COMPUTE!'s SECOND BOOK OF

COMMODORE

64 GAMES



Sixteen games, from realistic simulations to challenging machine language arcade games, for the Commodore 64. Explore oceans, swat sharks, or run for president in the best games from COMPUTE! Publications.

A **COMPUTE! Books** Publication $12.95

COMPUTE!'s SECOND BOOK OF COMMODORE 64 GAMES

COMPUTE!™ Publications, Inc.
One of the ABC Publishing Companies
Greensboro, North Carolina

Commodore 64 is a trademark of Commodore Electronics, Ltd.

The following articles were originally published in *COMPUTE!* magazine, copyright 1984, COMPUTE! Publications, Inc.: "Quatrainment" (February), "Worm of Bemer" (April), "Olympiad" (June).

The following articles were originally published in *COMPUTE!'s Gazette* magazine, copyright 1984, COMPUTE! Publications, Inc.: "Astro-PANIC!" (February), "Haunted Mansion" (February), "CUT-OFF!" (March), "Sea Route to India" (March), "Props" (May), "SuperSprite" (May), "The Frantic Fisherman" (June), "Beekeeper" (July), "Campaign Manager" (August).

Copyright 1984, COMPUTE! Publications, Inc. All rights reserved.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the written permission of the copyright owner is unlawful.

Printed in the United States of America

ISBN 0-942386-64-7

10 9 8 7 6 5 4 3 2 1

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. Commodore 64 is a registered trademark of Commodore Electronics Limited.

Contents

Foreword v
Chapter 1. Ideas, Concepts, and Applications 1
Thinking It Through: How to Plan a Videogame
Orson Scott Card 3

Chapter 2. Text Adventure Games 23
Puzzles, Palaces, and Pilgrims: Writing Text Adventures for the Commodore 64
Gary McGath 25
Time Capsule
David Florance 44

Chapter 3. Strategy Games 61
Sea Route to India: A Historical Simulation for the 64
M.J. Winter 63
Quatrainment
Sean Puckett (64 Version by Gregg Peele) 76
Campaign Manager
Todd Heimarck 82

Chapter 4. Creature Features 121
The Frantic Fisherman
David Lacey (64 Version by Kevin Martin) 123
Beekeeper
Daniel Gray (64 Translation by Kevin Martin) 133
Props
Philip I. Nelson 139
Worm of Bemer
Stephen D. Fultz (64 Translation by Kevin Martin) 159

Chapter 5. Arcade Games 169
Brunhilde Loves Bruno
Philip I. Nelson 171
SuperSprite
Nick Sullivan 190
Olympiad
Kevin Woram and Mike Buhidar, Jr. 197

Burn Rubber
Jonathan Cook 204
Haunted Mansion
Calvin Overhulser (64 Translation by Kevin Martin) ... 211

Chapter 6. Machine Language Games 221
CUT-OFF!
Tom R. Halfhill 223
Astro-PANIC!
Charles Brannon 232
Nessie
Tom R. Halfhill (64 Version by Charles Brannon) 240

Appendices 255
A. A Beginner's Guide to Typing In Programs 257
B. How to Type In Programs 259
C. The Automatic Proofreader
Charles Brannon 261
D. Using the Machine Language Editor: MLX
Charles Brannon 265

Index 273

Foreword

Games are an important part of almost every home computer user's library. You *may* have bought your Commodore 64 to keep track of your finances or to learn how to program. But if you're like most people, you soon found out that computer games were challenging, educational, and simply fun to play. With its dazzling graphics, versatile sound chip, and large memory, the 64 is the quintessential game machine.

COMPUTE!'s Second Book of Commodore 64 Games gives you 16 games that make good use of those game-playing capabilities. Several of these games have never been published before; others are some of the best from recent issues of *COMPUTE!* magazine and *COMPUTE!'s Gazette* magazine. Each is ready to type in and play.

There are simulations that put you in the fifteenth century, rounding the Cape of Good Hope in search of trade routes to India. Or let you run for the presidency. A text adventure game, where you use words instead of a joystick, gives you a dangerous prison, far in the future, to explore. Action games put you in the cockpit of a Formula I racer, in a superhero's suit, or in a fishing boat besieged by sharks and thunderclouds. There are even fast-action machine language games with hyperactive aliens, the Loch Ness monster, or speeding trails of light.

But just like COMPUTE!'s best-selling *First Book of Commodore 64 Games*, this collection isn't merely for entertainment. There are articles that detail how to write an adventure game of your own and how to plan a videogame's design. And because each program is listed for you to see, you'll be able to examine the programmer's techniques and perhaps pick up methods to use in your *own* games. In fact, many of the articles include extensive explanations of how the games were written.

Typing these games into your computer, even the machine language games, is easy when you use "The Automatic Proofreader" and the "Machine Language Editor: MLX." Both are included in this book, along with instructions on how to use them. They'll virtually assure that you type in the games correctly the first time.

If you enjoy computer games, you'll find *COMPUTE!'s Second Book of Commodore 64 Games* filled with outstanding examples of the best from COMPUTE! Publications. These 16 games will provide you with hours of entertainment.

What more could a game player want?

1
Ideas, Concepts, and Applications

Thinking It Through

How To Plan A Videogame

Orson Scott Card

Designing and writing a videogame isn't just programming. Before you begin typing those POKEs, PEEKs, GOTOs, and GOSUBs, you should have an idea of what your game will do, how it does it, and why.

You don't create a videogame by just sitting down at the computer and starting to write BASIC or machine language commands. A videogame is a complex program, and if you don't think it through in advance, you're begging for hours and hours of needless revision and debugging—with a good chance that you'll end up with a second-rate game.

But if you have a well-thought-out plan before you start writing the first line of code, programming can be smooth and pleasant.

The plan I'm talking about isn't a matter of flowcharts and diagrams and dull calculations. That's the engineering approach to game creation. It works for building bridges or CPUs, but good games aren't engineered—they're created.

The process of planning a game may seem much like daydreaming. You visualize the figures, the scenes, the objects on the screen. You think of the way they move, the way they're controlled. What happens when they bump into each other. How they change each other. It's as though you leaned back in your softest chair and spent a few hours telling yourself stories.

Or, to put it another way, you're play-testing a game that hasn't been programmed yet.

Every gamewright creates games out of his or her own imagination; so no two games will ever be exactly alike, unless one game designer is deliberately trying to copy another's work. But the basic demands of designing are very similar from game to game.

So let's spend the next few pages planning a game together. At each stage, I'll remind you of how a particular arcade game handled the problem, and then decide how the game

we're creating together will do it. At the same time, you'll undoubtedly think of things I didn't think of, and by the time you're through reading this chapter, you may well have a complete game design of your own ready for you to sit down and start programming. I hope so!

Getting the Idea

A videogame idea can begin almost anywhere. You might be playing a conventional game and suddenly realize, "This could be better on the computer." So you begin to think of ways to simulate baseball or a board game or charades on the computer. It doesn't even have to be a game. How would you duplicate the work of a traffic cop and turn it into a game? Or a presidential election? (See the game "Campaign Manager" elsewhere in this book for the answer to that question.) Games that are based on conventional games or real-world activities are called *simulations*.

You might notice a setting that is particularly dramatic: the naked girders and beams of a skyscraper under construction; the dramatic arroyos and mesas of the Arizona desert; the vast distances and three-dimensional movements of outer space. It doesn't even have to be a setting that actually exists. The designer of *Joust* may well have gotten his flying islands from seeing Roger Dean's fantastic illustrations for the covers of Yes albums. Once you have a dramatic *milieu*, or setting, you develop the game from there: What could happen in this world I've imagined?

You might start with a movement, or *play mechanic*, you want to create. There are many different play mechanics already in use. For instance, the *Pong*-type games use paddle controllers to move the player's figure instantly from one point on a line to another. Many games use direct joystick movement—move the joystick in a direction and the figure moves that way on the screen. Others are much more complex, however. In *Joust*, a left-right joystick determines the horizontal direction and speed, but because the figure is a knight mounted on a flying ostrich, vertical movement is done by repeatedly pressing a button that causes the ostrich's wings to flap and the bird to rise. No one had ever used this play mechanic before, and the game may well have begun with that idea. Similarly, *Mario Bros.* introduced the "bump from beneath" play mechanic, in which the figure attacks enemies by getting one level

beneath them, jumping up, and bumping into the floor the enemy is walking on. This is only a small variation from the *Donkey Kong* play mechanic—yet it makes a huge difference in the feel and play of the game.

There isn't anything wrong in adapting ideas from other games, as long as you make enough changes that the game becomes your own. It's only natural for you to look at what another designer did with a game and think, "Why wasn't *this* done, too?" *Donkey Kong*, for instance, obviously owes a debt to *Space Panic*, a ladder game, by way of *Jump Man*, which introduced the press-the-button-to-jump play mechanic. But the *Donkey Kong* designers got rid of the holes and added moving obstacles—barrels and flames. The designer of *Lode Runner* (by Doug Smith, for Brøderbund) also started with *Space Panic*—you'll recognize the "stonework" floors and ladders—but he went in quite a different direction from *Donkey Kong*. Hole digging was kept, but other features were added, and each new screen became a new geometric puzzle. Both *Lode Runner* and *Donkey Kong* stand on the shoulders of the earlier game, *Space Panic*. But both are so different that they qualify as genuinely new and creative games.

Wherever your game idea comes from, though, you will eventually need to take all these things into account. You'll have to come up with a play mechanic and a milieu and adapt any real-world features that might be part of your game. And since you have known and loved (and probably hated!) quite a few videogames, you'll be borrowing or avoiding features from other gamewrights' work, whether you mean to or not.

The Story of the Game

Most fast-action games have a story, and since I'm a story writer by profession, it's hardly a surprise that this is where I start *my* game designs. Some stories are pretty rudimentary, like *Space Invaders* or *Asteroids:* The bad guys are coming, so either shoot or get shot. Others are more complex, like the story line of *Donkey Kong*: Ape catches girl, Mario climbs up and performs insanely heroic acts until he reaches girl, ape falls down, and girl kisses Mario. It's the variety of obstacles, milieus, and actions Mario must perform that give the story line its complexity.

Let's begin creating a game together—starting with a story.

You're a knight in a castle in the middle of a river, near the place where the river flows into the sea. An enemy has sailed

his fleet up the river and has anchored it around the castle, laying siege to your fortress. His primitive cannons are pounding away at the castle walls, gradually wearing them down. Even if you manage to survive the artillery barrage, you can't get any food supplies or ammunition into or out of the castle.

In the daytime, you can fight back by firing your four cannons at the enemy fleet. However, you have only a limited supply of ammunition. Every shot must count.

At night, you can put out a small boat and attach mines to the sides of enemy ships, just under the waterline. When the mines explode in the morning, any ships you've successfully mined sink. During the night you can also use your small boat to run the blockade, getting more ammunition from your confederates on shore.

When your castle is worn completely away, or when your little boat is sunk, you lose; when all the enemy ships are sunk or so damaged that they sail away, you win.

Not much of a story. No characterization and the plot is pretty repetitive. But that's deceptive. The player supplies the characterization. The main character is the player, of course. And the player supplies the plot complications. In early plays of the game, the guy in the little boat is going to be pretty clumsy and slow, and the cannoneer in the castle is not going to be much of a shot. But after a while, the boatman will be rowing circles around the anchored siege ships, and the cannoneer will score hits with every projectile.

How to Control a Cannon

Now that we have an idea, we need to expand on it, to think of how it might actually play as a game. We started with an idea—let's go on to the play mechanic.

What does the player control? There are really two parts to the game. In the daytime, the player controls the castle cannons. We'll make it a four-cornered castle, with a cannon in each corner. He can aim and fire only one cannon at a time. The cannon he's firing will be white; the other three cannons will be black.

How should he select which cannon he'll fire? We could have the player press the 1 key to fire cannon 1, the 2 key to fire cannon 2, and so on. But that forces the player to take his eyes off the screen and choose one of four different keys. Too complicated.

We could designate one key as the cannon select key. Just press it, and the next cannon would be selected, proceeding clockwise around the castle. That simplifies things—just one key to select. But remember, there also has to be a way for the player to aim the cannon *and* a way to fire it. That's at least two more controls.

The simplest choice of all, at least for the player, is to have the program automatically go from one cannon to the next. Fire cannon 1, and cannon 2 is automatically selected; fire cannon 2, and cannon 3 is then ready to fire. If cannon 3 is out of ammunition, the program can sense that and skip over it. It's simple, because the player doesn't have to *do* anything. He only has to notice the color change to know which cannon he's controlling.

Of course, it means that even if he wants to fire cannon 2 five times in a row, he can't; he has to fire each of the others first. But that just adds a little challenge to the game. You can explain this by adding another element to the story: "The cannons have to cool down once they're fired before you can reload them."

There's an even simpler solution, though. We can put a central tower in the middle of the castle (the castle's keep) and put a single cannon on top of it. Then there's no selection at all. Just the one cannon, which you aim and fire until your gunpowder runs out.

Simplicity

Four different ways to program the same basic idea. Which one is best? I don't know. I think the first two are too complicated for a fast-action game, but you might not think so, and they have the advantage of giving the player more freedom of choice. I think the last one, the single-cannon solution, is acceptable (and much easier to program), but having four cannons appeals to me more. If you left it just to my own preference, I'd choose the four cannons.

Does this choice seem familiar? It might. The arcade game *Missile Command* had three missile bases, each with its own supply of missiles. But when it was translated to the home machine, it was simplified: one base instead of three. We faced a similar choice here. Time after time you will have to make such choices between simplicity and complexity.

You want a general rule? If you're a novice programmer, go for simplicity—it's easier to program.

You want another general rule? If you're an expert programmer, go for simplicity. It's also easier to play.

But I like the four cannons, automatically selected, better than the single cannon in the middle of the fort. So I'll break my own general rule and go for slightly more complex programming—but never for needless complexity in playing. Maybe that's the best rule of all: Make it as tough as you want on yourself, but keep the play mechanic easy on the player.

Aiming the Cannon

All this, though, and we haven't even decided how to *aim* the cannon!

We could have the player type in the coordinates on the screen where the cannon is aiming. But that's very complex and way, way too slow. We could use the joystick to choose one of eight directions. But what if the target isn't in one of those eight directions?

I go for the rotating cannon. Press down the Cursor Left-Right key, and the cannon rotates clockwise. Let the key up, and the cannon stops rotating. Wherever it's pointing when you stop is where it aims. It will take only a little trial and error for the player to learn to aim pretty exactly. (However, rotating cannons will mean we have to have as many different cannon shapes in memory as there are possible directions to aim! Every decision has a cost.)

Press another key and it fires. Done. Two keys, one to aim, one to fire.

But wait a minute. In the real world (this is partly a simulation, remember) cannoneers also have to determine range. They control range by deciding the elevation of the gun and changing the amount of gunpowder in the charge. You could control that by. . . .

Forget it. This is a game, not an artillery textbook. If you're a cannoneering purist, you might want to go through the agony of programming all the math. But you'd better find another purist to play it, because the game will be complex and slow. As far as I'm concerned, I'll let an imaginary cannoneer figure the elevation and charge. As long as the player selects the correct horizontal aim, the cannonball will automatically go just the right distance. Another victory for the simple approach.

The Computer Control Movements

There are other play mechanics to worry about. First, the movement of the enemy ships. Do they just stay in the same place, or do they move? Do they always appear in the same places every time you play, or are their places randomly chosen?

There are many different options, but my choice would be this: The program will define 20 or so areas, each of which might hold a ship. By defining these areas, we can be sure none of the ships will overlap. However, we'll start with only five enemy ships. The program will randomly decide which areas will be a slightly different arrangement of the enemy ships from the time before. That will help keep the game interesting and challenging.

If we were programming in machine language, it would be a simple matter to assign each ship its individual course around the castle and then let it move in a regular pattern. At machine language speeds, you could keep a dozen ships moving without slowing down the game. In BASIC, however, each ship movement would slow down the game. And since I just decided this will be an all-BASIC game, we can't have the ships move.

At least, not constantly. Anything that happens constantly adds to the amount of time that passes *after* the program has checked for the player's instructions and *before* it checks again. The less often the program checks for the player's instructions, the slower and less responsive the game will feel. Too much of that, and it isn't a fast-action game anymore!

But we can have *occasional* movement of the ships; the story provides a perfect excuse. The current of the river always pushes the ships downstream, but as the tide flows upriver, the current moves in the opposite direction. The tide flows twice each day, once in the daytime and once in the night. When the tide flows, the ships swing around so that they are on the upriver side of their anchors. When the tide ebbs and the river's current takes over, the ships swing around to the downriver side of their anchors.

That means that half the day all the ships will face one way, and half the day they'll face the other. It's a simple change, but it adds to the completeness of the milieu; it supports the story, and it helps the realism of the simulation.

Controlling the Boat

In a two-phase game like this, it's good if the two phases can be controlled in a similar way. For instance, we can let the same key that rotates the cannon for aiming also rotate the boat. Press the Cursor Left-Right key and the boat rotates counterclockwise. Press the key we used to fire the cannon, and the boat leaves behind a mine.

We've got directions and mine-laying taken care of, but what about movement through the water. We need a third control, to give the boat speed. We could steal a page from *Joust* and let the oars move each time we press a certain key—the space bar, for instance. Each time we press the bar the boat gives a surge in the direction we're pointing the boat. If the boat is going against the current, the surge is weaker; if with the current, the surge is much stronger. And if we don't row at all, the boat will drift in the direction of the current. If the boat leaves the screen, it's gone!

That's one solution. If I were working in machine language, it's the one I'd probably use. There is a simpler choice, however. You could use a joystick throughout the game. During the cannon phase, pushing the joystick in *any* direction makes the cannon rotate, and pressing the fire button makes it fire. In the boat phase, however, the boat moves exactly in the direction the joystick points and keeps moving in that direction as long as the joystick is pushed. The fire button attaches mines. This has the virtue of being simpler. Also, since the boat can move only in either four or eight directions, depending on whether you allow diagonals, you need to have only one boat shape for each direction. Programming and play will be much easier. It was important to have the cannon be able to aim in many different directions because it couldn't move, it could only aim. Since the boat *can* move around, having only four directions of movement isn't a serious limitation. And we can keep the tidal drift in both versions.

The Planning Outline

I've gone into a lot of detail up to this point in order to show you what the thought process can be like—how many options you can think of and reject and the reasons for some decisions. From now on, so that this article doesn't become the whole book, we'll move much more quickly through an outline of the

decisions that need to be made. You can use this outline for almost any arcade game plan.

Play mechanic. What does the player control in the game, and how? (This is what we've already discussed; we ended up with a cannon and a boat in the different phases of the game, both controlled with a joystick.)

Simulation. In what ways does the game correspond to real-world activities? To what degree can you duplicate reality without making the game too complex to play? (This was when I decided to let the cannon aim, but not determine range, and when I decided to use the joystick to move the boat in only four possible directions.)

Milieu. What is the setting? What do we see on the screen besides the moving figures? This is more than just decoration. If you have an airplane game, clouds drifting across the screen add to the illusion of reality. It makes the player feel more like he's really flying. (In our game, let's make the flow of the river left to right. That means that the banks of the river will be across the top and bottom of the screen. By using character graphics, we can PRINT each shore in a single string! The castle will be right in the middle, but because the TV screen is wider than it is tall, the castle will be rectangular, too. It will be a top view, as a bird sees it.)

Missiles. This is a generic term. The player's figure is the screen object whose movement the player controls. Once a missile is launched, however, the player has little or no control over it. The missile can be the ball in a football or baseball simulation, the bullet in a shoot-out, or even the player-figure's fist if it can be "launched" against an enemy. (The only missile we use is the ball fired by the cannon, which goes straight in the direction it's fired until it either hits an enemy ship or goes off the screen.)

Collisions. What happens when the player-figure bumps into something on the screen? The figure can respond to the object it bumped into as if it were:

1. Transparent. The figure just keeps going as if the object weren't there.
2. A wall. The figure can't move any further *toward* the object, but if it collides at an angle, it can slide along it.
3. A tar baby. Once the figure touches the object, it's stuck there.

4. A bomb. Touching the object is deadly.
5. A billiard cushion. Touching the object makes you bounce off at an angle.
6. A balloon. When you touch the object, it disappears, but you're just fine.

NOTE: The same responses are possible for missiles.

During the boat phase, the shore and the castle are walls—the boat can slide along them. However, the enemy ships are tar babies—when you touch one you stick until you attach a mine and release yourself. The ships are then marked, and at the end of the nighttime phase, all the mines explode at once.

During the cannon phase, only the cannonball treats the ships as balloons—they explode when the ball hits them but everything else is transparent. Since I don't want the game to be that easy, the castle and the shore are also balloons if the cannonball is fired so that it collides with them. You yourself, by clumsy aiming, can destroy the castle *and* the shoreline. This means that when you go to the shore to replenish your supplies, it might not be there anymore. And if you're really clumsy, you can help the enemy finish off your own castle.

But what about the enemy's cannonballs? To save programming headaches, let's eliminate them. Instead, from time to time a randomly selected, character-sized piece of the castle will explode and disappear, leaving bare rock behind. It can be assumed that an enemy cannonball hit that spot.

Reward and punishment. Games are like life. You obey the rules because when you do, "good" things happen, and when you don't, "bad" things happen. The most common reward is the score—it gets higher each time you do the "good" things, like blasting the enemy out of the water. There are other rewards, however. Story awards, for example. In *Donkey Kong*, Mario gets the girl. There are also puzzle awards. Just solving the problem on one level, so that you can finish it, is very rewarding. The best games have scores, story rewards, *and* puzzle rewards.

(In our game, if you don't ever get the enemy ships, they'll wipe out your castle. If you do get them, you get points and eventually complete the screen by forcing the enemy ships to go home or by sinking them all. The scoring is fairly complex. Every time one of your castle blocks is destroyed, you lose points; if one of your cannons is blown up, you lose the cannon and a lot more points. However, you get some points just for

staying alive, and for the number of cannonballs you have left at the end of a game "day.")

Communication. The player needs to be told a lot of information during the game. Did my missile hit its target? Did the enemy score a hit against me? What in the world am I supposed to *do*? Which object do I control? The single most useful tool you have in communicating with the player is *sound*. Different sounds mean different things—and you don't have to be watching a particular spot on the screen to get the message. However, explosions and movement also communicate. You'll also use displays of numbers on the screen to tell the player his score—numbers or little pictures (icons) to show how many "lives" the player has left. You'll want introductory and closing screens to give more involved messages or tell part of the story in words.

(In our game, a popping sound says that the cannon has been fired or the mine has been attached; a low "boing" says that a piece of the castle has been blown up; a swishing sound says that the tide is about to change; an explosion says that a ship has been hit; a much louder, long one, followed by a glug-glug sound, says that a ship has sunk; and sad and happy music signify defeat and victory. Explosions are shown by making a character flash white several times. The score is constantly displayed on screen.)

Win-lose conditions. The game has to end sometime, even if you're ending only a current level. You have to decide what conditions end the game, and then check from time to time to see if those conditions have been met. The simplest way to do this is to have a variable—XX, for instance—that usually has a value of 0. Then, in any subroutine that has the power to end the game (usually a collision subroutine or a timer subroutine), XX is set to 1 for defeat and 2 for victory. A line in the main loop reads ON XX GOTO 900,920. That jump will occur only when it's time for the game to end. But because it occurs only from the main loop, it's much easier to end the FOR-NEXT loops you might be in at the time.

(In our game, you can lose when all four of your cannons or your entire castle is destroyed, or when you've used up all your ammunition and mines without destroying the entire enemy fleet. You can win by scoring a certain number of cannonball hits against the enemy and/or by sinking a certain number

of ships and/or by staying alive until the enemy runs out of ammunition.)

Levels. Computer games almost always go on forever. When you meet the win conditions, the game starts over, but it's harder. This allows a novice to get the hang of the game without getting instantly destroyed, while more experienced players still find higher levels more challenging.

(In the early levels of our game, the enemy will have less ammunition; the player cannot blow up parts of his own castle; there are fewer enemy ships so that the enemy fires less often and the player can blow up all the ships more easily; and the enemy cannot hit the player's cannons. These features are changed with each level, until at expert levels the player doesn't have enough ammunition to survive without picking up some from the shore and bringing it back; there are as many enemy ships as can fit on the screen without overlapping; the player can damage himself; the cannons can be blown up; the current in the river is stronger, making it harder to control the boat; and the enemy ships can withstand more hits from cannonballs before they have to go home to repair the damage.)

Animation. If you're particularly ambitious, there are a lot of extras you can add to enhance realism. These things don't actually affect the play of the game, but they *do* make it more fun to play. Simple animation of figures on the screen is easy enough, using either sprites or custom characters, and it can be done with almost no reduction of playing speed. However, you can also add much more complex animated sequences when play action is stopped. These are like small movies that help support the story—like the opening sequence of *Donkey Kong* in which the gorilla carries the kicking girl up the ladders—or make the milieu complete and believable—like the riderless ostrich in *Joust* that must make its way offscreen after the knight is defeated before they can reappear together on a platform. The game would play just as well without these extra sequences, but some of the fun would be gone.

Translate the Plan into a Program

This is where the fun turns into work. You've jotted down your ideas, you've play-tested the game in your imagination, you're satisfied that it's wonderful, and you can't wait to play.

Now you've got to stop and do your homework. Not fun at all.

Setting Up Video Memory

How large should the castle be? The screen is 40 characters wide and 25 characters high. Subtract a row at the top and the bottom for the shoreline, and a column at the right that we won't use, because PRINTing characters in the rightmost position on a line can mess up the lines below. To leave room for maneuvering, the castle will be only 5 characters high and 9 characters wide. Each corner tower will be a 2 × 2 square, and each cannon will be a sprite. The cannons can point in 16 possible directions, animated by changing the shape-table vectors at the end of the screen memory. The player-controlled boat will also be a sprite, with 4 possible positions. The enemy ships will be complex custom characters—each ship will be made up of 2 characters put together. They will be PRINTed on the screen as strings. They will usually face either left or right, and all the ships will change direction together, which greatly simplifies animation of the direction changes.

To make all of this work, you need to map video memory and design your character sets and sprite shape tables before you even begin to program. Lots of tedious calculation, changing dots into bits and bits into decimal numbers, loading different shape table vectors into arrays that correspond to directional numbers.

And that's the easy part.

Setting Up the Main Loop

You need to design the program so that the main loop does as little as possible. The less the main loop does, the more often it repeats, and the faster the game plays.

This game is a little more complex because it has two main loops, one for the daylight phase and one for the nighttime phase. A timer decides how long each phase lasts. Both main loops must check, from time to time, to see if the time is up and the tide should change or the phase should end.

What needs to be in the main loop? *Always*, the routine that gets instructions from the player. In the daylight phase, we check first to see if a cannon was fired. If yes, then jump to the subroutine that carries the cannonball to its target, where there either is or isn't an explosion. In machine language, we could let the player start aiming or firing the next cannon while the cannonball kept moving, but in BASIC it's better to make the player wait for the cannonball to hit before letting

him fire again. BASIC would lose too much speed trying to do it all at once.

If the player didn't say to fire, then we check to see if he wants to aim. If so, we jump to the subroutine that moves the cannon.

When that's over, we check to see if it's time for an enemy ship to fire again. If it is, we jump to the other main loop. Otherwise, we go back and start over. Simple enough—the main loop will be fast and tight.

The main loop for the nighttime phase is not so tight. The fire button only places a mine (there's no missile to keep track of), but movement is more complex. The cannon can never run into anything, but the boat can run into enemy ships, the shore, or the castle. The main loop must check to see if the player wants to move. If so, we jump to the movement subroutine. The subroutine checks to see if the boat has bumped into anything, and if so, what. It might jump to one of the routines that handle collisions, and the movement is changed accordingly. Also, there is the pull of the current that can gradually move the boat whether the player wants it to move or not. This can't be executed every time through the loop, or the player will never be able to row against the current. Finally, we check for the end of the phase and close the loop.

Planning for All the Subroutines

Programming goes much more smoothly if you've decided before you start what subroutines you'll need and where they will be. It helps you keep track of what's going on if you put related subroutines near each other. For instance, in a BASIC program you might want to have all the sound subroutines begin in the 900s, and all the collision subroutines in the 800s.

I usually start every subroutine or group of subroutines at an even 100 line number. Let's chart all the routines that we'll need in this game program.

0–99 Initialization. These lines jump to the setup routines and set certain parameters. They're executed only once.

100–199 Daytime phase main loop. These lines check for the player's instructions, jump to the aim or fire routines if necessary, then check the timer for the end of the phase.

200–299 Fire routine. These lines are executed only if the player has pressed the fire button during the daytime phase main loop. First the routine checks to see which cannon has

been fired and what direction it is firing. Then it moves the cannonball in that line, checking each new character it crosses to see if it has collided with anything. If it collides with a ship, it jumps to the ship explosion routine. If it collides with the shore, it jumps to the shore explosion routine. If it collides with the castle, it jumps to the castle explosion routine. If it reaches the edge of the screen without collisions, the cannonball disappears and we return from the subroutine. In any event, whenever the cannon fires, we also execute the decrease ammunition routine.

300–399 Ship explosion routine. Accessed only from the fire routine, these lines subtract something from the ship's "strength" value, which starts out higher in the harder levels. If the ship's strength is 0 or less, we jump to the sinking ship routine. If the ship's strength count is above 0, we execute the explosion sound and flash the colors of the ship, cancel the missile's movement, jump to the score change routine, and return to the daytime phase main loop.

400–499 Shore explosion routine. Accessed only from the fire routine, at the easier levels these lines do nothing but end the missile's movement and return to the daytime phase main loop. At higher levels, the shoreline character is flashed while the explosion sound is executed, then the shoreline character is removed.

500–599 Sinking ship routine. Accessed from either the ship explosion routine or the tidal change routine, this animated sequence causes the ship to sink, decrements the count of ships by one, executes the glub-glub sound, jumps to the score change routine, and then returns to the current main loop.

600–699 Aim routine. Until the joystick is no longer being moved or the fire button is pressed, the current cannon is rotated. This is done by flipping from one cannon sprite shape to another. The same variable that indexes the sprite shapes also indexes the direction variable, which is used in the fire routine. If the joystick is released, we return to the daytime phase main loop; if the fire button is pressed, we jump to the fire routine, then return to the main loop.

700–799 Score change routine. This routine can be accessed from many points, but the score change variable must already be set. The value of the score change variable is added to the current score (a negative value will, of course, subtract from the score) and the new score is displayed on the screen.

800–899 Tidal change routine. This routine stops all other action while the ships swing around from their old positions to their new positions. The current flow variable is set to either *left* or *right* (−1 or 1). Then we check to see if the changing tide has caused a collision between a ship and the player's boat. If yes, we jump to the sunk boat routine. If it is also time to change from one phase to another (which will happen every second tidal change), we set the phase change variable, which causes one main loop to jump to the starting point of the other. If the tidal change is the beginning of a new daylight phase, check all the enemy ships. If their mine set variable is on, then execute the sinking ship routine for that ship.

900–999 Enemy shots routine. These lines select a random location on the perimeter of the castle. If the chosen character is bare ground—that is, if the chosen castle section has already been exploded—there is a chance that the next castle section inward will be selected. If that castle section has also been selected, then there is a chance that the *next* castle section will be selected, and so on. Anytime the random choice decides not to select the next castle section, we jump to the castle explosion routine. If the chosen castle section is also an ammunition storage location, we jump to the decrease ammunition routine. At higher levels, if the chosen castle section is also directly under a cannon, we jump to the blow up cannon routine. The odds of the next castle section being chosen change, making the selection more likely at higher levels, which will have the effect of wiping out the castle faster.

1000–1099 Decrease ammunition routine. This decreases the supply of cannonballs during the daylight phase and the supply of mines during the nighttime phase. If the amount of ammunition reaches 0, then the player can't fire (or attach mines) until the supply is replenished by a visit to the shore.

1100–1199 Castle explosion routine. This causes the selected castle section to flash and disappear, leaving bare earth behind. (Actually, it's tempting to have water characters replace exploded castle sections, so that the castle appears to be eaten away by the river as well. Something to think about.) The oops sound executes.

1200–1299 Blow up cannon routine. These lines cause a cannon to flash, crumble, and vanish. That cannon is removed. The explosion sound *and* the oops sound are executed. If it

was the last cannon, the lose-the-game variable may be set—we can decide later.

