099 RETURN :REM BACK TO MAIN PROGRAM
1100 REM * INITIAL SUSPENSE MESSAGE
1110 PRINT "[CLEAR]ON YOUR MARK..."
1115 GOSUB 5000
1120 PRINT "[CLEAR]GET SET..."
1125 GOSUB 5000
1130 GRAPHICS 2+16:REM (BLACK-OUT)
1135 POSITION 8,6:PRINT #6;"GO!!"
1140 GOSUB 5000
1199 RETURN :REM RETURN TO PROGRAM
1200 REM * INTERMITTENT MESSAGES
1210 PRINT "[CLEAR]      ARE YOU STILL
      SURE?"
1299 RETURN
1400 REM * MORE INTERMITTENT MESSAGES
1410 PRINT "[CLEAR]DO YOU WANT TO CHAN
      GE YOUR BET?"
1499 RETURN
1600 REM * ANOTHER SUSPENSE MESSAGE
1610 PRINT "[CLEAR]WHAT A THRILLER!
      WHAT SUSPENSE!"
1699 RETURN
2000 REM * THE WINNER
2010 PRINT "[CLEAR]  THE TORTOISE WINS
      AGAIN!"
2099 RETURN
5000 REM * DELAY (PAUSE) ROUTINE
5010 FOR WAIT=1 TO 600:NEXT WAIT
5099 RETURN

```



You'll notice that some of the BASIC vocabulary and programming ideas are unfamiliar. They are more advanced, and it is beyond the scope of this book to explain all of them. Still, you can copy and RUN the advanced programs and use them in your games.

The new BASIC words, GOSUB and RETURN, will be explained in the next chapter. Just briefly, the GOSUB statement tells the computer to go to the line number indicated, just like a GOTO statement. When the program reaches a corresponding RETURN statement (not to be confused with the [RETURN] key), it goes back to where it left off from the main body. This way, you don't have to keep track of where you left off and where to resume. Writing programs in this way is called *modular programming* and is generally accepted as good programming style.



For example, after RUNning this program you will probably have a better idea of how simple games can be made more complex as well as dressed up and made more attractive. We'll end this chapter with a number of dressings you can add to games. Many involve BASIC vocabulary and programming concepts we have already introduced. You should be able to create your own version of these dressings as well as use some of the more complex ones whose structure you may not completely understand.

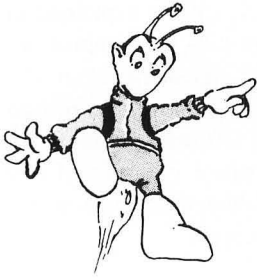


## Review:

## Concepts

**Text Window**—When you PRINT while in a GRAPHICS mode, the text is printed in a small “text window” at the bottom of the screen. This is an ideal place for placing messages, and also for allowing you to INPUT values or information needed by the program.

**Nested Loops**—The ability to use more than one loop, sometimes having loops within other loops, is one of the most powerful features of a computer programming language. It saves a lot of extra programming and makes it easy to do a lot of tasks, especially those which require going through the same operation over and over for different values. However, for the beginning programmer, it can be confusing. The following is an example of a nested loop:



```

10 REM NESTED LOOP
20 GRAPHICS 7:COLOR 1
30 FOR X=0 TO 159
40 FOR Y=0 TO 79
50 PLOT X,Y
60 NEXT Y
70 NEXT X

```

Note that the NEXT Y statement comes before the NEXT X statement. This is because the Y loop must be completed before the NEXT X value will be changed. Thus, while X=0, this loop changes the value of Y from 0, 1, 2, 3, 4, all the way to 79. It does line 50 eighty times before changing X to 1; then it does it 80 more times before the NEXT X value. All told, line 50 is executed 12,800 times in the course of these two loops. . . and at a very fast pace!

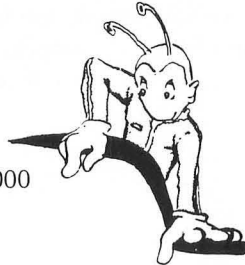
**Delay (Wait) Loop**—A delay or pause is an “empty” FOR-NEXT loop. It causes the computer to count silently to itself while doing nothing else. You can control the pause in action by varying the number of inactive cycles the computer goes through.

Here is a slow loop:

```

10 FOR WAIT=1 TO 2000
20 NEXT WAIT

```

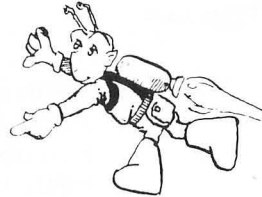


This one is faster:

```

10 FOR WAIT=1 TO 200
20 NEXT WAIT

```



## BASIC Vocabulary

**STEP**—The STEP function creates regular jumps in numerical values and is used to modify FOR-NEXT loops. Thus:

```

10 FOR X=1 TO 5 STEP 2
20 PRINT X
30 NEXT X

```



would begin a 1, then STEP up by 2 to 3 and then STEP up another 2 to 5. When RUN this program will print:

1  
3  
5

This process of regularly changing the value of a variable is also called *incrementing*.

Here is another example:



```
10 FOR X=1 TO 10 STEP 5
20 PRINT X
30 NEXT X
```

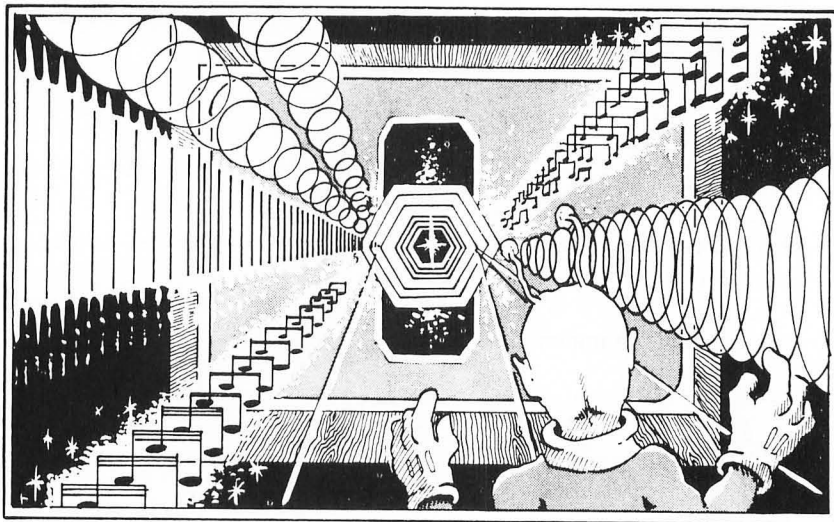
This program will print out:

1  
6

because 1 is the first number and  $1+5=6$ . No other numbers will be printed since  $6 \text{ STEP } 5$  is 11. Eleven is out of the range of X whose highest value is 10 as stated in line 10 of this program.

### Special Keys

**[ESC] [CLEAR]**—Using the [ESC] key allows you to put commands normally used in the Direct mode (executed immediately) into the Deferred mode (executed within a program). To clear the screen within your program, do the following after the program line number. Press the [ESC] key once. Then hold the [CTRL] key down and at the same time press the [CLEAR] key. This procedure will be shown as just [ESC] [CLEAR] within program listings used in this book.

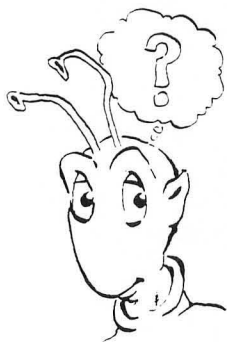


## DRESSING UP GAME PROGRAMS

Have you ever seen one of the new electronic scoreboards that many stadiums and arenas have installed recently? A player hits a homerun or scores a touchdown and the scoreboard explodes, flashes the player's name in a dozen colors, sets off sirens, and runs cartoons. The play is dressed up by the display on the scoreboard. In the same way you can dress up the games you program using your ATARI Home Computer. There are probably no special effects that those fancy boards can produce that you can't program on your computer. The only limitations you face are the size of your TV screen and the power of its loudspeakers.

In this section we will introduce you to a number of ways of dressing up your game programs: for example, adding fancy endings, introducing the game in unusually interesting ways, applauding winners, and encouraging losers to try again. The tone and interest your games create can be enhanced by the way you dress them. Our suggestions are only a start. Invent your own beginnings and endings, try a dozen for the same game, and test them out on friends to find those that add the most fun and style to the game.

Start by using this dressing. It is a reinforcing message for the winner, printed in large letters. To get large text on the ATARI computer, we can use either of two special GRAPHICS modes: GRAPHICS 1 or GRAPHICS 2. GRAPHICS 1 prints letters which are standard height and double width. GRAPHICS 2 prints text which is double height and double width. In order to print text in these modes, we have to add something to a standard PRINT statement. Instead of PRINT "TEXT", we use PRINT #6;






"TEXT". This is because we are printing to a special kind of GRAPHICS screen or "device" which your ATARI Home Computer knows as device #6. Try this example:

```
10 GRAPHICS 1
20 PRINT #6; "HELLO"
```

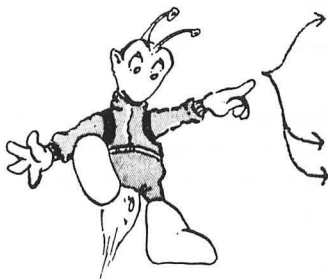


The word "HELLO" will print in the upper-left corner of the screen. The ATARI computer can also PRINT large lower-case letters, but this requires some more advanced concepts, so we won't introduce it now. You can actually PRINT the text in small letters by pressing the [CAPS/LOWR] key. However, they will appear on the screen as capital letters, but in a different color in GRAPHICS 1 and 2. Try this:

```
PRINT #6; "hello" 
```

We can also specify where we want the text printed on the screen by using the POSITION statement. In GRAPHICS 1 mode, we have 20 available horizontal locations (numbered 0-19) and 20 available vertical locations, not including the text window at the bottom (numbered 0-19).

Here's a very simple program ending which congratulates the winner of a game. Using line numbering conventions just introduced in this chapter, this "dressing" will begin at line 2000.



```
2000 REM * CONGRATULATIONS
2002 REM *   TO THE WINNER
2004 REM *   GRAPHICS 2+16
2006 REM *
2010 GRAPHICS 2+16
2020 POSITION 6,5
2030 PRINT #6;"YOU WON!!"
2040 FOR WAIT=1 TO 1000:NEXT WAIT
```

In line 2010, GRAPHICS 1 has been changed to GRAPHICS 2 (extra large letters), and the number 16 has been added. This gives us a complete GRAPHICS 2 screen, without the small text "window" at the bottom. The



Be sure to always return to the standard keyboard mode by pressing the [SHIFT] and [CAPS/LOWR] keys after you've finished typing lower case letters!

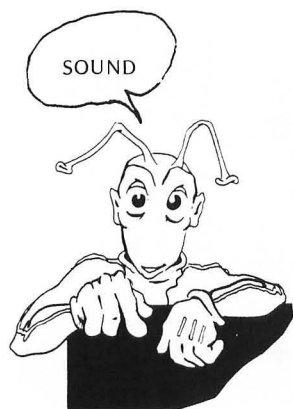
same thing can be done with GRAPHICS 1: Change line 2010 to GRAPHICS 1+16 and RUN this ending again. How are the two different?


Line 2020 POSITIONs the beginning of the message at a particular location on the screen. It uses the same idea as PLOT X,Y. X is the column number, Y is the row number, and the 0,0 point is located in the upper-left corner of the screen. In GRAPHICS 2 mode, line 2020 tells the computer to start at 6 character-size spaces from the left margin and 5 spaces down from the top of the screen.



Here is a summary of the permissible values for POSITIONing large text in GRAPHICS modes 1 and 2.

GRAPHICS MODE	X	Y
GRAPHICS 1	0 - 19	0 - 19
GRAPHICS 1+16 (full screen)	0 - 19	0 - 23
GRAPHICS 2	0 - 19	0 - 9
GRAPHICS 2+16 (full screen)	0 - 19	0 - 11



Another way to liven up a winning message is to use the ATARI BASIC SOUND command.  Here's a short ending which uses standard-size text, along with musical reinforcement. Be sure to turn up the volume on your TV set so that you can hear it!

```
2100 REM * CONGRATULATIONS
2102 REM *   WITH SOUND
2104 REM *
2110 FOR I=1 TO 100
```




Look at Part III, Chapter Seven, for a detailed description of the SOUNDX, Y, Z, W command.



```

2115 POSITION INT(RND(1)*30),INT(RND(1)*23)
2120 PRINT "YOU WON!";
2130 SOUND 0,RND(1)*250,10,8
2140 NEXT I

```

Now let's look at dressing up the beginning of a program. Here is how to put a title page at the beginning of your game. You can use this program as a simple, no-frills way of introducing your game to the user. The program name is displayed on the screen in double-size text (lines 20 and 30) until the [START] key is pressed and then the program continues. PEEK(53279)  in line 100 is a command that tells the computer to check to see if any of the special console keys, [START], [SELECT], or [OPTION], have been pressed. The [START] key has a value of six; if the key has not been pressed, the computer goes back and checks again until it (the [START] key) has been pressed.

Type NEW and enter this program:

```

10 REM * TITLE PAGE FOR PROGRAM
12 REM *
18 REM * DOUBLE SIZE TEXT WITH WINDOW
20 GRAPHICS 2
28 REM * SET CURSOR AT COLUMN 3,LINE 4
30 POSITION 3,4
38 REM * PRINT PROGRAM NAME IN
40 PRINT #6;"PROGRAM NAME"
98 REM * WAIT UNTIL START IS HIT
100 IF PEEK(53279)<>6 THEN GOTO 100
198 REM * PROGRAM NOW STARTS HERE
200 PRINT "THANK YOU."

```



You may know about pressing the [START] key, but perhaps another player won't. You can let them know by adding a line of instruction to your program. Press the [SYSTEM RESET] key, LIST your program, and add the following line:

```
50 PRINT "PRESS START KEY TO BEGIN"
```

Now let's explore ways to dress up that title page. First, let's make the background of the screen and the text window the same color (black).





The PEEK(X) command is more complex than the ones we'll explain in this beginning book. If you already understand PEEK(X), however, there are some PEEK numbers that are useful for game programming in Appendix III.

Just add this line:

```
25 SETCOLOR 2,0,0:REM TEXT WINDOW BLACK 
```



Next, you can have the color of the letters in your program name change in a pulsating effect. This is very simple. Just add these two lines:

```
52 SETCOLOR 0,HUE,6: REM COLOR REGISTER  
CONTROLLING THE LETTERS OF NAME  
60 HUE=HUE+1  
```

Line 100 must also be changed to this:

```
100 IF PEEK(53279)<>6 THEN 52
```

Note that the effect is quite nice. If you would like the colors to change at a slower rate, just add a delay loop as follows:

```
90 FOR WAIT=1 TO 200:NEXT WAIT
```

Now, you don't have to change the color of the letters. You could change the background color instead. Just change the 0 in line 52 to a 4, so it reads:

```
52 SETCOLOR 4,HUE,6:REM BACKGROUND COLOR
```

Notice, however, that the text window has now returned.

But wait, there is more. How about having the title blink off for half a second before changing color? This also is easy to arrange! Just turn line 52 into line 55 and add a new line 52 and 53. Be sure to return line 55 back into its original form. Now we have:



For a detailed explanation of the SETCOLOR X, Y, Z command, see Part III, Chapter Eight.




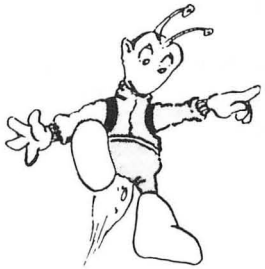
HUE, like WAIT, is just a variable name, like X, Y, or A. ATARI BASIC allows you to use long variable names (up to 255 characters—but no spaces) so that you can read and understand your programs more easily.

```

52 SETCOLOR 0,0,0:REM * BLANK OUT THE
   TITLE
53 FOR WAIT=1 TO 100:NEXT WAIT:REM *
   THIS IS A DELAY LOOP
55 SETCOLOR 0,HUE,6:REM * PUT COLOR
   BACK INTO TITLE

```

How would you like your title to be displayed upside down every other blink? This only sounds complicated. It's all a matter of one POKE.  POKE 755,4 turns all characters on the screen upside down. POKE 755,2 returns the characters back to normal. Just add these lines for this unique effect:



```

45 FLIP=2:REM * SET INITIAL VALUE FOR
   'FLIP' VARIABLE TO UPSIDE DOWN
46 REM * NOTE: 'FLIP=2' IS SHORT FOR
   'LET FLIP=2'
70 POKE 755,FLIP:FLIP=FLIP-6:REM *
   NOW FLIP TITLE UP AND DOWN
101 POKE 755,2:REM * RETURN CHARACTERS
   TO NORMAL BEFORE STARTING PROGRAM

```

If upside down is a bit too much for you, how about switching it to lower-case letters and then back to upper-case letters? This is just as easy as upside down. Just add the following lines:

```

45 FLIP=224:REM INITIALIZE VARIABLE FOR
   FLIP (POKE)
70 POKE 756,FLIP:FLIP=450-FLIP:REM FLIP
   TITLE UP & DOWN
101 POKE 756,224:REM RETURN CHARACTERS
   TO NORMAL UPPER CASE MODE

```



POKE is a special command which places a numerical value directly into a specified location in the ATARI computer's memory. As such, it's like a BASIC LET X= statement. However, POKE is used to do things you cannot do in ATARI BASIC like flipping letters in the title upside down. For more information of POKE and its inverse, PEEK, see the Appendix and the ATARI BASIC Reference Manual. POKE should be used with some care, since it may cause changes to your program which are undesirable if you don't understand what you are doing.

This change now gives a title in alternating upper-case and lower-case letters. However, it also results in an interesting and usually undesirable side effect: Whenever the title appears in lower-case letters, so do a lot of colored hearts! Now, if you like this effect, there is no problem. If you don't, there is unfortunately no simple way to get rid of the hearts due to the way your ATARI Home Computer's character codes are organized. We can use this feature to understand a little more about color changes. We won't explain exactly how these changes work here, but you may want to try the following modifications to your program and refer to Chapter Eight for more examples of how colorful your ATARI Home Computer can be!

Retype line 40 as follows:

```
40 PRINT #6; "program name"
```

To get lower-case letters, press the [CAPS/LOWER] key once before typing the letters you want. To get back to upper-case letters, hold down either [SHIFT] key and press the [CAPS/LOWER] key again. This key is like a switch; it turns upper/ lower case on and off.

Now, RUN the program. What happened? The hearts are still there half of the time, but the title is always the same color. Typing the title in lower-case letters did not put the letters in lower case.

Change line 55 to read:

```
55 SETCOLOR 1,HUE,6
```

This causes a color change in the title. Now enter this line and RUN the program again. What happens?

```
52 SETCOLOR 0,0,0
```

Did your program work the way you thought it would? Or, did you get lost somewhere along the way? In this next program we have combined many of the lines from the past few pages so that you can use them to create an interesting title page for one of your programs. You can try it out now or come back to it later.

```
10 REM * TITLE PAGE FOR YOUR PROGRAM
12 REM *
18 REM * DOUELE-SIZE TEXT WITH WINDOW
20 GRAPHICS 2
24 REM * SET BACKGROUND COLOR TO BLACK
25 SETCOLOR 2,0,0
30 POSITION 3,4
```

```
40 PRINT #6;"PROGRAM NAME"
45 FLIP=224
50 PRINT "PRESS START KEY TO BEGIN"
52 SETCOLOR 0,0,0
53 FOR WAIT=1 TO 200:NEXT WAIT
55 SETCOLOR 0,HUE,6:REM * COLOR TITLE
60 HUE=HUE+1
70 POKE 756,FLIP:FLIP=450-FLIP
90 FOR WAIT=1 TO 200:NEXT WAIT
100 IF PEEK(53279)<>6 THEN GOTO 52
101 POKE 756,224
200 REM * PROGRAM THEN BEGINS HERE
```

## GETTING MORE COMPLEX

At this point, we would like to show you some really complex ways to dress up your programs. However, in order to do this we need some special notation that deals with the complex features of your ATARI 400 or ATARI 800 Home Computers, such as: inverse video letters (letters which appear as dark on a white background instead of the usual white on dark); graphic symbols; and control characters that allow you to print anywhere on the screen (up, down, left, and right). The Code Works™, publishers of IRIDIS™ have created excellent programs and documentation for the ATARI Home Computer. They have developed a special system of notation to deal with these features, and they have been kind enough to give us permission to use their system for the programs in the book which use these special features.

The notation has two basic rules: anything underlined is in inverse video, and anything in brackets is special. Anything else you see is just what it appears to be.

A single character or punctuation mark in brackets represents a graphics character. Enter these characters by holding down the control key (marked CTRL) while pressing the key specified in the brackets. For example, [S] is a control-S and [.] is a control-period. A control-S will appear on the screen as a cross. A control-period is a diamond. When a graphics character is to be displayed in inverse video, the character in the brackets will be underlined. [S] represents the graphics heart in inverse video.

A word in brackets denotes a special control function. [CLEAR] will clear the screen. [BELL] buzzes the console speaker. To enter the [CLEAR] character you must press the [ESC] (ESCAPE) key once and then hold down the shift while you press the CLEAR key. On the screen the [CLEAR] character will look like a curved arrow pointing to the left. To enter the [BELL] character you would again press the [ESC] key; then hold the CTRL (CONTROL) key while you press the number 2 key. The following chart explains how to enter all of the special function characters.

The programs listed in this book differ from CODEWORDS™ listings in only one way: this is when the same special character is entered more than once consecutively. The CODEWORKS™ uses 3[,] to represent three graphics 'heart' characters. The listings in this book represent three 'hearts' like this: [,][,][,]

---

If you're interested in finding out more about IRIDIS™, here's the address:

The Code Works™  
P.O. Box 550  
Goleta, California 93017



Atari:	Us:	You type:
␣	{A}	ctrl-A
␣	{B}	ctrl-B
␣	{C}	ctrl-C
␣	{D}	ctrl-D
␣	{E}	ctrl-E
␣	{F}	ctrl-F
␣	{G}	ctrl-G
␣	{H}	ctrl-H
␣	{I}	ctrl-I
␣	{J}	ctrl-J
␣	{K}	ctrl-K
␣	{L}	ctrl-L
␣	{M}	ctrl-M
␣	{N}	ctrl-N
␣	{O}	ctrl-O
␣	{P}	ctrl-P
␣	{Q}	ctrl-Q
␣	{R}	ctrl-R
␣	{S}	ctrl-S
␣	{T}	ctrl-T
␣	{U}	ctrl-U
␣	{V}	ctrl-V
␣	{W}	ctrl-W
␣	{X}	ctrl-X
␣	{Y}	ctrl-Y
␣	{Z}	ctrl-Z

Atari:	Us:	You type:
␣	{,	ctrl-comma
␣	{.}	ctrl-period
␣	{;}	ctrl-semicolon
␣	{BACK}	ESC BACK
␣	{BELL}	ESC ctrl-2
␣	{CLEAR}	ESC shift-less or ESC ctrl-less
␣	{CLR TAB}	ESC ctrl-TAB
␣	{DEL CHAR}	ESC ctrl-BACK
␣	{DEL LINE}	ESC shift-BACK
␣	{DOWN}	ESC ctrl-equals
␣	{ESC}	ESC ESC
␣	{INS CHAR}	ESC ctrl-greater
␣	{INS LINE}	ESC shift-greater
␣	{LEFT}	ESC ctrl-plus
␣	{RIGHT}	ESC ctrl-star
␣	{SET TAB}	ESC shift-TAB
␣	{TAB}	ESC TAB
␣	{UP}	ESC ctrl-minus

----	Note	----
comma	is ,	
equals	is =	
greater	is >	
less	is <	
minus	is -	
period	is .	
plus	is +	
semicolon	is ;	
star	is *	

How Atari characters  
appear in Iridis listings.

Now, here's a program that uses some of these special features. Notice that line 30 contains the word START with underlined letters. This means that you press the key with the ATARI logo on it and that the letters underlined will appear on the screen in inverse video (for highlighting). Be sure to press the inverse video key *again* after typing this word in order to return to normal character mode.

```

10 REM * TITLE DISPLAY
12 REM *
20 GRAPHICS 2:REM * LARGE TEXT
30 PRINT "PRESS START TO BEGIN":REM *
   THIS IS PRINTED IN THE SMALL TEXT
   WINDOW AT BOTTOM OF SCREEN
40 POSITION 3,4:PRINT #6;"PROGRAM NAME"
98 REM * CHECK TO SEE IF START
100 IF PEEK(53279)<>6 THEN GOTO 40
198 REM * PROGRAM BEGINS NOW
200 PRINT "THANK YOU."
```

First, let's add line 25 to set the text window color to black like the main screen background:

```
25 SETCOLOR 2,0,0:REM TEXT WINDOW BLACK
```

Now let's have the title start at the top of the screen and move down. Change line 100 to THEN 29 and the 4 in line 40 to ROW:

```
40 POSITION 3,ROW:PRINT #6;"PROGRAM NAME";
100 IF PEEK(53279)< >6 THEN GOTO 29
```

Add these lines:

```
29 PRINT #6;"[CLEAR]";:REM CLEAR
   SCREEN
50 ROW=ROW+1:IF ROW>9 THEN ROW=0
```

Remember that the [CLEAR] in line 29 means that you are to press the [ESC] key and then press the [CTRL] and [CLEAR] keys together. This allows the text window to be cleared within your program.

Now your program title races down the screen, probably a bit faster than you would like. So let's add a delay:

```
90 FOR WAIT=1 TO 200:NEXT WAIT
```

You can have your title page change colors as it goes down the screen by adding this line:

```
80 SETCOLOR 0,0,ROW*17+238:REM CHANGE  
THE COLOR OF TITLE
```

Notice that we are changing the color by changing the luminance level. It seems that a luminance level over 14 will affect the color as well. The effect is quite pleasing and something you can well take advantage of.

One more effect you might like to see in action. Add line 1 to any of the above programs:

```
1 POKE 77,128:REM SCREEN PROTECTION ON
```

Now the built-in screen-protection color change begins. Interesting! Or try using the value 126 instead of 128 for a few seconds delay before it kicks in.

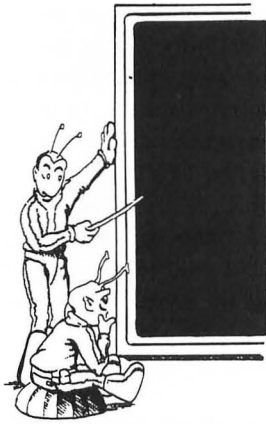
That's it for spicing up a simple title page. Experiment with the information on the last few pages, make a few changes, explore the possibilities—you take it from here!





## Chapter 3

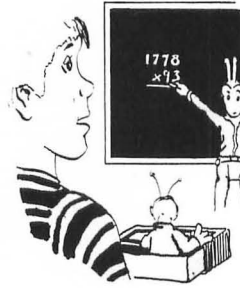
# Drill and Basic Skills



Drill is a bore! Number facts, times tables, spelling lists, and other mechanical aspects of learning require repetition and memorization. About the only pleasure they provide derives from having mastered them to the point where you don't have to practice them again. Using your ATARI computer, it is possible to set up drill programs and dress them up to make drill more interesting than using flash cards and memorizing lists.

There are a number of drill-and-practice programs that can be adapted to the particular needs of children. The first part of this chapter will deal with number facts: addition, subtraction, multiplication, division, and algebra. The second half of this chapter will include some simple programs to help with the practicing of spelling, vocabulary, and grammar.

We will start with programs that use some of the BASIC vocabulary already introduced and show how they can be modified to help children with the mechanical facts of reading, writing, and arithmetic. Then we'll present a number of more sophisticated programs which use games to provide the same practice that drill does.

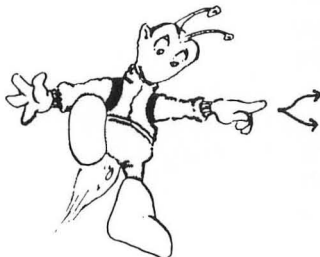


## NUMBER FACTS

This first program should interest four-, five-, and six-year-olds. In this program the computer puts a number on the screen and asks the player to INPUT the next number. If the player guesses the right number, another number is put up for guessing. If not, the computer types "TRY AGAIN" and gives the player another chance. In this program, as in most good interactive programs, a message or 'prompt' is PRINTed before you INPUT your response. Thus, in line 120 the computer PRINTs, "WHAT NUMBER COMES RIGHT AFTER \_\_\_\_\_", where the blanks are replaced by the value of X. The variable X gets its value using the RND (random number) function at line 110 and will be an integer between 0 and 10. When you see the "?" appear on the screen, the computer is awaiting the INPUT of your answer.

Remember to include "prompts" in your games that tell the player what he or she is supposed to do.

RUN this simple program; it will continue to RUN until you press the [BREAK] or the [SYSTEM RESET] key. It is a "bare-bones" version of many different kinds of programs which involve practicing number facts or guessing an unknown number.



```

10 REM * FIRST DRILL FOR LITTLE ONES
12 REM * (C) 1980 BY HERBERT KOHL
14 REM * (no dress-up)
16 REM *
100 PRINT "[CLEAR]"; REM * CLEAR SCREEN
105 REM * CHOOSE RANDOM NUMBER 0 - 10
110 LET X=INT(RND(1)*11)
120 PRINT "WHAT NUMBER COMES RIGHT AFTER
    ";
130 PRINT X;
200 REM *
202 REM * GET THE ANSWER
  
```

```

204 REM *
210 INPUT Y
220 IF Y=X+1 THEN GOTO 2000:REM * CORRECT ANSWER
230 GOTO 3000:REM * INCORRECT ANSWER
2000 REM *
2002 REM * MESSAGE FOR CORRECT ANSWER
2004 REM *
2010 PRINT :PRINT "GOOD"
2020 PRINT :PRINT
2030 GOTO 110
3000 REM *
3002 REM * MESSAGE FOR INCORRECT ANSWER
3004 REM *
3010 PRINT :PRINT "TRY AGAIN..."
3020 PRINT
3030 GOTO 120

```

This simple program can be dressed up in a number of different ways. A branch can be added so that after each guess the player can decide whether to play again or end the game. Or this game can be modified to use color and large letters and numbers on the screen display so that it will be easier for younger children to read. The next version of this program includes these modifications, using some new BASIC vocabulary. We have already been introduced to GRAPHICS 2 (large text) and PRINT #6 in Chapter Two. DIM A\$, INPUT A\$, GRAPHICS 0, and END will be explained later in this chapter. The two most important new BASIC statements in this new version of the program are GOSUB and RETURN.



GOSUB and RETURN represent a very powerful programming idea, that of a *subroutine*. A subroutine is a small program within a program. For example, in the modification of the simple drill program shown next, we use four subroutines. They are located at lines 2000 through 2099, 3000 through 3099, 4000 through 4060, and 5000 through 5099. The first GOSUB at line 320 goes to the subroutine starting at line 2000. It is used to PRINT the message, "GOOD!" if you get the correct answer. Then, it RETURNS back to line 330 and goes on with the program. Line

330 is also a GOSUB, this time using the subroutine at line 5000. This subroutine causes the computer to pause briefly after PRINTing this message. Line 400 begins a series of GOSUBs which are used if you didn't get the correct answer. First, the subroutine at 3000 is used and then the pause routine at line 5000 is used. Note that you can do a GOSUB from one subroutine to another. Both of the subroutines at lines 2000 and 3000 "call" the subroutine at line 5000 to do a short pause before RETURNing to the main program.

```

10 REM * SIMPLE DRILL FOR LITTLE ONES
12 REM *      COPYRIGHT 1981 BY
14 REM *      HERB KOHL AND TED M. KAHN
16 REM * uses large letters & numbers
20 REM *
50 DIM A$(1)
70 REM * PROBLEMS IN GROUPS OF 10
100 FOR I=1 TO 10
110 REM * CHOOSE NUMBER FROM 0 - 10
120 LET X=INT(RND(1)*11)
200 REM * USE GRAPHICS 2 TEXT
210 GRAPHICS 2
220 PRINT #6;"[CLEAR]WHAT NUMBER COMES"
230 PRINT #6;"RIGHT AFTER ";X;"?"
300 REM * GET THE ANSWER
310 INPUT Y:PRINT #6;Y
315 REM * IF CORRECT, THEN DO ANOTHER
320 IF Y=X+1 THEN GOSUB 2000:GOTO 340
325 REM * OTHERWISE, TRY AGAIN...
330 GOSUB 3000:GOTO 210
340 NEXT I
400 REM * ASK TO PLAY AGAIN?
410 GOSUB 4000
420 GOTO 100
2000 REM *
2010 REM * CORRECT ANSWER MESSAGE
2020 REM *
2030 PRINT #6:REM * PRINT BLANK LINE
2040 PRINT #6;"GOOD!"
2050 GOSUB 5000:REM * PAUSE BRIEFLY
2099 RETURN
3000 REM *
3010 REM * INCORRECT ANSWER MESSAGE
3020 REM *
3030 PRINT #6:REM * PRINT BLANK LINE

```







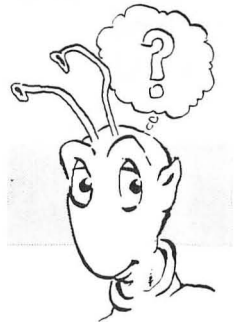
```

3040 PRINT #6;"NO, TRY AGAIN..."
3050 GOSUB 5000:REM * PAUSE BRIEFLY
3099 RETURN
4000 REM *
4010 REM * PLAY AGAIN?
4020 REM *
4030 PRINT "CLEAR:DO YOU WANT TO PLAY A
GAIN":PRINT "(YES OR NO)";
4040 INPUT A$:REM * IF "Y" GO BACK
4050 IF A$="Y" THEN RETURN
4060 GOTO 9999:REM * IF "N" THEN END
5000 REM *
5010 REM * PAUSE OR DELAY ROUTINE
5020 REM *
5030 FOR WAIT=1 TO 600:NEXT WAIT
5099 RETURN
9999 GRAPHICS 0:END :REM * END OF PROGRA
M

```

A couple of quick explanations about some statements in this program might help reduce its seeming complexity. Actually, it's the same basic program as Version 1. Line 320 shows an interesting use of IF-THEN. If you follow the THEN part of this BASIC IF-THEN statement with more than one BASIC command separated by the colon (:), each will be done in order IF the condition is true. Therefore the sequence for line 320 is as follows:

1. Check to see IF  $Y=X+1$
2. IF it is, THEN jump to the subroutine at line 2000 (GOSUB 2000); after RETURNing, GOTO line 340.
3. IF Y is not equal to  $X+1$ , THEN go on with the next statements (line 325 and 330)



Similarly, line 330 first jumps to the subroutine at line 3000 (GOSUB 3000). After a RETURN is reached at line 3099, it comes back and finishes line 330, that is, it then goes to line 210.

If you ever have trouble following the flow between these subroutines and the main program, think of each RETURN statement as a telegraph message which says, "Send me back to where I was called with the last GOSUB." Tracing a program's "flow" in this fashion is a very good way to understand its structure.

The next programs are examples of arithmetic drills and can be used to practice addition, subtraction, multiplication, and division. Depending

upon the size of the numbers you choose, these programs can be used with 5- to 12-year olds. For example, an addition drill with numbers under 5 is appropriate for kindergarten and first grade. By making a slight modification in the program, you can have the computer create addition drills with numbers up to 10, 100, 1000, or any figure you choose. For convenience, all the programs illustrated here will use numbers up to and including 10. You can change the numbers used in the drill by modifying the line as indicated by Ta\*ri or Junior.

In this simple program the computer gives you an addition problem. If you guess correctly it gives you another problem. If not, you get additional chances to get the right answer.



```

10 REM * VERY SIMPLE ADDITION DRILL
20 REM * (C) 1981 BY HERBERT KOHL
30 REM * (without subroutines)
40 REM *
100 PRINT "C[CLEAR]HOW MANY ADDITION PROBLEMS"
105 PRINT "WOULD YOU LIKE";
110 INPUT N
120 FOR I=1 TO N
130 REM * SIZE OF NUMBERS DEPENDS ON
135 REM * VALUES IN LINES 140 & 150
140 LET X=INT(RND(1)*10)+1
150 LET Y=INT(RND(1)*10)+1
160 PRINT "WHAT IS THE ANSWER?"
170 PRINT X;" + ";Y;" = ";
200 INPUT Z
210 IF Z=X+Y THEN GOTO 300
220 PRINT :PRINT "TRY AGAIN...":PRINT
230 GOTO 170
300 PRINT :PRINT "YOU GOT IT!"
310 IF I<N THEN PRINT "HERE'S ANOTHER ONE TO TRY"
320 PRINT
330 NEXT I

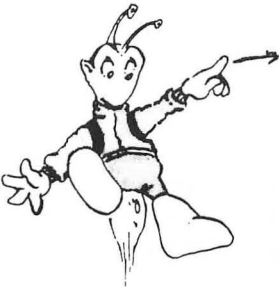
```

By substituting the word multiplication for addition in line 100 and a multiplication sign (\*) for the addition sign (+) in lines 170 and 210, you can create this multiplication drill program:

```

10 REM * SIMPLE MULTIPLICATION DRILL
20 REM * (C) 1981 BY HERBERT KOHL
30 REM * (without subroutines)
40 REM *
100 PRINT "CLEAR HOW MANY MULTIPLICATION
PROBLEMS"
105 PRINT "WOULD YOU LIKE";
110 INPUT N
120 FOR I=1 TO N
130 REM * SIZE OF NUMBERS DEPENDS ON
135 REM * VALUES IN LINES 140 & 150
140 LET X=INT(RND(1)*10)+1
150 LET Y=INT(RND(1)*10)+1
160 PRINT "WHAT IS THE ANSWER?"
170 PRINT X;" * ";Y;" = ";
200 INPUT Z
210 IF Z=X*Y THEN GOTO 300
220 PRINT :PRINT "TRY AGAIN...":PRINT
230 GOTO 170
300 PRINT :PRINT "YOU GOT IT!"
310 IF I<N THEN PRINT "HERE'S ANOTHER ON
E TO TRY"
320 PRINT
330 NEXT I

```



Subtraction drill requires a modification of the drill program if positive integer answers are required. The reason is that if the computer chooses 8 as X and 9 as Y in the above addition program, the subtraction version would yield "8 - 9 = ?" as a problem, and this has no positive solution. Modifying the program is not a difficult task, however. All we have to do is be sure that X is bigger than Y before we tell the computer to print out the problem. In the next example we added a branch at line 155 to take care of this. The IF-THEN statement says that IF X is smaller than Y, THEN choose another pair of numbers; IF not, print out the subtraction problem.

```

10 REM * SIMPLE SUBTRACTION DRILL
20 REM * (C) 1981 BY HERBERT KOHL
30 REM * (without subroutines)
40 REM *
100 PRINT "CLEAR HOW MANY SUBTRACTION P
ROBLEMS"
105 PRINT "WOULD YOU LIKE";
110 INPUT N

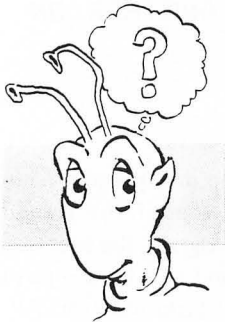
```



```

120 FOR I=1 TO N
130 REM * SIZE OF NUMBERS DEPENDS ON
135 REM * VALUES IN LINES 140 & 150
140 LET X=INT(RND(1)*10)+1
150 LET Y=INT(RND(1)*10)+1
155 IF X<Y THEN GOTO 140
160 PRINT "WHAT IS THE ANSWER?"
170 PRINT X;" - ";Y;" = ";
200 INPUT Z
210 IF Z=X-Y THEN GOTO 300
220 PRINT :PRINT "TRY AGAIN...";PRINT
230 GOTO 170
300 PRINT :PRINT "YOU GOT IT!"
310 IF I<N THEN PRINT "HERE'S ANOTHER ON
E TO TRY"
320 PRINT
330 NEXT I

```



Division drills also require modification if we are to avoid fractions in the answer or division by 0. Division by zero is not a problem if we use only the numbers from 1 to 10 in our programs. If you have forgotten how this works refer back to Chapter One and the section on "The Elusive and Powerful RND(1)."

To get whole-number answers for division problems, the easiest strategy to adopt is to have the computer select two whole numbers: X and Y. Multiply them together:  $X * Y = W$ . Use the answer W to set up the problem:  $W / Y = ?$ ; INPUT another number, Z; then if  $Z = X$  the division program is correct.

This strategy is accomplished in lines 140 through 155 and 170 through 210 of the following division-drill program:

```

10 REM *   SIMPLE DIVISION DRILL
20 REM * (C) 1981 BY HERBERT KOHL
30 REM *   (without subroutines)
40 REM *
100 PRINT "[CLEAR]HOW MANY DIVISION PROB
LEMS"
105 PRINT "WOULD YOU LIKE";
110 INPUT N
120 FOR I=1 TO N
130 REM * SIZE OF NUMBERS DEPENDS ON
135 REM * VALUES IN LINES 140 & 150
140 LET X=INT(RND(1)*10)+1
150 LET Y=INT(RND(1)*10)+1

```





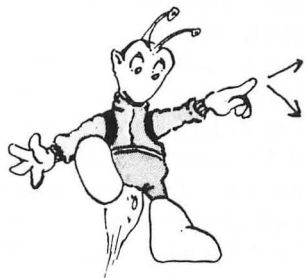
```

155 LET W=X*Y
160 PRINT "WHAT IS THE ANSWER?"
170 PRINT W;" / ";Y;" = ";
200 INPUT Z
210 IF Z=W/Y THEN GOTO 300
220 PRINT :PRINT "TRY AGAIN...":PRINT
230 GOTO 170
300 PRINT :PRINT "YOU GOT IT!"
310 IF I<N THEN PRINT "HERE'S ANOTHER ON
E TO TRY"
320 PRINT
330 NEXT I

```

Now you can write addition, subtraction, multiplication, and division drill programs for any numbers you choose. In order to make these basic programs more interesting, they can be dressed up with sound, color, and graphics. If you cared to, you could even have your ATARI Home Computer play the *Star Spangled Banner*, explode firecrackers, and display a rainbow every time someone gets 10 or 20 (or any number you choose) of the problems correct in the series programmed.

Here is a modest dressing of the simple addition-drill program. It uses the new BASIC word DIM at line 50 and a new type of variable shown by NAME\$ in lines 50, 70, and 320. These enable the computer to handle letters as well as numbers and will be explained fully later in this chapter. You can use them in this program now and understand some things about *how* they are used.



```

10 REM * VERY SIMPLE ADDITION DRILL
20 REM *   WITH A LITTLE DRESSING
30 REM * (C) 1981 BY HERBERT KOHL
40 REM *
50 DIM NAME$(20)
60 PRINT "[CLEAR]WHAT IS YOUR NAME";
70 INPUT NAME$
100 PRINT "[CLEAR]HOW MANY ADDITION PROBLEMS"
105 PRINT "WOULD YOU LIKE, ";NAME$;
110 INPUT N
120 FOR I=1 TO N
130 REM * SIZE OF NUMBERS DEPENDS ON
135 REM * VALUES IN LINES 140 & 150
140 LET X=INT(RND(1)*10)+1
150 LET Y=INT(RND(1)*10)+1

```

```

160 PRINT "WHAT IS THE ANSWER?"
170 PRINT X;" + ";"Y;" = ";"
200 INPUT Z
210 IF Z=X+Y THEN GOTO 300
220 PRINT :PRINT "TRY AGAIN...":PRINT
230 GOTO 170
300 REM * CONGRATULATIONS
310 FOR T=1 TO 10
→ 320 PRINT "GREAT JOB, ";"NAME$;"! ";"
330 NEXT T
390 NEXT I

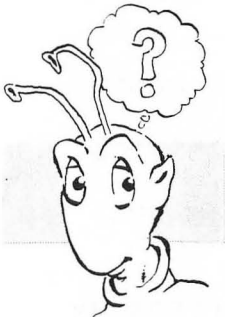
```

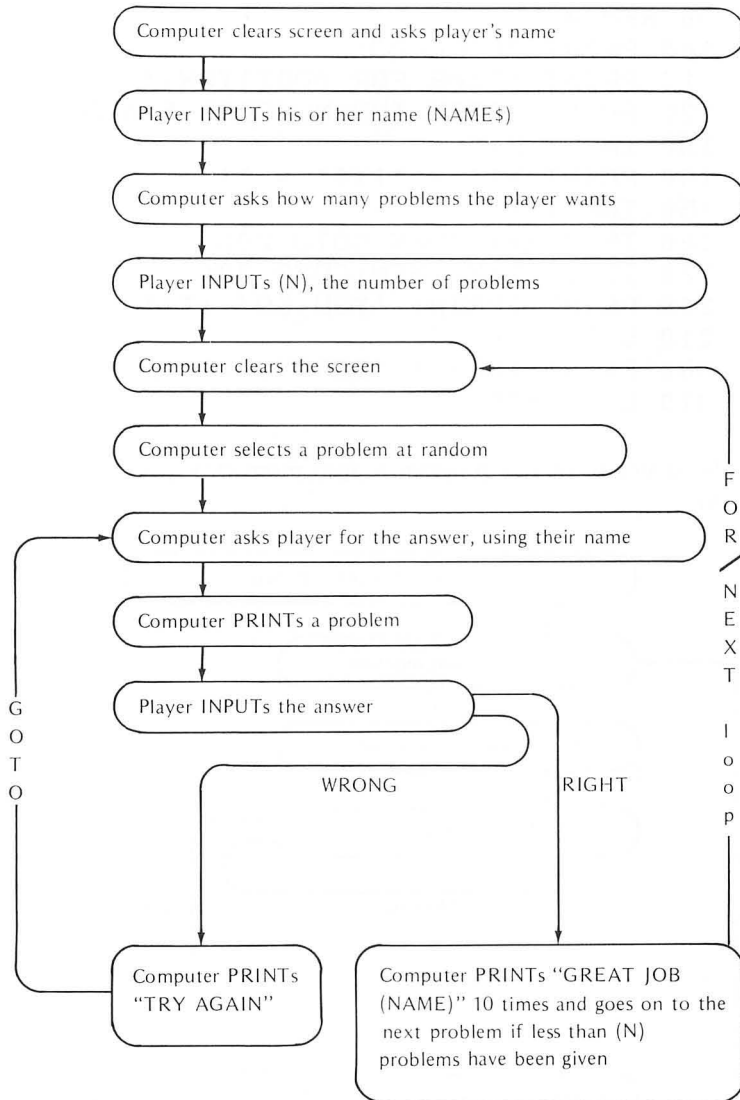


This program asks the player for his or her name and then asks how many problems the player wants, using the player's name. If the answer to a problem is correct, the computer then PRINTs "GREAT JOB, NAME\$" (where NAME\$ is replaced by the actual name of the player) 10 times. The player is then asked to try another problem, up to the total number of problems he or she requested at the beginning of the program. The program is nearly identical to the simple addition-drill program we presented before, except that it is *personalized* by referring to the player by name and we have added a little dressing to congratulate the player after getting the correct answer on lines 300 through 330.

Take a look at the structure of this program and the dressings that are used. Line 60 asks the player for his or her name, and lines 50 and 70 enable the computer to store that name in its memory in the variable NAME\$. Every time you ask the computer to PRINT NAME\$, you'll get the name stored in NAME\$. Line 60 and 125 clear the screen, and lines 100 and 160 personalize messages for the game. Finally, lines 300 through 330 tell the machine to PRINT out the player's name next to the words "GREAT JOB" 10 times, and line 390 ends a FOR-NEXT loop which instructs the computer to present another problem.

If you were to do a diagram or "flow chart" of this program, it might look like this:





Here is another drill program for addition in which the player rather than the computer decides upon the numbers to be used. Do you see the line in the program where this happens?

```

10 REM *   ADDITION OF TWO NUMBERS
20 REM *   (YOU PICK THEM)

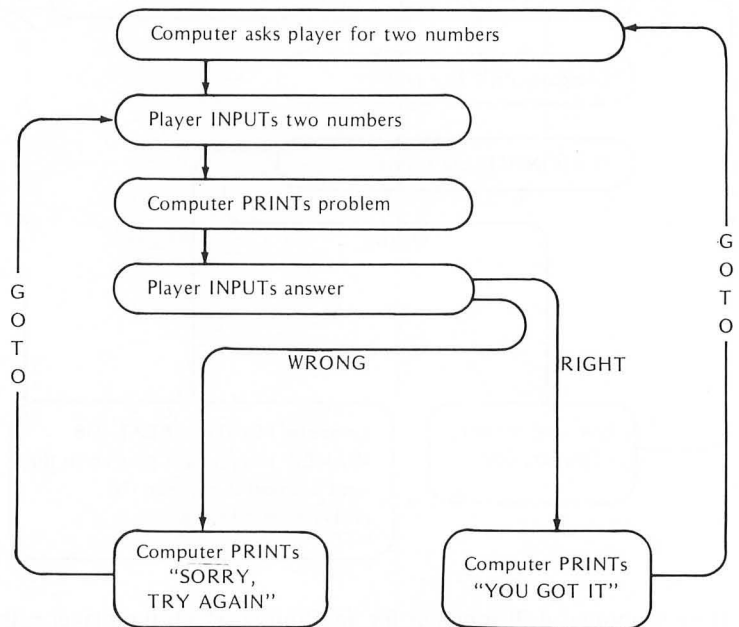
```

```

30 REM *
100 PRINT "CLEAR"
110 PRINT "TIME FOR ADDITION."
120 PRINT "PICK TWO NUMBERS & ADD THEM"
130 INPUT X,Y
140 PRINT X;" + ";Y;" = ";
150 INPUT Z
160 IF Z=X+Y THEN GOTO 200
170 IF Z<>X+Y THEN GOTO 300
200 PRINT :PRINT "YOU GOT IT!"
210 GOTO 120
300 PRINT :PRINT "SORRY, TRY AGAIN"
310 GOTO 140

```

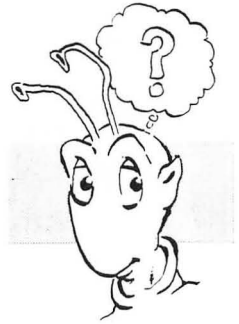
The structure of this program is seen more clearly in the following diagram:



This program can be modified for multiplication, subtraction, or division. However, if you try to do it for subtraction and division and want the answers to be positive integers, be careful about the instructions you have the computer PRINT.



The next arithmetic drill program does the same thing as flash cards. The player picks the times table he or she wants to drill, and the computer comes up with the problems and checks the answers. This same program can be modified to provide problems for addition, subtraction, or division by a particular number. As before, division provides a particularly interesting problem if we want to limit ourselves to integer results. The solution to this problem is reviewed again following the LISTing of this program.



```

10 REM *    SIMPLE VERSION OF TIMES
20 REM *          TABLE DRILL
30 REM *    (C) 1981  BY HERBERT KOHL
40 REM *
50 DIM NAME$(20)
60 PRINT "CLEAR]WHAT IS YOUR NAME";
E$;
70 INPUT NAME$
80 PRINT "CLEAR]WHICH NUMBER FROM YOUR
TIMES TABLE"
90 PRINT "DO YOU WANT TO PRACTICE, ";NAM
E$;
100 INPUT N
120 REM * SIZE OF NUMBERS DEPENDS ON
125 REM * VALUE OF X IN LINE 130
130 LET X=INT(RND(1)*10)+1
140 PRINT "WHAT IS THE ANSWER, ";NAME$
150 PRINT X;" * ";N;" = ";
160 INPUT Z
170 IF Z=X*N THEN GOTO 200
180 PRINT :PRINT "TRY AGAIN...":PRINT
190 GOTO 150
200 PRINT :PRINT "YOU GOT IT!"
210 PRINT "HERE'S ANOTHER ONE TO TRY"
220 PRINT
230 GOTO 130

```

It is somewhat more challenging to provide a drill program for practice in division. One reason is that division by 0 has no meaning in ordinary arithmetic. Second, the ATARI Home Computer is designed to carry out the answers to mathematical calculations to more decimal places than most children or adults need or are interested in knowing.

A program has already been LISTed which provides simple drill in division. An interesting challenge is to modify this program so that it can



handle any division problem, check to be sure that no division by zero passes through, and allows you to pick your own values for the divisor. You already have all the BASIC vocabulary to do this!

Your ATARI Home Computer can do complex mathematics, and for that reason it can be used to set up practice programs for learning algebra. The computer can set up equations, solve them, question you about solutions, and correct your answers.

Here is a program with simple addition equations that yields answers that are positive integers. This program might be a good way to introduce people to algebra.

```

10 REM * SOME ALGEBRA WITH POSITIVE
20 REM *      INTEGER SOLUTIONS
30 REM * equations in one variable
40 REM * (C) 1981 BY HERBERT KOHL
50 PRINT "[CLEAR]"
100 LET X=INT(RND(1)*11)
110 LET Y=INT(RND(1)*11)
120 IF X>Y THEN GOTO 100
130 PRINT "HERE IS A SIMPLE ALGEBRA PROBLEM."
140 PRINT
150 PRINT "WHAT IS A CORRECT VALUE FOR W"
160 PRINT X;" + W = ";Y
170 PRINT "W="";INPUT W
180 IF W=Y-X THEN GOSUB 200;GOTO 100
190 PRINT "SORRY... TRY AGAIN";GOTO 150
200 REM *
210 REM * GOOD ANSWER
220 REM *
230 PRINT :PRINT "WELL DONE! NOW TRY THIS ONE"
240 RETURN

```

The next program uses equations of the form:  $W=AX+AY$ .

```

10 REM * SETTING UP ANOTHER SIMPLE
20 REM *      DRILL IN ALGEBRA
30 REM * (C) 1981 BY HERBERT KOHL
40 REM *

```

```

100 LET X=INT(RND(1)*21)+1
110 LET Y=INT(RND(1)*21)+1
120 PRINT "[CLEAR]"
130 PRINT "HERE IS A SIMPLE EQUATION."
135 PRINT "PICK AN A AND SOLVE IT FOR W"
140 PRINT "(A*";X;") + (A*";Y;") = W"
150 PRINT "CHOOSE AN A";
160 INPUT A
170 LET W=A*X+A*Y
180 PRINT :PRINT "WHAT IS YOUR SOLUTION ";
190 INPUT N
200 IF N=W THEN GOTO 300
210 PRINT "TRY AGAIN...":GOTO 190
300 PRINT :PRINT "YOU GOT IT!"
999 END

```



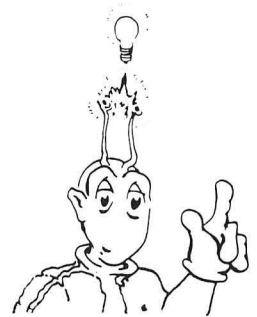
Notice the word END at line 999. This command does just what it says. It formally ENDS the program. In this case, after you get the right answer the computer does not provide a new equation. To begin the program again, you have to type RUN and press the RETURN key. There are times when you want the computer to RUN a program and then stop. That's the role of the END command.

Here is another program that provides practice with some algebraic concepts and provides exercise in "inductive thinking skills." Drill and practice need not be limited to basic arithmetic facts. Learning to think is vital to learning to solve more complex problems, and "thinking skills" need exercise too!

The goal of this game is to determine the number which comes next in a series. It is recommended for students in the upper elementary and junior-high grades as well as for older students or adults. It involves more BASIC vocabulary than you know at present. However, you can type it into your computer, RUN it, and experiment in making changes.

When you RUN the program, you will notice that it first asks for a "rule-maker" to INPUT three numbers. This is so that the game can be played by two people: The rule-maker INPUTs their numbers and then the other player tries to guess the rule by looking at each new number in the series. The rule-maker could be a parent, teacher, or friend.

The game is especially good practice for algebra since the values used here are all of the same form (linear function of the form  $Y=A*X+B$ ). Try this game out, and see if you can improve on it!



```

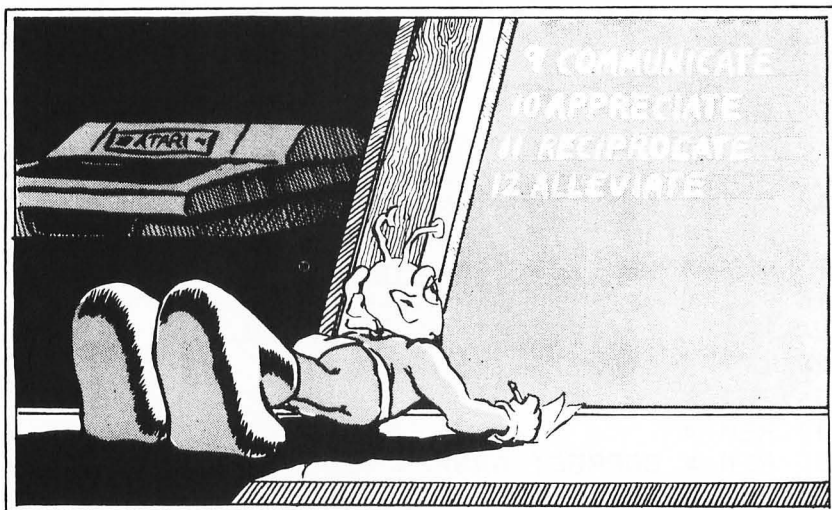
10 REM *      <<< WHAT NEXT >>>
20 REM *   (C) 1981  by Ted M. Kahn
30 REM *   -A game of practice in
40 REM *   inductive thinking skill
50 GOSUB 300
60 GRAPHICS 0
70 DIM A$(1),GUESS$(7)
80 REM * MYSTERY NUMBER = A*X+B
90 GRAPHICS 0:PRINT "FOR THE RULE-MAKER
ONLY:":PRINT
100 TRAP 100:PRINT "INPUT A VALUE FOR X
(1-10)";:INPUT X:IF X<1 OR X>10 THEN X=A
BS(INT(X/10))
110 TRAP 110:PRINT :PRINT "INPUT A VALUE
FOR A (0-10)";:INPUT A:IF A<0 OR A>10 TH
EN A=ABS(INT(A/10))
120 TRAP 120:PRINT :PRINT "INPUT A VALUE
FOR B (0-10)";:INPUT B:IF B<0 OR B>10 T
HEN B=ABS(INT(B/10))
130 TRAP 130:PRINT "HOW MANY GUESSES ALL
OWED";:INPUT GUESS
140 LET GUESS=GUESS+1:LET GUESS$="GUESSE
S"
150 GRAPHICS 2:REM * BIG LETTERS
160 COLOR 1
170 LET GUESS=GUESS-1:IF GUESS<1 THEN GO
TO 260
180 IF GUESS=1 THEN LET GUESS$="GUESS"
190 PRINT "[CLEAR]WHAT'S THE NEXT NUMBER
IN THIS SERIES?"
200 PRINT "(YOU HAVE ";GUESS$;" ";GUESS$;
" LEFT)";
210 PRINT #6;" ";X;
220 LET X=A*X+B
230 TRAP 230:INPUT Y
240 IF Y=X THEN GOSUB 440:GOSUB 610:GOTO
90
250 GOSUB 550:GOTO 170
260 PRINT "[CLEAR]THE NEXT NUMBER WAS ";
X
270 PRINT "THE SECRET RULE WAS ";A;" * X
+ ";B
280 GOSUB 610
290 END
300 REM *

```

```

310 REM * TITLE PAGE
320 REM *
330 GRAPHICS 1
340 COLOR 1
350 POSITION 5,4
360 PRINT #6;"what next?":PRINT #6
370 PRINT #6;"  MYSTERY NUMBERS"
380 POSITION 4,10
390 PRINT #6;"(C) 1981 BY"
400 PRINT #6;"    ted m. kahn"
410 PRINT "(PRESS START TO BEGIN...)"
420 IF PEEK(53279)<>6 THEN 420
430 RETURN
440 REM *
450 REM * CORRECT ANSWER
460 REM *
470 GOSUB 680
480 FOR I=1 TO 5
490 GRAPHICS 2:COLOR 2
500 POSITION 4,5:PRINT #6;"you got it!"
510 FOR J=50 TO 100:SOUND 1,J,10,10:NEXT
  J
520 SOUND 1,0,0,0
530 NEXT I
540 RETURN
550 REM *
560 REM * INCORRECT ANSWER
570 REM *
580 PRINT :PRINT "NO, THAT'S NOT IT.":PR
INT "HERE'S THE NEXT ONE..."
590 GOSUB 680
600 RETURN
610 REM *
620 REM * WANT ANOTHER GAME?
630 REM *
640 PRINT "DO YOU WANT TO PLAY AGAIN";
650 INPUT A$
660 IF A$="Y" THEN RETURN
670 GOTO 290:REM * END OF GAME
680 REM *
690 REM * PAUSE (WAIT) ROUTINE
700 REM *
710 FOR WAIT=1 TO 300:NEXT WAIT
720 RETURN

```

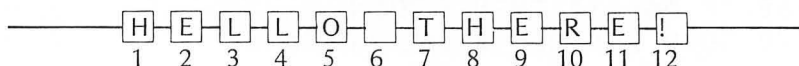


### SPELLING DRILL AND A LITTLE GRAMMAR: USING 'STRING VARIABLES'

There are some new BASIC words that have to be introduced in order to write drill programs or games involving letters and words. The reason for this is that ATARI BASIC handles words differently than it handles numbers. To understand this, you must first understand the concept of *character strings*. You have already used character strings before, but you probably weren't aware of them. Every time you PRINT a message, you are PRINTing a character string. For example:

PRINT "HELLO THERE!"

In this statement, the letters, spaces or blanks, and the punctuation mark (!) are all characters and the entire message is a string of characters. You might think of this as a string of beads, with each bead containing one character. In the above example, the string contains exactly 12 characters, as shown here.



Anything used inside the quotation marks in ATARI BASIC is a character string, sometimes called a *literal* string, since it is used literally just as you see it. Just about anything can be part of a character string, including numbers, special graphics characters (those produced by pressing

down on the [CTRL] key and at the same time typing another key), and cursor controls (up, down, left, and right). However, pressing the four cursor-control keys must be preceded by pressing the [ESC] key if used in this fashion. Each string has a length, measured by the exact number of characters in the string, and each may be PRINTed on the screen. Here are some other examples of character strings:

- "NOUN"—(4 characters; a character string with one word)
- "12 BUCKLE MY SHOE"—(17 characters)
- "this is lower case"—(18 characters)
- "inverse video"—(13 characters; done by pressing the ATARI logo key after the first quote (") mark and before the final quote mark)
- "●▲●"—(four graphics characters obtained using the [CTRL] key with the T and H keys; ● is [T] and ▲ is [H]).

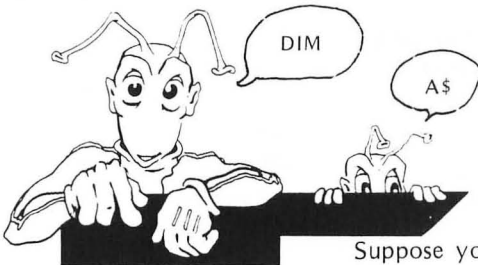
Strings, like numbers, may be used directly or they may be stored in memory locations, labeled by a *variable name*. We have already used many variable names for numbers such as X, Y, or NUMBER. If we want to store character strings in memory using ATARI BASIC, we must do the following:

1. Reserve a place for the string variable by using a statement called DIM (for DIMension). The DIM statement for one or more string variables consists of the word DIM, followed by the string-variable name which always ends with a dollar sign (\$), and the maximum number of characters to be stored in the variable.

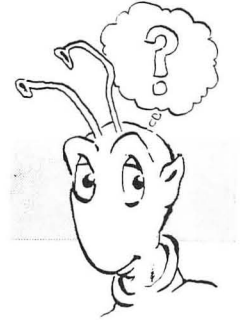
2. In order to give the variable a value, use the LET or INPUT statements.


3. Once the string variable has a value (whatever character string of words, letters, symbols, etc., you put into it), you can PRINT its value, just as you can with numeric variable like X or Y.

Does all this sound confusing to you? Don't worry! With a little patience and practice with some simple examples, you'll get the idea quickly.



Suppose you want to reserve space in the memory of your ATARI Home Computer for a word that has 10 or fewer letters. You would begin by giving a name to a string variable that will contain your word. The



name will end in a dollar sign (\$) to signify that the variable will always be used for character strings. This name could be A\$, B\$, NAME\$, or W10\$. In fact, in ATARI BASIC the variable name can begin with any letter, followed by a combination of up to 254 letters or numbers! Never use a blank space as part of the variable name, although it is fine to use spaces as characters in the string itself. We recommend that you choose names that are meaningful to you, as long as they don't include embedded ATARI BASIC words. 

Let's make life easy and choose the name A\$. In order to tell the computer to reserve space for the word or characters in A\$, we use the DIM statement. Following the words DIM A\$, we put the maximum number of characters to be stored in A\$, enclosing this value in parentheses. So the whole thing looks like this:

```
10 DIM A$(50)
```

This tells your ATARI Home Computer that the variable called A\$ will be able to hold letters or strings of up to 50 characters.

Now that you have saved the space, you can INPUT any word or other string of up to 50 characters in length and store it in memory using the INPUT statement. Add this line to your program:

```
20 INPUT A$
```

If you want to have what you INPUT displayed on the screen, add this next line:

```
30 PRINT A$
```

Now, let's add a message or prompt to the program which tells the player what to do. Add this line, using your own words between the quotes if you wish:

```
15 PRINT "WHAT'S YOUR WORD";
```

LIST the whole program. What do you think it will do? RUN it and see! What happens when you INPUT a word or sentence having more than 50 characters? Experiment with the amount of space you reserved in the DIM statement by changing the number in parentheses in line 10.

As you probably discovered, this program will cut off any characters at the end of your INPUT that are beyond the number of characters



Using words like NEXT\$, LET\$, RUN\$, GOTO\$, etc., may cause problems, since they may be confused with the BASIC commands.



DIMensioned for A\$. This can be quite useful, however. Make the following changes in your program and RUN it again:

```
10 DIM A$(1)
```

and add this line:

```
40 GOTO 15
```

Try several different INPUTs (for example, YES, NO, ADDITION, etc.). What actually gets stored in A\$? This small program allows us to check for simple YES-NO answers. Because A\$ will contain only the first letter of the INPUT, we can immediately tell which word was entered. A\$ will contain Y if YES was INPUT and N if NO was INPUT. You could also use it to ask the player to select from a list or menu of options, such as the type of arithmetic problems to be practiced: A for ADDITION, S for SUBTRACTION, etc. An example of this type of program is given at the end of this chapter.

We already used DIM and INPUT with NAME\$ earlier in the chapter to personalize a program. Look at this simple adaptation of an earlier drill-and-practice program. It should be clear how NAME\$ is used:

```
10 DIM NAME$(20)
20 PRINT "WHAT IS YOUR NAME";
30 INPUT NAME$
40 PRINT "HI, ";NAME$
45 PRINT "HERE'S AN ADDITION PROBLEM FOR
   YOU TO TRY"
50 PRINT "2+2=";
60 INPUT X
70 IF X=4 THEN PRINT "GOOD JOB ";NAME$
80 IF X<>4 THEN PRINT "BETTER LUCK NEXT
   TIME, ";NAME$
```

Notice that there are no quotation marks used around NAME\$ here. If you surrounded NAME\$ with quotes in lines 40, 70, or 80 the computer would literally PRINT the characters "NAME\$" rather than the name stored in the variable labeled NAME\$.

Before introducing some drills and games using letters and words, we need to introduce one other aspect of string variables. As we said earlier, strings are composed of individual characters. Each string stored in a string variable has a length and a certain number of individual characters at any given time. We can find out what any individual character contains by making use of an ATARI BASIC string function, A\$(I,J). For example,



here is a very simple program for young children to practice identifying words that begin with different letters of the alphabet. The computer chooses a letter at random and asks the child to input a word that begins with that letter. The computer then informs the child whether he has chosen a word that fulfills the original condition.



```

10 REM * LETTER IDENTIFICATION GAME
20 REM *      FOR YOUNG CHILDREN
30 REM * (C) 1981 BY FRONA STAR &
40 REM *      TED M. KAHN
50 DIM A$(1),ALPHABET$(26),X$(1),WORD$(2
0)
60 LET ALPHABET$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
70 GRAPHICS 2
80 PRINT "[CLEAR]LET'S PLAY A LETTER GAM
E."
85 PRINT "I'LL TYPE A LETTER AND YOU TYP
E A ";PRINT "WORD THAT BEGINS WITH MY LE
TTER";GOSUB 500
90 PRINT "[CLEAR]"
95 PRINT #6;"[CLEAR]"
100 LET L=INT(RND(1)*26)+1
110 A$=ALPHABET$(L,L)
120 POSITION 9,4
130 PRINT #6;A$
140 INPUT WORD$;LET X$=WORD$(1,1)
150 IF X$=A$ THEN GOSUB 200;GOTO 90
160 PRINT "NO, ";WORD$;" DOESN'T BEGIN W
ITH ";A$;". ";PRINT "TRY ANOTHER WORD"
170 GOTO 140
200 REM * REWARD FOR CORRECT WORD
210 PRINT #6;"[CLEAR]"
220 POSITION 1,4
230 PRINT #6;A$;" IS FOR ";WORD$
240 GOSUB 500
250 RETURN
500 REM * PAUSE/DELAY
510 FOR WAIT=1 TO 1000:NEXT WAIT
520 RETURN

```

This program uses several character strings, all of which are DIM-ensioned in line 50. The string ALPHABET\$ contains the 26 letters of the alphabet in order (line 60). WORD\$ is used to store the word a child

INPUTs (line 140). At line 100, a random number from 1-26 is selected, and then the letter at this position in the alphabet is used as a stimulus for the child's response (line 110). Let's examine this line and line 140 a little more carefully:

```
110 A$=ALPHABET$(L, L)
    (This is short for LET A$=ALPHABET$(L, L)
```

This line says to take the value of L and use this to find the "substring" of ALPHABET\$ beginning with character number L and ending with character number L. Since this substring begins and ends with the same character, it will choose the Lth character in the alphabet. For example, if the value of L was chosen to be 5, the 5th letter or "E" would be chosen from ALPHABET\$ and assigned to A\$. Similarly,

```
140 INPUT WORDS$: X$=WORDS$(1, 1)
```

says to INPUT a word and assign the first character (letter) of the word to the variable X\$.

This is a very simple version of this game, although it has a nice reward of actually typing a phrase based on the word the child chooses if it begins with the letter chosen, all in big (GRAPHICS 2) text. It does not check for correct spelling, however. A drill for spelling practice is coming up next.

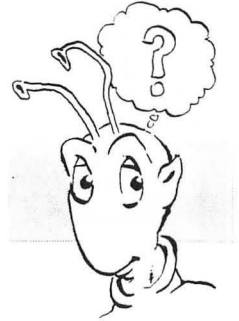
## A Spelling Drill Program

Here is a program that can be used to drill a list of five words from a weekly spelling list. The words could even be in foreign languages, such as French, Spanish, or Latin, except that accents and capitalized letters can't be used in this version. While the program is limited to only five words at a time, it is worthwhile to store on tape or diskette, since it can be used each week with a new list (INPUT by the student).

For the most part, the program uses only ATARI BASIC vocabulary words with which you are already familiar. One new BASIC statement, however, may be found at line 125, the ON-GOSUB statement:

```
125 ON W GOSUB 200, 300, 400, 500, 199
```

This powerful statement allows us to specify a number of different sub-routine calls, depending upon the value of the named variable after the



In this program, words are limited to a maximum of 10 letters each. You may increase this maximum by changing the DIM statements in lines 20-27.



BASIC word ON (in this case, it's W). This one statement replaces the following five BASIC statements:

```
125 IF W=1 THEN GOSUB 200
126 IF W=2 THEN GOSUB 300
127 IF W=3 THEN GOSUB 400
128 IF W=4 THEN GOSUB 500
129 IF W=5 THEN GOSUB 199
```

The statement always assumes integer values for the variable (e.g., W) and the line numbers for the GOSUBs must be specified exactly to correspond with the values 1,2,3,4 . . . for W (in order). Note that in this program the subroutines at lines 200-500 are each only one line long.

```
10 REM * SIMPLE SPELLING DRILL PROGRAM
15 REM * (C) 1980 BY HERBERT KOHL
20 DIM A$(10),B$(10),C$(10),D$(10)
25 DIM E$(10),M$(10)
30 PRINT "CLEAR"
35 PRINT "WRITE DOWN 5 OF YOUR ";
40 PRINT "SPELLING WORDS":PRINT "(ONE PE
R LINE, FOLLOWED BY RETURN)"
45 INPUT A$,B$,C$,D$,E$
50 PRINT "CLEAR"
55 PRINT "ONE OF YOUR SPELLING WORDS WIL
L BE"
60 PRINT "FLASHED ON THE SCREEN BRIEFLY.
":PRINT "STUDY IT."
65 GOSUB 700
70 GRAPHICS 2
75 FOR W=1 TO 5
80 PRINT #6;"CLEAR"
85 POSITION 6,4
90 PRINT #6;A$
95 GOSUB 700:PRINT #6;"CLEAR"
100 PRINT "NOW TRY TO SPELL THE WORD"
105 INPUT M$
110 IF M$<>A$ THEN GOSUB 600:GOTO 105
115 POSITION 4,4:PRINT #6;"WAY TO GO!"
120 GOSUB 700
125 ON W GOSUB 200,300,400,500,199
130 NEXT W
199 END
200 A$=B$:RETURN
```

```

300 A$=C$:RETURN
400 A$=D$:RETURN
500 A$=E$:RETURN
600 REM * INCORRECT SPELLING
610 PRINT "NOT QUITE... TRY AGAIN"
620 RETURN
700 REM * PAUSE ROUTINE
710 FOR WAIT=1 TO 1000:NEXT WAIT
720 RETURN

```

A second version of this same program is somewhat more flexible and makes use of two more new BASIC words: READ and DATA. These words, like FOR and NEXT and GOSUB and RETURN, are used in pairs: READ A\$ performs the same function as INPUT A\$, except that the contents of A\$ is READ from the first available DATA statement. A DATA statement can appear anywhere in the program and contains the string (or numerical data, for numeric variables) data to be read into the A\$ variable in the READ statement. Each time a READ statement is executed, the next item in the DATA "list" is READ. This may include going on to a new DATA statement (if available) in order to find the next item.

In the next program, the spelling DATA is inserted at line 6000 and beyond. Five French words, representing possible spelling or vocabulary items, are currently located in line 6000. These can be replaced by any five words having 10 letters or less. If you want to increase the number of words to be drilled, change line 100 (the beginning of the FOR-NEXT loop) so that W goes from 1 to the total number of words in your DATA list; also, add as many words as necessary in succeeding DATA statements, after line 6000. Remember to separate each word from the ones before or after it with a comma.

Again, this program may be stored away for future use any time you need to practice the spelling of a list of words.



```

10 REM *  << SPELLING DRILL #2 >>
20 REM *      copyright 1980 by
30 REM *  HERBERT KOHL & TED KAHN
40 DIM A$(10),X$(10)
50 PRINT "CLEAR EACH WORD WILL BE "
60 PRINT "BRIEFLY FLASHED ON THE SCREEN"
70 PRINT "HERE THEY ARE..."
90 GOSUB 500
100 GRAPHICS 2
110 FOR W=1 TO 5

```

```

120 READ A$
130 PRINT "NOW STUDY THE NEXT WORD"
140 PRINT #6;"[CLEAR]"
150 POSITION 0,4:PRINT #6,A$
160 GOSUB 500
170 PRINT #6;"[CLEAR]"
180 PRINT "[CLEAR]NOW SPELL THE WORD"
190 INPUT X$
200 IF X$=A$ THEN GOSUB 300:GOTO 220
210 IF X$<>A$ THEN GOSUB 400:GOTO 140
220 NEXT W
230 END
300 REM * CORRECT SPELLING
310 PRINT "YOU GOT IT!  GOOD GOING!"
320 GOSUB 500
330 RETURN
400 REM * INCORRECT SPELLING
410 PRINT "NOT QUITE... TRY AGAIN."
420 GOSUB 500
430 RETURN
500 REM * PAUSE/DELAY
510 FOR WAIT=1 TO 1000:NEXT WAIT
520 RETURN
599 REM * DATA IS FROM LINE 600 ON-
600 DATA MANGER,PORTER,VENIR,FAIRE,TOUCH
   ER

```



### Review:

**Subroutine**—A subroutine is a program within a program. It could be a dressing, an ending, an instruction to a player to try again. Subroutines are convenient ways of grouping separable parts of programs. For example, this is a simple pause subroutine:

```

1000 REM PAUSE ROUTINE
1110 FOR WAIT=1 TO 1000:NEXT WAIT
1199 RETURN

```

### Concepts

### New BASIC Vocabulary

**GOSUB and RETURN**—GOSUB is used to tell the computer to “jump” to a subroutine which begins at the line number specified, and RETURN ends the subroutine, sending the program back to the place where it

left off. The power of subroutines is that they can often be used in many different programs to accomplish the same function. Here is a use of the pause subroutine:

```
10 PRINT "GUESS A NUMBER FROM 1 TO 10"
20 INPUT X
30 GOSUB 1000
40 IF X=7 THEN PRINT "YOU GOT IT"
:   etc.
:
1000 FOR Z=1 TO 1,000
1010 NEXT Z
1020 RETURN
```

Of course this program is incomplete, but it illustrates the use of subroutines. At line 30 the program jumps to line 1000, which creates a pause in the game. After the pause, line 1020 instructs the computer to return to line 40 and continue with the rest of the program.

**ON W GOSUB**—The ON-GOSUB statement makes it possible to have the computer go through a number of different subroutines according to the value of a variable W. The value of W must be specified clearly using for example a simple FOR-NEXT loop like this:

```
100 FOR W=1 TO 3
110 PRINT "HERE'S SOMETHING INTERESTING TO READ"
120 ON W GOSUB 200, 300, 400
130 FOR Y=1 TO 250:NEXT Y
140 NEXT W
150 END
200 PRINT "IT WAS THE BEST OF TIMES"
210 PRINT "IT WAS THE WORST OF TIMES"
220 RETURN
300 PRINT "E=(M*C*C)"
310 RETURN
400 PRINT "WHAT IS BLACK AND WHITE AND PURPLE ALL
      OVER"
410 RETURN
```

**END**—This statement means just what it says, END the program.

**DIM A\$(X)**—The DIM statement reserves a specified number of places in the computer's memory for the variable A\$. These places can hold letters and special characters as well as numbers. DIM A\$(10) reserves 10 places for A\$. An INPUT statement is used to store something in A\$, and you can PRINT what is in A\$ whenever you want. Here's an example:

```

10 DIM A$(25)
20 PRINT "WHAT IS YOUR NAME?"
30 INPUT A$
40 PRINT "HELLO";A$

```

**A\$(X, Y)**—**A\$(X, Y)** lets you get a particular letter or group of letters out of a DIMensioned string. If A\$="ABCDE" then

A\$(1, 3)=A, B, C the first three letters, and

A\$(4, 5)=D, E the fourth and fifth letters

A\$(1, 1)=A and, in general, the first letter

A\$(X, X) will give you whatever is at the X<sup>th</sup> place of A\$.

**READ-DATA**—**READ-DATA** allows you to put multiple data into a variable. For instance, here's a **READ-DATA** program.

```

10 FOR X=1 TO 3
20 READ X
30 PRINT X+40
40 NEXT X
50 DATA 17, 18, 19

```

This program says that for X=1 to 3, read the **DATA** at line 50, add 40 to each X and print the answer. The result of this simple program would be:

```

57
58
59

```

A program such as the following would allow you to put in 100 different numbers at line 50 and have the computer square each number and print out the answer:

```

10 FOR X=1 TO 100
20 READ X
30 PRINT X*X
40 NEXT X
50 DATA

```

### **BASIC Vocabulary Introduced in Part I**

A\$	IF ... THEN ...	RND(1)
A\$(X, Y)	INPUT A\$	RUN
	INPUT X	SETCOLOR
	INT(X)	



CLOAD	INT(RND(1))	SETCOLOR X, Y, Z
COLOR X	INT(RND(1)*10)+1	SOUND W, X, Y, Z
CSAVE	LET X=	
CONT	LIST	X<Y
	NEW	X<=Y
		X>Y
	ON W GOSUB	X>=Y
DIM A\$(X)	PLOT X, Y	X<>Y
DRAWTO X, Y	POSITION X, Y	X+Y
END	PRINT	X-Y
FOR-NEXT	PRINT #6	X*Y
FOR-STEP-NEXT		X/Y
GOSUB-RETURN	READ-DATA	X\Y
GOTO	READY	< >
GRAPHICS 2		
GRAPHICS 7	REM	

#### Delimiters

“ ”  
;  
( )  
:  
,

#### Concepts

Branching	Loop
Delay Loops	Nested Loops
Error Messages	Program
Graphic Modes	Subroutine
Line Number	Text Window
Logical Line	

#### Special Keys

[BACKSPACE]	[INSERT]	[SYSTEM RESET]
[BREAK]	[RETURN]	[CTRL][INSERT]
[CTRL]	[SHIFT]	[CTRL][DELETE]
[DELETE]	[SHIFT][CLEAR] or	[CTRL][↑] (up)
[ESC]	[CTRL][CLEAR]	[CTRL][↓] (down)
[ESC][CLEAR]	[SHIFT][DELETE]	[CTRL][←] (left)
[ESC][CTRL]	[SHIFT][INSERT]	[CTRL][→] (right)



## Part II

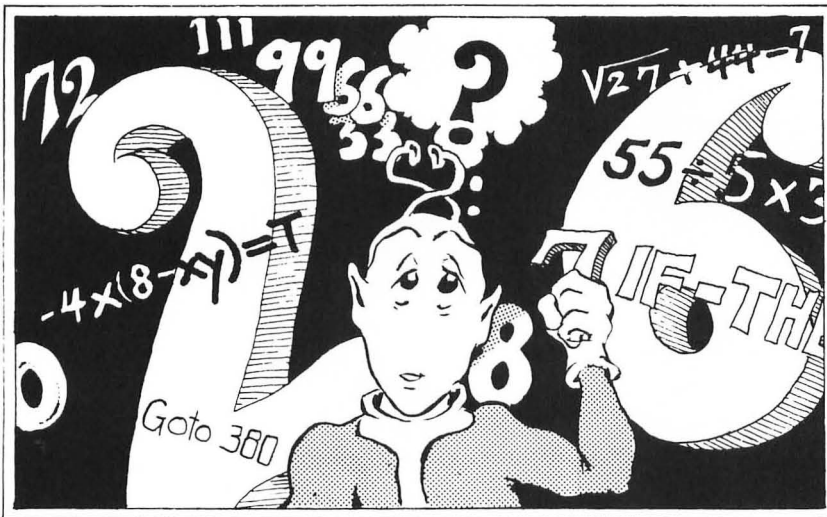
# Games for the ATARI Home Computer

In the first part of this book we presented an introduction to ATARI BASIC using game programs. In this section we will provide many games you can copy and dress up using your ATARI 400 or 800 Home Computer. Some of the easier game programs should be easy for you to understand if you have worked through Part I. However, many of the game programs are quite complex and involve a deeper knowledge of programming than we can provide in this book. That is no reason to skip them. Enter the programs in your computer's memory and play the games. In order to avoid excess work, save these programs using your ATARI 410 Program Recorder or ATARI 810 Disk Drive. Also list and change the game programs. You'll be able to recognize subroutines we mentioned before, PRINT statements, loops and branches. Consult your *ATARI BASIC Reference Manual* for unfamiliar BASIC commands. You'll find that your knowledge of programming will grow quickly as you work with your computer.



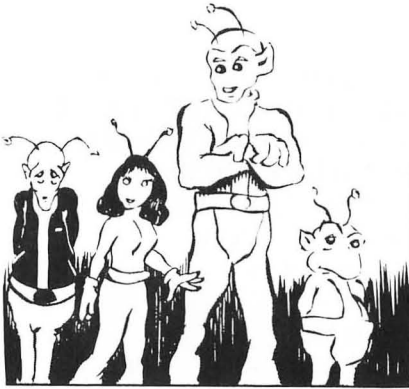
## Chapter 4

# Number and Logic Games



Probably the simplest games to program for a computer involve the symbols with which the computer is most familiar: numbers. We can use numbers in a variety of ways, from just guessing what number the computer (or your friend) is thinking of, to using the computer to design "Magic Squares." Most of these games involve, at some level or other, the use of random numbers and the RND(1) function. By using the computer to generate random numbers, we have the capability of creating an infinite number of games which use elements of chance. The computer can be our dice thrower, our "spinner," our playing opponent (or even our partner), and our record keeper. Furthermore, because we can take advantage of the use of the television screen, the computer can also be our game board or our playing space. And being ecology-minded, this means we don't have to keep using new pieces of paper or cardboard to make up new games: They can all be done on the computer!

Let's begin with some simple guessing games. Remember: These are given as your game "seeds"—they will grow and flower as they are given the proper attention. By dressing them up, you will start to develop a whole game library, which you may store and bring out whenever you want to show your friends how you have learned to use your ATARI Home Computer!



## GUESS THE NUMBER—FOUR VARIATIONS

The computer chooses a number at random, and you try to guess it. In order to limit the infinite possibilities (and therefore, be able to “play”), we must give the computer a range of acceptable numbers. This first program does just that—picks a number between 1 and 10, and you try to guess what it is. It is the same program we presented in Chapter One.

### Version 1

```

10 REM * SIMPLE GUESS-MY-NUMBER**
20 PRINT "I HAVE PICKED A NUMBER"
25 PRINT "BETWEEN 1 AND 10."
30 PRINT "TRY TO GUESS IT."
40 LET X=INT(RND(1)*10)+1
50 PRINT "YOUR GUESS";
60 INPUT Y
70 IF Y=X THEN PRINT "YOU GOT IT!":END
80 PRINT "NO, TRY AGAIN . . .":PRINT
90 GOTO 50

```

This is the simplest possible version. No frills—the computer picks a number between 1 and 10 (line 40), and you try to guess it. It will keep going until you get it.

Here is a simple improvement on the program above: It asks you whether you want to play again (once you get the right answer, that is!). It uses the YES/NO subroutine introduced in Chapter Three.

**Version 2**

```

5 DIM A$(1)
10 REM ** GUESS-MY-NUMBER #2 **
15 REM ** ADD "PLAY AGAIN" FEATURE **
20 PRINT "I HAVE PICKED A NUMBER"
25 PRINT "BETWEEN 1 AND 10."
30 PRINT "TRY TO GUESS IT."
40 LET X=INT(RND(1)*10)+1
50 PRINT "YOUR GUESS";
60 INPUT Y
70 IF Y=X THEN 100
80 PRINT "NO, TRY AGAIN . . .":PRINT
90 GOTO 50
100 PRINT "YOU GOT IT!"
110 PRINT "DO YOU WANT TO PLAY AGAIN?"
120 INPUT A$
130 IF A$="N" THEN END
140 CLR
150 GOTO 20

```

Version 3 adds another improvement—it allows you to see the answer, if you so choose, after each incorrect guess. (See lines 70-90.)

**Version 3**

```

5 DIM A$(1)
10 REM ** GUESS-MY-NUMBER #3 **
15 REM * ADD "PLAY AGAIN" FEATURE & **
17 REM * "[CLEAR] WHAT'S THE ANSWER?" **
20 PRINT "I HAVE PICKED A NUMBER"
25 PRINT "BETWEEN 1 AND 10."
30 PRINT "TRY TO GUESS IT."
40 LET X=INT(RND(1)*10)+1
50 PRINT "YOUR GUESS";
60 INPUT Y
70 IF Y=X THEN 100
80 PRINT "TYPE Y TO TRY AGAIN OR"
85 PRINT "TYPE N TO SEE THE ANSWER";
87 INPUT A$: IF A$="Y" THEN GOTO 50
90 PRINT "MY NUMBER IS";X:GOTO 110
110 PRINT "YOU GOT IT!"
110 PRINT :PRINT "DO YOU WANT TO PLAY AGAIN?"
130 IF A$="Y" THEN CLR:GOTO 20
140 PRINT "BYE FOR NOW"
150 END

```



Version 4 adds a very important improvement: *hints* (lines 140–150). This way, you know whether your guess was too small or too large, and then you can adjust your next guess according to the hint you just received.

This version also allows you to expand the range of the number the computer will choose (see lines 40–50). Up to now, all versions have used the range 1–10; this one will use the range 1 to MAX where MAX is the top of the range that you input. It also keeps track of how many guesses you use and will quit if you don't get it after MAX guesses.

### Version 4

```

5 DIM A$(1)
10 REM ** GUESS-MY-NUMBER #4 **
20 REM ** (WITH HINTS) **
30 PRINT "I WILL PICK A NUMBER BETWEEN"
40 PRINT "1 AND (YOU FILL IN)";
50 INPUT MAX
60 PRINT "THEN, YOU TRY TO GUESS IT."
70 LET X=INT(RND(0)*MAX)+1
80 PRINT :PRINT "OK, I'VE GOT A NUMBER."
100 FOR I=1 TO MAX
110 PRINT :PRINT "YOUR GUESS";
120 INPUT Y
130 IF Y=X THEN GOTO 250
140 IF Y>X THEN PRINT "NO, THAT'S TOO HIGH. . .
    TRY AGAIN": GOTO 200
150 IF Y<X THEN PRINT "NO, THAT'S TOO LOW . . .TRY
    AGAIN"
200 NEXT I
210 PRINT :PRINT "SORRY. MY NUMBER WAS"; X
220 GOTO 290
250 PRINT :PRINT "YOU GOT IT IN"; I;" GUESSES!"
290 POP :PRINT "DO YOU WANT TO PLAY AGAIN (YES OR
    NO)";
295 INPUT A$
300 IF A$="Y" THEN CLR:GOTO 30
310 PRINT "THANKS FOR PLAYING"
399 END

```

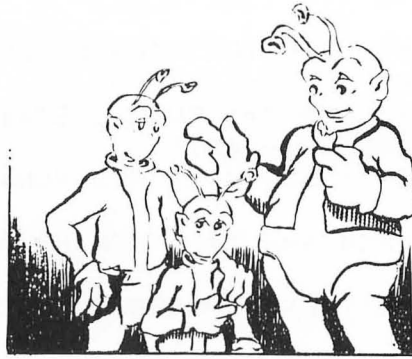
Here's an interesting twist on the typical number-guessing games. In this game, the computer guesses *your* number instead of the other way around. By watching the way the program chooses its guesses, you can learn something about one of the most effective strategies for playing guessing games of this type!

```

10 REM << COMPUTER GUESSES YOUR # >>
20 DIM GAME(100)
30 SWITCH=53279:START=6:G=0
40 POKE SWITCH,0
50 PRINT "[CLEAR]THINK OF A NUMBER BETWE
EN"
60 PRINT "1 AND 100. WHEN YOU'VE GOT"
70 PRINT "IT, PRESS START."
80 REM *WAIT TILL START IS PUSHED *
90 IF PEEK(SWITCH)<>START THEN 90
100 LAST=0
110 PRINT :PRINT "OK, NOW I'LL GUESS & Y
OU TYPE"
120 PRINT " 2 IF I'M TOO HIGH;"
130 PRINT " 1 IF I'M TOO LOW; &"
140 PRINT " 0 IF I'M RIGHT ON."
150 NOW=40+INT(RND(1)*20)+1
160 PRINT
170 FOR I=1 TO 10
180 PRINT "GUESS # ";I;" = ";NOW;" (2,1
, OR 0) ";
190 TRAP 220:INPUT X
200 IF X=0 THEN 260
210 IF X=2 OR X=1 THEN GOSUB 380:GOTO 24
0
220 PRINT "PLEASE JUST TYPE 2,1, OR 0";
230 GOTO 190
240 NEXT I
250 GOSUB 480:GOTO 270
260 PRINT :PRINT "I GOT IT IN ";I;" GUES
SES."
270 PRINT :PRINT "PLAY AGAIN (1=YES, 2=NO) ";
280 INPUT PA
290 G=G+1:GAME(G)=I
300 IF PA=1 THEN 50
310 SUM=0
320 FOR J=1 TO G
330 SUM=SUM+GAME(J)
340 NEXT J
350 PRINT :PRINT "MY AVERAGE FOR ";G;" G
AMES WAS:" :PRINT :PRINT " ";SUM/G;" GU
ESSES."
360 PRINT :PRINT "THANKS FOR PLAYING!"

```

```
370 END
380 REM * GUESS TOO HIGH OR LOW *
390 A=NOW
400 IF I=1 AND X=1 THEN LAST=100
410 DIF=ABS(NOW-LAST)
420 IF DIF<4 AND X=2 THEN NOW=NOW-1:GOTO
  460
430 IF DIF<4 AND X=1 THEN NOW=NOW+1:GOTO
  460
440 IF X=2 THEN NOW=NOW-INT(DIF/2):GOTO
  460
450 IF X=1 THEN NOW=NOW+INT(DIF/2)
460 LAST=A
470 RETURN
480 REM *
490 PRINT :PRINT "ONE OF THE FOLLOWING H
AS OCCURRED:"
500 PRINT "  1. MY STRATEGY HAS FAILED"
510 PRINT "  2. YOU'RE NOT PLAYING BY TH
E RULES."
520 PRINT :RETURN
```



## GETTING TO 100—THREE VERSIONS

Following are three versions of a very simple game: Be the first one to reach 100. This game can be played by two or more people or by one and a computer. Each person takes a turn selecting a number, which is added to the current total. The person who equals or breaks 100 on his turn is the winner.

This game has a great deal of similarity to NIM games, which we will introduce shortly. The game can be changed in a number of ways, such as limiting the range in which a player may select his/her next number, making the provision that the winner must get *exactly* 100 (rather than breaking it), etc. It may be dressed up with fancy graphics (such as showing an animal moving closer to a finish line), congratulations for the winner, consolation for the loser, and ways to heighten the suspense of who will finish first (see Chapter Two for some of these suggestions).

In the first version of the game, we have just the bare structure, without any limitations on range and barely any instructions. The second version shows how this structure may be expanded, providing an introduction, instructions, and protection so that the user may only pick a number in a specified range (in this case, 1–10). There is also a color/sound show for the winner. Finally, the third version adds competition with the computer, where the computer picks its number at random. In all three versions, either reaching or exceeding 100 is permissible; this may be easily changed so that reaching 100 exactly is the criteria for winning.

### Version 1—Getting to 100

```
$ REM ** GETTING TO 100:
2 REM ** THE BASIC STRUCTURE**
10 LET T=0
20 INPUT X
```

```

30 LET T=T+X
40 PRINT "TOTAL = ";T
50 IF T<100 THEN GOTO 20
60 PRINT "YOU WIN"

```

### Version 2—Getting to 100

```

1 REM **GETTING TO 100:
2 REM **DRESSED UP WITH INSTRUCTIONS
3 REM ** LIMIT ON YOUR INPUT **
5 PRINT "THE OBJECT OF THIS GAME IS TO BE"
6 PRINT "THE FIRST TO GET TO 100"
7 PRINT "YOU MAY ONLY USE NUMBERS FROM 1 TO 10."
8 PRINT "EACH TIME YOU ENTER A NUMBER,"
9 PRINT "IT IS ADDED TO THE TOTAL"
10 LET T=0
12 PRINT "TYPE IN A NUMBER BETWEEN 1 AND 10";
20 INPUT X
25 IF X<1 OR X>10 THEN PRINT "NUMBER MUST BE
    BETWEEN 1 & 10";GOTO 20
30 LET T=T+X
40 PRINT "TOTAL = ";T
50 IF T<100 THEN GOTO 12
60 PRINT "YOU WIN";
70 SETCOLOR 2, INT(RND(1)*16),4
80 SOUND 0,RND(1)*250,10,8
120 GOTO 60

```

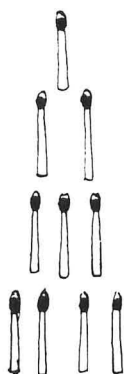
### Version 3—Getting to 100

```

10 REM *      <<< GET 100 >>>
20 REM *      copyright 1980 by
30 REM *      Herb Kohl & Ted Kahn
40 DIM WIN$(7)
50 PRINT "[CLEAR]THE OBJECT OF THIS GAME
    IS TO BE"
60 PRINT "THE FIRST TO GET TO 100"
70 PRINT "YOU MAY ONLY USE NUMBERS FROM
    1 TO 10."
80 PRINT "EACH TIME YOU ENTER A NUMBER,"
90 PRINT "IT IS ADDED TO THE TOTAL"
100 LET T=0
110 PRINT :TRAP 110

```

```
120 PRINT "CHOOSE YOUR NUMBER (1 THRU 10
) ";
130 INPUT GUESS
140 X=INT(GUESS)
150 IF X<1 OR X>10 THEN PRINT "NUMBER MU
ST BE BETWEEN 1 & 10 ";;GOTO 130
160 PRINT "[CLEAR]"
170 PRINT "YOU PICK ";X
180 LET T=T+X
190 PRINT "TOTAL = ";T
200 IF T>=100 THEN WIN$="YOU win";GOTO 2
70
210 C=INT(RND(1)*10)+1
220 PRINT :PRINT "I PICK ";C
230 T=T+C
240 PRINT "TOTAL = ";T:PRINT
250 IF T>=100 THEN WIN$="I win";GOTO 270
260 GOTO 120
270 GRAPHICS 2+16
280 FOR LOOP=1 TO 250
290 POSITION INT(RND(1)*14),INT(RND(1)*1
2)
300 PRINT #6;WIN$;
310 SOUND 1,RND(1)*255,6,8
320 NEXT LOOP
330 RUN
```



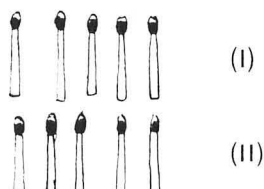
## NIM

NIM is an ancient Chinese gambling game that is usually played with matches. A simple version played with two piles of matches is easy to figure out and some gamblers use the two-pile version to lure people into playing more complex versions where figuring out a running strategy is not so easy. Here's the setup for a two-pile version of NIM. (There can be any number of matches in either pile.)

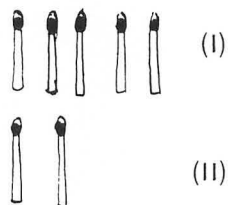


The players are to take turns removing as many matches as they please (at least one) from one of the piles. The players are free to decide which of the piles they will take matches from. The players can also decide whether the object of the game is to take the last match or to force one's opponent to take the last match. Here's an example of the game with the winner being the player who takes the last match:

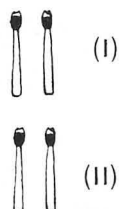
Setup:



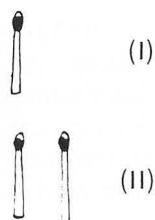
Player 1:  
(takes 3 matches from  
pile II)



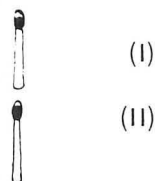
Player 2:  
(takes 3 matches from  
pile I)



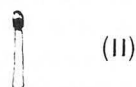
Player 1:  
(takes 1 match from  
pile I)



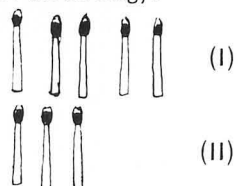
Player 2:  
(takes 1 match from pile II,  
thus forcing a win for him or  
herself since Player 1 can only  
draw matches from one column)



Player 1:  
(takes match from pile I and  
Player 2 gets the last match.)



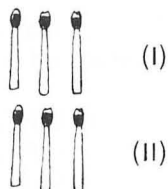
In this version of the two-pile game, the strategy is to force your opponent to draw from two equal piles. You can then keep equalizing the piles until you get last draw. Here's a simple version of the strategy:





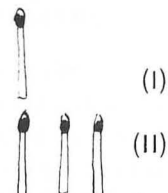
If the first player takes two matches from I, equalizing the columns, the second player can be forced to lose:

After Player 1 moves



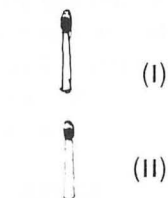
Now the second player can pick from either pile and will still be forced to face equal columns.

Player 2 moves, taking two from Column I

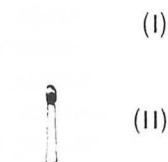


and Player 1 counters by equalizing the columns and forcing a win (by guaranteeing that he or she will get the last match)

Player 1

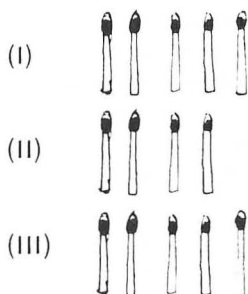


Player 2:



Player 2 takes the last match and wins.

The game of NIM becomes much more complex when three or more piles of matches are involved. Here's a game involving three piles. The rules are the same. Try this a few times before entering the more complex world of NIM that you can explore with your ATARI computer:



The player to take the last match wins.

NIM is a subtraction game and a very simple computer version of it can be programmed in only five lines:

```
10 INPUT X
20 INPUT Y
30 X=X-Y
40 IF X>0 THEN GOTO 20
50 PRINT "YOU WON"
```

In this program X is the game's starting number and Y the number to be subtracted in the race towards zero. The goal of the game (which can be played by any number of people, though two or three is probably best) is to be the player who reaches 0. You can be either 1 or 2.

Line 30 decreases the value of X each time a player puts in a value for Y, and line 40 keeps the game going until 0 is reached. When 0 is reached line 50 commands the computer to print "YOU WON." Players take turns putting in 1 or 2.

Here's a slightly dressed up version of this game which adds instructions and lets you know how big X is after each player's turn:



```
10 REM SIMPLE NIM WITH NUMBERS
20 PRINT "YOU START WITH X"
30 INPUT X
40 PRINT "TAKE AWAY 1 OR 2"
50 PRINT "PLAYER WHO TAKES LAST NUMBER WINS"
60 PRINT "WHICH WILL YOU TAKE AWAY,1 OR 2"
70 INPUT Y
80 X=X-Y
85 PRINT "NUMBER LEFT",X
90 IF X>0 THEN GOTO 60
100 IF X=0 THEN PRINT "YOU WON"
```

It is also possible, using concepts already introduced in Part I, to program a visual version of this simple NIM-type game. The version here uses 10 as the beginning number, but by changing the variables you could go up to 75 without running off the screen in GRAPHICS MODE 7.

```
10 REM THIS GAME INVOLVES TAKING ONE
11 REM OR TWO AWAY AND SEEING THE GAME
12 REM ON THE SCREEN
13 REM ANY NUMBER OF DOTS CAN BE USED
14 REM AS LONG AS THEY CONFORM TO GRAPHICS
20 GRAPHICS 7:COLOR 1
```

```

30 LET A=15
40 LET W=10
50 FOR X=1 TO 10
60 PLOT 10*X,A
70 NEXT X
80 PRINT "Y=?":INPUT Y
91 LET A=A+5
100 LET W=W-Y
110 FOR Z=1 TO W
120 PLOT 10*Z,A
130 NEXT Z
140 IF W=0 THEN GOTO 200
150 GOTO 80
200 PRINT "NIM":END

```

In Europe NIM is often played using wooden matches. It is possible to dress up our last version of NIM by using GRAPHICS 3 and color variations. Here is a simple subroutine which generates up to 15 matches on the screen. Try it several times.

```

1000 REM ROUTINE TO DRAW MATCHES
1010 GRAPHICS 3
1012 PRINT "HOW MANY MATCHES (1-15)";
1014 INPUT N
1020 FOR K=1 TO N*2 STEP 2
1030 COLOR 1
1035 PLOT K,5
1040 COLOR 2
1050 DRAWTO K,10
1060 NEXT K

```

Here's a full version of NIM for two players using an interesting graphical representation. This program was written by Dale Disharoon.

```

0 REM *      <<< NIM >>>
10 REM * COPYRIGHT 1981
20 REM * by Dale Disharoon
30 GRAPHICS 0:SETCOLOR 2,0,0
40 POSITION 14,4:PRINT " << NIM >> "
50 POSITION 2,8:PRINT " How high will yo
ur ":PRINT " pyramid be? (3-10) "
60 TRAP 60:POSITION 25,9:INPUT HIGH
70 IF HIGH<3 OR HIGH>10 THEN 60

```



```

80 POSITION 2,15:PRINT " 1 - Player taki
ng last block wins "
90 PRINT " 2 - Player taking last block
loses "
100 TRAP 100:POSITION 11,18:PRINT " Ente
r 1 or 2 ";;INPUT LAST
110 IF LAST<1 OR LAST>2 THEN 100
120 PRINT "[CLEAR]";POKE 752,1
130 POSITION 14,1:PRINT " << NIM >> "
140 DIM X(HIGH,HIGH),Y(HIGH,HIGH),B(10)
150 FOR H=1 TO HIGH
160 B(H)=H
170 FOR W=1 TO H
180 READ X:X(H,W)=X:Y(H,W)=H+4
190 POSITION X(H,W),Y(H,W):PRINT CHR$(16
0);CHR$(130);
200 NEXT W:PRINT " ";H:NEXT H
210 DATA 11,10,12,9,11,13,8,10,12,14,7,9
,11,13,15,6,8,10,12,14,16,5,7,9,11,13,15
,17,4,6,8,10,12,14,16,18
220 DATA 3,5,7,9,11,13,15,17,19,2,4,6,8,
10,12,14,16,18,20
230 POSITION 23,6:PRINT "* Last block *"
:POSITION 28,7
240 IF LAST=1 THEN PRINT "Wins"
250 IF LAST=2 THEN PRINT "Loses"
260 P=P+1:IF P>2 THEN P=1
270 TRAP 270:POSITION 2,17:PRINT "[DEL L
INE] Player #";P;" takes from level-";;I
NPUT H
280 IF H<1 OR H>HIGH THEN PRINT "[UP][DE
L LINE]Sorry, there is no level ";H:FOR
T=1 TO 500:NEXT T:GOTO 270
290 IF B(H)=0 THEN PRINT "[UP][DEL LINE]
Level ";H;" is gone";FOR T=1 TO 500:NEXT
T:GOTO 270
300 TRAP 300:POSITION 2,19:PRINT "[DEL L
INE] Player #";P;" takes how many-";;INP
UT N
310 IF N<1 THEN PRINT "[UP][DEL LINE]I c
an't let you do that";FOR T=1 TO 500:NEX
T T:GOTO 300
320 IF N>B(H) THEN PRINT "[UP][DEL LINE]
There aren't that many in level ";H:FOR

```

```

T=1 TO 500:NEXT T:GOTO 300
330 B(H)=B(H)-N
340 FOR T=B(H)+1 TO B(H)+N:POSITION X(H,
T),Y(H,T):PRINT " ":GOSUB 380:NEXT T
350 POSITION 2,17:PRINT "[DEL LINE][DOWN
][DEL LINE]"
360 GOSUB 400
370 GOTO 260
380 FOR X=21 TO 0 STEP -3:SOUND 0,X,10,8
:NEXT X:SOUND 0,0,0,0:RETURN
390 FOR X=50 TO 0 STEP -2:SOUND 0,X,10,8
:SOUND 1,X+1,10,8:NEXT X:SOUND 0,0,0,0:S
OUND 1,0,0,0:RETURN
400 FOR T=1 TO HIGH:IF B(T) THEN POP :RE
TURN
410 NEXT T:POSITION 5,17
420 IF LAST=1 THEN FOR T=1 TO 5:GOSUB 39
0:NEXT T:PRINT "   ***   Player #";P;" wi
ns!   ***   ":GOTO 450
430 SOUND 0,5,4,8:FOR T=1 TO 100:NEXT T:
SOUND 0,0,0,0
440 PRINT "   ***   Player #";P;" loses _
***   "
450 OPEN #1,4,0,"K:"
460 POSITION 14,22:PRINT "Press RETURN"
470 GET #1,T:RUN

```


## DICE

It is very easy to simulate the rolling of dice using the RND(1) function of your ATARI.  $(\text{RND}(1)*6)+1$  will generate a random number between 1 and 6, which is what happens when you roll one die.  $(\text{RND}(1)*12)+1$  produces the same result as rolling two dice. What RND(1) function would produce the same result as rolling four dice?


Here's a simple program that prints out numbers that result from random selections of numbers from 1 to 6 which are the same as tosses of 1 die:

```
10 - R=INT(RND(1)*6)+1
20 - PRINT R
30 - GOTO 10
```

A slight modification results in having the computer print out dots (as you would see on the die) instead of numbers:

```
5 REM DICE1 (FIRST REPRESENTATION): THROW ONE DIE
  & SHOW THE RESULT
6 REM (C) BY TED M. KAHN, 1980
10 R=INT(RND(1)*6)+1
20 FOR I=1 TO R
30 PRINT "[T]"; 
40 NEXT I
45 PRINT
50 GOTO 10
```

Here's a further elaboration in which two dice are tossed once and you are given the choice of whether you want to roll them again or not:

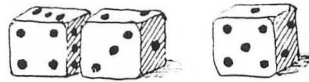
```
5 REM DICE2 (SECOND REPRESENTATION): THROW TWO
  DICE & SHOW THE RESULT
6 REM (C) BY TED M. KAHN, 1980
8 DIM Y$(1)
10 R1=INT(RND(1)*6)+1
15 R2=INT(RND(1)*6)+1
20 FOR I=1 TO R1
30 PRINT "[T]"; 
40 NEXT I
45 PRINT ,
50 FOR J=1 TO R2
```



Remember, [T] means hold down the [CTRL] key and, at the same time, type the letter "T." This will give a graphics dot (•).

```
60 PRINT "[T]";  
70 NEXT J  
80 PRINT :PRINT  
90 PRINT "THROW AGAIN";  
95 INPUT Y$:IF Y$="Y" THEN PRINT :GOTO 10  
100 END
```

Following are some complex games of chance that involve the computer making random choices and simulating the tossing of dice.



## THRICE DICE

Games of chance and gambling are among the oldest games known, and they are still popular in most cultures in the world today.

Using the concepts we introduced about random numbers,  $(\text{INT}(\text{RND}(1)*6)+1)$ , this game simulates rolling three dice and shows the results using the GRAPHICS 3 mode of the ATARI. In fact, the subroutine from lines 870-1080 (along with the DIMension statement at line 20) may be modified for use in other games in which you need a dice-throwing routine. The game also uses some fancy dressings designed by Len Lindsay to give the introduction a "neon-sign" or flashing advertising effect. These effects are created from lines 130-220 and make use of the large text in GRAPHICS 2+16 (full screen).

The actual game also involves some betting strategy. You start with \$100 and may bet as little or as much as you like on any number (1-6) being thrown. If the number comes up on *at least* one die, you win double your bet; if it comes up on two dice, you win *triple*; and if it comes up on all three dice (odds are one chance in 216), you win.