1300–1399 Nighttime phase main loop. These lines check for player instructions. If the player calls for boat movement (joystick is pushed), jump to the boat movement routine. If the player doesn't call for any movement, check the timer to see if a tidal change is in order; if so, jump to the tidal change routine. If not, go back to the beginning of the loop.

1400–1499 Boat movement routine. In this routine try to move the boat in whatever direction the player has requested. If there is a collision with anything but a water character, jump to the boat collision handler. If the movement takes the boat off the edge of the playing area, jump to the lost boat routine. Otherwise, execute the movement and return to the nighttime phase main loop. Remember that if the movement is either diagonally or directly with the current, it is tripled, and that if it is diagonally or directly against the current, it is halved. Therefore the normal movement increment should be two scan lines or color clocks.

1500–1599 Boat collision handler. Test to see what you've collided with. If it's a ship, jump to the ship bump routine. If it's the castle, jump to the castle dock routine. If it's the shore, jump to the shore dock routine.

1600–1699 Ship bump routine. The only movement that can take the boat away from contact with the ship is the exact opposite of the joystick movement that caused the collision. The only other way to release the boat is to attach a mine. These lines cause a low humming noise and constantly check to see if the player wants to attach a mine or back out of the way. If the player backs away, execute the backing movement and return to the nighttime main loop. If the player attaches a mine, check to see if there are any mines. If so, turn on the mine set variable for that ship: MS (*shipnumber*)=1. If there are no mines left, ignore the request. Once a mine is set, the player is free to move away and the humming stops.

1700–1799 Shore dock routine. If the boat has no mines, its mine supply is renewed. If it has any mines at all, the boat is loaded with cannonballs—the boat-loaded variable is set to 1. The swishing sound is executed and the boat is pushed away from shore automatically. We return to the nighttime main loop.

1800–1899 Castle dock routine. If the boat-loaded variable is set, the ammunition variable is increased and the blip sound is executed. The number of cannonballs added depends on the level. Whether the boat was loaded or not, the boat is pushed away from the castle and the swishing sound is executed.

1900–1999 Sunk boat routine. The boat sinks under the water and disappears; either we jump to the daytime phase or we set the player-loses variable—we can decide that later.

2000–2099 Sound routines. These include the glub-glub, explosion, oops, blip, and swishing sounds.

2100–2199 Player loses routine. The player has lost a "life." Check to see how many lives remain and either go to the closing routine or next life routine.

2200–2299 Player wins routine. The player has completed the level (sunk all enemy ships or inflicted intolerable damage on the fleet). Add 1 to the number of lives left, do a score change, raise the level by 1, and do the next life routine. (If we want, we can also make the enemy ships sail away.)

2300–2399 Closing routine. PRINT the final score and any final messages. Update the high score and PRINT it; ask the player if he wants to play again. If so, reset variables to 0 (except high score) and start the program near the beginning. If not, restore video memory to its normal configuration and END.

2400–2499 Next life routine. These lines duplicate many of the functions of lines 0–99, except that some initialization routines don't need to be repeated—like defining the character set and setting up video memory. At the end of this routine we jump to the beginning of the daytime loop.

2500–2599 Video memory. Executed only at the beginning of play, this routine sets up the character sets, sprite shapes, and screen memory.

2600–2699 Screen setup. The original castle is set up as one or two strings to be PRINTed all at once. Likewise, the two shores are set up in strings.

2700–2799 Fleet setup. The enemy ships are assigned their locations—how many there are depends on the level. If we want to, we can animate their arrival.

2800–2899 Player setup. The player's cannons and boat are put on the screen.

2900–2999 Initialize variables. The lines from 2900 to 2929 set the values of variables that are only set at the beginning of play. The lines from 2930 to 2959 set the values of variables that are reset when the game is restarted. The lines from 2960 to 2999 set the values of variables that are reset at the beginning of a life.

3000–? DATA statements. These lines contain the character shapes and sprite shapes used in the setup routines.

This outline of subroutines is still pretty rough—a lot of refinements would be needed during the actual programming. However, by laying out this kind of plan in advance, you would know where to find the major routines and have a good reminder of what routines you need to write. Some of these routines would use only a few of the 100 possible line numbers. Others might be much more tightly packed within their hundred-line range. Still, you'll always know that routines start on even-hundred line numbers, which makes a search much easier. Want to see all the lines in the sunk boat routine? Check your plan, find that the routine starts at line 1900, and you simply type LIST 1900–1999. This subroutine outline may not sound like much, but it does give you a great deal of control over the programming process.

Naming the Variables

Many of your routines will need to use the same variable to get information. The timer variable may be used by many different routines; the directional vector that is used in the cannon aiming routine will also be used in the cannon firing routine to decide what direction the projectile should travel. The best way to keep track of the variables is to name them and write them down so that you can refer to them again and again as you program. You also need to decide which variables are arrays, so that you can use other variables as indices into a table—I've sometimes had one integer array variable indexing another, which was used as the index in a string array—all of them indexed by a loop control variable and a directional variable.

If this sounds hopelessly complex, don't worry. Reading about it is complex. Sitting down and doing it is much easier.

Of course, doing it so that the program actually *works*—that's hard, which is why so much of a programmer's time is spent figuring out why it doesn't work and fixing the bugs.

Getting Back to the Real World

In the real world, you'll never plan *everything* in your game in advance. You may *think* you've got it planned, but once your mind is working, it doesn't stop. You'll have more ideas in the middle of programming, or you'll run into a programming hurdle you don't know how to get over. You'll start to improvise, and the plan will begin to bear less and less resemblance to what you end up with.

Don't worry, though. It'll be much better than the plan. Yet without the plan, many of the eventual improvements would not have been possible.

The point of planning ahead is not to build a dam to block your creativity from flowing. Instead, you dig a channel into which your creativity flows, like a stream. When that river gets too strong, it jumps the bank or creates bayous along the edges. But the channel helps it flow strong and deep. When you have a clear idea of where you're going, you have a much better chance of getting there quickly.

In fact, so much creation goes on after the initial planning is done that I'd be willing to bet that every person who reads this chapter could set out to program a game that followed my plan perfectly—and no two games would look or play alike. In fact, I'd expect that many games would look and play so differently that an uninformed observer would never suspect that they began from the same plan.

Your own games may be much simpler than this one, or much more complex. The most important thing is to design a game that you would love to play—if your heart isn't in it, your game won't be any fun. You can't fool the players.

And when you're through planning and programming your game, you'll know the secret: Game designing is the best videogame of all. No matter how brilliant your game is, no one will ever have as much excitement and frustration and satisfaction and fun playing it as you had in creating it.

2
Text Adventure Games

Puzzles, Palaces, and Pilgrims

Writing Text Adventures for the Commodore 64

Gary McGath

Programming text adventure games, those popular interactive games where you communicate to the computer through words, is an art in itself. It's not quite the same as creating an arcade-style videogame. Here, Gary McGath, who has written a book on just this subject, explains some of the basics of writing text adventures.

A text adventure is an interactive computer game in which the player assumes the role of a character in a story. As the player, you control the character's actions by typing in commands, and the computer responds with a text description of what your character experiences.

The world of most text adventures is composed of a number of *rooms,* or locations. Your character moves from place to place, or from room to room, where objects or other characters may or may not be found. Sometimes these objects and characters aid you, other times they're dangerous. By using the appropriate commands, you can pick up, examine, and even use these objects and characters.

While professionally written adventure programs often comprehend complicated sentences as commands, many adventures get by with simple two-word commands. The vocabulary of even the best of them is quite limited, and they have to indicate to you whether they "understand" any particular command.

The following dialogue is typical of a text adventure. (Your commands are printed in boldface and the computer's responses in normal text.)

You are in a small room lined with shelves. There are doors to the north and west.
There is a gem on the shelf.

Take gem

Your hand is stopped by an invisible shield around the gem.

Examine shield

I don't know the word *shield.*

North

You are in a north-south hallway.

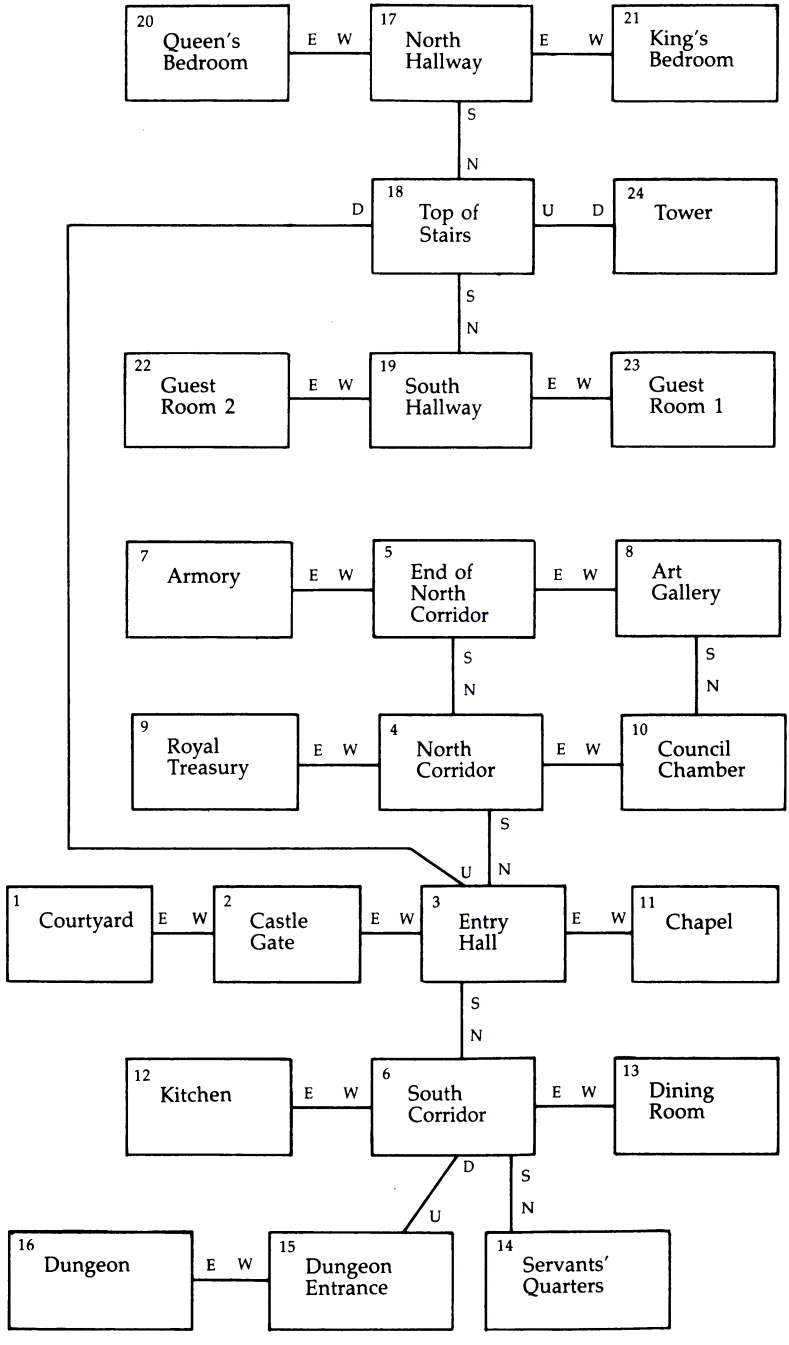
Writing a text adventure offers you a chance to exercise your imagination and set up logical puzzles for your friends. It requires no special screen formatting or sound effects, and the program is doing nothing between moves; these facts make text adventure programs easy to debug. And once you've written your first adventure, you can do more of them just by changing the rooms and puzzles in your old program.

Mapmaking

While the first steps in designing a text adventure are to create the *story line* (what will happen) and the *milieu* (where things will happen), we'll assume you've already done that. (To get an idea of how to create the story line and milieu, read "Thinking It Through: How to Plan a Videogame" elsewhere in this book.) We'll be concerned mainly with the actual programming techniques you'll use, as well as some of the more practical design processes in this article.

Once you've decided on what your world is, and what will happen in it, you need to design a map of the rooms. (Remember that they don't have to actually be *rooms;* we're using that as a generic term. They can be places on a road, paths in a forest, or even corners of a field.) Draw a map with a box for each room and connecting lines labeled with the directions that lead from one room to another (north or south, for instance). Give each room a number and a short description. The room in which the character starts should be room 1. Figure 1 shows the map of an example text adventure game.

Figure 1. The Adventure's Map



Objects, Verbs, and Consequences

In this planning stage, you also need to make several other decisions. Choose the objects that will be in the adventure and where each will be initially located. Some objects might not be in any room at all until the player does something to make them appear. You should also assign numbers to the objects.

Your program also needs a list of the verbs that will be accepted as commands. Certain verbs (or words that function as verbs) are almost mandatory, such as NORTH, SOUTH, EAST, WEST, TAKE, DROP, EXAMINE, LOOK, INVENTORY, and QUIT. Other verbs that might be helpful include ENTER, CLIMB, SHAKE, MOVE, TURN, FIGHT, OPEN, EAT, DRINK, CLOSE, and READ. Abbreviations, such as I for INVENTORY and N for NORTH, are easier for the player to remember and use. Allowing the use of equivalent alternatives, like GET and TAKE, which should mean the same thing, can reduce player frustration. Remember, the difficulties in an adventure should come from the logical puzzles, *not* from figuring out how to talk to the program.

What consequences do specific actions have? Will opening a box reveal a gem, or will it set off an explosion? Will pressing a switch start a machine? Will magic words transport the character into a new room? Consequences could include appearances and disappearances, changes in the character's abilities, alteration of the paths between rooms, and transportation from one location to another.

Some actions may have special consequences only under restricted circumstances. A special tool may be needed, such as a crowbar to open a crate. If this tool isn't in the character's inventory, the action won't have the desired effect and might even backfire.

Things may happen independently of the player's actions as well. A troll might be wandering around the adventure's world. Or the character's lamp might go out after a certain number of moves.

When you've considered all these things and made your choices, you know what you want the adventure to do. Only now should you worry about the details of the program. As you discover what's easy to program and what isn't, you might change your mind about what features to include. But just as when you program any game, you should start with an

overall plan. It will save you countless hours of wasted time later on.

Assigning Variables

Now you're ready to actually begin programming your text adventure game. We'll go through the process step by step, outlining and illustrating exactly how to do it.

The first step is to to assign variables to the important parameters of the adventure. It's easier to remember what these variables mean than it is recall a number; using these varables also makes it simple to alter the program if you later decide to change the parameters. One of the first statements of the program, even before the DIM statements, should look something like:

```
10 NR=21:NV=14:NO=16:NI=10:ND=6
```

NR is the number of rooms, NV the number of verbs, NO the number of objects, NI the number of items, and ND the number of different directions the character can move in. (Note: An *object* is any word that can be used as the second word of a command, whether it corresponds to a physical object or not. An *item* is an object that is located in a room; it usually designates a physical object.)

Adventure Arrays

Next you need to translate the layout of your adventure into a set of data structures. Let's look at each of the required structures and the purpose it serves.

Access array. This is the translation of your map into terms the computer can understand. It's defined by the statement:

DIM AC(NR,ND)

To use the access array (AC), the directions in which the character moves must be translated into numbers. Let's assume the following translation:

North	= 1	South	= 2
East	= 3	West	= 4
Up	= 5	Down	= 6

The value of AC(NR,ND) specifies which room is reached by going in direction ND from room NR. If this value is 0, it means the character can't go that way from that room.

Room description array. This array is defined by:

DIM RD$(NR)

Each of its entries is a string that gives the description of the room— for example, "You are standing on a wide bridge."

Room flag array. Flags are indicators of whether a condition is true or false. A flag takes only 1 bit, so you can have up to 16 different flags in the room flag array. The array is defined by:

DIM RF(NR)

The different flags are defined as powers of two: 1 might indicate that the room is too cold, 2 that magic works, and 4 that water is present. The value of RF(R) for room R consists of the logical OR of all the flag values that are true for that room. If a room is cold and allows magic but doesn't have any water, then its entry in the array would be a 3 (1 OR 2).

Verb array. This is an array of the possible first words of commands, defined by:

DIM VB$(NV)

You should decide how many letters in a word are going to be significant and chop the verbs in this array down to that size. For instance, if two letters are significant, then TAKE must be stored as TA. It's a good idea to limit the number of significant letters, so that two-fingered typists have less work to do. Many simple adventure games designate only two letters as significant.

Object array. This is an array of the possible second words of commands, (objects) defined by:

DIM OB$(NO)

Once again, all words in this array should contain only as many letters as are significant.

Verb token array. This serves to translate verbs into numbers. It is dimensioned by:

DIM VT(NV)

The entries in this array correspond to entries in the verb array. The values stored consist of numbers from one to the *number of distinguishable verbs* in the game. This number is normally smaller than NV, since similar verbs such as GET and TAKE, or NORTH and N, are not distinguishable. If VB$(2)="N" and VB$(3)="NORTH", then VT(2) and VT(3)

will have the same value. This lets the program be indifferent to the word that was actually typed.

Object token array. This array translates the second word of a command into a number. It is defined by:

DIM OT(NO)

Its elements correspond to the object array. However, the elements can be a little trickier than the verb token array's elements. Remember that not all *objects* are *items.* It's convenient to have the object tokens fall into two series. Items, which are objects that have a particular location, can be numbered from 1 to NI. Other objects, including directions and magic words, can be numbered starting with 101. This makes it easy to add new items without disrupting your numbering system.

Item description array. This contains a text description for each item. Its definition is:

DIM ID$(NI)

The text description of an item could be the same as the word in the object array for it, but often is a little more. For instance, the object array might have the word LAMP for an object described in the item description array as "Old oil lamp."

Item location array. This locates each item and is defined by:

DIM IL(NI)

There are three possibilities for where an item is located. It could be in a room, in the character's inventory, or nowhere at all. The third case indicates an item that's been destroyed or one that's not yet available. A positive number in the item location array indicates which room the item is in. A zero says that the character is carrying the item. A negative one specifies that the item isn't to be found.

Item flag array. This is similar to the room flag array in concept, except that it specifies conditions that are true or false of items rather than rooms. It is defined by:

DIM FI(NI)

(It would make sense to call the array IF, but that's a reserved word in BASIC.) Specific bits in the elements of the array are used to indicate such properties as whether the item can be carried or not.

More Variables

Finally you'll need to set a few more variables, such as:

VB Verb token obtained from the last command entered.

OB Object token obtained from the last command. It can be 0 if only one word was typed.

RM Room the character occupies.

NC Number of items the character is carrying.

MI Maximum number of items the character may carry.

NI may never exceed MI.

MC Move counter. This indicates how many moves have occurred since the adventure started. It can serve as a timer for various events.

DF Description request flag. This variable is set to 0 after the current room is described to the player. If a description is required before the next move (because the character went into a new room or decided to LOOK around again), it's set to 1 to get the description displayed. Leaving it at 0 saves having the same description repeated every move.

Specific situations will undoubtedly call for a few more variables, but the arrays and variables listed here will provide the major part of what a simple adventure needs.

The Main Loop

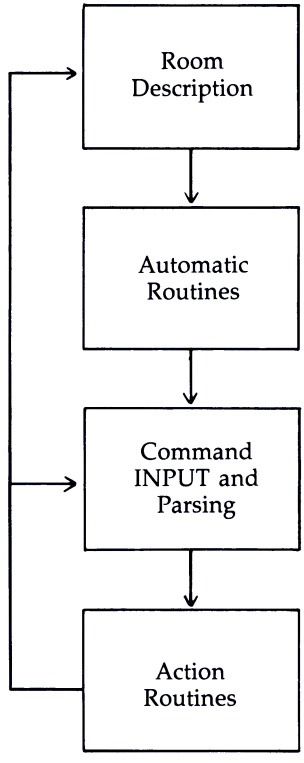
An adventure program consists of two parts: the initialization and the main loop. The initialization section includes dimensioning arrays and setting up data. We've already looked at some of the initialization section of our example adventure. It uses READ and DATA statements to set up all the initial values. Once the initialization is done, however, the main loop takes over. It runs until the game is completed. The overall flow of the main loop would be something similar to that shown in Figure 2.

The major portions of the main loop, as shown in Figure 2, are the room description, the automatic routines, the command INPUT and parsing, and the action routines. Let's consider how to program each of these in turn.

Room Description

Whenever the character moves into a new room, the surroundings change. If the player asks to LOOK at the room again, the room description routine provides this information.

Figure 2. The Main Loop



There are two things to be described: the room itself and whatever items it contains.

This routine isn't long and could look like this:

```
400 IF DF=0 THEN 600
410 PRINT RD$(RM)
420 F=0
430 FOR I=1 TO NI
440 IF IL(I)<>RM THEN 490
450 IF F=0 THEN PRINT "YOU SEE:":F=1
460 PRINT ID$(I)
490 NEXT I
```

The description request flag in line 400 determines whether this section of the program is executed or skipped over. Remember that 0 indicates the latter. If it is 0, then, this entire

routine is bypassed. If it *is* executed, describing the room consists simply of printing the appropriate element of the room descripion array. That's line 410. Then in line 430, a FOR-NEXT loop executes, which goes through each item in the item location array. For each item that's located in the current room (F=0), it prints the corresponding element of the item description array (done in line 450 and 460). This way the player will see what each room contains.

Automatic Routines

The next section of the main loop takes care of events that aren't directly caused by the player's commands. We can call these routines *automatic*, for they happen independently of what's typed in. An adventure can be written without any automatic routines, but having even a few things outside the player's control gives a much greater sense of realism and excitement.

Automatic routines can be controlled by the move counter, random numbers, or a combination of the two. The commands the player gives can have an effect as well. A passage may close four turns after the character enters a room, or a wraith may start stalking the character only after he's touched a crypt. Extra variables can be used to indicate the move on which something will happen. In the following example routine, MM is a variable indicating the move in which a wall collapses, opening a new passage between rooms 8 and 9.

```
700 MC=MC+1
710 IF MC<>MM THEN 800
720 AC(8,3) = 9: AC(9,4) = 8
730 IF RM=8 THEN PRINT "THE EASTERN";
740 IF RM=9 THEN PRINT "THE WESTERN";
750 IF RM=8 OR RM=9 THEN PRINT " WALL COLLAPSES, O
    PENING A NEW PASSAGE."
```

MC is the move counter, our timer, so to speak. Each time through the main loop, it's incremented by 1 in line 700. Assuming we earlier set MM to the desired turn number (say 8), then this automatic routine would not be executed until MC equals MM—in other words, on turn 8. Line 710 insures this. Line 720 actually creates the opening between the rooms. The message then displays, specifying which wall has crumbled. If the character is in room 8, for instance, the eastern

wall has fallen, and the character can now move in that direction.

The position of automatic routines in the program is important. Usually they should come *after* the room description, so that the player finds out where his or her character is before being told what happens. Some automatic routines, however, are better placed after the player has completed the move. This conveys the feeling that what happened immediately followed the move. For instance, if a flock of bats carries the character out of a room every time he tries to enter, the player may not even see the room until it's discovered how to get the bats out.

Command INPUT and Parsing

At this point the program stops talking to the player; instead, it's the player's turn to communicate with the program. To do this, the program must accept a command and *parse* it. To parse a command simply means to break it up into its components and identify their relationships—an easy job when it consists of just two words.

Here's the first section of an INPUT and parsing routine:

```
1000 INPUT C$
1010 L=LEN(C$):IF L=0 THEN 1000
1020 C1$="":C2$="":C2=0:X=0
1030 FOR I=1 TO L
1040 A$=MID$(C$,I,1)
1050 IF A$<>" " THEN 1080
1060 IF C2$<>"" THEN 1200
1070 X=1:GOTO 1090
1080 IF X=0 THEN C1$=C1$+A$:GOTO 1090
1085 C2$=C2$+A$
1090 NEXT I
```

The program receives a command through the INPUT statement. As the player enters words, a string is created. Then the program separates the two words by looking for one or more spaces between them. (It's best that it be tolerant of more than one space between words, as well as spaces after the command. INPUT automatically strips leading spaces, so they don't pose a problem.) The above program section receives the player's INPUT (line 1000) and creates two strings, C1$ and C2$ (lines 1080 and 1085). Spaces between words are also checked for in line 1050.

The following lines continue the routine:

```
1200 C1$=LEFT$(C1$,6): C2$=LEFT$(C2$,6)
1210 FOR I=1 TO NV
1220 IF VB$(I)=C1$ THEN VB=VT(I):GOTO 1250
1230 NEXT I
1240 PRINT "I DON'T KNOW THE VERB ";C1$:GOTO 1000
1250 IF C2$="" THEN OB=0:GOTO 1400
1255 FOR I=1 TO NO
1260 IF OB$(I)=C2$ THEN OB=OT(I):GOTO 1400
1270 NEXT I
1280 PRINT "I DON'T KNOW THE OBJECT ";C2$:GOTO 100
     0
```

The two strings, C1$ and C2$, are the first and second words of the command. The next step is to translate these strings into the verb token and the object token. This means looking them up in the verb array and object array and getting the corresponding elements of the verb token array and object token array. Lines 1220 and 1240 in the section of the routine above do this for the verb and object respectively. Note the checks and messages displayed if the verb and/or object do not exist in the appropriate array.

The two strings must be truncated to the number of significant characters in order to match the strings in the arrays. Line 1200 assumes truncation to 6 characters.

In the case of a one-word command, C2$ will be the empty string, so the object token will be set to 0 (line 1250).

Action Routines

Once the program has the command in the form of the verb token and the object token, it's ready to determine what those commands will do. We can call the parts of the program that do this the *action routines*. This section will be the largest portion of the program; however, since it consists of a lot of small pieces, it isn't very difficult to write.

Before figuring out what a specific verb does, the program should do some general checking to determine whether the object is reasonable. If the object is an item, it has to be either in the room or in the character's inventory. If it's somewhere else, the character can't do anything with it. If the object isn't an item, then only a few verbs will work with it, so the program should make sure that the verb is an appropriate one. NORTH, for example, isn't something the character can TAKE, EAT, or OPEN. Only GO makes sense.

The following routine assumes that the direction object tokens (NORTH, UP, etc.) are numbers 101 to 106, that GO is verb 10, that SHAZAM is object 107, and that SAY is verb 12.

(In a language that was more generous with names than BASIC, we could assign a variable name to each verb. Trying to think of a two-letter name for each verb that would mean anything, though, is a hopeless exercise. So at this point we resign ourselves to using numbers.)

```
1400 IF OB<100 THEN 1600
1405 REM IT'S NOT AN ITEM
1410 IF OB<=106 AND VB<>10 THEN 8000
1420 IF OB=107 AND VB<>12 THEN 8000
1430 GOTO 2000
1599 REM IT IS AN ITEM
1600 IF IL(OB)<>RM AND IL(OB)<>0 THEN PRINT "IT IS
     N'T HERE.":GOTO 1000
8000 PRINT "THAT'S SILLY!":GOTO 1000
```

Line 1400 of the routine checks to see if it's an item (with an object token less than 100). If it is, the program jumps to line 1600, where it's determined whether the item is in the room or in the character's inventory. If neither, then the message IT ISN'T HERE displays. The program chides the player with THAT'S SILLY! if a direction (NORTH, UP, etc.) is requested and GO isn't used with it. The player will also see the message if something like SAY (VB=12) SHAZAM (OB=107) is typed in.

Notice that if the command is rejected, the program goes back to the command INPUT (through the GOTO 1000 statements in lines 1600 and 8000), rather than letting anything happen automatically.

If these checks turn up no problems, the program falls through to the action routine for the specific verb. The tool used is the GOTO statement found in line 1430 above. It sends the program to line 2000, shown below:

```
2000 ON VB GOTO 3000,3100,3200,3300,3400,3500,3600
     ,3700
2010 ON VB-8 GOTO 3800,3900,4000,4100,4200,4300,44
     00,4500
```

Several of these statements will usually be necessary because of the 64's 80-character line limitation. Remember that an ON statement will simply fall through to the next statement if the

variable is out of range. Thus, if the variable is 9, it falls through line 2000 to line 2010, where it would access the first line listed, 3800 (9−8=1). Using this technique, we can call up to 16 different verb routines in the above example.

Each of the line numbers in lines 2000 and 2010 is the start of the action routine for a particular verb.

Certain verbs will be standard in most adventures, so they can be discussed in some detail here. Others will have effects that are peculiar to the situation. They're the ones that make your adventure unique. Once you've seen how the standard verbs work, though, you shouldn't have much trouble adding your own special ones.

Directional verbs and GO. There are two ways a player might specify moving in a direction. Either a simple direction (for instance, EAST or just E), or GO and a direction (GO EAST) could be entered. It isn't much trouble to include both. A common area of the program can be used to handle all directional movement, using a direction variable that the specific commands set before accessing the actual movement.

For a one-word command, the direction acts as the verb. In this case, it just sets the direction variable and goes to the common routine. The line below illustrates the one-word command NORTH.

```
3100 D=1:GOTO 3620
```

You'll recall that earlier we decided to use 1 as the directional number for NORTH. All that's done in the above line is to set D (the directional variable) to 1 and then GOTO a line that checks to see if that direction leads anywhere. (More on that in a bit.)

However, the GO command has to translate its object into a direction before going to the common routine. It's easy to do this if the direction objects are numbered appropriately, so that subtracting a number from the object token gives the right index into the access array. Take a look at the following lines:

```
3700 IF OB<=100 OR OB>106 THEN 8000
3710 D=OB-100:GOTO 3620
```

Notice that if the object (OB) is *not* a direction (checked for in line 3700), then the program jumps to line 8000, where the message THAT'S SILLY! is printed. The direction variable D is set in line 3710. If OB equals 101, for instance, signifying that

the direction is NORTH, then D equals one. The program then moves to line 3620.

The common routine uses the access array to determine where the move will take the character. The next segment is this common routine used by both one- and two-word commands.

```
3620 IF AC(RM,D)=0 THEN PRINT "YOU CAN'T GO THAT W
     AY.":GOTO 400
3630 RM=AC(RM,D):DF=1:GOTO 400
```

A value of 0, as mentioned before, means that a given direction doesn't lead anywhere. If the command does take the character somewhere, the description request flag is also set, so that the player can see the new room. Both of the lines above take the program back to the routine that describes the room.

TAKE. This command transfers, or attempts to transfer, an item from the current room to the character's inventory. The program has to determine whether the item can be picked up and whether it can be carried. The character might already be carrying as much as allowable. Taking an item might also have side effects, like making another item visible or setting off a trap. The program doesn't have to check whether the object is in the room, since that has already been determined. However, it *does* have to check whether the character is already carrying the item. Take a look at the lines below to see how that can be programmed:

```
4200 IF (FI(OB) AND CF)=0 THEN PRINT "YOU CAN'T PI
     CK THAT UP.":GOTO 400
4210 IF IL(OB)=0 THEN PRINT "YOU ALREADY HAVE IT!"
     :GOTO 400
4220 IF IC=5 THEN PRINT "YOU'RE CARRYING TOO MUCH
     {SPACE}ALREADY.":GOTO 400
4230 IL(OB)=0:IC=IC+1:PRINT "TAKEN."
4240 REM SIDE EFFECTS GO HERE
4290 GOTO 400
```

This assumes that flag CF (in line 4200) in the item flag array indicates whether or not an item can be taken. If your character already has the item, then line 4210 prints a message to that effect. Note that a limit of 5 items is set in line 4220. If IC (the variable keeping track of the numbers of items carried) equals 5, the character can't take anything else. Line 4230 actually TAKEs the item by placing it in the character's

inventory (IL(OB)=0), increments the number of items held, and prints a message that the TAKE was successful.

DROP. The reverse of TAKE, it's even simpler, since an item that is being carried can normally be dropped.

```
4300 IF OB=0 THEN PRINT "DROP WHAT?":GOTO 1000
4310 IF IL(OB)<>0 THEN PRINT "YOU DON'T HAVE IT!":
     GOTO 400
4320 IL(OB)=RM:PRINT "DROPPED."
4330 IC=IC-1
4390 GOTO 400
```

The only question is if the item is in the character's inventory, which is checked in line 4310. The object is transferred to the room (line 4320) and the inventory count is decremented (line 4330). Again, side effects are possible.

INVENTORY. All this command does is list the items the character is carrying. This involves going through all the items and listing the ones that have a location of 0.

```
4400 PRINT "YOU ARE CARRYING:"
4410 FOR I=1 TO NI
4420 IF IL(I)=0 THEN PRINT ID$(I)
4430 NEXT I
4440 IF IC=0 THEN PRINT "NOTHING."
4450 GOTO 400
```

Line 4420 PRINTs the items the character is carrying. If IC (the number of items carried) is 0, a message indicating that the character holds nothing is displayed.