```

10 REM *      <<< THRICE DICE >>>
20 REM *  (C) 1980 by Len Lindsay
30 DIM DX(2),DY(2),N(2),A$(6)
40 CS=100
50 MAXCS=900000
60 COL=1
70 GRAPHICS 2+16:PRINT #6;"THRICE DICE"
80 POSITION 0,2:PRINT #6;"TO BEGIN"
90 PRINT #6;"HIT start"
100 POSITION 0,7:PRINT #6;"FOR INSTRUCTI
    ONS"
110 PRINT #6;"HIT option"
120 POSITION 2,11:PRINT #6;"(C) 1980"
130 KEY=PEEK(53279)
140 IF KEY=3 THEN GOSUB 630:GOTO 70
150 IF KEY=5 THEN CS=CS+100:IF CS>1000 T
    HEN CS=CS+900:IF CS>10000 THEN CS=CS+900
    0:IF CS>100000 THEN CS=CS+90000
160 IF CS>MAXCS THEN CS=100

```



```

170 IF KEY=6 THEN 230
180 POSITION 0,5:PRINT #6;"CASH=$";CS;"
select
190 SETCOLOR 1,0,RND(1)*256
200 FOR DELAY=1 TO 99:NEXT DELAY
210 PN=12-PN:SETCOLOR 2,0,PN*RND(1)*256
220 GOTO 130
230 GRAPHICS 3
240 SETCOLOR 2,0,0
250 PRINT "[CLEAR]";
260 PRINT "YOU HAVE $";CS;" CASH."
270 IF CS>999999 THEN 1140
280 PRINT "HOW MUCH WILL YOU BET";
290 TRAP 250
300 POKE 764,255
310 INPUT A$
320 IF A$="H" OR A$="HELP" THEN GOSUB 63
0:GOTO 230
330 BET=INT(VAL(A$))
340 IF BET<1 THEN 250
350 IF BET>CS THEN 250
360 PRINT "WHAT NUMBER WILL YOU BET ON";
370 TRAP 360
380 POKE 764,255
390 INPUT A$
400 IF A$="H" OR A$="HELP" THEN GOSUB 63
0:GOTO 230
410 CS=CS-BET
420 PRINT "[CLEAR]"
430 NUM=INT(VAL(A$))
440 IF NUM=0 OR NUM>6 THEN 370
450 TRAP 34567
460 FOR A=0 TO RND(1)*10
470 GOSUB 870
480 PRINT "[CLEAR]YOU BET $";BET;" ON ";
NUM;" ";
490 NEXT A
500 MATCH=0
510 FOR A=0 TO 2
520 IF NUM=N(A) THEN MATCH=MATCH+1
530 NEXT A
540 WIN=0
550 IF MATCH>0 THEN WIN=BET*(MATCH+1)
560 CS=CS+WIN

```

```

570 PRINT MATCH;" MATCH."
580 PRINT "YOU GET $";WIN;" CASH BACK!"
590 IF CS<1 THEN 1130
600 GOTO 260
610 GOTO 610
620 END
630 GRAPHICS 1+16
640 PRINT #6;"THIS IS A GAMBLING"
650 PRINT #6;"DICE GAME. YOU BET "
660 PRINT #6;"MONEY ON ONE OF THE"
670 PRINT #6;"6 POSSIBLE NUMBERS."
680 PRINT #6
690 PRINT #6;"if none of the dice"
700 PRINT #6;"show that number"
710 PRINT #6;"you lose the bet."
720 PRINT #6
730 PRINT #6;"IF 1 MATCHES YOU"
740 PRINT #6;"GET DOUBLE YOUR BET"
750 PRINT #6;"BACK (5 BET WINS 10)"
760 PRINT #6;"if two dice match"
770 PRINT #6;"you get triple back"
780 PRINT #6
790 PRINT #6;"-IF ALL DICE MATCH"
800 PRINT #6;"YOU GET 4 TIMES BACK"
810 PRINT #6;"SOUNDS GOOD - HUH!!"
820 PRINT #6
830 PRINT #6
840 PRINT #6;"hit start to return";
850 GOSUB 1090:IF KEY<>6 THEN 850
860 RETURN
870 FOR D=0 TO 2
880 N(D)=INT(RND(1)*6)+1
890 NEXT D
900 FOR D=0 TO 2
910 DX(D)=INT(RND(1)*6)+13*D
920 DY(D)=INT(RND(1)*13)
930 NEXT D
940 PRINT #6;"[CLEAR]"
950 FOR D=0 TO 2
960 COL=3-COL:COLOR COL
970 FOR R=DY(D) TO DY(D)+6
980 PLOT DX(D),R:DRAWTO DX(D)+6,R
990 NEXT R
1000 NEXT D

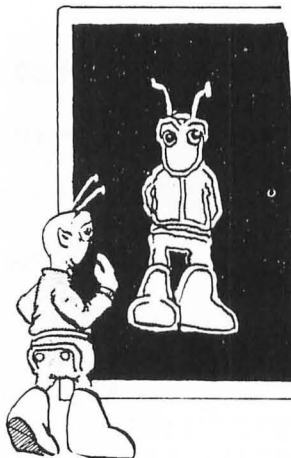
```

```

1010 COLOR 0
1020 FOR D=0 TO 2
1030 IF N(D)>1 THEN PLOT DX(D)+1,DY(D)+1
:PLOT DX(D)+5,DY(D)+5
1040 IF N(D)>3 THEN PLOT DX(D)+1,DY(D)+5
:PLOT DX(D)+5,DY(D)+1
1050 IF N(D)>5 THEN PLOT DX(D)+1,DY(D)+3
:PLOT DX(D)+5,DY(D)+3
1060 IF N(D)/2>INT(N(D)/2) THEN PLOT DX(
D)+3,DY(D)+3
1070 NEXT D
1080 RETURN
1090 IF PEEK(53279)<>7 THEN 1090
1100 KEY=PEEK(53279):IF KEY=7 THEN 1100
1110 IF PEEK(53279)<>7 THEN 1110
1120 RETURN
1130 PRINT "YOU HAVE NO MONEY LEFT":GOTO
1150
1140 PRINT "YOU BROKE THE BANK!!!"
1150 PRINT "HIT START TO PLAY AGAIN.";
1160 GOSUB 1090
1170 IF KEY<>6 THEN 1160
1180 RUN

```

This is a complex program and is not easy to copy. Once you get it working be sure to copy it on tape or disk unless you have a lot of time and enjoy the mechanical work of typing a long program.



## REVERSE

REVERSE was invented by Peter L. Sessions of People's Computer Co. a number of years ago and was first popularized in the magazine *PCC* (later known as *People's Computers, Recreational Computing*, and now a part of *COMPUTE!* magazine). It has since become very popular on a large number of mini- and microcomputers. The version given here was designed by one of the authors to show how home computers like the ATARI 400™ or ATARI 800™ can make a good game even better, using a little animation, color, and sound.

REVERSE involves manipulating a string of digits so that the final string is in numerical order. The game begins by the computer presenting a series of one-digit numbers in scrambled order; each digit occurs only once, and the size of the string may vary in different versions of the game (the longer the string, the more difficult it is). So, for example, let's say we had a string of five digits; we might start off with

34215

Our object is to end up with 12345. To get there, our only legal "move" is to *reverse* a set number of these digits, always beginning from the left-hand side of the string. Each reversal thus changes the string, moving the order closer and closer to the desired goal (one hopes). With our example above, the game might proceed like this:

34215      Reverse 2 (that is, take the first two numbers and reverse them).

43215      Reverse 4 (reverse the *first four* numbers).

12345      and we've won in only two turns!

Now, this one was pretty easy, but they get harder with longer strings: The classic game is played with nine digits (1-9). It can be proven mathematically that any string of nine digits can be properly ordered in  $15$  turns ( $2n-3$ ), where  $n$  is the number of digits in the string; however, no one has yet (to our knowledge) figured out an algorithm which will find the *smallest* number of moves for any given initial nine-digit string. After playing this game several times, perhaps you'll be the one!

Here's the program for the game:

```

10 REM *      <<< REVERSE >>>
20 REM *      Adapted by Ted Kahn
30 REM *      from P.L. Sessions P.C.C.
40 DIM NAME$(20),A$(10)
50 DIM XX$(1)
60 DIM ELIST$(9),MYLIST$(9),RLIST$(9)
70 NAME$="REVERSE"
80 GOSUB 290
90 OPEN #1,4,0,"K:"
100 PRINT "WELCOME TO THE GAME OF REVERS
E"
110 PRINT "DO YOU NEED INSTRUCTIONS?";:G
OSUB 870:IF XX=41 THEN GOSUB 1000
120 PRINT "[CLEAR]HOW MANY DIGITS WOULD
YOU":PRINT "LIKE (3-9)";
130 GOSUB 870:MAX=XX:REM INPUT
140 PRINT XX;
150 IF MAX<3 OR MAX>9 THEN PRINT "[CLEAR
]SORRY...ONLY NUMBERS BETWEEN 3-9":GOTO
120
160 GOSUB 420:REM INITIAL SEQUENCE
170 TURN=TURN+1
180 PRINT "[CLEAR]"
190 PRINT "TURN # ";TURN
200 PRINT :PRINT "REVERSE HOW MANY? ";
210 GOSUB 870:REM INPUT
220 R=XX:PRINT R;
230 FOR WW=1 TO 200:NEXT WW
240 IF R<2 OR R>MAX THEN 270
250 GOSUB 620
260 GOTO 170
270 PRINT "CAN ONLY REVERSE FROM 2 - ";M
AX;" DIGITS":FOR WW=1 TO 1200:NEXT WW
280 GOTO 180
290 X=12

```

```

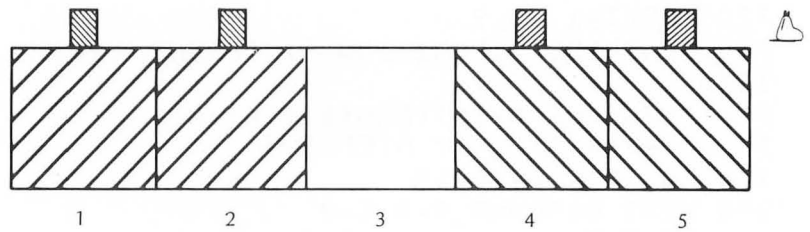
300 GRAPHICS 2
310 FOR I=1 TO LEN(NAME$)
320 POSITION X+1,3
330 PRINT #6;NAME$(LEN(NAME$)-I+1,LEN(NA
ME$))
340 SOUND 0,INT(RND(0)*255)+1,10,12
350 FOR WW=1 TO 80:NEXT WW
360 X=X-1
370 NEXT I
380 SOUND 0,0,0,0
390 POSITION 9,5:PRINT #6;"by"
400 POSITION 6,7: ? #6;"ted kahn"
410 RETURN
420 FOR K=1 TO MAX
430 MYLIST$(K,K)="0":ELIST$(K,K)="0"
440 NEXT K
450 FOR I=1 TO MAX
460 RN=INT(RND(0)*MAX)+1
470 IF ELIST$(RN,RN)<>"0" THEN 460
480 MYLIST$(I,I)=STR$(RN):ELIST$(RN,RN)=
STR$(RN)
490 NEXT I
500 GRAPHICS 2
510 X=9+INT(MAX/2)
520 FOR I=1 TO MAX
530 GRAPHICS 2
540 POSITION X,4
550 PRINT #6;MYLIST$(I,I)
560 SOUND 1,INT(RND(0)*100)+(2*I),10,10:
WW=5:GOSUB 690
570 SOUND 1,0,10,10:WW=5:GOSUB 690
580 X=X-1
590 NEXT I
600 SOUND 1,0,0,0
610 RETURN
620 FOR S=1 TO R
630 RLIST$(R-S+1,R-S+1)=MYLIST$(S,S)
640 NEXT S
650 MYLIST$(1,R)=RLIST$(1,R)
660 IF MYLIST$=ELIST$ THEN 720
670 GOSUB 500
680 RETURN
690 REM WAIT LOOP
700 FOR WAIT=1 TO WW:NEXT WAIT
710 RETURN

```

```

720 FOR I=1 TO 9
730 GRAPHICS 2:POSITION 5,4:PRINT #6;MYL
IST$
740 SETCOLOR 4,INT(RND(0)*16),8
750 FOR Q=15 TO 30 STEP 0.5
760 SOUND 0,Q,10,10
770 NEXT Q:SOUND 0,0,0,0
780 NEXT I
790 PRINT "CLEAR]YOU WON IN ";TURN;" TU
RN";:IF TURN<>1 THEN PRINT "S";
800 PRINT "!":SETCOLOR 4,0,0
810 TRAP 840:PRINT "WOULD YOU LIKE TO PL
AY AGAIN (Y,N)";
820 INPUT A$:IF A$(1,1)="N" THEN 840
830 TURN=0:POP :? "CLEAR]":GOTO 120
840 PRINT "THANKS FOR PLAYING!":END
870 REM INPUT SUBROUTINE
880 SOUND 0,200,10,8:FOR WS=1 TO 100
890 NEXT WS:SOUND 0,0,0,0:POKE 764,255
900 GET #1,XX:XX=XX-48
910 TRAP 920
920 TRAP 34567
930 RETURN
1000 POKE 752,1:PRINT "CLEAR]YOU WILL B
E GIVEN SOME SCRAMBLED":PRINT "NUMBERS F
ROM 1 TO THE NUMBER YOU"
1010 PRINT "CHOOSE. ( 3 - 9 ) PRESS RET
URN":GOSUB 870
1020 PRINT "CLEAR]BY REVERSING BLOCKS O
F NUMBERS TRY":PRINT "TO UNSCRAMBLE THEM
SO THEY ARE"
1030 PRINT "IN THE USUAL ORDER OF 1,2,3,
4,5 ETC.":PRINT "PRESS RETURN":GOSUB 87
0
1040 PRINT "CLEAR]WHEN YOU REVERSE A BL
OCK OF NUMBERS":PRINT "THE REVERSAL ALWA
YS STARTS FROM THE"
1050 PRINT "LEFT. FOR EXAMPLE... PRESS
RETURN":GOSUB 870
1060 PRINT "CLEAR]IF YOU REVERSE 3 NUMB
ERS OF THIS":PRINT "COMBINATION: 6421735
YOU WOULD GET"
1070 PRINT "2461735. ONLY THE 6,4, AND
2 ARE":PRINT "REVERSED. READY? PRESS
RETURN":GOSUB 870:RETURN

```



## QWERT

The object of QWERT is to switch the colored boxes located in squares 1 and 2 with those in squares 4 and 5. There are two legal moves for doing this:

1. Move any box into an *empty* neighboring square by typing the number of the box to be moved;
2. Jump *over* a box *into* an empty square, again by typing the number of the box to be moved.

With some planning, this can be done in *eight moves*. Readers interested in mathematics may wish to prove this for themselves and should also compare this game with the traditional puzzles, *The Towers of Hanoi*, or the "Missionaries and Cannibals" puzzles.

```

10 REM *          <<< QWERT >>>
20 REM *    developed by Len Lindsay
30 REM *    and modified by Ted Kahn
40 REM * Copyright 1980 by Len Lindsay
50 GOSUB 930
60 DIM A$(10),PA$(1),P(7)
70 DIM XX$(1)
80 GRAPHICS 3
90 SETCOLOR 1,5,8
100 P(0)=3
110 P(1)=1
120 P(2)=1
130 P(3)=0
140 P(4)=2
150 P(5)=2
160 P(6)=3
170 P(7)=3
180 GOSUB 490:REM DRAW BOARD

```



The small square markers at the top are to identify the goal of switching the colors underneath these markers.



```

190 MV=0:REM MOVES
200 GOSUB 590:REM DRAW PIECES
210 IF I>0 THEN 250
220 ? " [CLEAR]":? "THIS IS THE GAME OF Q
WERT"
230 ? :? "WOULD YOU LIKE INSTRUCTIONS (Y
,N)":GOSUB 970:REM GET
240 IF XX$="Y" THEN GOSUB 760
250 PRINT "[CLEAR]";
260 PRINT "   1       2       3       4       5"
:PRINT "MOVE #";MV+1
270 TRAP 250
280 GOSUB 970:REM GET
290 M=XX
300 IF M<1 OR M>5 THEN ? "ILLEGAL MOVE--
TRY AGAIN":GOSUB 920:GOTO 260
310 REM
320 P=P(M):REM PIECE TO MOVE?
330 IF P=0 THEN ? "CAN'T DO THAT--TRY AG
AIN":GOSUB 920:GOTO 260
340 GOSUB 370:REM CHECK MOVE & MOVE IT
350 GOTO 250:REM NEXT MOVE
360 END
370 IF P(M+1)=0 THEN P(M+1)=P:P(M)=0:GOT
O 440
380 IF P(M+2)=0 THEN P(M+2)=P:P(M)=0:GOT
O 440
390 IF P(M-1)=0 THEN P(M-1)=P:P(M)=0:GOT
O 440
400 IF M>1 THEN IF P(M-2)=0 THEN P(M-2)=
P:P(M)=0:GOTO 440
410 PRINT "YOU CAN'T MOVE THAT PIECE"
420 FOR WW=1 TO 300:NEXT WW
430 RETURN
440 MV=MV+1
450 GOSUB 590:REM DRAW PIECES
460 IF P(1)=2 AND P(2)=2 AND P(4)=1 AND
P(5)=1 THEN 680:REM WINNER
470 IF MV>22 THEN SETCOLOR 4,12,4:PRINT
"[CLEAR]YOU SEEM LOST":GOTO 720
480 RETURN
490 REM DRAW BOARD
500 COLOR 3
510 PLOT 2,12:DRAWTO 32,12

```

```

520 FOR X=2 TO 32 STEP 6
530 FOR Y=13 TO 17
540 PLOT X,Y
550 NEXT Y
560 NEXT X
570 PLOT 2,18:DRAWTO 32,18
580 RETURN
590 FOR PF=3 TO 27 STEP 6
600 COLOR P((PF+3)/6)
610 FOR Q=13 TO 17
620 PLOT PF,Q:DRAWTO PF+4,Q
630 NEXT Q
640 NEXT PF
650 COLOR 1:PLOT 23,11:PLOT 29,11
660 COLOR 2:PLOT 5,11:PLOT 11,11
670 RETURN
680 PRINT "[CLEAR]"
690 PRINT "YOU WON !!!";
700 SETCOLOR 4,4,4
710 IF MV=8 THEN PRINT "-PERFECT PLAYING
!"
720 PRINT "- TRY AGAIN?"
730 GOSUB 970:REM GET
740 IF XX$="Y" THEN 80
750 GRAPHICS 0:END
760 ? "[CLEAR]The object of this game is
to switch"
770 ? "the boxes on the left with the bo
xes"
780 ? "on the right in as few moves as"
790 ? "possible ( press RETURN to contin
ue)";:GOSUB 970
800 ? "[CLEAR] 1      2      3      4
5"? "The boxes are numbered as shown
here"
810 PRINT "HIT RETURN TO CONTINUE";:GOSU
B 970
820 ? "[CLEAR]A box may be moved to an e
mpty space"
830 ? "by typing its current number."
840 ? "(RETURN to continue)";:GOSUB 970
850 ? "[CLEAR]You may move either"
860 ? "(1) into an empty neighboring spa
ce"

```

```

870 ? "or (2) jump over 1 box into an "
880 ? "empty space ( RETURN to continue)
";:GOSUB 970
890 ? "[CLEAR]Try to do it in as few as
8 moves!"
900 ? "GOOD LUCK! (Press RETURN to start
)";:GOSUB 970
910 RETURN
920 FOR WW=1 TO 100:NEXT WW:RETURN
930 GRAPHICS 1:POSITION 8,4:PRINT #6;"QW
ERT"
940 POSITION 4,6:PRINT #6;" (C) 1980 BY"
:? #6;"      LEN LINDSAY  "?:? #6;"
      &"?:? #6;"      TED KAHN"
950 FOR WW=1 TO 99:NEXT WW
960 RETURN
970 REM 'GET' SUBROUTINE
980 SOUND 0,180,10,8
990 FOR S=1 TO 100:NEXT S
1000 SOUND 0,0,0,0
1010 POKE 764,255
1020 TRAP 1020
1030 CLOSE #4
1040 OPEN #4,4,0,"K:"
1050 GET #4,XX
1060 CLOSE #4
1070 XX$=CHR$(XX)
1080 TRAP 1110
1090 XX=0
1100 XX=VAL(XX$)
1110 TRAP 34567
1120 RETURN

```

Here are three other programs to copy and play. The first game is a version of the well-known game BINGO. The second game, PHANTOM, is a mystery game which mixes letters and numbers. The third game, BRAINBUSTER, is a more complex version of the QWERT game we saw earlier. The instructions for all three of these games are included in the programs themselves (note that BRAINBUSTER requires the use of an ATARI joystick controller).

```

10 REM *          <<< BINGO >>>
20 REM *          Copyright 1980 by
30 REM *          Len Lindsay
40 GRAPHICS 2
50 DIM LINE$(40),SPACE$(40),PART$(5),B$(
2),I$(2),N$(2),G$(2),O$(2):REM ONLY NEED
ED FOR CARD MAKER
60 LMARG=2:POKE 82,LMARG
70 DIM N(75)
80 DIM BINGO$(5)
90 DIM XX$(3)
100 DIM PICK(5)
110 BINGO$="BinGo"
120 SETCOLOR 2,14,4
130 PRINT #6;"      HIT start"
140 PRINT #6;"      when ready"
150 PRINT #6;"      TO PLAY"
160 PRINT #6;"      bingo?"
170 IF PEEK(53279)<>6 THEN SOUND 0,RND(1
)*255,10,15:GOTO 170
180 SOUND 0,0,0,0
190 GRAPHICS 2
200 PRINT #6;"wait please"
210 GOSUB 7000:REM INIT ARRAY
220 PRINT "HOW MANY BINGO CARDS DO YOU W
ANT"
230 PRINT "PRINTED (HIT RETURN FOR NONE)
?"
240 GOSUB 15000
250 IF XX=0 THEN 270:REM NO CARDS PRINTE
D
260 FOR CARDPRINT=1 TO XX:GOSUB 10000:NE
XT CARDPRINT
270 NF=0
280 PRINT #6;"shuffling"
290 GOSUB 7100:REM RANDOMIZE

```

```

300 NP=NP+1
310 IF NP=76 THEN PRINT "COME NOW! EVERY
    NUMBER IS CALLED":GOTO 3010
320 NM=N(NP):REM PICK NUMBER
330 GRAPHICS 2:PRINT #6;" THE number is:
    "
340 PRINT #6
350 PRINT #6," ";
360 GOSUB 4000:REM PRINT THE NUMBER
370 SETCOLOR 2,5,4
380 PRINT "[CLEAR]HIT 'B' IF SOMEONE HAS
    BINGO"
390 PRINT "HIT RETURN FOR THE NEXT NUMBE
    R"
400 GOSUB 15100:REM GET
410 IF XX$="B" THEN 1000
420 GOTO 300
430 END

1000 REM VERIFY BINGO CALL
1010 PRINT "[CLEAR]";:REM CLEAR WINDOW
1020 IF NP<4 THEN POSITION 1,5:PRINT #6;
    "NOT ENOUGH NUMBERS":PRINT #6;"WERE CALL
    ED YET":GOTO 370
1030 FREEFLAG=0:REM INIT
1040 GRAPHICS 1
1050 PRINT #6;"ENTER BINGO NUMBERS"
1060 PRINT #6;"DON'T INCLUDE THE"
1070 PRINT #6;"PRECEDING LETTER"
1080 PRINT #6
1090 PRINT #6;"ENTER f FOR FREE"
1100 PRINT #6;"ENTER 1 FOR LIST"
1110 PRINT #6
1120 FOR LOOP=1 TO 5
1130 PICK(LOOP)=0
1140 NEXT LOOP
1150 PRINT "ENTER THE WINNING NUMBERS"
1160 PRINT "(DON'T INCLUDE THE PRECEDING
    LETTER"
1170 GOSUB 15000:REM INPUT
1180 IF XX$="L" THEN GOSUB 2000:PRINT "E
    NTER WINNING NUMBERS"
1190 IF XX$="F" THEN IF FREEFLAG=0 THEN
    FREEFLAG=1:RT=RT+1:GOTO 1290
1200 IF XX=0 THEN PRINT "ENTER JUST THE
    DIGITS - ENTER F FOR FREE (BUT ONLY ON

```

```

CE)":GOTO 1170
1210 PICK(RT)=XX:IF RT<1 THEN 1250:REM S
KIP CHECK FOR REPEATS
1220 FOR LOOP=0 TO RT-1
1230 IF XX=PICK(LOOP) THEN PRINT "THAT W
AS ALREADY ENTERED- AGAIN?":PICK(LOOP)=0
:POP :GOTO 1170
1240 NEXT LOOP
1250 FOR LOOP=1 TO NP
1260 IF N(LOOP)=XX THEN RT=RT+1:POP :GOT
O 1290
1270 NEXT LOOP
1280 PRINT "THAT NUMBER IS NOT PICKED YE
T":GOSUB 14000:GOTO 370
1290 IF RT=5 THEN GOTO 3000
1300 GOTO 1170
2000 REM PRINT ALL CALLED NUMBERS
2010 GRAPHICS 1:ROW=0:COL=1:REM INIT
2020 FOR LOOP=1 TO NP
2030 NM=N(LOOP)
2040 POSITION COL,ROW
2050 ROW=ROW+1:IF ROW>18 THEN ROW=0:COL=
COL+4
2060 GOSUB 4000
2070 NEXT LOOP
2080 RETURN
3000 REM SOMEONE WON
3010 GRAPHICS 2
3020 FOR X=1 TO 3
3030 PRINT #6;"** a winner **"
3040 NEXT X
3050 PRINT #6
3060 PRINT #6;"ANOTHER GAME?";
3070 GOSUB 15100:REM GET
3080 IF XX$="Y" THEN RUN
3090 END
4000 REM PRINT THE BINGO NUMBER WITH
PRECEDING LETTER
4010 LT=INT((NM-1)/15+1)
4020 PRINT #6;BINGO$(LT,LT);NM
4030 RETURN
7000 REM INIT AND FILL ARRAY
7010 FOR X=0 TO 75
7020 N(X)=X

```

```

7030 NEXT X
7040 RETURN
7100 REM SHUFFLE THE ARRAY OF NUMBERS
7110 FOR X=1 TO 75
7120 X1=INT(RND(1)*75)+1
7130 X2=N(X)
7140 N(X)=N(X1)
7150 N(X1)=X2
7160 NEXT X
7170 RETURN
10000 REM PRINT BINGO CARD
10010 GOSUB 10900:REM SHUFFLE
10020 LINE$=" +-----+-----+-----+-----+
---+-----+
10030 SPACE$=" | | | |
| |
10040 PART$=" | "
10050 LPRINT :LPRINT " OFFICIAL ATAR
I BINGO CARD"
10060 LPRINT :LPRINT
10070 LPRINT " B I N
G O"
10080 FOR LINE=1 TO 5
10090 B$=STR$(N(LINE)):IF LEN(B$)<2 THEN
B$(LEN(B$)+1)=" "
10100 I$=STR$(N(LINE+15))
10110 N$=STR$(N(LINE+30)):IF LINE=3 THEN
N$="XX"
10120 G$=STR$(N(LINE+45))
10130 O$=STR$(N(LINE+60))
10140 LPRINT LINE$
10150 LPRINT SPACE$
10160 LPRINT PART$;B$;PART$;I$;PART$;N$;
PART$;G$;PART$;O$;" | "
10170 LPRINT SPACE$
10180 NEXT LINE
10190 LPRINT LINE$
10200 FOR LOOP=1 TO 7:LPRINT :NEXT LOOP
10210 RETURN
10900 REM SHUFFLE FOR CARD
10910 FOR LOOP=1 TO 61 STEP 15
10920 FOR LOOP2=LOOP TO LOOP+14
10930 TEMP=INT(RND(1)*15)+LOOP
10940 TEMP2=N(LOOP2)

```

```

10950 N(LOOP2)=N(TEMP)
10960 N(TEMP)=TEMP2
10970 NEXT LOOP2
10980 NEXT LOOP
10999 RETURN
14000 REM PAUSE
14001 REM PAUSE IS DEFAULT AT 999
14002 REM TO SET YOUR OWN PAUSE
14003 REM SIMPLY DO THIS ON THE CALL
14004 REM
14005 REM PAUSE=300 : GOSUB 14020
14006 REM
14007 REM NOTE GOSUB AT 14020 IF YOU
14008 REM SET YOUR OWN PAUSE LENGTH
14009 REM
14010 PAUSE=999
14020 FOR LOOP=1 TO PAUSE
14030 NEXT LOOP
14099 RETURN
15000 REM INPUT SUBROUTINE
15010 PRINT "[BELL]";
15020 POKE 764,255
15030 TRAP 15030
15040 INPUT XX$
15050 TRAP 15070
15060 XX=VAL(XX$)
15070 TRAP 34567:RETURN
15100 REM 'GET' SUBROUTINE
15110 PRINT "[BELL]";
15120 POKE 764,255
15130 TRAP 15130
15140 CLOSE #4
15150 OPEN #4,4,0,"K:"
15160 GET #4,XX
15170 CLOSE #4
15180 XX$=CHR$(XX)
15190 TRAP 15220
15200 XX=0
15210 XX=VAL(XX$)
15220 TRAP 34567:RETURN

```



```

10 REM <<< PHANTOM >>>
20 REM LEN LINDSAY,1980
30 DIM LETTER(10),XX$(3),FOUND(9),NUMBER
  (3),GUESS$(3),CLASS$(20)
40 REM FOLLOWING ARE VARIABLES USED FOR
  TARGET LINE NUMBERS
50 INSTRUCTIONS=1000:HINT=3000:DONE=9000
  :EXPLAIN=2000:TRANSLATE=8000:REPLY=7000
60 CHECK=6000:ASK=270:RATING=4000:WINNER
  =5000:RIGHT=5500:WRONG=5700
70 HINTCOUNT=0:GUESSCOUNT=0:FOUNDCOUNT=0
  :WRONGCOUNT=0:REM INIT
80 GRAPHICS 1:SETCOLOR 2,0,0
90 FOUND(LOOP)=0:REM INIT ARRAY
100 FOR LOOP=0 TO 9:LET LETTER(LOOP)=ASC
  ("A")+LOOP:FOUND(LOOP)=0:NEXT LOOP:REM I
  NIT ARRAYS
110 LET LETTER(10)=-1:REM INIT FOR INCOR
  RECT INPUT CHECK
120 POSITION 2,5:PRINT #6;"phantom numbe
rs"
130 FOR LOOP=0 TO 9:REM SHUFFLE
140 MOVE=INT(RND(1)*10)
150 TEMP=LETTER(MOVE)
160 LET LETTER(MOVE)=LETTER(LOOP)
170 LET LETTER(LOOP)=TEMP
180 NEXT LOOP
190 GUESSCOUNT=1:REM INIT
200 PRINT "[CLEAR]at any time you may ty
  pe HELP (for"
210 PRINT "instructions), HINT (to revea
  l one"
220 PRINT "digit), or END (to stop).]"
230 PRINT "Do you need instructions?";
240 GOSUB 15000
250 PRINT #6;"[CLEAR]";
260 IF XX$(1,1)="Y" THEN GOTO INSTRUCTIO
  NS
270 PRINT "[CLEAR]";
280 IF HINTCOUNT+FOUNDCOUNT>9 THEN GOTO
  DONE
290 FOR LOOP=65 TO 74
300 IF LOOP=70 THEN PRINT :PRINT " ";
310 PRINT CHR$(LOOP);"=";

```

```

320 FOR LOOP2=0 TO 9
330 IF LETTER(LOOP2)=LOOP THEN IF FOUND(
LOOP2)>0 THEN PRINT LOOP2;:POP :GOTO 360
340 NEXT LOOP2
350 PRINT "?";
360 PRINT " ";
370 NEXT LOOP
380 PRINT
390 PRINT "WHAT DO YOU THINK?"
400 GOSUB 15000:REM INPUT ROUTINE
410 GUESSCOUNT=GUESSCOUNT+1:REM ERRONEOU
S GUESSES ARE SUBTRACTED BY EXPLAIN
420 IF LEN(XX$)<>3 THEN GOTO EXPLAIN
430 GUESS$=XX$:REM XX$ FROM INPUT ROUTIN
E IS ASSIGNED TO GUESS$
440 IF GUESS$="HEL" THEN GOTO INSTRUCTIO
NS
450 IF GUESS$="HIN" THEN GOTO HINT
460 IF GUESS$="END" THEN GOTO DONE
470 IF GUESS$(1,1)>"J" OR GUESS$(1,1)<"A
" THEN GOTO EXPLAIN
480 GOSUB TRANSLATE
490 IF GUESS$(3,3)>"J" OR GUESS$(3,3)<"A
" THEN GOTO CHECK:REM CHECK FOR =
500 REM *** GOSUB TRANSLATE
510 IF GUESS$(2,2)="+" THEN ANSWER=NUMBE
R(1)+NUMBER(3):GOTO REPLY
520 IF GUESS$(2,2)="-" THEN ANSWER=NUMBE
R(1)-NUMBER(3):GOTO REPLY
530 IF GUESS$(2,2)="*" THEN ANSWER=NUMBE
R(1)*NUMBER(3):GOTO REPLY
540 GOTO EXPLAIN
550 IF GUESS$(2,2)<>"=" THEN GOTO EXPLAI
N
1000 PRINT "[CLEAR]";:REM INSTRUCTIONS
1010 PRINT "The ten digits (0123456789)
have"
1020 PRINT "each been replaced by letter
s"
1030 PRINT "(ABCDEFGHIJ). START=Continue
";:GOSUB 2920
1040 PRINT "[CLEAR]Each letter is used o
nly once, and"
1050 PRINT "all digits have been assigne
d one"

```

```

1060 PRINT "letter as its disguise. STAR
T=More";GOSUB 2920
1070 PRINT "[C]LEAR]You must figure out w
hat digit each"
1080 PRINT "letter represents. You are a
llowed to"
1090 PRINT "ask simple arithmetic expres
sions"
1100 PRINT "such as: A+C or D-F. STA
RT=More";GOSUB 2920
1110 PRINT "[C]LEAR]I will tell you the l
etter that"
1120 PRINT "represents the last digit of
the "
1130 PRINT "answer to your problem."
1140 GOSUB 2900
1150 PRINT "[C]LEAR]I will not tell you i
f the answer has"
1160 PRINT "more than one digit, nor if
it is"
1170 PRINT "positive or negative. Good l
uck!!!"
1180 GOSUB 2900
1999 REM GOTO EXPLAIN WHICH FOLLOWS IMM
EDIATELY
2000 REM EXPLAIN
2010 GUESSCOUNT=GUESSCOUNT-1:REM ERROR S
O SUBTRACT A GUESS TO BE FAIR
2020 PRINT "[C]LEAR]";REM CLEAR SCREEN
2030 PRINT "To guess the identity of a l
etter you"
2040 PRINT "type the letter, followed by
an equal"
2050 PRINT "sign, then the digit that yo
u think"
2060 PRINT "is its disguised digit. STA
RT=More";GOSUB 2920
2070 PRINT "[C]LEAR]type A=3 RETURN"
2080 PRINT " [C]Z]C]R]C]R]>the NUMBER
"
2090 PRINT " [C]Z]C]R]C]R]>an EQUAL SI
GN"
2100 PRINT " [C]Z]C]R]C]R]>the LETTER
START=More";

```

```

2110 GOSUB 2920
2120 PRINT "[CLEAR]If you can't guess a
letters identity"
2130 PRINT "you can submit a simple addi
tion,"
2140 PRINT "subtraction, or multiplicati
on prob:"
2150 PRINT "such as DxF or J-A STA
RT=More";GOSUB 2920
2160 PRINT "[CLEAR]type A-C RETURN"
2170 PRINT " [CZ]C]C]C]>any LETTER
(A-J)"
2180 PRINT " [CZ]C]C]C]>OPERATION S
IGN(+ - x)"
2190 PRINT " [CZ]C]C]C]>any LETTER(A
-J) START=More";GOSUB 2920
2199 GOTO ASK
2900 REM HIT START TO CONTINUE ROUTINE
2910 PRINT "HIT START TO CONTINUE";
2920 IF PEEK(53279)<>6 THEN 2920
2930 IF PEEK(53279)=6 THEN 2930:REM WAIT
TILL LET UP ON START KEY
2999 RETURN
3000 REM HINT
3010 HINTCOUNT=HINTCOUNT+1
3020 GOSUB 7900:REM POSITION CURSOR FOR
PRINT#6
3030 FOR LOOP=9 TO 0 STEP -1
3040 IF FOUND(LOOP)=0 THEN FOUND(LOOP)=1
:PRINT #6;"-->";CHR$(LETTER(LOOP)+32);"="
";LOOP;:POP :GOTO ASK
3050 NEXT LOOP
3060 PRINT "NO HINT":REM SHOULD NEVER EX
ECUTE THIS LINE OR THE NEXT LINE
3099 GOTO ASK
4000 REM RATING
4010 SCORE=1000
4020 IF HINTCOUNT=0 THEN 4060
4030 FOR LOOP=1 TO HINTCOUNT
4040 SCORE=SCORE/2
4050 NEXT LOOP
4060 SCORE=SCORE-WRONGCOUNT*10
4070 SCORE=SCORE-GUESSCOUNT*2
4080 SCORE=INT(SCORE)

```

```

4090 IF SCORE<0 THEN SCORE=10:REM MINIMUM SCORE
4099 RETURN
5000 REM WINNER
5010 PRINT "YOU WIN. ALL IDENTITIES ARE UNCOVERED"
5020 GOTO DONE
5500 REM RIGHT GUESS
5510 PRINT "[BELL]";:REM BEEP
5520 GOSUB 7900:REM POSITION CURSOR FOR PRINT#6
5530 PRINT #6;GUESS$;"yes";
5599 RETURN
5700 REM WRONG GUESS
5710 WRONGCOUNT=WRONGCOUNT+1
5720 GOSUB 7900:REM POSITION CURSOR FOR PRINT#6
5730 PRINT #6;GUESS$;"no";
5799 RETURN
6000 REM CHECK IF = IS RIGHT
6010 IF GUESS$(2,2)<>"=" THEN GOTO EXPLAIN
6020 TRAP EXPLAIN:REM TRAP NON NUMERIC
6030 NUMBER(3)=VAL(GUESS$(3,3))
6040 TRAP 34567:REM RELEASE TRAP
6050 IF ASC(GUESS$(1,1))<>LETTER(NUMBER(3)) THEN GOSUB WRONG:GOTO ASK
6100 IF FOUND(NUMBER(3))>0 THEN GOSUB 7900:PRINT #6;GUESS$;"---";:GOTO ASK:REM ALREADY FOUND IT
6200 GOSUB RIGHT
6300 FOUNDcount=FOUNDcount+1
6310 FOUND(NUMBER(3))=2
6400 IF FOUNDcount+HINTcount>9 THEN GOTO WINNER
6500 GOTO ASK
7000 REM REPLY
7010 ANSWER=ABS(ANSWER)
7020 ANSWER=INT(ANSWER)
7030 IF ANSWER>9 THEN ANSWER=ANSWER-INT(ANSWER/10)*10:GOTO 7010
7100 GOSUB 7900:REM SET CURSOR FOR PRINT #6

```

```

7110 IF ROW=20 THEN ROW=0:COL=COL+6:IF C
OL>15 THEN PRINT #6,"[CLEAR]";:ROW=0:COL
=1
7120 POSITION COL,ROW
7130 PRINT #6;GUESS$;"=";CHR$(LETTER(ANS
WER))
7300 GOTO ASK
7900 ROW=ROW+1
7910 IF ROW=20 THEN ROW=0:COL=COL+6:IF C
OL>15 THEN PRINT #6,"[CLEAR]";:ROW=0:COL
=1
7920 POSITION COL,ROW
7999 RETURN
8000 REM TRANSLATE
8010 NUMBER(1)=10:NUMBER(3)=10:REM INIT
8020 FOR LOOP=1 TO 3 STEP 2
8030 FOR TEST=0 TO 9
8040 IF CHR$(LETTER(TEST))=GUESS$(LOOP,L
OOP) THEN NUMBER(LOOP)=TEST
8050 NEXT TEST
8060 NEXT LOOP
8099 RETURN
9000 REM DONE
9010 GRAPHICS 0
9020 PRINT "[CLEAR]";:REM CLEAR SCREEN
9030 PRINT "THANK YOU FOR PLAYING"
9040 PRINT
9050 GUESSCOUNT=GUESSCOUNT-1
9060 PRINT "WITH ONLY ";GUESSCOUNT;" GUE
SSES"
9070 PRINT "          AND ";WRONGCOUNT;" WRO
NG GUESS";:IF WRONGCOUNT<>1 THEN PRINT "
ES";
9080 PRINT
9090 PRINT "          AND ";HINTCOUNT;" HINT
";:IF HINTCOUNT<>1 THEN PRINT "S";
9100 PRINT
9110 IF FOUNDcount=0 THEN PRINT "YOU COU
LD DEDUCE NONE OF THE NUMBERS":GOTO 9250
9120 PRINT "YOU FOUND THE IDENTITY OF "
9130 IF FOUNDcount=10 THEN PRINT "ALL OF
THE NUMBERS":GOTO 9150
9140 PRINT FOUNDcount;" OF THE NUMBERS."
9150 PRINT

```

```

9160 GOSUB 14000
9170 FOR LOOP=0 TO 9
9180 PRINT LOOP;" WAS REPLACED BY ";CHR$(
  LETTER(LOOP));
9190 IF FOUND(LOOP)>1 THEN PRINT " (YOU
  FOUND IT)";
9200 PRINT
9210 NEXT LOOP
9220 PRINT
9230 GOSUB RATING
9240 PRINT "YOUR RATING IS: ";SCORE
9250 RESTORE 9500+SCORE/10
9260 READ CLASS$
9270 PRINT "YOUR CLASS IS: ";CLASS$
9280 PRINT
9290 PRINT "HIT START FOR ANOTHER CASE";
9510 DATA BEGINNER
9520 DATA ROOKIE
9535 DATA PRIVATE
9540 DATA CHIEF
9545 DATA INSPECTOR
9550 IF PEEK(53279)<>6 THEN 9550
9560 RUN
9570 DATA CHIEF INSPECTOR
9580 DATA DETECTIVE
9586 DATA CHIEF DETECTIVE
9590 DATA SLEUTH
9594 DATA CHIEF SLEUTH
9597 DATA SUPER SLEUTH
9600 DATA SUPER SLEUTH***
9999 END
14000 REM WINNING SOUNDS
14010 FOR L1=40 TO 0 STEP -8
14020 FOR L2=40 TO L1 STEP -1:SOUND 0,L2
  ,10,10:NEXT L2
14030 NEXT L1:SOUND 0,0,0,0
14099 RETURN
15000 REM INPUT SUBROUTINE WITH BEEP
15001 REM XX$ IS STRING TO BE INPUT
15002 REM XX IS THE NUMERIC VALUE OF THE
  INPUT
15003 REM MUST DIM XX$(N) - N IS THE L
  IMIT YOU WISH ON THE INPUT STRING LENGTH
15010 PRINT "[BELL]";:REM BUZZER

```

```
15020 POKE 764,255:REM CLEAR BUFFER
15030 TRAP 15030:REM TRAP ERROR ON INPUT
15040 INPUT XX$:REM INPUT STRING
15050 TRAP 15090:REM TRAP IF NO NUMERIC
AVAILABLE IN STRING
15060 XX=VAL(XX$):REM NUMERIC
15090 TRAP 34567:REM SPRING TRAP
15099 RETURN
```



```

1 REM * <<< BRAINBUSTER (BRAINBOX)>>>
2 REM *      Copyright (C) 1980 by
3 REM *      Len Lindsay & Ted M. Kahn
5 GRAPHICS 2
10 GOSUB 20000
12 DIM A$(1),PZ(10)
100 MV=1
101 SETCOLOR 4,0,0
110 GOSUB 6000:REM DRAW BOARD
115 GOSUB 7000:REM RANDOM PIECES
116 GOSUB 7100:REM DRAW PIECES
118 GOSUB 30000:REM GIVE INSTRUCTIONS
120 TRAP 120
121 PRINT "[CLEAR]MOVE # ";MV;"?"
130 IF STRIG(JS)=0 THEN 130:REM WAIT TIL
L BUTTON LET UP
131 ST=STICK(JS):IF STRIG(JS)=1 THEN 131
138 GOSUB 140+(ST-5)
139 GOTO 155
140 M=3:RETURN
141 M=9:RETURN
142 M=6:RETURN
144 M=1:RETURN
145 M=7:RETURN
146 M=4:RETURN
148 M=2:RETURN
149 M=8:RETURN
150 M=5:RETURN
155 IF PZ(M)=0 THEN PRINT "[CLEAR]CHOOSE
ONLY LIT BOXES - YOUR MOVE":GOTO 130
160 MV=MV+1
190 PZ(M)=0:REM TURN OFF CHOSEN BOX
191 GOSUB 7100:REM UPDATE BOARD
250 GOSUB 300+M*10
260 WN=0
265 GOSUB 7100
270 FOR Q=1 TO 9:WN=WN+PZ(Q):NEXT Q
280 IF WN=8 AND PZ(5)=0 THEN 7500:REM WI
NNER *****
290 IF WN=0 THEN 7600:REM LOSER **
300 GOTO 120:REM NEXT MOVE
310 PZ(2)=1-PZ(2)
311 PZ(4)=1-PZ(4)
312 PZ(5)=1-PZ(5)

```

```

319 RETURN
320 PZ(1)=1-PZ(1)
321 PZ(3)=1-PZ(3)
329 RETURN
330 PZ(2)=1-PZ(2)
331 PZ(5)=1-PZ(5)
332 PZ(6)=1-PZ(6)
339 RETURN
340 PZ(7)=1-PZ(7)
341 PZ(1)=1-PZ(1)
349 RETURN
350 PZ(8)=1-PZ(8)
351 PZ(4)=1-PZ(4)
352 PZ(6)=1-PZ(6)
353 PZ(2)=1-PZ(2)
359 RETURN
360 PZ(9)=1-PZ(9)
361 PZ(3)=1-PZ(3)
369 RETURN
370 PZ(8)=1-PZ(8)
371 PZ(4)=1-PZ(4)
372 PZ(5)=1-PZ(5)
379 RETURN
380 PZ(7)=1-PZ(7)
381 PZ(9)=1-PZ(9)
389 RETURN
390 PZ(8)=1-PZ(8)
391 PZ(5)=1-PZ(5)
392 PZ(6)=1-PZ(6)
399 RETURN
999 END
1000 JS=99
1010 FOR X=0 TO 3
1020 IF STRIG(X)=0 THEN JS=X:GOSUB 1100
1030 NEXT X
1040 IF JS>3 THEN 1010
1050 RETURN
1100 IF STRIG(JS)=0 THEN 1100:REM WAIT T
    ILL BUTTON RELEASED
1199 RETURN
6000 COLOR 3
6010 FOR Z=0 TO 12 STEP 6
6015 PLOT 10,Z:DRAWTO 28,Z
6020 FOR X=10 TO 28 STEP 6

```

```

6030 FOR Y=1 TO 5
6040 PLOT X,Y+Z
6050 NEXT Y
6060 NEXT X
6070 NEXT Z
6080 PLOT 10,18;DRAWTO 28,18
6099 RETURN
7000 FOR R=0 TO 10
7010 FZ(R)=INT(RND(0)+0.5)
7020 NEXT R
7099 RETURN
7100 PN=0
7110 FOR Z=12 TO 0 STEP -6
7120 FOR X=1 TO 13 STEP 6
7122 PN=PN+1;COLOR FZ(PN)
7125 FOR Y=1 TO 5
7130 PLOT X+10,Y+Z;DRAWTO X+14,Y+Z
7140 NEXT Y
7150 NEXT X
7160 NEXT Z
7190 RETURN
7500 GOSUB 8000;REM UPDATE CUMULATIVE
7570 PRINT "CLEAR;YOU WON IN ";MV;" MOV
ES"
7571 SETCOLOR 4,4,4
7575 PRINT "PLAY AGAIN";
7576 INPUT A$
7590 IF A$="Y" THEN 100
7595 GRAPHICS 0;END
7600 GOSUB 8000;REM UPDATE
7605 PRINT "CLEAR;YOU CAN'T WIN NOW"
7606 SETCOLOR 4,12,4
7610 GOTO 7575
8000 NG=NG+1
8010 AV=INT((AV*(NG-1)+MV)/NG)
8099 RETURN
20000 GRAPHICS 2;POSITION 4,1;PRINT #6;"
brain buster";POSITION 1,3;? #6;"COPYRIG
HT (C) 1980";POSITION 9,4;? #6;"BY"
20001 POSITION 0,6;? #6;"      LEN LINDSA
Y";POSITION 9,7;? #6;"AND";POSITION 6,8;
? #6;"TED KAHN"

```

```

20002 PRINT "THIS GAME REQUIRES A JOYSTI
CK."?:? "TO START, PRESS THE RED FIRE BUT
TON"
20004 PRINT "ON YOUR JOYSTICK"
20005 JS=99
20010 FOR X=0 TO 3
20020 IF STRIG(X)=0 THEN JS=X
20030 NEXT X
20040 IF JS>3 THEN 20010
20050 GRAPHICS 3
20099 RETURN
30000 PRINT "[CLEAR]THE OBJECT OF THIS G
AME IS TO FINISH"?:? "WITH ALL BUT THE CE
NTER BOX LIT UP"
30005 PRINT "(PRESS THE RED BUTTON TO CO
NTINUE)...":GOSUB 1000
30010 PRINT "[CLEAR]YOU SWITCH THE POSIT
IONS OF THE"?:? "BOXES BY MOVING YOUR JOY
STICK IN THE"
30020 PRINT "DIRECTION OF THE BOX YOU CH
OOSE AND"?:? "THEN PRESSING THE FIRE BUTT
ON...":GOSUB 1000
30030 PRINT "[CLEAR]YOU MAY ONLY CHOOSE
BOXES WHICH ARE"?:? "ALREADY LIT. CHOOSE
NG A LIT BOX "
30040 PRINT "WILL AFFECT THE STATE (LIT
OR NOT) OF"?:? "NEARBY BOXES...":GOSUB 1
000
30050 PRINT "[CLEAR]REMEMBER: MOVE THE
JOYSTICK UP TO "?:? "CHOOSE TOP CENTER, D
OWN FOR BOTTOM"
30060 PRINT "CENTER; LEFT, RIGHT, OR DIA
GONALLY.":GOSUB 1000
30070 PRINT "[CLEAR]WHEN YOU HAVE MADE Y
OUR CHOICE (KEEP"?:? "ONE HAND ON THE JOY
STICK), THEN PRESS"
30080 PRINT "THE FIRE BUTTON...":GOSUB 1
000
30090 PRINT "[CLEAR]IF YOU TURN OFF A CO
RNER"
30100 PRINT "ITS THREE NEIGHBORS WILL CH
ANGE TOO":GOSUB 1000
30110 PRINT "[CLEAR]IF YOU TURN OFF THE
MIDDLE OF A SIDE"

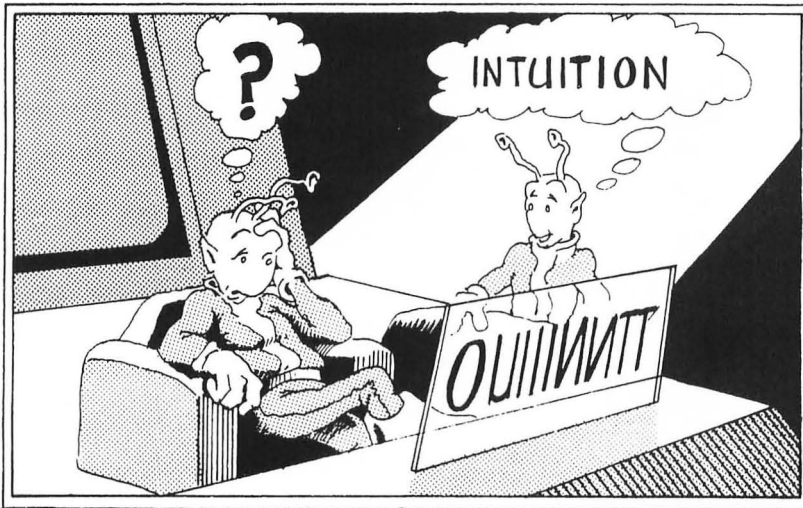
```

```
30120 PRINT "THE TWO NEIGHBORS ON THAT S  
IDE CHANGE":GOSUB 1000  
30130 PRINT "CLEAR IF YOU TURN OFF THE  
VERY CENTER BOX"  
30140 PRINT "THE MIDDLE BOX ON EACH SIDE  
CHANGES":GOSUB 1000  
30999 RETURN
```



## Chapter 5

# Word and Guessing Games



With your ATARI 400™ or ATARI 800™ Home Computer you can create dozens of challenging and amusing word and guessing games. They range from simple letter-guessing games to complex astrology programs that tell people about their personality and future. You can develop word drills, write silly stories, and create party games. This chapter will introduce you to all of these different types of play with language. It is important to remember that the ATARI computer handles words differently than numbers. If you've forgotten about DIM statements and the way in which memory space is reserved for words, review the sections in Part I that introduce DIM, A\$, A(X, X) and CLR.

We'll begin with a very simple word- or letter-guessing program that can be modified in dozens of ways.





## A SIMPLE WORD-GUESSING PROGRAM

This six-line program lets one person store a word or letter in the computer's memory, clears the screen immediately, and then provides an opportunity for someone else to guess the letter or word:

```
10 DIM A$(5),B$(5)
20 INPUT A$
30 PRINT "[CLEAR]"
40 INPUT B$
50 IF A$=B$ THEN PRINT "GOT IT":END
60 GOTO 30
```

Notice that line 10 DIMensions two variables that allow the computer to store words up to five letters long. Of course, you can DIMension A\$ and B\$ to much longer words if you care to. Also, note that line 30 clears the screen. That is done by typing the line number and PRINT"

```
30 PRINT"
```

and then pressing the ESCAPE key first and then the SHIFT and CLEAR keys together, and finally typing " at the end of the line. When you do this the following should appear on your screen:

```
30 PRINT "↑ "
```

Now here's a slightly dressed-up version of this simple two (or more) person game:

```
5 PRINT "PUT IN A 5 LETTER WORD FOR"
6 PRINT "A FRIEND TO GUESS"
10 DIM A$(5),B$(5)
20 INPUT A$
```



```

25 PRINT "[CLEAR]"
26 PRINT "TRY TO GUESS MY 5 LETTER WORD"
30 INPUT B$
40 IF A$=B$ THEN PRINT "GOOD YOU'VE GOT IT":END
50 GOTO 30

```

Here's a third version in which the computer gives you a hint about how close you are to guessing the target word. Lines 100 to 120 have the computer print each letter that appears in the same place in the word guessed and the word to be guessed. Thus,

if the target word was HEART  
and your guess was START  
the computer would print ART



```

5 REM BEGINNING OF WORD GUESSING GAME
10 DIM A$(5)
20 DIM B$(5)
30 PRINT "PICK A 5 LETTER WORD FOR YOUR FRIEND TO GUESS"
40 INPUT A$
50 PRINT "[CLEAR]"
60 PRINT "GUESS A 5 LETTER WORD"
70 INPUT B$
80 IF A$=B$ THEN PRINT "YOU GOT IT":END
90 PRINT "THESE LETTERS ARE IN YOUR WORD"
100 FOR X=1 TO 5
110 IF A$(X,X)=B$(X,X) THEN PRINT A$(X,X)
120 NEXT X
130 PRINT "TRY AGAIN"
140 GOTO 70

```

Several more complex changes in this third version of the program will make the hint more interesting.

```

ADD: 25 DIM C$(5)
      75 C$="00000"
      105 R=INT(RND(1)*5)+1
CHANGE:
      110 IF C$(R,R)= "0" THEN C$(R,R)=A$(X,X): PRINT A$(R,R)
      115 IF C$(R,R)<>"0" THEN 105

```

This will print the letters of the actual word in random (scrambled) order, regardless of what you input (unless it's the correct word, of course!).



## MEDITATION WITH WORD IMAGERY

This program was suggested by Dr. Dean Brown, a long-time friend of one of the authors, who believes that computers have great potential for opening up new dimensions of thought and perception. This particular program is based upon the very simple concept of presenting a word for a brief instant, resulting in the mind's creating a vivid picture or image. If this is done with a number of words within a brief time, you are able to link these images together to form even more complex images.

The selection of 4-letter words (lines 1000-1002) was not an arbitrary choice: Four-letter words may be unique in their ability to create these mental images. You may replace these words with a set of your own, as long as there are 10 words per line (line 10 also limits you to 4-letter words; you may change this, if you like, by changing the length of the dimension of WORD\$ in line 10).

```

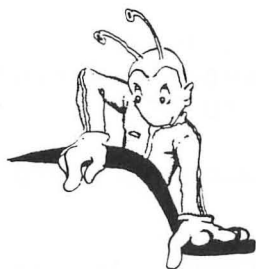
10 REM * < MEDITATIVE WORD IMAGERY >
20 REM *      suggested by Dean Brown
30 DIM WORD$(4)
40 GRAPHICS 1+16
50 POSITION 8,10
60 R=INT(RND(1)*2)+1000
70 RESTORE R
80 NOW=INT(RND(1)*10)+1
90 FOR I=1 TO NOW
100 READ WORD$
110 NEXT I
120 REM * FLASH WORD *
130 MAX=300+INT(RND(1)*600)+1:GOSUB 170
140 PRINT #6;WORD$
150 MAX=300:GOSUB 170
160 GOTO 40
170 REM PAUSE (WAIT) ROUTINE
180 FOR WW=1 TO MAX:NEXT WW
190 RETURN

```


200 END

1000 DATA WIND,RAIN,FIRE,SAND,SELF,FARM,  
TREE,LAKE,ROCK,LAND

1001 DATA FOOD,FACE,SLUM,PLUM,BALL,SALT,  
WALL,GAME,MATE,MIST



## A THREE-LETTER WORD DICTIONARY AND THREE-LETTER WORD GAMES

Many word games (such as Scrabble™, crossword puzzles, and Jotto™)  depend upon using some dictionary to determine a list of allowable words for the game. You can make dictionaries for your own computer word games and store them on cassette or disk. In that way, after the initial work of typing in the dictionary, you will have permanent dictionaries which you can use over and over in different games. One caution, however: be sure the line numbers in your dictionary are different than the line numbers in the other parts of your game programs.

In order to get you started, we've put together a dictionary of three-letter English words. The program used to store the dictionary will also search to determine if a given three-letter combination is actually a word, and it is very useful for a wide variety of word games. We suggest that you store lines 16499-19000 as a subroutine on tape or diskette, using LIST "C:" or LIST "D:WORDS" (or whatever file name you wish). The program can then be merged with your actual game when you are ready, using ENTER "C:" or ENTER "D:WORDS". You may also wish to save the recognition routine (lines 0-320) as well, either as a separate routine or in combination with the dictionary.