LOOK. This is one of the simplest commands; it just sets the description request flag with a line such as:

```
4500 DF=1:GOTO 400
```

QUIT. Even simpler, except that it's nice to make sure the player really means it:

```
4600 PRINT "DO YOU REALLY WANT TO QUIT";
4610 INPUT Y$
4620 IF LEFT$(Y$,1)<>"Y" THEN 1000
4630 END
```

More Unusual Commands

Other verbs vary from one adventure game to another. EXAMINE can give you additional information about items. FEEL, SMELL, and TOUCH might serve a similar purpose.

The process of examination might also cause other, previously hidden, items to appear. OPEN could be another way to reveal a hidden item. Words like CUT and BURN might have interesting effects on items, but unless an appropriate tool is in the character's inventory, these commands would simply return a message like "You can't do that."

Having a few commands that do nothing but return a standard response is useful, just because it adds to the number of commands that get an interesting answer without adding much to the programming effort. For instance, the verb BREAK with any object might get the response "Vandalism won't help your situation." This will also leave the player wondering whether there's some object that *could* be broken for a useful result.

Commands like CLIMB or ENTER might work on certain objects to provide a way of getting from one room to another, in addition to the directional commands. (Don't use GO for this, please. In spite of what some adventure game programmers think, you don't *go a door.*)

Other commands might also surprise the player by transporting the character from one place to another. For instance, taking an item might cause a trap door to open, dropping the character into the room below. Magic words can serve this purpose. A magic word may be restricted in its use to a certain room, so it provides passage only from that room to another.

What Goes Into It?

The mechanics of writing an adventure program are only part of the job, just as grammar and spelling are only part of what goes into writing. The other part is what you actually have to say. Creating the content of an adventure can't be reduced to a cookbook approach. Still, some general guidelines are possible.

Quests and hunts. There are two basic types of adventure: the quest and the treasure hunt. In a quest adventure, you're given a particular goal to achieve, such as solving a mystery or obtaining a single treasure. In a treasure hunt, you're trying to find as many treasures as possible to get a high score.

The quest adventure is an all-or-nothing proposition. The program can give you a score to indicate how close you've

come to success, but you probably won't be satisfied until you solve it. The treasure hunt offers more satisfaction to the beginning adventurer, since if even a few treasures are found, there's a sense of accomplishment. If a quest is like climbing a mountain, a treasure hunt can be compared to hiking across a series of low hills. Each one has its own kind of satisfaction.

Make the pieces fit. In either case, all the pieces should fit together. This is more obvious for a quest—each step is part of a developing story. Even in a treasure hunt, though, everything should be set against a common background and story line. If it's set in a world of Greek mythology, Wotan and Brunhilde shouldn't appear without good reason. If you've chosen a science-fiction setting, it shouldn't have magical elements that don't fit. Humorous events can certainly liven up an adventure, but they shouldn't be jarringly out of place.

The puzzles should be interrelated. Otherwise, what you end up with is a series of small puzzles rather than one complete adventure. Solving one puzzle should provide a tool that's needed for solving the next one. The various items required should be scattered around so that the character has to go back and forth among the rooms, rather than having everything too neatly at hand.

Don't cheat. The puzzles should always be logical. The solution should make sense, at least once the player has stumbled upon it. A puzzle that reduces the player to trying actions at random has failed. If the way to summon a genie in your adventure is to kiss a coconut, be sure to provide some clue that will suggest that action! If you don't, you'll have a hard time getting people to play your second adventure.

Traps should not be sprung unexpectedly. It should be possible for the player to get a hint of danger ahead before walking into it, perhaps by requiring the player to examine things carefully. This doesn't mean that everything should be so easy that a player can solve it the first time. It means that at the end of the puzzle or game, the player sees the program was "playing fair." One adventure game I've played, for example, requires the character to crawl through a passage to survive, yet there was no indication that the passage was dangerous. This forces the player to rely on knowledge gained in a "previous life," something not as realistic as many players would like.

Just as when you create any game, the art of text adventure writing is much like the art of storytelling. To keep the player interested, interesting things have to happen. One event should follow reasonably from another and lead to a climax. Because it is a form of storytelling, the text adventure offers you, the author, a chance to express yourself, something not often found in other forms of videogames. When you write an adventure, you're doing more than creating a game; you're creating a world.

Time Capsule

David Florance

There are almost as many ways to program text adventure games as there are games themselves. "Time Capsule," although a relatively simple adventure game, tests your logic and intuition as you try to escape a dangerous future. Best of all, the programmer has included detailed notes about how he created the game.

Captured by an unknown enemy and transported into the future, you struggle to escape the labyrinthlike prison so that you can return to your own time and sound the warning. That's the story line of "Time Capsule." But there's more, for although dangers and rewards are built into the game, you create the plot through your character's actions. Do you try to bluff your way past the guards? How do you know what area of the prison is safe, and who will help you? Your decisions create the game. That's why playing text adventure games is so much fun. No two games seem to end up the same.

Typing In the World

Entering Time Capsule is easy when you use "The Automatic Proofreader," found in Appendix C. Before you begin typing in this adventure, make sure you read the appendix and have a copy of the Proofreader program on tape or disk. Using the Proofreader almost guarantees that you'll have a working copy of the game the first time you type it in. It's almost impossible, if you follow the instructions in Appendix C, to make a typing error.

Once you've typed in and saved the game, load and run it to play. You'll see a title screen, some simple instructions, and then your character appears in a room. The rest is up to you. You know how you would like the adventure to end—with you safely back in your own time so that you can warn the world about these dangerous enemies and their time capsule—but how you get to that ending is unknown. The game isn't difficult. Not really. But you have to think things through. Use logic and common sense. Do you take the shoes or not? What would be the purpose? Try it and find out.

I've purposefully not included a list of the adventure's

vocabulary. Part of the fun in playing a game like this is discovering how to communicate with the computer. The game doesn't require an esoteric vocabulary; just think a bit before you type something in, especially if your first request was ignored. Sometimes you'll miss the obvious.

Preplanning

It's important that you have a clear view of what your game will be about. Getting your ideas on paper is not a bad way to begin. Ask yourself some questions, such as, "Do I want characters in my game?", "What will be the solution?", and "Where does the adventure take place?" Once you've got the design to your liking, you'll be ready to program. Don't be afraid to let the game develop as you write it, though. As you get deeper into the actual programming, you'll find certain things work better than you expected, while other things you thought essential are not practical. Be ready to make adjustments.

Setting up the program's variables is also important. If you want to take advantage of the 64's graphics capabilities (if for nothing else than to liven up the display), this is the time to set variables for screen color, border color, and memory locations. I like to use variables that are easy to recognize and remember, such as CS for Color of Screen and CB for Color of Border. Logic plays the most important role in any programming, so use variables you're comfortable with and that make sense to you. You'll be glad you did later on. The most likely and economical place for your variables is at the beginning. There they will not interfere with the rest of the program. Another good reason for setting your variables at the top of your program is that all values will be set before execution of the main part of your program. Then, as you want to set parameters, you can call the variable instead of typing in the numbers each time. This is just one thing that makes BASIC so easy to use.

Another practice I've found useful in writing text adventures is having a structured numbering pattern. Leave yourself room to come back to a part of the program and insert statements. A good way to do this is to begin numbering by 10s, even by 20s, to provide space for anything you might later want to add. The first statement should be a low number, say 10. After you've set your variables you may want to skip down

to line number 100 or so. This will remind you to keep your variable initialization at the beginning. If you make that statement number low, you will be less likely to put statements in front of it. Skipping down also provides a visual separation of your initialization procedures from the heart of the program.

IF-THEN?

There are many ways to program text adventures. What I've done in writing Time Capsule is an example of a simpler way to program, one that doesn't use a lot of complicated formulas. In order to keep from being too technical, and to show how BASIC can be used to construct stimulating adventures, I didn't shy away from using the IF-THEN statement. It's less logical to me to use them in scattered places throughout the program than to put most of the important ones together in a group. This is, of course, a personal preference. You may want to try other ways. I've shown in the program two techniques that allow you to get around using a long list of IF-THEN statements, and you'll undoubtedly find others of your own.

One way is using a FOR-NEXT loop to count the number of times you want the IF-THEN executed. This involves using variables to let the 64 know what it's counting and what to do on certain conditions. For instance, lines 92–120 in Time Capsule could have been translated into a list of IF-THEN statements that would look like this:

```
100 IFMID$(D$,3,1)=CHR$(32)THEND2$=MID$(D$,4,LEN(D
    $))
110 IFMID$(D$,4,1)=CHR$(32)THEND2$=MID$(D$,5,LEN(D
    $))
120 IFMID$(D$,5,1)=CHR$(32)THEND2$=MID$(D$,6,LEN(D
    $))
130 IFMID$(D$,6,1)=CHR$(32)THEND2$=MID$(D$,7,LEN(D
    $))
140 IFMID$(D$,7,1)=CHR$(32)THEND2$=MID$(D$,8,LEN(D
    $))
```

As you can see, the way I've done it in the program is much shorter and does the same thing.

Secondly, use of the ON-GOTO can also replace a long list of IF-THEN statements. For instance, line 53 in Time Capsule replaced (from an earlier version of the game) this list of IF-THENs:

```
53 IF PZ=1 THEN 8Ø1Ø
54 IF PZ=2 THEN 851Ø
55 IF PZ=3 THEN 871Ø
56 IF PZ=4 THEN 891Ø
57 IF PZ=5 THEN 9Ø2Ø
58 IF PZ=6 THEN 922Ø
59 IF PZ=7 THEN 942Ø
6Ø IF PZ=7 THEN 962Ø
61 IF PZ=8 THEN 984Ø
```

Ultimately this uses half the memory of a long list of IF-THENs. In general, try to find ways to cut down on IF-THEN statements simply as a matter of economy. Don't be hesitant to use them if you can't find a better way, because sometimes there *is* no better way, or the alternative would be even more involved.

Parsing

Perhaps the key to writing text adventures in BASIC is finding a way to have the computer read, or *parse*, the player's instructions. On the Commodore 64, you can create an almost limitless vocabulary. Having the computer acknowledge a certain number of characters in the player's input is the method I've used. First, the program reads the entire player response. Then it separates the response into two parts: the first command, or keyword, and whatever else remains. Having the response divided into two parts enables the 64 to determine how it will act on that response. There are 15 keywords in the game. If one of these is not used as the first word of the response, the 64 lets the player know that this is unrecognized. In other words, unless the player begins the response with a keyword recognized by the program, the program will not "understand" it. After the program recognizes a response as a keyword, it identifies the keyword. Then the program shifts to the routine where that particular keyword's actions are stored. Here the program uses its knowledge of the rest of the response to react to the player's instructions. If it doesn't recognize the rest of the instruction, it lets the player know. If the instruction is recognized, the program executes the entire command. This simulates "talking" to your 64. It will seem as though the 64 is alive!

Determining how the program responds to certain words is entirely up to you. However, I've found that players enjoy a

text adventure more if the responses are simple. Long phrasing, such as "Walk two steps to the left and pick up the can," makes for lots of typing. Try to design your program so that it recognizes and acknowledges short phrases, such as "Get can" or "East." This makes it easy for computer novices to play your game. Testing the program, as you'll undoubtedly do many times, becomes easier, too.

Most commercial text adventures allow the player to pick up and use items along the way. I've included this feature in Time Capsule. Although strings play an important role in this facet of programming text adventures, they are by no means the only way to handle this matter. READ and DATA statements would be others. Since you'll want to alter the selection of items according to what the player has done, I think strings are as good a way to do this as any, and are not too complicated to use. For instance, Time Capsule defines A$ as the items used in the display you see in the first room.

```
5000 A$="WINDOW,SEAT,SHOES,CAN"
```

All the items you'll see in the game are contained in strings. The rest of these items are defined in lines 5010–5045 of the program.

In Time Capsule, the player can pick up certain items. As the player does this, the item is removed from the string. This is a simple matter of redefining the strings as selections are made. Be careful not to redefine a string before or after the player has made a choice, however.

Playability

To keep your program readable, you may want to designate nomenclature for the different places the player explores. I chose to separate the puzzles, and it seems to work well. Designations of PUZZLE 1, PUZZLE 2, and so on help remind you where the player is in the program. You'll find this helpful when editing the program as well. The 64 can then narrow down the actions it may take rather than review all its options before making a decision. Let's say a player types in a response that is not intended to work in the present situation, but is intended to work at some other point in the game. If the program knows which puzzle the player is on, it can determine not only the entire situation, but also the immediate

situation. This will prevent the program from allowing a player to jump ahead, whether by intention or accident. Changing screen and border colors for each puzzle helps in solving the game and also enhances the program to keep players interested.

Pacing your game so that it's not too slow is vital. One could easily get bored with puzzles that are too difficult, so it's a good idea to throw in some easier ones periodically. On the other hand, a text adventure that can be solved in one sitting is not much fun either. Use your own judgment. Remember, *you* know the solution, the players don't.

Adventures of Your Own

If you're familiar with BASIC, you shouldn't have any trouble creating your own text adventure game. Some of the simplest programming tools and techniques can form the heart of such a game. You don't have to know machine language or even how to program complicated graphics. These may be useful when you're writing an arcade-style game, but text adventures are different. Programming is almost secondary to the story you want to tell. Use your imagination. The rest is easy.

Time Capsule

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
10 CB=53280:CS=53281:SC=1024:ES=1984:CC=55296:EC=5
   6256:H=41:S=40:V=39:PZ=1                    :rem 218
15 CV=54272:Y1=1:PRINT"{CLR}{YEL}":POKECB,2:POKECS
   ,2:GOTO5000                                 :rem 143
17 IFSKTHEN30                                   :rem 39
20 GOSUB 13000:FORY=205TO1STEP-5:PRINT"{3 DOWN}
   {10 RIGHT}{RVS}TIME CAPSULE{OFF}"1984+Y1;
                                               :rem 156
22 Y1=Y1+5:POKE CV+24,Y/15:NEXT:FORTY=1TO1000:NEXT
                                               :rem 84
23 IF AC=20THEN9610                            :rem 22
25 IF AC=23THEN60020                           :rem 67
27 INPUT"{CLR}{4 DOWN}{4 RIGHT}INSTRUCTIONS(Y/N)";
   IS$                                         :rem 95
28 IFIS$="Y"THENGOSUB50000                     :rem 55
30 SK=0                                        :rem 110
32 GOSUB13200:ONPZGOTO 9000,9200,9400,9600,9800,10
   000,10200,10400,10600,10800                 :rem 235
50 PRINT:PRINT"{YEL}YOU SEE:"                  :rem 175
```

```
53 ONPZGOTO8010,8510,8710,8910,9020,9220,9420,9620
   ,9840                                          :rem 142
75 PRINT:PRINT:INPUT"{CYN}ACTION{WHT}";D$:PRINT"
   {WHT}"                                         :rem 227
78 IFAC=21THEN15024                                :rem 77
79 IFAC=22THEN15028                                :rem 83
80 D1$=LEFT$(D$,3)                                 :rem 165
90 G=3:G1=4                                        :rem 67
92 FORTY=1TO5                                      :rem 69
100 IFMID$(D$,G,1)=CHR$(32)THEND2$=MID$(D$,G1,LEN(
    D$))                                           :rem 192
110 G=G+1:G1=G1+1                                  :rem 124
120 NEXTTY                                         :rem 127
1000 REM D1$=KEYWORD:TEST FOR KEYWORD              :rem 38
1030 IFD1$=MID$(B$,47,1)THEN3050                   :rem 201
1040 IFD1$=MID$(B$,45,1)THEN3070                   :rem 202
1050 IFD1$=MID$(B$,43,1)THEN3090                   :rem 203
1060 IFD1$=MID$(B$,41,1)THEN3110                   :rem 195
1070 IFD1$=MID$(B$,37,3)THEN3130                   :rem 205
1080 IFD1$=MID$(B$,17,3)THEN3150                   :rem 206
1090 IFD1$=MID$(B$,13,3)THEN3170                   :rem 205
2000 IFD1$=MID$(B$,9,3)THEN3190                    :rem 156
2010 IFD1$=MID$(B$,5,3)THEN3210                    :rem 146
2020 IFD1$=MID$(B$,1,3)THEN3230                    :rem 145
2030 IFD1$=MID$(B$,21,3)THEN3250                   :rem 198
2040 IFD1$=MID$(B$,25,3)THEN3270                   :rem 205
2050 IFD1$=MID$(B$,29,3)THEN3290                   :rem 212
2060 IFD1$=MID$(B$,33,3)THEN3310                   :rem 201
2065 IFD1$=MID$(B$,49,3)THEN3320                   :rem 214
2070 IFD1$=MID$(B$,53,3)ORD1$=MID$(B$,57,3)THEN335
     0                                             :rem 243
2080 IFD1$=MID$(B1$,1,3)THEN3050                   :rem 200
2090 IFD1$=MID$(B1$,6,3)THEN3070                   :rem 208
2095 IFD1$=MID$(B1$,17,3)THEN3110                    :rem 2
3000 IFD1$=MID$(B1$,11,3)THEN3090                  :rem 246
3010 IFD1$=MID$(B1$,17,3)THEN3110                  :rem 246
3040 GOTO8500                                      :rem 205
3050 REM KEYWORD E(AST)                             :rem 79
3052 ONPZGOTO3053,3057,8700,8700,8700,8700,8700,87
     00,3060,8700                                  :rem 226
3053 PZ=2:GOTO32                                   :rem 188
3057 PZ=3:GOTO32                                   :rem 193
3060 PZ=8:GOTO32                                   :rem 192
3070 REM KEYWORD W(EST)                            :rem 103
3076 ONPZGOTO8700,3078,3078,8700,8700,8700,8700,30
     80,10800,8700                                 :rem 30
3078 POKECB,2:POKECS,2:PZ=PZ-1:GOTO32               :rem 66
3080 PZ=9:GOTO32                                   :rem 195
3090 REM KEYWORD S(OUTH)                           :rem 185
```

```
3092 ONPZGOTO8700,8700,3097,8700,8700,8700,3099,87
     00,8700,8700                                :rem 250
3097 PZ=4:GOTO32                                 :rem 198
3099 PZ=8:GOTO32                                 :rem 204
3110 REM KEYWORD N(ORTH)                         :rem 170
3115 ONPZGOTO8700,8700,8700,3122,8700,8700,8700,31
     19,8700,8700                                :rem 228
3119 POKECB,6:POKECS,12:PZ=7:GOTO32              :rem 162
3122 POKECS,2:POKECB,2:PZ=PZ-1:GOTO32             :rem 56
3130 REM KEYWORD SI(T)                            :rem 17
3135 PRINT"WHY DO YOU WANT TO DO THAT? REMEMBER,
     {3 SPACES}GET BACK TO YOUR OWN TIME!":rem 167
3140 FORTY=1TO3000:NEXT:GOTO32                   :rem 129
3150 REM KEYWORD RUN                             :rem 199
3155 PRINT"DON'T BE SCARED--FEAR BREEDS FEAR.":FOR
     TY=1TO5000:NEXT:GOTO32                      :rem 162
3170 REM KEYWORD SHO(W)                          :rem 102
3172 IFD2$="ID"ORD2$="PAPERS"ANDHJTHEN10050:rem 21
3175 GOTO8700                                    :rem 216
3190 REM KEYWORD INV                             :rem 195
3194 PRINT"YOU HAVE:"                            :rem 253
3196 ONIVGOTO3200,3205,3198,3206,3207,3208,3199,32
     01                                          :rem 224
3198 PRINTI$:PRINTE1$:FORCE=10TO2000:NEXT:GOTO32
                                                 :rem 45
3199 PRINTC1$:PRINTE1$:FORCE=10TO2000:NEXT:GOTO32
                                                 :rem 89
3200 PRINTI$:FORCE=1TO2000:NEXT:GOTO32           :rem 140
3201 PRINTI$:PRINTE1$:PRINTC1$:PRINTC2$:FORTY=1TO3
     000:NEXT:GOTO32                             :rem 211
3205 PRINTE1$:FORCE=1TO2000:NEXT:GOTO32          :rem 190
3206 PRINTC1$:FORCE=1TO2000:NEXT:GOTO32          :rem 189
3207 PRINTC1$:PRINTI$:FORCE=10TO2000:NEXT:GOTO32
                                                 :rem 34
3208 PRINTI$:PRINTE1$:PRINTC1$:FORCE=10TO2000:NEXT
     :GOTO32                                     :rem 132
3209 FORCE=1TO2000:NEXT:GOTO32                    :rem 97
3210 REM KEYWORD DRO(P)                           :rem 85
3215 I$="":E1$="":GOSUB7000:GOTO32               :rem 103
3230 REM KEYWORD GET                             :rem 177
3231 IFLO=0 ANDD2$=MID$(A$,13,5)THEN GOTO3239
                                                 :rem 232
3232 IFLO=111ANDD2$=MID$(A$,13,5)THEN3235         :rem 15
3233 IFPZ=3ANDD2$=MID$(E$,4,6)THEN3240           :rem 144
3235 IFD2$="CAN"THEN3237                          :rem 71
3236 GOTO8000                                    :rem 207
3237 PRINT"YOU DRANK IT-WHATEVER IT WAS TASTED
     {5 SPACES}HORRIBLE!"                          :rem 2
3238 FORTY=1TO3000:NEXT:GOTO32                   :rem 137
```

```
3239 I$=MID$(A$,13,5):A$="WINDOW,SEAT,CAN":LO=111:
     GOSUB7000:IV=1:GOTO32                    :rem 180
3240 E1$=MID$(E$,4,6):E$="":GOSUB7000:GOTO32
                                              :rem 151
3250 REM KEYWORD TAK(E)                       :rem 73
3252 IFD2$<>"POD"THEN8800                     :rem 149
3253 IFPZ=4THENINPUT"{CLR}{5 DOWN}{4 RIGHT}LEVEL(1
     -4)";LL:ONLLGOTO15000,15010,15020,15030
                                              :rem 14
3255 GOTO8900                                 :rem 217
3267 GOSUB7000:GOTO15                         :rem 242
3270 REM KEYWORD TAL(K)                       :rem 82
3271 IFPZ<>9THEN 8400                         :rem 180
3272 IFD2$="TO MAN"ORD2$="MAN"THEN 3274       :rem 142
3273 GOTO8400                                 :rem 212
3274 PRINT"{BLK}THE MAN SAYS,'YOU HAVE BEEN VERY S
     MART{2 SPACES}TO MAKE";                  :rem 133
3275 PRINT" IT THIS FAR. TAKE THIS IGNITION PLUG.
     {SPACE}IT WILL START THE GREEN ";        :rem 146
3276 PRINT"VEHICLE{3 SPACES}WITH THE BLUE TOP. GOO
     D LUCK.'":IV=8:C2$="PLUG"                :rem 144
3280 GOTO75                                   :rem 114
3285 GOSUB7000:GOTO15                         :rem 242
3290 REM KEYWORD USE                          :rem 196
3292 IFD2$="COMPUTERS"THENPRINT"YES, BUT HOW?":GOT
     O50                                      :rem 73
3294 IFD2$="CARD"ANDC1$<>""THEN9850           :rem 196
3295 IFD2$="POD"THEN3252                      :rem 91
3298 GOTO8900                                 :rem 224
3310 REM KEYWORD LOO(K)                       :rem 86
3312 L2$="OUT ":L1$=MID$(A$,1,6):IFL2$+L1$=D2$THEN
     3315                                     :rem 120
3314 GOTO8500                                 :rem 209
3315 PRINT"YOU SEE A VEHICLE OUTSIDE":GOSUB7000:GO
     TO30                                     :rem 44
3320 REM KEYWORD ENT(ER)                      :rem 160
3322 ONPZGOTO8700,8700,8700,3325,8700,3327,8700,87
     00,8700,8700                             :rem 234
3325 IFD2$="POD"THEN3253                      :rem 86
3327 IFD2$="DOOR"ANDLL=2THEN PRINT"YOU CANNOT WITH
     OUT VERIFICATION":GOTO3330               :rem 58
3330 FORTY=1TO3000:NEXT:GOTO32                :rem 130
3350 REM KEYWORD HIT OR KIL(L)                :rem 215
3355 PRINT"{CLR}{4 DOWN}{4 RIGHT}YOUR ATTEMPT AT V
     IOLENCE WAS UNWISE."                     :rem 56
3357 PRINT"{4 DOWN}{4 RIGHT}YOU HAVE BEEN EXTERMIN
     ATED."                                   :rem 78
3359 FORTY=1TO5000:NEXT:GOTO60000             :rem 32
5000 A$="WINDOW,SEAT,SHOES,CAN"               :rem 136
```

```
5010 B$="GET,DRO,INV,SHO,RUN,TAK,TAL,USE,LOO,SIT,N
     ,S,W,E,ENT,HIT,KIL"                    :rem 111
5015 B1$="EAST,WEST,SOUTH,NORTH"            :rem 245
5020 C$="ROOMS EAST & WEST":E$="ID PAPERS":F$="ELE
     VATION POD"                              :rem 0
5030 G$="COMPUTERS ALL AROUND":H$="GUARDS & DOOR":
     M1$="PEOPLE EVERYWHERE"                :rem 135
5040 M2$="{BLU}PEOPLE COMING FROM AND GOING TO ROO
     MS{3 SPACES}AHEAD."                    :rem 189
5045 M3$="BLACK SUITED MAN"                 :rem 250
5100 GOTO17                                 :rem 103
7000 PRINT"OK"                               :rem 50
7500 FORCE=10TO3000:NEXT:RETURN             :rem 210
8000 PRINT"YOU CAN'T GET THAT":GOSUB 13300:FORCE=1
     0TO2000:NEXT:GOTO32                    :rem 162
8010 PRINTA$:GOTO75                         :rem 154
8400 PRINT"TALK IS CHEAP.":FORCE=10TO2000:NEXT:GOT
     O32                                    :rem 241
8500 PRINT"DOES NOT COMPUTE":GOSUB 13300:FORCE=10T
     O2000:NEXT:GOTO32                      :rem 133
8510 PRINTC$:GOTO75                         :rem 161
8700 PRINT"YOU CAN'T":GOSUB 13300:FORCE=1TO3000:NE
     XT:GOTO32                              :rem 105
8710 PRINTE$:GOTO75                         :rem 165
8800 PRINT"TRY 'GET' (NOT GUARANTEED TO WORK BUT
     {3 SPACES}IT SOUNDS BETTER)"             :rem 0
8810 FORTY=1TO3000:NEXT:GOTO32              :rem 138
8900 PRINT"NO CAN DO":GOSUB13300:FORDI=1TO4000:NEX
     T:GOTO32                                :rem 41
8910 PRINTF$:GOTO75                         :rem 168
9000 REM PUZZLE 1                           :rem 200
9005 POKECS,2:POKECB,2                       :rem 61
9010 PRINT"{CLR}{2 DOWN}{BLK}YOU ARE IN DIMLY LIT
     {SPACE}ROOM.":GOTO50                   :rem 247
9020 PRINTG$:GOTO75                         :rem 162
9200 REM PUZZLE 2                           :rem 203
9210 POKECS,4:POKECB,4:PRINT"{CLR}{2 DOWN}{BLK}YOU
      ARE IN A HALLWAY.":PZ=2:IFIV=0THEN9215
                                            :rem 164
9213 GOTO50                                 :rem 109
9215 PRINT"YOU LOOKED SUSPICIOUS WITH NO SHOES ON.
     ":FORZ=1TO1000:NEXT:GOTO15007          :rem 208
9220 PRINTH$:GOTO75                         :rem 165
9400 REM PUZZLE 3                           :rem 206
9405 POKECS,2:POKECB,2                       :rem 65
9410 PRINT"{CLR}{2 DOWN}{BLK}YOU ARE IN A LARGE RO
     OM WITH AN OVAL{4 SPACES}TABLE IN ITS CENTER.
                                            :rem 242
9415 PZ=3:IV=2:IFI$THENIV=3:GOTO 50         :rem 129
```

```
9420 PRINTM1$:GOTO75                                    :rem 221
9600 REM PUZZLE 4                                       :rem 209
9610 POKECB,6:POKECS,12:PRINT"{CLR}{2 DOWN}{BLK}YO
     U ARE IN A CORRIDOR."                               :rem 3
9615 PZ=4:GOTO50                                        :rem 200
9620 PRINTM2$:GOTO75                                    :rem 224
9800 REM PUZZLE 5                                       :rem 212
9805 POKECS,6:POKECB,12                                 :rem 122
9810 PRINT"{CLR}YOU ARE ON THE FOURTH FLOOR. THE W
     ALLS{2 SPACES}OF THE HUGE CHAMBER ARE ";
                                                        :rem 25
9820 PRINT"LINED WITH{6 SPACES}HUNDREDS OF COMPUTE
     RS."                                               :rem 46
9830 PZ=5:AC=0:GOTO 50                                  :rem 243
9840 PRINTM3$:GOTO75                                    :rem 229
9850 NR=INT(RND(1)*100)+1                               :rem 108
9860 PRINT"{CLR}{BLK}":POKECB,11:POKECS,5               :rem 167
9865 INPUT"{4 DOWN}{8 RIGHT}TERMINAL #";BN              :rem 38
9870 IFBN<NRTHENPRINT"TRY TERMINAL RIGHT":GOTO9865
                                                        :rem 251
9880 IFBN>NRTHENPRINT"TRY TERMINAL LEFT":GOTO9865
                                                        :rem 171
9890 IFBN=NRTHENPRINT"{RVS}CORRECT TERMINAL{OFF}"
                                                        :rem 232
9895 FORTY=1TO3000:NEXT:GOSUB40000                      :rem 110
9900 PRINT"{CLR}AS3E43 382F6 9JDH8GEU LOW978H H14Q
     SN.":INPUT JK$                                     :rem 28
9910 IF JK$="Q"THEN60000                                :rem 18
9920 IF JK$="TRANSLATE"THEN9940                         :rem 80
9925 IFJK$="LOG ON"THEN9940                             :rem 38
9930 GOTO 9900                                          :rem 224
9940 PRINT"{CLR}{RVS}SYSTEM ACCESS COMMAND{OFF}":I
     NPUTKJ$:IFKJ$="Q" THEN 60000                       :rem 112
9950 IFKJ$="INFORMATION"THEN 9975                       :rem 243
9960 GOTO 9940                                          :rem 231
9975 PRINT"{CLR}{CYN}{4 DOWN}YOU SHOULD LOOK FOR A
      MAN WITH A BLACK{2 SPACES}SUIT ON."               :rem 20
9976 PRINT"{DOWN}YOU ARE TO GO TO THE FIRST LEVEL
     {SPACE}NOW.{3 SPACES}THE POD IS RETURNING..."
                                                        :rem 29
9980 FORTY=1TO6000:NEXT:GOTO9600                         :rem 0
10000 REM PUZZLE 6                                      :rem 245
10010 PRINT"{CLR}{RED}YOU ARE ON THE SECOND LEVEL.
       THERE ARE{2 SPACES}ARMED GUARDS STANDING ";
                                                        :rem 57
10020 PRINT"IN FRONT OF THE{3 SPACES}ENTRANCE TO A
      NOTHER ROOM."                                     :rem 201
10030 PZ=6:AC=0:HJ=34:GOTO50                            :rem 132
10050 GOSUB7000:PRINT"YOU ARE INSIDE THE COMPUTER
      {SPACE}ACCESS BANK.YOU ARE GIVEN ";               :rem 174
```