```

10 REM * <<< 3 LETTER DICTIONARY >>>
20 REM *   bypassing string arrays
30 REM *   Copyright 1980 by
40 REM *   Len Lindsay
50 DIM T$(3),W$(3)
60 TRAP 60:PRINT "[CLEAR]ENTER A 3 LETTE
R WORD"
70 X=X+1:SETCOLOR 2,X,2

```



Scrabble™ is a registered trademark of Selchow & Righter Company.

```

80 INPUT W$
90 IF LEN(W$)<>3 THEN 60
100 RESTORE 10000+100*ASC(W$)
110 FOR TEST=1 TO 45
120 READ T$
130 IF T$=W$ THEN POP :GOTO 160
140 NEXT TEST
150 GOTO 60
160 PRINT "CORRECT - TRY AGAIN?"
170 INPUT T$
180 IF T$(1,1)<>"N" THEN 60
16490 REM * BEGINNING OF DATA FOR
16492 REM * THE 3 LETTER DICTIONARY
16494 REM *
16496 REM * Add your own words
16498 REM * on the appropriate lines
16500 DATA ACE,ACT,ADD,ADO,ADZ,AFT,AGE,A
GO,AHA,AID,AIL,AIM,AIR,ALB,ALE,ALL,AND,A
NT,ANY,APE,ARC,ARE,ARK,ARM,ART,ASH
16510 DATA AUK,AWE,AWL,AXE,AYE
16600 DATA BAA,BAD,BAG,BAH,BAN,BAR,BAT,B
AY,BED,BEE,BEG,BET,BEY,BIB,BID,BIG,BIN,B
IT,BOA,BOB,BOO,BOP,BOW,BOX,BOY,BRA
16610 DATA BUD,BUG,BUM,BUN,BUR,BUS,BUT,B
UY
16700 DATA CAB,CAD,CAL,CAM,CAN,CAP,CAR,C
AT,CAW,CHI,COD,COG,CON,COO,COP,COT,COW,C
OY,CRY,CUB,CUD,CUE,CUM,CUP,CUR,CUT
16800 DATA DAB,DAD,DAM,DAW,DAY,DDT,DEN,D
EW,DID,DIE,DIG,DIM,DIN,DIP,DNA,DOE,DOG,D
ON,DOT,DRY,DUB,DUD,DUE,DUG,DUN,DUO
16810 DATA DYE
16900 DATA EAR,EAT,EBB,EEL,EFT,EGG,EGO,E
KE,EKG,ELF,ELK,ELL,ELM,EMU,END,EON,ERA,E
RE,ERG,ERR,ESP,ETA,EVE,EWE,EYE
17000 DATA FAD,FAG,FAN,FAR,FAT,FAY,FEE,F
EN,FEW,FEY,FEZ,FIB,FIE,FIG,FIN,FIR,FIT,F
IX,FLU,FLY,FOB,FOE,FOG,FOP,FOR,FOX
17010 DATA FRO,FRY,FUN,FUR
17100 DATA GAB,GAG,GAL,GAM,GAP,GAR,GAS,G
AT,GAY,GEE,GEL,GEM,GET,GIG,GIN,GNU,GOB,G
OD,GOO,GOT,GUM,GUN,GUT,GUY,GYP
17200 DATA HAD,HAG,HAH,HAM,HAT,HAW,HAY,H
EM,HEN,HER,HEW,HEX,HEY,HID,HIE,HIM,HIP,H

```

IS, HIT, HMM, HOB, HOD, HOE, HOG, HON, HOP  
 17210 DATA HOT, HOW, HUB, HUE, HUG, HUH, HUM, H  
 UN, HUT  
 17300 DATA ICE, ICY, ILK, ILL, IMP, INK, ION, I  
 OU, IRE, IRK, ITS, IVY  
 17400 DATA JAB, JAG, JAM, JAR, JAW, JAY, JET, J  
 EW, JIB, JIG, JOB, JOG, JOT, JOY, JUG, JUT  
 17500 DATA KEG, KEN, KEY, KID, KIN, KIP, KIT  
 17600 DATA LAB, LAC, LAD, LAG, LAM, LAP, LAW, L  
 AX, LAY, LED, LEE, LEG, LEI, LEK, LET, LEU, LID, L  
 IE, LIP, LIT, LOG, LOT, LOW, LOX, LSD, LUG  
 17610 DATA LYE  
 17700 DATA MAD, MAN, MAP, MAR, MAT, MAW, MAY, M  
 EN, MET, MEW, MID, MIL, MIN, MIX, MOA, MOB, MOD, M  
 OM, MOO, MOP, MOW, MUD, MUG, MUM  
 17800 DATA NAB, NAG, NAF, NAW, NAY, NEE, NET, N  
 EW, NIB, NIL, NIP, NIT, NIX, NOD, NOR, NOT, NOW, N  
 TH, NUE, NUT  
 17900 DATA OAF, OAK, OAR, OAT, OBI, ODD, ODE, O  
 FF, OFT, OHM, OIL, OLD, ONE, OOH, OFE, OPT, ORB, O  
 RE, OUR, OUT, OVA, OWE, OWL, OWN  
 18000 DATA PAD, PAL, PAN, PAP, PAR, PAT, PAW, P  
 AY, PEA, PEG, PEN, PEP, PER, PET, PHI, PIE, PIG, P  
 IN, PIP, PIT, PLY, POD, POI, POP, POT, POW  
 18010 DATA POX, PRO, PRY, PSI, PUB, PUG, FUN, F  
 UP, PUT  
 18200 DATA RAG, RAM, RAN, RAP, RAT, RAW, RAY, R  
 ED, REV, RHO, HIB, RID, RIG, RIM, RIP, RNA, ROB, R  
 OC, ROD, ROE, ROT, ROW, RUB, RUE, RUM, RUN  
 18210 DATA RUT, RYE  
 18300 DATA SAC, SAD, SAG, SAL, SAP, SAT, SAW, S  
 AX, SAY, SEA, SEC, SEE, SET, SEW, SEX, SHE, SHH, S  
 HY, SIC, SIN, SIP, SIR, SIT, SIX, SKI, SKY  
 18310 DATA SLY, SOB, SOD, SOL, SON, SOP, SOS, S  
 OT, SOW, SOX, SOY, SPA, SPY, STY, SUB, SUE, SUM, S  
 UN, SUP  
 18400 DATA TAB, TAG, GAI, TAN, TAP, TAR, TAT, T  
 AU, TAW, TAX, TEA, TEE, TEN, THE, TIC, TIE, TIN, T  
 IP, TIT, TOE, TOM, TON, TOO, TOP, TOT, TOW  
 18410 DATA TOY, TRY, TUB, TUG, TUN, TUX, TWI, T  
 WO  
 18500 DATA UGH, UHF, URN, USA, USE  
 18600 DATA VAN, VAT, VET, VEX, VHF, VIA, VIE, V  
 IF

18700 DATA WAD,WAG,WAN,WAR,WAS,WAX,WAY,W  
EE,WED,WEE,WEN,WET,WHO,WHY,WIG,WIN,WIT,W  
OE,WON,WOO,WOW  
18900 DATA YAK,YAM,YAP,YAW,YEA,YEN,YES,Y  
ET,YEW,YIP,YON,YOU  
19000 DATA ZED,ZIP,ZOO,ZZZ



## USING YOUR DICTIONARY

Once you have stored the three-letter-word dictionary on tape (use LIST "C:") or on disk (use LIST "D:NAME" where NAME is whatever you want to call it in under eight letters), you may use it in any number of word games. The procedure is as follows:

1. Develop your game and type it into the computer, being sure that your last line number does not exceed 9999.

2. Load the dictionary in by using

ENTER "C:" (from tape) or  
ENTER "D:NAME" (from disk)

This will "overlay" the dictionary so that it does not wipe out your previous program.

3. Debug your new program and make any modifications you wish. Now you can store the entire program as one complete entity.

For example, here's a very simple program which just lists what words are available in the dictionary. Try it and see how easy it can be to "compose" word games in this fashion:

```
1 REM *** LIST OF ALL 3-LETTER WORDS *
2 REM *** AVAILABLE IN DICTIONARY ***
3 REM *** LEN LINDSAY, 2/28/80 ***
5 DIM W$(3)
10 POKE 82,0
20 PRINT "[CLEAR] HERE ARE THE THREE-
  LETTER WORDS I KNOW:"
30 READ W$
40 IF W$="ZZZ" THEN END
50 PRINT W$," ";
60 GOTO 30
9999 REM *** 3-LETTER WORDS FOLLOW**
```

Here's another three-letter-word game that uses the dictionary. This game is based on Lewis Carroll's famous word game, *Doublets*. *Doublets* are a form of word transformations. You begin with two words, for example RUN and SIT, and try to transform RUN into SIT. The rules for transformation are (1) that you can only change one letter at a time, (2) that each change must result in a legitimate word, and (3) that letters cannot be switched around. Here's one way to turn RUN to SIT in three steps:

RUN  
SUN  
SIN  
SIT

Here's a program for three-letter doublets using SUN and RED as the target words and utilizing the dictionary to determine whether words used in the transformation are allowable or not:

```

10 REM <<< SUNTORED >>>
20 REM LEN LINDSAY 1980
22 REM *      *** IMPORTANT ***
24 REM *  YOU MUST ADD THE 3 LETTER
26 REM *  DICTIONARY DATA STATEMENTS
28 REM *  STARTING AT LINE 16490
30 DIM S$(3),E$(3),W$(3),T$(3),Q$(3)
40 GRAPHICS 1
50 PRINT "DO YOU WANT TO USE YOUR OWN WO
RDS";
60 INPUT T$
70 IF T$(1,1)="Y" THEN GOSUB 9000:GOTO 1
00
80 S$="SUN":REM THE STARTING WORD
90 E$="RED":REM THE ENDING WORD
100 GRAPHICS 1
110 PRINT #6;"CHANGE ";S$;" TO ";E$
120 LINE=2:REM INIT START LINE TO PRINT
WORDS ON
130 W$=S$
140 POSITION 1,LINE
150 PRINT #6;W$
160 LINE=LINE+1
170 IF W$=E$ THEN 300:REM WIN
180 IF LINE>18 THEN PRINT #6;"YOU SEEM L
OST":GOTO 8000
190 PRINT "WHAT WORD COMES NEXT?"
200 INPUT T$
210 RESTORE 10000+100*ASC(T$)
220 FOR TEST=1 TO 45
230 READ Q$
240 IF Q$="ZZZ" THEN POP :GOTO 270
250 IF Q$=T$ THEN POP :GOTO 400
260 NEXT TEST

```

```

270 PRINT "I DON'T KNOW THAT WORD - ANOT
HER?"
299 GOTO 200
300 FOR L1=40 TO 0 STEP -8
310 FOR L2=40 TO L1 STEP -1:SOUND 0,L2,1
0,10:NEXT L2
320 NEXT L1:SOUND 0,0,0,0
330 POSITION 0,19:PRINT #6;"WINNER WITH
";LINE-3;" WORDS"
399 GOTO 8000
400 MATCH=0
410 FOR TEST=1 TO 3
420 IF T$(TEST,TEST)=W$(TEST,TEST) THEN
MATCH=MATCH+1
430 NEXT TEST
440 IF MATCH=2 THEN W$=T$:GOTO 140
499 PRINT "EXACTLY 1 LETTER MUST CHANGE"
;GOTO 200
8000 PRINT "WOULD YOU LIKE TO TRY AGAIN"
;
8010 INPUT T$
8020 IF T$(1,1)="N" THEN END
8030 PRINT "WITH THE SAME WORDS";
8040 INPUT T$
8050 IF T$(1,1)="Y" THEN GOTO 100
8060 PRINT "DO YOU WANT TO USE YOUR OWN
WORDS";
8070 INPUT T$
8080 IF T$(1,1)="Y" THEN GOSUB 9000:GOTO
100
8099 GOTO 100
8199 GOTO 100
9000 PRINT "[CLEAR]"
9010 PRINT "WHAT IS THE STARTING WORD";
9020 INPUT T$
9030 GOSUB 9500
9040 IF T$="***" THEN PRINT "ILLEGAL WOR
D - TRY AGAIN":GOTO 9010
9050 S$=T$
9110 PRINT "WHAT IS THE ENDING WORD";
9120 INPUT T$
9130 GOSUB 9500
9140 IF T$="***" THEN PRINT "ILLEGAL WOR
D - TRY AGAIN":GOTO 9110

```

```

9150 IF S$=T$ THEN PRINT "ENDING WORD CA
N'T BE THE SAME":GOTO 9110
9160 E$=T$
9200 RETURN
9500 RESTORE 10000+100*ASC(T$)
9520 FOR TEST=1 TO 45
9550 READ Q$
9560 IF Q$="ZZZ" THEN POP :T$="***":GOTO
9599
9570 IF Q$=T$ THEN POP :GOTO 9599
9580 NEXT TEST
9590 T$="***"
9599 RETURN
10000 REM * ADD 3 LETTER DICTIONARY
10010 REM * DATA STATEMENTS TO THIS
10020 REM * PART OF THE PROGRAM -
10030 REM * LINE 16490 TO 19000

```

You may want to get together with friends who have ATARI Home Computers and make your own four- or five-letter-word dictionary. Obviously the longer the words, the more complex and interesting the games you can play.

Here are some game ideas that can be easily programmed and use the three-letter-word dictionary:

1. Set up a game that begins with a single consonant. The goal is to see who can generate the most three-letter words that begin with that letter. Another goal can be to see who can come up with the most three-letter words that have that letter in any position.
2. Similar game challenges can be created with vowels. Pick a vowel. How many three-letter words can be generated that contain that vowel? Who comes up with the most words?
3. Pick any two or three words from the computer dictionary at random. How many other three-letter words can you make with the words you got? If you could pick any words you wanted, which three-letter words would you pick to generate the most words?
4. Given five (or any other number larger than four) letters, at least one of which is a vowel, how many three-letter words can you generate? This challenge can be done by one person as a solitaire game or by any number of people as a contest. A programming challenge is to build a scoreboard into the game so that each person gets a point for every correct three-letter word.

Here's a note about identifying three-letter words from Len Lindsay. It's meant for advanced programmers:

## THREE-LETTER-WORD IDENTIFIER FOR ATARI

"Here is a quick but neat program for ATARI users. You may already know that ATARI computers do not allow string arrays. But this restriction does not rule out word games and the like.

"ATARI allows DATA to be RESTORED beginning at any line number, even a variable line number under software control. I use this feature to avoid using a word array. Instead I use the DATA statements as my array of sorts.

"I also rely on using the ASC keyword of ATARI BASIC, which will tell me the ATASCII character code of the first letter in any string.

"In order to identify all three-letter words, the computer must have a complete list to use as its guide. I have used DYMAX's list of all legal three-letter words from the American Heritage Elementary School Dictionary. These are in DATA statements 16500 to 19000.

"One last fact is used by my program, allowing no more than one second to identify a correct three-letter word. There are 45 words that begin with the letter S, more than any other letter. Thus my loop beginning in line 200 need only check 45 entries at the maximum, which can be done in under a second.

"The following is the program that will ask for you to input any word. It will then tell you if the first three letters of your input is a valid three-letter word. Very simple, but a good basis for future expansion."

```

0 REM (C) 1980 LEN LINDSAY
10 DIM W$(3), T$(3) : REM THIS IS ALL THE SPACE WE WILL
   NEED TO RESERVE
100 TRAP 100 : PRINT "[CLEAR] ENTER A
   3 LETTER WORD" : REM SET OUR TRAP
105 X=X+1 : SETCOLOR 2,X,2 : REM CHANGE THE BACK-
   GROUND COLOR
110 INPUT W$
120 IF LEN(W$) < > 3 THEN 100 : REM NOT A THREE LETTER
   WORD
130 RESTORE 10000+100*ASC(W$) : REM RESTORE DATA AT
   FIRST LETTERS LINE
200 FOR TEST = 1 TO 45 : REM MAXIMUM NUMBER OF
   WORDS TO CHECK
210 READ T$
220 IF T$=W$ THEN POP : GOTO 300 : REM CORRECT & POP
   OUT OF LOOP
230 IF T$="ZZZ" THEN 100 : REM END OF WORDS AND NO
   MATCH
240 NEXT TEST

```

```
250 GOTO 100 : REM NO MATCH FOUND
300 PRINT "CORRECT - TRY AGAIN?"
310 INPUT T$
320 IF T$ <> "N" THEN 100 : REM ANSWER "N" TO STOP
16499 REM DATA STATEMENTS FOLLOW
```

## SERIES

Here is a game in which the object is to link words together using their first and last letters. One player types in a three-letter word and the other player must follow with a word whose *first* letter is the same as the *last* letter of the previous word. For example, here is a typical sequence:

PEN  
 NOT  
 TIN  
 etc.

Remember to ENTER your stored three-letter-word dictionary after you type in this program before you begin play.

```

10 REM <<< SERIES >>>
20 REM LEN LINDSAY 1980
22 REM *      *** IMPORTANT ***
24 REM *  YOU MUST ADD THE 3 LETTER
26 REM *  DICTIONARY DATA STATEMENTS
28 REM *  STARTING AT LINE 16490
30 NUMBER=554
40 DIM PICKED(NUMBER),PLAYER1$(8),PLAYER
2$(8)
50 DIM IN$(15),LASTW$(14),STARTL$(1)
60 DIM TEST$(14),S(1),P(1),T(1)
70 GAMEOVER=9000
80 OPTION=3010:GOSUB OPTION-10
90 GOSUB 2000:REM GRAPHICS & COLORS
100 PRINT #6;"3 LETTER WORD SERIES"
110 POSITION 6,8:PRINT #6;"TO BEGIN"
120 POSITION 5,9:PRINT #6;"HIT start"
130 POSITION 2,2:PRINT #6;"FOR INSTRUCTI
ONS"
140 POSITION 5,3:PRINT #6;"HIT option"
150 POSITION 6,11:PRINT #6;" ATARI "
160 KEY=PEEK(53279)
170 IF KEY=3 THEN GOSUB 1000:GOTO 90
180 IF KEY=5 THEN GOSUB OPTION:OPTION=OP
TION+10:GOSUB 4020:IF OPTION>3030 THEN O
PTION=3000
190 IF KEY=6 THEN 250
200 POSITION 0,5:PRINT #6;PLAYER1$;" VS
";PLAYER2$;"HIT select TO CHANGE"

```

```

210 SETCOLOR 3,0,RND(1)*256
220 FOR DELAY=1 TO 50:NEXT DELAY
230 SETCOLOR 2,0,PN:PN=PN+2
240 GOTO 160
250 GRAPHICS 2+16:PRINT #6;"WAIT"
260 FOR A=1 TO NUMBER:PICKED(A)=0:NEXT A
:REM INIT
270 S(0)=0:S(1)=0
280 T(0)=0:T(1)=0
290 COUNT=0
300 PLAYER=1
310 TEST$=PLAYER2$:GOSUB 5000:PLAYER2$=T
EST$
320 PASS=0
330 PLAYER=1-PLAYER
340 POSITION 0,10:GOSUB 6900
350 POSITION 0,11:GOSUB 6900
360 IF PASS>1 THEN GOTO GAMEOVER
370 COUNT=COUNT+1
380 GOSUB 6000
390 POSITION 1,3:PRINT #6;"LAST WORD WAS
:"
400 POSITION 1,4:GOSUB 6900:REM ERASE LI
NE
410 IF LEN(LASTW$)>1 THEN IF LASTW$(1,1)
<="Z" THEN TEST$=LASTW$:GOSUB 5000:POSIT
ION 1,4:PRINT #6;TEST$
420 POSITION 1,6:PRINT #6;"NEXT WORD MUS
T"
430 POSITION 3,7:PRINT #6;"START WITH: "
;
440 STARTL$=CHR$(INT(RND(1)*26)+65)
450 IF LEN(LASTW$)>1 THEN STARTL$=LASTW$
(LEN(LASTW$))
460 PRINT #6;CHR$(ASC(STARTL$)+160)
470 POSITION 1,8
480 IF P(PLAYER)=0 THEN PRINT #6;"ENTER
YOUR WORD"
490 IF P(PLAYER)=1 THEN PRINT #6;"MY WOR
D IS: "
500 GOSUB 6900
510 IF P(PLAYER)=1 THEN GOSUB 7000:GOTO
670:REM COMPUTER TURN
520 CHAR=1

```



```

530 IN$=""
540 POSITION 1,9:PRINT #6;IN$
550 OPEN #1,4,0,"K"
560 POKE 764,255:REM CLEAR BUFFER
570 IF PEEK(764)=255 THEN SETCOLOR 3,0,R
ND(1)*256:GOTO 570
580 GET #1,IN
590 CLOSE #1
600 IF IN=126 AND CHAR>1 THEN CHAR=CHAR-
1:IN$(CHAR,CHAR)=" ":GOTO 540:REM DELETE
HIT
610 IF IN=155 THEN 670:REM RETURN HIT
620 IF IN<65 AND IN<>32 OR IN>90 THEN 55
0:REM NOT ALPHABETIC
630 IN$(CHAR,CHAR)=CHR$(IN)
640 CHAR=CHAR+1
650 IF CHAR>15 THEN CHAR=15
660 GOTO 540
670 IF LEN(IN$)<1 THEN 500
680 IF CHAR>1 THEN IN$=IN$(1,CHAR-1):REM
MINUS TRAILING SPACES
690 IF IN$="PASS" THEN PASS=PASS+1:GOTO
330
700 IF ASC(IN$)<>ASC(STARTL$) THEN PASS=
PASS+1:PRINT #6;"1st letter's not ";STAR
TL$:GOTO 900
710 RESTORE
720 FOR TEST=1 TO NUMBER
730 READ TEST$
740 IF TEST$=IN$ THEN POP :GOTO 800
745 IF TEST$(1,1)>IN$(1,1) THEN TEST=NUM
BER
750 NEXT TEST
760 PRINT #6;"not a legal word"
770 GOTO 900:REM CONTINUE
800 IF PICKED(TEST)>0 THEN PRINT #6;"tha
t word is used hit start":GOSUB 4000:G
OSUB 6000:GOTO 340
810 PICKED(TEST)=PLAYER+1
820 PRINT #6;"that word works"
830 LASTW$=IN$
840 S(PLAYER)=S(PLAYER)+1
850 T(PLAYER)=T(PLAYER)+LEN(IN$)
900 PRINT #6;"HIT start";

```

```

910 GOSUB 6050
920 GOSUB 4000:IF KEY<>6 THEN 920
930 GOTO 320
999 END
1000 GRAPHICS 1+16:GOSUB 2010
1010 PRINT #6;"THREE LETTER WORDS"
1020 PRINT #6;"ARE THE LEGAL WORDS"
1030 PRINT #6;"THE LAST LETTER OF"
1040 PRINT #6;"THE PREVIOUS WORD"
1050 PRINT #6;"MUST BE THE FIRST"
1060 PRINT #6;"LETTER OF THE NEXT"
1070 PRINT #6;"WORD. NO REPEATS"
1075 PRINT #6
1080 PRINT #6;"type 'PASS' if you"
1090 PRINT #6;"can not move."
1095 PRINT #6
1100 PRINT #6;"AN ILLEGAL MOVE IS"
1110 PRINT #6;"TREATED AS PASSING"
1120 PRINT #6;"YOUR TURN"
1125 PRINT #6
1130 PRINT #6;"THE GAME IS OVER"
1140 PRINT #6;"IF BOTH PLAYERS PASS"
1200 PRINT #6
1900 PRINT #6;"HIT start"
1990 GOSUB 4000:IF KEY<>6 THEN 1990
1999 RETURN
2000 GRAPHICS 2+16
2010 SETCOLOR 0,2,8:REM CAPS - REG TEXT
2020 SETCOLOR 1,12,10:REM lower LUMIN -
OPTIONS IN STARTER
2030 SETCOLOR 2,8,12:REM CAPS WINDOW - T
ITLE IN STARTER
2040 SETCOLOR 3,4,6:REM lower - NAMES OF
THE SPECIAL KEYS - VIBRATES NAMES
2050 SETCOLOR 4,0,0:REM BACKGROUND
2099 RETURN
3000 PLAYER1$="human" ":PLAYER2$="compu
ter":P(0)=0:P(1)=1
3009 RETURN
3010 PLAYER1$="computer":P(0)=1
3019 RETURN
3020 PLAYER2$="human" ":P(1)=0
3029 RETURN
3030 PLAYER1$="human" ":P(0)=0

```

```

3039 RETURN
4000 IF PEEK(53279)<>7 THEN 4000
4010 KEY=PEEK(53279):IF KEY=7 THEN SETCO
LOR 3,0,RND(1)*255:GOTO 4010
4020 IF PEEK(53279)<>7 THEN 4020
4099 GOSUB 2010:RETURN
5000 FOR A=1 TO LEN(TEST$)
5010 TEST$(A,A)=CHR$(ASC(TEST$(A,A))+128
)
5020 NEXT A
5099 RETURN
6000 GOSUB 2010
6010 TEST$=PLAYER1$:GOSUB 5000:PLAYER1$=
TEST$
6020 TEST$=PLAYER2$:GOSUB 5000:PLAYER2$=
TEST$
6030 POSITION 0,0
6040 PRINT #6;PLAYER1$;" VS ";PLAYER2$
6050 POSITION 1,1:PRINT #6;S(0);";";T(0)
6060 POSITION 13,1:PRINT #6;S(1);";";T(1
)
6900 PRINT #6;" ";
6999 RETURN
7000 RESTORE
7010 FOR TEST=1 TO NUMBER
7020 SETCOLOR 3,3,RND(1)*256
7030 READ IN$
7040 IF ASC(IN$)=ASC(STARTL$) AND PICKED
(TEST)=0 THEN POP :GOTO 7100
7050 NEXT TEST
7060 IN$="PASS"
7070 POSITION 1,10
7080 PRINT #6;"I PASS! HIT start"
7090 GOSUB 4000:IF KEY<>6 THEN 7090
7100 CHAR=LEN(IN$)+1
7110 POSITION 1,9:PRINT #6;IN$
7199 RETURN
9000 POSITION 1,10
9010 PRINT #6;"[CLEAR]"
9020 FOR LL=20 TO 0 STEP -10
9030 FOR L=50 TO LL STEP -1.5:SOUND 0,L,
10,10:NEXT L:SOUND 0,0,0,0
9040 NEXT LL
9050 GOSUB 6000

```

```
9060 PRINT #6
9070 POSITION 3,10:PRINT #6;"GAME IS OVE
R"
9080 POSITION 4,11:PRINT #6;"HIT start";
9090 GOSUB 4000:IF KEY<>6 THEN 9090
9099 RUN
10000 REM * ADD 3 LETTER DICTIONARY
10010 REM * DATA STATEMENTS TO THIS
10020 REM * PART OF THE PROGRAM -
10030 REM * LINES 16490 TO 19000
```

## SECRET CODES

Computers are used throughout the world to create and solve secret codes. You can use your ATARI Home Computer to make up your own codes and code and decode messages.

The simplest code consists of substituting one letter for another. Here's a game program which will allow you to make your own letter substitutions. After you enter the code, the computer will test you by flashing a letter and asking you the code for that letter. This provides a good method for memorizing a code you create:



```

5 REM *** CODE83 ALPHABET CODE GUESSING
  GAME
6 REM (C) BY TED M. KAHN, 1980
10 DIM A$(26),B$(26),G$(1),X$(1)
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 PRINT "[CLEAR] INPUT YOUR SECRET CODE FOR EACH
  PRINT "LETTER OF THE ALPHABET. USE EACH"
32 PRINT "LETTER ONLY ONCE!"
40 FOR I=1 TO 26
45 PRINT "THE CODE FOR ";A$(I,I)," = ";
50 INPUT X$:IF I=1 THEN GOTO 58
52 FOR J=1 TO (I-1)
54 IF B$(J, J)=X$ THEN POP:PRINT "YOU ALREADY USED
  THAT ONE":GOTO 45
56 NEXT J
58 B$(I, I)=X$
60 NEXT I
70 REM *** NOW FOR THE GUESSING GAME*
80 PRINT "[CLEAR]":REM *** CLEAR SCREEN***
90 LET R=INT(RND(1)*26)+1
100 PRINT "WHAT'S THE CODE FOR ";A$(R,R);
110 INPUT G$
120 IF G$=B$(R,R) THEN PRINT "GREAT! YOU GOT IT!":
  GOTO 90
130 PRINT "NO, THE CODE IS";B$(R,R)
140 GOTO 90

```

By the way, if you let each letter stand for itself you have a program that can be used to help a child who is just learning to read to memorize the alphabet.

Your ATARI Home Computer has graphic symbols as well as letters. You get graphic symbols by pressing the [CTRL] key and a letter simultaneously. There is a chart of all the graphics symbols in your ATARI

BASIC Reference Manual. Here is a code program in which each letter is represented by the GRAPHIC symbol on its key. For example [H] is coded by ▲ and [T] by ●. This program will test you on the location of the GRAPHIC symbols on your keyboard:

```

10 REM *   <<< GRAPHICS CODES >>>
20 REM * For learning which keys
30 REM * correspond to which control
40 REM * graphics characters
50 REM * (C) by Ted M. Kahn, 1980
60 DIM A$(26),B$(26),G$(1)
70 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
80 REM * LINE 90 CONTAINS [CTRL] GRAPHIC
  S CODES OF EACH LETTER OF THE ALPHABET,
  IN ORDER
90 B$="[A][B][C][D][E][F][G][H][I][J][K]
  [L][M][N][O][P][Q][R][S][T][U][V][W][X][
  Y][Z]"
100 REM * NOW FOR THE GUESSING GAME*
110 PRINT "[CLEAR]"
120 LET R=INT(RND(1)*26)+1
130 PRINT :PRINT "WHAT LETTER GOES WITH
  ";B$(R,R);
140 INPUT G$
150 IF G$=A$(R,R) THEN PRINT "GREAT! YOU
  GOT IT!":GOTO 120
160 PRINT "NO, THE LETTER FOR ";B$(R,R);
  " IS ";A$(R,R)
170 GOTO 120

```

Here's a complete encoding and decoding program. You can enter your own code and get a coded message out. You can also type a coded message in, and the computer will decode it for you. (If you want the messages printed on your ATARI 820, ATARI 822, or ATARI 825 printer, replace the word PRINT with LPRINT in lines 450, 470, 630, and 640.) Also, don't forget the space at the end of the string ALPHA\$ in line 80. It's very important!

```

10 REM * <<< ENCODING & DECODING >>>
20 REM *   (C) 1981 by Ted M. Kahn
30 REM * Alphabetic codes only. All
40 REM * other characters will remain
50 REM * as is

```

```

60 REM
70 DIM ALPHA$(27),CODE$(27),MESSAGE$(100)
,CMESSAGE$(100),A$(1)
80 LET ALPHA$="ABCDEFGHIJKLMNOPQRSTUVWXYZ "
90 REM
100 REM * ENCODING
110 REM
120 PRINT "[CLEAR]TYPE IN YOUR ENCODING
SYSTEM,":PRINT "ONE LETTER PER LINE."
130 PRINT "(FOR EXAMPLE: 'A=E' MEANS 'A
WILL BE REPLACED BY E')":PRINT
140 FOR I=1 TO 26
150 PRINT ALPHA$(I,I);"=";
160 INPUT A$:IF I=1 THEN 210
170 FOR J=1 TO I-1
180 IF CODE$(J,J)<>A$ THEN 200
190 PRINT "YOU'VE ALREADY USED THAT LETT
ER":PRINT :POP :GOTO 150
200 NEXT J
210 CODE$(I,I)=A$
220 NEXT I
230 CODE$(27,27)=" "
240 PRINT :PRINT "DO YOU WANT TO ENCODE
OR DECODE":PRINT "A MESSAGE";
250 INPUT A$
260 IF A$="E" THEN GOSUB 290
270 IF A$="D" THEN GOSUB 490
280 GOTO 240
290 REM
300 REM * ENCODE A MESSAGE
310 REM
320 PRINT "[CLEAR]"
330 MESSAGE$="":CMESSAGE$=""
340 PRINT "TYPE IN THE MESSAGE YOU WANT
TO ENCODE"
350 PRINT "(UP TO A MAXIMUM OF 100 CHARA
CTERS)":PRINT
360 INPUT MESSAGE$
370 L=LEN(MESSAGE$)
380 FOR I=1 TO L
390 FOR J=1 TO 27
400 IF MESSAGE$(I,I)<>ALPHA$(J,J) THEN G
OTO 420

```

```

410 CMESSAGE$(I,I)=CODE$(J,J):POP :GOTO
440
420 NEXT J
430 IF J>27 THEN CMESSAGE$(I,I)=MESSAGE$
(I,I)
440 NEXT I
450 PRINT "YOUR ENCODED MESSAGE IS:"
460 REM * REPLACE 'PRINT' IN LINES 450 &
470 WITH 'LPRINT' FOR PRINTOUT
470 PRINT CMESSAGE$
480 RETURN
490 REM
500 REM * DECODE A MESSAGE
510 REM
520 CMESSAGE$="":MESSAGE$=""
530 PRINT :PRINT "TYPE IN THE ENCODED ME
SSAGE":PRINT
540 INPUT CMESSAGE$
550 L=LEN(CMESSAGE$)
560 FOR I=1 TO L
570 FOR J=1 TO 27
580 IF CMESSAGE$(I,I)<>CODE$(J,J) THEN G
OTO 600
590 MESSAGE$(I,I)=ALPHA$(J,J):POP :GOTO
620
600 NEXT J
610 IF J>27 THEN MESSAGE$(I,I)=CMESSAGE$
(I,I)
620 NEXT I
630 PRINT "[CLEAR]YOUR DECODED MESSAGE I
S:"
640 PRINT MESSAGE$:RETURN
650 REM * REPLACE 'PRINT' IN 630 & 640
660 REM * WITH 'LPRINT' FOR PRINTOUT

```



## ANAGRAMS

Anagrams are ways of coding words by mixing up their letters. UNS is an anagram of SUN and MSANAGRA is an anagram of ANAGRAMS. Here are several programs that you can dress up to make fancy anagram games.

In the first program you anagram a word, put it in the computer's memory, and have other people try to guess the original word.

```

10 REM *      <<< ANAGRAM 1 >>>
20 REM *      (C) 1981 by Ted M. Kahn
30 REM *      longest word is 30 letters
40 DIM WORD$(30),SWITCH$(30),ANAGRAM$(30)
50 ANAGRAM$=""
60 PRINT :PRINT "TYPE IN THE WORD YOU WIS
SH TO ANAGRAM"
70 INPUT WORD$:L=LEN(WORD$)
80 GOSUB 120
90 PRINT "THE ANAGRAM IS:"
100 PRINT ANAGRAM$
110 GOTO 50
120 REM *      ANAGRAM SUBROUTINE
130 FOR I=1 TO L
140 SWITCH$(I,I)="0"
150 NEXT I
160 FOR J=1 TO L
170 X=INT(RND(1)*L)+1
180 IF SWITCH$(X,X)="1" THEN GOTO 170
190 SWITCH$(X,X)="1"
200 ANAGRAM$(X,X)=WORD$(J,J)
210 NEXT J
220 IF ANAGRAM$=WORD$ THEN GOTO 120
230 RETURN

```

This second program provides a general routine for scrambling the letters of a word. You provide the word, and the computer scrambles it.

```

10 REM GENERAL ANAGRAM PROGRAM
15 PRINT "[CLEAR]"
20 DIM WORD$(40),ANA$(40),COUNT(40)
30 PRINT "TYPE IN YOUR WORD";
35 TRAP 30

```

```

40 INPUT WORD$:ANA$="":L=LEN(WORD$)
42 REM CLEAR LETTER-REPEAT COUNTER
45 FOR X=1 TO L:COUNT(X)=0:NEXT X
50 FOR I=1 TO L:REM SCRAMBLE LETTERS
60 R=INT(RND(0)*L)+1:REM SHIFT LETTER
70 IF COUNT(R)=1 THEN 60
75 COUNT(R)=1
80 ANA$(R,R)=WORD$(I,I)
90 NEXT I
92 REM VERIFY ANAGRAM
95 IF WORD$=ANA$ THEN ANA$="":GOTO 45
100 PRINT "THE ANAGRAM IS: ";ANA$
105 PRINT
110 GOTO 30

```

Finally, here is a program that provides anagrams and a game challenge. It's designed for two players, one who selects the word to be guessed and the other who tries to guess it from its anagram.

```

10 REM * << ANAGRAM GUESSING GAME >>
20 REM * (C) 1981 by Ted M. Kahn
30 REM * longest word is 30 letters
40 DIM WORD$(30),SWITCH$(30),ANAGRAM$(30),GUESS$(30)
50 GRAPHICS 0
60 ANAGRAM$=""
70 PRINT :PRINT "TYPE IN THE WORD YOU WANT YOUR FRIEND":PRINT "TO GUESS:"
80 INPUT WORD$:L=LEN(WORD$)
90 GOSUB 240
100 GRAPHICS 2
110 POSITION 5,4
120 PRINT "OK. HERE'S MY WORD. TRY TO GUESS IT":PRINT "(YOU HAVE 10 GUESSES)"
130 PRINT #6;ANAGRAM$:PRINT
140 FOR G=1 TO 10
150 PRINT "GUESS ";G;
160 INPUT GUESS$
170 IF GUESS$=WORD$ THEN GOSUB 350:POP :GRAPHICS 0:GOTO 60
180 IF G<>10 THEN PRINT "NO, TRY AGAIN"
190 NEXT G
200 PRINT "THE ANSWER IS ";WORD$
210 FOR WAIT=1 TO 1000:NEXT WAIT

```

```
220 GRAPHICS 0
230 GOTO 60
240 REM * ANAGRAM SUBROUTINE
250 FOR I=1 TO L
260 SWITCH$(I,I)="0"
270 NEXT I
280 FOR J=1 TO L
290 X=INT(RND(1)*L)+1
300 IF SWITCH$(X,X)="1" THEN GOTO 290
310 SWITCH$(X,X)="1"
320 ANAGRAM$(X,X)=WORD$(J,J)
330 NEXT J:IF ANAGRAM$=WORD$ THEN 240
340 RETURN
350 REM * YOU GOT IT
360 GRAPHICS 2:POSITION 2,4
370 PRINT #6;"YOU GOT IT!"
380 FOR WAIT=1 TO 500:NEXT WAIT:RETURN
```



## FRACTURED TALES

Fractured tales are partially written stories that you can complete and turn into silly or wild tales. They are a bit like *Mad Libs*<sup>TM</sup>. Here's a fractured tale to complete:

One day the sad \_\_\_\_\_ took a long \_\_\_\_\_.

The next day a \_\_\_\_\_ went \_\_\_\_\_. The two  
met and \_\_\_\_\_.

A simple program for just about any fractured tale could begin by taking this form:

```
10 REM A SIMPLE FRACTURED PHRASE
20 DIM A$(25), B$(25)
30 PRINT "ONE DAY A ";
40 INPUT A$
50 PRINT "WENT TO THE ";
60 INPUT B$
70 PRINT "HERE IS YOUR SENTENCE:"
80 PRINT "ONE DAY A ";A$;" WENT TO THE ";B$
```

(Note: Remember the spaces in line 80.)

Note that on line 30 there is a space between A and the closing quote marks, and on line 50 there is a space between the opening quote marks and W and another space between E and the closing quotes.

By adjusting the DIMension sizes and changing the story, you can make up your own fractured tales. Here's one more complex fractured tale that is fun to play with friends or at parties.



*Mad Libs* is a registered trademark of Price, Sloane, & Co.

```

10 REM * <<< SLIGHTLY FRACTURED >>>
20 REM * <<< FAIRY TALES >>>
30 REM * developed by Robert A. Kahn
40 REM * modified by Ted M. Kahn
50 DIM Y$(20),F$(20),R$(20),J$(20)
60 DIM B$(20),L$(20),A$(20),V$(20)
70 PRINT "[CLEAR]";REM CLEAR THE SCREEN
80 PRINT " A SLIGHTLY FRACTURED FAIRY TA
LE";PRINT :PRINT
90 PRINT "WHAT'S YOUR NAME";
100 INPUT Y$:PRINT
110 PRINT "AND THE NAME OF"
120 PRINT "YOUR BEST FRIEND";
130 INPUT F$:PRINT
140 PRINT "THE NAME OF YOUR"
150 PRINT "FAVORITE RELATIVE";
160 INPUT R$:PRINT
170 PRINT "JOB TITLE OF A KIND OF PERSON
."
180 PRINT "WHO ALWAYS BUGS YOU";
190 INPUT J$:PRINT
200 PRINT "THE NAME OF YOUR BOSS";
210 INPUT B$:PRINT
220 PRINT
230 PRINT "JUST A FEW MORE QUESTIONS..."
."
240 PRINT
250 PRINT "NAME FOR A TYPE OF"
260 PRINT "LARGE BUILDING";
270 INPUT L$:PRINT
280 PRINT "ONE WORD FOR WHAT YOU WOULD L
IKE TO DO";
290 PRINT "WHEN YOU'RE ANGRY";
300 INPUT A$:PRINT
310 PRINT "NAME OF A LUXURY VEHICLE YOU'
VE"
320 PRINT "ALWAYS WANTED TO OWN";
330 INPUT V$
340 PRINT "[CLEAR]"
350 PRINT " THE THREE LITTLE PIGS
."
360 PRINT

```

```

370 PRINT "ONCE THERE WERE THREE LITTLE
PIGS"
380 PRINT "NAMED ";Y$;"", ";F$;" AND ";R$
;","
390 PRINT
400 PRINT "A HUNGRY ";J$;" NAMED ";B$
410 PRINT "WANTED TO EAT THEM FOR SUPPER
."
420 PRINT
430 PRINT "TERRIFIED, THEY HID IN THEIR
";L$;"."
440 PRINT "FURIOUS, ";B$;" SHOUTED: "
450 PRINT "'I'LL HUFF, AND I'LL PUFF AND
I'LL"
460 PRINT A$;" YOUR ";L$;" DOWN!"
470 PRINT
480 PRINT "NO FOOLS, THE THREE PIGS SNEA
KED OUT "
490 PRINT "THE BACK DOOR AND ESCAPED IN
THEIR"
500 PRINT V$;"."
510 PRINT
520 PRINT "                                T H E   E N D"
530 END

```

A variation on the fractured tale is the fractured letter. Here's a program for a fractured letter to your local senator. (This version assumes you have an ATARI 820, 822, or 825 printer. If you don't, change all LPRINT statements to PRINT.)

```

10 REM * <<< A TYPICAL FORM LETTER >>>
20 REM * PRINTS LETTER ON YOUR PRINTER
30 REM * by Robert Kahn & Lee Berman
40 REM * slightly modified by Ted Kahn
50 DIM D$(20),N$(30),A$(35),C$(20),S$(10
),Z$(5),M$(10),G$(40),P$(40)
60 PRINT "[CLEAR]BE SURE YOUR LINE PRINT
ER IS ON!";PRINT
70 PRINT "INPUT DATA FOR FORM LETTER:"
80 PRINT :PRINT
90 PRINT "DATE OF LETTER";
100 INPUT D$:PRINT :PRINT
110 PRINT "NAME OF CITIZEN";
120 INPUT N$:PRINT

```

```

130 PRINT "STREET ADDRESS";
140 INPUT A$:PRINT
150 PRINT "CITY";
160 INPUT C$:PRINT
170 PRINT "STATE";
180 INPUT S$:PRINT
190 PRINT "ZIP CODE";
200 INPUT Z$:PRINT :PRINT
210 PRINT "CONTRIBUTION";
220 INPUT M$:PRINT
230 PRINT
240 PRINT "YOUR GROUP'S NAME";
250 INPUT G$:PRINT
260 PRINT
270 PRINT "PROBLEM";
280 INPUT P$:PRINT
290 REM ARK -- LETTER GENERATED HERE
300 PRINT "[CLEAR]";PRINT "TOP MARGIN?";
:INPUT S:FOR I=1 TO S:LPRINT :NEXT I
310 LPRINT
320 LPRINT "          OFFICE OF
      "
330 LPRINT "          THE HONORABLE SENAT
OR "
340 LPRINT "          RALPH WINDBAG
      "
350 LPRINT :LPRINT
360 LPRINT "          ";D$
370 LPRINT :LPRINT
380 LPRINT N$
390 LPRINT A$
400 LPRINT C$;"", ";S$;" ";Z$
410 LPRINT
420 LPRINT "DEAR ";N$;"":
430 LPRINT
440 LPRINT "I'D LIKE TO THANK YOU, PERSONALLY,"
450 LPRINT "FOR YOUR GENEROUS CONTRIBUTION OF"
460 LPRINT M$;" ". GROUPS LIKE YOURS,"
470 LPRINT G$;"", WHO ARE"
480 LPRINT "WILLING TO STAND UP AND BE COUNTED,"
490 LPRINT "HAVE MADE IT POSSIBLE FOR US
      TO"

```

```

500 LPRINT "TRIUMPH IN OUR AGGRESSIVE CA
MPAIGN "
510 LPRINT "AGAINST ";P$;"!"
520 LPRINT
530 LPRINT "MANY PEOPLE ARE WORRIED THAT
NO ONE"
540 LPRINT "HERE IN THE STATE CAPITAL HA
S TIME"
550 LPRINT "FOR SMALL GROUPS LIKE YOURS.
"
560 LPRINT "BUT I WANT TO REASSURE YOU,
"
570 LPRINT "PERSONALLY, THAT ";G$;" WILL
ALWAYS"
580 LPRINT "BE MY PRIME CONCERN WHILE IN
OFFICE."
590 LPRINT
600 LPRINT "                                SINCERELY YOURS,
"
610 LPRINT "                                SENATOR RALPH WI
NDBAG"

```

A final variant of fractured tales involves *computer-composed stories*. You INPUT part of the story and the computer picks words at random out of a dictionary you program in to complete the story. Using techniques described in the section on three-letter-word games and those illustrated here, it is easy to develop programs for computer-generated science-fiction plots or "headlines" advertising the latest science-fiction movies. The program to generate these gems is not difficult to write, and you may keep a growing dictionary of your favorite title "parts" in much the same way as was shown in the dictionary for word games.

Below are some samples of the kind of "horror announcements" you might expect from a program of this kind. In these examples, drawn from a program developed at the Lawrence Hall of Science, University of California, Berkeley, the words in CAPITALS are generated by the computer; those in *small letters* are input by the user:

### Story 1

EARTH SCIENTISTS DISCOVER GIANT prehistoric birds WHICH (WHO) fly under the influence of alcohol AND ARE usually vegetarian AND . . . CANNOT BE KILLED BY toothpicks, SO THEY EAT US ALL UP (YUMMY)!!  
THE END



## Story 2

EARTH SCIENTISTS DISCOVER GIANT snow peas WHICH (WHO)  
 LOOK UPON US AS A SOURCE OF NOURISHMENT AND. . .CANNOT  
 BE KILLED BY pleasant meditating giraffes, SO SCIENTISTS INVENT  
 A WEAPON THAT TURNS THEM INTO peanut butter.  
 THE END

These programs usually have two forms: One in which the computer does all the work and just randomly selects the different parts and orders them together, and another in which the human provides some of the input, which is then used later (at often unpredictable moments).

The writing of this program will be left to the reader. One helpful hint is that the ON / GOSUB S1, S2, S3, S4, . . . statement can be very useful in structuring this program. Also, you may wish to create separate character strings for different sentence parts: For example, VERB\$="EATS CRUSHES TRAMPLES FLIES MELTS"; SUBJECTS\$="FLY-ING-SAUCERS MIDDLE-AGED RHINOCEROSES GARGANTUAN-THREE-LEGGED-LIZARDS" etc. Using a corresponding numerical array, you can keep track of where each individual "part" begins by noting its starting character number within the string. Then a random number can be used to select which one of several subjects, verbs, etc., is to be used.

Programs of this kind are designed for the wild and uninhibited, but can also be used to create beautiful peaceful scenarios. It all depends what you give the computer to work with.



## FORTUNE TELLER

Who hasn't wanted a private oracle or fortune teller? In this program, you can turn your ATARI into a fortune teller . . . in a manner is speaking. But then think about it: It's a computer, isn't it; don't some folks claim computers are never wrong!?

```

10 REM *    <<< FORTUNE TELLER >>>
20 REM *    copyright (C) 1980
30 REM *    by Kohl, Kahn & Lindsay
40 DIM NAME1$(20),NAME2$(20),NAME3$(20),
NAME4$(20),TEXT$(20)
50 PRINT "[CLEAR]"
60 PRINT "I AM THE ORACLE!"
70 PRINT "I WILL TELL YOU WHO YOU SHALL
MARRY"
80 PRINT "IF YOU GIVE ME 4 POSSIBLE NAME
S"
90 PRINT
100 GOSUB 240:NAME1$=TEXT$
110 GOSUB 240:NAME2$=TEXT$
120 GOSUB 240:NAME3$=TEXT$
130 GOSUB 240:NAME4$=TEXT$
140 PRINT "[CLEAR]"
150 PRINT "I'M LOOKING INTO MY CRYSTAL B
ALL..."
160 PRINT :FOR WW=1 TO 500:NEXT WW
170 PRINT "YOU WILL MARRY ";
180 LET C=INT(RND(1)*4)*10:GOTO 190+C
190 PRINT NAME1$:END
200 PRINT NAME2$:END
210 PRINT NAME3$:END
220 PRINT NAME4$:END
230 END
240 PRINT "NAME ";:INPUT TEXT$
250 PRINT
260 RETURN

```

## THE ORACLE

Here's a program you can use to show your friends what a wise machine you have! You can prepare different answers ahead of time (see lines 101–104); if you want more answers available, add them in numerical order beginning at line 105. Also, change line 45 so that the random number chosen (CHOICE) will fit the exact number of answers you have in the DATA statements. For example, if you wanted a total of 10 answers, your DATA statements would range from line 101–110, and line 45 would read:

```
45 CHOICE=INT(RND(1)*10)+1
10 REM *** ORACLE ***
15 REM HERB KOHL & LEN LINDSAY, 1980
20 DIM QUESTION$(80),ANSWER$(40)
25 PRINT "[CLEAR]"
30 PRINT "WHAT QUESTION DO YOU HAVE
   FOR":PRINT "THE ORACLE"
40 INPUT QUESTION$
45 CHOICE=INT(RND(1)*4)+1
50 RESTORE 100+CHOICE
60 READ ANSWER$
70 PRINT
75 PRINT "THE ORACLE SAYS,"
80 PRINT ANSWER$
85 PRINT :PRINT
90 GOTO 30
101 DATA "IT ALL DEPENDS ON YOU!"
102 DATA "ONLY IF YOU ASK. . ."
103 DATA "YES—BUT ONLY ONCE!"
104 DATA "IF YOU DON'T TELL"
```

## YOUR FIRST COMPUTER HOROSCOPE PROGRAM

This is a very simple variation on the ORACLE and FORTUNE-TELLING program which we've just seen. You may fill in your own horoscopes for each sign by changing the content of the DATA statements, lines 140-250. However, your horoscopes will have to be no longer than two lines (80 characters total). This is because the loop from lines 90-120 systematically checks each individual DATA statement to see which one fits your sign (or at least the first three letters of it!).

```

10 REM *   <<< ASTROLOGY PROGRAM >>>
20 REM *   copyright (C) 1980
30 REM *   by Kohl, Kahn & Lindsay
40 DIM SIGN$(10),TEXT$(80)
50 PRINT "[CLEAR]"
60 PRINT :PRINT :PRINT :PRINT "WHAT IS Y
OUR SIGN";
70 INPUT SIGN$
80 RESTORE :PRINT
90 FOR LOOP=1 TO 12
100 READ TEXT$
110 IF TEXT$(1,3)=SIGN$(1,3) THEN PRINT
TEXT$:POP :GOTO 60
120 NEXT LOOP
130 PRINT "I DO NOT KNOW THAT SIGN":GOTO
60
140 DATA LEO: YOU THRIVE ON FUN AND GAM
ES
150 DATA ARIES: YOU AREN'T TOO BASHFUL A
BOUT GOING AFTER WHAT YOU WANT
160 DATA LIBRA: DON'T OVERDO THE PLEASUR
E--LOOK OUT FOR WORK ADVANCES TOO
170 DATA VIRGO: YOU'RE COOL AS A CUCUMBE
R BUT YOU NEED SOMEONE TO CONFIDE IN
180 DATA CANCER: DON'T LET YOUR EMOTION
AL UPS AND DOWNS GET TO YOU
190 DATA GEMINI: YOU'RE NEVER ALONE--YOU
R CHARM WILL TAKE CARE OF YOU
200 DATA PISCES: LOOK BEYOND THE SURFAC
E WORD OR TOUCH
210 DATA TAURUS: YOU'RE POSSESSIVE BUT
DON'T LET IT KEEP YOU DOWN
220 DATA SCORPIO: DON'T BE GUILTY--JUST
BE GRATEFUL (AND SHOW IT)

```

```
230 DATA AQUARIUS: YOU ALWAYS SURPRISE E  
EVERYONE SO MAYBE NOW IT'S YOUR TURN  
240 DATA CAPRICORN: TAKE RISKS THAT APP  
EAL TO YOUR SENSE OF GROWTH  
250 DATA SAGITARIUS: YOU'RE DYNAMITE SO  
GO TO IT NOW
```

## FORTUNE COOKIES!

Here is a variation on the Horoscope Program. (Have the computer print them out using a line printer!) Use a cookbook for the recipe and serve them to your friends and family. You can modify the Horoscope-Astrology program fairly easily to become a Fortune-Cookie Generator!

## SILLY QUESTIONS, SILLY ANSWERS

This is a program which illustrates the old computer proverb:

Garbage In, Garbage Out

However, it may come in useful at parties when you want to show your friends just how intelligent your computer really is!

```

10 REM * < SILLY QUESTIONS/ANSWERS >
20 REM * (C) 1980 by Herbert Kohl,
30 REM * Ted Kahn and Len Lindsay
40 REM * Demonstrates use of
50 REM * GOSUB & RETURN
60 DIM QUESTION1$(80),QUESTION2$(80),QUE
STION3$(80),A$(7)
70 DIM ANSWER1$(80),ANSWER2$(80),ANSWER3
$(80),TEXT$(80)
80 PRINT "[CLEAR]"
90 A$="A":GOSUB 210:QUESTION1$=TEXT$
100 A$="ANOTHER":GOSUB 210:QUESTION2$=TE
XT$
110 GOSUB 210:QUESTION3$=TEXT$
120 PRINT "[CLEAR]"
130 A$="A":GOSUB 250:ANSWER1$=TEXT$
140 A$="ANOTHER":GOSUB 250:ANSWER2$=TEXT
$
150 GOSUB 250:ANSWER3$=TEXT$
160 PRINT "[CLEAR]"
170 PRINT QUESTION1$;"?":GOSUB 290
180 PRINT QUESTION2$;"?":GOSUB 290
190 PRINT QUESTION3$;"?":GOSUB 290
200 END
210 PRINT "TYPE IN ";A$;" SILLY QUESTION
,":REM MUST BE LESS THAN TWO LINES LONG
220 INPUT TEXT$
230 PRINT
240 RETURN
250 PRINT "TYPE IN ";A$;" SILLY ANSWER"
260 INPUT TEXT$
270 PRINT

```

```
280 RETURN
290 CHOICE=INT(RND(1)*3)+1
300 IF CHOICE=1 THEN PRINT ANSWER1$
310 IF CHOICE=2 THEN PRINT ANSWER2$
320 IF CHOICE=3 THEN PRINT ANSWER3$
330 PRINT
340 RETURN
```

## COLOR PREFERENCE

This is really another oracle program, only this time it has an added bonus: The computer actually changes the color of the screen according to your preference. (The use of color will be explained much more thoroughly in Chapter Eight.)