```
10060 PRINT"A CARD. YOU LEAVE. THE{4 SPACES}POD IS
       RETURNING......."                          :rem 73
10070 FORTY=1TO6000:NEXT:C1=1:C1$="CARD":PZ=6
                                                  :rem 188
10080 IV=4:IFI$THENIV=5                           :rem 127
10085 IFE1$THENIV=7                               :rem 105
10090 IFI$<>""ANDE1$<>""THENIV=6                   :rem 32
10100 GOTO9600                                    :rem 250
10200 REM PUZZLE 7                                :rem 248
10210 GOTO15000                                    :rem 35
10400 REM PUZZLE 8                                :rem 251
10405 POKECS,8:POKECB,8:FL=0                      :rem 174
10410 PRINT"{CLR}{BLU}THE MAN HAS DISAPPEARED. YOU
       HEAR MEN{3 SPACES}TALKING IN A FOREIGN ";
                                                  :rem 134
10420 PRINT"LANGUAGE. YOU ALSO HEAR ENGINES RUNNIN
      G TO THE WEST."                             :rem 165
10430 PZ=8:GOTO50                                 :rem 239
10600 REM PUZZLE 9                                :rem 254
10610 PRINT"{CLR}{CYN}YOU ARE IN A GARAGE THAT LEA
      DS TO THE{3 SPACES}FRONT OF THE BUILDING."
                                                  :rem 232
10620 PZ=9:GOTO50                                 :rem 241
10800 REM PUZZLE 10                                :rem 40
10810 POKECB,4:POKECS,4                           :rem 109
10820 PRINT"{CLR}YOU ARE OUTSIDE THE BUILDING. THE
      RE ARE SEVERAL VEHICLES THERE ";             :rem 22
10830 INPUT"NOW. WHICH IS THEVEHICLE YOU WANT";VH$
                                                  :rem 22
10840 IF VH$="GREEN WITH BLUE TOP" OR{2 SPACES}VH$
      ="BLUE AND GREEN" THEN 10842                :rem 21
10841 IF VH$<> "GREEN AND BLUE" THEN 10890:rem 170
10842 POKECB,6:POKECS,14:Q=1                      :rem 158
10843 FORBV=1TO10:GOSUB 13400                     :rem 163
10844 IFQTHENPOKECS,5:POKECB,6:PRINT"{BLU}":Q=0:GO
      TO10850                                      :rem 25
10845 POKECS,6:POKECB,14:PRINT"{GRN}":Q=1  :rem 202
10850 PRINT"{CLR}YOU CHOSE THE CORRECT VEHICLE. YO
      U ARE{2 SPACES}NOW ON YOUR WAY BACK TO ";
                                                  :rem 135
10860 PRINT"YOUR OWN TIME{3 SPACES}PERIOD. YOU CAN
       SAVE THE WORLD FROM{5 SPACES}DISASTER!"
                                                   :rem 78
10865 NEXTBV                                      :rem 219
10870 GETAG$:IFAG$=""THEN 10870                   :rem 177
10880 GOTO60000                                    :rem 48
10890 PRINT"{CLR}":POKECS,7:POKECB,7:PRINT"
      {4 DOWN}YOU HAVE BEEN ARRESTED AND FORCED ";
                                                  :rem 190
```

```
10900 PRINT"TO{4 SPACES}TAKE A KNOCKOUT PILL."
                                              :rem 161
10910 FORTY=1TO3000:NEXT:PRINT"{4 DOWN}YOU WILL WA
      KE SHORTLY.":POKECS,0:POKECB,0          :rem 144
10920 FORTY=1TO10000:NEXT:CLR:SK=30:GOTO10:rem 114
13000 FORI=CV TO CV+24:POKEI,0:NEXT           :rem 78
13010 POKE CV+24,15:POKE CV+5,128:POKE CV+6,248
                                              :rem 54
13020 FOR J=100 TO 245:POKE CV,J:POKE CV+4,129:FOR
       G=1 TO 50:NEXT:NEXT                    :rem 164
13040 POKE CV+12,128:POKE CV+13,128           :rem 109
13045 FORT=100 TO 145:POKE CV+8,T:POKE CV+11,33:NE
      XT                                      :rem 194
13050 FORT=147 TO 100 STEP-1:POKE CV+8,T:POKE CV+1
      1,17                                    :rem 227
13060 NEXT:POKE CV+8,0                        :rem 250
13100 RETURN                                  :rem 213
13200 FORI=CV TO CV+24:POKE I,0:NEXT          :rem 80
13210 POKE CV+24,15:POKE CV+5,128:POKE CV+6,64
                                               :rem 4
13220 FOR X=1 TO 25:R1=INT(RND(0)*(245-100+1))+100
      :POKE CV+1,R1:POKE CV+4,17              :rem 56
13230 NEXT:POKECV+1,0:POKE CV+24,0:RETURN :rem 251
13300 FORI=CV TO CV+24:POKE I,0:NEXT:POKE CV+24,15
      :POKE CV+5,128:POKE CV+6,248            :rem 204
13310 POKE CV+1,5:POKE CV+4,33:FOR B3=1 TO 890: NE
      XT:POKE CV+1,0:POKE CV+24,0             :rem 218
13320 RETURN                                  :rem 217
13400 FORI= CV TO CV+24:POKEI,0:NEXT          :rem 82
13410 POKE CV+24,15:POKE CV+5,64:POKE CV+6,128
                                               :rem 6
13420 FOR H=160 TO 189:POKE CV+1,H:POKE CV+4,33:NE
      XT:POKE CV+1,0:POKE CV+24,0             :rem 41
13430 RETURN                                  :rem 219
15000 REMLL=1                                 :rem 224
15002 PRINT"{CLR}YOU ARE ON THE FIRST LEVEL. THE M
      AN YOU ARE LOOKING FOR IS ";            :rem 64
15004 PRINT"WALKING SOUTH.":FORTY=1TO5000:NEXT:PZ=
      7:IV=IV:IFFLTHEN50                      :rem 178
15005 PRINT"{3 DOWN}THE AUTHORITIES COME UP AND QU
      ESTION{4 SPACES}HIM. THEY NOW LOOK ";
                                              :rem 239
15006 PRINT"AT YOU. {RVS}OH NO!{OFF}"         :rem 134
15007 FORTY=1TO5000:NEXT:PRINT"{DOWN}YOU HAVE BEEN
       ARRESTED AND REIMPRISONED."            :rem 35
15008 FORTY=1TO5000:NEXT:CLR:SK=30:GOTO10     :rem 72
15010 REMLL=2                                 :rem 226
15015 GOTO 10000                               :rem 38
15020 PRINT"{CLR}{3 DOWN}THE THIRD LEVEL IS THE IN
      TERROGATION{4 SPACES}CENTER."           :rem 180
```

```
15022 PRINT"{DOWN}YOU HAVE NOW BEEN BRAINWASHED.":
      AC=21:GOTO75                              :rem 158
15024 PRINT"{2 DOWN}YOU HAVE TRIED TO ESCAPE AND H
      AVE BEEN{2 SPACES}CAUGHT."                :rem 215
15026 PRINT"{DOWN}YOU CANNOT RESPOND TO QUESTIONS
      {SPACE}ABOUT{3 SPACES}YOUR LIFE AND ARE ABOU
      T ";                                      :rem 116
15027 PRINT"TO BE{11 SPACES}EXTERMINATED.":AC=22:G
      OTO75                                     :rem 240
15028 PRINT"{2 DOWN}YOUR EXPLANATION WAS INSUFFICI
      ENT. YOU{2 SPACES}HAVE BEEN EXTERMINATED."
                                                :rem 24
15029 FORUU=1TO5000:NEXT:GOTO60000              :rem 74
15030 REMLL=4                                   :rem 230
15040 GOTO9800                                   :rem 4
40000 REM COMPUTER                              :rem 71
40010 FL=30:PRINT"{CLR}{WHT}":FORT=CCTOEC+40:POKET
      ,1:NEXT                                   :rem 79
40020 POKECB,1:POKECS,1:FORTY=1TO1000:NEXT:POKECB,
      0:POKECS,0                                :rem 33
40025 GOSUB 42500:FORTY=1 TO 5000               :rem 26
40030 NEXT:POKECB,6:POKECS,6:FORK=SCTOSC+V:POKEK,1
      02:NEXT:FORM=ESTOES+V                     :rem 203
40040 POKEM,102:NEXT:FORD=SC+STOESSTEPS:POKED,102:
      NEXT                                      :rem 17
40045 FORE=SC+VTOES+VSTEPS:POKEE,102:NEXT       :rem 174
40050 FORZA=SC+133TOSC+221STEPS:POKEZA,78:NEXT:FOR
      ZA=SC+132TOSC+221STEPS                    :rem 136
40055 POKEZA,233:NEXT:FORUM=SC+135TOSC+223STEPS:PO
      KEUM,118:NEXT                             :rem 51
40057 FORUM=SC+136TOSC+224STEPS:POKEUM,233:NEXT
                                                :rem 244
40060 FORGE=SC+143TOSC+231STEPS:POKEGE,78:NEXT:FOR
      GE=SC+142TOSC+230STEPS                    :rem 95
40065 POKEGE,223:NEXT                           :rem 246
40070 FORGE=SC+144TOSC+232STEPS:POKEGE,118:NEXT:FO
      RGE=SC+145TOSC+233STEPS                   :rem 147
40080 POKEGE,127:NEXT                           :rem 246
40090 FORZK=SC+451TOSC+467:RC=INT(RND(1)*26)+1:POK
      EZK,RC:NEXT                               :rem 10
40150 GETQW$:IFQW$=""THEN40050                  :rem 228
40200 RETURN                                    :rem 214
42500 FORL=CVTOCV+24:POKEL,0:NEXT                :rem 91
42510 POKECV+24,15                              :rem 231
42520 POKECV+19,72                              :rem 239
42530 POKECV+20,129                              :rem 27
42540 POKECV+18,33                              :rem 237
42550 FORT=1TO75:D=INT(RND(1)*198)+3             :rem 63
42560 POKECV+15,D:POKECV+14,50:POKECV+18,17:rem 23
```

```
42570 FORG=1TO100:NEXT:NEXT                    :rem 196
42580 POKECV+18,32:RETURN                      :rem 10
50000 POKE CS,9:POKECB,3:PRINT"{CLR}{CYN}":rem 169
50010 PRINT"{DOWN}{12 RIGHT}{RVS}TIME CAPSULE"
                                               :rem 130
50020 PRINT"{3 DOWN}{2 RIGHT}DURING A SEARCH IN OU
      TER SPACE FOR{4 SPACES}{2 RIGHT}MISSING ";
                                               :rem 253
50030 PRINT"ASTRONAUTS YOU ARE CAPTURED{3 SPACES}
      {2 RIGHT}BY THE ENEMY. ";                :rem 167
50040 PRINT"THEY PLACE YOU IN THEIR {2 RIGHT}SECRE
      T WEAPON: THE NEW ";                     :rem 131
50050 PRINT" {RVS}TIME CAPSULE{OFF}."          :rem 217
50060 PRINT"{DOWN}{2 RIGHT}THIS IS WHERE ALL THE O
      THER ";                                  :rem 221
50070 PRINT"{10 SPACES}{2 RIGHT}ASTRONAUTS HAVE BE
      EN DISAPPEARING TO."                     :rem 161
50080 PRINT"{DOWN}{2 RIGHT}YOU'RE TRANSPORTED TO T
      HE YEAR {RVS}2185{OFF}.{2 SPACES}";      :rem 188
50090 PRINT"{2 RIGHT}ESCAPE WILL BE DIFFICULT."
                                               :rem 65
50110 PRINT"{DOWN}{2 RIGHT}YOUR MISSION: RETURN TO
       YOUR OWN TIME ";                        :rem 238
50120 PRINT"{2 SPACES}TO WARN YOUR LEADERS OF THE
      {SPACE}ENEMY'S{5 SPACES}ACTIONS. "       :rem 160
50130 PRINT"{DOWN}{2 RIGHT}PRESS {RVS}RETURN{OFF}"
                                               :rem 38
50140 GETTP$:IFTP$=""THEN50140                 :rem 221
50150 IFTP$<>CHR$(13)THEN50000                 :rem 166
50160 POKECB,9:POKECS,3:PRINT"{CLR}{BLK}" :rem 161
50170 PRINT"{DOWN}{12 RIGHT}{RVS}TIME CAPSULE"
                                               :rem 137
50180 PRINT"{DOWN}{2 RIGHT}YOU MAY USE SHORT PHRAS
      ES SUCH AS{5 SPACES}";                   :rem 147
50190 PRINT"{2 RIGHT}'GET CAN'. TO MOVE FROM PLACE
       TO{6 SPACES}{2 RIGHT}PLACE DO NOT USE ";
                                               :rem 153
50200 PRINT"'GO': SIMPLY TYPE{4 SPACES}{2 RIGHT}TH
      E DIRECTION IN WHICH YOU WISH TO{4 SPACES}";
                                               :rem 226
50210 PRINT"{2 RIGHT}MOVE. FOR EXAMPLE, 'EAST' OR
      {SPACE}'E' TO{3 SPACES}{2 RIGHT}GO EAST. "
                                                 :rem 4
50220 PRINT"{DOWN}{2 RIGHT}THE COMPUTER WILL TELL
      {SPACE}YOU IF A WORD{4 SPACES}YOU HAVE USED
      {SPACE}";                                :rem 100
50230 PRINT"IS ILLEGAL. PLAY SMART.{3 SPACES}THE A
      NSWERS ARE EASY!"                        :rem 169
```

```
50240 PRINT"{DOWN}{2 RIGHT}PRESS {RVS}RETURN{OFF}"
                                                  :rem 40
50250 GET TP$:IF TP$=""THEN 50250                :rem 225
50260 IFTP$<>CHR$(13)THEN50160                   :rem 175
50270 RETURN                                     :rem 222
59999 END                                        :rem 240
60000 PRINT"{CLR}":POKECS,1:POKECB,1:INPUT"{BLK}
      {6 DOWN}{4 RIGHT}PLAY AGAIN";FF$           :rem 250
60005 IFFF$="Y"THEN RUN                          :rem 56
```

3
Strategy Games

Sea Route to India:

A Historical Simulation for the 64

M. J. Winter

Here's your chance to make history on the "Sea Route to India." Following in the wake of Portuguese explorers, you can find gold and adventure. That is, if you don't starve, get sunk by pirates, or capsize in a terrible storm.

Simulations are more than just games. Although often played like a game, they try to recreate a slice of the real world. Using actual historical information, the simulation designer forces the player to make decisions. If the simulation is well-researched and well-written, those decisions should be very similar to the ones the actual participants had to make.

These kinds of games are usually very popular, for they can be played again and again, with no two games seeming the same. Simulations are also somewhat educational, especially if they're based on a period in history. Not only are they fun to play, but you end up learning something at the same time.

One of the earliest games for PET computers was a simulation called *Westward Ho.* You became a turn-of-the-century pioneer, trying to cross the country in a covered wagon. Decisions had to be made about purchasing food, supplies, and ammunition. Various experiences—hunting, Indian attacks, settlements—occurred on each leg of the journey. By playing the game several times, you learned where to spend money, how to hunt, and whether to trust strangers. Luck, however, was a factor in success, just as it was in reality. PET users of all ages played the game over and over until their characters finally reached the West Coast.

But *Westward Ho* was only an abbreviated version of *Oregon Trail,* another simulation. That game's designers took great pains to produce an accurate imitation of reality; they used prices from contemporary catalogs and calculated frequencies and likely locations of Indian attacks by studying

historical accounts. The result was a historical game that was both interesting and informative.

Sail the Bounding Main

"Sea Route to India" uses a similar technique, drawing on the voyages made by Portuguese explorers in the fifteenth century. You'll have to make some of the same decisions as these explorers, even keeping track of your ship's supplies and your crew's morale.

Type the program in and save it. You should use "The Automatic Proofreader," found in Appendix C, to enter Sea Route to India. It will almost insure that you type it in correctly the first time. Once you've entered the program, load and run it.

You'll see the map of Europe, Africa, and part of Asia appear on the screen. Press *B* to begin the game. If this is your first time playing Sea Route to India, press the *Y* key to read the short instructions. If you look at the program listing, you'll see that the subroutine beginning at line 15000 introduces the game and sets up the rules. You'll probably want to skip the directions if you've played the simultion before. Hit any other key to go directly to the start of the game.

Your goal is to sail from Lisbon around Africa to India. During the voyage, you encounter the same dangers faced by the real explorers: hunger, thirst, pirates, natives, weather, mutiny, and attack by Arab traders. If you make the right decisions, keep your crew fed and happy, and have a good bit of luck besides, you might just make it. The risks are great, but the rewards can be even greater.

The voyage is charted in weeks on a map displayed on the screen. Lines 500–800 contain the loop for each week. The miles you sail depend on the weather. Sometimes the winds are favorable, other times fierce storms batter your ship. You can even be becalmed for weeks on end. Each week, your store of water, food, and supplies decreases by one unit. And if your journey lasts more than 30 weeks, the crew's morale also drops by one.

Each week something happens. Line 560 sends the program to the appropriate event. In the early parts of the voyage, you may sight whales, meet other ships, or sail into terrible storms. After you navigate around the Cape of Good

Hope and pick up your Indian pilot at Malindi, you can be attacked at any time by Arab pirates.

Ship's Log

At the end of each week, the program assesses your situation. If you sailed far enough to visit the Canary or Cape Verde Islands, then your water, food, supplies, and crew morale are restored. The ship's log is updated, and the game map shows your progress. Lines 91–93 define the variable DT$ (dots); three characters are needed for each dot. On the map, one dot represents 200 miles. Then, if there are no fatal shortages (such as no water or food or crew morale at zero), the voyage continues.

Your ship's record is also updated at the end of each week. *Weeks Out* and *Miles Sailed* are increased, while the other catagories of *Food, Water, Supplies, Gold,* and *Crew Morale* can be raised or lowered, depending on what happened that week. Whenever your food or supplies fall below one, or your crew's morale below two, the game ends. You can find the checks for these situations in lines 11010–11035.

Occasionally, you'll come across another ship on the high seas. It's your choice whether you want to approach or flee. If it's a friendly ship, its captain may challenge you to a race. Program line 3060 places the ships at the right side of the screen; a string of DELETEs is printed several times to move the ships to the left. If you win the race, your crew is happier; not only can you win the wager, but their morale can increase. However, they become disgruntled by a loss, and their morale will fall.

Running short of food? Explorers often hunted whales while at sea, and you can do the same. Lines 1000–1250 contain the whale hunting routine. The whales are PRINTed within a long string (F$) of shifted spaces. The string is cyclically rearranged and the leftmost 40 characters PRINTed each time. The program checks the keyboard, then moves the whales until you press *H,* which drops the harpoon. The program then alternately moves the whales and lowers the harpoon.

To check whether the harpoon hits a whale, the screen is opened for INPUT in line 1100. The entire row of the screen to the right of the harpoon is considered INPUT. If the first character is not a shifted space, a whale has been hit.

Landfall

The subroutine beginning at line 4000 describes the sighting of a river mouth. Landing offers you a chance to get food and water and to cheer up the crew. Shore leave was important even then! Sometimes, depending on the random result of line 4060, natives appear. As many early explorers discovered, they are unpredictable. Sometimes they're friendly and trade gold for goods (which raises your crew's morale); other times they attack.

If they do attack, type RUN and press RETURN quickly. The clock is set to 0 in line 4320 to time how long it took you to enter RUN. After you press RETURN, the program looks at the clock. If more than 200 jiffies have passed (line 4340), the natives attack and kill you. If you were fast, your landing party escapes, although the food and water it gathered is lost.

The same timing technique is used when the Arab pirates' dhows attack. The Arabs are fiercely determined to protect their trading routes. Vasco da Gama himself was nearly trapped by them more than once.

India at Last

If you make the right decisions, avoid starving to death, keep your crew happy, and outrun Arab pirates, you may find your way to India. If so, you'll be congratulated. To play the simulation again, respond to the prompt on the screen. You can review the instructions if you want, or go directly to another game.

It took the Portuguese more than one voyage to reach the riches of the subcontinent; if you made it the first time, you're either a very good explorer or very lucky. Whatever the result, you'll be sure to have fun and learn at the same time.

Sea Route to India

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
70 DIM M%(7),M$(6):X=RND(-TI)                    :rem 45
71 FG=1.6:KB=151:HP=38:PRINT"{CLR}":IFPEEK(1024)=3
   2THENKB=197:HP=29                             :rem 54
72 M$(0)="":M$(1)="STOPPED AT CANARY ISLANDS":M$(5
   )="SIGHTED CALICUT"                          :rem 201
73 DH$="{2 SPACES}{L}{DOWN}{3 LEFT}{*}{RVS}
   {3 SPACES}{*}{DOWN}{4 LEFT}{4 SPACES}{DOWN}
   {4 LEFT}{4 SPACES}{DOWN}{5 LEFT}{RVS}£
   {3 SPACES}{OFF}£{DOWN}{3 LEFT}{L}{DOWN}
   {4 LEFT}{*}{RVS}{6 SPACES}{OFF}£"            :rem 122
```

```
74 M$(2)="CAPE VERDE ISLANDS":M$(3)="ROUNDED CAPE
   {SPACE}OF GOOD HOPE"                          :rem 65
76 M$(4)="PICKED UP INDIAN PILOT"                :rem 116
77 YS$=" -{DOWN}{LEFT}{RVS}+{DOWN}{LEFT}{OFF}-
   {DOWN}{2 LEFT}{*}{RVS} {OFF}£":HS$="{RED} -
   {DOWN}{LEFT}{RVS}Z{OFF}{DOWN}{LEFT}-{DOWN}
   {2 LEFT}{*}{RVS} {OFF}£"                      :rem 108
78 MS$="{RIGHT}"+CHR$(20)+"{DOWN}":MS$=MS$+MS$:MS$
   =MS$+MS$                                      :rem 12
80 WH$(1)="NO WIND AT ALL":WH$(2)="VERY CALM":WH$(
   3)="FAIR WINDS"                               :rem 120
82 WH$(4)="GOOD WINDS":WH$(5)="GOOD WINDS":WH$(6)=
   "STRONG WINDS"                                :rem 250
84 M%(1)=50:M%(2)=100:M%(3)=150:M%(4)=200:M%(5)=25
   0:M%(6)=300:M%(7)=50                          :rem 225
90 GOSUB16000                                    :rem 224
91 DT$="{2 LEFT}.{DOWN}{LEFT}.{2 LEFT}.{2 LEFT}.
   {DOWN}{LEFT}.{DOWN}{LEFT}.{DOWN}{LEFT}.{DOWN}
   {LEFT}.{2 LEFT}.{DOWN}{LEFT}.{DOWN}{LEFT}.
   {DOWN}{LEFT}.{DOWN}{OFF}.{RVS}{OFF}.{DOWN}{OFF}
   .{RVS}{OFF}.{RVS}{OFF}.{DOWN}{LEFT}.{DOWN}
   {LEFT}.{DOWN}{LEFT}.{DOWN}{LEFT}.{DOWN}{LEFT}."
                                                 :rem 139
92 DT$=DT$+"{DOWN}{LEFT}.{DOWN}{LEFT}.{DOWN}{OFF}.
   {DOWN}{OFF}.{RVS}{OFF}.{RVS}{OFF}.{RVS}{OFF}.
   {UP}{LEFT}.{UP}{OFF}.{UP}{OFF}.{UP}{LEFT}.{UP}
   {OFF}.{UP}{LEFT}.{UP}{LEFT}.{UP}{LEFT}.{UP}
   {OFF}.{UP}{OFF}.{RVS}{OFF}.{RVS}{OFF}.{RVS}
   {OFF}."                                       :rem 54
93 DT$=DT$+"{RVS}{OFF}.{UP}{OFF}.{UP}{LEFT}."
                                                 :rem 15
104 DEF FNR(X)=INT(RND(1)*X+1)                   :rem 176
105 F$="{2 SHIFT-SPACE}<.{2 @}V{13 SHIFT-SPACE}(.
    {2 I}N{34 SHIFT-SPACE}"                      :rem 17
106 F$=F$+"{28 SHIFT-SPACE}"                     :rem 151
107 F$=F$+"<.{2 +}V{40 SHIFT-SPACE}(,{2 +}N
    {15 SHIFT-SPACE}"                            :rem 114
108 F$=F$+"{28 SHIFT-SPACE}"                     :rem 153
110 D$="{HOME}{32 DOWN}"                         :rem 174
120 S$="{3 SPACES}{M}{2 SPACES}{M}{DOWN}{5 LEFT}
    {3 +} {2 +}{DOWN}{6 LEFT}{3 £} {2 £}{DOWN}
    {6 LEFT}{3 +} {2 +}{DOWN}{4 LEFT}{G} {M}
    {DOWN}{6 LEFT}{*}{RVS} ZZZ {OFF}£"     :rem 135
122 SS$="{3 SPACES}{M}{2 SPACES}{M}{DOWN}{5 LEFT}
    {3 +} {2 +}{DOWN}{6 LEFT}{3 £} {2 £}{DOWN}
    {6 LEFT}{3 +} {2 +}{DOWN}{4 LEFT}{G} {M}
    {DOWN}{6 LEFT}{*}{RVS}{5 SPACES}{OFF}£ :rem 44
```

```
125 GOSUB15000                                    :rem 14
130 GOTO500                                       :rem 98
500 ML=8:GOSUB10000:FORWK=1TO52:Z=FRE(0):FORI=1TO1
    0:GETA$:NEXT                                  :rem 140
510 GOSUB14000:POKE53281,3:REMWEATHER,MILES
                                                  :rem 170
520 GOSUB10000:REM LOG                             :rem 8
530 GOSUB11000:REM SITUATION                      :rem 232
550 X=FNR(GG):IFGG=6THENX=2*FNR(4)-1              :rem 167
555 IFML<1200ANDX=4THENX=1                         :rem 87
560 ONXGOSUB1000,2000,1000,4000,5000,6000,6000
                                                  :rem 113
790 FD=FD-1:SP=SP-1:WT=WT-1:IFWK>30THENCH=CH-1
                                                  :rem 129
800 NEXTWK                                        :rem 121
1000 REM CATCH WHALE ROUTINE                      :rem 159
1002 DZ=17+INT(8*RND(1))                           :rem 53
1005 PRINT"{CLR}WHALES SIGHTED"                   :rem 246
1006 PRINT"{DOWN}TRY YOUR LUCK? Y OR N"           :rem 173
1007 A$="":GETA$:IFA$="N"THEN 1155                 :rem 37
1008 IFA$<>"Y"THEN1007                            :rem 203
1010 PRINT"{CLR}{DOWN}{11 SPACES}W{DOWN}{LEFT}[+]*
     -{DOWN}{3 LEFT}[+] V"                        :rem 227
1020 PRINT"{8 SPACES}[*]{RVS}{3 SPACES}{OFF}£"
                                                  :rem 192
1030 PRINT"[5]JK{SHIFT-SPACE}JKJKJKJKKJKJKJKJKJKJK
     JK{BLK}"                                     :rem 36
1040 PRINT"{HOME}PRESS H{OFF} "                   :rem 16
1050 GOSUB1200                                     :rem 9
1055 IFPEEK(KB)<>HPTHEN1050                       :rem 100
1058 DC=0:PRINTLEFT$(D$,3)TAB(13)" {DOWN}{LEFT}-
     {DOWN}{LEFT}V{DOWN}{LEFT}";                  :rem 56
1060 DC=DC+1:GOSUB1200:PRINTLEFT$(D$,3+DC)TAB(13)"
      {DOWN}{LEFT}-{DOWN}{LEFT}V{DOWN}{LEFT}";:GOT
     O1070                                        :rem 78
1070 IFDC<>DZ-6THEN1060                           :rem 79
1100 B$="*":OPEN3,3:INPUT#3,B$:CLOSE3:IFLEFT$(B$,1
     )<>"{SHIFT-SPACE}"THEN1150                   :rem 230
1110 PRINTLEFT$(D$,3+DC)TAB(13)" {DOWN}{LEFT}-
     {DOWN}{LEFT}V{LEFT}{UP} {DOWN}{LEFT}-{DOWN}
     {LEFT}V {2 UP}MISSED";:GOTO1155              :rem 230
1150 PRINT"{7 UP}GOOD SHOT":FD=FD+2               :rem 222
1155 PRINTLEFT$(D$,23) "{4 UP}PRESS {RED}RETURN
     {BLK}"                                       :rem 147
1157 A$="":GETA$:IFA$<>CHR$(13)THEN1157           :rem 142
1159 RETURN                                       :rem 176
1200 PRINTLEFT$(D$,DZ)LEFT$(F$,39)                :rem 196
1210 F$=MID$(F$,2)+LEFT$(F$,1)                     :rem 20
1245 REM{4 SPACES}A$="":GETA$:IFA$<>"L"THEN1245
                                                  :rem 72
```

```
1250 RETURN                                          :rem 168
2000 REMFOREIGN SHIPS                                :rem 55
2010 PRINT"{CLR}{RED}"TAB(25)S$                      :rem 73
2020 PRINT"{BLK}SHIP SIGHTED"                        :rem 97
2030 PRINT"{RVS}A{OFF}PPROACH OR {RVS}F{OFF}LEE"
                                                     :rem 233
2040 A$="":GETA$:IFA$<>"A"ANDA$<>"F"THEN2040
                                                     :rem 137
2050 IFA$="A"ANDRND(1)>.2THEN3000                    :rem 70
2060 ES=.5:IFA$="F"THENES=.8:GOTO2100                :rem 149
2070 PRINT"{DOWN}IT'S A PIRATE SHIP!":PRINT"{DOWN}
     YOU TURN AND FLEE"                              :rem 110
2100 REMFLEE ROUTINE                                 :rem 233
2110 IFRND(1)>ESTHENPRINT"{2 DOWN}ALAS.":PRINT"
     {DOWN}THEY CATCH AND SINK YOU":GOTO17000
                                                     :rem 41
2199 GOTO3140                                        :rem 214
3000 FL=0:REM RACE                                   :rem 251
3001 PRINT"{CLR}ITS CAPTAIN CHALLENGES YOU TO A RA
     CE{DOWN}":IFRND(1)>.5THEN3003                   :rem 192
3002 PRINT"3 PIECES OF HIS GOLD FOR 3{2 SPACES}BAR
     RELS OF{2 SPACES}YOUR SUPPLIES.":GOTO3005
                                                     :rem 8
3003 FL=1:PRINT"3 BARRELS OF HIS SUPPLIES AGAINST"
                                                     :rem 4
3004 PRINT"3 PIECES OF YOUR GOLD."                   :rem 188
3005 PRINT"{DOWN}DO YOU ACCEPT? {RVS}Y{OFF} OR
     {RVS}N{OFF}?"                                   :rem 248
3006 A$="":GETA$:IFA$="Y"THEN3017                    :rem 48
3007 MS$="{RIGHT}"+CHR$(20)+"{DOWN}":MS$=MS$+MS$:M
     S$=MS$+MS$                                      :rem 103
3008 IFA$="N"THENRETURN                              :rem 157
3009 GOTO3006                                        :rem 206
3010 YS$="{BLK} -{DOWN}{LEFT}{RVS}+{DOWN}{LEFT}
     {OFF}-{DOWN}{2 LEFT}[*]{RVS} {OFF}£":HS$="
     {RED} -{DOWN}{LEFT}{RVS}Z{OFF}{DOWN}{LEFT}-
     {DOWN}{2 LEFT}[*]{RVS} {OFF}£"                  :rem 82
3017 MS$="{RIGHT}"+CHR$(20)+"{DOWN}":MS$=MS$+MS$:M
     S$=MS$+MS$                                      :rem 104
3050 PRINT"{CLR}{BLK}";                              :rem 247
3060 PRINTTAB(36)YS$:PRINT"{2 DOWN}"TAB(36)HS$
                                                     :rem 52
3070 FORT=1TO1000:NEXT                               :rem 80
3075 YX=INT(RND(1)*10)+25:HX=INT(RND(1)*9)+25:IFHX
     =YXTHENYX=YX+1                                  :rem 171
3080 MX=YX:W$="YOUR":IFYX<HXTHENMX=HX:W$="HIS"
                                                     :rem 4
3090 FORJ=1TOMX                                      :rem 179
3092 IFYX<JTHEN3095                                  :rem 148
```

```
3093 PRINT"{HOME}"MS$                                  :rem 119
3095 IFHX<JTHEN3100                                    :rem 121
3096 PRINT"{HOME}{6 DOWN}"MS$                          :rem 224
3100 NEXTJ                                              :rem 77
3110 PRINT"{BLK}{HOME}{15 DOWN}"W$" SHIP WINS"
                                                       :rem 108
3120 IFMX=YXTHENGP=GP-(FL=0)*3:SP=SP+3*FL: CH=CH+2
                                                        :rem 29
3130 IFMX=HXTHENGP=GP-FL*3:SP=SP+3*(FL=0): CH=CH-2
                                                        :rem 15
3140 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                        :rem 75
3145 A$="":GETA$:IFA$<>CHR$(13)THEN3145                :rem 140
3150 RETURN                                            :rem 169
4000 REMRIVER                                           :rem 48
4010 IFRND(1)>.7THEN RETURN                            :rem 108
4020 PRINT"{CLR}YOU SPY A RIVER.":PRINT"{DOWN}WILL
      YOU GO ASHORE FOR FOOD AND WATER?"               :rem 103
4025 PRINT"{DOWN}{RVS}Y{OFF} OR {RVS}N{OFF}"
                                                        :rem 61
4030 A$="":GETA$:IFA$<>"Y"ANDA$<>"N"THEN4030
                                                       :rem 171
4040 IFA$="N"THEN CH=CH-2:RETURN                       :rem 134
4050 PRINT"{2 DOWN}YOU LAND AND REPLENISH."
                                                       :rem 131
4060 IFRND(1)>.5THEN4800                                :rem 91
4070 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                        :rem 78
4075 A$="":GETA$:IFA$<>CHR$(13)THEN4075                :rem 146
4080 PRINT"{CLR}NATIVES APPEAR{3 SPACES}O
     {3 SPACES}O{DOWN}{6 LEFT}J{RVS}{RED} {BLK}
     {OFF}K {Z}{RVS}{GRN} {OFF}{BLK}{X}{DOWN}
     {6 LEFT}V{3 SPACES}V"                             :rem 108
4082 PRINT"{DOWN}{RVS}A{OFF}PPROACH OR {RVS}F{OFF}
     LEE?"                                              :rem 66
4083 A$="":GETA$:IFA$<>"A"ANDA$<>"F"THEN4083
                                                       :rem 155
4085 IFA$="F"THEN4800                                  :rem 135
4090 IFRND(1)>.5THEN4300                                :rem 89
4100 PRINT"{3 DOWN}THE NATIVES TRADE GOLD FOR YOUR
      TRINKETS"                                          :rem 4
4110 GP=GP+10:CH=CH+1:GOTO4800                         :rem 229
4300 PRINT"{CLR}{2 SPACES}O{3 SPACES}O{DOWN}
     {6 LEFT}J{RVS}{YEL} {BLK}{OFF}K↑{Z}{RVS}{RED}
      {OFF}{BLK}{X}{DOWN}{6 LEFT}V - V";               :rem 138
4302 PRINT"{2 UP}{5 SPACES}O{3 SPACES}O{DOWN}
     {6 LEFT}J{RVS}{GRN} {BLK}{OFF}K↑{Z}{RVS}{1}
     {BLK}{OFF}{X}{DOWN}{6 LEFT}V - V"                 :rem 197
```

```
4305 PRINT"{3 DOWN}MORE NATIVES APPEAR!":PRINT"
     {DOWN}RUN FOR THE SHIP!"                    :rem 37
4310 PRINT"{DOWN}TYPE {BLU}RUN{BLK} AND PRESS RETU
     RN"                                         :rem 208
4320 TI$="000000"                                 :rem 43
4330 INPUTA$:IFA$<>"RUN"THEN4330                  :rem 153
4340 IFTI<200THEN4500                             :rem 189
4350 PRINT"{DOWN}TOO SLOW. YOU'RE DEAD.":GOTO17000
                                                  :rem 117
4500 PRINT"{DOWN}WHEW! YOU SAVED YOUR SKIN BUT LOS
     T YOUR"                                      :rem 119
4505 PRINT"FOOD AND WATER":CH=CH-1                 :rem 8
4510 GOTO4810                                     :rem 208
4800 FD=10:WT=10:CH=CH+1                          :rem 96
4810 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                  :rem 80
4820 A$="":GETA$:IFA$<>CHR$(13)THEN4820           :rem 142
4840 RETURN                                       :rem 176
5000 REMSTORRM                                    :rem 144
5010 IFRND(1)>.5THENRETURN                        :rem 107
5015 POKE53281,12                                 :rem 140
5020 PRINT"{CLR}TERRIBLE STORM"                   :rem 25
5025 IFRND(1)>.9THEN PRINT"{DOWN}SHIPWRECK AND PER
     ISH":GOTO17000                               :rem 48
5030 PRINT"{DOWN}YOU RIDE IT OUT, BUT LOSE SUPPLIE
     S":PRINT"{DOWN}OVERBOARD."                   :rem 13
5040 SP=SP-4                                      :rem 173
5050 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                  :rem 77
5060 A$="":GETA$:IFA$<>CHR$(13)THEN5060           :rem 136
5070 POKE53281,3:RETURN                           :rem 119
6000 REM ARAB DHOWS                               :rem 69
6010 PRINT"{CLR}HOSTILE WATERS"                   :rem 25
6020 PRINT"{DOWN}ARAB TRADERS WILL TRY TO KEEP YOU
      OUT"                                        :rem 201
6030 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                  :rem 76
6040 A$="":GETA$:IFA$<>CHR$(13)THEN6040           :rem 134
6045 IFRND(1)>.5THENRETURN                        :rem 116
6050 PRINT"{CLR}{DOWN}";TAB(8)DH$:PRINT"{HOME}
     {DOWN}"TAB(20)DH$                            :rem 244
6060 PRINT"{HOME}{9 DOWN}ARAB DHOWS APPEAR"
                                                  :rem 157
6070 PRINT"{DOWN}TYPE {BLU}FLEE{BLK} AND PRESS RET
     URN"                                         :rem 252
6072 TI$="000000"                                 :rem 49
6074 INPUTA$:IFA$<>"FLEE"THEN6074                 :rem 206
6076 IFTI<200THEN6090                             :rem 203
6080 PRINT"{DOWN}THEY SINK YOU.":GOTO17000:rem 181
```

```
6090 PRINT"{DOWN}YOUR PILOT ESCAPES THEM.":rem 232
6100 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                              :rem 74
6110 A$="":GETA$:IFA$<>CHR$(13)THEN6110    :rem 130
6120 RETURN                                 :rem 169
10000 REM LOG BOOK                          :rem 226
10002 GOSUB16000:Q=INT(ML/200)              :rem 200
10003 IFQ>1THENPRINTLEFT$(D$,5);"{9 RIGHT}";LEFT$(
      DT$,3*Q)                              :rem 93
10005 PRINT"{HOME}{BLK}{2 SPACES}SHIP'S RECORD"
                                            :rem 215
10006 WK$=RIGHT$("{2 SPACES}"+STR$(WK),4)  :rem 204
10007 ML$=RIGHT$("{2 SPACES}"+STR$(ML),4)  :rem 187
10008 FD$=RIGHT$("{2 SPACES}"+STR$(FD),4)  :rem 158
10009 SP$=RIGHT$("{2 SPACES}"+STR$(SP),4)  :rem 209
10010 GP$=RIGHT$("{2 SPACES}"+STR$(GP),4)  :rem 177
10011 CH$=RIGHT$("{2 SPACES}"+STR$(CH),4)  :rem 154
10012 WT$=RIGHT$("{2 SPACES}"+STR$(WT),4)  :rem 219
10020 PRINTLEFT$(D$,14)TAB(20)"WEEKS OUT{4 SPACES}
      ";WK$                                 :rem 95
10030 PRINTTAB(20)"MILES SAILED ";ML$      :rem 115
10040 PRINTTAB(20)"FOOD{9 SPACES}";FD$      :rem 97
10042 PRINTTAB(20)"WATER{8 SPACES}";WT$    :rem 223
10045 PRINTTAB(20)"SUPPLIES{5 SPACES}";SP$:rem 204
10050 PRINTTAB(20)"GOLD{9 SPACES}";GP$     :rem 109
10060 PRINTTAB(20)"CREW MORALE{2 SPACES}";CH$
                                            :rem 45
10070 PRINT"{DOWN}"TAB(14)M$(G)            :rem 112
10073 IFG=1ORG=2THENPRINTTAB(14)"TOOK ON FOOD & WA
      TER";                                  :rem 9
10074 IFG=4THENPRINTTAB(14)"AT MALINDI"     :rem 99
10075 IFG=5THENPRINTTAB(10)"{DOWN}{RVS}YOU MADE IT
      !";:PRINT" {RVS}CONGRATULATIONS!{HOME}";
                                              :rem 7
10077 IFG=5THEN PRINT"HISTORY WAS WRONG.{UP}":GOTO
       17040                                :rem 31
10080 PRINTLEFT$(D$,24)"PRESS C"            :rem 52
10090 A$="":GETA$:IFA$<>"C"THEN10090       :rem 183
10095 RETURN                                :rem 223
11000 REM EVAL SITUATION                    :rem 190
11010 IFFD<1THENPRINT"{CLR}OUT OF FOOD":PRINT"
      {DOWN}YOU DIE OF STARVATION.":GOTO17000
                                            :rem 229
11015 IF WT<1THENPRINT"{CLR}OUT OF WATER":PRINT"
      {DOWN}YOU DIE OF THIRST.":GOTO17000   :rem 57
11020 IFSP<1THENPRINT"{CLR}OUT OF SUPPLIES":PRINT"
      {DOWN}YOU DIE":GOTO17000              :rem 126
11030 IFCH<2THENPRINT"{CLR}CREW MUTINIES.":PRINT"
      {DOWN}THEY FORCE YOU TO TURN BACK." :rem 202
```

```
11035 IFCH<2THEN17000                                  :rem 169
11100 RETURN                                           :rem 211
14000 POKE53281,7: REM WEATHER                         :rem 184
14002 WH=FNR(7):G=0:GG=5:CM=M%(WH)*FG                  :rem 137
14005 PRINT"{CLR}{6 DOWN}WEATHER"                      :rem 212
14010 IFWH=7THEN14140                                  :rem 193
14030 PRINTWH$(WH):IFWH<3THENCH=CH-1                   :rem 239
14034 IFML<800ANDML+CM>800THENG=1:WT=10:FD=FD+3:SP
      =SP+6:IFFD<10THENFD=10                           :rem 170
14036 IFML<1500ANDML+CM>1500THENG=2:WT=10:FD=FD+3:
      SP=SP+6:IFFD<10THENFD=10                          :rem 9
14038 IFML<5000ANDML+CM>5000THENG=3                    :rem 57
14039 IFML<6600ANDML+CM>6600THENG=4                    :rem 73
14040 IFML>6600THENGG=6                                :rem 91
14042 ML=ML+CM:Q=INT(ML/2+.5):IFML>9000THENG=5
                                                       :rem 2
14045 GOTO14155                                        :rem 55
14140 PRINT"{DOWN}STEADY RAIN":PRINT"{DOWN}YOU REF
      ILL WATER TANKS":WT=10                           :rem 46
14155 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                       :rem 131
14157 A$="":GETA$:IFA$<>CHR$(13)THEN14157              :rem 246
14200 RETURN                                           :rem 215
15000 PRINT"{2 UP}{4 RIGHT}DO YOU WANT TO READ THE
       INSTRUCTIONS?"                                  :rem 187
15001 PRINT"PRESS {RVS}Y{OFF} FOR YES, {RVS}ANY OT
      HER KEY{OFF} FOR NO."                            :rem 255
15002 WT=10:GP=20                                      :rem 177
15003 GET A$:IF A$="" GOTO 15003                        :rem 31
15004 IF A$<>"Y" GOTO 15130                             :rem 56
15010 PRINT"{CLR}MANUEL THE FIRST, KING OF PORTUGA
      L,"                                              :rem 10
15015 PRINT"{DOWN}BELIEVES THERE MUST BE A SEA-ROU
      TE TO "                                          :rem 238
15017 PRINT"{DOWN}INDIA. HE HAS OFFERED A PRIZE FO
      R"                                               :rem 137
15018 PRINT"{DOWN}FINDING IT. VASCO DA GAMA IS GOI
      NG TO"                                           :rem 117
15019 PRINT"{DOWN}TRY. HIS SHIPS WILL BE READY SOO
      N. BUT"                                           :rem 6
15020 PRINT"{DOWN}YOU HAVE A SHIP THAT CAN LEAVE T
      ODAY."                                           :rem 143
15021 PRINT"{2 DOWN}YOU DECIDE TO TRY YOUR LUCK."
                                                       :rem 213
15022 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                       :rem 125
15023 A$="":GETA$:IFA$<>CHR$(13)THEN15023              :rem 232
15025 WT=10:GP=20                                      :rem 182
15030 PRINT"{CLR}OTHERS HAVE TRIED. SOME DIED IN S
      TORMS,"                                          :rem 241
```

```
15031 PRINT"{DOWN}SOME OF THIRST OR STARVATION. SO
      ME WERE"                                  :rem 196
15032 PRINT"{DOWN}MURDERED BY PIRATES, OTHERS BY N
      ATIVES."                                  :rem 204
15033 PRINT"{DOWN}UNHAPPY CREWS MUTINIED. ARAB TRA
      DERS"                                     :rem 63
15034 PRINT"{DOWN}HAVE KILLED TO PROTECT THEIR ROU
      TES."                                     :rem 8
15035 PRINT"{DOWN}ALL THESE COULD HAPPEN TO YOU."
                                                :rem 51
15036 PRINT"{2 DOWN}{RVS}WORDS OF ADVICE{OFF}: NOT
       ALL STRANGE SHIPS"                       :rem 5
15037 PRINT"{DOWN}HOLD PIRATES. NATIVES CAN BE FRI
      ENDLY."                                   :rem 77
15038 PRINT"{DOWN}FRESH FOOD, GOOD WEATHER, AND AN
      "                                         :rem 118
15039 PRINT"{DOWN}INCREASE IN GOLD KEEPS YOUR CREW
       HAPPY."                                  :rem 147
15050 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                :rem 126
15060 A$="":GETA$:IFA$<>CHR$(13)THEN15060 :rem 234
15130 PRINT"{CLR}{DOWN}"TAB(30){2 SPACES}SS$
                                                :rem 196
15140 PRINT"{3 DOWN}PRESS {RVS}L{OFF} TO SET SAIL
      {SPACE}FROM LISBON"                       :rem 47
15150 A$="":GETA$:IFA$<>"L"THEN15150      :rem 196
15155 FORI=1TO30:PRINT"{HOME}"MS$:PRINT"{UP}"MS$
      {12 SPACES}:NEXT                          :rem 96
15160 SP=50:CH=10:FD=10                         :rem 7
15180 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                :rem 130
15185 A$="":GETA$:IFA$<>CHR$(13)THEN15185 :rem 250
15200 RETURN                                    :rem 216
16000 POKE53281,3:PRINT"{CLR}":IFML=0THEN PRINT"
      {CLR}{BLK} SEA ROUTE TO INDIA":POKE53280,3
                                                :rem 82
16004 PRINT"{BLK}RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
      RRR{YEL}"                                 :rem 130
16005 REM PRINT"{YEL}"                          :rem 79
16010 PRINTTAB(8)"{RVS}{10 SPACES}{OFF}{7 SPACES}
      {RVS}{7 SPACES}"                          :rem 223
16012 PRINTTAB(8)"{RVS}{2 SPACES}{OFF}£{3 SPACES}
      [2 J] {RVS}{19 SPACES}"                   :rem 244
16014 PRINTTAB(8)"[2 U]{3 SPACES}[C]{2 SPACES}
      {RVS}{23 SPACES}"                         :rem 107
16016 PRINTTAB(4)"[B]{3 SPACES}{RVS}£{3 SPACES}
      {OFF}{4 SPACES}{RVS}{19 SPACES}"          :rem 73
16018 PRINTTAB(7)"{RVS}£{8 SPACES}{OFF}[*]{RVS}
      {12 SPACES}{OFF}£"                        :rem 169
```

```
16020 PRINTTAB(6)"{RVS}£{9 SPACES}{OFF} {*}{RVS}
      {2 SPACES}{*}{OFF}{2 SPACES}{*}{RVS}
      {4 SPACES}{OFF}£"                          :rem 3
16022 PRINTTAB(3)"{B}{2 SPACES}{RVS}{10 SPACES}{*}
      {OFF} {*}{RVS}{2 SPACES}{*}{OFF}{2 SPACES}
      {RVS}{3 SPACES}{OFF}£"                    :rem 24
16024 PRINTTAB(6)"{RVS}{11 SPACES}{*}{OFF} {*}
      {RVS} {OFF}£{2 SPACES}{RVS}{2 SPACES}{OFF}
      £"                                        :rem 40
16026 PRINTTAB(6)"{RVS}{12 SPACES}{*}{OFF} £
      {3 SPACES}{*}{RVS} "                      :rem 75
16028 PRINTTAB(6)"{*}{RVS}{12 SPACES}{OFF}£
      {5 SPACES}{*}"                            :rem 59
16030 PRINTTAB(10)"{*}{RVS}{7 SPACES}{OFF}£"
                                                :rem 128
16032 PRINTTAB(11)"{RVS}{6 SPACES}{OFF}£"       :rem 164
16034 PRINTTAB(11)"{RVS}{6 SPACES}{OFF}"        :rem 253
16036 PRINTTAB(11)"{RVS}{6 SPACES}{OFF}"        :rem 255
16038 PRINTTAB(11)"{RVS}{6 SPACES}{OFF}"         :rem 1
16039 PRINTTAB(11)"{RVS}{5 SPACES}{OFF}£"       :rem 171
16040 PRINTTAB(11)"{RVS}{5 SPACES}{OFF} {RVS}{H}"
                                                :rem 192
16042 PRINTTAB(11)"{RVS}{4 SPACES}{OFF}£ {RVS} "
                                                :rem 183
16044 PRINTTAB(11)"{RVS}{3 SPACES}{OFF}£
      {2 SPACES}{RVS} "                         :rem 185
16046 PRINTTAB(11)"{*}{RVS} {OFF}£"             :rem 136
16048 IFML>0THENPRINT"{BLK}";:RETURN            :rem 126
16050 PRINTTAB(20)"{DOWN}{BLK}PRESS B{OFF} TO BEGI
      N";                                       :rem 156
16060 A$="":GETA$:IFA$<>"B"THEN16060            :rem 188
16070 RETURN                                    :rem 222
17000 PRINTLEFT$(D$,23) "PRESS {RED}RETURN{BLK}"
                                                :rem 123
17010 A$="":GETA$:IFA$<>CHR$(13)THEN17010       :rem 228
17020 PRINT"{CLR}{2 DOWN}ON MAY 20, 1498":PRINT"
      {DOWN}VASCO DA GAMA REACHED CALICUT ON THE"
                                                :rem 10
17025 PRINT"{DOWN}WEST COAST OF INDIA, AFTER
      {2 SPACES}"                               :rem 54
17030 PRINT"{DOWN}A VOYAGE OF 11 MONTHS AND 9500 M
      ILES.{4 DOWN}"                            :rem 70
17040 PRINT"{DOWN}PLAY AGAIN? {RVS}Y{OFF} OR {RVS}
      N{OFF}"                                   :rem 67
17050 A$="":GETA$:IFA$="Y"THENRUN               :rem 142
17060 IFA$<>"N"THEN17050                        :rem 42
17070 END                                       :rem 214
```

Quatrainment

Sean Puckett

64 Version by Gregg Peele

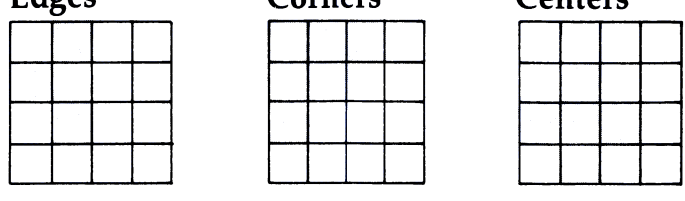
Fast thinking and logic are required for "Quatrainment," as you race the clock and plan your moves to match a master pattern. Joystick required.

"Quatrainment" pits you against the clock as you try to match a pattern generated by the computer. But this is more than a simple matching game. Depending on where you position the cursor, different groups of squares are turned off or on, creating patterns that seem even further from your goal. There is a way to duplicate the master pattern, but it takes planning and clear thinking. Not only do you have to reproduce the pattern, but you're trying to do it in the fewest moves and the shortest time possible.

Using "The Automatic Proofreader" found in Appendix C, type in Quatrainment. The Proofreader makes it easy to enter an error-free copy of the program the first time. After you've saved the program to tape or disk, load and run it.

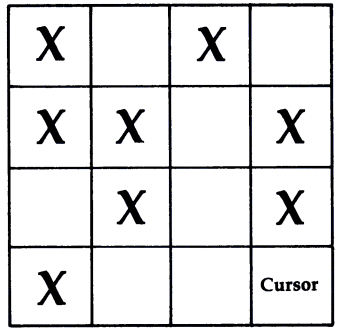
The game board is drawn on the left of the screen, and the master pattern is displayed at the right. A timer and move counter appear at the top of the screen. You'll see a cursor in one of the squares of the game board. To change the game board pattern, use the joystick plugged into port 2. Move the cursor onto the square you want, then press the joystick fire button. Part of the pattern toggles from on to off, or from off to on, depending on whether you are in the middle, in a corner, or at an edge of the board. *On* squares are shown with X's, *off* squares as blank. The different ways the pattern can change are:

Figure 1. Patterns



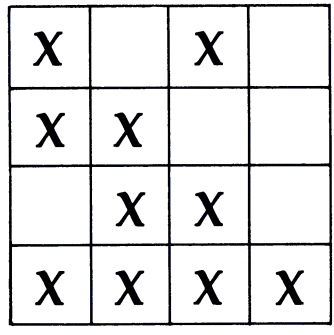
(*Centers* are defined as the four squares in the middle of the game board.) Squares that are darkened in Figure 1 will toggle on if previously off, or off if previously on. For example, if you see this on the screen:

Figure 2. Game Board—Before



and place the cursor in the lower right corner of the board, when you press the fire button, the pattern will change to:

Figure 3. Game Board—After



The two squares in the far right column that were on are now off. Three other squares (two on the bottom row, the third up and to the left of the bottom right corner) are now toggled on. You continue to turn squares on and off in this manner, trying to duplicate the master pattern on the right. Don't despair if you seem to be creating jumbled patterns; there *is* a way to re-create the master pattern.

When the game board and the master pattern match, your weighted score is displayed, based on the elapsed time and the number of moves made. Strive for the *lowest* score.

Quatrainment

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
10 PRINT"{CLR}":POKE214,10:PRINT:POKE211,13:A$="QU
   ATRAINMENT":POKE646,0                          :rem 196
20 FORT1=1TOLEN(A$):PRINTMID$(A$,T1,1);:FORT=1TO20
   0:NEXT:NEXT:FORT=1TO500:NEXT                   :rem 214
30 RN=16:REMFOR RANDOM INITIAL GRID CHANGE LINE 30
    TO RN=RND(0)*15+1                             :rem 159
40 PRINT"{CLR}";TAB(26);"TIME:"                   :rem 123
45 PRINT"{DOWN}{14 LEFT}MOVES:";MO                :rem 124
50 PRINT"{3 DOWN}{14 LEFT}EDGES {D}{B}"           :rem 210
53 PRINT"{3 DOWN}{14 LEFT}CORNERS {RVS} {OFF}{V}
   {DOWN}{2 LEFT}{V}"                             :rem 137
54 PRINT"{2 DOWN}{14 LEFT}CENTERS {UP}{D}{DOWN}
   {LEFT}{RVS}{F}{OFF}{V}"                         :rem 14
100 GOTO140                                       :rem 95
110 FORL=1TO4:FORB=1TO4:D(L,B)=PEEK(C(L,B))-9:NEXT
    B:NEXTL:RETURN                                :rem 60
115 FORT=1TO500:NEXT                              :rem 241
120 FORL=1TO4:FORB=1TO4:IFB(L,B)<>D(L,B)THENRETURN
                                                  :rem 161
130 NEXT:NEXT:SC=VAL(TI$)/16+MO/5:PRINT"{CLR}
    {6 DOWN}"TAB(10)"YOU HAVE MATCHED IT!{DOWN}"
                                                  :rem 198
135 PRINTTAB(15);"SCORE:";INT(SC);"{DOWN}":rem 134
136 PRINTTAB(11);"PLAY AGAIN Y OR N?"              :rem 77
137 IFPEEK(197)<>39ANDPEEK(197)<>25THEN137:rem 188
138 IF PEEK(197)=25THENRUN                        :rem 10
139 END                                           :rem 116
140 CO=54272:POKE53281,12:POKE53280,0             :rem 42
150 FORJ=0TO20STEP5                               :rem 170
160 FORT=1024TO1804STEP40:POKET+J,160             :rem 219
170 POKET+54272+J,1:NEXT:NEXT                      :rem 14
180 FORJ=0TO20STEP5                               :rem 173
190 FORT=1024TO1024+20:POKET+J*40,160:POKET+54272+
    J*40,1                                        :rem 159
200 NEXT:NEXT                                      :rem 74
210 POKE 1569,79:POKE1569+CO,7:POKE1577,80:POKE157
    7+CO,7                                        :rem 191
220 POKE1889,76:POKE1889+54272,7:POKE1897,122:POKE
    1897+CO,7                                     :rem 112
230 FORT=1609TO1849STEP40:POKET,116:POKE T+CO,7:PO
    KET+8,106:POKET+8+CO,7:NEXT                   :rem 240
240 FORT=1570TO1576:POKET,119:POKET+320,111:POKE T
    +CO,7:POKET+320+CO,7:NEXT                      :rem 4
250 FORJ=0TO4STEP2:FORT=1611TO1611+240STEP40:POKET
    +J,66:POKET+CO+J,7:NEXT:NEXT                  :rem 208
260 FORT=1650TO1650+6:POKET,64:POKET+CO,7:POKET+80
    ,64:POKET+80+CO,7                             :rem 50
```

```
270 POKET+160,64:POKET+160+CO,7:NEXT          :rem 140
280 FORU=1TO4:FORT=1TO4:C(T,U)=1528+2*T+80*U:NEXTT
    :NEXTU                                    :rem 13
290 DATA1106,1111,1116,1121                   :rem 83
300 FORT=1TO4:READ E:A(T,1)=E:A(T,2)=E+200:A(T,3)=
    E+400:A(T,4)=E+600:NEXT                   :rem 102
310 GOSUB570:X=1:Y=1:GOSUB500                  :rem 0
315 TI$="000000"                              :rem 251
320 JO=15-(PEEK(56320)AND15):IF JO<>1ANDJO<>2ANDJO
    <>4ANDJO<>8THEND=5:GOTO340                :rem 74
330 D=LOG(JO)/LOG(2)+1                        :rem 211
340 ONDGOSUB390,410,450,430,470               :rem 164
350 IF(PEEK(56320)AND16)THEN375               :rem 246
360 GOSUB910:MO=MO+1                          :rem 190
370 IF(PEEK(56320)AND16)=0THEN370              :rem 96
375 PRINT"{HOME}";TAB(32);RIGHT$(TI$,5);"{DOWN}
    {5 LEFT}";MO                              :rem 19
380 GOTO320                                   :rem 105
390 IFY-1<=0THEN480                            :rem 86
400 Y=Y-1:GOSUB 500:RETURN                     :rem 74
410 IFY+1=5THEN480                             :rem 22
420 Y=Y+1:GOSUB500:RETURN                      :rem 74
430 IFX+1=5THEN480                             :rem 23
440 X=X+1:GOSUB500:RETURN                      :rem 74
450 IFX-1<=0THEN480                            :rem 82
460 X=X-1:GOSUB500:RETURN                      :rem 78
470 GOSUB500:RETURN                           :rem 202
480 RETURN                                    :rem 124
490 GOTO320                                   :rem 107
500 P1=PEEK(A(X,Y)):P2=PEEK(A(X,Y)+1):P3=PEEK(A(X,
    Y)+40):P4=PEEK(A(X,Y)+41)                  :rem 80
510 POKEA(X,Y),213:POKEA(X,Y)+1,201:POKEA(X,Y)+40,
    202:POKEA(X,Y)+41,203                      :rem 57
520 POKEA(X,Y)+CO,2:POKEA(X,Y)+1+CO,2:POKEA(X,Y)+4
    0+CO,2:POKEA(X,Y)+41+CO,2                 :rem 164
530 POKEA(X,Y),P1:POKEA(X,Y)+1,P2:POKEA(X,Y)+40,P3
    :POKEA(X,Y)+41,P4                         :rem 243
535 P1=0:P2=0:P3=0:P4=0:GOSUB110:GOSUB120     :rem 162
540 RETURN                                    :rem 121
570 WE=INT(RND(0)*8)+1:FORJ=1TOWE*RN:READ Q:NEXT
                                              :rem 214
580 FORY=1TO4:FORX=1TO4:READQ:IFQ=0THEN600:rem 211
590 GOSUB610                                  :rem 181
600 NEXTX:NEXTY:GOSUB640:GOSUB680:RETURN      :rem 197
610 POKEA(X,Y),77:POKEA(X,Y)+1,78:POKEA(X,Y)+40,78
    :POKEA(X,Y)+41,77                         :rem 162
620 POKEA(X,Y)+CO,2:POKEA(X,Y)+1+CO,2:POKEA(X,Y)+4
    0+CO,2:POKEA(X,Y)+41+CO,2                 :rem 165
630 RETURN                                    :rem 121
```

```
640 FORX=1TO4:FORY=1TO4:B(X,Y)=PEEK(A(X,Y))
                                          :rem 169
650 IFB(X,Y)=32THENB(X,Y)=B(X,Y)-9:GOTO670:rem 254
660 B(X,Y)=B(X,Y)                         :rem 185
670 NEXTY:NEXTX:RETURN                     :rem 32
680 FORY=1TO4:FORX=1TO4:READP              :rem 135
690 IFPTHENPOKEC(X,Y),86:POKEC(X,Y)+CO,1   :rem 219
700 NEXTX:NEXTY:RETURN                     :rem 26
710 DATA1,1,1,1, 1,0,0,1, 1,0,0,1, 1,1,1,1 :rem 82
720 DATA0,0,0,0, 0,1,1,0, 0,1,1,0, 0,0,0,0 :rem 75
730 DATA0,1,1,0, 1,0,0,1, 1,0,0,1, 0,1,1,0 :rem 80
740 DATA1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1 :rem 89
750 DATA1,0,0,1, 0,1,1,0, 0,1,1,0, 1,0,0,1 :rem 82
760 DATA1,1,1,1, 0,0,0,0, 0,0,0,0, 1,1,1,1 :rem 83
770 DATA0,0,0,1, 0,0,0,1, 0,0,0,1, 0,0,0,1 :rem 80
775 DATA1,0,0,1, 0,0,0,0, 0,0,0,0, 1,0,0,1 :rem 85
780 DATA0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0 :rem 77
790 DATA0,0,0,0, 1,0,0,1, 1,0,0,1, 0,0,0,0 :rem 82
800 REM REVERSE                           :rem 152
810 POKEA(C,D),109-PEEK(A(C,D)):POKEA(C,D)+1,110-(
    PEEK(A(C,D)+1))                        :rem 24
820 POKEA(C,D)+40,110-PEEK(A(C,D)+40):POKEA(C,D)+4
    1,109-PEEK(A(C,D)+41)                  :rem 78
830 POKEA(C,D)+CO,2:POKEA(C,D)+1+CO,2      :rem 79
840 POKEA(C,D)+40+CO,2:POKEA(C,D)+41+CO,2:P1=0:P2=
    0:P3=0:P4=0                           :rem 185
860 RETURN                                :rem 126
870 REM SET DATA POINTER                  :rem 170
910 REM{2 SPACES}WHICH ONES{2 SPACES}TO CHANGE
                                          :rem 111
920 IFX+Y<>2THEN950                       :rem 127
930 FORC=2TO3:D=1:GOSUB810:NEXT:FORD=1TO3:C=1:GOSU
    B810:NEXT                              :rem 35
940 D=2:C=2:GOSUB810:RETURN               :rem 169
950 IF X+Y<>8THEN980                      :rem 139
960 FORC=3TO2STEP-1:D=4:GOSUB810:NEXT:FOR D=4TO2ST
    EP-1:C=4:GOSUB810:NEXT                 :rem 98
970 C=3:D=3:GOSUB810:RETURN               :rem 174
980 IF X+Y<>5THEN1020                     :rem 173
990 IF X<>4THEN 1020                       :rem 41
1000 FORC=3TO2STEP-1:D=1:GOSUB810:NEXT:FORD=1TO3:C
     =4:GOSUB810:NEXT                     :rem 229
1010 C=3:D=2:GOSUB810:RETURN              :rem 207
1020 IFX+Y<>5THEN1060                     :rem 211
1030 IFX<>1THEN1060                        :rem 76
1040 FORC=2TO3:D=4:GOSUB810:NEXT:FORD=4TO2STEP-1:C
     =1:GOSUB810:NEXT                     :rem 235
1050 C=2:D=3:GOSUB810:RETURN              :rem 211
1060 REM CHECK EDGES                      :rem 113
```