```

10 REM * <<< COLOR PREFERENCE >>>
12 REM *      copyright (C) 1980 by
14 REM *      Kahn, Kohl, & Lindsay
16 DIM TEXT$(30),MESSAGE$(30)
18 PRINT "[CLEAR]"
20 SETCOLOR 1,1,0
22 SETCOLOR 2,0,10
24 PRINT "WHICH COLOR DO YOU LIKE MOST"
26 PRINT "RED, BLUE, OR YELLOW"
28 TEXT$="";NC=0
30 INPUT TEXT$
32 TRAP 40
34 IF TEXT$(1,1)="R" THEN NC=1;GOTO 42
36 IF TEXT$(1,1)="B" THEN NC=2;GOTO 42
38 IF TEXT$(1,1)="Y" THEN NC=3;GOTO 42
40 PRINT "[CLEAR]PLEASE CHOOSE ONLY ";GO
   TO 26
42 RESTORE 100+ASC(TEXT$)
44 PRINT "[CLEAR]";
46 READ MESSAGE$
48 ON NC GOSUB 200,210,220
50 PRINT :PRINT "YOU ARE ";MESSAGE$
52 PRINT :PRINT "NEXT PERSON, PLEASE."
54 GOSUB 300
56 GOSUB 300;GOTO 18
166 DATA PLACID AND PEACEFUL
182 DATA HOTBLOODED AND INDEPENDENT
189 DATA IMPATIENT BUT OPTIMISTIC
200 SETCOLOR 2,3,4;RETURN
210 SETCOLOR 2,8,2;RETURN
220 SETCOLOR 2,1,12;RETURN
300 WAIT=600;REM DEFAULT WAIT LENGTH
310 FOR LOOP=1 TO WAIT
320 NEXT LOOP
330 RETURN

```



Here are a number of more complex programs. The first one, which we call USSTATES, will help your children (or you) learn the names of all 50 states. This game is a variation on the SERIES game which we introduced earlier.

```

10 REM <<< USSTATES >>>
20 REM LEN LINDSAY, 1980
30 NUMBER=50
40 DIM PLAYER1$(8),PLAYER2$(8)
50 DIM IN$(15),LASTW$(14),STARTL$(1),TES
   T$(14)
60 DIM PICKED(NUMBER),S(1),P(1),T(1)
70 GAMEOVER=9000
80 OPTION=2110:GOSUB OPTION-10
90 GOSUB 2000:REM GRAPHICS & COLORS
100 PRINT #6;"   STATE NAMES"
110 POSITION 6,8:PRINT #6;"TO BEGIN"
120 POSITION 5,9:PRINT #6;"HIT start"
130 POSITION 2,2:PRINT #6;"FOR INSTRUCTI
   ONS"
140 POSITION 5,3:PRINT #6;"HIT option"
150 POSITION 6,11:PRINT #6;" ATARI "
160 KEY=PEEK(53279)
170 IF KEY=3 THEN GOSUB 1000:GOTO 90
180 IF KEY=5 THEN GOSUB OPTION:OPTION=OP
   TION+10:GOSUB 2220:IF OPTION>2130 THEN O
   PTION=2100
190 IF KEY=6 THEN 250
200 POSITION 0,5:PRINT #6;PLAYER1$;" VS
   ";PLAYER2$;"HIT select TO CHANGE"
210 SETCOLOR 3,0,RND(1)*256
220 FOR DELAY=1 TO 50:NEXT DELAY
230 SETCOLOR 2,0,PN:PN=PN+2
240 GOTO 160
250 GRAPHICS 2+16:PRINT #6;" WAIT"
260 FOR A=1 TO NUMBER:PICKED(A)=0:NEXT A
   :REM INIT
270 S(0)=0:S(1)=0
280 T(0)=0:T(1)=0
290 COUNT=0
300 PLAYER=1

```

```

310 TEST$=PLAYER2$:GOSUB 2300:PLAYER2$=T
EST$
320 PASS=0
330 PLAYER=1-PLAYER
340 POSITION 0,10:GOSUB 6900
350 POSITION 0,11:GOSUB 6900
360 IF PASS>1 THEN GOTO GAMEOVER
370 COUNT=COUNT+1
380 GOSUB 6000
390 POSITION 1,3:PRINT #6;"LAST WORD WAS
;"
400 POSITION 1,4:GOSUB 6900:REM ERASE LI
NE
410 IF LEN(LASTW$)>1 THEN IF LASTW$(1,1)
<="Z" THEN TEST$=LASTW$:GOSUB 2300:POSIT
ION 1,4:PRINT #6;TEST$
420 POSITION 1,6:PRINT #6;"NEXT WORD MUS
T"
430 POSITION 3,7:PRINT #6;"START WITH: "
;
440 STARTL$=CHR$(INT(RND(1)*26)+65)
450 IF LEN(LASTW$)>1 THEN STARTL$=LASTW$
(LEN(LASTW$),LEN(LASTW$))
460 PRINT #6;CHR$(ASC(STARTL$)+160)
470 POSITION 1,8
480 IF P(PLAYER)=0 THEN PRINT #6;"ENTER
YOUR WORD"
490 IF P(PLAYER)=1 THEN PRINT #6;"MY WOR
D IS: "
500 GOSUB 6900
510 IF P(PLAYER)=1 THEN GOSUB 7000:GOTO
600:REM COMPUTER TURN
520 CHAR=1
530 IN$=" "
540 POSITION 1,9:PRINT #6;IN$
550 OPEN #1,4,0,"K"
560 POKE 764,255:REM CLEAR BUFFER
570 IF PEEK(764)=255 THEN SETCOLOR 3,0,R
ND(1)*256:GOTO 570
580 GET #1,IN
590 CLOSE #1
600 IF IN=126 AND CHAR>1 THEN CHAR=CHAR-
1:IN$(CHAR,CHAR)=" ":GOTO 540:REM DELETE
HIT

```

```

610 IF IN=155 THEN 670:REM RETURN HIT
620 IF IN<65 AND IN<>32 OR IN>90 THEN 55
0:REM NOT ALPHABETIC
630 IN$(CHAR,CHAR)=CHR$(IN)
640 CHAR=CHAR+1
650 IF CHAR>15 THEN CHAR=15
660 GOTO 540
670 IF LEN(IN$)<1 THEN 520
680 IF CHAR>1 THEN IN$=IN$(1,CHAR-1):REM
MINUS TRAILING SPACES
690 IF IN$="PASS" THEN PASS=PASS+1:GOTO
330
700 IF ASC(IN$)<>ASC(STARTL$) THEN PASS=
PASS+1:PRINT #6;"1st letter's not ";STAR
TL$:GOTO 840
710 RESTORE
720 FOR TEST=1 TO NUMBER
730 READ TEST$
740 IF TEST$=IN$ THEN POP:GOTO 780
750 NEXT TEST
760 PRINT #6;"not a legal word"
770 GOTO 840:REM CONTINUE
780 IF PICKED(TEST)>0 THEN PRINT #6;"tha
t word is used hit start":GOSUB 2200:G
OSUB 6000:GOTO 340
790 PICKED(TEST)=PLAYER+1
800 PRINT #6;"that word works"
810 LASTW$=IN$
820 S(PLAYER)=S(PLAYER)+1
830 T(PLAYER)=T(PLAYER)+LEN(IN$)
840 PRINT #6;"HIT start";
850 GOSUB 6100
860 GOSUB 2200:IF KEY<>6 THEN 860
870 GOSUB 7100:GOTO 320
999 END
1000 GRAPHICS 1+16:GOSUB 2010
1010 PRINT #6;"NAMES OF STATES ARE"
1020 PRINT #6;"THE LEGAL WORDS."
1030 PRINT #6;"THE LAST LETTER OF"
1040 PRINT #6;"THE PREVIOUS WORD"
1050 PRINT #6;"MUST BE THE FIRST"
1060 PRINT #6;"LETTER OF THE NEXT"
1070 PRINT #6;"WORD. NO REPEATS"
1075 PRINT #6

```

```

1080 PRINT #6;"type 'PASS' if you"
1090 PRINT #6;"can not move."
1095 PRINT #6
1100 PRINT #6;"AN ILLEGAL MOVE IS"
1110 PRINT #6;"TREATED AS PASSING"
1120 PRINT #6;"YOUR TURN"
1125 PRINT #6
1130 PRINT #6;"THE GAME IS OVER"
1140 PRINT #6;"IF BOTH PLAYERS PASS"
1200 PRINT #6
1900 PRINT #6;"HIT start"
1990 GOSUB 2200:IF KEY<>6 THEN 1990
1999 RETURN
2000 GRAPHICS 2+16
2010 SETCOLOR 0,2,8:REM CAPS - REG TEXT
2020 SETCOLOR 1,12,10:REM lower LUMIN -
    OPTIONS IN STARTER
2030 SETCOLOR 2,8,12:REM CAPS WINDOW - T
    ITLE IN STARTER
2040 SETCOLOR 3,4,6:REM lower - NAMES OF
    THE SPECIAL KEYS - VIBRATES NAMES
2050 SETCOLOR 4,0,0:REM BACKGROUND
2099 RETURN
2100 PLAYER1$="human"    ":PLAYER2$="compu
    ter":P(0)=0:P(1)=1
2109 RETURN
2110 PLAYER1$="computer":P(0)=1
2119 RETURN
2120 PLAYER2$="human"    ":P(1)=0
2129 RETURN
2130 PLAYER1$="human"    ":P(0)=0
2139 RETURN
2200 IF PEEK(53279)<>7 THEN 2200
2210 KEY=PEEK(53279):IF KEY=7 THEN SETCO
    LOR 3,0,RND(1)*255:GOTO 2210
2220 IF PEEK(53279)<>7 THEN 2220
2299 GOSUB 2010:RETURN
2300 FOR A=1 TO LEN(TEST$)
2310 TEST$(A,A)=CHR$(ASC(TEST$(A,A))+128
    )
2320 NEXT A
2399 RETURN
6000 GOSUB 2010

```

```

6010 TEST$=PLAYER1$:GOSUB 2300:PLAYER1$=
TEST$
6020 TEST$=PLAYER2$:GOSUB 2300:PLAYER2$=
TEST$
6030 POSITION 0,0
6050 PRINT #6;PLAYER1$;" VS ";PLAYER2$
6100 POSITION 1,1:PRINT #6;S(0);"!";T(0)
6110 POSITION 13,1:PRINT #6;S(1);"!";T(1
)
6900 PRINT #6;" ";
6999 RETURN
7000 RESTORE
7010 FOR TEST=1 TO NUMBER
7020 SETCOLOR 3,3,RND(1)*256
7030 READ IN$
7040 IF ASC(IN$)=ASC(STARTL$) AND PICKED
(TEST)=0 THEN POP :GOTO 7100
7050 NEXT TEST
7060 IN$="PASS"
7070 POSITION 1,10
7080 PRINT #6;"I PASS! HIT start"
7090 GOSUB 2200:IF KEY<>6 THEN 7090
7100 GOSUB 2300
7110 CHAR=LEN(IN$)+1
7120 POSITION 1,9:PRINT #6;IN$
7199 RETURN
9000 POSITION 1,10
9010 PRINT #6;"[CLEAR]"
9020 FOR L=30 TO 10 STEP -0.5:SOUND 0,L,
10,10:NEXT L:SOUND 0,0,0,0
9030 GOSUB 6000
9040 PRINT #6
9050 POSITION 3,10:PRINT #6;"GAME IS OVE
R"
9060 POSITION 4,11:PRINT #6;"HIT start";
9070 GOSUB 200:IF KEY<>6 THEN 9070
9080 RUN
10650 DATA ALABAMA,ALASKA,ARIZONA,ARKANS
AS
10660 REM B
10670 DATA CALIFORNIA,COLORADO,CONNECTIC
UT
10680 DATA DELAWARE

```

```

10690 REM E
10700 DATA FLORIDA
10710 DATA GEORGIA
10720 DATA HAWAII
10730 DATA IDAHO,ILLINOIS,INDIANA,IOWA
10740 REM J
10750 DATA KANSAS,KENTUCKY
10760 DATA LOUISIANA
10770 DATA MAINE,MARYLAND,MASSACHUSETTS,
MICHIGAN,MINNESOTA,MISSISSIPPI,MISSOURI,
MONTANA
10780 DATA NEBRASKA,NEVADA,NEW HAMPSHIRE
,NEW JERSEY,NEW MEXICO,NEW YORK,NORTH CA
ROLINA,NORTH DAKOTA
10790 DATA OHIO,OKLAHOMA,OREGON
10800 DATA PENNSYLVANIA
10810 REM Q
10820 DATA RHODE ISLAND
10830 DATA SOUTH CAROLINA,SOUTH DAKOTA
10840 DATA TENNESSEE,TEXAS
10850 DATA UTAH
10860 DATA VERMONT,VIRGINIA
10870 DATA WASHINGTON,WEST VIRGINIA,WISC
ONSIN,WYOMING
10880 REM X
10890 REM Y
10900 REM Z
10910 DATA ZZZ

```

Now, here is TOESITCH. It is a two-player game. Each player has a list of words to choose from. The goal of the game is to choose three words which contain the same letter. More specific instructions are included in the program.

```

10 REM <<< TOESITCH >>>
20 REM LEN LINDSAY 1980
30 DIM PLAYER1$(8),PLAYER2$(8),A$(1),P$(
4),W$(4),WORD$(4)
40 DIM P(2),W(9),L(1,26)
50 GOSUB 2000:REM SETCOLORS
60 GM=2110
70 GOSUB 2100:REM INIT PLAYERS
80 GAMEOVER=9000

```

```

90 WIN=99:REM INIT
100 PRINT #6;"   toes itch game"
110 GOSUB 4400:REM INIT
120 POSITION 6,8:PRINT #6;"TO BEGIN"
130 PRINT #6;"   HIT START"
140 POSITION 2,2:PRINT #6;"FOR INSTRUCTI
ONS"
150 PRINT #6;"   HIT OPTION"
160 POSITION 6,11:PRINT #6;" ATARI"
170 KEY=PEEK(53279)
180 IF KEY=3 THEN GOSUB 1000:GOTO 100
190 IF KEY=5 THEN GOSUB GM:GM=GM+10:GOSU
B 2520:IF GM>2130 THEN GM=2100
200 IF KEY=6 THEN 290
210 POSITION 0,5:PRINT #6;PLAYER1$;" VS
";PLAYER2$;"HIT SELECT TO CHANGE"
220 SETCOLOR 2,0,RND(1)*256
230 FOR DELAY=1 TO 150:NEXT DELAY
240 GOTO 170
250 REM
260 REM -TO CHANGE WORDS, JUST CHANGE TH
E WORDS IN DATA IN LINE 280
270 REM
280 DATA SLIM,WORM,GAME,ITCH,TOES,TALK,V
INE,PONY,SAND
290 FOR A=1 TO 9:W(A)=1:NEXT A
300 PLAYER=2
310 WORDLEFT=9
320 GOSUB 2000
330 GOSUB 6000
340 PLAYER=2-PLAYER:REM SWITCH PLAYERS
350 IF WORDLEFT=0 THEN GOTO GAMEOVER
360 GOSUB 2010
370 IF P(PLAYER)=1 THEN GOSUB 3000:REM H
UMAN
380 IF P(PLAYER)=0 THEN GOSUB 7000:REM C
OMPUTER
390 GOSUB 4500:REM SEE IF 3 LETTERS SAME
400 GOTO 340
999 END
1000 GRAPHICS 1+16
1030 PRINT #6;"TWO PLAYERS TAKE"
1040 PRINT #6;"TURNS PICKING A"
1050 PRINT #6;"WORD FROM A LIST"

```

```

1060 PRINT #6;"IN THE CENTER OF"
1070 PRINT #6;"THE SCREEN."
1075 PRINT #6
1080 PRINT #6;"select=change word"
1090 PRINT #6;"start=take that word"
1100 PRINT #6;"THE LETTERS IN EACH"
1110 PRINT #6;"WORD ARE THE MAJOR"
1120 PRINT #6;"FOCUS OF THIS GAME."
1125 PRINT #6
1130 PRINT #6;"the first player to"
1140 PRINT #6;"have three of any"
1150 PRINT #6;"letter wins."
1155 PRINT #6
1160 PRINT #6;"THE GAME IS A DRAW"
1170 PRINT #6;"IF ALL WORDS ARE"
1180 PRINT #6;"PICKED AND NO ONE"
1190 PRINT #6;"WON YET."
1195 PRINT #6
1200 PRINT #6;"HIT start TO RETURN"
1900 GOSUB 2500:IF KEY<>6 THEN 1900
1999 RUN
2000 GRAPHICS 2+16
2010 REM
2011 SETCOLOR 3,4,4:REM RED
2012 SETCOLOR 2,0,14:REM WHITE
2013 SETCOLOR 1,8,6:REM BLUE
2014 SETCOLOR 4,0,0:REM BLACK
2015 SETCOLOR 0,2,8:REM GOLD
2099 RETURN
2100 PLAYER1$="computer":PLAYER2$="human
____":P(0)=0:P(2)=1:RETURN
2110 PLAYER2$="computer":P(2)=0:RETURN
2120 PLAYER1$="human____":P(0)=1:RETURN
2130 PLAYER2$="human____":P(2)=1:RETURN
2500 IF PEEK(53279)<>7 THEN 2500
2510 KEY=PEEK(53279):IF KEY=7 THEN SETCO
LOR PLAYER+1,0,RND(1)*255:FOR WW=1 TO 50
:NEXT WW:GOTO 2510
2520 IF PEEK(53279)<>7 THEN 2520
2599 RETURN
3000 WORDUP=INT(RND(1)*9+1):REM INIT
3010 IF WORDUP>9 THEN WORDUP=1
3020 IF W(WORDUP)<>1 THEN WORDUP=WORDUP+
1:GOTO 3010

```



```

3030 GOSUB 6000
3040 IF P(PLAYER)=0 THEN 3080
3050 GOSUB 2510
3060 IF KEY=5 THEN POSITION 7,WORDUP:WOR
DUP=WORDUP+1:GOTO 3010
3070 IF KEY<>6 THEN 3030
3080 W(WORDUP)=PLAYER:GOSUB 6000
3090 WORDLEFT=WORDLEFT-1
3099 RETURN
4000 REM P$ IN AND P$ OUT (+128)
4010 FOR DEL=1 TO 4
4020 P$(DEL,DEL)=CHR$(ASC(P$(DEL,DEL))+1
28)
4030 NEXT DEL
4099 RETURN
4100 REM HIGHLIGHT THE WINNING LETTER
4101 REM WORDS ARE W$
4102 REM WINNING LETTER IS A$
4110 FOR KEY=1 TO 4
4120 IF W$(KEY,KEY)=A$ THEN W$(KEY,KEY)=
CHR$(ASC(A$)+128)
4130 NEXT KEY
4199 RETURN
4400 FOR DEL=0 TO 1
4410 FOR KEY=1 TO 26
4420 L(DEL,KEY)=0
4430 NEXT KEY
4440 NEXT DEL
4499 RETURN
4500 REM UPDATE LETTER COUNT
4501 REM WORD IS W(WORDUP)
4510 FOR KEY=1 TO 4
4520 LTR=ASC(WORD$(KEY,KEY))-64
4530 L(PLAYER*0.5,LTR)=L(PLAYER*0.5,LTR)
+1
4540 IF L(PLAYER*0.5,LTR)=3 THEN POP :PO
P :A$=CHR$(LTR+64):WIN=PLAYER:GOTO GAMEO
VER
4550 NEXT KEY
4599 RETURN
6000 POSITION 0,0
6010 PRINT #6;PLAYER1$
6020 POSITION 12,0
6030 PRINT #6;PLAYER2$

```

```

6100 RESTORE
6110 FOR ROW=1 TO 9
6120 READ W$
6130 FOR VERT=0 TO 2
6140 POSITION VERT*7+1,ROW
6150 F$="      "
6160 IF W(ROW)=VERT THEN F$=W$:IF ROW=WORDUP THEN WORD$=W$:GOSUB 4000
6170 PRINT #6;F$;
6180 NEXT VERT
6190 NEXT ROW
6199 RETURN
7000 IF W(5)=1 THEN WORDUP=5:GOSUB 3010:
GOTO 7999
7010 GOSUB 3000
7999 RETURN
9000 REM GAMEOVER
9010 RESTORE
9020 FOR DEL=1 TO 9
9030 READ W$
9040 GOSUB 4100
9050 IF W(DEL)=PLAYER THEN POSITION PLAY
ER*7+1,DEL:PRINT #6;W$
9060 NEXT DEL
9070 FOR LL=30 TO 1 STEP -0.2:SOUND 0,LL
,10,10:NEXT LL:SOUND 0,0,0,0
9100 POSITION 4,10
9110 IF WIN<3 THEN PRINT #6;"THREE ";CHR
$(LTR+192);" WINS"
9120 IF WIN>2 THEN PRINT #6;"NO ONE WINS
":WORDUP=0:GOSUB 6000
9200 POSITION 6,11:PRINT #6;"HIT START";
9210 ZW=1
9220 SETCOLOR 2,RND(1)*16,4
9230 IF WIN>2 THEN 9260
9240 ZW=1-ZW:SETCOLOR PLAYER+1,1,ZW*RND(
1)*255
9250 FOR DEL=1 TO 50:NEXT DEL
9260 IF PEEK(53279)=7 THEN 9220
9999 RUN

```

Finally, here's a VOCABULARY game in which you enter the words to be learned as DATA (line 1000 and following). In addition to the word to be guessed, you enter "hints" for the player; these hints are: a *synonym*

for the word; its *definition*; which *syllable* is accented; and how many *vowels* are in the word. Each DATA statement from line 1000 on has the word and each of these hints, separated by commas. (Note that the program already has four examples already, at lines 1000, 1010, 1020, and 1030). Because this game allows you to put in your own words, it can be used to help children with weekly vocabulary or foreign language classes as well.

```

10 REM * <<< WANTED: VOCABULARY >>>
20 REM *      (C) 1980 Len Lindsay
30 REM *      idea from
40 REM *      Calculators/Computers
50 PRINT "ADD YOUR OWN WORDS IN LINES 10
00 ON ":FOR WAIT=1 TO 1000:NEXT WAIT
60 DIM PART$(18),WORD$(18),SYNONYM$(18),
MEANING$(36),ACCENT$(18),SYLLABLE$(10),VO
WEL$(10),CONSONANT$(10),TEXT$(18)
70 DIM XX$(18)
80 DIM NAME$(18)
90 POKE 82,2:REM SET LEFT MARGIN
100 SETCOLOR 2,2,6
110 SETCOLOR 4,2,6
120 SETCOLOR 1,2,2
130 REM
140 WORDS=0:REM INITIALIZE - THEN COUNT
HOW MANY WORDS ARE AVAILABLE
150 RESTORE 1000+10*(WORDS+1)
160 TRAP 200:REM THIS LETS US CONTINUE W
HEN WE RUN OUT OF DATA
170 READ PART$:REM IF NOTHING LEFT TO RE
AD -ERROR- AND GOTO TRAP 200
180 WORDS=WORDS+1:REM INCREASE COUNT
190 GOTO 150:REM SEE IF MORE WORDS
200 PICK=INT(RND(1)*WORDS)+1:REM CHOOSE
A RANDOM WORD TO USE
210 RESTORE 1000+PICK*10:REM RESTORE THE
DATA TO START AT CHOSEN WORD
220 READ PART$
230 READ WORD$
240 READ SYNONYM$
250 READ MEANING$
260 READ SYLLABLE
270 READ ACCENT
280 READ VOWEL

```

```

290 RESTORE 900+SYLABLE:READ SYLABLE$
300 RESTORE 900+ACCENT:READ TEXT$:READ A
CCENT$
310 RESTORE 900+VOWEL:READ VOWEL$
320 RESTORE 900+LEN(WORD$)-VOWEL:READ CO
NSONANT$
330 GOSUB 6000
340 POSITION 20,1:PRINT PART$
350 POSITION 20,3:PRINT SYNONYM$
360 CLUES=0:REM INITIALIZE CLUE COUNT
370 POSITION 2,18:REM READY FOR QUESTION
S
380 PRINT "[DEL LINE][DEL LINE][DEL LINE
][DEL LINE][DEL LINE][DEL LINE][DEL LINE
][DEL LINE]";:REM DELETE LINES FROM BOTTO
M OF SCREEN
390 PRINT "WHAT DO YOU THINK THE WORD IS
?"
400 GOSUB 15000:REM INPUT ROUTINE
410 IF XX$=WORD$ THEN 750
420 PRINT "YOU DID NOT IDENTIFY THE WORD
!";
430 PAUSE=100:GOSUB 14020:REM PAUSE
440 CLUE=CLUE+1
450 ON CLUE GOSUB 470,530,600,680
460 GOTO 370
470 POSITION 21,5
480 PRINT "[C.][LEFT]";
490 GOSUB 14000:REM PAUSE
500 PRINT "[BELL]";
510 PRINT SYLABLE$
520 RETURN
530 POSITION 21,11
540 PRINT "[C.][LEFT]";
550 GOSUB 14000:REM PAUSE
560 PRINT "[BELL]";:REM BEEP
570 IF ACCENT$="NONE" THEN ACCENT$="ONLY
ONE SYLABLE"
580 PRINT ACCENT$
590 RETURN
600 POSITION 21,7
610 PRINT "[C.][LEFT]";
620 GOSUB 14000:REM PAUSE
630 PRINT "[BELL]";:REM BEEP

```

```

640 PRINT VOWEL$
650 POSITION 21,9
660 PRINT CONSONANT$
670 RETURN
680 CLUE=3
690 PRINT "NO MORE CLUES, BUT WATCH CLOS
ELY AND"
700 PRINT "YOU WILL GET A PEEK OF THE WO
RD"
710 PRINT "          ";WORD$;"[UP]"
720 GOSUB 14000:REM PAUSE
730 PRINT "[DEL LINE]";
740 RETURN
750 PRINT "YES! THAT IS CORRECT!"
760 CASE=CASE+1
770 PRINT "YOU HAVE NOW SOLVED ";CASE;"
CASE";:IF CASE>1 THEN PRINT "S";
780 PRINT
790 PRINT "WOULD YOU LIKE ANOTHER CASE?"
;
800 GOSUB 15100:REM GET ROUTINE
810 IF XX$="N" THEN END
820 IF XX$="Y" THEN 200
830 GOTO 800
840 REM DATA TO CHANGE 1 TO ONE OR FIRST
ETC...
900 DATA NONE,NONE
901 DATA ONE,FIRST
902 DATA TWO,SECOND
903 DATA THREE,THIRD
904 DATA FOUR,FOURTH
905 DATA FIVE,FIFTH
906 DATA SIX,SIXTH
907 DATA SEVEN,SEVENTH
908 DATA EIGHT,EIGHTH
909 DATA NINE,NINTH
910 DATA TEN,TENTH
999 END
1000 REM PART / WORD / SYNONYM / MEANING
/ SYLLABLES / ACCENTED SYLLABLE / VOWELS
1001 REM IF ONLY ONE SYLLABLE - SET ACCEN
TED SYLLABLE TO 0
1010 DATA ADJECTIVE,CENTRAL,MIDDLE,NEAR
THE CENTER,2,1,2

```

```

1020 DATA VERB,FORD,PASS THROUGH,CROSS W
ATER BY WADING,1,0,1
1030 DATA NOUN,SCYTHE,BLADE,LONG CURVED
BLADE ON LONG HANDLE,1,0,2
6000 PRINT "[CLEAR]";:REM CLEAR SCREEN
6100 PRINT "[Q]RJRJRJRJRJRJRJRJRJRJRJRJRJR
RJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJR
JRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJR
[ ]"
6110 PRINT "I _____ WANTED:
      | "
6120 PRINT "I _____
      | "
6130 PRINT "I _____ ALIAS:
      | "
6140 PRINT "I _____
      | "
6150 PRINT "I _____ SYLABLES:
      | "
6160 PRINT "I _____
      | "
6170 PRINT "I _____ VOWELS:
      | "
6180 PRINT "I _____
      | "
6190 PRINT "I _____ CONSONANTS:
      | "
6200 PRINT "I _____
      | "
6210 PRINT "I ACCENTED SYLLABLE:
      | "
6220 PRINT "[A]-----
-----[D]"
6230 PRINT "I _____ LAST KNOWN MEANIN
G      | "
6240 PRINT "I _____
      | "
6245 PRINT "I _____
      | "
6250 PRINT "[UP]";MEANING$
6255 PRINT "I _____
      | "
6260 PRINT "[Z]RJRJRJRJRJRJRJRJRJRJRJRJRJR
RJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJRJR

```

```

]C]
6999 RETURN
14000 REM * PAUSE SUBROUTINE
14001 REM PAUSE IS DEFAULT AT 999
14002 REM TO SET YOUR OWN PAUSE
14003 REM SIMPLY DO THIS ON THE CALL
14004 REM
14005 REM PAUSE=300 : GOSUB 14020
14006 REM
14007 REM NOTE GOSUB AT 14020 IF YOU
14008 REM SET YOUR OWN PAUSE LENGTH
14009 REM
14010 PAUSE=999
14020 FOR LOOP=1 TO PAUSE
14030 NEXT LOOP
14099 RETURN
15000 REM * INPUT SUBROUTINE
15001 REM XX$ IS STRING TO BE INPUT
15002 REM XX IS THE NUMERIC VALUE OF THE
  INPUT
15003 REM MUST DIM XX$(N) - N IS THE L
  IMIT YOU WISH ON THE INPUT STRING LENGTH
15004 REM
15010 PRINT "[BELL]";
15020 POKE 764,255:REM CLEAR BUFFER
15030 TRAP 15030:REM TRAP ERROR ON INPUT
15040 INPUT XX$:REM INPUT STRING
15050 TRAP 15090:REM TRAP IF NO NUMERIC
  AVAILABLE IN STRING
15060 XX=VAL(XX$):REM NUMERIC
15090 TRAP 34567:REM SPRING TRAP
15099 RETURN
15100 REM * GET SUBROUTINE
15101 REM XX$ IS THE CHARACTER INPUT
15102 REM XX IS THE NUMERIC EQUIVALENT O
  F INPUT CHARACTER
15103 REM USES FILE #4
15104 REM
15110 PRINT "[BELL]";
15120 POKE 764,255:REM CLEAR BUFFER
15130 TRAP 15130:REM TRAP ERROR ON GET
15139 CLOSE #4:REM MAKE SURE FILE IS CLO
  SED BEFORE OPENING IT

```

```
15140 OPEN #4,4,0,"K:":REM OPEN KEYBOARD
      FOR A GET
15150 GET #4,XX:REM GET ASCII VALUE OF K
      EY HIT
15155 CLOSE #4:REM CLOSE KEYBOARD FILE
15160 XX$=CHR$(XX):REM CONVERT TO STRING
      CHARACTER OF KEY HIT
15165 REM PRINT XX$;
15170 TRAP 15190:REM SET TRAP BEFORE NUM
      ERIC CONVERSION
15175 XX=0:REM INITIALIZE TO ZERO INCASE
      NOT NUMERIC
15180 XX=VAL(XX$):REM SET XX AS THE DIGI
      T HIT - 0 IF NONE
15190 TRAP 34567:REM SPRING TRAP
15199 RETURN
```





## SOME USEFUL SUBROUTINES FOR WORD AND GUESSING GAMES

Here are two programs that demonstrate elementary operations on strings of letters which are basic to most word games. The first program shows how to choose a random letter of the alphabet and print it (A\$ is a string variable which is set = the letters of the alphabet, in order; a random number between 1 and 26 is chosen (line 30), and the character represented by this number (A\$(Y,Y)) is printed. (See explanation on preceding page.)

The second program shows how to choose a group of letters from within a string: Line 30 chooses only one letter, beginning (and ending) with the third character in the string (in this case, the letter "C"); line 40 prints the whole string, from the 1st to 10th characters.

```

5 REM GETTING RANDOM LETTER FROM LIST
10 DIM A$(26)
20 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
30 LET Y=INT(RND(1)*26)+1
40 PRINT A$(Y,Y)

```

```

5 REM HOW TO GET A LETTER OR
6 REM A GROUP OF LETTERS OUT OF A$
10 DIM A$(10)
20 A$="ABCDEFGHIJ"
30 PRINT A$(3,3)
40 PRINT A$(1,10)

```

## Big Letters

Here's a subroutine that lets you display big letters on the screen as you type them. It is especially good for young children.

```
10 REM *      <<< BIGLETS >>>
20 REM *  (C) 1980 by Len Lindsay
30 REM *  This routine shows how you
40 REM *  can get your ATARI to display
50 REM *  big letters on the screen
60 REM *  as you type them.  Especially
70 REM *  good for very young children
80 TRAP 160
90 REM *  For different size of
100 REM *  letters use GRAPHICS 1+16
110 GRAPHICS 2+16
120 OPEN #1,4,0,"K"
130 GET #1,KEY
140 PRINT #6;CHR$(KEY);
150 GOTO 130
160 TRAP 160;GOTO 130
```

## Part III

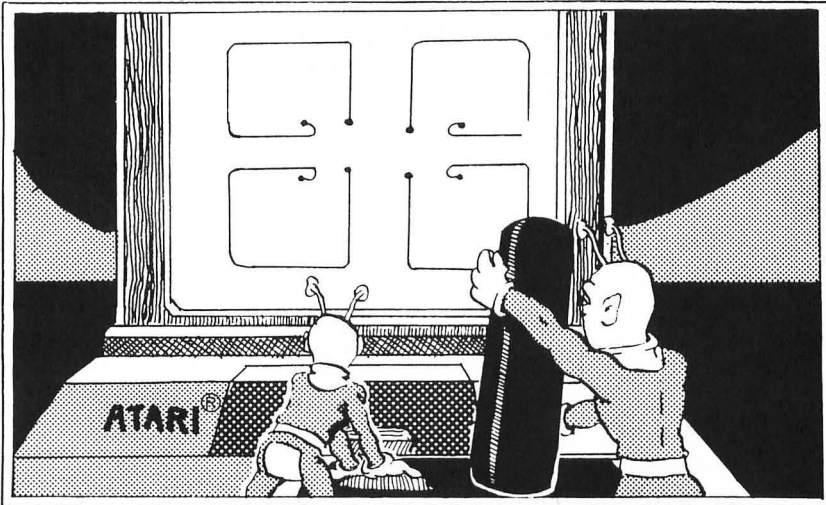
# The ATARI Special



Your ATARI 400 or ATARI 800 Home Computer has a number of special features that allow you to draw and create animations, control 128 different color and luminance combinations, and compose music using a range of 256 notes for up to four voices. Color, sound, and graphics are ATARI specials. In this section, we'll show you some of the possibilities of using and combining these three modes and provide some games that depend upon color, sound, and movement. After familiarizing yourself with our games and game programs, you should be able to modify our games and develop new ones. For additional information about sound, color, and graphics, consult your *ATARI 400/800 BASIC Reference Manual*.

## Chapter 6

# Graphics



ATARI BASIC allows you to use three different text modes (numbered 0-2) and any one of six different graphics modes (modes 3-8) for the ATARI 400 or 800 Home Computer. Each graphics mode, in effect, turns your TV screen into a graph paper on which you can draw. You'll notice there is also a small text window at the bottom of the screen which can be used to print instructions or input variables. Here is a summary of the text and graphics modes of the ATARI 400 and ATARI 800 Home Computers.

<i>GR. MODE</i>	<i>MODE TYPE</i>	<i>HORIZ. (COLUMNS)</i>	<i>SCREEN FORMAT</i>	
			<i>VERT. (ROWS) SPLIT SCREEN</i>	<i>VERT. (ROWS) FULL SCREEN</i>
0	TEXT	40	-	24
1	TEXT	20	20	24
2	TEXT	20	10	12
3	GRAPHICS	40	20	24
4	GRAPHICS	80	40	48
5	GRAPHICS	80	40	48
6	GRAPHICS	160	80	96
7	GRAPHICS	160	80	96
8	GRAPHICS	320	160	192

In Part I we introduced graphics commands such as PLOT and DRAW-TO and illustrated their use as we built up the Tortoise and Hare race. You may want to review that section before proceeding with the games and demonstrations in this chapter.



To use GRAPHICS 8, you must have a minimum of 16K of RAM memory in your ATARI Home Computer.

## SOME SIMPLE PLOTTING GAMES

Here are a few games that can help you become familiar with plotting points in different graphics modes. The first game simply allows you to name a point and then the computer plots it for you. This program uses GRAPHICS 3, but you should rewrite the program and try it with other graphics modes, as well:

```

5 REM PICK A POINT AND PLOT IT
10 GRAPHICS 3:COLOR 1
20 PRINT "PICK A POINT AND PLOT IT"
25 PRINT "X=";;INPUT X
30 PRINT "Y=";;INPUT Y
40 PLOT X,Y
50 GOTO 20

```

Because of the GOTO command at line 50, you can keep on plotting point after point. Try to draw pictures using this simple program. Start with a face or a simple figure.

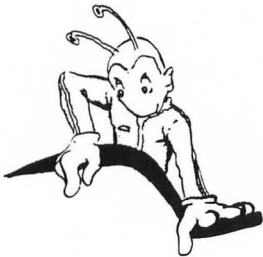
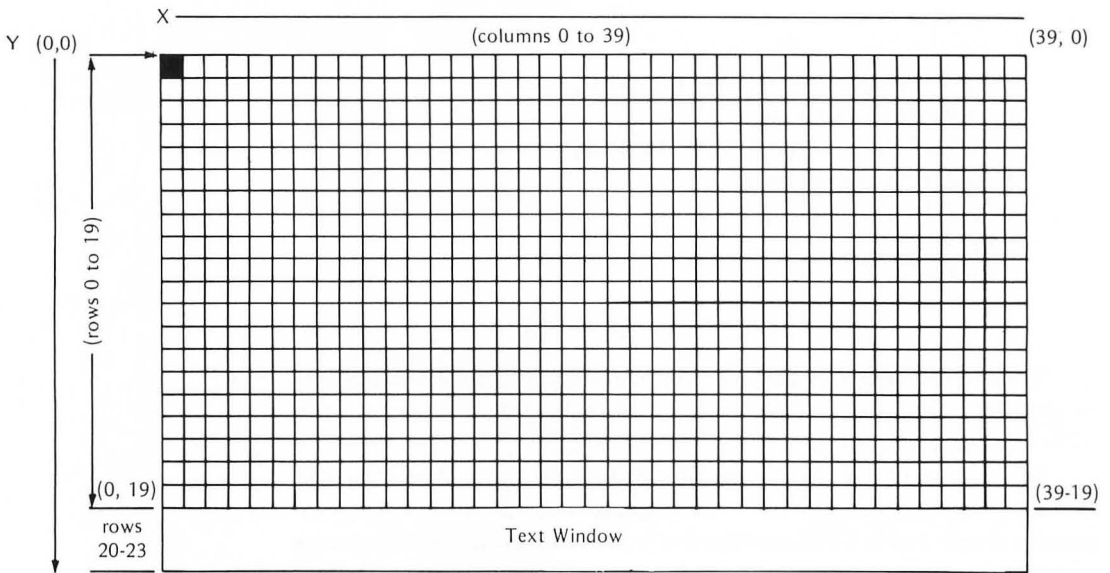
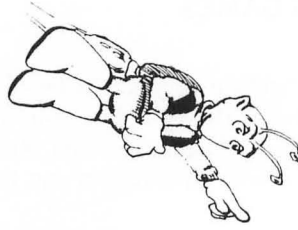
Here's a game that is the opposite of the one above. The computer picks a point, plots it, and asks you to guess which point it is. The program is simple but it takes considerable practice to guess the coordinates of a point.

```

10 REM SIMPLE GRAPHICS LEARNING GAME
20 GRAPHICS 3:COLOR 1
30 LET X=INT(RND(1)*39)
40 LET Y=INT(RND(1)*19)
50 PRINT "HERE IS A POINT. GUESS ITS COORDINATES."
60 PLOT X,Y
70 PRINT "X=";;INPUT W
75 PRINT "Y=";;INPUT Z
80 IF X=W AND Y=Z THEN PRINT "CORRECT":GOTO 100
90 PRINT "TAKE ANOTHER GUESS":GOTO 50
100 FOR WAIT=1 TO 200:NEXT WAIT:GOTO 20

```

Remember when trying this that the GRAPHICS 3 screen is arranged as shown here:



Here's another program to help you familiarize yourself with the graphics grid. The computer gives you the X and Y coordinates, and you have to guess where the point is. After a pause the computer plots the point so that you can correct yourself.

```


5 REM GUESS WHERE THE POINT IS
10 GRAPHICS 3:COLOR 1
20 LET X=INT(RND(1)*39)
30 LET Y=INT(RND(1)*19)
40 PRINT "GUESS WHERE THIS POINT IS"
50 PRINT X;"",Y
60 PRINT "POINT TO WHERE YOU THINK IT IS"
70 FOR M=1 TO 3500:NEXT M
80 PLOT X,Y
90 FOR N=1 TO 2500:NEXT N
100 PRINT "[CLEAR]"
110 GOTO 20

```



Here's a more sophisticated program for the same game. In this version, when you think you know where the point will be plotted, press any key on the keyboard. Press a key also to get the next point. If you wish to have a clear screen for each point, change line 80 to read:

```
80 GOTO 10
```

```
1 REM A SOPHISTICATED WAY TO , PROGRAM PLOT- 
   TING GUESSING GAMES
2 REM OPEN-GET LEARNING, TOO
5 OPEN #1,4,0, "K:"
10 GRAPHICS 3
15 COLOR 1
20 X=INT(RND(1)*39)
25 Y=INT(RND(1)*19)
27 ?X;",";Y
30 ? "GUESS WHERE IT WILL BE"
40 GET #1,Q
50 PLOT X,Y
60 GET #1,Z
70 ? "[CLEAR]"
80 GOTO 20
```



? is another form of the PRINT command in ATARI BASIC. We use it because it's shorter and saves typing.

## RECTCENT or GUESS THE CENTER

One of the best ways to learn more about your ATARI Home Computer is to write games that help you practice learning new concepts about programming.

This game, written by Len Lindsay, is a great way to practice learning the coordinate system used in the different ATARI GRAPHICS modes. The program only uses GRAPHICS 3, but you may wish to modify it so that you can practice the other graphics modes as well.

In this game, you try to guess the X (column) and Y (row) location of the center of a rectangle shown on the screen. The program has two modes of operation—in the first, “hints” or feedback are given after each guess so that you can narrow down the range of your guess. At the end of a successful guess, you may elect to turn off the hint or feedback “marks” by pressing the OPTION key. (The program also allows you to redo a guess, if you so desire.)

This program helps children (and adults) to learn to *estimate* a correct answer, based on a minimal number of clues. This skill is essential to successful problem solving, especially when orders of magnitude (“how close”) are more important than an exact or correct answer.

Another suggested modification of this game is to give the player different feedback if it takes fewer guesses to get the correct answer. Thus you might have different levels for different scores, such as:

1-2 guesses	Super Estimator
3-4 guesses	Surveyor
5-6 guesses	Novice
etc.	

```

10 REM *   <<< GUESS THE CENTER >>>
20 REM *   (C) 1980 Len Lindsay
30 REM *           Idea from
40 REM *   CALCULATOR/COMPUTERS
50 DIM XX$(3)
60 GRAPHICS 3
70 REM FOLLOWING WILL BE USED AS COLOR R
  EGISTER NUMBERS
80 LINE=1
90 CENTER=3
100 MARK=2
110 OFF=0
120 HINT=MARK
130 COLUMN=INT(RND(1)*34)+3
140 ROW=INT(RND(1)*16)+2

```

```

150 GOSUB 640:REM CORNERS
160 COLOR LINE
170 PLOT COLUMN-2,ROW-2
180 DRAWTO COLUMN+2,ROW-2
190 DRAWTO COLUMN+2,ROW+2
200 DRAWTO COLUMN-2,ROW+2
210 DRAWTO COLUMN-2,ROW-2
220 PRINT "[CLEAR]BELL]CAN YOU GUESS WH
ERE THE CENTER OF "
230 PRINT "THE RECTANGLE IS?"
240 PRINT "FIRST - WHAT COLUMN (3-36)";
250 GOSUB 680:REM INPUT ROUTINE
260 COLUMNGUESS=INT(XX+0.05)
270 IF COLUMNGUESS<3 OR COLUMNGUESS>36 T
HEN 220
280 COLOR HINT
290 PLOT COLUMNGUESS,0
300 PLOT COLUMNGUESS,19
310 PRINT "[CLEAR]BELL]NOW WHAT ROW IS
THE CENTER IN"
320 PRINT "(2-17) OR HIT OPTION TO REDO
COLUMN"
330 TRAP 310:POKE 764,255:REM SET TRAP A
ND CLEAR BUFFER
340 IF PEEK(53279)=3 THEN COLOR OFF:PLOT
COLUMNGUESS,0:PLOT COLUMNGUESS,19:GOTO
220
350 IF PEEK(764)=255 THEN 340
360 GOSUB 710:REM INPUT ROUTINE
370 ROWGUESS=INT(XX+0.05)
380 IF ROWGUESS<2 OR ROWGUESS>17 THEN 31
0
390 COLOR HINT
400 PLOT 0,ROWGUESS
410 PLOT 39,ROWGUESS
420 COLOR CENTER
430 PLOT COLUMNGUESS,ROWGUESS
440 IF COLUMNGUESS=COLUMN AND ROWGUESS=R
OW THEN 520
450 PRINT "[CLEAR]BELL]OOPS - LOOK AT W
HERE YOUR GUESS WAS."
460 PRINT "COLUMN ";COLUMNGUESS;" , ROW "
;ROWGUESS;" MISSED THE CENTER"
470 PRINT " HIT START TO TRY THE SAME ON
E AGAIN"

```

```
480 PRINT "OR FOR A NEW RECTANGLE HIT SE  
LECT";  
490 IF PEEK(53279)=6 THEN GOSUB 610:GOTO  
150  
500 IF PEEK(53279)=5 THEN GOSUB 610:GOTO  
130  
510 GOTO 490  
520 RIGHT=RIGHT+1  
530 PRINT "[CLEAR][BELL]YOU GOT IT - THA  
T MAKES ";RIGHT;" RIGHT"  
540 PRINT "HIT START TO TRY ANOTHER"  
550 PRINT "HIT OPTION TO TURN HINT MARKS  
";  
560 IF HINT=OFF THEN PRINT "ON"  
570 IF HINT=MARK THEN PRINT "OFF"  
580 IF PEEK(53279)=6 THEN GOSUB 610:GOTO  
130  
590 IF PEEK(53279)=3 THEN GOSUB 610:GOSU  
B 630:GOTO 130  
600 GOTO 580  
610 GRAPHICS 3  
620 RETURN  
630 HINT=MARK+OFF-HINT:REM SWITCH  
640 COLOR HINT  
650 PLOT 0,0:PLOT 0,19:PLOT 39,0:PLOT 39  
,19  
660 RETURN  
670 END  
680 REM INPUT SUBROUTINE WITH BEEP  
690 PRINT "[BELL]";  
700 POKE 764,255  
710 TRAP 710  
720 INPUT XX$  
730 TRAP 750  
740 XX=VAL(XX$)  
750 TRAP 34567  
760 RETURN
```

## ANIMATION

Not only can you draw pictures using the PLOT and DRAWTO commands, but you can also make your pictures move across the screen, change form, and generally become "animate." In fact, most animated cartoons these days are drawn using computer techniques similar to the ones we'll show you.

Here is a basic animation program. It moves a dot across your screen. Try it before going on to more complex animation.

```

10 REM *    <<< ANIMATION 1 >>>
20 REM *    moving dot
30 GRAPHICS 3
40 COLOR 2
50 FOR I=0 TO 39
60 GRAPHICS 3
70 PLOT I,10
80 GOSUB 110
90 NEXT I
100 END
110 FOR W=1 TO 300:NEXT W:RETURN

```

Note how this program works. The loop beginning at line 50 first sets the graphics mode, then plots a point, then goes to a pause subroutine, and then before plotting another point it resets the graphics mode, thus clearing the screen.

By resetting the graphics mode each time, you erase the last point plotted. That is what creates the illusion of a point moving across the screen. If you eliminate line 60 as below, you get a line growing across the screen, not a point moving:

```

10 GRAPHICS 3:COLOR 2
50 FOR I=1 TO 39
70 PLOT I,10
80 GOSUB 1000
90 NEXT I
100 END
1000 FOR W=1 TO 500:NEXT W:RETURN

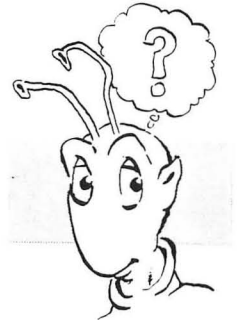
```

If you want to speed up the point or slow it down, all you have to do is modify the pause loop on line 1000. For example,

```

1000 FOR W=1 TO 10:NEXT W:RETURN

```



is fast, and

```
1000 FOR W=1 TO 5000:NEXT W:RETURN
```

is lethargic.

To make a face move across the screen, some simple modifications of the first program are all that need be done. Line 50 is modified to allow the boundaries of the face to be properly repeated in new positions. Lines 70-90 replace line 70 of the previous program to include more than one dot to be plotted (i.e., the face). Now try this program and see what happens.

```
10 REM *   <<< ANIMATION 2 >>>
20 REM *   draws moving face
30 GRAPHICS 3
40 COLOR 2
50 FOR I=4 TO 35
60 GRAPHICS 3
70 PLOT I-2,8:PLOT I+2,8
80 PLOT I,10
90 PLOT I-1,12:PLOT I,12:PLOT I+1,12
100 GOSUB 130
110 NEXT I
120 END
130 FOR W=1 TO 300:NEXT W:RETURN
```

Here's a dressed-up program that includes a border for the face:

```
10 REM *   <<< ANIMATION 3 >>>
20 REM *   face with border
30 FOR I=4 TO 74
40 GRAPHICS 5:COLOR 2
50 GOSUB 130
60 PLOT I-2,18:PLOT I+2,18
70 PLOT I,20
80 PLOT I-1,22:PLOT I,22:PLOT I+1,22
90 GOSUB 120
100 NEXT I
110 END
120 FOR W=1 TO 10:NEXT W:RETURN
130 REM BORDER
140 PLOT I-4,15:DRAWTO I+4,15
```

```

150 DRAWTO I+4,26:DRAWTO I-4,26
160 DRAWTO I-4,15
170 RETURN

```

Here's a fancier animation that involves a "collision," using GRAPHICS 5 mode:

```

10 REM *    <<< ANIMATION 4 >>>
20 REM * face collides with square
30 REM * moving in opposite direction
40 GRAPHICS 5
50 COLOR 2
60 FOR I=4 TO 34
70 GRAPHICS 5
80 GOSUB 250:REM DRAW BORDER TO FACE
90 PLOT I-2,18:PLOT I+2,18
100 PLOT I,20
110 PLOT I-1,22:PLOT I,22:PLOT I+1,22
120 GOSUB 300:REM DRAW THE SQUARE FROM THE OTHER SIDE
130 GOSUB 230:REM PAUSE
140 NEXT I
150 FOR J=1 TO 25
160 SOUND 0,INT(RND(1)*256),INT(RND(1)*8)*2,10
170 SETCOLOR 4,INT(RND(1)*16),INT(RND(1)*8)*2
180 COLOR 3:PLOT 36,22:DRAWTO INT(RND(1)*80),INT(RND(1)*40)
190 NEXT J
200 GRAPHICS 5:PRINT "THE REAL END"
210 END
220 REM * PAUSE
230 FOR W=1 TO 10:NEXT W:RETURN
240 REM * BORDER OF FACE
250 PLOT I-4,15:DRAWTO I+4,15
260 DRAWTO I+4,26:DRAWTO I-4,26
270 DRAWTO I-4,15
280 RETURN
290 REM * BOX MOVING LEFT
300 PLOT 70-I,18:DRAWTO 76-I,18
310 DRAWTO 76-I,26:DRAWTO 70-I,26
320 DRAWTO 70-I,18
330 RETURN

```

Using these examples you can experiment with your own mini-cartoons. The BASIC vocabulary used so far has been introduced in Part I of the book. Now we'll present some more complex examples of what can be done using some special GRAPHICS features of your ATARI Home Computer.



## MOVING MESSAGES

The following programs show ways to display a moving message, using your ATARI Home Computer's GRAPHICS 1 or GRAPHICS 2 (large text) mode.

The first one shows some simple animation, accomplished by continuous display of the message in pieces (see line 190).

```

10 REM * <<< MESSAGE BILLBOARD >>>
20 REM * (C) LEN LINDSAY, 1980
30 DIM MESSAGE$(200)
40 DIM SP$(200)
50 GRAPHICS 2
60 PRINT "ENTER MESSAGE, PLEASE"
70 INPUT MESSAGE$
80 SP$=" "
90 SP$(LEN(SP$)+1)=MESSAGE$
100 MESSAGE$=SP$
110 MESSAGE$(LEN(MESSAGE$)+1)=" "
120 ML=LEN(MESSAGE$)
130 POKE 752,1
140 PRINT "[CLEAR]";REM CLEAR TEXT WINDO
W
150 SETCOLOR 2,0,0
160 FOR P=1 TO ML
170 EL=P+19:IF EL>ML THEN EL=ML
180 POSITION 0,5
190 PRINT #6;MESSAGE$(P,EL)
200 FOR W=1 TO 99:NEXT W
210 NEXT P
220 GOTO 160

```

If you use GRAPHICS 1 or GRAPHICS 2, you must follow the word PRINT with the symbols "#6;" (followed by a string or text containing your message). As we explained earlier, "#6" is a name ATARI BASIC understands which refers to displaying large text on the graphics screen in these two special modes. Here are two examples:

```

4000 GRAPHICS 1
5000 PRINT #6;$MESSAGE
      or
4000 GRAPHICS 2
5000 PRINT #6;"MY MESSAGE"

```

(Note that in both of the above examples, there is a split screen; the top half is for the large text display, and the bottom "window" is for standard size text. PRINT #6 only refers to the top half of the screen.)

The next program, written by Bill Carris of Atari, is a very simple but entertaining way to introduce your ATARI computer to a friend. It uses a combination of some nice color and GRAPHICS 7 effects and then flashes your name (or your friend's name) in big letters. It also uses sound effects!