```
1070 IF(X>1ANDX<4)AND(Y=1ORY=4)THENC=X-1:D=Y:GOSUB
     810:C=X+1:GOSUB810:GOSUB1100          :rem 105
1080 IF(Y>1ANDY<4)AND(X=1ORX=4)THEND=Y-1:C=X:GOSUB
     810:D=Y+1:GOSUB810:GOSUB1100          :rem 108
1090 GOTO1160                              :rem 203
1100 IFY=1THEND=Y+1:C=X:GOSUB810           :rem 226
1110 IFY=4THEND=Y-1:C=X:GOSUB810           :rem 232
1120 IFX=4THENC=X-1:D=Y:GOSUB810           :rem 232
1130 IFX=1THENC=X+1:D=Y:GOSUB810           :rem 228
1140 RETURN                                :rem 166
1150 REM CHECK CENTERS                      :rem 29
1160 IF (X=1)OR(Y=1)OR(X=4)OR(Y=4)THEN 1200
                                           :rem 144
1170 D=Y+1:C=X:GOSUB810:C=X-1:D=Y:GOSUB810 :rem 59
1180 D=Y-1:C=X:GOSUB810:C=X+1:D=Y:GOSUB810 :rem 60
1190 C=X:D=Y:GOSUB810                       :rem 10
1200 RETURN                                :rem 163
1210 PRINT"YOU WIN "                       :rem 128
```

Campaign Manager

Todd Heimarck

Make campaign appearances, conduct polls, wage media blitzes, and take stands on issues in this national election simulation. The right strategy and tactics can lead you to the White House.

The Democratic delegates are gathered in Moscone Center, wearing straw hats, carrying balloons and signs. The floor fights are done. The time has come to nominate.

"Maryland?"

"Mister Chairman—the great state of Maryland, The Free State, home of the World Champion Baltimore Orioles, casts all of its ten votes for the senator from Arizona."

The chairman bangs his gavel. The cheers and jeers subside.

Power Broker

The convention is deadlocked. You control a large block of undecided delegates. It's all up to you.

The vice president from Rhode Island has good charisma and intelligence, but you know his health is not the best. The reverend from Arkansas may look good on television, but he's too conservative. And though the senator from Arizona is experienced, he's not very bright. Perhaps the New Jersey doctor? No, you decide that the senator from Ohio is best; your party will also get the home region advantage in the populous Heartland.

Now it's the Republicans' turn. Of the five choices, the woman from South Carolina is the best all-around candidate. She has high charisma and fundraising appeal, which translates into effective television ads.

It's time to hit the campaign trail.

Nine Weeks of Campaigning

The Democratic candidate starts with nine million dollars and 59 health points. He rests two days (to build up some health),

followed by two days of fundraising. Campaign stops in Illinois and Texas sway the voters slightly to the Democratic side.

The Republican campaigns in her home state of South Carolina. She then moves on to North Carolina, Virginia, and Florida, followed by a couple of days of recuperation.

As the campaign progresses, the Democrat concentrates on personal appearances in the industrial Northeast, plus forays into the larger states of Texas, California, Florida, and others. The Republican candidate does less actual campaigning, preferring to spend more time on fundraising to pay for expensive television ads.

In the crucial eighth week, both candidates rest and fundraise in preparation for last-minute campaigning. The Democrat does a media blitz in the Pacific, Southern, and Atlantic states. The Republican hits the Heartland, Arklatex, and the urban Northeast.

Election Night

Initial returns from New England show that the Republicans swept the region, but the larger states of New York and Pennsylvania went Democratic. The Republicans won most states from Ohio to the Great Plains, but the Democrats picked up the Southern Atlantic states, except for Florida. Texas went with the G.O.P., while the rest of the region went Democratic. The Rocky Mountain states were solidly Republican. But the Democrats won the Pacific States.

The final results show the Republicans winning six of nine regions, capturing the presidency with 315 electoral votes to the Democrats' 223. Three of the four biggest states voted Democratic, but Ohio and Illinois, with 47 electoral votes between them, made the difference. The TV ads in the last week edged those two key states into the Republican camp.

It's Long—and Well Worth It

"Campaign Manager" is a two-player national-election simulation written completely in machine language (ML). Make the right decisions and your candidate will be elected. Make mistakes and you'll have to wait another four years.

This game is long. No doubt about it. It has to be to include the multiple options and to make it as realistic and true-to-life as it is. It's fast because it's an ML program. Consisting of a long stream of numbers, ML programs used to be difficult

to type in. We've solved that problem with MLX, the "Machine Language Editor." By using MLX, you can almost guarantee an error-free program the first time you type in Campaign Manager. Make sure you read Appendix D, "Using the Machine Language Editor: MLX" and have a copy of the MLX program saved on tape or disk *before* you begin entering Campaign Manager. It will still take time, but MLX will help immensely.

Before loading MLX, you have to protect part of BASIC memory by typing the following line in direct mode (with no line numbers):

POKE 642,40:SYS 58260

You'll see the usual cold start message, but there will be less than the normal 39K of free memory. You can now load MLX and begin typing in Campaign Manager.

MLX will ask for two addresses. They are:

Starting address: 2049
Ending address: 9518

The program uses about 10K, although it was crunched down to about 7K. Since it's such a long program, you'll probably enter it in parts. If you do, make sure you follow the MLX instructions for loading and saving. *Type in the POKE 642,40:SYS 58260 before loading MLX at the start of each session.* MLX also has a numeric keypad option, which should save you some time.

When you are finished typing in Campaign Manager, save it to tape or disk, perhaps even making a couple of backup copies as well. Turn your 64 off, then on again, load the program as if it were a BASIC program, (LOAD"*filename*",8 for disk or LOAD"*filename*" for tape) and type RUN. The first few bytes look like a BASIC program that says SYS 2061. You don't have to remember the SYS; it's built into the program.

Managing the Campaign

You have nine weeks to campaign. Each week you have to plan your moves and enter them on the itinerary. You have two defensive moves, resting and fundraising. And you have two ways to gain votes, campaigning (personal appearances) and advertising on television.

At the beginning of each turn, you'll see a medium-resolution map of the U.S. that indicates which way each state is leaning. The *MAP* option allows you to move a cursor

around the country, to identify which states are which. (See explantion below for joystick and keyboard movement instructions.) If the Republicans are ahead in that state, it's colored red. Democratic states are cyan, or light blue. If you're using a black-and-white TV for your display, the Republican states show in a darker shade.

Since you have only 63 days (9 weeks of 7 days), you have enough time to campaign in each state at least once. In terms of electoral votes, however, California with 47 votes is 16 times more important than some of the smaller 3-vote states like South Dakota or Vermont. Generally, it makes more sense to campaign heavily in the ten biggest states, which some people call *megastates:*

State	Electoral Votes
California	47
New York	36
Texas	29
Pennsylvania	25
Illinois	24
Ohio	23
Florida	21
Michigan	20
New Jersey	16
North Carolina	13

Winning the election requires 270 electoral votes out of a possible 538. The ten megastates account for 254 votes, just 16 shy of a majority.

At the beginning of the campaign, each state has a large pool of undecided voters. As the game progresses, these voters make up their minds and the pool diminishes.

Each state also has a built-in bias towards one party, based on past elections for president, senator, governor, and so on. The District of Columbia, for example, is staunchly Democratic. A Democratic candidate will automatically get seven campaign points there, compared to a Republican's two. Wisconsin leans toward the Republicans, but just barely; Republicans get five campaign points to four for the Democrats.

Since the Republicans have won three of the last four elections, including the landslide victory in 1972, you might expect them to begin the game with a huge advantage. But if you look at nonpresidential elections, you'll find a lot of states that elect Democratic governors, senators, and representatives and then vote for a Republican president. And a lot of those basically Democratic states were split by third-party campaigns, such as Wallace in '68 or Anderson in '80.

To even things up, and make the game more playable, the Democrats begin with an electoral vote advantage of 282 to 256, although 4 of the megastates (Pennsylvania, Ohio, Florida, and North Carolina) are just barely leaning to the Democratic side. The Republicans have the advantage of beginning with 29 of the 51 states. (Since the District of Columbia has 3 electoral votes, it counts as a state in the game.) Most of the states west of the Mississippi are Republican, while the Democrats have most of the industrial Northeast and the South.

In addition to the natural political leanings, each state has its own stands on five general issues: unemployment/inflation, poverty/crime, agriculture, education, and defense. A very urban state might be conservative on crime, but not care much about agriculture, for example. Each candidate also has certain stands on these issues. When you campaign or advertise in a state, you can get up to 3 extra campaign points for each issue, *if* your stand agrees with the voters'.

Finally, the candidate you choose has a campaign-effectiveness rating, based on charisma and intelligence. The campaign-effectiveness factor adds campaign points each time you campaign.

Choosing a Candidate

At the beginning of Campaign Manager, you choose which party will go first, and decide if one candidate will be the incumbent. You might want to flip a coin, the winner choosing either a party or whether to play first or second. In testing, we found that the second player has the slight advantage of making the last move. Being incumbent gives you some extra campaigning strength and is not recommended if you want an even game.

Note that all choices are made with a joystick. (It's easier to play if each player has a joystick, although one can be

shared. It doesn't matter which port it's plugged into.) Move the pointer with your joystick and press the firebutton *twice* to make your choice. If you don't have a joystick, you can use the keyboard instead; I, J, K, and L are up, left, down, and right respectively. Press M in place of the fire button.

Players then pick the candidate who will represent their party. Five randomly chosen candidates are available. To the right of the candidate's stats is the YES/NO counter. Before making your choice, say NO to each possibility, until you've seen all five. After selecting a candidate, enter the candidate's name from the keyboard.

The heart of the game is the actual campaign, but in some ways the convention is just as important. Nominate a terrible candidate and you'll spend most of your campaign trying to catch up.

Healthy, Wealthy, and Wise

A candidate's personality greatly affects the outcome of the election. In the lower left-hand corner you should see a list of five attributes, each associated with a number from one (worst) to eight (best). With a couple of exceptions, the ideal candidate is the one with straight eights.

First is *charisma* (CHAR), which is personal magnetism, panache, the ability to influence and excite people. This is the most important personality trait, because it is part of both campaign effectiveness and advertising effectiveness.

Stamina (STAM) affects your health. A candidate with low stamina will have to rest frequently, to regain health and strength.

Intelligence (INTL) adds points to campaign effectiveness and last-minute campaigning.

Experience (EXPR) helps you with fundraising. If your candidate has lots of experience, he or she has more contacts and connections for raising money. Since experience comes with age, it does count against your health.

Appeal (APPL) also contributes to fundraising attempts. But if you have maximum appeal (eight) you may be tainted by your affiliations with special-interest groups, and there's a backlash when you advertise. It's best to have an appeal of six or seven.

The candidate's attributes are generated by adding three random numbers, so it's more likely to get a middle number (four or five) than one of the extremes.

The personality traits translate into these five campaign factors:

Campaign Effectiveness (CHAR*2 + INTL)
Strength/Health (STAM*4+9 − EXPR)
Fundraising Appeal (EXPR*3 + APPL)
TV Ads (APPL OR 8 + CHAR)
Last-Minute Campaigning (INTL + STAM)

The significance of each factor is explained below in the section "Getting Votes."

Here I Stand

Next to the personality factors are the candidate's stands on various issues. You see five issues, each with a sliding scale of one (at the far left, representing liberal) to six (conservative). A Republican who wants to get tough on crime, for example, will have a rank of six. A Democrat who wants to solve the unemployment problem will have a rating of one in that category.

Candidates will range from two to five on the issues of agriculture and education. On the other three issues, the Democrats will have stands from one to four; the Republicans will go from three to six.

You will generally get more votes with middle-of-the-road-beliefs. Look for a candidate with twos or threes if you're the Democrat. Fours and fives are best for the Republican. The exceptions are agriculture and education, where you do best with a three or a four.

Common sense tells you which issues are important in most states. Agriculture is a major issue in the farming states. Your stand on defense makes a difference in states with a lot of military-related industry.

The candidate's personality is generally more crucial than the stands on issues. If you have a lot of charisma, intelligence, and appeal, it doesn't matter that you may be radical on one or two issues.

If you have five very bad candidates, hit RUN/ STOP–RESTORE, type RUN, and try again. It's not much fun to run a campaign you're destined to lose.

Beginning the Campaign

After the nominees have been chosen, the first week begins. You may notice that some states have changed colors. That's because each nominee gets the equivalent of campaigning once in each state. If one is much more charismatic, or happens to hit the right issues, a state may jump over to his or her side. In addition, each candidate gets a home state and home region advantage.

You should develop a strategy. If your appeal and charisma are strong, you might want to concentrate on television ads. If your candidate has a strong anti-crime stance, visit the more urban states. At the very least, you should plan to visit each of the megastates.

You begin in your home state; it's traditional to campaign there once. And the first week usually means some fundraising and resting, as purely defensive moves.

The Doctor, Treasurer, Pollster, and Pilot

Under the week's itinerary should be two numbers. At the beginning of each week, your treasurer tells you how much money you have, up to about $25 million. Your personal physician figures out how healthy you are. At most you'll have 255 health points.

If you fall below four million dollars any time during the week, television advertising will be useless. If you have less than one million, you won't be able to pay the pollster (you won't even see the bar graph to the left of the U.S. map). When your bank account falls to zero, the campaign is paralyzed until you sponsor a fundraiser. You can't even afford to pay your doctor or staff.

It takes time away from campaigning, but you have to raise money once in a while. Each fundraising point (EXPR*3 + APPL) is worth $200,000.

Campaigning is exhausting, and even though it takes time, once in a while you have to rest. When you decide to catch some Z's, the itinerary will be filled with (you guessed it) Z's. Each day of rest gains double your strength factor, plus campaign effectiveness, plus the number of states you are winning. A high campaign effectiveness gives you optimism; you rest better. If you're losing, you worry about it and toss and turn. Resting two days in a row gets you 16 extra health points.

There are two reasons to keep your health up. First, when you campaign in a state, you get an extra campaign point for every 32 health points you possess. Second, if your health falls below eight you look haggard and stutter; campaigning does you no good.

The treasurer counts dollars, the doctor counts your health, and your pollster counts votes.

The pollster does three things. First, you get a bar chart which shows how many electoral votes would go to the Democrats and Republicans if the election were to be held today. You can see it to the left of the map. The gray bar marked *U* represents undecided states too close to call. Second, you have a map of the U.S. to show you, at a glance, which way each state is leaning. Republican states are red, Democratic states are blue. These first two services are part of the standard pollster's contract, and cost you nothing. Of course, if your money drops lower than one million, you have to stop paying the pollster; all you get is the map.

The third service is the most important—regional polls. To get a poll of all states in a region, move the cursor to POLL (below MAP on the main menu) and press the fire button twice. You'll see a bar chart showing which way each state in the region is leaning, from one (half a character wide) to four (two characters). The poll reflects the political situation at the beginning of the week; whatever campaigning you have planned for the week is not included. A state with a thin bar is just barely inclined toward one side or the other and can usually be taken with a single campaign stop.

Don't use polls in the first couple of weeks—most states start out fairly even and you will not learn much. But polling can be vital towards the end of the game. If New York is firmly committed to you, forget about further efforts in that state. And if you find a whole region weakly supporting your opponent, you can hit it with TV ads and score a few dozen electoral votes.

Regional polls cost $100,000 and are not available if you begin the week with less than a million dollars.

The final character in your entourage is the jet pilot. Your jet can carry you on short hops within a region for almost nothing. But if you travel to a new region, you shell out $100,000 for fuel and maintenance. As long as you're in a region, you might as well stay there a few days, to avoid spend-

ing a lot on travel expenses. Again, you don't actually move to a new region until you have campaigned in one of the states. You can use the travel option to conduct regional polls; you'll pay $100,000 for the poll and another $100,000 if you decide to campaign in a region. Don't campaign and you won't be charged for travel.

Guests and Fish

Benjamin Franklin once said that after three days, guests and fish begin to smell. The same principle applies to campaigning.

Campaign once and you gain some votes. Stay for a second day and the voters of a state are flattered; you gain a couple of bonus votes. But stick around for a third or fourth day and you've overstayed your welcome.

Getting Votes

Each state begins with 255 undecided voter points. Your main goal is to use campaigning and television advertising to move a few points from undecided to firmly committed to you. And you have to maintain your health and money.

The effects of a personal appearance can vary. You get up to 3 points for each issue (if the state agrees with your stance), 1 point for every 32 health points, and up to 24 for your campaign effectiveness (CHAR*2 + INTL), plus a 2-point bonus if it's the second day you've been in the state.

If your money is down to 0, you get no campaign points at all. If your health is below 8, you get only a single vote.

Each campaign stop decreases your health and money. And it's possible to run out in the middle of the week, making each succeeding visit ineffective until you rest or raise money. Let's say you go to Connecticut and impress 23 of the 255 undecideds. The pool of available voters is reduced by that number. Half of 23 (11 points, in this case, due to rounding off the decimal) is charged against your health. Half again (5 times $100,000, or $500,000) is subtracted from your money. In addition, each state has some people who don't agree with you, so a quarter of your total (five points) goes to your opponent as a sort of backlash. If you had previously been in a different region, travel expenses of $100,000 are subtracted.

Television advertising is a little different. It affects every state in the region and quickly swings voters to your side. To advertise, first travel to the region and, before doing any ads,

make at least one campaign stop to establish your presence there. After campaigning once, you can advertise as many times as you like.

Unlike resting and campaigning, the effects of advertising are not cumulative from day to day. If you advertise two days in a row, you don't get bonus points. Advertising does grow in strength from week to week, however, and will be more effective towards the end of the campaign.

Because you flood the region with ads, it's possible to bring a whole section of the country to your side. But it *is* expensive. In each state, advertising credits you with half your campaign effectiveness, half your TV ads' effectiveness rating, points for issues, plus two times the week number (in week seven, for example, you get fourteen extra campaign points).

The cost is the usual one fourth of campaign points gained, plus double the TV ads' effectiveness. The large regions can cost a lot. Going on TV in the Atlantic States (nine) or in the Rocky Mountain States (eight) can easily deplete your treasury.

On the day you plan to advertise, you must have at least four million dollars. If you don't, you waste the day and gather no new votes. So, if you begin the week with five million dollars and campaign in six states, it's likely that by Saturday there'll be less than four million remaining. Your ad campaign will be useless.

There's one more item you can choose: RECONSIDER. If you make a mistake, this option will cancel all your choices and you can start the week anew. Your itinerary is not permanent until you fill out the seventh day and press the fire button (if you pull on the joystick instead, it's the same as reconsidering).

Here's a summary of the commands on the main menu:

- CAMPAIGN—allows you to make a personal appearance in one of the states in the region you're in. Gains votes, costs health and money.
- TV ADS—floods the region with advertising. Reduces health and costs a lot of money, but can quickly deliver a big chunk of votes. Doesn't work if you have less than four million dollars.
- FUNDRAIS—raises money for your campaign. Takes a day, gains no votes, costs nothing.

- REST—builds up your health points. Gains no new votes, costs nothing.
- MAP—moves the cursor around the map, prints the state name, electoral votes, and region number. For information only, costs nothing.
- POLL—provides a bar graph showing which way the states in the region are leaning. Costs $100,000 immediately. Not available if money falls below a million.
- RECONSIDER—erases the week's itinerary if you make a mistake.
- TRAVEL—takes you to a new region of the country. Costs $100,000, which isn't charged to you until you actually campaign there.

The Home Stretch

The ninth week is usually the most hectic. If you sponsored some fundraisers in week eight, you'll want to spend a lot on television advertising in the regions you have a chance in. Polls can tell you which states are most vulnerable.

After both candidates have finished their last week of campaigning, a couple of things happen. The last region to be visited by a candidate gives a few extra votes to him or her. And the last-week routine goes into action as all the undecided voters make up their minds. The candidates get their last-minute campaigning points (INTL + STAM) added to each state in the country. The undecided voters are split between the candidates, and ties are resolved based on the built-in bias to one party or the other.

The map is drawn for the final time. The final bar chart appears to the left. This should indicate at a glance which candidate won. Beginning with Region 1 (New England), the electoral votes are displayed, with region totals underneath.

The winner is the candidate with the most electoral votes. There is a slight chance that there will be a tie, in which case you'd have to flip a coin.

After the votes are counted, you can play *what-if?* A close game generally brings comments like, "I should have campaigned a little more in Texas or California or New York (or whatever state you thought you had won)."

Game Etiquette

There are a few traditional rules of Campaign Manager etiquette:

First, since the joystick routine reads both joysticks, try to avoid interfering with your opponent's choices. Put down your joystick when it's not your turn.

Second, when you have filled out your itinerary and the prompt PRESS FIREBUTTON TO CONTINUE comes up, don't press the firebutton. Let your opponent study what moves you made. Then he or she can press the firebutton.

Third, since polls cost money, they should be kept private. When the other player is doing a poll, look away from the screen.

On to the White House

Campaign Manager is a true simulation. It accurately represents different candidates' strengths and weaknesses, campaign strategies and tactics, and allows for that bane of all politicians, a bit of luck. You'll find the game entertaining, but highly educational as well. We all can't run for the presidency. But with Campaign Manager, you can at least see how you *would* have done, given the chance.

Campaign Manager

For easy entry of this machine language program, be sure to read "Using the Machine Language Editor: MLX," Appendix D. Also make sure to enter POKE 642,40:SYS 58260 in direct mode before loading MLX in preparing to type in this program.

```
2049 :011,008,010,000,158,050,238
2055 :048,054,049,000,000,000,158
2061 :032,110,012,032,241,012,196
2067 :032,122,017,032,108,031,105
2073 :069,250,204,204,204,204,136
2079 :220,192,000,000,000,005,192
2085 :229,255,167,255,255,255,173
2091 :255,178,030,128,000,000,122
2097 :219,095,250,031,255,255,130
2103 :255,255,143,045,000,004,245
2109 :245,037,255,255,031,255,115
2115 :255,255,241,197,250,076,061
2121 :255,248,095,095,255,255,252
2127 :255,255,143,191,175,245,063
2133 :255,115,037,245,255,255,223
2139 :255,252,204,254,250,247,017
2145 :035,076,032,015,247,255,245
2151 :255,255,255,250,254,162,254
2157 :250,047,018,000,095,021,028
2163 :255,255,227,255,092,252,171
2169 :204,060,204,000,000,127,204
```

```
2175 :175,255,255,255,250,255,036
2181 :204,060,207,176,000,001,013
2187 :242,255,255,191,255,239,040
2193 :175,250,247,224,000,000,017
2199 :000,001,051,127,255,255,072
2205 :242,255,255,240,000,000,125
2211 :079,160,128,000,119,255,136
2217 :047,225,035,127,000,000,091
2223 :013,255,000,096,000,007,034
2229 :176,000,000,000,126,000,227
2235 :000,211,058,000,112,000,056
2241 :002,000,000,000,001,250,190
2247 :000,016,000,160,000,000,119
2253 :000,000,000,000,000,001,206
2259 :032,000,000,000,000,000,243
2265 :032,227,008,032,041,009,054
2271 :032,078,009,096,173,014,113
2277 :220,041,254,141,014,220,095
2283 :165,001,041,251,133,001,059
2289 :169,209,133,252,169,057,206
2295 :133,254,160,000,132,251,153
2301 :132,253,177,251,145,253,184
2307 :136,208,249,198,252,198,220
2313 :254,169,055,197,254,208,122
2319 :239,165,001,009,004,133,054
2325 :001,173,014,220,009,001,183
2331 :141,014,220,173,024,208,039
2337 :041,240,009,014,141,024,246
2343 :208,096,169,057,133,252,186
2349 :133,254,169,080,133,251,041
2355 :169,208,133,253,032,068,146
2361 :009,169,024,133,251,169,044
2367 :216,133,253,198,254,160,253
2373 :039,177,251,145,253,136,046
2379 :016,249,096,169,255,141,233
2385 :003,056,169,240,141,002,180
2391 :056,169,015,141,001,056,013
2397 :162,000,142,000,056,134,075
2403 :251,138,032,117,009,138,016
2409 :032,114,009,232,224,016,220
2415 :208,243,096,234,074,074,016
2421 :041,003,168,185,000,056,058
2427 :160,003,145,251,136,016,066
2433 :251,230,251,230,251,230,036
2439 :251,230,251,096,169,054,162
2445 :133,252,169,000,133,251,055
2451 :168,170,224,188,208,001,082
2457 :096,189,025,008,072,074,105
2463 :056,106,074,074,145,251,097
```

```
2469 :032,181,009,104,041,015,035
2475 :009,032,145,251,032,181,053
2481 :009,232,208,224,201,032,059
2487 :208,004,009,192,145,251,224
2493 :200,192,025,240,001,096,175
2499 :169,000,145,251,168,024,184
2505 :169,026,101,251,133,251,108
2511 :144,002,230,252,096,012,175
2517 :001,003,002,014,160,004,141
2523 :185,212,009,153,032,208,250
2529 :136,016,247,173,017,208,254
2535 :009,064,141,017,208,096,254
2541 :032,247,009,032,110,010,165
2547 :032,185,010,096,169,147,114
2553 :032,210,255,160,003,032,173
2559 :087,010,169,144,032,210,139
2565 :255,169,171,032,210,255,073
2571 :169,163,032,101,010,169,143
2577 :167,032,210,255,162,015,090
2583 :160,003,032,082,010,169,223
2589 :170,032,210,255,169,154,251
2595 :032,210,255,169,160,032,125
2601 :101,010,169,144,032,210,195
2607 :255,169,165,032,210,255,109
2613 :202,208,223,160,003,032,113
2619 :082,010,169,174,032,210,224
2625 :255,169,172,032,101,010,036
2631 :169,173,032,210,255,169,055
2637 :146,032,210,255,096,169,217
2643 :013,032,210,255,169,032,026
2649 :032,210,255,136,208,250,156
2655 :169,018,032,210,255,096,107
2661 :160,025,032,210,255,136,151
2667 :208,250,096,169,004,133,199
2673 :254,169,044,133,253,169,111
2679 :054,133,252,169,000,133,092
2685 :251,169,000,168,162,015,122
2691 :177,251,208,007,032,160,198
2697 :010,202,208,246,096,145,020
2703 :253,200,208,240,041,063,124
2709 :170,189,192,055,041,192,220
2715 :017,247,145,247,096,024,163
2721 :169,026,101,251,133,251,068
2727 :144,002,230,252,169,040,236
2733 :024,101,253,133,253,144,057
2739 :002,230,254,160,000,096,153
2745 :169,015,133,249,169,216,112
2751 :133,254,169,044,133,253,153
2757 :133,247,169,004,133,248,107
```

```
2763 :169,034,133,252,169,173,109
2769 :133,251,160,024,177,251,181
2775 :201,000,240,043,133,002,066
2781 :041,063,170,189,192,055,163
2787 :041,015,145,253,169,192,018
2793 :036,002,240,025,048,008,080
2799 :189,120,034,032,147,010,003
2805 :208,015,080,007,169,192,148
2811 :032,155,010,208,006,189,083
2817 :121,034,032,147,010,234,067
2823 :136,016,203,169,025,024,068
2829 :101,251,133,251,144,002,127
2835 :230,252,198,249,208,001,133
2841 :096,169,040,024,101,247,190
2847 :133,247,144,002,230,248,011
2853 :169,040,024,101,253,133,245
2859 :253,144,165,230,254,208,017
2865 :161,173,018,208,072,101,014
2871 :162,074,074,074,168,104,199
2877 :229,162,074,141,032,208,139
2883 :140,036,208,096,031,067,133
2889 :065,077,080,065,073,071,248
2895 :078,032,077,065,078,065,218
2901 :071,069,082,013,000,162,226
2907 :018,160,008,024,032,240,061
2913 :255,162,000,189,071,011,017
2919 :240,006,032,210,255,232,054
2925 :208,245,160,005,169,001,129
2931 :141,134,002,169,018,032,099
2937 :210,255,162,040,173,134,071
2943 :002,073,003,141,134,002,226
2949 :169,163,032,210,255,202,140
2955 :208,250,136,208,235,169,065
2961 :146,076,210,255,169,146,123
2967 :133,254,169,000,133,253,069
2973 :162,000,232,236,137,036,192
2979 :240,047,189,137,036,133,177
2985 :249,041,007,133,247,165,243
2991 :249,074,074,074,074,041,249
2997 :007,133,248,160,002,032,251
3003 :230,011,165,247,160,001,233
3009 :032,230,011,169,255,160,026
3015 :005,145,253,169,005,024,032
3021 :101,253,133,253,076,159,156
3027 :011,169,000,170,168,185,146
3033 :068,034,157,000,120,232,060
3039 :232,200,192,052,208,243,070
3045 :096,145,253,200,200,145,244
3051 :253,096,169,145,133,248,255
```

```
3057 :169,000,133,247,230,247,243
3063 :133,254,170,162,000,189,131
3069 :189,036,133,249,074,074,240
3075 :074,074,133,250,189,240,195
3081 :036,133,251,074,074,133,198
3087 :252,074,074,133,253,160,193
3093 :004,162,004,181,249,072,181
3099 :041,003,024,105,001,145,090
3105 :247,104,074,074,041,003,064
3111 :024,105,003,010,010,010,201
3117 :010,017,247,145,247,136,079
3123 :202,016,226,230,247,160,108
3129 :002,169,015,049,247,170,197
3135 :232,138,010,010,010,010,217
3141 :133,002,138,005,002,145,238
3147 :247,136,208,235,230,247,098
3153 :230,247,230,247,230,247,232
3159 :230,254,166,254,224,051,242
3165 :208,157,096,169,255,141,095
3171 :015,212,169,128,141,018,014
3177 :212,141,024,212,096,162,184
3183 :064,169,000,157,000,143,132
3189 :157,064,143,202,208,247,114
3195 :169,128,141,138,002,169,102
3201 :008,032,210,255,032,149,047
3207 :011,032,250,026,032,108,082
3213 :027,032,128,023,032,139,010
3219 :009,032,030,028,032,217,239
3225 :008,032,237,011,032,217,178
3231 :009,032,237,009,169,158,005
3237 :032,210,255,032,090,011,027
3243 :032,030,020,032,050,011,090
3249 :032,026,031,173,107,031,065
3255 :240,245,032,217,009,032,190
3261 :096,012,162,004,160,005,116
3267 :032,163,028,141,021,143,211
3273 :141,035,037,162,007,160,231
3279 :009,032,163,028,162,000,089
3285 :160,000,201,000,240,007,053
3291 :041,001,240,002,202,200,137
3297 :136,142,015,143,140,079,112
3303 :143,032,046,017,208,003,168
3309 :076,157,012,096,169,000,235
3315 :141,036,037,169,128,133,119
3321 :247,169,143,133,248,169,078
3327 :005,133,002,160,005,162,210
3333 :003,173,027,212,041,003,208
3339 :149,249,202,208,246,169,210
3345 :001,037,250,024,105,001,179
```

```
3351 :101,251,101,252,145,247,096
3357 :136,208,228,160,006,173,172
3363 :027,212,041,003,170,192,168
3369 :008,240,010,192,009,240,228
3375 :006,173,021,143,240,002,120
3381 :232,232,232,138,145,247,255
3387 :200,192,011,208,226,173,045
3393 :027,212,041,063,240,249,129
3399 :201,052,176,245,145,247,113
3405 :200,173,015,143,145,247,232
3411 :208,009,173,027,212,041,241
3417 :007,010,010,145,247,032,028
3423 :220,014,198,002,208,157,126
3429 :160,000,140,045,017,169,120
3435 :128,133,247,169,143,133,036
3441 :248,173,045,017,201,005,034
3447 :176,236,170,240,006,032,211
3453 :220,014,202,208,250,238,233
3459 :045,017,160,005,177,247,014
3465 :153,015,143,136,208,248,016
3471 :160,006,162,000,177,247,127
3477 :157,027,143,200,232,224,108
3483 :005,208,245,177,247,141,154
3489 :012,143,141,010,143,200,042
3495 :177,247,141,013,143,032,152
3501 :228,014,032,238,014,208,139
3507 :003,076,106,013,032,046,199
3513 :017,240,169,162,000,134,139
3519 :248,160,006,024,032,240,133
3525 :255,173,021,143,205,035,005
3531 :037,240,002,162,012,134,022
3537 :247,189,158,020,240,006,045
3543 :032,210,255,232,208,245,117
3549 :169,063,032,210,255,166,092
3555 :247,160,010,169,044,157,246
3561 :158,020,232,136,208,249,212
3567 :032,228,255,240,251,201,166
3573 :013,240,039,201,032,240,242
3579 :008,201,065,144,239,201,085
3585 :091,176,235,230,248,166,123
3591 :248,224,011,240,019,164,145
3597 :247,153,158,020,041,063,183
3603 :157,005,004,230,247,169,063
3609 :047,157,006,004,208,208,143
3615 :032,038,015,032,046,017,211
3621 :240,149,173,015,143,041,030
3627 :002,024,109,016,143,010,091
3633 :109,018,143,141,022,143,113
3639 :173,017,143,010,010,105,001
```