```

10 REM *          <<< GOODNAME >>>
20 REM *   By Bill Carris,  ATARI INC
30 PRINT "[CLEAR]"
40 DIM A$(9)
50 PRINT "ENTER NAME - UP TO 9 CHARACTER
S:"
60 INPUT A$
70 GRAPHICS 7+16
80 COLOR 1
90 Z=2:Y=5:X=155
100 PLOT X,Y
110 X=X+2*Z
120 SOUND 0,X,14,8
130 DRAWTO X,Y
140 Z=Z-(2*Z)
150 Y=Y+Z
160 DRAWTO X,Y
170 IF Z>0 THEN Z=Z+2
180 Z=Z-3
190 IF Y<70 THEN 110
200 FOR DD=1 TO 100:SOUND 3,DD,6,10
210 GRAPHICS 2+16:POSITION 3,8:PRINT #6;
"HELLO, ";A$:NEXT DD
220 FOR XYZ=1 TO 20:NEXT XYZ
230 GOTO 70

```

Note lines 70 and 210 in Bill's program. Line 70 reads:

```
70 GRAPHICS 7+16
```

and line 210 begins

```
210 GRAPHICS 2+16:
```

By adding 16 to the GRAPHICS modes in this fashion, the “split screen” is eliminated, thus giving a full screen for displaying large text or graphics designs. For more information on the use of this technique, refer to your *ATARI 400/800 BASIC Reference Manual*.

## SOME GEOMETRIC EXPERIMENTS

## Introducing DYNAMIC COMPUTER ART

Here are a number of programs that should give you a sense of the graphic possibilities of your ATARI Home Computer. We'll start with some squares:

```

10 REM *          <<< SQRVES >>>
20 REM *   (C) 1980 by Len Lindsay
30 REM * curved design made with lines
40 GRAPHICS 7+16
50 FOR X=0 TO 95 STEP 5
60 COLOR 1
70 PLOT 0,X:DRAWTO 1.5*X,95
80 COLOR 2
90 DRAWTO 143,95-X
100 COLOR 3
110 DRAWTO 143-1.5*X,0
120 COLOR 2
130 DRAWTO 0,X
140 NEXT X
150 GOTO 150

```

And now for circles and spiders, which is a more complex use of circular forms:

```

10 REM *          <<< CIRCLES >>>
20 REM *   (C) 1980 by Len Lindsay
30 TRAP 210
40 GRAPHICS 7+16:LUM=12
50 CENTERX=RND(1)*160
60 CENTERY=RND(1)*96
70 DISTANCE=RND(1)*25+25
80 FOR SHAPEX=0 TO DISTANCE
90 LUM=18-LUM:SETCOLOR 2,5,LUM
100 SHAPEY=SQR(DISTANCE*DISTANCE-SHAPEX*
SHAPEX)
110 FOR DIF=0.25 TO 0.75 STEP 0.25
120 COLOR DIF*4
130 PLOT CENTERX+SHAPEX*DIF,CENTERY+SHAP
EY*DIF

```

```

140 PLOT CENTERX-SHAPEX*DIF,CENTERY+SHAP
EY*DIF
150 PLOT CENTERX+SHAPEX*DIF,CENTERY-SHAP
EY*DIF
160 PLOT CENTERX-SHAPEX*DIF,CENTERY-SHAP
EY*DIF
170 NEXT DIF
180 NEXT SHAPEX
190 GOTO 50
200 END
210 TRAP 210:GOTO 10+PEEK(186)+PEEK(187)
*256

```

```

10 REM *      <<< SPIDERS >>>
20 REM *      a variation on circles
30 REM *      (C) 1980 by Len Lindsay
40 TRAP 220
50 GRAPHICS 7+16:LUM=12
60 CENTERX=RND(1)*160
70 CENTERY=RND(1)*96
80 DISTANCE=RND(1)*25+25
90 FOR SHAPEX=0 TO DISTANCE
100 LUM=18-LUM:SETCOLOR 2,5,LUM
110 SHAPEY=DISTANCE-SQR(DISTANCE*DISTANC
E-SHAPEX*SHAPEX)
120 FOR DIF=0.25 TO 0.75 STEP 0.25
130 COLOR DIF*4
140 PLOT CENTERX+SHAPEX*DIF,CENTERY+SHAP
EY*DIF
150 PLOT CENTERX-SHAPEX*DIF,CENTERY+SHAP
EY*DIF
160 PLOT CENTERX+SHAPEX*DIF,CENTERY-SHAP
EY*DIF
170 PLOT CENTERX-SHAPEX*DIF,CENTERY-SHAP
EY*DIF
180 NEXT DIF
190 NEXT SHAPEX
200 GOTO 60
210 END
220 TRAP 220:GOTO 10+PEEK(186)+PEEK(187)
*256

```

Now for some concentric scintillating diamonds of changing sizes and colors:

```

10 REM *      <<< DIAMONDS >>>
20 REM * (C) 1980 by Len Lindsay
30 LUM=12
40 GRAPHICS 7+16
50 TRAP 220:REM FOR OUT OF RANGE
60 CENTERX=RND(1)*160
70 CENTERY=RND(1)*96
80 DISTANCE=RND(1)*25+25
90 FOR SHAPEX=0 TO DISTANCE
100 LUM=18-LUM:SETCOLOR 2,5,LUM
110 SHAPEY=DISTANCE-SHAPEX
120 FOR DIF=0.25 TO 0.75 STEP 0.25
130 COLOR DIF*4
140 PLOT CENTERX+SHAPEX*DIF,CENTERY+SHAPEY*DIF
150 PLOT CENTERX-SHAPEX*DIF,CENTERY+SHAPEY*DIF
160 PLOT CENTERX+SHAPEX*DIF,CENTERY-SHAPEY*DIF
170 PLOT CENTERX-SHAPEX*DIF,CENTERY-SHAPEY*DIF
180 NEXT DIF
190 NEXT SHAPEX
200 GOTO 60
210 END
220 TRAP 220:GOTO 10+PEEK(186)+PEEK(187)
    *256

```

In this program you'll be seeing in triplicate. Don't worry, it's not your eyes.

```

10 REM *      <<< LINES >>>
20 REM *      by Len Lindsay
30 DIM X(3),Y(3)
40 GRAPHICS 7+16
50 FOR R=0 TO 3
60 X(R)=INT(RND(1)*160)
70 Y(R)=INT(RND(1)*96)
80 NEXT R
90 FOR R=0 TO 3
100 X=INT(RND(1)*160):Y=INT(RND(1)*96)

```

```

110 COLOR R
120 SETCOLOR 2,RND(1)*256,4:REM THIS
    COMMAND CHANGES THE LINE COLORS
130 PLOT X(R),Y(R)
140 DRAWTO X,Y
150 COLOR 0:PLOT X(R),Y(R)
160 NEXT R
170 GOTO 90

```

And for a little flash to the previous program, try this:

```

5 REM LINES2 (RANDOM LINES WITH CHANGING COLORS,
    GIVING FLASHING EFFECTS)
6 REM BY LEN LINDSAY
10 DIM X(3),Y(3)
20 GRAPHICS 7+16
100 FOR R=0 TO 3
110 X(R)=INT(RND(1)*160)
120 Y(R)=INT(RND(1)*96)
130 NEXT R
200 FOR R=0 TO 3
210 X=INT(RND(1)*160):Y=INT(RND(1)*96)
220 COLOR R
225 SETCOLOR 2,RND(1)*256,4:REM THIS COMMAND
    CHANGES THE LINE COLORS
230 PLOT X(R),Y(R)
240 DRAWTO X,Y
245 COLOR 0:PLOT X(R),Y(R)
250 NEXT R
300 GOTO 200

```

Finally, here is one more graphic design which you may enjoy.

```

10 REM *      <<< SPHERES >>>
20 REM *  (C) 1980 by Len Lindsay
30 TRAP 210
40 GRAPHICS 7+16:LUM=12
50 CENTERX=RND(1)*160
60 CENTERY=RND(1)*96
70 DISTANCE=RND(1)*25+25
80 FOR SHAPEX=0 TO DISTANCE
90 LUM=18-LUM:SETCOLOR 2,5,LUM
100 SHAPEY=SQR(DISTANCE*DISTANCE-SHAPEX*
    SHAPEX)

```

```
110 FOR DIF=0.25 TO 0.75 STEP 0.25
120 COLOR DIF*4
130 PLOT CENTERX+SHAPEX,CENTERY+SHAPEY*D
IF
140 PLOT CENTERX-SHAPEX,CENTERY+SHAPEY*D
IF
150 PLOT CENTERX+SHAPEX,CENTERY-SHAPEY*D
IF
160 PLOT CENTERX-SHAPEX,CENTERY-SHAPEY*D
IF
170 NEXT DIF
180 NEXT SHAPEX
190 GOTO 50
200 END
210 TRAP 210:GOTO 10+PEEK(186)+PEEK(187)
*256
```



## STRAIGHT LINES OR THERE'S MORE IN MATHEMATICS THAN MEETS THE EYES

The next program is a popular graphics demonstration program at ATARI. Written by programmers at ATARI shortly after the ATARI Home Computers were first introduced, this program shows what you can do using only straight lines! Who says mathematics has to be dull? Look especially carefully at the last segment, where color and motion tricks fool the eye! (Note: This program requires an ATARI 400 or ATARI 800 Home Computer with a minimum of 16K of RAM.)

```

10 REM *          <<< PRETTY >>>
20 REM *      Graphics demonstrations
30 REM *      by ATARI Software Engineers
40 GOSUB 1000
50 C=0:Q=1:SETCOLOR 1,5,5
60 DEG
70 XI=80:YI=50
80 GRAPHICS 7+16
90 PLOT XI,YI
100 FOR I=1 TO 1000 STEP 5
110 Q=Q+1
120 IF Q>3.5 THEN Q=1
130 COLOR Q
140 R=I/10
150 T=I
160 X=R*COS(T)
170 Y=R*SIN(T)
180 IF Y+YI<0 THEN 240
190 PLOT X+XI,Y+YI
200 X=(I+C)/16*COS(I+C+90)
210 Y=(I+C)/16*SIN(I+C+90)
220 DRAWTO X+XI,Y+YI
230 NEXT I
240 SETCOLOR 2,8,2
250 SETCOLOR 1,8,5
260 SETCOLOR 0,8,8
270 GOSUB 370
280 SETCOLOR 0,8,2

```

PRETTY, copyright © 1981, Atari, Inc. This program is one of several in a collection from the *Graphics/Sound Demonstration* disk (APX-20028), available from the Atari Program Exchange, P.O. Box 427, Sunnyvale, Ca. 94086. (Interesting information on color and sound effects is also provided in the manual for this disk.) This program is reprinted here with permission of Atari, Inc.

```

290 SETCOLOR 2,8,5
300 SETCOLOR 1,8,8
310 GOSUB 370
320 SETCOLOR 1,8,2
330 SETCOLOR 0,8,5
340 SETCOLOR 2,8,8
350 GOSUB 370
360 GOTO 240
370 FOR K=1 TO 13:NEXT K
380 RETURN
1000 REM      AL'S DEMO
1010 DEG
1020 GRAPHICS 8+16
1030 COLOR 1
1040 SETCOLOR 2,0,0
1050 FOR I=1 TO 360 STEP 5
1060 X=319*I/360
1070 Y=80+80*SIN(I)
1080 IF I>270 THEN 1100
1090 PLOT 0,0
1100 DRAWTO X,Y
1110 IF I<90 THEN 1130
1120 DRAWTO 319,159
1130 NEXT I
1140 GOSUB 6000
2000 REM      DELDOM
2010 DEG
2020 A=INT(1.9*160)
2030 GRAPHICS 8+16
2040 SETCOLOR 2,0,0
2050 FOR I=0 TO 160 STEP 5
2060 B=INT(I/2)
2070 COLOR 1
2080 PLOT 0,B
2090 DRAWTO I,160
2100 PLOT A,B
2110 DRAWTO A-I,160
2120 PLOT 0,160-B
2130 DRAWTO I,0
2140 PLOT A,160-B
2150 DRAWTO A-I,0
2160 NEXT I
2170 GOSUB 6000
3000 REM      RANDOM BOXES

```

```

3010 GRAPHICS 7+16
3020 FOR I=1 TO 32
3030 X1=INT(RND(0)*160)
3040 Y1=INT(RND(0)*96)
3050 X2=INT(RND(0)*160)
3060 Y2=INT(RND(0)*96)
3070 IF X1=X2 OR Y1=Y2 THEN 3030
3080 COLOR INT(RND(0)*3+1)
3090 PLOT X1,Y1
3100 DRAWTO X1,Y2
3110 DRAWTO X2,Y2
3120 DRAWTO X2,Y1
3130 DRAWTO X1,Y1
3140 NEXT I
3150 GOSUB 6000
4000 REM COLLAPSING BOXES
4010 TRAP 4170:J=0
4020 GRAPHICS 7+16
4030 X1=INT(RND(0)*80)
4040 Y1=INT(RND(0)*48)
4050 X2=X1+INT(RND(0)*80)
4060 Y2=Y1+INT(RND(0)*48)
4070 IF X1=X2 OR Y1=Y2 THEN 3030
4080 COLOR INT(RND(0)*3+1)
4090 PLOT X1,Y1
4100 DRAWTO X1,Y2
4110 DRAWTO X2,Y2
4120 DRAWTO X2,Y1
4130 DRAWTO X1,Y1
4140 X1=X1+2:Y1=Y1+2
4150 X2=X2-1:Y2=Y2-1
4160 GOTO 4080
4170 J=J+1:TRAP 4170
4180 GOSUB 6000
4190 IF J<4 THEN 4020
4200 TRAP 40000
5000 REM MOIRE
5010 MX=320:MY=192:MODE=8
5020 MX=MX-1:MY=MY-1
5030 GRAPHICS MODE+16
5040 IF MODE=8 THEN SETCOLOR 2,0,0
5050 FOR Y=MY TO 0 STEP -1
5060 IF PEEK(764)<>255 THEN RETURN
5070 Q=Y:GOSUB 5300

```

```
5080 PLOT 0,Y
5090 DRAWTO MX/2,MY
5100 NEXT Y
5110 FOR X=0 TO MX
5120 IF PEEK(764)<>255 THEN RETURN
5130 Q=X:GOSUB 5300
5140 PLOT X,0
5150 DRAWTO MX/2,MY
5160 NEXT X
5170 FOR Y=0 TO MY
5180 IF PEEK(764)<>255 THEN RETURN
5190 Q=Y:GOSUB 5300
5200 PLOT MX,Y
5210 DRAWTO MX/2+1,MY
5220 NEXT Y
5230 RETURN
5240 FOR X=MX TO 0 STEP -1
5250 Q=X:GOSUB 5300
5260 PLOT X,MY
5270 DRAWTO MX/2,MY/2
5280 NEXT X
5290 RETURN
5300 IF Q=125 THEN Q=13
5310 IF Q=155 THEN Q=11
5320 COLOR Q
5330 RETURN
5340 END
6000 FOR K=1 TO 500
6010 NEXT K:RETURN
```



## HIGH RESOLUTION DRAWING

The next program is an example of the excellent graphics capabilities of your ATARI Home Computer when it uses its highest graphics resolution capabilities (GRAPHICS 8). Written by Tandy Trower, this program draws a picture of an ATARI 400! (This program also requires a minimum of 16K RAM.)

```

10 REM * <<< ATARI 400 DRAWING >>>
20 REM *   by Tandy Trower (ATARI)
30 DIM Y$(1)
40 PRINT "[CLEAR]WOULD YOU BELIEVE THAT
THE ATARI ";PRINT "COMPUTER CAN DRAW ITS
OWN"? "ATARI 400 PORTRAIT";
50 INPUT Y$
60 IF Y$="N" THEN PRINT :PRINT "WELL, YO
U'RE IN FOR A SURPRISE..."
70 IF Y$<>"N" THEN PRINT :PRINT "WATCH A
N ARTISTIC GENIUS AT WORK..."
80 FOR WAIT=1 TO 1400:NEXT WAIT
90 DIM A$(10)
100 GRAPHICS 8
110 SETCOLOR 4,12,4
120 POKE 752,1
130 SETCOLOR 2,0,14:SETCOLOR 1,0,2
140 DL=PEEK(560)+PEEK(561)*256
150 D1=PEEK(DL+4)+PEEK(DL+5)*256
160 PX=16:PY=43
170 A$="ATAR 400"
180 FOR U=1 TO LEN(A$):D2=57344+((ASC(A$
(U,U))-32)*8):D3=D1+PY*40+PX+U-1:FOR Z=0
TO 7:POKE D3+Z*40,PEEK(D2+Z)
190 NEXT Z:NEXT U
200 COLOR 3:PLOT 162,44:DRAWTO 162,49:PL
OT 163,44:DRAWTO 163,49
210 COLOR 1
220 FOR I=1 TO 7:PLOT 117,42+I:DRAWTO 12
5,42+I:NEXT I
230 COLOR 2:PLOT 118,48:DRAWTO 121,45:DR
AWTO 124,48
240 PLOT 121,44:DRAWTO 121,48
250 COLOR 1:PLOT 112,40:DRAWTO 195,40:DR
AWTO 195,53:DRAWTO 112,53:DRAWTO 112,40
260 REM LARGE BOX

```

```
270 PLOT 86,17:DRAWTO 112,17:PLOT 200,17
:DRAWTO 226,17
280 DRAWTO 250,76:DRAWTO 62,76:DRAWTO 86
,17
290 PLOT 79,38:DRAWTO 108,38:PLOT 204,38
:DRAWTO 233,38
300 PLOT 86,17:DRAWTO 79,38:DRAWTO 62,76
310 PLOT 226,17:DRAWTO 233,38:DRAWTO 250
,76
320 PLOT 110,27:DRAWTO 201,27
330 DRAWTO 204,39:DRAWTO 204,68
340 DRAWTO 171,68:DRAWTO 171,62:DRAWTO 1
39,62:DRAWTO 139,68
350 DRAWTO 107,68:DRAWTO 107,39:DRAWTO 1
10,27
360 PLOT 139,68:DRAWTO 139,72:DRAWTO 171
,72:DRAWTO 171,68
370 Y=63:FOR I=0 TO 9 STEP 2:PLOT 140,Y+
I:DRAWTO 170,Y+I:NEXT I
380 X=112:FOR I=1 TO 90 STEP 3:PLOT X+I,
17:DRAWTO X+I,23:NEXT I
390 PLOT 81,27:DRAWTO 56,27
400 DRAWTO 28,122
410 DRAWTO 44,122:DRAWTO 46,125:DRAWTO 2
66,125:DRAWTO 268,122
420 DRAWTO 284,122
430 DRAWTO 256,27:DRAWTO 231,27
440 PLOT 28,122:DRAWTO 27,129
450 DRAWTO 82,129:DRAWTO 89,142:DRAWTO 2
21,142:DRAWTO 228,129
460 DRAWTO 285,129:DRAWTO 284,122
470 PLOT 27,129:DRAWTO 28,135
480 DRAWTO 82,135:DRAWTO 87,144:DRAWTO 2
23,144:DRAWTO 228,135
490 DRAWTO 283,135:DRAWTO 284,130
500 PLOT 85,132:DRAWTO 225,132
510 PLOT 53,80:DRAWTO 259,80
520 DRAWTO 268,122
530 PLOT 53,80:DRAWTO 44,122
540 PLOT 236,83:DRAWTO 244,122
550 PLOT 57,83:DRAWTO 236,83
560 PLOT 56,83:POSITION 47,122:POKE 765,
1:XIO 18,*6,0,0,"S:"
570 PLOT 53,135:DRAWTO 57,142
```

```

580 DRAWTO 85,142
590 PLOT 259,135:DRAWTO 255,142:DRAWTO 2
24,142
600 POKE 765,1:PLOT 84,132:POSITION 88,1
41:XIO 18,#6,0,0,"S:"
610 REM SELECT/OPTION ETC.
620 PLOT 240,83:DRAWTO 248,122
630 PLOT 239,82:PLOT 247,123
640 PLOT 244,83:DRAWTO 252,122
650 PLOT 256,83:DRAWTO 264,122
660 PLOT 245,83:DRAWTO 256,83
670 PLOT 246,89:DRAWTO 257,89
680 PLOT 246,91:DRAWTO 258,91
690 PLOT 247,92:DRAWTO 258,92
700 PLOT 248,97:DRAWTO 259,97
710 PLOT 250,107:DRAWTO 261,107
720 PLOT 250,108:DRAWTO 261,108
730 PLOT 251,114:DRAWTO 263,114
740 PLOT 252,116:DRAWTO 263,116
750 PLOT 252,117:DRAWTO 263,117
760 PLOT 253,122:DRAWTO 264,122
770 PLOT 252,100:DRAWTO 255,100
780 PLOT 251,101:DRAWTO 256,101
790 PLOT 251,102:DRAWTO 257,102
800 PLOT 252,103:DRAWTO 256,103
810 COLOR 4
820 PLOT 253,101:DRAWTO 254,101
830 X=59:Y=86:NK=15:L=6:GOSUB 1040
840 NC=9
850 X=71:Y=86:NK=13:L=1:GOSUB 1070
860 X=75:Y=93:NK=12:L=6:GOSUB 1040
870 X=183:Y=93:NK=2:L=1:GOSUB 1070
880 X=77:Y=100:NK=13:L=6:GOSUB 1040
890 X=185:Y=100:NK=4:L=1:GOSUB 1070
900 X=83:Y=107:NK=11:L=6:GOSUB 1040
910 X=167:Y=107:NK=3:L=1:GOSUB 1070
920 X=89:Y=114:NK=1:L=106:GOSUB 1040
930 X=57:Y=93:NK=1:L=10:GOSUB 1040
940 NC=13:X=57:Y=93:NK=1:L=1:GOSUB 1070
950 X=221:Y=93:NK=1:L=12:GOSUB 1040
960 X=59:Y=100:NK=1:L=10:GOSUB 1040
970 X=59:L=1:FL=1:GOSUB 1070
980 X=59:Y=107:NK=1:L=16:GOSUB 1040
990 X=59:L=1:NC=19:GOSUB 1070

```





## TEXT AND CHARACTER GRAPHICS

The last two programs in this chapter show how you can do graphics and animation without even using the special GRAPHICS modes on your ATARI Home Computer. Both of these programs use GRAPHICS 0, which is the standard text mode you use to write ATARI BASIC programs.


The first program shows how you can get interesting wave effects using the letters of a word. Try using your name or a word that may evoke motion or rhythm. Experiment and expand your imagination!

```

10 REM *    <<< SINEWAVE SCREEN >>>
20 REM *    (C) 1980 LEN LINDSAY
30 DIM XX$(10)
40 POKE 752,1
50 PRINT "[CLEAR]TYPE IN A WORD - UP TO
10 CHARACTERS"
60 GOSUB 180
70 L=6
80 FOR Z=0 TO 99 STEP 0.3
90 X=20-LEN(XX$)/2+(13*SIN(Z))
100 FOR LOOP=1 TO X
110 PRINT " ";
120 NEXT LOOP
130 PRINT XX$
140 L=L+0.1:IF L=19 THEN L=6
150 SETCOLOR 1,0,L
160 NEXT Z
170 RUN
180 PRINT "[BELL][BELL]";
190 POKE 764,255
200 TRAP 200
210 INPUT XX$
220 TRAP 240
230 XX=VAL(XX$)
240 TRAP 34567
250 RETURN

```

The second program, written by Tandy Trower, uses special *character graphics* symbols (using [CTRL] with letters on the keyboard), *sound effects* (be sure to turn up the volume on your TV speaker so you can hear this), and *clever animation* to create what might be called a “dynamic dictionary definition,” i.e., a definition that really *does* what it says! Be

sure to copy this program carefully, remembering that bracketed letters mean to hold down [CTRL] with the letter, and underlined letters require first pressing the ATARI symbol key, etc. 

```

10 REM *   <<< HELICOPTER >>>
20 REM * Demo of animation & graphics
30 REM * with basic (no GR. commands)
40 REM * by Tandy Trower (1980)
50 GRAPHICS 0:SETCOLOR 2,9,12:SETCOLOR 1
  ,0,0:SETCOLOR 4,0,6
60 S=50
70 DIM A$(20),B$(20),C$(20),D$(20),E$(20)
  )
80 POKE 752,1:PRINT "[CLEAR]"
90 X=13:Y=19
100 POSITION 2,2:PRINT "HELICOPTER: an a
 ircraft whose support":PRINT " in the ai
  r is derived from the "
110 PRINT " reaction of a stream of air
  driven":PRINT " downward by one or more
  lifting "
120 PRINT " rotors turning about substan
  tially":PRINT " vertical axes."
130 A$="[R][R][R][R][R][R][S][R][R][R][
  R][R][R]"
140 B$="      [F][M][M]_[J][N][N][N][N][F]
  "
150 C$="      [V] [C]_[I][M][M][M][M][C]"
160 D$="      [B][M][M][M]"
170 E$="      [M][M][M][M][M]"
180 POSITION X,Y:PRINT A$:POSITION X,Y+1
  :PRINT B$:POSITION X,Y+2:PRINT C$:POSITI
  ON X,Y+3:PRINT D$
190 POSITION X,Y+4:PRINT E$;
200 FOR T=0 TO 1000:NEXT T
210 FOR T=0 TO 300:NEXT T
220 R=20:S=240:M=1
230 FOR I=0 TO R:GOSUB 410:A$="      [S
  ]      ":POSITION X,Y:? A$
240 FOR D=0 TO S-(I*12):NEXT D
250 SOUND 1,220,6,M
260 A$="[R][R][R][R][R][R]+[R][R][R][R]
  [R][R]":POSITION X,Y:? A$:GOSUB 410:M=M+
  0.1:NEXT I

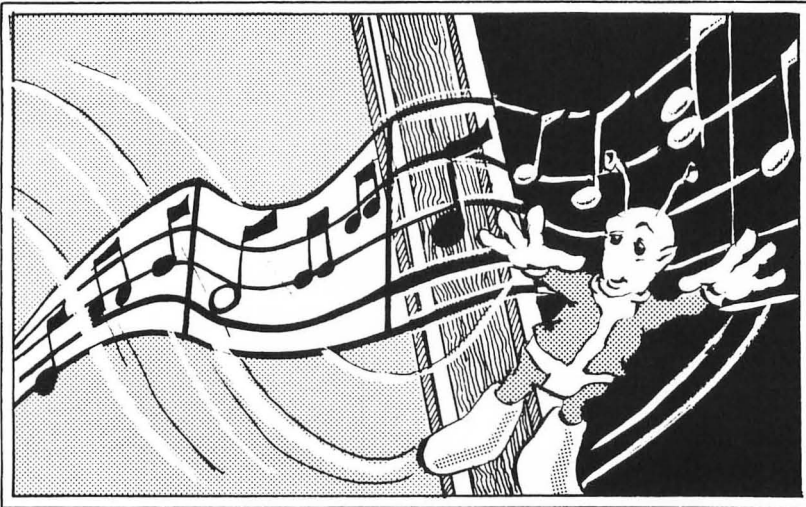
```





## Chapter 7

# Sound and Music





The ATARI BASIC SOUND command has four variables: SOUND X, Y, Z, W. The X variable allows you to select one of four voices. X ranges from 0 to 3, and each voice requires a separate SOUND statement.

The Y variable allows you to control 255 different pitches. It ranges from 1 to 255 and *the higher the value of the variable, the lower the pitch* of the sound produced. Middle C has a value of 121. Here's a chart of the pitch value of the ordinary 12-tone scale:

Table of Pitch Values for the Musical Scale

HIGH NOTES	C	29
	B	31
	A#-B <sup>b</sup>	33
	A	35
	G#-A <sup>b</sup>	37
	G	40
	F#-G <sup>b</sup>	42
	F	45
	E	47
	D#-E <sup>b</sup>	50
	D	53
	C#-D <sup>b</sup>	57
	C	60
	B	64
	A#-B <sup>b</sup>	68
	A	72
	G#-A <sup>b</sup>	76
	G	81
	F#-G <sup>b</sup>	85
	F	91
	E	96
	D#-E <sup>b</sup>	102
	D	108
	C#-D <sup>b</sup>	114
MIDDLE C	C	121
	B	128
	A#-B <sup>b</sup>	136
	A	144
	G#-A <sup>b</sup>	153
	G	162
	F#-G <sup>b</sup>	173
	F	182

LOW NOTES	E	193
	D#-E <sup>b</sup>	204
	D	217
	C#-D <sup>b</sup>	230
	C	243

The Z variable controls distortion and allows you to create sound effects, as some of the programs in this chapter will illustrate. Z can range from 0 to 14. Ten (10) will give you a pure tone, which you should use when composing music or other distortion-free sound programs. Even numbers are the only ones used for distortion control. Experiment with them and you can develop special effects, game accompaniments, and sound tracks to accompany your animations.

The fourth variable, W, is the volume control, and it ranges from a barely audible 1 to a loud 15. Eight (8) is considered a sensible medium setting for most sound programs. To turn the sound register off entirely, do SOUND X,0,0,0.

Here's a summary of the ranges of the four SOUND variables:

	<i>Voice</i>	<i>Pitch</i>	<i>Distortion</i>	<i>Volume</i>
SOUND	X,	Y,	Z,	W
	0-3	1-255	0-14 (even nos.) only)	1-15 (0 for rest or off)

Now let's play. The first few simple programs will experiment with pitch, the second variable.

## PITCH

Here are some random sounds that will give you a sense of the range of your ATARI Home Computer.

```
5 REM RANDOM SOUND
10 SOUND 0, RND(1)*255, 10, 12
20 FOR X=1 TO 200:NEXT X
30 GOTO 10
```

A more complex program follows that will illustrate the sound effects that can be produced by your ATARI. This program will cycle through the different sounds which your ATARI will produce using the SOUND command. It is important to remember that many more different tone qualities may be created by playing with the duration and volume, as well as by combining up to all four voices together—each with its own unique sound.

NOTE: Remember that in the ATARI SOUND command, the higher the number used for Pitch, the lower the actual frequency of the sound produced. These "pitches" range from 255 (lowest) to 1 (highest), with 0 being used to turn off a particular sound register.

```
10 REM <<< ATARI SOUND TEST >>>
20 REM * Set speed manually
30 REM * by Ted Kahn, 1980
40 TONE=0
50 PRINT "CLEAR":POSITION 14,0:?"ATARI SOUNDS"
60 ? :?"THIS PROGRAM RUNS THROUGH SOME OF THE":?"SOUNDS WHICH YOUR ATARI CAN MAKE."
70 ? "THE PROGRAM WILL LET YOU CHOOSE A":?"PAUSE, SO THAT YOU CAN HEAR THE RANGE"
80 ? "OF SOUNDS SLOWLY OR QUICKLY.":?" :?" :?" :?PAUSE=2000:GOSUB 280
90 GOSUB 240
100 POSITION 2,12
110 PRINT "PITCH","TONE"
120 POKE 752,1:REM TURN CURSOR OFF
130 FOR PITCH=255 TO 0 STEP -1
140 POSITION 2,14:PRINT PITCH," "
150 POSITION 12,14:PRINT TONE
160 SOUND 0,PITCH,TONE,8
```



```
170 GOSUB 280
180 NEXT PITCH
190 GOSUB 280
200 SOUND 0,0,0,0:POKE 752,0
210 TONE=TONE+2:IF TONE<=16 THEN 90
220 POKE 752,0:REM CURSOR BACK ON
230 END
240 PRINT "[CLEAR]";POSITION 2,8:PRINT "
SET PAUSE BETWEEN 0-1000"
250 PRINT "(0=NO PAUSE, 1000=LONG PAUSE)
";:INPUT PAUSE
260 PAUSE=INT(PAUSE):IF PAUSE<0 OR PAUSE
>1000 THEN 240
270 RETURN
280 REM PAUSE BETWEEN NOTES & TONES
290 FOR WAIT=0 TO PAUSE:NEXT WAIT
300 RETURN
```

Now for a few dressed-up versions of the first sound program with some visual effects added.

**THE DANCING DOT**

```

5 REM RANDOM SOUND
6 REM WITH DOT JUMPING AROUND THE SCREEN
10 SOUND 0,RND(1)*255, 10, 12
20 FOR X=1 TO 200:NEXT X
25 GRAPHICS 7:COLOR 1
26 PLOT RND(1)*158,RND(1)*78
30 GOTO 10

```

**NERVOUS LINES**

```

5 REM RANDOM SOUND
6 REM WITH LINES ALL AROUND
10 SOUND 0,RND(1)*255, 10, 12
20 FOR X=1 TO 200:NEXT X
25 GRAPHICS 7:COLOR 1
26 PLOT RND(1)*158,RND(1)*78
27 DRAWTO RND(1)*155,RND(1)*78
30 GOTO 10

```

Note: Try changing line 25 to be line 8. How does this change the effect?

**A RANDOM LINE/SOUND PATTERN**

(Notice that by changing around the loop the lines now remain on the screen.)

```

5 REM RANDOM SOUND
6 REM WITH LINES ALL AROUND
7 REM SPEEDED UP
8 REM BY CHANGING AROUND THE LOOP
9 REM THE LINES STAY ON THE SCREEN
10 GRAPHICS 7:COLOR 1
20 FOR X=1 TO 20:NEXT X
25 SOUND 0,RND(1)*255, 10, 12
26 PLOT RND(1)*158,RND(1)*78
27 DRAWTO RND(1)*155,RND(1)*78
30 GOTO 25

```

## A SIREN OR WHISTLE

Some simple effects:

```

10 REM *      <<< SOUND 1 >>>
20 FOR I=255 TO 1 STEP -4
30 SOUND 0,I,10,10
40 GOSUB 70
50 NEXT I
60 END
70 FOR W=1 TO 100:NEXT W:RETURN

```

## A ROCKET LIFT-OFF



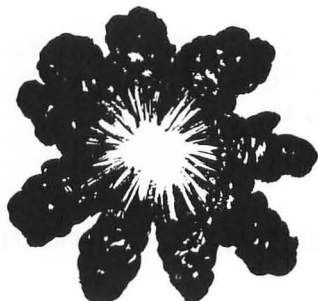
```

10 REM *      <<< SOUND PROGRAM 2 >>>
20 REM *      ROCKET LIFT OFF
30 FOR I=255 TO 1 STEP -4
40 SOUND 0,I,8,10
50 GOSUB 80
60 NEXT I
70 END
80 FOR W=1 TO 50:NEXT W:RETURN

```

## AN EXPLOSION

The reverse of lift-off:



```

10 REM *      <<< SOUND 3 >>>
20 REM * explosion (reverse rocket)
30 FOR I=1 TO 255 STEP 4
40 SOUND 0,I,8,10
50 GOSUB 80
60 NEXT I
70 END
80 FOR W=1 TO 5:NEXT W:RETURN

```

Now for an explosion followed by a simple melody:

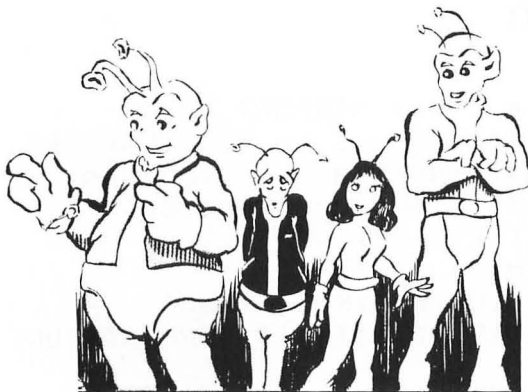
```

10 REM *      <<< SOUND 4 >>>
20 REM * explosion and melody
30 WAIT=50
40 FOR T=1 TO 20
50 SOUND 0,INT(RND(1)*256),10,10
60 GOSUB 140:NEXT T
70 SOUND 0,0,0,0
80 WAIT=3
90 FOR I=1 TO 255 STEP 4
100 SOUND 0,I,8,10
110 GOSUB 140
120 NEXT I
130 END
140 FOR W=1 TO WAIT:NEXT W:RETURN

```

## THE AIRPORT

```
10 REM *   <<< AIRPORT >>>
20 REM * Original by Bill Carris
30 REM * Modified by Ted Kahn
40 REM *   and Len Lindsay
50 DIM TEXT$(40)
60 GRAPHICS 2+16
70 SETCOLOR 0,8,RND(1)*256
80 POSITION 0,6:TEXT$="THE AIRPORT AT NI
GHT"
90 FOR LOOP=1 TO LEN(TEXT$)
100 PRINT #6;TEXT$(LOOP,LOOP);
110 FOR PAUSE=1 TO 10:NEXT PAUSE
120 NEXT LOOP
130 FOR X=227 TO 1 STEP -1
140 SOUND 0,X,8,12
150 FOR PAUSE=1 TO 16:NEXT PAUSE
160 NEXT X
170 FOR X=1 TO 200
180 SOUND 0,X,8,12
190 NEXT X
200 FOR PAUSE=1 TO 100:NEXT PAUSE
210 SOUND 0,0,0,0
220 RUN
```



## VOICE, DISTORTION, AND VOLUME

Here's a simple tune in four voices:

```

5 REM A SIMPLE QUARTET
10 FOR X=1 TO 50
20 SOUND 0, X, 10, 8
30 SOUND 1, X+2, 10, 8
40 SOUND 2, 100-X, 10, 8
50 SOUND 3, 2*X, 10, 8
60 FOR W=1 TO 50:NEXT W
70 NEXT X

```

Here's the tune dressed up with some jumping dots, followed by a simple program that illustrates how middle C is distorted by changing the Z variable.

```

5 REM A SIMPLE QUARTET
10 FOR X=1 TO 50
15 GRAPHICS 3:COLOR 1
16 PLOT RND(1)*5,RND(1)*5
17 PLOT RND(1)*5+2,RND(1)*5+2
18 PLOT RND(1)*30,RND(1)*19
19 PLOT RND(1)*25,RND(1)*19
20 SOUND 0, X, 10, 8
30 SOUND 1, X+2, 10, 8
40 SOUND 2, 100-X, 10, 8
50 SOUND 3, 2*X, 10, 8
60 FOR W=1 TO 50:NEXT W
70 NEXT X

```

```

10 REM EXPERIMENT WITH DISTORTION
20 FOR X=0 TO 14 STEP 2
30 SOUND 0, 121, X, 8
40 FOR Y=1 TO 1000:NEXT Y
50 NEXT X

```

This program illustrates the changes in volume that are possible. It also uses middle C:

```

10 REM CHANGES IN VOLUME
20 FOR W=1 TO 15
30 SOUND 0, 121, 10, W
40 FOR M=1 TO 200:NEXT M
50 NEXT W

```

You can write programs of your own by building on these few simple examples and what you know of graphics. Here are some other sound experiments:

Turn your ATARI Home Computer into *A Musical Keyboard* using this program:

```

10 REM * KEYBOARD ORGAN *****
20 REM * SIMULATED TOY ORGAN SOUNDS*
30 REM * (TO GET LOWER TONES, PRESS*
40 REM * [CAPS-LOWR] OR THE ATARI *
50 REM * "INVERSE VIDEO" KEY)      *
60 REM * (C) TED M. KAHN, 1980      *
70 REM *****
80 OPEN #1,4,0,"K"
90 PRINT "[CLEAR]PRESS ANY KEY ON THE KE
YBOARD TO GET":PRINT "A NEW TONE."
100 GET #1,N
110 FOR I=1 TO 16 STEP 0.5
120 SOUND 1,N,10,I
130 NEXT I
140 GOTO 100

```

Now you can make it sound like *A Toy Piano*.

```

10 REM * KEYBOARD PIANO *****
20 REM * SIMULATED PIANO SOUNDS *
30 REM * (JUST PRESS ANY KEY)    *
40 REM * (C) TED M. KAHN, 1980  *
50 REM *****

```



```

60 OPEN #1,4,0,"K"
70 PRINT "[CLEAR]PRESS ANY KEY ON THE KE
YBOARD TO GET":PRINT "TOY PIANO SOUNDS."
80 GET #1,N
90 FOR I=16 TO 0 STEP -1
100 SOUND 1,N,10,I
110 NEXT I
120 GOTO 80

```

Here is a program that will give you chords:

```

10 REM * KEYBOARD CHORDS *****
20 REM * TO GET CHORDS, PRESS THE *
30 REM * 'START','SELECT' OR *
40 REM * 'OPTION' KEYS. *
50 REM * (C) LEN LINDSAY, 1980 *
60 REM *****
70 PRINT "[CLEAR]PRESS THE 'START', 'SEL
ECT' OR":PRINT "'OPTION' KEYS FOR CHORDS
"
80 FOR Z=1 TO 3:SOUND Z,0,10,0:NEXT Z
90 N=PEEK(53279):IF N=7 THEN 90
100 RESTORE 200+N
110 FOR Z=1 TO 3:READ A:SOUND Z,A,10,6
120 NEXT Z
130 GOTO 90
203 DATA 162,108,128
205 DATA 182,121,144
206 DATA 243,96,162

```



## CHORD BUILDER

This program, created by Ted Kahn, allows you to build your own chords and experiment with harmonies. You'll need a joystick, which should be put in the first jack on the left of your computer. Four lines will appear on the screen, and by using the joystick to move the cursor along the line, you can control the pitch. Press the red button when you want to fix a sound on one line. Then move to the next line to choose another sound. You can build up two-, three-, or four-note chords using this program. It's also an excellent program for those learning about harmony and music theory.

```

10 REM * <<< CHORD BUILDER >>>
20 REM * (C) 1980 By Ted M. Kahn
30 POKE 82,1
40 POKE 752,1:REM CURSOR OFF
50 DIM H$(40),SCALE$(40),PITCH(37)
60 DIM BLANKLINE$(37),LAST(4)
70 REM INITIALIZE CONSTANTS
80 H$="1[W][S][W][S][S][W][S][W][S][W][S]
  2[W][S][W][S][S][W][S][W][S][W][S]3[W][S][W][S][S][W][S][W][S]4"
90 FOR I=1 TO 4:LAST(I)=1:NEXT I
100 SCALE$="C D EF G A BC D EF G A BC D
  EF G A BC "
110 BLANKLINE$=""
    ""
120 RESTORE 340
130 FOR I=1 TO 37
140 READ P:PITCH(I)=P
150 NEXT I
160 GOSUB 570:REM INSTRUCTIONS
170 FOR Y=1 TO 4
180 GOSUB 380
190 NEXT Y
200 REM POSITION JOYSTICK
210 FOR VOICE=0 TO 3 STEP 1
220 MOVE=0:X=LAST(VOICE+1)
230 POSITION X,20-(VOICE*5)
240 SOUND VOICE,PITCH(LAST(VOICE+1)),10,
  6
250 IF STRIG(0)=0 THEN 310
260 POSITION X,20-(VOICE*5):PRINT "^";:G
  OSUB 500:PRINT "[LEFT] ";:GOSUB 500

```

```

270 IF STICK(0)=7 AND X<37 THEN X=X+1:PR
INT "[RIGHT]";GOSUB 440:MOVE=MOVE+1:IF M
OVE=1 THEN GOSUB 520
280 POSITION X,20-(VOICE*5):PRINT "^";G
OSUB 500:PRINT "[LEFT] ";:GOSUB 500
290 IF STICK(0)=11 AND X>1 THEN X=X-1:PR
INT "[LEFT]";GOSUB 440:MOVE=MOVE+1:IF MO
VE=1 THEN GOSUB 520
300 GOTO 250
310 POSITION X,20-(VOICE*5):PRINT "[T]";
:GOSUB 470:LAST(VOICE+1)=X
320 NEXT VOICE
330 POP :GOTO 200
340 DATA 243,230,217,204,193,182,172,162
,153,144,136,128
350 DATA 121,114,108,102,96,91,85,81,76,
72,68,64
360 DATA 60,57,53,50,47,45,42,40,37,35,3
3,31
370 DATA 29
380 REM DRAW SCALE LINE
390 POSITION 1,(Y*5)-3
400 PRINT "VOICE ";CHR$(53-Y+128)
410 PRINT SCALE$
420 POSITION 1,(Y*5)-1:PRINT H$
430 RETURN
440 REM SOUND & MOVE POINTER
450 SOUND VOICE,PITCH(X),10,6
460 RETURN
470 REM WAIT FOR SYNCHRONIZATION
480 FOR WW=1 TO 300:NEXT WW
490 RETURN
500 FOR W1=1 TO 5:NEXT W1
510 RETURN
520 REM ERASE PREVIOUS NOTE
530 POSITION 1,20-(VOICE*5)
540 PRINT BLANKLINE$;
550 POSITION X,20-(VOICE*5)
560 RETURN
570 REM INSTRUCTIONS FOR USE
580 PRINT "[CLEAR]Use a joystick in port
#1 to change":PRINT "notes. Press trig
ger to move to next"

```

```
590 PRINT "voice.":PRINT :PRINT "PRESS S  
TART WHEN READY"  
600 IF PEEK(53279)<>6 THEN 600  
610 PRINT "[CLEAR]":POSITION 14,0  
620 PRINT "CHORD BUILDER"  
630 RETURN
```

## COLOR AND SOUND COMBINATIONS

Here's another demonstration of the mixing of simple color and sound effects. You can probably modify the graphics to make them more interesting!

```

0 REM *    <<< COLOR & SOUND >>>
5 REM * From collection at ATARI
10 N1=0:N2=0:GOSUB 1000
20 GOSUB 2000
30 GOSUB 3000
40 GOSUB 1000
50 GOSUB 2000
60 N2=61:GOSUB 3000
70 N2=46:GOSUB 1000
80 GOSUB 2000
90 GOSUB 3000
100 N1=48:N2=41:GOSUB 1000
110 N1=46:N2=36:GOSUB 2000
120 N1=41:N2=34:GOSUB 3000
130 N1=36:N2=31:GOSUB 1000
140 GOSUB 2000
150 GOSUB 3000
160 N1=48:N2=41:GOSUB 1000
170 GOSUB 2000
180 GOSUB 3000
190 N1=46:N2=36:GOSUB 1000
200 GOSUB 2000
210 N1=48:GOSUB 3000
220 N1=54:N2=34:GOSUB 1000
230 N1=61:N2=36:GOSUB 2000
240 N1=69:N2=41:GOSUB 3000
250 N1=73:N2=46:GOSUB 1000
260 GOSUB 2000
270 N1=69:N2=54:GOSUB 3000
280 SOUND 0,183,10,8
290 SOUND 1,82,10,8
300 SOUND 2,69,10,8
310 SOUND 3,61,10,8
320 FOR W=1 TO 50:NEXT W
330 N1=61:N2=0:N3=0:GOSUB 3000
340 N1=73:N2=46:GOSUB 1000

```

COLOSOUN, copyright © 1981, Atari, Inc. This program is one of several in a collection from the *Graphics/Sound Demonstration* disk (APX-20028), available from the Atari Program Exchange, P.O. Box 427, Sunnyvale, Ca. 94086. (Interesting information on color and sound effects is also provided in the manual for this disk.) This program is reprinted here with permission of Atari, Inc.

```

350 GOSUB 2000
360 N1=46:N2=0:GOSUB 3000
370 N1=61:N2=41:GOSUB 1000
380 N1=46:N2=36:GOSUB 2000
390 N1=41:N2=34:GOSUB 3000
400 N1=36:N2=31:GOSUB 1000
410 GOSUB 2000
420 N1=46:GOSUB 3000
430 N1=48:N2=41:GOSUB 1000
440 GOSUB 2000
450 N1=0:N2=36:GOSUB 3000
460 N1=54:N2=34:GOSUB 1000
470 GOSUB 2000
480 N1=0:GOSUB 3000
490 N1=61:N2=34:GOSUB 1000
500 N2=36:GOSUB 2000
510 N1=69:N2=41:GOSUB 3000
520 N1=73:N2=61:GOSUB 1000
530 N2=46:GOSUB 2000
540 N1=82:N2=61:GOSUB 3000
550 SOUND 0,183,10,8
560 SOUND 1,92,10,8
570 SOUND 2,46,10,8
580 FOR W=1 TO 75:NEXT W
590 RUN
1000 SOUND 0,183,10,8
1010 SOUND 1,N1,10,8
1020 GRAPHICS 3+16:COLOR 1
1030 SETCOLOR 0,8,2
1040 PLOT 0,12:DRAWTO 5,12
1050 SOUND 2,N2,10,8
1060 FOR W=1 TO 25:NEXT W:RETURN
2000 SOUND 0,137,10,8
2010 SOUND 1,N1,10,8
2020 GRAPHICS 3+16:COLOR 1
2030 SETCOLOR 0,1,8
2040 PLOT 0,12:DRAWTO 10,12
2050 SOUND 2,N2,10,8
2060 FOR W=1 TO 25:NEXT W:RETURN
3000 SOUND 0,145,10,8
3010 SOUND 1,N1,10,8
3020 GRAPHICS 3+16:COLOR 1
3030 SETCOLOR 0,4,2
3040 PLOT 0,12:DRAWTO 15,12
3050 SOUND 2,N2,10,8
3060 FOR W=1 TO 25:NEXT W:RETURN

```

## JAZZ—COMBINING COLOR AND SOUND

This program, done by programmers at Atari, shows how two concepts can be combined. It makes use of the same type of rectangular patterns as the ALBERS program we will introduce in Chapter Eight, but it adds a blues chord progression and the ability to vary the speed (tempo). The smaller the number (between one and nine), the faster the tempo.

```

10 REM *          <<< JAZZ >>>
20 REM * by Software Group at ATARI
30 OPEN #3,4,0,"K:";GRAPHICS 17
40 CHORD=1:THNOT=1:PTR=1
50 DIM COLPTR(4)
60 ? #6;? #6;"      ENTER TEMPO ";
70 IF PEEK(764)=255 THEN 70
80 GET #3,T:PRINT #6;T-48:GET #3,O:CLOSE
  #3:TEMPO=T-48
90 DIM BASE(3,4)
100 DIM LOW(3)
110 DIM LINE(16)
120 DIM JAM(3,7)
130 DIM COLS(7)
140 DIM LUMS(7)
150 FOR X=1 TO 3
160 FOR Y=1 TO 4
170 READ A:BASE(X,Y)=A
180 NEXT Y
190 NEXT X
200 FOR X=1 TO 3:READ A:LOW(X)=A
210 NEXT X
220 FOR X=1 TO 16:READ A:LINE(X)=A:NEXT
  X
230 FOR X=1 TO 3
240 FOR Y=1 TO 7
250 READ A:JAM(X,Y)=A:NEXT Y:NEXT X
260 FOR X=1 TO 7:READ A:COLS(X)=A:NEXT X
270 FOR X=1 TO 7:READ A:LUMS(X)=A:NEXT X
280 FOR X=1 TO 4:COLPTR(X)=1:NEXT X

```

JAZZ, copyright © 1981, Atari, Inc. This program is one of several in a collection from the *Graphics/Sound Demonstration* disk (APX-20028), available from the Atari Program Exchange, P.O. Box 427, Sunnyvale, Ca. 94086. (Interesting information on color and sound effects is also provided in the manual for this disk.) This program is reprinted here with permission of Atari, Inc.

```

290 GOSUB 810
300 GOSUB 470
310 T=T+1
320 GOSUB 350
330 GOSUB 750
340 GOTO 300
350 REM PROCESS HIGH STUFF
360 IF RND(0)<0.25 THEN RETURN
370 IF RND(0)<0.5 THEN 420
380 NT=NT+1
390 IF NT>7 THEN NT=7
400 GOSUB 980
410 GOTO 450
420 NT=NT-1
430 IF NT<1 THEN NT=1
440 GOSUB 1040
450 SOUND 2,JAM(CHORD,NT),10,NT*2
460 RETURN
470 REM PROCESS BASE STUFF
480 IF BASS=1 THEN 550
490 BDUR=BDUR+1
500 IF BDUR<>TEMPO THEN 520
510 BASS=1:BDUR=0
520 SOUND 0,LOW(CHORD),10,4
530 SOUND 1,BASE(CHORD,THNOT),10,4
540 RETURN
550 SOUND 0,0,0,0
560 SOUND 1,0,0,0
570 BDUR=BDUR+1
580 IF BDUR<>1 THEN 660
590 BDUR=0:BASS=0
600 THNOT=THNOT+1
610 IF THNOT<>5 THEN 660
620 THNOT=1
630 PTR=PTR+1
640 IF PTR=17 THEN PTR=1
650 CHORD=LINE(PTR)
660 RETURN
670 DATA 162,144,136,144,121,108,102,108
,108,96,91,96
680 DATA 243,182,162
690 DATA 1,1,1,1,2,2,2,2,1,1,1,1,3,2,1,1
700 DATA 60,50,47,42,40,33,29
710 DATA 60,50,45,42,40,33,29

```

```

720 DATA 81,68,64,57,53,45,40
730 DATA 0,2,4,8,12,15,0
740 DATA 0,2,2,2,2,2,12
750 REM SET COLOR POINTERS TO COLORS
760 FOR X=1 TO 3
770 SETCOLOR X-1,COLS(COLPTR(X)),LUMS(CO
LPTR(X))
780 NEXT X
790 SETCOLOR 4,COLS(COLPTR(4)),LUMS(COLP
TR(4))
800 RETURN
810 GRAPHICS 7+16
820 COLOR 2
830 FOR X=20 TO 140
840 PLOT X,20
850 DRAWTO X,70
860 NEXT X
870 COLOR 1
880 FOR X=40 TO 120
890 PLOT X,30
900 DRAWTO X,60
910 NEXT X
920 COLOR 0
930 FOR X=60 TO 100
940 PLOT X,40
950 DRAWTO X,50
960 NEXT X
970 RETURN
980 REM SHIFT COLORS OUT
990 FOR X=4 TO 2 STEP -1
1000 COLPTR(X)=COLPTR(X-1)
1010 NEXT X
1020 COLPTR(1)=NT
1030 RETURN
1040 FOR X=1 TO 3
1050 COLPTR(X)=COLPTR(X+1)
1060 NEXT X
1070 COLPTR(4)=NT
1080 RETURN

```





## THE TERRORS OF WAR (SIGHTS AND SOUNDS)

This last sound program is a collection of visual and sound effects which may be incorporated into games or other diversions. It also demonstrates that animation doesn't always have to be complex—it's the suggestion that counts. It is also an example of mixed media—color, sound, animation . . . the works!

```

10 REM * <<< THE HORRORS OF WAR >>>
20 REM * Sound and visual effects
30 REM * from ATARI
40 DIM A$(10)
50 POKE 752,0
60 R=INT(RND(0)*4)+1
70 ON R GOTO 90,250,400,560
80 GOTO 60
90 REM * BOMB *
100 GRAPHICS 3+16
110 FOR N=20 TO 125
120 SOUND 0,N,10,10:X=INT(N/6)
130 COLOR 0:PLOT 10,X-1
140 COLOR 1:PLOT 10,X
150 FOR Z=1 TO 15:NEXT Z
160 NEXT N
170 COLOR 0:PLOT 10,X
180 FOR N=0 TO 15
190 SETCOLOR 4,4,15-N
200 SOUND 0,20,0,15-N
210 SOUND 1,227,6,15-N
220 FOR Z=1 TO 30:NEXT Z
230 NEXT N
240 GOTO 80
250 REM * HELICOPTER *
260 GRAPHICS 0
270 POKE 752,1
280 A$="HELICOPTER"
290 FOR N=1 TO 25
300 POSITION 28-N,10:PRINT #6;A$
310 SOUND 0,10,0,12
320 FOR Z=1 TO 5:NEXT Z
330 SOUND 0,0,0,0
340 FOR Z=1 TO 20:NEXT Z
350 SOUND 1,220,6,8
360 POSITION 28-N,10:PRINT #6;"

```

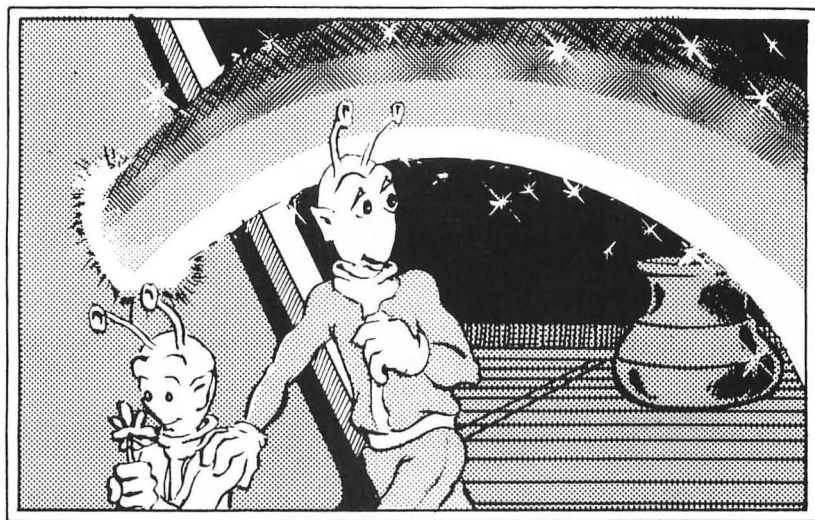
```

"
370 NEXT N
380 SOUND 1,0,0,0
390 GOTO 80
400 REM * MACHINE GUN *
410 GRAPHICS 7+16
420 FOR N=1 TO 40
430 FOR Z=0 TO 15 STEP 3
440 SOUND 0,60,6,15-Z
450 NEXT Z
460 SOUND 0,0,0,0
470 FOR Z=1 TO 5:NEXT Z
480 Y=INT(RND(0)*6)
490 IF Y1>85 OR Y1<5 THEN Y1=40
500 IF Y>3 THEN Y1=Y1+Y
510 IF Y<=3 THEN Y1=Y1-Y
520 COLOR 1:PLOT 4*N-1,Y1
530 NEXT N
540 Y1=40
550 GOTO 80
560 REM * EXPLOSIONS *
570 GRAPHICS 7+16
580 FOR N=0 TO 15
590 SETCOLOR 4,6,15-N
600 SOUND 0,100,0,15-N
610 FOR Z=1 TO 15:NEXT Z
620 NEXT N
630 FOR Z=1 TO 20:NEXT Z
640 FOR N=0 TO 15
650 SETCOLOR 4,10,6
660 SOUND 0,80,0,15-N
670 FOR Z=1 TO 5:NEXT Z
680 NEXT N
690 FOR N=1 TO 15
700 SETCOLOR 4,2,15-N
710 SOUND 0,150,0,15-N
720 FOR Z=1 TO 20:NEXT Z
730 SETCOLOR 4,8,15-N
740 NEXT N
750 GOTO 80

```

## Chapter 8

### Color



The mastery of the use of color for the ATARI 400 and ATARI 800 computers is somewhat more difficult than the mastery of sound and graphics. We'll introduce you to some ways you can use and control color in your own program, but you should consult the *ATARI 400/800 BASIC Reference Manual* and other sources if you want to go more deeply into programming in color.

We suggest that before experimenting with color yourself you type in (and save on cassette or disk) the following program which shows off the 128 colors which the ATARI Home Computer is capable of generating. Each of the 16 available colors (numbered 0-15) has eight variable "luminance" levels or intensities. By using the special function keys (SELECT, OPTION, and START), you can see how large text letters appear in these different colors, and what the background color looks like in each various shade (with SETCOLOR 4 for this example). Lines 200-330 may be eliminated after you've used the program a few times. You should choose either GRAPHICS mode 1 or 2, and replace M in line 60 with the specific GRAPHICS mode you want.