```
3645 :009,056,237,019,143,141,154
3651 :023,143,173,027,212,041,174
3657 :031,010,109,023,143,105,238
3663 :032,141,008,143,173,015,079
3669 :143,041,004,109,019,143,032
3675 :010,109,019,143,109,020,245
3681 :143,141,024,143,010,109,155
3687 :018,143,105,048,141,009,055
3693 :143,173,020,143,009,008,093
3699 :109,016,143,141,025,143,180
3705 :173,015,143,041,007,024,012
3711 :109,018,143,109,017,143,154
3717 :141,026,143,162,000,173,010
3723 :012,143,232,221,127,036,142
3729 :176,250,142,032,143,142,006
3735 :011,143,142,033,143,032,143
3741 :132,027,173,021,143,205,090
3747 :035,037,240,003,076,241,027
3753 :012,173,015,143,041,003,044
3759 :141,129,143,032,243,027,122
3765 :169,000,141,129,143,174,169
3771 :033,143,189,127,036,168,115
3777 :202,189,127,036,170,202,095
3783 :032,247,027,032,132,027,184
3789 :173,021,143,205,035,037,051
3795 :208,213,032,250,026,032,204
3801 :108,027,096,169,016,024,145
3807 :101,247,133,247,096,032,055
3813 :237,009,032,205,021,032,253
3819 :038,015,096,169,015,133,189
3825 :253,169,022,133,254,169,217
3831 :029,133,167,162,240,160,114
3837 :016,032,184,020,173,021,187
3843 :143,240,013,162,010,189,248
3849 :117,020,041,063,157,156,051
3855 :006,202,208,245,173,021,102
3861 :143,205,035,037,240,003,172
3867 :238,125,006,162,020,160,226
3873 :021,032,163,028,096,174,035
3879 :021,143,189,040,037,032,245
3885 :210,255,169,017,133,253,058
3891 :169,025,133,254,169,000,033
3897 :133,167,162,081,160,016,008
3903 :032,184,020,169,031,032,019
3909 :210,255,169,020,133,253,085
3915 :169,025,133,254,169,009,066
3921 :133,167,162,171,160,016,122
3927 :032,184,020,162,019,232,224
3933 :160,015,024,032,240,255,051
```

```
3939 :162,049,138,032,210,255,177
3945 :232,224,055,208,247,056,103
3951 :032,240,255,224,024,208,070
3957 :230,173,012,143,010,170,087
3963 :189,220,033,041,063,141,042
3969 :171,006,232,189,220,033,212
3975 :041,063,141,172,006,162,208
3981 :018,160,002,024,032,240,105
3987 :255,174,021,143,189,040,201
3993 :037,032,210,255,174,013,106
3999 :143,048,014,160,004,189,205
4005 :049,016,032,210,255,232,191
4011 :136,208,246,240,013,162,152
4017 :000,189,228,016,240,006,088
4023 :032,210,255,232,208,245,085
4029 :169,158,133,247,169,020,061
4035 :133,248,160,000,173,021,162
4041 :143,205,035,037,240,002,095
4047 :160,012,177,247,240,006,025
4053 :032,210,255,200,208,246,084
4059 :173,021,143,240,032,162,222
4065 :010,189,117,020,041,063,153
4071 :157,248,006,202,208,245,017
4077 :169,020,133,253,169,022,235
4083 :133,254,169,009,133,167,084
4089 :162,210,160,016,032,184,245
4095 :020,162,004,160,160,189,182
4101 :016,143,009,048,153,039,157
4107 :007,152,056,233,040,168,155
4113 :202,016,240,162,004,160,033
4119 :160,152,024,125,027,143,142
4125 :168,185,046,007,009,064,252
4131 :153,046,007,152,056,233,170
4137 :040,041,248,168,202,016,244
4143 :232,096,083,069,078,032,125
4149 :071,079,086,032,082,069,216
4155 :080,032,082,069,086,032,184
4161 :032,077,083,032,068,082,183
4167 :062,032,086,061,080,032,168
4173 :071,069,078,032,027,044,142
4179 :000,027,044,000,255,044,197
4185 :068,069,077,079,067,082,019
4191 :065,084,073,067,032,067,227
4197 :065,078,068,073,068,065,006
4203 :084,069,032,044,044,044,168
4209 :044,044,044,000,255,156,144
4215 :047,032,067,072,065,082,228
4221 :032,088,000,255,047,032,067
4227 :083,084,065,077,032,088,048
```

```
4233 :000,255,047,032,073,078,110
4239 :084,076,032,088,000,255,166
4245 :047,032,069,088,080,082,035
4251 :032,088,000,255,047,032,097
4257 :065,080,080,076,032,088,070
4263 :000,000,000,000,255,031,197
4269 :085,078,069,077,080,000,050
4275 :255,080,079,086,084,089,084
4281 :000,255,065,071,082,073,219
4287 :067,000,255,069,068,085,223
4293 :067,078,000,255,068,070,223
4299 :069,078,083,000,000,000,177
4305 :000,255,031,073,078,070,204
4311 :076,078,000,255,067,082,005
4317 :073,077,069,000,000,000,184
4323 :000,157,080,082,069,083,186
4329 :073,068,069,078,084,032,125
4335 :000,255,151,080,076,065,098
4341 :089,069,082,032,091,049,145
4347 :000,255,068,069,077,079,031
4353 :067,082,065,084,073,067,183
4359 :000,255,067,079,078,086,060
4365 :069,078,084,073,079,078,218
4371 :000,010,166,000,005,032,232
4377 :000,255,030,032,047,032,165
4383 :078,079,000,255,032,047,010
4389 :032,089,069,083,000,000,054
4395 :000,000,000,173,005,004,225
4401 :072,169,000,133,162,133,206
4407 :198,169,032,197,162,208,253
4413 :252,162,023,189,098,017,034
4419 :041,063,157,004,004,202,026
4425 :016,245,032,026,031,173,084
4431 :107,031,240,248,162,023,122
4437 :104,157,004,004,202,016,060
4443 :250,173,107,031,041,016,197
4449 :096,058,070,073,082,069,033
4455 :066,085,084,084,079,078,067
4461 :032,084,079,032,067,079,226
4467 :078,084,073,078,085,069,070
4473 :058,173,035,037,205,021,138
4479 :143,208,011,238,036,037,032
4485 :173,036,037,201,010,208,030
4491 :001,096,032,237,009,032,034
4497 :205,021,032,038,015,169,113
4503 :007,141,000,143,032,244,206
4509 :020,162,005,160,012,032,036
4515 :163,028,170,208,003,076,043
4521 :003,018,202,208,003,076,167
```

```
4527 :147,018,202,208,003,076,061
4533 :197,018,202,208,003,076,117
4539 :239,018,202,208,006,032,124
4545 :043,029,076,155,017,202,203
4551 :208,008,032,022,019,208,184
4557 :205,076,003,018,202,208,149
4563 :014,032,046,017,240,197,245
4569 :173,011,143,141,032,143,092
4575 :076,141,017,202,240,003,134
4581 :076,155,017,076,200,019,004
4587 :162,000,169,128,024,109,059
4593 :032,143,168,169,000,133,118
4599 :253,169,014,133,254,169,215
4605 :030,133,167,076,184,020,095
4611 :032,235,017,174,032,143,124
4617 :189,127,036,202,056,253,104
4623 :127,036,072,105,003,168,014
4629 :162,003,032,163,028,201,098
4635 :000,208,007,032,043,029,090
4641 :104,076,006,018,133,002,116
4647 :104,197,002,176,003,076,085
4653 :155,017,198,002,174,032,111
4659 :143,202,189,127,036,024,004
4665 :101,002,174,000,143,157,122
4671 :000,143,133,251,134,252,208
4677 :032,250,019,169,030,032,089
4683 :210,255,165,251,010,170,112
4689 :189,220,033,032,210,255,252
4695 :189,221,033,032,210,255,003
4701 :169,032,032,210,255,189,212
4707 :000,120,072,170,169,000,118
4713 :032,205,189,104,201,010,078
4719 :176,005,169,032,032,210,223
4725 :255,169,032,032,210,255,046
4731 :169,152,032,210,255,173,090
4737 :032,143,009,048,032,210,091
4743 :255,206,000,143,208,003,182
4749 :076,007,020,076,006,018,088
4755 :174,000,143,169,240,157,006
4761 :000,143,134,252,032,250,196
4767 :019,169,129,032,210,255,205
4773 :162,000,189,112,021,240,121
4779 :006,032,210,255,232,208,090
4785 :245,173,032,143,009,048,059
4791 :032,210,255,206,000,143,005
4797 :208,003,076,007,020,076,067
4803 :155,017,174,000,143,169,085
4809 :255,157,000,143,134,252,118
4815 :032,250,019,169,154,032,095
```

```
4821 :210,255,162,000,189,125,130
4827 :021,240,006,032,210,255,215
4833 :232,208,245,206,000,143,235
4839 :208,003,076,007,020,076,109
4845 :155,017,174,000,143,169,127
4851 :000,157,000,143,134,252,161
4857 :032,250,019,169,155,032,138
4863 :210,255,169,090,162,005,122
4869 :032,210,255,202,208,250,138
4875 :206,000,143,208,003,076,135
4881 :007,020,076,155,017,173,209
4887 :009,143,201,010,176,001,051
4893 :096,206,009,143,032,153,156
4899 :033,174,032,143,189,127,221
4905 :036,133,248,202,189,127,208
4911 :036,133,247,169,150,133,147
4917 :249,169,004,133,250,169,003
4923 :047,133,251,133,252,166,017
4929 :247,228,248,208,003,076,051
4935 :046,017,165,249,024,105,165
4941 :040,133,249,144,002,230,107
4947 :250,189,000,144,133,253,028
4953 :133,254,162,004,006,254,134
4959 :202,208,251,006,254,176,168
4965 :028,169,037,133,251,006,213
4971 :254,176,020,169,032,133,123
4977 :251,006,254,176,012,169,213
4983 :037,133,252,006,254,176,209
4989 :004,169,032,133,252,160,107
4995 :000,169,032,145,249,200,158
5001 :165,251,145,249,200,165,032
5007 :252,145,249,169,047,133,114
5013 :251,133,252,006,253,176,196
5019 :028,169,042,133,252,006,017
5025 :253,176,020,169,032,133,176
5031 :252,006,253,176,012,169,011
5037 :042,133,251,006,253,176,010
5043 :004,169,032,133,251,160,160
5049 :007,165,251,145,249,200,178
5055 :165,252,145,249,230,247,199
5061 :076,058,019,032,103,023,252
5067 :174,032,143,232,232,232,224
5073 :160,031,024,032,240,255,183
5079 :169,058,032,210,255,162,077
5085 :003,160,013,032,163,028,108
5091 :201,000,208,006,032,043,205
5097 :029,076,200,019,201,010,000
5103 :208,003,076,155,017,141,071
5109 :032,143,076,155,017,169,069
```

```
5115 :022,056,229,252,170,160,116
5121 :032,024,032,240,255,096,168
5127 :032,046,017,208,003,076,133
5133 :217,017,032,104,025,032,184
5139 :250,026,032,108,027,032,238
5145 :132,027,076,122,017,169,056
5151 :000,133,253,169,010,133,217
5157 :254,169,030,133,167,162,184
5163 :049,160,020,076,184,020,040
5169 :255,018,144,160,213,211,026
5175 :197,160,202,207,217,189,203
5181 :160,000,255,160,211,212,035
5187 :201,195,203,160,207,210,219
5193 :160,000,255,201,202,203,070
5199 :204,146,205,018,160,203,247
5205 :197,217,211,000,255,018,215
5211 :155,080,076,091,049,032,062
5217 :080,065,082,084,089,146,131
5223 :000,255,031,068,069,077,091
5229 :079,067,082,065,084,073,047
5235 :067,000,255,082,069,080,156
5241 :085,066,076,073,067,065,041
5247 :078,000,255,018,155,032,153
5253 :073,078,067,085,077,066,067
5259 :069,078,084,146,000,255,003
5265 :031,032,032,032,078,079,173
5271 :078,069,032,032,032,000,138
5277 :255,032,080,076,065,089,242
5283 :069,082,032,049,032,000,171
5289 :255,032,080,076,065,089,254
5295 :069,082,032,050,032,000,184
5301 :000,000,000,134,251,132,186
5307 :252,208,011,200,152,024,010
5313 :101,251,133,251,144,002,051
5319 :230,252,166,253,228,254,046
5325 :208,001,096,230,253,164,133
5331 :167,024,032,240,255,160,065
5337 :000,162,255,177,251,016,054
5343 :016,200,177,251,240,217,044
5349 :032,210,255,202,016,250,170
5355 :240,209,200,208,241,170,223
5361 :200,208,237,169,000,133,164
5367 :253,169,014,133,254,169,215
5373 :146,032,210,255,169,144,185
5379 :032,210,255,169,030,133,064
5385 :167,162,072,160,021,032,111
5391 :184,020,174,021,143,189,234
5397 :037,037,041,063,141,071,155
5403 :004,173,036,037,009,048,078
```

```
5409 :141,078,004,173,032,143,092
5415 :010,010,010,024,109,032,234
5421 :143,170,173,032,143,009,203
5427 :048,141,150,004,160,000,042
5433 :189,037,036,041,063,240,151
5439 :007,153,152,004,232,200,043
5445 :208,242,096,009,058,000,170
5451 :255,032,032,032,087,069,070
5457 :069,075,032,032,032,000,065
5463 :009,058,000,009,032,000,195
5469 :009,032,000,255,031,032,196
5475 :032,067,065,077,080,065,229
5481 :073,071,078,000,255,032,102
5487 :032,084,086,032,065,068,222
5493 :083,032,032,157,000,255,164
5499 :032,032,070,085,078,068,232
5505 :082,065,073,083,000,255,175
5511 :032,032,082,069,083,084,005
5517 :032,032,032,032,000,255,012
5523 :018,155,032,077,065,080,062
5529 :032,032,032,032,032,032,089
5535 :000,255,032,080,079,076,169
5541 :076,032,032,032,032,032,145
5547 :000,255,146,150,082,069,105
5553 :067,079,078,083,073,068,113
5559 :069,082,000,255,084,082,243
5565 :065,086,069,076,032,032,037
5571 :032,032,154,000,009,032,198
5577 :000,000,000,000,169,014,128
5583 :133,253,169,025,133,254,150
5589 :169,028,133,167,169,030,141
5595 :032,210,255,162,034,160,048
5601 :023,032,184,020,169,043,184
5607 :141,076,006,169,046,141,042
5613 :140,007,173,036,037,208,070
5619 :011,169,020,162,008,032,133
5625 :210,255,202,208,250,096,190
5631 :174,008,143,169,000,032,013
5637 :205,189,162,023,160,030,006
5643 :024,032,240,255,169,030,249
5649 :032,210,255,174,009,143,072
5655 :224,100,176,022,169,032,234
5661 :032,210,255,224,010,176,168
5667 :013,032,210,255,032,210,019
5673 :255,138,009,048,032,210,221
5679 :255,096,169,000,032,205,036
5685 :189,173,184,007,141,185,164
5691 :007,162,006,173,027,212,134
5697 :041,015,201,010,176,247,243
```

```
5703 :009,048,157,185,007,202,167
5709 :208,239,169,060,141,184,054
5715 :007,141,188,007,032,122,068
5721 :022,169,052,133,248,169,114
5727 :000,133,247,168,162,002,039
5733 :149,252,202,016,251,032,235
5739 :166,022,169,032,162,002,148
5745 :149,249,202,016,251,032,244
5751 :211,022,096,169,017,133,255
5757 :252,169,000,133,251,166,072
5763 :251,228,252,208,001,096,143
5769 :160,000,024,032,240,255,080
5775 :162,000,189,040,037,032,091
5781 :210,255,169,037,032,210,038
5787 :255,232,224,003,208,240,037
5793 :230,251,076,130,022,166,012
5799 :247,232,232,134,247,200,179
5805 :196,248,208,001,096,189,087
5811 :000,120,074,133,002,185,181
5817 :000,144,162,002,041,238,004
5823 :240,006,202,041,014,240,166
5829 :001,202,181,252,024,101,190
5835 :002,176,216,149,252,076,050
5841 :166,022,169,004,133,248,183
5847 :169,000,133,247,160,002,158
5853 :169,015,133,002,185,037,250
5859 :037,041,063,145,247,136,128
5865 :016,246,169,040,024,101,061
5871 :247,133,247,169,000,101,112
5877 :248,133,248,160,002,162,174
5883 :002,169,016,024,117,252,063
5889 :149,252,176,013,181,249,253
5895 :145,247,202,136,016,239,224
5901 :198,002,016,218,096,072,103
5907 :169,037,149,249,104,074,033
5913 :074,074,041,001,009,036,004
5919 :076,007,023,011,035,000,183
5925 :255,042,032,083,000,255,192
5931 :042,032,077,000,255,042,235
5937 :032,084,000,255,042,032,238
5943 :087,000,255,042,032,084,043
5949 :000,255,042,032,070,000,204
5955 :255,042,032,083,000,011,234
5961 :044,000,255,092,032,048,032
5967 :048,060,048,048,048,060,135
5973 :048,048,048,000,255,154,126
5979 :032,072,069,065,076,084,233
5985 :072,032,000,000,000,000,201
5991 :169,028,032,210,255,169,198
```

```
5997 :000,133,253,169,015,133,044
6003 :254,169,030,133,167,162,006
6009 :000,160,128,032,184,020,133
6015 :096,169,128,133,248,169,046
6021 :000,133,247,169,000,133,047
6027 :250,133,249,169,000,133,049
6033 :253,133,254,168,162,001,092
6039 :032,241,023,162,000,160,001
6045 :004,189,037,036,145,247,047
6051 :200,232,224,008,208,245,000
6057 :162,008,172,054,025,136,214
6063 :230,253,165,253,201,010,007
6069 :208,003,076,225,023,169,117
6075 :255,145,247,200,169,028,207
6081 :145,247,200,169,042,145,117
6087 :247,200,165,253,009,048,097
6093 :145,247,200,169,031,145,118
6099 :247,200,232,189,037,036,128
6105 :145,247,208,247,200,076,060
6111 :175,023,032,002,024,169,136
6117 :000,162,004,145,247,200,219
6123 :202,208,250,076,017,024,244
6129 :162,001,160,000,189,054,039
6135 :025,145,247,200,232,236,052
6141 :054,025,208,244,096,162,018
6147 :001,189,089,025,145,247,187
6153 :200,232,236,089,025,208,231
6159 :244,096,169,001,133,253,143
6165 :133,254,208,009,230,253,084
6171 :165,253,201,010,208,001,097
6177 :096,230,248,169,009,024,041
6183 :101,249,133,249,169,000,172
6189 :101,250,133,250,032,241,028
6195 :023,166,249,160,004,165,050
6201 :253,073,048,145,247,200,255
6207 :200,189,037,036,240,006,003
6213 :145,247,232,200,208,245,066
6219 :166,253,189,127,036,133,211
6225 :250,172,054,025,136,165,115
6231 :254,010,170,169,048,133,103
6237 :251,133,252,169,255,145,018
6243 :247,200,169,028,145,247,111
6249 :200,169,042,145,247,200,084
6255 :169,154,145,247,200,165,167
6261 :254,201,010,144,007,230,195
6267 :252,233,010,076,118,024,068
6273 :101,251,133,251,165,252,002
6279 :145,247,200,165,251,145,008
6285 :247,200,169,032,145,247,157
```

```
6291 :200,169,151,145,247,200,235
6297 :189,220,033,145,247,200,163
6303 :232,189,220,033,145,247,201
6309 :200,202,169,032,145,247,136
6315 :200,169,048,133,251,133,081
6321 :252,189,000,120,201,010,181
6327 :144,007,230,252,233,010,035
6333 :076,181,024,101,251,133,187
6339 :251,165,252,145,247,200,175
6345 :165,251,145,247,200,169,098
6351 :032,145,247,200,169,000,232
6357 :145,247,200,230,254,165,174
6363 :254,197,250,240,003,076,215
6369 :086,024,032,002,024,165,046
6375 :250,133,254,166,253,202,209
6381 :189,127,036,133,002,232,188
6387 :189,127,036,056,229,002,114
6393 :133,002,169,008,229,002,024
6399 :133,002,048,038,169,009,142
6405 :145,247,200,169,035,145,178
6411 :247,200,169,000,145,247,251
6417 :200,198,002,048,019,169,141
6423 :009,145,247,200,169,032,057
6429 :145,247,200,169,000,145,167
6435 :247,200,198,002,016,237,167
6441 :169,000,162,004,145,247,000
6447 :200,202,208,250,076,025,240
6453 :024,035,009,035,000,255,155
6459 :032,032,032,032,032,032,251
6465 :032,032,032,032,000,009,202
6471 :044,000,255,018,154,037,067
6477 :144,205,193,208,160,160,123
6483 :160,160,160,160,146,000,101
6489 :014,255,028,042,077,069,062
6495 :078,085,032,032,032,032,130
6501 :032,000,000,173,011,143,204
6507 :141,032,143,169,008,141,229
6513 :000,143,206,000,143,208,045
6519 :001,096,174,000,143,189,210
6525 :000,143,208,009,032,177,182
6531 :026,032,208,026,076,115,102
6537 :025,016,023,106,176,003,230
6543 :076,166,027,173,024,143,240
6549 :010,109,009,143,144,002,054
6555 :169,255,141,009,143,076,180
6561 :115,025,172,009,143,240,097
6567 :203,072,162,000,232,221,033
6573 :127,036,176,250,236,011,241
6579 :143,240,009,142,032,143,120
```

```
6585 :142,011,143,206,009,143,071
6591 :104,032,207,025,032,233,056
6597 :025,032,093,026,032,140,033
6603 :026,076,115,025,133,002,068
6609 :133,251,198,251,165,251,178
6615 :010,010,024,101,251,133,232
6621 :251,133,253,169,146,133,026
6627 :252,169,145,133,254,096,252
6633 :173,008,143,041,248,208,030
6639 :005,169,001,133,255,096,130
6645 :169,003,024,109,021,143,202
6651 :168,177,251,133,255,173,128
6657 :010,143,016,003,230,255,146
6663 :096,197,002,208,009,169,176
6669 :002,032,087,026,169,255,072
6675 :133,002,165,002,141,010,216
6681 :143,173,008,143,160,005,145
6687 :074,136,208,252,032,087,052
6693 :026,173,022,143,032,087,008
6699 :026,160,006,136,208,001,068
6705 :096,185,026,143,209,253,193
6711 :208,007,169,003,032,087,049
6717 :026,208,238,170,202,138,019
6723 :209,253,208,007,169,001,146
6729 :032,087,026,208,224,232,114
6735 :232,138,209,253,208,217,056
6741 :240,240,024,101,255,133,054
6747 :255,096,160,005,177,251,011
6753 :056,229,255,176,004,198,247
6759 :255,208,243,145,251,165,090
6765 :255,170,172,021,143,200,046
6771 :024,113,251,144,002,169,050
6777 :255,145,251,152,073,003,232
6783 :168,138,074,074,113,251,177
6789 :144,002,169,255,145,251,075
6795 :096,070,255,208,001,096,097
6801 :173,008,143,056,229,255,241
6807 :176,002,169,000,141,008,135
6813 :143,070,255,208,001,096,162
6819 :173,009,143,056,229,255,004
6825 :176,002,169,000,141,009,154
6831 :143,096,160,000,162,015,239
6837 :173,021,143,240,002,162,154
6843 :240,134,251,162,052,202,204
6849 :208,003,132,002,096,189,055
6855 :000,144,037,251,240,243,090
6861 :200,208,240,165,002,024,020
6867 :109,023,143,010,109,022,115
6873 :143,109,008,143,144,003,255
6879 :024,169,255,141,008,143,195
```

```
6885 :173,010,143,208,010,169,174
6891 :016,109,008,143,176,003,178
6897 :141,008,143,169,000,141,075
6903 :010,143,096,169,146,133,176
6909 :252,169,000,133,251,169,203
6915 :000,170,240,007,160,005,073
6921 :230,251,136,208,251,232,037
6927 :224,052,208,001,096,160,244
6933 :001,177,251,200,056,241,179
6939 :251,208,006,032,088,027,127
6945 :076,007,027,176,010,234,051
6951 :073,255,024,105,001,160,145
6957 :128,208,002,160,008,133,172
6963 :253,132,254,041,224,240,171
6969 :002,208,020,070,254,165,008
6975 :253,041,016,240,002,208,055
6981 :010,070,254,165,253,041,094
6987 :008,208,002,070,254,165,014
6993 :254,157,000,144,076,007,207
6999 :027,173,000,144,041,240,200
7005 :240,004,169,001,208,002,205
7011 :169,016,141,000,144,157,214
7017 :000,144,096,162,052,202,249
7023 :240,018,189,000,144,041,231
7029 :015,240,004,169,067,208,052
7035 :002,169,130,157,192,055,060
7041 :208,235,096,173,021,143,237
7047 :072,162,063,189,064,143,060
7053 :157,128,143,189,000,143,133
7059 :157,064,143,189,128,143,203
7065 :157,000,143,202,208,235,074
7071 :104,073,001,141,021,143,130
7077 :096,173,009,143,201,040,059
7083 :144,067,174,032,143,189,152
7089 :127,036,133,250,202,189,090
7095 :127,036,133,249,198,249,151
7101 :173,036,037,010,024,109,066
7107 :025,143,133,255,230,249,206
7113 :165,249,197,250,240,014,036
7119 :032,023,028,070,255,032,135
7125 :093,026,032,140,026,076,094
7131 :189,027,173,009,143,056,048
7137 :237,025,143,144,005,237,248
7143 :025,143,176,002,169,001,235
7149 :141,009,143,076,115,025,234
7155 :162,000,160,052,134,249,232
7161 :132,250,230,249,165,249,244
7167 :197,250,240,019,174,129,240
7173 :143,134,255,032,023,028,108
```

```
7179 :032,044,026,070,255,032,214
7185 :093,026,076,251,027,096,074
7191 :032,207,025,032,038,026,127
7197 :096,162,000,169,000,157,101
7203 :000,063,202,208,250,169,159
7209 :000,170,168,185,010,031,093
7215 :157,000,063,185,018,031,245
7221 :157,064,063,232,232,232,009
7227 :200,192,007,208,236,185,063
7233 :010,031,157,000,063,157,227
7239 :001,063,157,002,063,185,030
7245 :018,031,157,064,063,169,067
7251 :252,141,248,007,169,253,129
7257 :141,249,007,162,007,169,056
7263 :012,157,039,208,202,016,217
7269 :250,169,001,141,029,208,131
7275 :169,001,141,016,208,169,043
7281 :004,141,000,208,169,050,173
7287 :141,001,208,169,054,141,065
7293 :002,208,169,056,141,003,192
7299 :208,169,000,160,004,153,057
7305 :002,031,136,016,250,169,229
7311 :034,141,007,031,169,173,186
7317 :141,006,031,169,054,141,179
7323 :009,031,169,000,141,008,001
7329 :031,096,169,000,133,253,075
7335 :169,004,141,000,208,152,073
7341 :032,250,030,133,252,138,240
7347 :032,250,030,133,251,141,248
7353 :001,208,169,012,141,039,243
7359 :208,173,016,208,009,001,038
7365 :141,016,208,173,021,208,196
7371 :009,001,141,021,208,032,103
7377 :026,031,173,107,031,240,049
7383 :248,041,019,240,244,170,153
7389 :041,016,208,039,138,041,192
7395 :001,240,017,173,001,208,099
7401 :197,251,240,227,198,253,063
7407 :056,233,008,141,001,208,118
7413 :208,217,173,001,208,197,225
7419 :252,240,210,230,253,024,180
7425 :105,008,141,001,208,208,160
7431 :200,169,000,141,039,208,252
7437 :032,026,031,173,107,031,157
7443 :240,248,041,016,208,007,011
7449 :169,012,141,039,208,208,034
7455 :176,173,021,208,041,254,136
7461 :141,021,208,165,253,096,153
7467 :162,007,189,002,031,149,071
```

```
7473 :247,202,016,248,169,001,164
7479 :141,040,208,173,021,208,078
7485 :009,002,141,021,208,032,218
7491 :026,031,173,107,031,240,163
7497 :248,106,176,020,106,176,137
7503 :067,106,176,110,106,176,052
7509 :005,106,176,005,144,231,240
7515 :076,240,029,076,231,030,005
7521 :165,248,240,221,173,003,123
7527 :208,056,233,004,141,003,236
7533 :208,198,248,165,248,106,002
7539 :176,003,076,036,030,165,089
7545 :253,233,026,133,253,176,171
7551 :002,198,254,165,251,056,029
7557 :233,025,133,251,144,003,154
7563 :076,036,030,198,252,076,039
7569 :036,030,165,248,201,029,086
7575 :240,169,173,003,208,024,200
7581 :105,004,141,003,208,230,080
7587 :248,165,248,106,176,123,205
7593 :165,253,105,026,133,253,080
7599 :144,002,230,254,165,251,197
7605 :024,105,025,133,251,144,095
7611 :104,230,252,076,036,030,147
7617 :165,247,208,003,076,066,190
7623 :029,173,002,208,056,233,132
7629 :004,141,002,208,198,247,237
7635 :165,247,106,144,076,165,090
7641 :253,233,001,133,253,176,242
7647 :002,198,254,165,251,056,125
7653 :233,001,133,251,176,057,056
7659 :198,252,076,036,030,165,224
7665 :247,201,049,208,003,076,001
7671 :066,029,173,002,208,024,237
7677 :105,004,141,002,208,230,175
7683 :247,165,247,106,144,003,147
7689 :076,036,030,165,253,105,162
7695 :001,133,253,144,002,230,010
7701 :254,165,251,024,105,001,053
7707 :133,251,144,005,230,252,018
7713 :076,036,030,169,001,133,222
7719 :249,165,248,074,144,004,155
7725 :006,249,006,249,165,247,199
7731 :106,176,002,006,249,160,238
7737 :000,177,251,133,002,165,017
7743 :249,049,253,208,038,169,005
7749 :192,036,002,048,013,165,013
7755 :002,041,063,170,189,120,148
7761 :034,133,002,076,106,030,206
```

```
7767 :080,007,169,000,133,002,222
7773 :076,106,030,165,002,041,001
7779 :063,170,189,121,034,133,041
7785 :002,162,015,160,016,024,228
7791 :032,240,255,169,149,032,220
7797 :210,255,169,032,162,007,184
7803 :032,210,255,202,016,250,064
7809 :169,157,162,007,032,210,098
7815 :255,202,016,250,165,002,001
7821 :208,003,076,066,029,041,052
7827 :063,010,170,189,220,033,064
7833 :032,210,255,189,221,033,069
7839 :032,210,255,169,032,032,121
7845 :210,255,189,000,120,170,085
7851 :201,010,176,005,169,032,252
7857 :032,210,255,169,000,032,107
7863 :205,189,169,029,032,210,249
7869 :255,169,144,032,210,255,230
7875 :169,018,032,210,255,169,024
7881 :160,032,210,255,165,002,001
7887 :041,063,162,000,232,221,158
7893 :127,036,176,250,138,105,021
7899 :176,032,210,255,169,146,183
7905 :032,210,255,076,066,029,125
7911 :173,021,208,041,253,141,044
7917 :021,208,162,007,181,247,039
7923 :157,002,031,202,016,248,131
7929 :096,234,010,010,010,024,121
7935 :105,050,096,000,000,000,250
7941 :000,000,000,000,000,192,197
7947 :192,224,240,224,192,200,003
7953 :255,255,153,129,195,195,175
7959 :129,153,255,169,000,141,102
7965 :107,031,173,000,220,041,089
7971 :031,073,031,208,045,173,084
7977 :001,220,041,031,073,031,182
7983 :208,036,032,228,255,208,246
7989 :001,096,056,233,073,144,144
7995 :222,170,232,233,005,176,073
8001 :216,138,041,002,240,004,194
8007 :138,073,001,170,169,000,110
8013 :141,107,031,056,042,202,144
8019 :208,252,141,107,031,173,227
8025 :000,220,045,001,220,041,104
8031 :016,240,246,169,006,101,105
8037 :162,197,162,208,252,096,154
8043 :000,032,250,026,032,177,112
8049 :026,165,002,201,026,144,165
8055 :003,032,132,027,032,122,211
```