```

10 REM *      <<< COLORLET >>>
20 REM * Demo of 128 ATARI colors
30 REM * using large text & background
40 REM (C)1980 LEN LINDSAY & TED KAHN
50 GOSUB 200
60 GRAPHICS M
70 PRINT #6;"CAPS"
80 PRINT #6;"REVERSE CAPS"
90 PRINT #6;"lower case"
100 PRINT #6;"reverse lower "
110 FOR XR=0 TO 4
120 PRINT "[CLEAR]THIS IS SETCOLOR ";XR;
    "... "
130 PRINT "HIT OPTION TO CHANGE COLOR LU
    MINANCE"
140 PRINT "HIT SELECT TO CHANGE COLOR"
150 PRINT "HIT START TO DO NEXT COLOR RE
    GISTER";
160 GOSUB 340
170 FOR Z=1 TO 99:NEXT Z
180 NEXT XR

```

```


190 GOTO 60
200 REM ** CHOOSE LETTER SIZE **
210 PRINT "[CLEAR]THIS IS A DEMONSTRATIO
N OF"
220 PRINT "          ATARI COLORS"
230 PRINT "USING A LARGE OR EXTRA LARGE
ALPHABET"
240 PRINT :PRINT "IT SHOWS HOW TEXT TYP
ED IN (GR,0) AS":? "          * CAPITAL LETT
ERS"
250 PRINT "          * REVERSE CAPITAL LETTE
RS"
260 PRINT "          * lower case letters"
270 PRINT " and * reverse lower case le
tters"
280 PRINT :PRINT "ARE ALL CAPITAL LETTER
S AND":? "CHANGE COLORS WHEN DISPLAYED I
N"
290 PRINT "GRAPHICS MODE 1 OR 2.":PRINT
:? "(IT ALSO SHOWS HOW THE BACKGROUND CA
N CHANGE COLOR)"
300 PRINT :PRINT :? "CHOOSE A GRAPHICS M
ODE FOR TEXT SIZE":? "(1=LARGE, 2=EXTRA
LARGE)";
310 TRAP 300
320 INPUT M:IF M<1 OR M>2 THEN 300
330 RETURN
340 A=PEEK(53279):IF A=7 THEN 340:REM NO
NE HIT YET
350 IF A=6 THEN 430:REM **NEW COLOR REGI
STER (RETURN)**
360 IF A=5 THEN XC=XC+1:IF XC>15 THEN XC
=0:REM ** NEW COLOR **
370 IF A=3 THEN XL=XL+2:IF XL>14 THEN XL
=0:REM ** NEW LUMINANCE LEVEL **
380 SETCOLOR XR,XC,XL
390 POSITION 1,8:PRINT #6;"setcolor
";
400 POSITION 10,8:PRINT #6;XR;",";XC;",";
;XL
410 IF PEEK(53279)<>7 THEN 410
420 GOTO 340
430 POSITION 10,8:PRINT #6;"          ":RE
TURN

```



There are two color commands in ATARI BASIC:

COLOR X  
SETCOLOR Y, Z, W

These commands do different things according to which GRAPHICS mode you are in. When you turn on your computer you are in GRAPHICS mode 0 and that is the mode commonly used for programming. In fact, the text window in GRAPHICS 1 through 8 stays in mode 0 even though the rest of the screen can be used for drawing or creating large letters.

In GRAPHICS 0 the letters are white, the background is blue, and the border is black. By using the SETCOLOR Y, Z, W command you can change the colors of the letters, the background, and the border. 

In SETCOLOR Y, Z, W, the Y instructs the computer to change the color of either the background or the border. If Y has a value of 2, you will change the color of the background. If Y=4, you will change the color of the border.

Z, the second variable, indicates which color you want to use. The value of Z is from 0 to 15 with the following color/number equivalents:  

0	gray
1	light orange (gold)
2	orange
3	red-orange
4	pink
5	maroon-red
6	purple blue
7	blue
8	blue
9	light blue
10	turquoise
11	green-blue
12	green
13	yellow-green
14	orange green
15	light orange



Remember! SETCOLOR Y, Z, W functions differently in the different graphics modes.



Colors will vary a bit with the type and adjustment of the TV or monitor you use.

The third SETCOLOR Y, Z, W variable, W, determines the luminance of the screen. It ranges over the *even* numbers from 0 to 14 (i.e., 0, 2, 4, 6, 8, 10, 12, 14). The higher the number, the brighter the background. When W=0, the letters will be very bright and the screen dullish. When W=14, the screen and letters will be very bright, and the letters will also appear tinted a tone close to that of the screen. At W=10, the letters will be of the same luminance and color of the screen so that they will blend into the screen and appear invisible.

Here are a few simple programs that illustrate the aspects of color in GRAPHICS 0 we've introduced:

1. Changing background colors at random with bright letters using SETCOLOR 2, INT(RND(1)\*16), 0

```
5 REM BRIGHT WHITE LETTERS WITH COLOR CHANGES
10 SETCOLOR 2,INT(RND(1)*16), 0
20 PRINT "HELP ME I'M LOST ON A MAP";
30 FOR X=1 TO 200:NEXT X
40 GOTO 10
```

2. Changing background colors at random with tinted letters—  
SETCOLOR 2, INT(RND(1)\*16), 14

```
5 REM BRIGHT LETTERS THE COLOR OF BACKGROUND
10 SETCOLOR 2,INT(RND(1)*16), 14
20 PRINT "HELP ME I'M LOST ON A MAP";
30 FOR X=1 TO 200:NEXT X
40 GOTO 10
```

3. Letters invisible, random color changes—  
SETCOLOR 2, INT(RND(1)\*16), 10

```
5 REM LETTERS AND BACKGROUND SAME LUMINANCE
6 REM SO LETTERS CANNOT BE SEEN
10 SETCOLOR 2, INT(RND(1)*16), 10
20 PRINT "HELP ME I'M LOST ON A MAP ";
30 FOR X=1 TO 200:NEXT X
40 GOTO 10
```

4. Changing the colors of the border—  
SETCOLOR 4, INT(RND(1)\*16), 14

```
5 REM NOW THE BORDER CHANGES
10 SETCOLOR 4, INT(RND(1)*16), 14
20 PRINT "HELP ME I'M LOST ON A MAP ";
```

```
30 FOR X=1 TO 200:NEXT X  
40 GOTO 10
```

Now here is a color version of the first game introduced in the book: guessing a number. This program which uses what you now know about ATARI color could be called:



## COLOR DRESSING FOR NUMBER GUESSING

```

5 REM COLOR DRESSING FOR NUMBER GUESSING
7 LET Y=INT(RND(1)*10)+1
8 FOR X=1 TO 20
10 PRINT "GUESS A NUMBER FROM 1 TO 10"
20 SETCOLOR 2,RND(1)*16, 14
30 SETCOLOR 4,RND(1)*16, 14
40 NEXT X
50 PRINT "[CLEAR]"
55 PRINT "YOUR GUESS IS ?"
60 INPUT A
70 IF A=Y THEN GOTO 1000
80 GOTO 8
1000 FOR W=1 TO 20
1010 SETCOLOR 2,RND(1)*16, 14
1020 SETCOLOR 4,RND(1)*16, 14
1030 PRINT "YOU GOT IT";
1040 NEXT W
1050 PRINT "[CLEAR]"
1055 FOR T=1 TO 500:NEXT T
1060 GOTO 7

```



You should now be able to incorporate SETCOLOR Y, Z, W in your own games in GRAPHICS 0. For GRAPHICS 3 through 7 you use SETCOLOR 4, Z, W to change the background and the command COLOR X to change the color of the points you plot. X can be 1, 2, or 3 and

COLOR 1	yellow points
COLOR 2	light green points
COLOR 3	blue points

Here's a simple program that generates all three colors at random points on the screen resulting in an infinite number of Mondrian-type patterns:

## A RANDOM COLOR MOSAIC

```

1 REM *** A RANDOM COLOR MOSAIC ***
2 REM LEN LINDSAY 12/79
5 GRAPHICS 3+16
10 FOR X=1 TO 3
20 COLOR X
30 PLOT INT(RND(0)*37),INT(RND(0)*23)
40 NEXT X
50 GOTO 10

```

If you add a SETCOLOR 4, Z, W command at line 6, you will be able to change the background color as well as the color of the points. For example, SETCOLOR 4, 5, 0 at line 6 will produce a blue, white, and yellow mosaic on a red-maroon background. You can use both COLOR X and SETCOLOR Y, Z, W to create four-color patterns on your screen and to dress up your programs. GRAPHICS modes 4, 5, 6, and 7 function in exactly the same way as GRAPHICS 3.

If you want to see the other color possibilities of your ATARI, here's a demonstration program of the 128 ATARI color/luminance combinations called *The Atari Rainbow*:

```


10 REM <<< RAINBOW >>>
20 REM * by the Software Engineers
30 REM * at ATARI INC.
40 PRINT "[CLEAR]THIS IS A PROGRAM WHICH
   DISPLAYS":? "ALL 128 OF YOUR ATARI'S CO
   LORS ":? "ON THE SCREEN AT THE SAME TIME
   !"
50 PRINT :PRINT "(IT DOESN'T USE NORMAL
   GRAPHICS MODES)"
60 PRINT :PRINT "THERE ARE 16 COLOR BARS
   , EACH WITH":? "8 DIFFERENT SHADES OR LU
   MINANCES"
70 ? :? "IT USES VERY ADVANCED FEATURES
   OF THE":? "ATARI'S DISPLAY SYSTEM..."
80 FOR WAIT=1 TO 1800:NEXT WAIT
90 DIM C$(24)
100 SETCOLOR 2,0,0:POKE 752,1:~ "[CLEAR]
   "
110 FOR I=1 TO 24
120 READ D
130 C$(I,I)=CHR$(D)
140 NEXT I
150 D=USR(ADR(C$))
160 END
170 DATA 162,0,173,11,212,201,32,208,249
   ,141
180 DATA 10,212,142,24,208,232,232,208,2
   46,142
190 DATA 24,208,240,232

```

As you may have noticed, color has been used in programs throughout the book. You may want to look back and see how it is woven into those programs now that you have some sense of the use of COLOR X and SET-COLOR Y, Z, W.

Let's look at a few other programs that demonstrate the color capacity of your ATARI Home Computer.

## A 3-D COLOR CUBE


The following program, written by Dave Thornburg, demonstrates how one can use the various luminance levels of the ATARI Home Computer to give three-dimensional graphic effects. In this program, Dave "shades" the different faces of a cube. After loading the program, just type any key on the keyboard, and continue typing a succession of keys to watch the changes happen! (Note: GRAPHICS 23 in line 50 is just GRAPHICS 7+16, i.e., GRAPHICS 7 with no text window.) 



```

10 REM <<< 3-D COLOR CUBE >>>
20 REM * By Dave Thornburg
30 PRINT "CLEAR]PRESS DIFFERENT KEYS TO
   ":PRINT "CHANGE COLORS."
40 FOR WW=1 TO 1200:NEXT WW
50 GRAPHICS 23
60 OPEN #1,4,0,"K:"
70 FOR I=0 TO 4:SETCOLOR I,9,4:NEXT I
80 X0=48:Y0=36
90 COLOR 1
100 FOR I=0 TO 40
110 PLOT X0,Y0+I:DRAWTO X0+40,Y0+I
120 NEXT I
130 COLOR 2
140 FOR I=1 TO 24
150 PLOT X0+I,Y0-I:DRAWTO X0+I+40,Y0-I
160 NEXT I
170 COLOR 3
180 FOR I=1 TO 24
190 PLOT X0+40+I,Y0-I:DRAWTO X0+40+I,Y0+
   40-I
200 NEXT I
210 FOR I=0 TO 2
220 GET #1,A
230 IF A<48 THEN A=48
240 SETCOLOR I,1,2*(A-48)
250 NEXT I
260 POKE 77,0:GOTO 210

```

 This program is used by permission of *COMPUTE! The Journal for Progressive Computing*, an excellent source of programs and ideas for Atari computer users. *COMPUTE!* is published by Small Systems Services, Inc., P.O. Box 5406, Greensboro, N.C. 27043.

## TRIANGLE—A SIMPLE DEMONSTRATION OF COLOR PLOTTING

This program is a series of variations on a theme—in this case, triangles. It shows how very simple commands may be used to produce aesthetic results. It also shows how new colors may be produced using a technique called “color artifacting.”

```

10 REM *          <<< TRIANGLE >>>
20 REM *          by ATARI staff
30 GRAPHICS 7+16
40 E=INT(300*RND(1))
50 D=INT(300*RND(1))
60 C=1
70 COLOR C
80 B=39
90 A=79
100 FOR S=1 TO D STEP E
110 FOR X=A TO B STEP -2
120 PLOT 80,A-X
130 DRAWTO 80+X,INT(A/S)
140 DRAWTO 80,X
150 DRAWTO 80-X,INT(A/S)
160 DRAWTO 80,A-X
170 COLOR C
180 NEXT X
190 C=C+1
200 NEXT S
210 SETCOLOR 0,T,2
220 T=T+1
230 GOTO 40

```

TRIANGLE, copyright © 1981, Atari, Inc. This program is one of several in a collection from the *Graphics/Sound Demonstration* disk (APX-20028), available from the Atari Program Exchange, P.O. Box 427, Sunnyvale, Ca. 94086. (Interesting information on color and sound effects is also provided in the manual for this disk.) This program is reprinted here with permission of Atari, Inc.

## MODERN ART A LA JOSEF ALBERS

This program demonstrates some color relationships using a form made popular by the 20th-century artist, Josef Albers. By focusing on a particular rectangular motif, Albers was able to explore many dimensions of color, such as depth, even though his paintings seemed quite simple. By varying the size and spacing of the rectangles, you can explore these relationships even more fully.

```

0 REM <<< ALBERS >>>
10 REM * Color paintings a la
20 REM * Josef Albers (20TH C. ART)
30 REM * By Ted M. Kahn 1980
40 SETCOLOR 0,1,2
50 SETCOLOR 1,2,4
60 SETCOLOR 2,3,8
70 X1=10:Y1=0
80 GRAPHICS 3+16
90 OSIZE=20:NSIZE=20
100 COLOR 1
110 GOSUB 290
120 NSIZE=16
130 GOSUB 360
140 COLOR 2
150 GOSUB 290
160 NSIZE=12
170 GOSUB 360
180 COLOR 3
190 GOSUB 290
200 FOR I=1 TO 15
210 FOR LUM=0 TO 14 STEP 2
220 SETCOLOR 0,I,LUM+2
230 SETCOLOR 1,I+1,LUM+4
240 SETCOLOR 2,I+2,LUM+6
250 GOSUB 410
260 NEXT LUM
270 NEXT I
280 GOTO 200
290 REM * DRAW SQUARE
300 FOR LINE=1 TO NSIZE+1
310 PLOT X1+LINE,Y1
320 DRAWTO X1+LINE,Y1+NSIZE-1
330 NEXT LINE

```

```
340 OSIZE=NSIZE
350 RETURN
360 REM * DETERMINE NEW SQUARE SIZE
370 X1=X1+(OSIZE-NSIZE)/2
380 Y1=Y1+OSIZE-NSIZE-1
390 RETURN
400 REM * PAUSE
410 FOR WW=1 TO 800:NEXT WW:RETURN
```

## USING PADDLE CONTROLLERS TO PLAY WITH COLOR AND SOUND

This is a very short program which demonstrates how you can use the ATARI paddles to control the display of color and sound. Lines 15 and 27 control the color and sound effects through sensing changes in the paddle position and depressed button (trigger), respectively. Plug the paddle into the first position on your computer.

```

10 REM *      <<< COLOR DOTS >>>
20 REM *    by Bill Carris (ATARI)
30 REM *      REQUIRES PADDLES
40 TRAP 100:GRAPHICS 0
50 PRINT "CLEAR]THIS PROGRAM SHOWS OFF
THE ATARI'S"? "COLORS, USING THE PADDLE
CONTROLLERS"
60 PRINT "TO CONTROL BOTH COLOR AND DYNA
MIC DOT"? "PATTERNS. IT ALSO HAS SOUND
EFFECTS,"
70 PRINT "CONTROLLED FROM THE RED BUTTON
S ON "? "THE PADDLES,""? :? "PRESS THE
START BUTTON WHEN YOU'RE "
80 PRINT "READY TO GO";
90 IF PEEK(53279)<>6 THEN 90
100 SETCOLOR 2,A,2
110 LET A=PADDLE(0)/6.2
120 IF A<38 THEN POKE 82,A:PRINT "CTJCTJ
"
130 SOUND 0,A,10,8
140 IF PTRIG(0)=0 THEN SOUND 0,A,6,A
150 GOTO 100

```

*Epilog on animation, graphics, and color.* There are many advanced tricks still awaiting you, including "player-missile graphics," display lists and display list interrupts, and the wonders of color of the new GTIA chip. For more advanced programmers, especially those interested in 6502 Assembly language programming, we recommend getting *De Re Atari - A Guide to Effective Programming*. It's available from the Atari Program Exchange (APX), Atari, Inc., P.O. Box 427, Sunnyvale, California 94086.



## Appendix I

# Errors and Error Messages: Helping You “Debug” Your Program

As we mentioned earlier, making mistakes is a normal part of the programming process, as it is in all creative learning. As you begin to develop your own game ideas, you will certainly go beyond the programs in this book, and, as we have, you will undoubtedly run into some “bugs.” Your ATARI Home Computer has already been programmed to help you try to track down some of these bugs. It does this in two ways: (1) As you type in your program, line by line, the lines are examined by a program called the ATARI BASIC Interpreter. This program checks to see if your statements are clear and understandable ATARI BASIC. If they are not, your ATARI Home Computer will put an “ERROR” message in the lines it doesn’t recognize and will also put the cursor on the actual letter or character which seemed to be the cause of the problem. This works very well for simple types of errors, such as misspelled BASIC keywords (e.g., “PINT” instead of “PRINT”). (2) The second type of errors cannot be detected until you actually RUN your program; for this reason, they are called “Run-time errors.” Whereas the first type of errors are called “syntax errors” (because the syntax of a BASIC statement didn’t match with what the Interpreter program expected), the second type are commonly known as “semantic errors.” This is because what you said may be technically correct, but how you used it created a problem. For example, you put a GOSUB statement in your program but forgot to RETURN. Or you may have had a FOR statement but forgotten a corresponding NEXT, etc. In any case, your intentions were honorable, but your ATARI Home Computer cannot work on intentions alone! It requires exactness.

This is where error messages come in helpful. There is a list of error messages below, taken directly from the *ATARI 400/800 BASIC Reference Manual*. There are 52 Error Messages listed in this manual, most of which are related to input/output problems for disk drives, cassette recorders, printers, etc. However, there are 17 errors (nos. 2-18), which are directly related to programming problems in BASIC. Unless otherwise noted here, the explanations here are identical to the descriptions given in Appendix B of the *BASIC Reference Manual*. What we will add here are additional hints for trying to track down program bugs which *may not* show up as overt errors. This process may often involve quite a bit of detective work, but will be greatly aided by the following programming tips:

- Try to write your programs in modular form. This means to try to use subroutines (GOSUB) whenever possible, especially for doing the same things repeatedly in different parts of the program.
- Try to maintain some consistency in the naming conventions used for variable names from program to program. It also makes transferring ideas from one program to the next much simpler.
- Use REM statements to document your program whenever possible. While these statements do not take up memory space, they will save you a lot of time should you ever want to come back to an old program after a long time and make some changes.
- In addition to using REM statements, make a list of the variable names you use (including arrays and string variables), with a short annotated description of each, including name, function, and special remarks about line numbers where crucial value changes take place.

The following are errors that you might encounter in writing programs using ideas from the first few chapters of this book. This particular group of errors is explained in somewhat more detail than those in Appendix B of *The ATARI 400/800 BASIC Reference Manual*. This is because they tend to be made more frequently, even by such people as the authors of this book! As discussed above, all of these are "Run-time" errors, i.e., they will not appear as errors until you RUN a program, because they are syntactically correct (the statements may look fine, but beware!).

ERROR #	DESCRIPTION & PROBABLE CAUSES
3	<i>Value error.</i> A likely place for this error to pop up is in GRAPHICS modes. The values for PLOT and DRAWTO must be non-negative, i.e., either 0 or positive. The following program will produce this error:

10 GRAPHICS 7  
 20 COLOR 1  
 30 PLOT -10,20

Remember: The X,Y coordinate system used for ATARI graphics is not the standard Cartesian system! See Chapter Two and Chapter Six for details.

- 5 *String Length Error.* This error occurs when you try to use a string which is longer than the space reserved for it in a DIM statement.

*Example:*

```
5 DIM A$(5)
6 LET A$="ABCDE"
10 LET A$(6,6)="F"
```

*Explanation*

This program will produce Error 5 at line 10 because there are only 5 spaces reserved for A\$ in line 5, but we tried to use 6 at line 10.

*Corrected*

```
5 DIM A$(10)
6 LET A$="ABCDE"
10 LET A$(6,6)="F"
```

*Advice:* If you are not sure how long your strings will be, it's always better to DIM more space than is necessary.

- 6 *Out of DATA Error.* This error occurs when you are using a READ statement to read data (numbers or strings) from DATA statements, but you've run out of data to read! The most likely occurrence of this error will be in a FOR-NEXT loop, as shown:

```
5 DIM M(10)
10 FOR I=1 TO 10
20 READ A: M(I)=A: REM
    You can't READ into an array element directly!
30 NEXT I
40 DATA 1,3,5,7,9,11,13,15,17
```

Notice that the FOR-NEXT loop in lines 10-30 will execute line 20 *ten* times; however, there are only *nine* data items in line 40. Another possible source for this error is failing to use the DATA statement you thought you were using, by forgetting to RESTORE the reading

of data to the proper starting line. For example, look at this program:

```
10 FOR I=1 TO 5
20 READ A:PRINT A+5
30 NEXT I
40 FOR J=1 TO 5
50 READ A:PRINT J*A
60 NEXT I
70 DATA 1,3,5,7,9
```

This program will produce error 6 at line 50. If we wanted to reuse the same DATA, then we could easily fix the error by inserting one line:

```
35 RESTORE 70
```

This would restore the data to be read to the beginning of the list at line 70, instead of leaving it after the end of five items.

Just remember: for every READ statement which the computer encounters, there must be a corresponding DATA statement, and the number of items in all the DATA statements must be sufficient for the number of READs.

## 8

*Input Statement Error.* This error is fairly easy to diagnose. It means that a non-numeric value was INPUT when a number was expected. For example:

```
10 INPUT A
20 PRINT A
RUN
?A
```

Here, the program will produce error 8 because the letter "A" is not a number, even though it is used in the program to name a numeric variable. A general rule. If you are expecting a person to input anything other than just numbers, use a string variable for INPUT:

```
5 DIM A$(1)
10 INPUT A$
```

20 PRINT A\$  
would not produce an error.

- 9 *Array or String DIM error.* This may be related to error 5, or it may be a simple numeric array error. Both arrays and string variables must have space reserved for them through DIM statements. If you try to refer to a part of an array or string which is outside the DIM size, you will get this error. Here are two examples:

```
10 DIM A(10), B(3,3)
20 LET A(12)=2          10 INPUT NAME$
30 LET B(3,4)=A(5)      20 IF NAME$="ABE"
                        THEN 10
```

In the left example, both arrays A and B were dimensioned but were referenced by subscripts out of the range in which they were dimensioned. In the example on the right, NAME\$ is referenced, but no DIM was included. This latter error is very common in ATARI BASIC, since its requirement to DIMension all string variables before they are used is unlike common other versions of BASIC. If you use string variables in ATARI BASIC, be sure to DIMension them first!

- 11 *Attempt to Divide by Zero or reference a number larger than  $10^{98}$  or smaller than  $10^{-99}$ .* Dividing by zero is mathematically undefined. Also, your ATARI Computer cannot handle numbers larger or smaller than the limits specified here. If this error occurs, check your calculations!
- 12 *Line not found.* A very common error, usually due to simple clerical carelessness. If you use a GOTO, GOSUB, or IF-THEN statement, be sure the line number specified actually exists. Two simple examples:

```
8 A=7                    10 GOSUB 100
10 PRINT "HELLO"         20 PRINT "HURRAY!"
15 IF A=5 THEN GOTO 5     200 LET X=30
20 A=A-2                 210 RETURN
25 GOTO 10
```

In the left example, error 12 would occur at line 15; in the right-hand example, this error would occur at line

10, since the subroutine is numbered "200" instead of 100.

13

*No Matching FOR Statement.* Somehow, the program had come to a NEXT statement for which it can't find a corresponding FOR. This most often occurs in two cases: (1) simple forgetfulness; and (2) using multiple FOR-NEXT loops, nested inside one another. Take a look at the program below, and see if you can find where Error 13 would occur:

```
10 FOR I=1 TO 10
20 PRINT I/15*22.9
30 FOR J=2 TO 10 STEP 2
40 PRINT J-I
50 NEXT I
60 NEXT J
```

Now, while this program may look fine, it isn't! If more than one FOR/NEXT loops are needed at the same time, all must be nested inside another completely! That is, this program would error-out at line 60. To correct it, the loop with J would have to be complete before the I-loop. If you changed:

```
50 NEXT J
60 NEXT I
```

this would solve the problem.

16

RETURN error. This means that the program reached a RETURN statement by accident (i.e., there was no GOSUB which resulted in the program's reaching this statement). The most common reason for this error is that a program reached the "subroutines" when it should have already stopped or ended. Here's an example:

```
10 DIM Y$(1)
20 PRINT "WOULD YOU PLAY AGAIN";
30 INPUT A$
40 IF A$="Y" THEN GOSUB 100
50 PRINT "THANKS ANYWAY."
```

```
100 PRINT "O.K., HERE WE GO AGAIN!"
110 RETURN
```

What will happen here is that the program will do a subroutine jump (GOSUB) at line 40 if the player answers "Yes"; however, there is nothing in the program which protects him from falling into line 100 after he returns from line 110. Thus, the following statements will be executed: 10,20,30,40,100,110,50,100,110. If the answer had been "NO", the sequence would have been: 10,20,30,40,50,100,110. Note that in *both cases*, because of lack of protection line 110 would have been executed. The solution here is easy and highly recommended for any programs using GOSUB/RETURN:

Put an END or STOP statement before you reach this point (in our example, it could have been line 90); then, we would always stop short of our subroutine problem.

```
90 END
```

141

*Cursor Out of Range.* This is related to Error 3. It simply means the cursor, either in text modes or graphics modes is not within the necessary boundaries for that mode. The table in Chapter Six summarizes what these boundaries are (see also *ATARI BASIC Reference Manual*). These programs will produce Error 141:

```
10 POSITION 40, 10   10 GRAPHICS 7:COLOR 1
20 PRINT "HELLO"   20 PLOT 10,10
                   30 DRAWTO 300,200
```

In the first example, we tried to put the cursor past its maximum horizontal value for normal text mode (GR. 0)—max is 39, so it is horizontally out of range. In the second example, we tried to DRAWTO a point out of range for this GR.7 mode—max horizontal & vertical is 159,95.

## Other Errors

If you ever encounter any other errors between error 19 and error 171, they have to do with peripherals or file-devices. As a first pass, check all your peripherals and make sure the plugs and connectors are secured. If you

get an error 144, it means you have a write-protected disk and are trying to write on it. Most others are fairly self-evident; if for any reason, you may need more information than this, then call ATARI Customer Support, Home Computer Division at (800)538-8737 (out of California) or (800)672-1430 (in California).

---



## Appendix II

# ATARI BASIC Cards— Reference Guide

# ATARI<sup>™</sup> BASIC CARDS QUICK GUIDE

by Len Lindsay

ABS(N)	ABS
└ numeric variable or constant	
ADR(A#)	ADR
└ string variable name	
A AND B	AND
└ first substatement	
└ second substatement	
ASC(A#)	ASC
└ string variable or constant enclosed in quotes	
ATN(N)	ATN
└ numeric variable or constant	
BYE	BYE
CHR\$(N)	CHR\$
└ numeric constant or variable	
CLOAD	CLOAD
CLOG(N)	CLOG
└ numeric constant or variable	
CLOSE #F	CLOSE
└ file number - numeric constant or variable (1-7)	
CLR	CLR
COLOR N	COLOR
└ color register (0-4)	
└ numeric constant or variable	
CONT	CONT
COS(N)	COS
└ numeric constant or variable	
CSAVE	CSAVE
DATA N	DATA
└ numeric or string constant	
DEG	DEG
DIM V(B)	DIM
└ numeric array dimensions or maximum # char in the string	
└ numeric or string variable name	
DRAWTO X,Y	DRAWTO
└ Y screen coordinate	
└ X screen coordinate	
END	END
ENTER "T:"	ENTER
└ type of device letter (C or D)	
EXP(N)	EXP
└ numeric constant or variable	
FOR L=S TO E STEP N	FOR
└ start loop value	
└ end loop value	
└ step size(optional)	
└ loop numeric variable name	
FRE(0)	FRE

ATARI is trademark of ATARI INC.

GET #F,N GET  
 T numeric variable name  
 T file number (1-7)  
 GOSUB T GOSUB  
 T target line number  
 T numeric variable or constant  
 GOTO T GOTO  
 T target line number  
 T numeric variable or constant  
 GRAPHICS R GRAPHICS  
 T graphic mode number (0-8)  
 T numeric constant or variable  
 IF A THEN B IF  
 T statement  
 T comparison  
 INPUT V INPUT  
 T string or numeric variable name  
 INPUT #F,V INPUT#  
 T strings or numeric variable name  
 T numeric file number (1-7)  
 INT(N) INT  
 T numeric constant or variable  
 LEN(S\$) LEN  
 T string variable or constant  
 LET A=B LET  
 T numeric or string constant or variable  
 T numeric or string variable name  
 LIST "T:", S,E LIST  
 T end line number  
 T start line number  
 T type device for list (optional)  
 T "name.ext"  
 LOAD "T:", S,E LOAD  
 T 0-3 char name extension  
 T optional 0-8 character name  
 T type device letter  
 LOCATE X,Y,N LOCATE  
 T numeric variable name  
 T Y coordinate  
 T X coordinate  
 LOG(N) LOG  
 T numeric variable or constant  
 LPRINT A LPRINT  
 T numeric or string constant or variable  
 NEW  
 NEXT L NEXT  
 T numeric variable name  
 T same as name used in FOR  
 NOT N NOT  
 T numeric constant or variable  
 NOTE #F,S,B NOTE  
 T byte number within sector  
 T sector number  
 T file number

ON N GOSUB L1,L2... ON  
 T second line number  
 T first line number  
 T numeric variable name  
 OPEN#F,I,P,"T:name.ext" OPEN  
 T 0-3 char extension  
 T optional 0-8 char name  
 T type of device letter  
 T printer code (0 if not used)  
 T I/O code: 4=in, 8=out, 12=both  
 T file number (1-7)  
 A OR B OR  
 T second statement  
 T first statement  
 PADDLE(P) PADDLE  
 T paddle number, constant or variable (0-7)  
 PEEK(L) PEEK  
 T memory location, numeric variable or constant  
 PLOT X,Y PLOT  
 T Y coordinate  
 T X coordinate  
 POINT #F,S,B POINT  
 T byte number within sector  
 T sector number  
 T file number  
 POKE L,V POKE  
 T value (0-255) to put in RAM  
 T location of RAM to put value  
 POP  
 POSITION X,Y POSITION  
 T Y coordinate  
 T X coordinate  
 PRINT A PRINT  
 T numeric or string  
 PRINT #F; A PRINT#  
 T numeric or string  
 T file number (1-7)  
 PTRIG(P) PTRIG  
 T paddle number (0-7)  
 T numeric constant or variable  
 PUT #F,V PUT  
 T numeric constant or variable (0-255) ASCII char code  
 T file number (1-7)  
 RAD  
 READ A READ  
 T numeric or string variable name  
 REM anything REM  
 RESTORE L RESTORE  
 T line number - constant or variable (default is 0)  
 RETURN RETURN  
 RND(0)  
 RUN "T:name.ext"  
 T optional device type and name

SAVE "T:name.ext"	SAVE
├──┬──┬──┐	
└──┬──┐	0-3 char name extension
└──┐	0-8 char optional name
└──┐	type device letter
SETCOLOR R,C,L	SETCOLOR
├──┬──┬──┐	
└──┬──┐	luminance
└──┐	(dark 0-14 light, even)
└──┐	color (0-15)
└──┐	register number (0-4)
SGN(N)	SGN
└──┐	numeric constant or variable
SIN(N)	SIN
└──┐	numeric constant or variable
SOUND R,P,T,V	SOUND
├──┬──┬──┬──┐	
└──┬──┬──┐	volume (soft 0-15 loud)
└──┬──┐	tone (0-14 even, 10=clear)
└──┐	pitch (high 0-255 low)
└──┐	register or voice number (0-3)
SQR(N)	SQR
└──┐	numeric variable or constant
STATUS	STATUS
FOR L=S TO E STEP N	STEP
├──┬──┬──┬──┐	
└──┬──┬──┐	step size(optional)
└──┬──┐	end loop value
└──┐	start loop value
└──┐	loop numeric variable name
STICK(N)	STICK
└──┐	stick number(0-7)
STOP	STOP
STRIG(N)	STRIG
└──┐	joystick number (0-7)
STR\$(N)	STR\$
└──┐	numeric variable or constant
IF A THEN B	IF
├──┬──┐	statement
└──┐	comparison
FOR L=S TO E STEP N	TO
├──┬──┬──┬──┐	
└──┬──┬──┐	step size(optional)
└──┬──┐	end loop value
└──┐	start loop value
└──┐	loop numeric variable name
TRAP L	TRAP
└──┐	line number variable or constant
USR(V)	USR
└──┐	input values (at least one)
VAL(S\$)	VAL
└──┐	string variable or constant
XIO C,#F,0,0,"T:"	XIO
├──┬──┬──┬──┐	
└──┬──┐	type device letter
└──┐	file number
└──┐	command number

**ASC(A\$)** **ASC**  
 T  
 Lstring variable or  
 constant enclosed in quotes  
 B=ASC("\*") B=42  
 \*Returns the numeric code (ATASCII  
 code) of a single string character.  
 see also VAL, STR\$, CHR\$

**ATARI™ BASIC CARDS**  
 (c) 1980 Len Lindsay  
 1929 Northport Dr#6, Madison, WI 53704

**A AND B** **AND**  
 T T  
 Lsecond substatement  
 Lfirst substatement  
 IF A>0 AND A<10 THEN GOSUB 2000  
 \*A logical operator. Statement is true  
 only if both substatements are true.  
 Evaluates to 1 if true or 0 if false.  
 see also OR, NOT

**ADR(A\$)** **ADR**  
 T  
 Lstring variable name  
 DIM A\$(5)  
 C=ADR(A\$) C=2085  
 \*Returns the starting storage address  
 in decimal of the specified string.

**ABS(N)** **ABS**  
 T  
 Lnumeric variable or constant  
 A=ABS(-25) A=25  
 \*Returns the absolute value of the  
 number.

**ATN(N)** **ATN**  
 T  
 Lnumeric variable or constant  
 B=ATN(56) B=1.55294108  
 \*Returns the arctangent of the number.

**BYE** **BYE**  
 \*Exits BASIC. Puts you into ATARI  
 Operating System  
 (ATARI COMPUTER - MEMO PAD)

**CHR\$(N)** **CHR\$**  
 T  
 Lnumeric constant or variable  
 PRINT CHR\$(65) A  
 \*Returns the character that has the  
 specified numeric code (ATASCII code).  
 see also ASC, STR\$, VAL

**CLOAD** **CLOAD**  
 \*Loads a program from cassette to RAM  
 memory. Does an automatic OPEN.  
 Computer will beep once. You then  
 press PLAY on the tape recorder and  
 hit RETURN.  
 see also CSAVE, ENTER, LIST, RUN, LOAD,  
 SAVE

CLOG(N)	CLOG	CONT	CONT
$\downarrow$ numeric constant or variable			
A=CLOG(83)	A=1.91907809		
*Returns the numbers log in BASE 10.			*Restarts the program at the line immediately following the statement it was executing when a BREAK occurred. Direct mode only. Any remaining unexecuted statements in the previous line are skipped.
see also LOG			see also STOP, END

CLOSE #F	CLOSE	COS(N)	COS
$\downarrow$ file number - numeric constant or variable (1-7)		$\downarrow$ numeric constant or variable	
CLOSE #1		A=COS(5)	A=0.283660887
*Closes file F (F is the file number).			*Returns the cosine of the number.
see also OPEN			see also SIN

CLR	CLR	CSAVE	CSAVE
*Clears all variables values back to 0 and deletes all DIMensioned strings and arrays.			*Saves a program from RAM memory to cassette. Computer beeps twice. You then press RECORD and PLAY on the tape recorder and hit RETURN. It does an automatic OPEN.
see also DIM			see also CLOAD, ENTER, LIST, RUN, LOAD, SAVE, OPEN

COLOR N	COLOR	100 DATA N	DATA
$\downarrow$ color register (0-4) numeric constant or variable		$\downarrow$ numeric or string constant	
A=3:COLOR A	register 3 now controls	8 DATA 15,SHOES	
Chooses the register to be used in the following graphic work.		9 READ A,B\$	A=15 and B\$="SHOES"
see also SETCOLOR, GRAPHICS, PLOT, LOCATE, DRAWTO			*Used only in deferred mode within a program. Any number of numeric or string data can follow the keyword DATA, separated by commas. DATA is input to program by READ.
			see also READ

**DEG DEG**

DEG  
PRINT COS(3)                      0.9986295421

\*Instructs the computer to use degrees in all following trig calculations. The default is in radians.

see also RAD

**ENTER "T:" ENTER**  
<sup>T</sup>  
 ↳ type of device letter (C or D)

ENTER "C:"

\*Enter program segment from tape or disk in untokenized form (saved with LIST). It will automatically be appended to any existing program.

see also LIST, CSAVE, CLOAD, SAVE, LOAD, RUN

**DIM V(B) DIM**

<sup>T</sup> <sup>T</sup>  
 |    ↳ numeric array dimensions or  
 |    ↳ maximum # char in the string  
 ↳ numeric or string variable name

DIM A\$(3)            or            DIM A(4,9)

\*Reserves space for strings and arrays (single or double). Multiple DIM per line is acceptable, separated by commas.

see also CLR

**DRAWTO X,Y DRAWTO**

<sup>T</sup> <sup>T</sup>  
 |    ↳ Y screen coordinate  
 ↳ X screen coordinate

PLOT 1,1  
DRAWTO 5,5

\*Draws a line from the last accessed screen point to the point specified. The top left corner of the screen is referred to as 0,0. (GRAPHICS 3-8)

see also PLOT, POSITION, PRINT#, GRAPHICS

**END END**

\*Program execution is halted. It may be restarted with CONT.

see also STOP, CONT

**EXP(N) EXP**

<sup>T</sup>  
 ↳ numeric constant or variable

PRINT EXP(3)                      20.0855365

\*Raises e (2.71828179) to specified power (exponent).

**FOR L=S TO E STEP N FOR**  
<sup>T</sup> <sup>T</sup>    <sup>T</sup>    <sup>T</sup>  
 | |    |    ↳ step size (optional)  
 | |    ↳ end loop value  
 |    ↳ start loop value  
 ↳ loop numeric variable name

FOR N=15 TO 1 STEP -1.5

\*Loop repeats itself each time it hits the NEXT until end value is exceeded.

see also TO, STEP, NEXT

**FRE(0) FRE**

PRINT FRE(0)

\*Returns the number of free bytes of RAM memory available to user.

GET #F,N

T T  
 | |  
 L numeric variable name  
 L file number (1-7)

OPEN #1,4,0,"K:"  
GET #1,N

\*Inputs a single byte from the device specified in the matching OPEN statement. ATASCII value assigned to the specified variable.

see also PUT, OPEN, CLOSE, INPUT

GOSUB T

T  
 L target line number  
 numeric variable or constant

A=6:GOSUB 1000+A\*10      branch to 1060

\*Branch to specified line number and return when done (RETURN encountered). May be used with ON.

see also ON, RETURN, POP, GOTO

GOTO T

T  
 L target line number  
 numeric variable or constant

GOTO 250

\*Program execution continues with line number specified. May be used with ON.

see also ON, GOSUB

GRAPHICS R

T  
 L graphic mode number (0-8)  
 numeric constant or variable

GRAPHICS 3      or      GR. 3

\*Specifies which graphics mode will be used for screen displays. Modes 0-2 are text modes, 3-8 are graphic modes. May be abbreviated as GR.

see also COLOR, SETCOLOR, PLOT

GET

IF A THEN B

T T  
 | |  
 L statement  
 L comparison

IF A<9 THEN PRINT "WINNER"

\*Allows conditional statement execution - only executed if comparison A is true. Relies on THEN to separate comparison A from statement B.

see also THEN

IF

INPUT V

T  
 L string or numeric variable name

INPUT N\$,P

\*Input is requested from the keyboard, terminated by hitting RETURN key. Multiple variables may be input, separated by commas.

see also INPUT #, GET

INPUT

INPUT #F,V

T T  
 | |  
 L string or numeric variable  
 | name  
 L numeric file number (1-7)

INPUT #1,A

\*Inputs data from the device specified in the matching OPEN statement. Multiple variables may be input if separated by commas.

see also OPEN, GET, INPUT

INPUT#

INT(N)

T  
 L numeric constant or variable

PRINT INT(1.8)

1

\*Returns the integer value of the number specified, truncated (rounded down).

INT







**PLOT X,Y**

```

  T T
  |  |  \Y coordinate
  |  |  \X coordinate

```

PLOT 1,Y+5

\*Plots a single point on the screen. Color determined by the last COLOR statement. Coordinates may be numeric constants or variables. Point 0,0 is screen top left.

see also DRAWTO, COLOR, GRAPHICS

**PLOT****POSITION X,Y**

```

  T T
  |  |  \Y coordinate
  |  |  \X coordinate

```

POSITION 4,6

\*Sets the cursor to the specified screen location. Point 0,0 is the top left corner. Coordinates are specified with numeric constants or variables.

see also PLOT

**POSITION****POINT #F,S,B**

```

  T T T
  | | |  \byte number within sector
  | | |  \sector number
  | | |  \file number

```

POINT #1,3,5

\*This is a DISK command used to specify an exact location on the diskette.

see also NOTE, OPEN

**POINT****PRINT A**

```

  T
  |  \numeric or string

```

PRINT "WINNER IS:";N\$

\*Items to print may be constants or variables. More than one may be used per print statement. Multiple items must be separated by commas or semi-colons. ? is abbreviation for PRINT.

see also PRINT#

**PRINT****POKE L,V**

```

  T T
  |  |  \value (0-255) to put in RAM
  |  |  \location of RAM to put value

```

POKE 82,LMARG

\*Places the value specified into the RAM memory location indicated.

see also PEEK

**POKE****PRINT #F ; A**

```

  T   T
  |   |  \numeric or string
  |   |  \file number (1-7)

```

PRINT#6;"YOU WIN"

\*Print to device identified by its OPEN statement (#6 is automatically OPEN for screen graphics). Items may be constants or variables.

see also PRINT, OPEN

**PRINT#****POP**

\*Removes the last FOR NEXT or GOSUB reference from the stack. Used to abnormally terminate the loop or subroutine. Execution continues with the next statement.

see also GOSUB, RETURN, NEXT, FOR

**POP****PTRIG(P)**

```

  T
  |  \paddle number (0-7)
  |  \numeric constant or variable

```

10 IF PTRIG(P)=1 THEN 10

\*Returns a 1 if the trigger button is not pressed, or a 0 if the button is pressed.

see also PADDLE, STICK, STRIG

**PTRIG**

```

PUT #F,V                                PUT
  T T
  |  | numeric constant or variable
  |  | (0-255) ATASCII char code
  |  | file number (1-7)

```

```
PUT#1,ASC("D")
```

\*This is the opposite of GET. It puts one byte representing the ATASCII code of any character to the device indicated in the OPEN statement.

see also GET, OPEN

RAD RAD

\*Instructs computer that all trig functions from here on will be in radians (not degrees). Default is radians.

see also DEG

```

READ A                                READ
  T
  └ numeric or string variable name

```

READ N,N\$

\*One or more variables can have values assigned to them from DATA statements with a READ statement. Optional multiple variables are separated by commas.

see also DATA

```
REM anything
```

```
*Allows comments to be included in the
program listings. Anything after the
REM will be ignored and execution
continued on next line.
```

RESTORE L                      RESTORE  
T  
└ line number - constant or  
  variable (default is 0)

RESTORE 1000+G

\*Allows DATA statements to be reused beginning with the line specified.

see also READ, DATA

RETURN RETURN

```
*Returns from subroutine back to line
that called it with GOSUB.
```

see also GOSUB, POP

RND(0) RND

```
X=INT(RND(0)*6)+1
```

```
*Returns a random number between 0 and 1 (exclusive).
```

```

RUN"T:name.ext"          RUN
-----
      |
      |Optional device type and name

```

RUN	runs program in memory
RUN"C:"	loads and runs tape program

\*Executes program starting at first line. Variables are set to 0 (arrays however are not zeroed out).

see also LOAD, CLOAD, SAVE, CSAVE

SAVE "T:name.ext"                      **SAVE**  
     T T T T  
     | |    └─0-3 char name extension  
     |    └─0-8 char optional name  
     └─type device letter

SAVE "C:"

\*Saves program to device specified.  
 Does an automatic OPEN.

see also CSAVE, LOAD, CLOAD, RUN

SETCOLOR R,C,L                      **SETCOLOR**  
     T T T  
     | | | luminance  
     | |    (dark 0-14 light, even)  
     |    └─color (0-15)  
     └─register number (0-4)

SETCOLOR 2,5,12

\*Stores color and luminance info in  
 one of the 5 color registers.

see also COLOR, GRAPHICS, PLOT

SGN(N)                                  **SGN**  
     T  
     └─numeric constant or variable

IF SGN(X)=1 THEN PRINT "FINISHED"

\*Returns +1 if the number is positive,  
 0 if it is 0, or -1 if it is negative.

SIN(N)                                  **SIN**  
     T  
     └─numeric constant or variable

A=SIN(B)

\*Returns the sine of the number.

see also COS

SOUND R,P,T,V                      **SOUND**  
     T T T T  
     | | | |  
     | | | └─volume (soft 0-15 loud)  
     | | └─tone (0-14 even, 10=clear)  
     | └─pitch (high 0-255 low)  
     └─register or voice number (0-3)

SOUND 2,141,10,8

\*Numbers may be constants or variables.  
 Sound comes through TV speaker and the  
 volume may be further adjusted with  
 its volume controls.

SQR(N)                                  **SQR**  
     T  
     └─numeric variable or constant

PRINT SQR(9)

\*Returns the square root of the  
 number.

STATUS                                  **STATUS**

\*Returns status.

FOR L=S TO E STEP N                      **STEP**  
     T T    T T  
     | |    | |  
     | |    └─step size (optional)  
     | |    └─end loop value  
     | └─start loop value  
     └─loop numeric variable name

FOR Q=1 TO 9 STEP .3

\*Step defaults to 1 if not specified.  
 Loop variable is incremented by step.

see also TO, FOR, NEXT

```
STICK(N)          STICK
  T
  └stick number(0-7)
```

```
250 IF STICK(ZJ)=15 THEN 250
```

```
*Returns the position
of joystick number N.
15 is the neutral
position. Other numbers
are as indicated.
```

10 14 6  
 \ | /  
 <11-15--7>  
 / | \  
 9 13 5

see also STRIG

```
IF A THEN B      THEN
|       |statement
└───┬──┘
    comparison
```

```
IF A=10 THEN PRINT "WINNER"
```

\*If the comparison is true then the statement after the THEN is executed. May be followed by another IF THEN.

see also IF

STOP STOP

\*Execution stops. May be restarted with CONT with variables intact.

see also END, CONT

```
FOR L=S TO E STEP N          TO
  T T T T step size (optional)
  | | | |
  | | | | end loop value
  | | | |
  | | | | start loop value
  | | | |
  | | | | loop numeric variable name
```

```
FOR A=1 TO 30
```

\*TO seperates the start value and end value for the loop variable.

see also STEP, FOR, NEXT

STRIG(N)	STRIG
↑	
Joystick number (0-7)	
numeric variable or constant	

```
IF STRIG(1)=0 THEN PRINT "FIRE!"
```

\*Returns a 1 if joystick N trigger button is not pressed, and returns 0 if it is pressed.

see also PTRIG, STICK

TRAP L TRAP  
T  
L line number variable or constant

```
250 TRAP 250 : INPUT N
```

\*If an error occurs, the last TRAP takes control and execution continues at the line it indicated. Once used it must be reset once again. It may be set or reset as often as you wish.

**STR\$(N)**                      **STR\$**

T  
└ numeric variable or constant

$$N\$ = \text{STR}\$(N)$$

```
*Converts a number into a
string made up of its numerals.
The number 154 becomes "154".
```

see also CHR\$, VAL, ASC

**USR(V)**                      **USR**

T  
└ Input values (at least one)

115R(15)

\*Performs a Machine Language routine using the input values specified.

## VAL(S\$)

T L-string variable or constant

N=VAL(N\$)

\*Converts S\$ reading left to right to the numeric value of the number (until a non numeric character is hit). If no numeric characters are found an error results.

see also STR\$, ASC, CHR\$

## WAIT

Line numbers must be from 0 - 32767.  
Margins- POKE 82,X left POKE 83,X right  
Graphics 3-8 open screen as file#6.  
A\$(LEN(A\$)+1)=B\$ to set A\$=A\$+B\$

Substrings- A\$(start char,end char)

9 IF PEEK(764)=255 THEN 9 waits for key  
PEEK(53279) for 4 special function keys

3=OPTION 5=SELECT 6=START 7=NONE DOWN

To stop/restart display hit CTRL and 1.

Take motor- POKE 54018,X 52=ON 60=OFF

Cursor off- POKE 752,X 1=OFF 0=ON

Logical- 1=TRUE 0=FALSE (usually)

ATASCII CAPS+32=lower Char+128=RVS Char

PEEK(195) yields the error number

TV protect- POKE 77,X 0=OFF 128=ON

## XIO C,#F,0,0,"T:"

T T L-type device letter  
I I L-file number  
L L-command number

## XIO 18,#6,0,0,"S:"

\*This may be used with disk operations and graphics. For graphics 2 dummy 0's are used. The example fills an area on the screen between points with the color set by POKE 765,C (C=color #).

see also POKE, PLOT, DRAWTO

C# color	C# color
0 GRAY	8 BLUE
1 GOLD	9 LIGHT BLUE
2 ORANGE	10 TURQUOISE
3 RED-ORANGE	11 GREEN-BLUE
4 PINK	12 GREEN
5 PURPLE	13 YELLOW-GREEN
6 PURPLE-BLUE	14 ORANGE-GREEN
7 BLUE	15 LIGHT ORANGE
DEFAULTS	COLOR TEXT 1 & 2
SETCOLOR 0, 2, 8	GOLD CAPS
SETCOLOR 1,12,10	LT GREEN LOWER
SETCOLOR 2, 8,12	DARK BLUE RVS CAPS
SETCOLOR 3, 4, 6	RED RVS LOWER
SETCOLOR 4, 0, 0	BLACK BACKGROUND

## TYPE DEVICE KEY / USED WITH KEYWORDS:

"S:" SCREEN ENTER "T:name option"  
"E:" EDITOR LIST "T:",start,end  
"K:" KEYBOARD LOAD "T:name option"  
"C:" CASSETTE OPEN#N,I,P,"T:name"  
"P:" PRINTER RUN "T:name option"  
"D:" DISK SAVE "T:name option"  
XIO C,#F,0,0,"T:"

Replace the T in the statement with the appropriate letter of the desired type of device. Examples: RUN"C:"

ENTER"C:" LIST"P:" XIO 18,#6,0,0,"S:"

## NOTE LOW MIDDLE HIGH HIGHER HIGHEST ???

	LOW	MIDDLE	HIGH	HIGHER	HIGHEST	???
B	128	64	32	16	8	4
Bb	136	68	34	17	8	4
A	144	72	36	18	9	5
G#	153	76	38	19	9	5
G	162	81	40	20	10	5
F#	172	86	43	21	11	6
F	182	91	45	23	11	6
E	193	96	48	24	12	6
I#	204	102	51	25	13	7
D	215	108	54	27	13	7
C#	229	114	57	28	14	7
C	243	121	60	30	15	8

## GRAPHIC MODE #

3 4 5 6 7 8

## SETCOLOR # USED

COLOR 1 0 0 0 0 0 1  
COLOR 2 1 4 1 4 1 2  
COLOR 3 2 0 2 0 2 1  
COLOR 4 4 4 4 4 4 2

## COLOR # CONTROLLED

SETCOLOR0 1 1&3 1 1&3 1 -  
SETCOLOR1 2 - 2 - 2 1&3  
SETCOLOR2 3 - 3 - 3 2&4  
SETCOLOR3 - - - - -  
SETCOLOR4 4 4 4 4 4 -

## MODES COL ROW/SPLIT BYTES

MODES	COL	ROW/SPLIT	BYTES
0 txt	40	24 /	- 993
1 txt	20	24 /	20 513
2 txt	20	12 /	10 261
3 smp	40	24 /	20 273
4 smp	80	48 /	40 537
5 smp	80	48 /	40 1017
6 smp	160	96 /	80 2025
7 smp	160	96 /	80 3945
8 smp	320	192 /	160 7900





## Appendix III

# ATARI Pokes

POKE is a BASIC keyword with most microcomputers, including the ATARI Home Computer. However, many people shy away from it since it affects the computer directly and immediately.

However, we will list and explain some POKes that you may be interested in, and if you use them correctly they can be very useful. There are many others listed in Appendix I of your *Atari 400/800 BASIC Reference Manual*.

### POKE 77,x

**POKE 77, 128**—This puts your computer into ATTRACT MODE. If any image is left on your TV screen for a long period of time, it might burn out some of the color phosphors making up the screen. If no keys have been pressed for a while, your ATARI Home Computer will jump into ATTRACT MODE, changing the colors from light to dark and so on to prevent any image from burning your screen. Location 77 keeps track of how long it has been since the keyboard was accessed. Each time the keyboard is touched, it resets itself to 0. It increments itself every few seconds. When it reaches 128 it jumps into ATTRACT MODE. Thus when you POKE 128 into location 77, it thinks that it is time to protect the screen and jumps to ATTRACT MODE.

**POKE 77, 0**—This is just the opposite of putting 128 into location 77. Put a 0 there and the ATARI resets its count towards ATTRACT MODE.

This is useful when you have the ATARI doing ANIMATED COMPUTER ART. The screen is changing, so there is no worry about burning the screen. So to prevent ATTRACT MODE from starting, just do this poke frequently in your program.

### POKE 755,x

Location 755 keeps track of how the characters will be displayed on the screen. Normally it is 2, but if you put a 4 there all the characters will be displayed upside down. POKE 755, 4 will flip characters upside down. POKE 755, 2 will return them to normal.

### POKE 764,x

This location keeps track of the last key pressed, until it is used by the computer. This is sometimes referred to as the keyboard buffer. The computer will always remember the last key hit. If you hit two before it can use them, it will forget the first one. POKE 765,255 will clear the buffer, even if you had hit a key. This is handy to use just before an INPUT, since it gives a clean start. For example, try this program:

```
10 PRINT "HIT A KEY"
20 FOR LOOP=1 TO 999 : NEXT LOOP
30 INPUT X
```

When you RUN this program and hit a key, notice that when the program next asks you for an INPUT, it types the key you hit previous to asking you as the first part of your answer. At times this is not wanted. Thus, add the following line to your program, and the buffer will be cleared out:

```
25 POKE 764,255 :REM CLEAR KEYBOARD BUFFER
```

Now when you hit a key, the computer forgets it just before asking for INPUT.

It also is useful to use PEEK with 764. That way you can watch for when a key is hit. For example:

```
100 IF PEEK(764)=255 THEN 100 :REM WAIT TILL KEY IS HIT
```

The above line will wait till a key is hit before continuing with the program.

**POKE 82,x      POKE 83,x**

These two locations control the margins on your screen. The left margin is controlled by 82 and the right margin by 83. You can change them at any time by issuing a POKE statement. To find out what the margins are now just try this:

```
PRINT PEEK(82), PEEK(83)
```

They normally are set at 2 for the left margin and 39 for the right. That gives you a line 38 characters long. You can fit 40 characters on the screen, though, if you wish. Just do this:

```
POKE 82, 0
```

Now you have the maximum length. Did you notice the word READY was over two characters to the left? Now try this one:

```
POKE 83, 10
```

Everything still looks the same, but just start typing. Type a whole line of Ks. Were you surprised when less than half way across the screen, they jumped down to the next line? Now clear your screen (don't use the [SYSTEM RESET] button—use [SHIFT] [CLEAR]). The margins are still different. They will remain different until you either issue another POKE to 82 or 83 or you hit the RESET button.



## Appendix IV

# Clocks, Time, and Timing Input

Your ATARI Home Computer has a “built-in” clock, which will enable you to write programs which need a way to keep track of time. This is most often useful when you want to “time” a player’s response. Here is a brief description of how you can make use of your ATARI’s “Jiffy Clock”:

Three bytes of memory keep “time” by storing the number of cycles used by your ATARI’s 6502 CPU. The cycles occur at intervals of 1/60 of a second. The passing of time is recorded in memory locations 18,19, and 20, beginning with the moment your ATARI Home Computer is powered up and ending with the moment at which the power is turned off. Programmers often call each 1/60-second cycle interval a “jiffy”; hence, the name “jiffy clock” for this method of keeping track of time.

Each of these memory locations acts as part of the clock; instead of measuring time in hours, minutes, and seconds, each location just keeps track of as many “jiffies” as it can hold (from 0-255). When the smallest one becomes full, it spills over into the next one, and so on. It’s really very much like a digital clock, except that the ATARI only knows how to count in “jiffies,” instead of seconds. Therefore, we need to do a little arithmetic to make it work for us.

Let’s look at an example. To start your clock running in any program, you first need a statement like this:

```
10 LET TIME1 = INT((PEEK(18)*65536) + (PEEK(19)(*256) +  
    PEEK (20));
```



```

100 TIME1=INT(((PEEK(18)*65536)+(PEEK(19)*256)+
    PEEK(20))/60)
110 PRINT "WHAT'S 4*7 ";
120 INPUT X
130 GOSUB 6000
140 PRINT "IT TOOK YOU ";TIMEGONE;"SECONDS TO
    ANSWER THAT."

```

Because you may use it quite often, it would probably be a good idea to make your "clock starter" a subroutine as well. You could do this by just renumbering line 100 (or line 10 in the first example) and adding a RETURN to end the subroutine. Then, to start the clock, you would just do a GOSUB to that line number:

```

100 GOSUB 5900:REM - Start clock running -
...
...
130 GOSUB 6000:REM - Get elapsed time -
...
...
5900 REM -- SUBROUTINE TO START THE CLOCK --
5910 TIME1 = INT(((PEEK(18)*65536)+(PEEK(19)*256)+
    PEEK(20))/60)
5999 RETURN
6000 REM -- SUBROUTINE TO GET ELAPSED TIME --
6010 NOW = INT(((PEEK(18)*65536)+(PEEK(19)*256)+
    PEEK(20))/60)
6020 TIMEGONE = NOW - TIME1
6030 RETURN

```





## Appendix V

# A Scorekeeper Subroutine for More Than One Player

### A Scorekeeper Subroutine for More Than One Player

```
10 REM—SCOREKEEPER FOR MORE THAN ONE PLAYER
15 REM—TED M. KAHN
20 PRINT "HOW MANY PLAYERS";
25 INPUT P
30 DIM S(P)
35 FOR I=1 TO P:LET S(P)=0:NEXT I
100 REM—TO CALL SCOREKEEPER
101 REM LET W=NUMBER OF WINNER
102 REM (E.G., PLAYER 1,2,3, . . . ETC.)
105 REM—THEN GOSUB 6000
6000 REM—ADD TO WINNER'S SCORE
6005 REM—AND SHOW TOTALS
6010 S(W)=S(W)+1
6020 GRAPHICS 1:REM SHOW SCORES
6030 FOR I=1 TO P
6040 PRINT #6;"PLAYER;I;"=";"S(I)"
6050 PRINT #6
6060 NEXT I
6099 RETURN
```



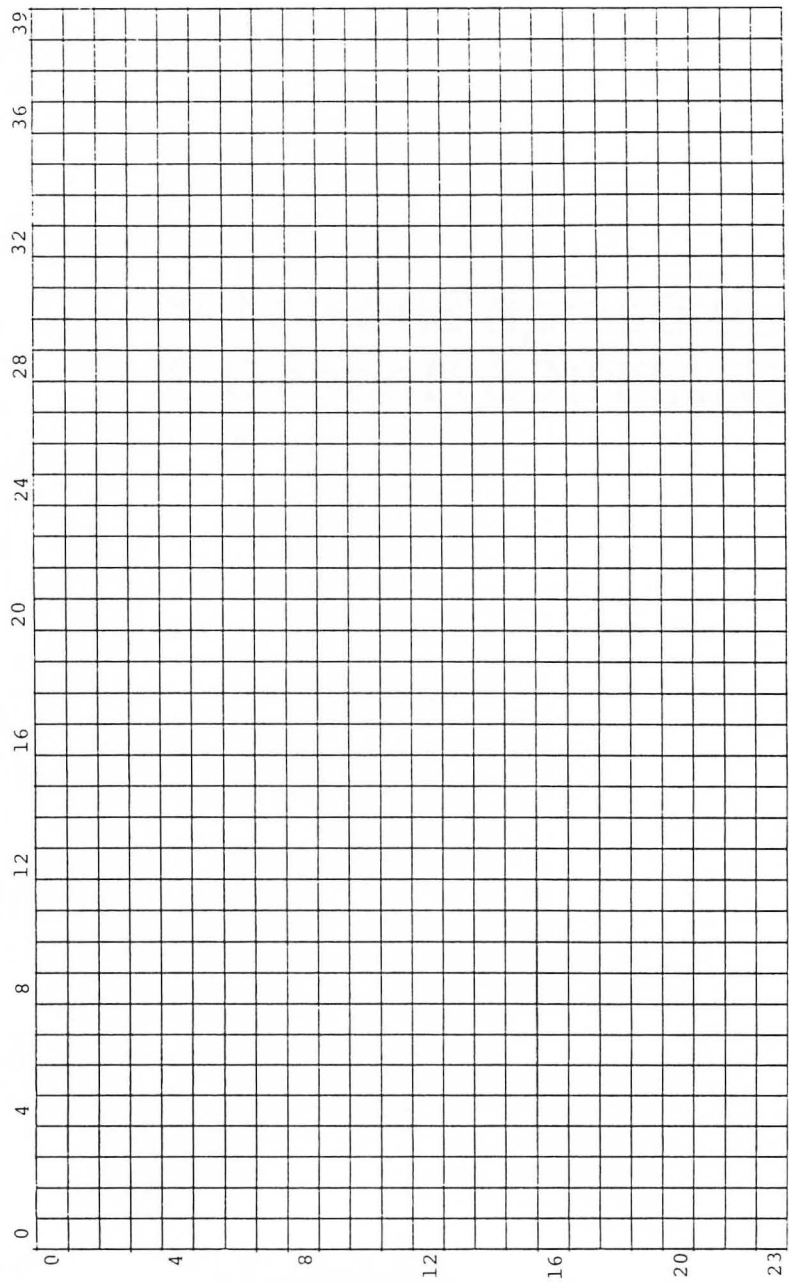
## Appendix VI

# Screen Design Sheets for GRAPHICS Modes 0-8



We wish to thank Mr. Bill Carris of the Home Computer Division of Atari, Inc. for his permission to reproduce these sheets. Mr. Carris developed them while he was working as Director of Atari Technical Services.

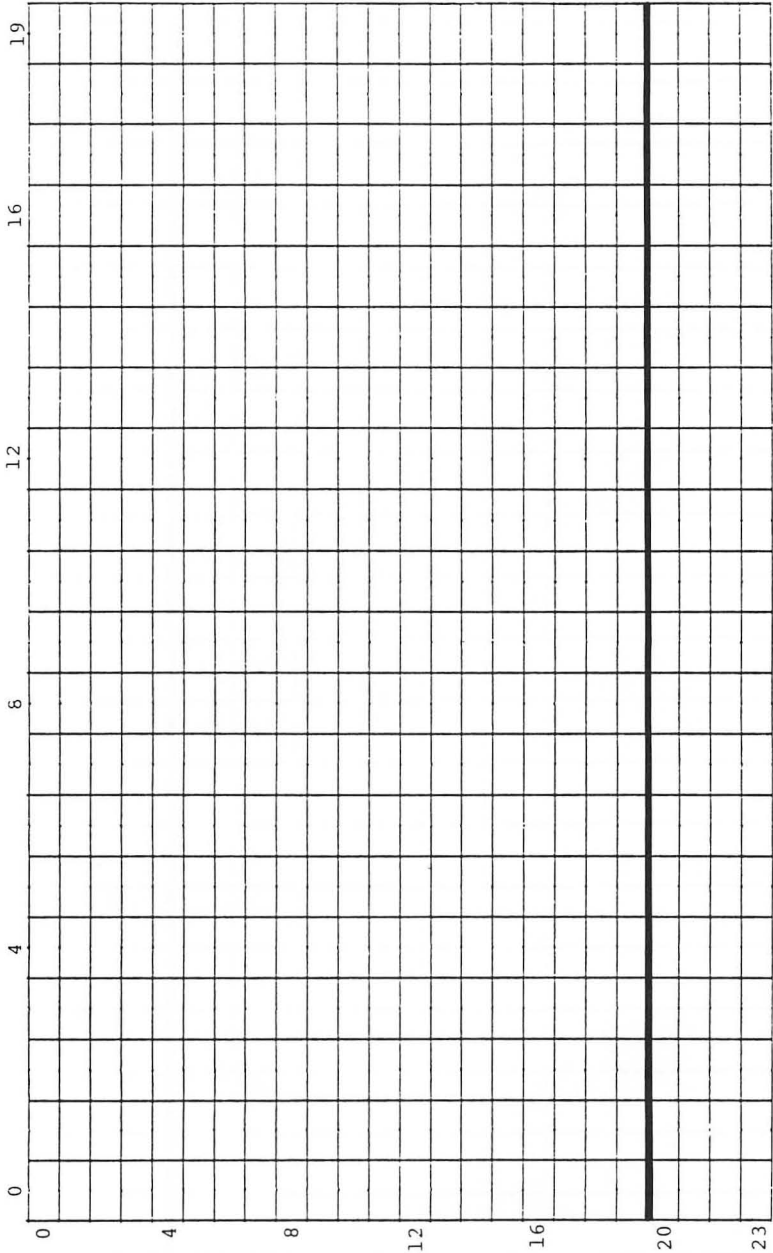
**Graphics Mode 0**



Notes: \_\_\_\_\_

**Graphics Mode 1**

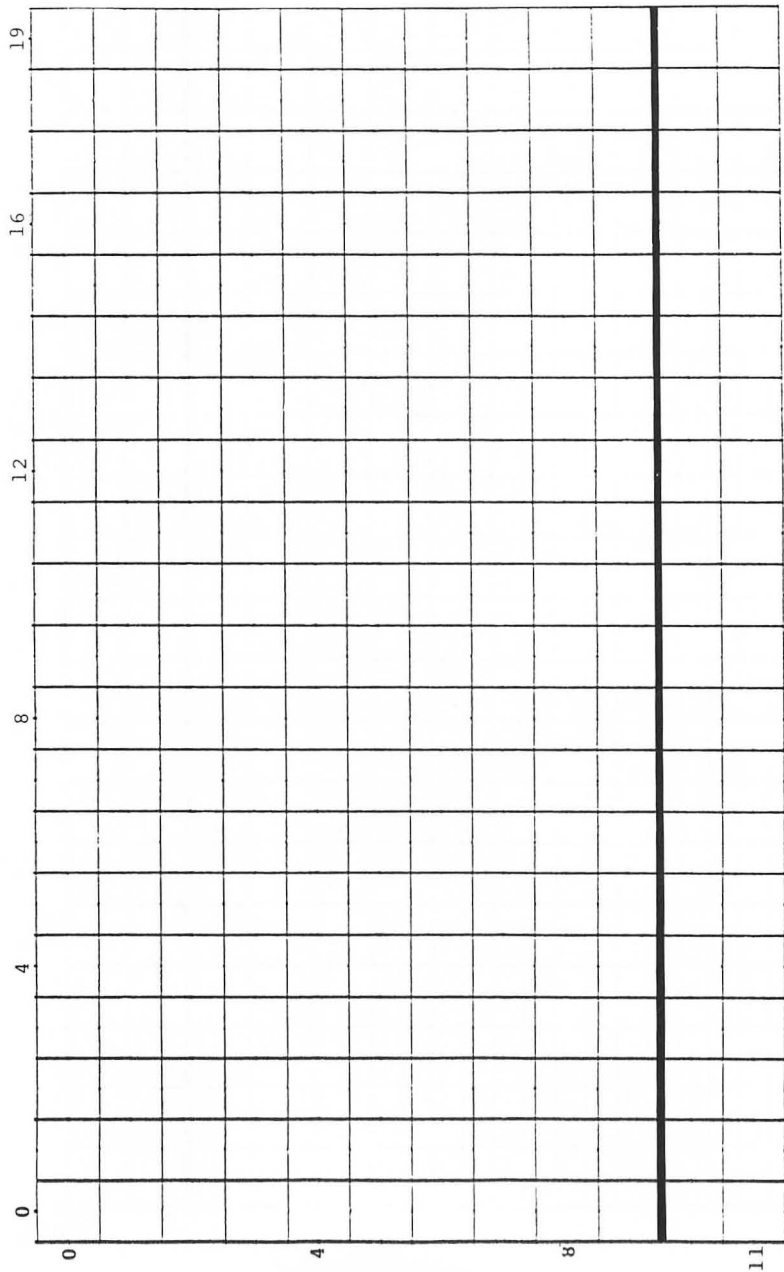
---



**Notes:** \_\_\_\_\_

**Graphics Mode 2**

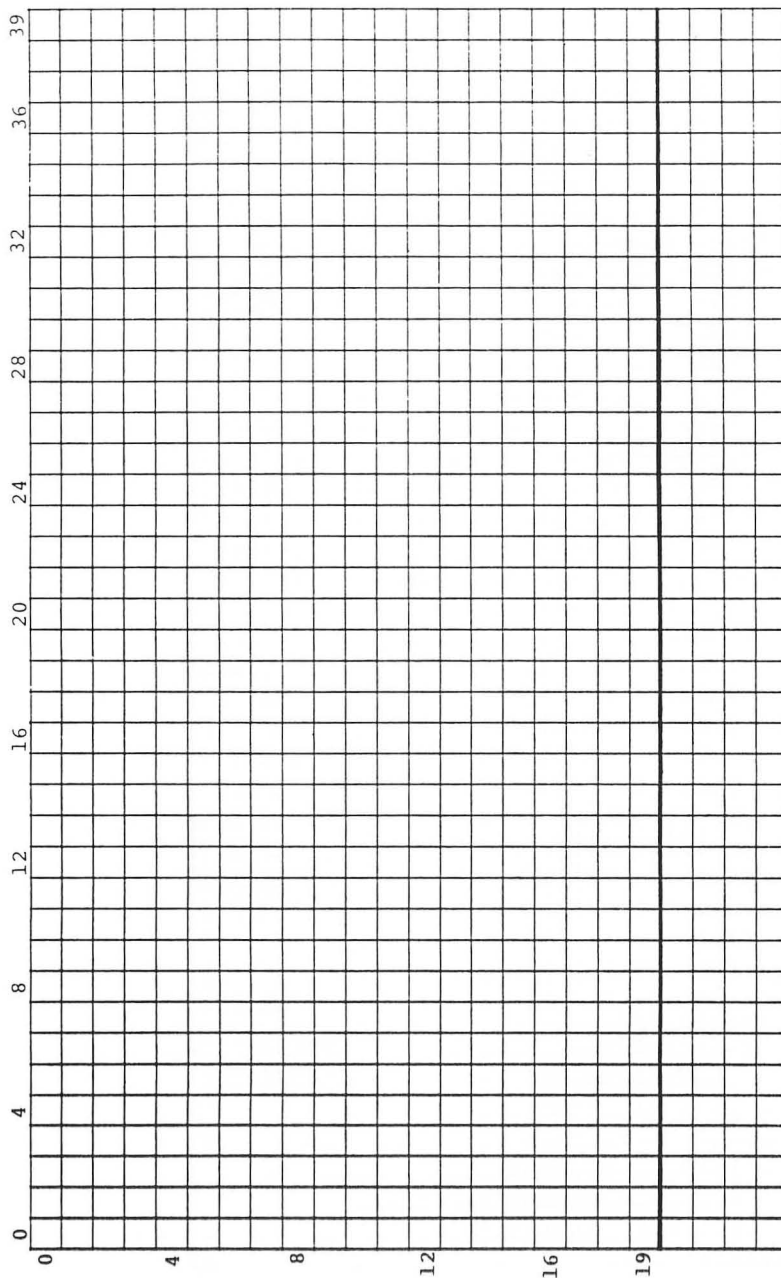
---



**Notes:** \_\_\_\_\_

# Graphics Mode 3

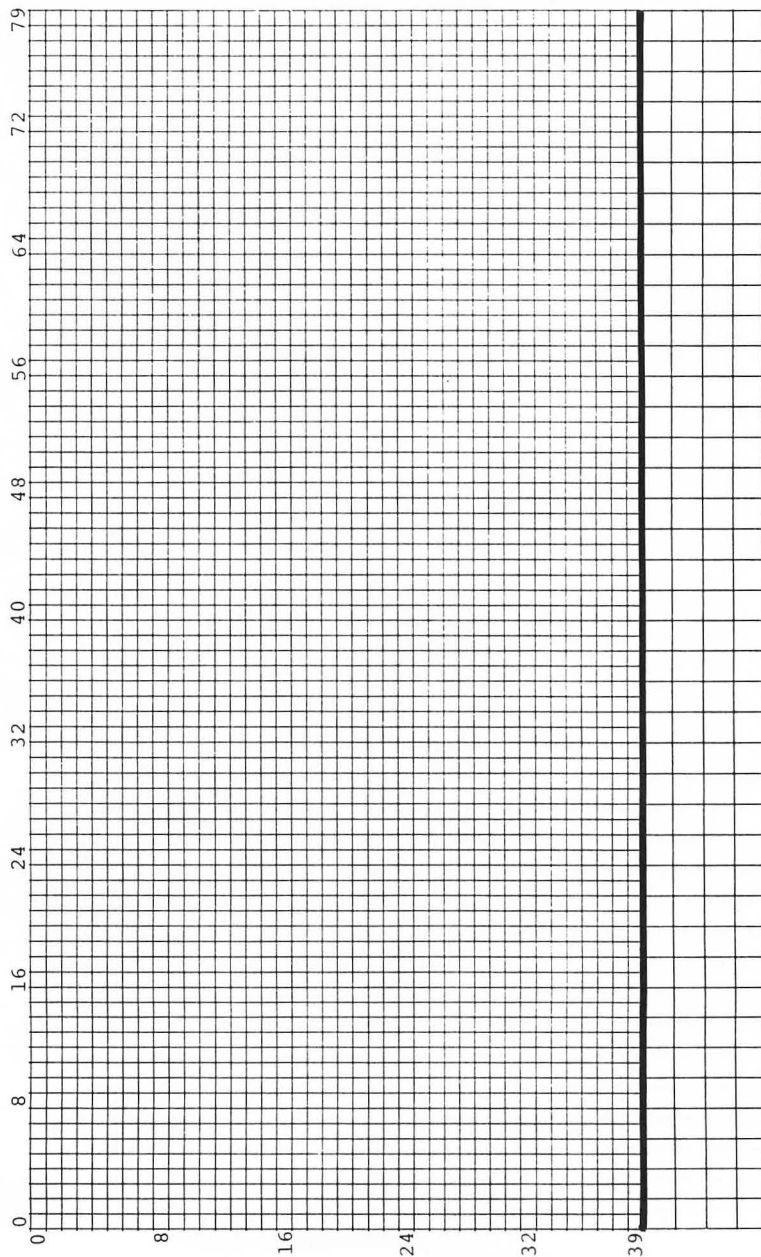
with Text Window



Notes: \_\_\_\_\_

## Graphics Mode 4 or 5

with Text Window

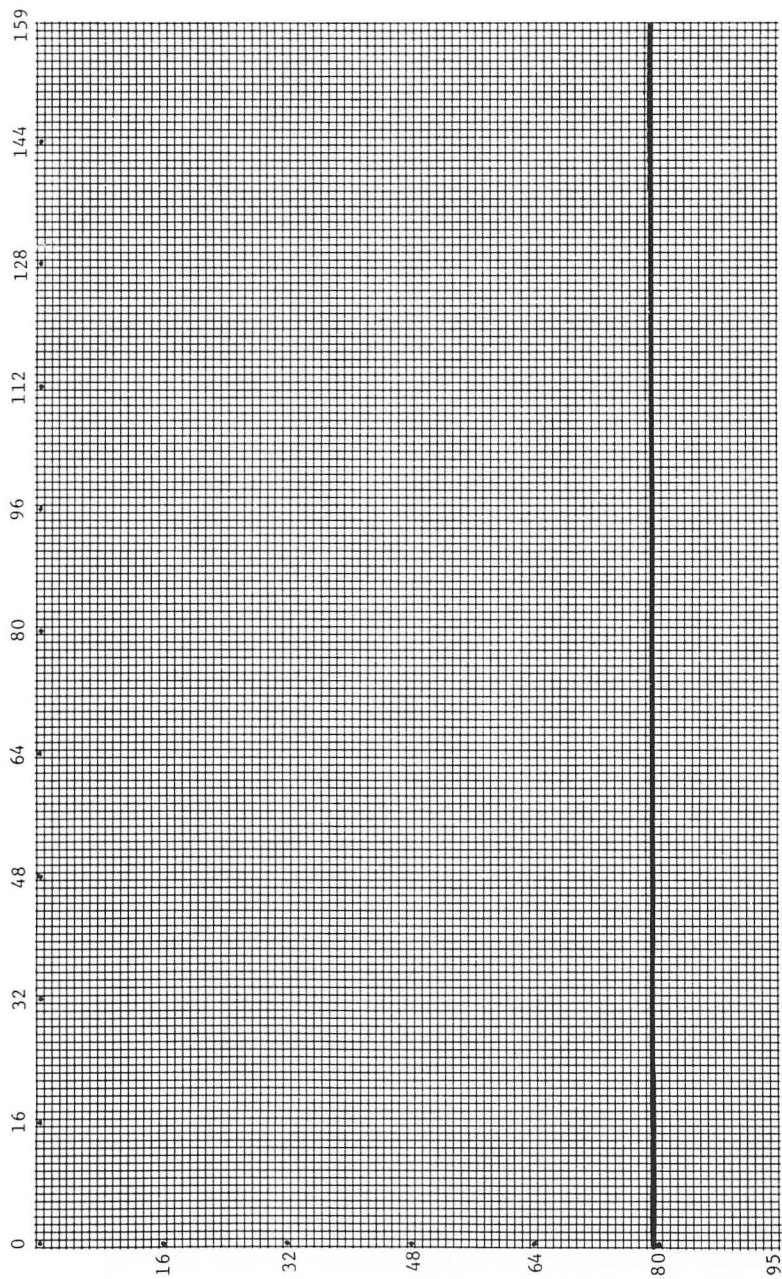


Notes: \_\_\_\_\_



## Graphics Mode 6 or 7

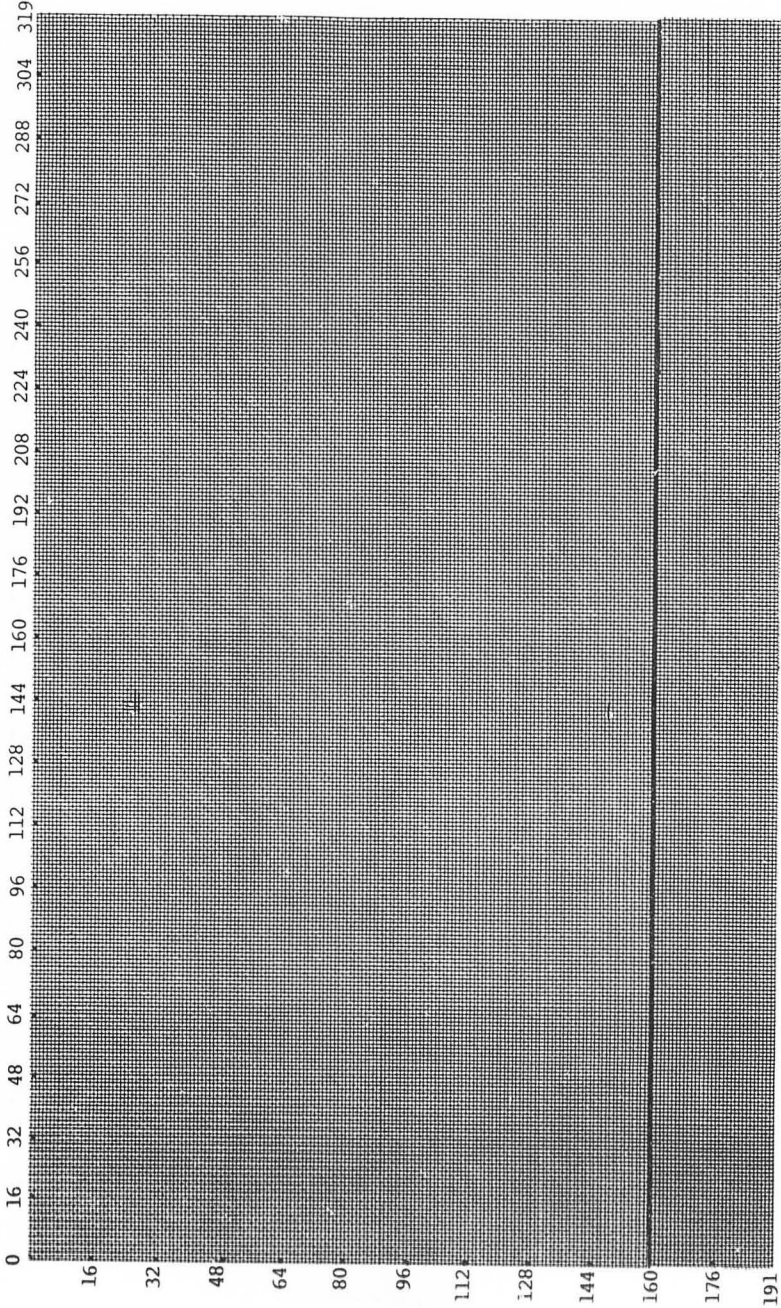
---



Notes: \_\_\_\_\_

**Graphics Mode 8**

---



**Notes:** \_\_\_\_\_

## Appendix VII

# ATASCII Character Codes

The following chart is designed to make conversion from ATASCII to character string easier. You may know what keys the user of your program is supposed to hit, but you may not know what their ATASCII values are. ATASCII Keyboard Chart to the rescue! Simply locate the key on the chart, and all possible ATASCII values are shown right on the key. NOTE that each of the 26 letters has 6 possible values, depending on the mode: shifted, reverse, or control. The lower left corner of the chart shows how to interpret each of these numbers.

Now, let's say for example, that you want the user to hit Y for yes, N for no, or H for help (it's nice to allow a HELP option). But you don't really care whether it is a capital or small letter, or even whether it is reverse or control. You simply want to know if one of those 3 keys was pressed (ignoring all others). First look up the keys and write down their values. In this example you would write:

Y = 89, 121, 25, 217, 249, 153

N = 78, 110, 14, 206, 238, 142

H = 72, 104, 8, 200, 232, 136

Now you simply write a subroutine to get a keypress, and another to check if it is one of the values you are expecting. If it is not a valid value, go back and get another keypress. If it is valid, return it as a capital Y, N, or H. This may sound involved and complicated, but it really is quite simple. Below is part of a sample program, including the two routines mentioned above.

UPPER CASE + 32 = LOWER CASE

REVERSE CHARACTER = NORMAL + 128

ESC 27	START & STOP DISPLAY	BUZZ 253	ERROR 136	—	—	—	—
	! 33 161	" 34 162	# 35 163	\$ 36 164	% 37 165	& 38 166	' 39 167
	1 49 177	2 50 178	3 51 179	4 52 180	5 53 181	6 54 182	7 55 183

CLR 158	┐ 17 145	T 23 151	┌ 5 133	— 18 146	● 20 148	I 25 153	— 21 149
SET 159	q 113 241	w 119 247	e 101 229	r 114 242	t 116 244	y 121 249	u 117 245
TAB 127	Q 81 209	W 87 215	E 69 197	R 82 210	T 84 212	Y 89 217	U 85 213

CTRL	┐ 1 129	+ 19 147	┌ 4 132	/ 6 134	\ 7 135	▲ 8 136	▴ 10 138
	a 97 225	s 115 243	d 100 228	f 102 230	g 103 231	h 104 232	j 106 234
	A 65 193	S 83 211	D 68 196	F 70 198	G 71 199	H 72 200	J 74 202

SHIFT	L 26 154	└ 24 152	J 3 131	I 22 150	12 130	_ 14 142
	z 122 250	x 120 248	c 99 227	v 118 246	b 98 226	n 110 238
	Z 90 218	X 88 216	C 67 195	V 86 214	B 66 194	N 78 206

CHARACTER	REG. CODE	REVERSE CODE	
Z	3	6	CONTROL
Y	2	5	SHIFT OR LOWER
X	1	4	CAPS

VALUES FOR: CHR\$(V)– GET # 1, V–POKE X, V–PEEK (X)– ASC (“X”)

— @ 64 192 8 56 184	— ( 40 168 9 57 185	— ) 41 169 0 48 176	CLEAR 125 CLEAR 125 ◀ 60 188	CHAR INST 255 LINE INST 157 ▶ 62 190	CHAR DELETE 254 LINE DELETE 156 BACK S 126	BREAK	SYSTEM RESET
▪ 9 137 i 105 233 l 73 201	▪ 15 143 o 111 239 0 79 207	♣ 16 144 p 112 240 P 80 208	↑ 28 - 95 223 - 45 173	↓ 29 l 124 252 = 61 189	RETURN 155		OPTION PEEK (53279)=3
▪ 11 139 k 107 235 K 75 203	▪ 12 140 l 108 236 L 76 204	♠ 123 251 : 58 186 ; 59 187	← 30 \ 92 220 + 43 171	→ 31 ^ 94 222 * 42 170	CAPS LOWR		SELECT PEEK (53279)=5
— 13 141 m 109 237 M 77 205	♥ 0 128 [ 91 219 , 44 172	♦ 96 224 ] 93 221 • 46 174	— ? 63 191 / 47 175	RVS OFF RVS ON	SHIFT		START PEEK (53279)=6

Chart designed &amp; compiled by Len Lindsay

SPACE  
32

ATASCII CHARACTER CODES

REVERSE SPACE  
160

©1980—Reprinted with permission—\*ATARI is trademark of ATARI Inc.

```

100 PRINT "YOUR CHOICE (Y,N, OR H):";
110 GOSUB 1000 : REM GET A VALID RESPONSE
120 IF C$="H" THEN GOSUB XXXX : REM ROUTINE
    FOR HELP
130 IF C$="Y" THEN GOSUB YYYYY : REM ROUTINE
    FOR YES
140 IF C$="N" THEN GOSUB ZZZZ : REM ROUTINE
    FOR NO ... [PROGRAM HERE]
999 END
1000 C$=" " : REM ROUTINE TO GET Y, N, OR H
1010 GOSUB 2000 : REM GET THE CODE OF THE KEY
    PRESSED
1020 IF CODE=89 OR CODE=121 OR CODE=25 OR
    CODE=217 OR CODE=249 OR CODE=153 THEN C$="Y"
1030 IF CODE=78 OR CODE=110 OR CODE=14 OR
    CODE=206 OR CODE=238 OR CODE=142 THEN C$="N"
1040 IF CODE=72 OR CODE=104 OR CODE=8 OR
    CODE=200 OR CODE=232 OR CODE=136 THEN C$="H"
1050 REM IF C$ STILL = " " THEN IT WAS NOT VALID
1060 IF C$=" " THEN 1010 : REM GET ANOTHER KEY
1099 RETURN : REM C$ NOW IS EITHER Y, N, OR H
2000 OPEN#1,4,0,"K:" : REM THIS COULD BE AT THE
    PROGRAMS BEGINNING
2010 GET#1,CODE: REM GET A KEY
2020 CLOSE#1 : REM NOT NEEDED IF LINE 2000 IS AT
    PROGRAMS BEGINNING
2099 RETURN : REM CODE IS THE VALUE OF THE KEY HIT

```

# Index

- ABS, 299, 301
- Addition drills, 82, 85-88, 89, 90-91
- Addition sign, 82
- ADR, 299, 301
- AIRPORT program, 261
- ALBERS program, 270, 286-87
- Algebra drills, 90-93
- ALPHABET CODE GUESSING GAME, 181
- ANAGRAM programs, 185-87
- AND, 299, 301
- Animated computer art, 314
- ANIMATION games, 229-32
- ASC, 299, 301
- ASTROLOGY game, 196-97
- ATARI BASIC cards, reference guide, 297-311
- ATARI BASIC Interpreter, 289
- ATARI logo key, 74, 95, 250
- ATARI printers, 182, 190, 197
- ATARI 410 Program Recorder, 6
  - saving a program on, 52
- ATARI 810 Disk Drive, 6, 52
- ATASCII character codes, 332-33
- ATN, 299, 302
- ATTRACT MODE, 313-14
- BASIC vocabulary, 4
  - conventions followed by, 27, 30
- BASIC vocabulary, explanations of, 10
  - A\$(X,Y), 97, 99, 104, 217
  - CLOAD, 52, 299, 301
  - COLOR X, 40, 45, 278, 281, 282, 299, 303
  - CONT, 8, 10, 49, 299, 304
  - CSAVE, 52, 299, 304
  - DIM A\$, 79, 80, 85, 95-97, 103-4, 291, 293, 299, 305
  - DRAWTO, 43, 45-46, 299, 305
  - END, 79, 81, 91, 103, 295, 299, 305
  - ENTER, 165, 169, 299, 306
  - FOR-NEXT, 38-39, 45, 47, 63, 86, 291, 294, 299, 300, 306, 311
  - FOR-STEP-NEXT, 55-56, 62-63, 299, 305
  - GOSUB, 59, 60, 61, 79-81, 102-3, 290, 293, 294-95, 300, 307
  - GOTO, 7, 10, 38, 223, 293, 300, 307
  - GRAPHICS 0, 79, 81, 249, 278, 279-80, 281, 295, 300, 307 (*see also* Text modes)
  - GRAPHICS 1, 64-66, 233, 276, 300, 307
  - GRAPHICS 2, 64-66, 233, 234-35, 276, 300, 307
  - GRAPHICS 3, 123, 128, 223-24, 226, 281, 282, 300, 307
  - GRAPHICS 4, 281, 282, 300, 307
  - GRAPHICS 5, 231, 281, 282, 300, 307
  - GRAPHICS 6, 281, 282, 300, 307
  - GRAPHICS 7, 40, 42, 44, 234-35, 281, 282, 300, 307
  - GRAPHICS 8, 245, 300, 307
  - IF . . . THEN, 28, 30-31, 38, 81, 83, 293, 299, 303, 308
  - INPUT A\$, 79, 292
  - INPUT X, 27-28, 30-31, 32, 53, 61, 95, 96, 97, 103, 292, 300, 308, 314
  - INT, 23, 31, 49, 300, 308
  - INT(RND(1)\*any integer), 23-24, 25, 31
  - INT(RND(1)\*any positive integer)+1, 24, 25, 27, 31, 126, 128
  - LET X=, 26-27, 31, 38, 95, 300, 309
  - LIST, 5, 7, 10, 67, 165, 169, 300, 309
  - LPRINT, 182, 190, 300, 310
  - NEW, 6, 10, 310
  - ON-GOSUB, 99-100, 103, 193, 300, 311
  - PEEK(X), 67, 69, 300, 311
  - PLOT X, Y, 42, 45, 300, 310
  - POKE, 69, 300, 310, 313-15
  - POSITION X, Y, 65, 66, 300, 309
  - PRINT, 6-7, 10, 53, 182, 190, 300, 309 (*see* Question mark)
  - PRINT #6;, 64-65, 233, 234, 300, 309
  - READ-DATA, 101, 104, 291, 292, 299, 300, 304, 308
  - READY, 5, 10
  - REM, 39-40, 44, 47, 57, 290, 300, 308
  - RESTORE, 291, 292, 300, 307
  - RETURN, 60, 61, 79, 80-81, 102-3, 294-95, 307
  - RND(1), 22, 23, 31, 126, 300, 307
  - RUN, 5, 7, 10, 300, 307
  - SETCOLOR Y, Z, W, 276, 278-79, 281, 282, 299, 301, 306
  - SOUND, 66-67, 299, 305
  - BASIC vocabulary, immediate mode for, 5-6
  - BIGLETS subroutine, 218
  - BINGO, 139, 140-44
  - Blinking square (*see* Cursor)
  - BRAINBUSTER, 139, 153-57
  - Branching, 28-31
    - definition of, 31
  - [BREAK] key, 8, 10, 11, 49, 78
  - BYE, 299, 302
  - [CAPS/LOWR] key, 65, 70
  - Character length, 8, 9, 46
  - Character strings
    - explanation of, 94-95, 193
    - function for, 97, 99, 104
    - storing in memory, 95-97
    - string length error, 291
  - CHORD BUILDER program, 265-67
  - CHR\$, 299, 301
  - [CLEAR] key, 14, 16, 20, 46, 57, 58, 63, 72, 74, 161
  - CLOG, 299, 303
  - CLOSE, 299, 303
  - CLR, 299, 303
  - Colon, 42, 46
  - COLOR (*see* BASIC vocabulary, explanations of, COLOR X)
  - COLOR & SOUND program, 268-69
  - COLOR PREFERENCE game, 200
  - Color programs, 276-88
    - ALBERS, 286-87

- COLOR DOTS, 288
- color plotting (*see* Color programs, TRIANGLE)
- colors available in, 276, 278, 301
- for demonstration of 128 ATARI colors, 276-77
- for demonstration of 128 color/luminance combinations (*see* RAINBOW) 282
- RANDOM COLOR MOSAIC, 281
- 3-D COLOR CUBE, 284
- TRIANGLE, 285
- using paddle controllers to play with color and sound, 288
- Commas, 9, 46
- Computer-composed stories, 192-93
- Control (*see* [CTRL] key)
- Control characters, 72
- COS, 299, 304
- [CTRL] key, 15-17, 19, 20, 72, 74, 95, 181, 249, 250
- Cursor
  - control keys for, 15-16, 19, 95
  - definition of, 14, 15, 21
  - out-of-range error, 295
  - relocating, 16, 20, 21
- Dancing dot program, 258
- Debugging (*see* Program debugging)
- DEG, 299, 305
- Delay loops, 56, 62, 68
- [DELETE] key, 16, 17, 18, 20, 21, 251
- Delimiters
  - colon, 42, 46
  - commas, 9, 46
  - definition of, 9
  - explanations of, 10-11
  - parentheses, 23, 24, 32
  - quotation marks, 6, 10
  - semicolons, 8, 9, 11
  - spaces, 8, 182
- DICE programs, 126-31
- DICTIONARY program, 165-68
  - games using 169-80
- Distortion control, 255
- Division drills, 84-85, 88, 89-90
- Dollar sign, 95, 96
- Dynamic computer art, 236-40
- Editing functions, 13, 15-19
- ENCODING & DECODING game, 182-84
- Error messages, 14-15, 290-96
  - definition of, 20
- Errors, types of, 289-96
  - array or string DIM error, 293
  - attempt to divide by zero, 293
  - cursor out of range, 295
  - input statement error, 292-93
  - line not found, 293-94
  - no matching FOR statement, 294
  - out of DATA error, 291-92
  - RETURN error, 294-95
  - run-time error, 289, 290
  - semantic error, 289
  - string length error, 291
  - syntax error, 289
  - value error, 290-91
- [ESC] key, 57, 58, 63, 72, 74, 95, 161, 251
- EXP, 299, 306
- Explosion programs, 260
- File devices, 295
- FOR (*see* BASIC vocabulary, explanations of)
- Fortune cookies, 197
- FORTUNE TELLER game, 194
- Fractured letterwriting, program for, 190-92
- Fractured tales, 188-93
  - program for, 189-90
- FRE, 299, 306
- Games
  - ALPHABET CODE GUESSING GAME, 181
  - anagrams, 185-87
  - ANIMATION, 229-32
  - ASTROLOGY, 196-97
  - being first to reach 100, 116-18
  - BINGO, 139, 140-44
  - BRAINBUSTER, 139, 153-57
  - COLOR PREFERENCE, 200
  - computer-composed stories, 192-93
  - DICE, 126-31
- Games, dressing up, 61, 64-75 (*see also* Programs)
  - calling player by name, 85-87, 97
  - congratulating player, 85-87
  - large letters and numbers, 79-81
  - "neon sign" or flashing advertising effect, 128
  - "prompt" messages in, 78
  - title page at beginning, 67-75
- Games (*see also* Programs)
  - ENCODING & DECODING, 182-84
  - FORTUNE TELLER, 194
  - fractured letterwriting, 190-92
  - fractured tales, 188-93
  - GOODNAME game, 234
  - GRAPHICS CODES, 182
  - GUESS THE CENTER, 226-28
  - horoscope, 196-97
  - MEDITATIVE WORD IMAGERY, 163-64
  - moving messages, 233-35
  - NIM, 119-25
  - number facts (counting), 78-79
  - number-guessing, 33-35, 111-15, 281
  - number and logic, 111-57
  - ORACLE, 195 (*see also* Games, COLOR PREFERENCE)
  - PHANTOM, 139, 145-52
  - plotting-guessing, 223-25
  - QWERT, 136-39
  - REVERSE, 132-35
  - secret codes, 181-84
  - SERIES, 175-80 (*see also* Games, types of, USSTATES)
  - SILLY QUESTIONS/ANSWERS, 198-99
  - spelling drills, 99-102
  - SUNTORED, 170-72
  - three-letter word, 165-80
  - TOESTITCH, 206-10
  - tortoise and the hare, 50, 52-61
  - USSTATES, 201-6
  - VOCABULARY, 210-16
  - word and guessing games, 159-218
  - word-guessing, 161-62
- GET, 300, 307
- GOODNAME game, 234
- Graphics characters, 72, 94-95, 181-82
  - program testing knowledge of location on keyboard, 182
- GRAPHICS CODES game, 182
- Graphics modes, 40, 44, 46, 65, 221-51, 278, 281, 301 (*see also* BASIC vocabulary, explanations of GRAPHICS 0-8)
- GUESS THE CENTER game, 226-28
- Horoscope game, 196-97
- HORRORS OF WAR program, 273-74
- HUE, 68
- Incrementing, definition of, 63
- INPUT (*see* BASIC vocabulary, explanations of, INPUT A\$ and INPUT X)
- [INSERT] key, 16, 18, 20, 21
- Inverse video letters, 72, 74, 95
- IRIDIS, 72
- JAZZ program, 270-72
- Jiffy clock, 317-18
- Joystick controller, 139, 265
- Keyboard buffer, 314
- KEYBOARD CHORDS program, 264



- Keyboard, features of, 4  
 arrow keys (*see* Keyboard, features of, cursor-control keys)  
 ATARI logo key, 74, 95, 250  
 [BREAK] key, 8, 10, 11, 49, 78  
 [CAPS/LOWR] key, 65, 70  
 [CLEAR] key, 14, 16, 20, 46, 57, 58, 63, 72, 74, 161  
 [CTRL] key, 15-17, 19, 20, 72, 74, 95, 181, 249, 250  
 cursor-control keys, 15-16, 19, 95  
 [DELETE] key, 16, 17, 18, 20, 21, 251  
 [ESC] key, 57, 58, 63, 72, 74, 95, 161, 251  
 [INSERT] key, 16, 18, 20, 21  
 [OPTION] key, 67, 226, 276  
 rapid repeat, 16  
 [RETURN] key, 5, 6, 7, 10, 18, 19, 21, 33, 39, 42, 52, 57  
 [SELECT] key, 67, 276  
 [SHIFT] key, 14, 18, 20, 21, 46, 57, 65, 70, 161, 251  
 [START] key, 67, 276  
 [SYSTEM RESET] key, 44, 46, 67, 78  
 KEYBOARD ORGAN program, 263  
 KEYBOARD PIANO program, 263-64
- LEN, 300, 309
- Line numbers  
 common error with, 293-94  
 definition of, 5, 9  
 function of, 5, 6, 7, 9
- Literal strings (*see* Character strings)
- LOAD, 300, 309
- LOCATE, 300, 310
- LOG, 300, 310
- Logical line, 46, 57  
 definition of, 9
- Loops, commands for  
 FOR-NEXT, 38-39, 45, 47, 63, 86, 291, 294, 299, 300, 306, 311  
 GOTO, 7, 10, 38, 223, 293, 300, 307  
 LET, IF-THEN, GOTO, 38
- Loops  
 definition of, 9  
 examples of, 28-31  
 setting up, 7-9
- Loops, types of  
 delay (WAIT), 56, 62, 68  
 infinite, 8, 9  
 nested, 53-54, 61-62
- MEDITATIVE WORD IMAGERY  
 game, 163-64
- Menu, creating, 97
- MESSAGE BILLBOARD game, 233
- Multiplication drills, 82-83, 88, 89
- Multiplication sign, 82
- Music programs (*see* Sound programs)
- Nervous lines program, 258
- Nested loops, 53-54, 61-62  
 error made with, 294
- NEXT (*see* BASIC vocabulary, explanations of, FOR-NEXT)
- NIM game, 119-25  
 program for, 123-25
- NOT, 300, 311
- NOTE, 300, 311
- Number and logic games, 111-57
- ON (*see* BASIC vocabulary, explanations of, ON\_GOSUB)
- OPEN, 300, 311
- [OPTION] key, 67, 226, 276
- OR, 300, 311
- ORACLE game, 195
- PADDLE, 300, 311
- Parentheses, 23, 24, 32
- Pause (*see* Delay loops)
- PEEK (*see* BASIC vocabulary, explanations of, PEEK(X))
- Peripherals, 295
- PHANTOM, 139, 145-52
- Pitch, 255, 256, 257-58, 265  
 program illustrating, 256-57
- PLOT (*see* BASIC vocabulary, explanations of, PLOT X, Y)
- Plotting guessing games, 223-25
- POINT, 300, 310
- POP, 310
- POSITION (*see* BASIC vocabulary, explanations of, POSITION X, Y)
- PRINT (*see* BASIC vocabulary, explanations of, PRINT)
- PRINT # (*see* BASIC vocabulary, explanations of, PRINT #6;)
- Program debugging, 290-96
- Programs (*see also* Subroutines, Games)  
 blinking title before changing color, 68-69  
 changing color of letters in a pulsating effect, 68  
 changing luminance level, 75  
 changing screen background color, 68  
 for choosing a random number, 22-27  
 clearing screen without erasing memory, 14  
 for color (*see* Color programs)  
 for colored hearts, 70, 72  
 color/sound show for winner, 116, 117  
 for continued calculation using one INPUT statement, 27-28, 32  
 correcting mistakes in, 13-21  
 correcting while writing, 17  
 definition of, 9  
 displaying title upside down every other blink, 69  
 drawing, 24  
 for drawing a moving face, 230-31  
 dressing up games (*see* Games, dressing up)  
 duplicating a BASIC statement (multiple line trick), 18-19  
 for dynamic computer art, 236-40  
 for a dynamic dictionary definition, 249-51  
 for graphics demonstration using straight lines and color, 241-44  
 using high resolution drawing, 245-48  
 giving hints to player, 113, 162, 226-28  
 for identifying three-letter words, 173-74  
 for interesting wave effects, 249  
 for keeping track of number of guesses, 113  
 for learning to estimate a correct answer (*see* GUESS THE CENTER game)  
 for learning harmony and music theory, 265-67  
 modular, 61  
 for music (*see* Sound programs)  
 numbering, 19-20  
 plotting points in different graphics modes (*see* Plotting guessing games)  
 removing from memory, 6  
 saving on Program Recorder, 52  
 for screen background and text window black color, 67-68  
 setting up loops, 7-9  
 for sound (*see* Sound programs)  
 switching from upper-case to lower-case letters and back, 69-70  
 for teaching the alphabet to children, 181  
 for testing knowledge of graphic characters, 182  
 using text and character graphics, 249-51  
 for three-letter word dictionary, 165-68  
 writing timing function for, 317-19  
 visualizing, 8  
 writing into memory, 5, 6, 7

- Programs, drill-and-practice type, 77-105  
 addition drills, 82, 85-88, 89, 90-91  
 algebra, 90-93  
 division drill, 84-85, 88, 89-90  
 for 5- to 12-year olds, 82-86  
 for 4-, 5-, and 6-year olds, 78-81  
 inductive thinking skills, 91-93  
 letter identification for young children, 98-99  
 multiplication drills, 82-83, 88, 89  
 spelling drills, 99-102  
 subtraction drills, 83-84, 88, 89  
 times table drill, 89  
 for two persons, rulemaker and player, 91-93  
 for upper elementary and junior-high grades, 91-93
- PTRIG, 300, 309
- PUT, 300, 308
- Question mark, 225
- Quotation marks, 6, 10
- QWERT game, 136-39
- RAD, 308
- Random line/sound pattern, 258
- Random number function (*see* BASIC vocabulary, explanations of, RND(1))
- Random numbers, plotting, 44
- READ (*see* BASIC vocabulary, explanations of, READ-DATA)
- Recognition routine, 165
- [RETURN] key, 5, 6, 7, 10, 18, 19, 21, 33, 39, 42, 52, 57
- REVERSE game, 132
- Program for, 133-35
- RND (*see* BASIC vocabulary, explanations of, RND(1))
- ROCKET LIFT OFF program, 259
- ROW, 74, 75
- Run-time errors, 289, 290
- SAVE, 299, 306
- Screen  
 changing margins on, 315  
 clearing without erasing memory, 14  
 color, 20  
 design sheets for graphics mode on, 325-32
- Screen-protection color change, 75
- Scrolling, preventing, 58
- Secret codes, 181-84
- [SELECT] key, 67, 276
- Semantic errors, 289
- Semicolons, 8, 9, 11
- SERIES game, 175-80
- SETCOLOR Y, Z, W, (*see* BASIC vocabulary, explanations of, SET-COLOR Y, Z, W)
- SGN, 299, 306
- [SHIFT] key, 14, 18, 20, 21, 46, 57, 65, 70, 161, 251
- SILLY QUESTIONS/ANSWERS game, 198-99
- SIN, 299, 306
- Siren program, 259
- Sound programs  
 AIRPORT, 261  
 CHORD BUILDER, 265-67  
 COLOR AND SOUND combinations, 268-69  
 the dancing dot, 258  
 explosion, 260  
 HORRORS OF WAR (sights and sounds), 273-74  
 interesting pitch, 256-57  
 JAZZ, 270-72  
 KEYBOARD CHORDS, 264  
 KEYBOARD ORGAN, 263  
 KEYBOARD PIANO, 263-64  
 nervous lines, 258  
 random line/sound pattern, 258  
 ROCKET LIFT OFF, 259  
 siren or whistle, 259  
 using voice, distortion, and volume, 262-64
- SOUND W, 254  
 range of, 255
- SOUND X, 254, 255  
 voices of, 254, 255
- SOUND Y, 254  
 pitch values for musical scale, 254-55
- SOUND Z, 254, 255  
 range of, 255  
 values for distortion control, 255, 262
- Spaces, 8, 182
- Special function characters, 72-73  
 graphics effects with, 249-51
- Special function keys, 276
- Spelling drills, 99-102
- SQR, 299, 305
- [START] key, 67, 276
- STATUS, 299, 305
- STEP (*see* BASIC vocabulary, explanations of, FOR-STEP-NEXT)
- STICK, 299, 304
- STOP, 295, 299, 304
- STRIG, 299, 304
- STR\$, 299, 304
- Subroutines, 290  
 clock starter, 317-18, 319  
 for creating 3-letter word dictionary, 165, 166-68
- definition of, 79, 102  
 for dice-throwing, 128, 130-31  
 error made with, 294-95  
 following flow between main program and, 81  
 pause, 102, 103, 229  
 scorekeeper for more than one player, 321  
 for timing a player's answer, 318-19  
 used in a simple drill, 79-81
- Subroutines, for word and guessing games, 217-18
- Subtraction drills, 83-84, 88, 89
- SUNTORED program, 170-72
- Syntax errors, 289
- [SYSTEM RESET] key, 44, 46, 67, 78
- Text graphics, 249-51
- Text modes  
 error with cursor in, 295  
 obtaining interesting wave effects in, 249  
 summary of, 222
- Text window, 278  
 definition of, 53, 61
- THREE-LETTER DICTIONARY program, 165-68
- Three-letter word games, 165-80
- Time, 317-19
- Times table drill, 89
- TO, 299, 303
- TOESTITCH game, 206-10
- TORTOISE & THE HARE games, 50, 52-61
- TRAP, 299, 303
- USR, 299, 303
- USSTATES game, 201-6
- VAL, 299, 302
- Variable names, 95, 290  
 A\$, 96-97, 217  
 HUE, 68  
 NAME\$, 85-86, 96, 97  
 WAIT, 56
- VOCABULARY game, 210-16
- Volume control, 255
- WAIT, 56
- Whistle program, 259
- Word and guessing games, 159-218
- Write-protected disk, 296
- X coordinate, 223, 224, 291
- XIO, 299, 302
- Y coordinate, 223, 224, 291





# Games & Recreations

**Herb Kohl   Ted Kahn   Len Lindsay**  
**with Pat Cleland**

**Atari Games and Recreations** offers a very different approach to introducing programming to the novice computer user. The authors encourage you to develop your own ideas for computer games and provide models from which to draw ideas for such games. You'll start with easy games that will serve as building blocks for more complex and creative programs. At the end of each chapter are sophisticated programs that will interest experienced programmers or beginners who want a challenge. You'll learn how to develop your own programming styles and have fun as you do it. And in the process you'll discover and master all the capabilities of your machine.

In addition to games, you'll find a special section on the graphics, sound, and color features of your Atari Home Computer System. You'll learn how to draw graphs, add color and sound to your games, and mix all the modes of the computer to create multimedia games and performances.

**Atari Games and Recreations** can also serve as a basic learning guide for kids and adults alike, and as a sourcebook for teachers. The book demonstrates how games form a beautiful domain for learning computer programming and that computer literacy need not be limited to the mathematically gifted or scientifically-minded user.

**Atari Games and Recreations** enables you to play with your computer and take power over it. Ultimately, you are the intelligence of your computer, so explore the games, experiment, and become a computer expert!

## Table of Contents:

**Part I:    LEARNING ABOUT THE ATARI PERSONAL COMPUTERS THROUGH GAMES**

1. Communicating with Your Computer
2. Theme and Variations: An Introduction to the Fine Art of "Dressing Up"
3. Drill and Basic Skills

**Part II:    GAMES FOR THE ATARI COMPUTER**

4. Number and Logic Games
5. Word and Guessing Games

**Part III:   THE ATARI SPECIAL**

6. Graphics
7. Sound
8. Color

Appendices

**RESTON PUBLISHING COMPANY, INC.**

*A Prentice-Hall Company*

Reston, Virginia

**0-8359-0242-0**