```
8061 :033,032,132,027,032,122,247
8067 :033,169,001,032,207,025,086
8073 :160,005,177,251,074,074,110
8079 :170,160,002,138,024,113,238
8085 :251,144,002,165,255,145,087
8091 :251,136,208,243,160,002,131
8097 :209,251,208,019,160,003,243
8103 :177,251,200,056,241,251,063
8109 :169,128,042,168,200,177,033
8115 :251,233,001,145,251,165,201
8121 :251,024,105,005,133,251,186
8127 :201,255,208,198,032,250,055
8133 :026,032,177,026,162,051,159
8139 :189,000,144,041,017,240,066
8145 :003,030,000,144,202,208,028
8151 :243,032,108,027,032,237,126
8157 :009,032,087,022,032,090,237
8163 :011,032,205,021,162,015,161
8169 :134,002,160,029,024,032,102
8175 :240,255,169,152,032,210,017
8181 :255,169,032,162,011,032,138
8187 :210,255,202,208,250,230,070
8193 :002,166,002,224,024,208,115
8199 :227,173,100,007,141,140,027
8205 :007,141,180,007,141,220,197
8211 :007,169,032,162,011,157,045
8217 :220,007,202,208,250,169,057
8223 :020,141,226,007,169,000,082
8229 :162,003,149,003,202,016,060
8235 :251,169,009,133,174,169,180
8241 :000,141,032,143,238,032,123
8247 :143,173,032,143,201,010,245
8253 :208,003,076,048,032,032,204
8259 :153,033,169,000,133,178,221
8265 :133,179,162,004,134,251,168
8271 :160,031,132,252,169,190,245
8277 :133,247,133,249,169,004,252
8283 :133,248,133,250,166,167,164
8289 :160,003,169,032,145,247,085
8295 :136,016,251,165,247,024,174
8301 :105,040,133,247,144,002,012
8307 :230,248,202,208,233,174,130
8313 :032,143,189,127,036,133,013
8319 :254,202,189,127,036,133,044
8325 :253,166,251,164,252,024,219
8331 :032,240,255,166,253,189,250
8337 :000,144,041,015,208,003,044
8343 :076,111,033,189,068,034,150
8349 :170,024,101,178,133,178,173
```

```
8355 :138,201,010,176,005,169,094
8361 :032,032,210,255,169,154,253
8367 :032,210,255,169,000,032,105
8373 :205,189,166,251,160,037,165
8379 :024,032,240,255,160,003,133
8385 :169,032,032,210,255,136,003
8391 :208,250,230,251,230,253,085
8397 :198,167,208,181,165,174,018
8403 :208,003,076,105,033,173,041
8409 :032,143,024,105,014,170,193
8415 :160,031,024,032,240,255,197
8421 :169,154,032,210,255,165,190
8427 :178,170,201,010,176,005,207
8433 :169,032,032,210,255,169,084
8439 :000,032,205,189,169,156,230
8445 :032,210,255,169,032,072,255
8451 :032,210,255,173,032,143,080
8457 :009,048,032,210,255,104,155
8463 :032,210,255,032,210,255,241
8469 :165,179,170,201,010,176,154
8475 :005,169,032,032,210,255,218
8481 :169,028,032,210,255,169,128
8487 :000,032,205,189,162,024,139
8493 :160,030,024,032,240,255,018
8499 :169,152,032,210,255,165,010
8505 :178,024,101,003,133,003,243
8511 :169,000,101,004,133,004,218
8517 :165,179,101,005,133,005,145
8523 :169,000,101,006,133,006,234
8529 :166,003,165,004,032,205,144
8535 :189,162,024,160,036,024,170
8541 :032,240,255,166,005,165,188
8547 :006,032,205,189,198,174,135
8553 :032,046,017,076,053,032,105
8559 :189,068,034,024,101,179,194
8565 :133,179,076,201,032,173,143
8571 :026,143,141,129,143,032,225
8577 :243,027,169,000,141,129,070
8583 :143,174,032,143,189,127,175
8589 :036,168,202,189,127,036,131
8595 :170,202,032,247,027,096,153
8601 :169,156,032,210,255,032,239
8607 :235,017,162,003,160,030,254
8613 :024,032,240,255,032,193,173
8619 :033,174,032,143,189,127,101
8625 :036,202,056,253,127,036,119
8631 :133,167,105,003,170,160,153
8637 :030,032,240,255,162,000,140
8643 :189,207,033,208,001,096,161
```

```
8649 :032,210,255,232,208,244,102
8655 :154,068,069,077,032,032,127
8661 :032,032,028,082,069,080,024
8667 :000,032,032,077,069,078,251
8673 :072,086,084,077,065,082,179
8679 :073,067,084,078,089,078,188
8685 :074,080,065,079,072,073,168
8691 :078,073,076,077,073,087,195
8697 :073,077,078,073,065,077,180
8703 :079,078,068,083,068,078,197
8709 :069,075,083,068,069,077,190
8715 :068,068,067,086,065,087,196
8721 :086,078,067,083,067,071,213
8727 :065,070,076,075,089,084,226
8733 :078,065,076,077,083,065,217
8739 :082,076,065,079,075,084,240
8745 :088,077,084,073,068,087,006
8751 :089,067,079,078,077,065,246
8757 :090,085,084,078,086,087,051
8763 :065,079,082,067,065,065,226
8769 :075,072,073,000,004,004,037
8775 :003,013,004,008,036,016,151
8781 :025,023,012,024,020,011,192
8787 :010,008,011,003,003,005,123
8793 :007,003,010,003,012,006,130
8799 :013,008,012,021,009,011,169
8805 :009,007,006,010,008,029,170
8811 :004,004,003,008,005,007,138
8817 :005,004,010,007,047,003,189
8823 :004,000,001,003,003,004,134
8829 :005,008,009,008,010,026,191
8835 :031,011,014,016,014,012,229
8841 :037,012,019,016,017,020,002
8847 :025,009,032,025,025,029,032
8853 :027,030,026,031,029,032,068
8859 :017,034,038,035,040,040,103
8865 :041,042,038,037,046,049,158
8871 :040,040,044,000,000,000,035
8877 :239,239,047,111,231,231,247
8883 :231,231,231,210,210,210,222
8889 :207,207,207,000,000,000,038
8895 :000,000,000,000,000,193,128
8901 :193,239,047,047,111,103,169
8907 :039,039,039,039,018,018,139
8913 :018,015,079,079,077,205,170
8919 :205,000,000,000,000,000,164
8925 :193,193,240,048,048,112,031
8931 :103,039,039,039,039,019,249
8937 :019,019,015,015,079,014,138
```

```
8943 :206,205,000,000,000,199,081
8949 :007,066,193,240,048,048,079
8955 :040,040,103,041,041,041,045
8961 :019,019,019,016,016,078,168
8967 :140,205,013,205,201,199,202
8973 :007,007,004,196,241,049,005
8979 :110,046,046,045,041,041,092
8985 :041,020,020,020,084,016,226
8991 :080,012,076,011,010,073,037
8997 :009,009,071,070,197,241,122
9003 :049,110,046,046,045,045,128
9009 :042,042,042,149,149,149,110
9015 :017,145,012,076,011,138,198
9021 :074,137,088,151,200,000,199
9027 :000,049,049,110,046,045,110
9033 :045,042,042,042,021,021,030
9039 :021,085,017,145,076,075,242
9045 :031,095,090,025,087,214,115
9051 :000,000,241,049,110,172,151
9057 :044,044,043,043,171,101,031
9063 :037,037,081,099,017,096,214
9069 :096,096,089,091,091,091,151
9075 :000,000,000,000,241,049,149
9081 :113,044,044,043,043,043,195
9087 :038,037,037,165,035,035,218
9093 :098,161,160,093,156,027,060
9099 :219,000,000,000,000,241,087
9105 :049,113,044,044,043,043,225
9111 :107,038,038,038,102,035,253
9117 :163,034,033,097,029,092,093
9123 :156,000,000,000,000,000,063
9129 :000,000,000,236,236,235,108
9135 :230,038,038,038,038,038,083
9141 :036,100,034,033,033,029,190
9147 :029,000,000,000,000,000,216
9153 :242,050,242,000,243,000,202
9159 :000,000,230,230,038,038,223
9165 :230,036,228,226,225,222,092
9171 :222,030,000,000,000,000,207
9177 :000,242,050,050,000,000,047
9183 :243,000,000,000,000,230,184
9189 :230,000,000,000,000,000,203
9195 :000,000,222,222,000,000,167
9201 :000,000,242,242,242,242,185
9207 :000,000,243,000,000,000,234
9213 :000,230,000,000,000,000,227
9219 :000,000,000,222,030,222,221
9225 :000,000,242,000,000,000,251
9231 :242,000,000,000,000,000,001
```

```
9237 :000,000,000,000,000,000,021
9243 :000,000,000,000,000,222,249
9249 :222,000,000,255,032,082,112
9255 :069,071,073,079,078,083,236
9261 :000,078,069,087,032,069,124
9267 :078,071,076,000,085,082,187
9273 :066,065,078,032,078,069,189
9279 :000,072,069,065,082,084,179
9285 :076,078,068,000,071,032,138
9291 :080,076,065,073,078,083,018
9297 :000,065,084,076,065,078,193
9303 :084,073,067,000,083,079,217
9309 :085,084,072,069,082,078,051
9315 :000,065,082,075,076,065,206
9321 :084,069,088,000,077,079,246
9327 :085,078,084,065,073,078,062
9333 :000,080,065,067,073,070,216
9339 :073,067,032,000,001,007,047
9345 :010,015,022,031,035,039,025
9351 :047,052,052,220,243,243,224
9357 :047,063,220,078,228,077,086
9363 :077,228,227,206,092,062,015
9369 :243,092,227,242,227,243,147
9375 :099,063,047,228,063,069,216
9381 :100,190,069,070,100,077,003
9387 :077,070,070,212,078,212,122
9393 :243,243,197,212,228,243,007
9399 :197,235,242,228,242,047,094
9405 :033,059,033,246,104,126,022
9411 :202,189,036,097,089,189,229
9417 :220,052,118,122,081,038,064
9423 :003,171,186,238,254,204,239
9429 :171,002,080,070,070,235,073
9435 :000,145,069,001,001,134,057
9441 :087,203,097,096,119,223,026
9447 :066,234,170,246,245,234,146
9453 :158,124,254,111,247,057,164
9459 :067,159,211,066,027,095,100
9465 :029,104,164,179,005,065,027
9471 :052,233,044,056,004,136,012
9477 :017,210,066,230,063,169,248
9483 :175,077,154,057,061,092,115
9489 :140,062,047,120,216,037,127
9495 :059,005,145,213,145,243,065
9501 :187,242,011,230,131,193,255
9507 :000,000,068,082,085,159,173
9513 :028,152,000,000,013,013,247
```

4
Creature Features

The Frantic Fisherman

David Lacey

64 Version by Kevin Martin

Idly floating in your boat, waiting for the fish to bite, is a fine way to relax. In this game, however, an angler's dream becomes a nightmare when sharks get the notion that you're the bait and the thunderclouds threaten you with gargantuan raindrops. It's good you remembered to bring your shark swatter and an umbrella.

The fish are biting, and you've managed to catch a few. But suddenly you notice the sky is clouding over, and to make things worse, ravenous sharks begin to circle your boat.

The object of "The Frantic Fisherman" is to survive. You score points by bopping the sharks with your bat and blocking raindrops with your umbrella. You start with three fishermen. Each time a shark or raindrop hits the boat, you lose the boat and one fisherman. However, a new fisherman is awarded for every 2,000 points.

Three keys are used to control movement. To move back and forth, use the less than (<) and greater than (>) keys. The space bar serves two functions. When the sharks approach, it controls the club. If a raindrop is falling, it controls the umbrella. You can use the shark swatter as many times as you like. The umbrella, though, can be opened only three times for each raindrop.

Controlling the Frenzy

If you think the game is too fast or slow, you can make the fisherman more or less frantic. Since the bulk of the 64 version is written in machine language, the speed controls are built-in. The four function keys give you four speeds, from very slow (f1) to frustratingly frantic (f7). The first speed (f1) is rather easy and is recommended only as practice. You can also pause the action by pressing the SHIFT/LOCK key. Pressing it again restarts the game. To end the game, press the back arrow (←) key.

Machine Language Speed

Frantic Fisherman combines the best of machine language (ML) and BASIC to create a game that's fast-moving, yet fairly easy to type in. You don't need to know anything about ML to enter or play this game. However, some of the program's structure may be of interest.

The program POKEs in the sprite and machine language data before it does anything else by GOSUBing to line 8000. Here the DATA statements are read by a BASIC loader. First the sprite pattern information is POKEd into memory. The data are in lines 1000–1064. If you made a mistake in typing in the data, an error message will display. If the numbers were correct, then the program POKEs in the machine language data, found in lines 49152–50346. The ML portion of Frantic Fisherman is placed in an area of memory starting at location 49152, where it is safe from BASIC. There's another check of these numbers to make sure you typed them in correctly. If there's an error, you'll see a message on the screen. LIST the program and carefully check it against the listing published here. After you've found the mistake, save the program to tape or disk (preferably using a new filename), then load and run it to see if it works.

The machine language section of the program controls the movement of the sharks and raindrops (which are sprites), reads the keyboard to switch the fisherman from one end of the boat to the other, displays the umbrella and bat when necessary, and keeps track of the score. It also allows you to choose the speed of the game by pressing the function keys. If all this were in BASIC, the game would play much slower. Only through machine language can you get a truly arcade-style game like The Frantic Fisherman.

Error-Free Entry

To make it easy to type in The Frantic Fisherman, be sure to read Appendix C and use the "Automatic Proofreader" program you'll find listed there. It will especially help as you enter the mass of numbers for the sprite and ML portions of the program.

After saving a copy of the game to tape or disk, load it into your 64. It takes a few moments for all the sprite and machine language data to be POKEd into the computer's memory. You'll see the title screen, and after pressing any key, the game begins. Watch out for the sharks!

The Frantic Fisherman

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
4 POKE56,60:CLR                                        :rem 123
5 GOSUB 8000                                           :rem 125
10 POKE 53280,0:POKE 53281,0                           :rem 182
20 PRINT "{CLR}{N}{5}{DOWN} {BLU}{A}*********{S}
   {3 SPACES}{5}PRESS ANY KEY TO BEGIN"                :rem 56
30 PRINT " {BLU}{9 SPACES}-"                           :rem 13
40 PRINT "{SHIFT-SPACE}-{2 SPACES}{A}******{X}
   {5 SPACES}{5}HIGH SCORE:";HS                        :rem 227
50 PRINT "{SHIFT-SPACE}{BLU}-{2 SPACES}{Z}******
   {S}{5 SPACES}{5}YOUR SCORE:";SC                     :rem 27
60 PRINT "{SHIFT-SPACE}{BLU}-{9 SPACES}-"              :rem 176
70 PRINT "{SHIFT-SPACE}-{2 SPACES}{A}******+*****
   {R}*****{R}*****{R}*{R}****{S}"                     :rem 54
80 PRINT "{SHIFT-SPACE}-{2 SPACES}-{6 SPACES}-
   {5 SPACES}-{5 SPACES}-{5 SPACES}- -{4 SPACES}-"
                                                       :rem 193
90 PRINT "{SHIFT-SPACE}-{2 SPACES}- {A}****{W}
   {2 SPACES}{U}{2 SPACES}- {A}*{S} {Q}*{S} {A}*
   {W} - {A}**{X}"                                     :rem 176
100 PRINT "{SHIFT-SPACE}-{2 SPACES}- -{4 SPACES}-
    {SPACE}{A}*{S} - - - - - - - - {Z}**{S}"
                                                       :rem 87
110 PRINT " -{2 SPACES}- -{4 SPACES}- - - - - - - -
    {SPACE}- - - -{4 SPACES}-"                         :rem 86
120 PRINT "{A}{E}**{E}*{E}***{S}{Z}*{X} {Z}*{E}*
    {X} {Z}*{X} {Z}*{X} {Z}*{E}****{X}"                :rem 73
130 PRINT "-{9 SPACES}- {5}USE SPACE TO RAISE UMBR
    ELLA"                                              :rem 123
140 PRINT "{BLU}-{2 SPACES}{A}******{X}{6 SPACES}
    {5}OR FEND OFF SHARK"                              :rem 249
150 PRINT "{BLU}-{2 SPACES}{Z}******{S}{3 SPACES}
    {5}USE < AND > TO MOVE FROM"                       :rem 158
160 PRINT "{BLU}-{9 SPACES}-{9 SPACES}{5}LEFT TO R
    IGHT"                                              :rem 37
170 PRINT "{BLU}-{2 SPACES}{A}*{R}****+*{S} {A}*
    {R}****{R}****{R}****{R}***{R}***{S}"; :rem 35
180 PRINT "-{2 SPACES}- - {A}**{W} - - - {A}**{W}
    {4 SPACES}-{4 SPACES}- {O} -{3 SPACES}-";
                                                       :rem 198
190 PRINT "-{2 SPACES}- - {Z}**{W} {Z}*{X} - {Z}*
    {S}- {A}**{W} -- -{A}*{S}-{A}*{S}-";  :rem 140
200 PRINT "-{2 SPACES}- {Q}**{S} - {A}*{S} - {A}*
    {X}- -{2 SPACES}- -- -- --- --";      :rem 146
210 PRINT "-{2 SPACES}- {Q}**{X} - - - - {Z}**{W}
    {SPACE}-{2 SPACES}- -- -- --- --";     :rem 20
```

```
220 PRINT "{Z}**{E}*{E}****{E}*{X} {Z}*{E}****{E}*
    {X}{2 SPACES}{Z}*{2 E}*{E}{X} {Z}{E}{X} {Z}{X}
    ";                                          :rem 235
230 GETA$:IFA$=""THEN230                        :rem 77
240 POKE53281,14:POKE53280,6                    :rem 38
245 PRINT"{CLR}"CHR$(142)"{BLK}SCORE:{19 SPACES}FI
    SHERMEN:"                                   :rem 77
250 PRINT"{2 DOWN}{WHT}{13 SPACES}{D}{RVS}
    {4 SPACES}{OFF}{F}"                         :rem 154
260 PRINT"{12 SPACES}{C}{RVS}{9 SPACES}{OFF}{3 I}
    {F}{5 SPACES}{D}{RVS}{2 SPACES}{OFF}{F}"
                                                :rem 117
270 PRINT"{4 SPACES}{D}{RVS}{4 SPACES}{OFF}{F}
    {3 SPACES}{C}{RVS}{13 SPACES}{OFF}{V}
    {2 SPACES}{D}{RVS}{7 SPACES}{OFF}{F}"       :rem 158
280 PRINT"{2 SPACES}{D}{RVS}{7 SPACES}{OFF}{F}
    {5 SPACES}{C}{RVS}{8 SPACES}{OFF}{V}{3 SPACES}
    {C}{RVS}{7 SPACES}{OFF}{V}"                 :rem 178
290 PRINT" {C}{RVS}{12 SPACES}{OFF}{V}{4 SPACES}
    {C}{RVS}{4 SPACES}{OFF}{V}{7 SPACES}{C}{RVS}
    {3 SPACES}{OFF}{V}"                         :rem 198
300 PRINT"{5 SPACES}{C}{RVS}{7 SPACES}{OFF}{V}"
                                                :rem 130
310 PRINT"{7 SPACES}{C}{RVS}{3 I}{OFF}"         :rem 171
311 PRINT"{5 DOWN}{22 SPACES}{RVS}£{OFF}{W}"
                                                :rem 187
312 PRINT"{21 SPACES}{RVS}£ {OFF}{W}"           :rem 103
313 PRINT"{20 SPACES}{RVS}£{2 SPACES}{OFF}{W}"
                                                :rem 104
314 PRINT"{19 SPACES}{RVS}£{3 SPACES}{OFF}{W}"
                                                :rem 105
315 PRINT"{18 SPACES}{RVS}£{4 SPACES}{OFF}{W}"
                                                :rem 106
316 PRINT"{17 SPACES}{RVS}£{5 SPACES}{OFF}{W}"
                                                :rem 107
317 PRINT"{16 SPACES}{RVS}£{6 SPACES}{OFF}{W}"
                                                :rem 108
318 PRINT"{16 SPACES}{Z}{6 E}{W}"               :rem 243
319 PRINT"{14 SPACES}{2}{*}{RVS}{10 SPACES}{OFF}£
    "                                           :rem 47
320 PRINT"{RVS}{BLU}{39 SPACES}{OFF}{BLK}";
                                                :rem 244
330 POKE2023,160:POKE2023+54272,6               :rem 16
340 SYS49152                                    :rem 155
350 PRINT"{HOME}{BLK}{12 DOWN}{12 RIGHT}PRESS RETU
    RN KEY"                                     :rem 138
360 GETA$:IFA$<>CHR$(13)THEN360                 :rem 4
370 S1=PEEK(829):S2=PEEK(830):S3=PEEK(831):rem 144
```

```
380 SC=INT(S1/16)*10+(S1AND15)+INT(S2/16)*1000+(S2
    AND15)*100                                  :rem 234
390 SC=SC+INT(S3/16)*100000+(S3AND15)*10000:rem 41
400 IF SC>HS THEN HS=SC                          :rem 47
410 GOTO10                                       :rem 47
1000 DATA192,0,0,224,0,0,112,0                  :rem 199
1001 DATA0,56,0,0,28,0,0,14                      :rem 58
1002 DATA0,0,7,0,0,3,128,0                        :rem 6
1003 DATA1,128,0,0,0,0,0,0                      :rem 254
1004 DATA0,0,0,0,0,0,0,0                        :rem 147
1005 DATA0,0,0,0,0,0,0,0                        :rem 148
1006 DATA0,0,0,0,0,0,0,0                        :rem 149
1007 DATA0,0,0,0,0,0,0,0                        :rem 150
1008 DATA1,128,0,3,128,0,7,0                    :rem 120
1009 DATA0,14,0,0,28,0,0,56                      :rem 66
1010 DATA0,0,112,0,0,224,0,0                     :rem 92
1011 DATA192,0,0,0,0,0,0,0                      :rem 253
1012 DATA0,0,0,0,0,0,0,0                        :rem 146
1013 DATA0,0,0,0,0,0,0,0                        :rem 147
1014 DATA0,0,0,0,0,0,0,0                        :rem 148
1015 DATA0,0,0,0,0,0,0,53                       :rem 205
1016 DATA7,0,0,63,224,0,255,248                  :rem 24
1017 DATA0,2,0,0,2,0,0,2                        :rem 157
1018 DATA0,0,2,0,0,2,0,0                        :rem 156
1019 DATA2,0,0,18,0,0,12,0                        :rem 7
1020 DATA0,0,0,0,0,0,0,0                        :rem 145
1021 DATA0,0,0,0,0,0,0,0                        :rem 146
1022 DATA0,0,0,0,0,0,0,0                        :rem 147
1023 DATA0,0,0,0,0,0,0,0                        :rem 148
1024 DATA16,0,0,56,0,0,124,0                    :rem 110
1025 DATA0,254,0,0,158,0,0,206                  :rem 215
1026 DATA0,0,124,0,0,0,0,0                      :rem 254
1027 DATA0,0,0,0,0,0,0,0                        :rem 152
1028 DATA0,0,0,0,0,0,0,0                        :rem 153
1029 DATA0,0,0,0,0,0,0,0                        :rem 154
1030 DATA0,0,0,0,0,0,0,0                        :rem 146
1031 DATA0,0,0,0,0,0,0,0                        :rem 147
1032 DATA0,0,0,0,0,0,0,0                        :rem 148
1033 DATA0,0,0,0,0,0,0,0                        :rem 149
1034 DATA4,0,0,12,0,0,28,0                        :rem 7
1035 DATA0,60,1,7,255,195,14,127                 :rem 79
1036 DATA255,31,255,255,127,255,255,56          :rem 141
1037 DATA127,255,3,255,195,0,0,1                 :rem 78
1038 DATA0,0,0,0,0,0,0,0                        :rem 154
1039 DATA0,0,0,0,0,0,0,183                         :rem 7
1040 DATA0,0,0,0,0,0,0,0                        :rem 147
1041 DATA0,0,0,0,0,0,0,0                        :rem 148
1042 DATA32,0,0,48,0,0,56,0                      :rem 65
1043 DATA128,60,0,195,255,224,255,254            :rem 81
```

```
1044 DATA112,255,255,248,255,255,254,255  :rem 240
1045 DATA254,28,195,255,192,128,0,0       :rem 239
1046 DATA0,0,0,0,0,0,0,0                  :rem 153
1047 DATA0,0,0,0,0,0,0,183                  :rem 6
1048 DATA0,0,0,0,0,0,0,0                  :rem 155
1049 DATA0,0,255,0,0,255,0,3              :rem 119
1050 DATA255,192,0,85,0,0,89,0            :rem 234
1051 DATA0,85,64,0,90,0,0,85              :rem 130
1052 DATA0,0,255,0,3,255,192,3            :rem 224
1053 DATA245,80,3,255,192,3,255,192       :rem 240
1054 DATA3,255,192,0,0,0,0,0              :rem 115
1055 DATA0,0,0,0,0,0,0,0                  :rem 153
1056 DATA0,0,0,0,0,0,0,0                  :rem 154
1057 DATA0,0,255,0,0,255,0,3              :rem 118
1058 DATA255,192,0,85,0,0,101,0            :rem 19
1059 DATA1,85,0,0,165,0,0,85              :rem 132
1060 DATA0,0,255,0,3,255,192,5            :rem 225
1061 DATA95,192,3,255,192,3,255,192       :rem 246
1062 DATA3,255,192,0,0,0,0,0              :rem 114
1063 DATA0,0,0,0,0,0,0,0                  :rem 152
1064 DATA 256                             :rem 130
8000 PRINT"{N}{CLR}{12 DOWN}{RIGHT}LOADING SPRITES
      AND MACHINE LANGUAGE"                :rem 87
8010 PRINT"{10 RIGHT}{4 DOWN}PLEASE BE PATIENT..."
                                           :rem 96
9000 I=248*64                             :rem 129
9010 READ A:IF A=256 THEN 9100              :rem 4
9020 POKE I,A:I=I+1:CK=CK+A:GOTO 9010      :rem 81
9100 IF CK<>19128 THEN PRINT"ERROR IN DATA (LINES
     {SPACE}1000-1064)":STOP                :rem 38
10000 I=49152:CK=0                        :rem 177
10010 READ A:IF A=256 THEN 10100            :rem 84
10020 POKE I,A:I=I+1:CK=CK+A:GOTO 10010   :rem 161
10100 IF CK<>139243 THEN PRINT"ERROR IN DATA (LINE
      S 49152-50346)":STOP                 :rem 250
10200 RETURN                              :rem 211
49152 DATA 169,3,141,64,3,169             :rem 161
49158 DATA 7,141,21,208,169,217             :rem 5
49164 DATA 141,1,208,169,1,141            :rem 198
49170 DATA 28,208,169,10,141,37             :rem 0
49176 DATA 208,169,7,141,38,208            :rem 13
49182 DATA 169,0,141,39,208,32            :rem 207
49188 DATA 60,193,169,25,141,60            :rem 10
49194 DATA 3,169,250,141,250,7            :rem 207
49200 DATA 169,209,141,5,208,169           :rem 53
49206 DATA 2,141,41,208,169,44            :rem 202
49212 DATA 32,238,193,32,156,195           :rem 55
49218 DATA 169,0,141,61,3,141             :rem 148
49224 DATA 62,3,141,63,3,32                :rem 45
```

```
49230 DATA 174,195,32,141,196,169          :rem 110
49236 DATA 32,141,71,3,169,0               :rem 100
49242 DATA 141,72,3,173,60,3                :rem 97
49248 DATA 141,67,3,32,4,196               :rem 112
49254 DATA 32,238,193,206,67,3             :rem 213
49260 DATA 208,245,141,4,212,32            :rem 244
49266 DATA 150,192,173,64,3,201            :rem 254
49272 DATA 0,208,226,169,0,133             :rem 201
49278 DATA 198,169,0,141,21,208              :rem 9
49284 DATA 169,0,141,4,212,169             :rem 208
49290 DATA 0,162,0,157,0,208                :rem 94
49296 DATA 232,224,17,208,248,96            :rem 66
49302 DATA 173,30,208,141,69,3             :rem 200
49308 DATA 173,65,3,201,1,240              :rem 145
49314 DATA 87,173,66,3,201,0               :rem 103
49320 DATA 240,46,173,69,3,41              :rem 154
49326 DATA 6,201,6,208,11,169              :rem 153
49332 DATA 117,32,31,195,32,179              :rem 2
49338 DATA 194,76,46,193,173,2             :rem 224
49344 DATA 208,201,225,208,3,76            :rem 254
49350 DATA 37,193,206,2,208,173              :rem 1
49356 DATA 2,208,201,255,208,5             :rem 203
49362 DATA 169,0,141,16,208,96             :rem 212
49368 DATA 173,2,208,201,115,144            :rem 46
49374 DATA 3,76,37,193,173,69              :rem 178
49380 DATA 3,41,6,201,6,208                 :rem 47
49386 DATA 11,169,117,32,31,195               :rem 7
49392 DATA 32,179,194,76,46,193             :rem 25
49398 DATA 238,2,208,96,173,3              :rem 173
49404 DATA 208,201,227,144,3,76            :rem 252
49410 DATA 37,193,173,69,3,41              :rem 161
49416 DATA 6,201,6,208,11,169              :rem 153
49422 DATA 80,32,31,195,32,202             :rem 196
49428 DATA 194,76,46,193,173,69             :rem 29
49434 DATA 3,41,3,201,3,240                 :rem 37
49440 DATA 4,238,3,208,96,32               :rem 109
49446 DATA 151,194,206,64,3,32             :rem 208
49452 DATA 156,195,162,30,32,106            :rem 49
49458 DATA 195,202,208,250,165,162         :rem 157
49464 DATA 201,192,144,38,169,0               :rem 4
49470 DATA 141,2,208,169,229,141            :rem 51
49476 DATA 3,208,169,253,141,249            :rem 64
49482 DATA 7,169,0,141,40,208              :rem 156
49488 DATA 141,16,208,141,65,3             :rem 210
49494 DATA 141,66,3,169,0,141              :rem 159
49500 DATA 27,208,173,30,208,96               :rem 2
49506 DATA 201,128,144,44,169,80            :rem 53
49512 DATA 141,2,208,169,229,141            :rem 48
49518 DATA 3,208,169,252,141,249            :rem 60
```

```
49524 DATA 7,169,0,141,40,208          :rem 153
49530 DATA 169,2,141,16,208,169          :rem 4
49536 DATA 0,141,65,3,169,1             :rem 54
49542 DATA 141,66,3,169,0,141          :rem 153
49548 DATA 27,208,173,30,208,96         :rem 14
49554 DATA 201,64,144,44,169,139        :rem 60
49560 DATA 141,2,208,169,100,141        :rem 39
49566 DATA 3,208,169,251,141,249        :rem 62
49572 DATA 7,169,6,141,40,208          :rem 162
49578 DATA 169,0,141,16,208,169         :rem 14
49584 DATA 1,141,65,3,169,0             :rem 57
49590 DATA 141,66,3,169,2,141          :rem 158
49596 DATA 27,208,173,30,208,96         :rem 17
49602 DATA 169,218,141,2,208,169        :rem 56
49608 DATA 100,141,3,208,169,251        :rem 45
49614 DATA 141,249,7,169,6,141         :rem 214
49620 DATA 40,208,169,0,141,16         :rem 198
49626 DATA 208,169,1,141,65,3          :rem 160
49632 DATA 169,1,141,66,3,169          :rem 164
49638 DATA 2,141,27,208,173,30         :rem 205
49644 DATA 208,96,201,44,208,17          :rem 7
49650 DATA 169,132,141,0,208,169        :rem 52
49656 DATA 255,141,248,7,32,227         :rem 11
49662 DATA 194,32,112,195,96,201        :rem 57
49668 DATA 46,208,17,169,212,141        :rem 62
49674 DATA 0,208,169,254,141,248        :rem 61
49680 DATA 7,32,227,194,32,112         :rem 207
49686 DATA 195,96,201,32,208,113        :rem 60
49692 DATA 173,68,3,201,0,208          :rem 157
49698 DATA 115,173,70,3,201,0          :rem 153
49704 DATA 208,108,173,65,3,201        :rem 253
49710 DATA 0,240,38,169,2,141          :rem 148
49716 DATA 41,208,169,209,141,5          :rem 6
49722 DATA 208,169,250,141,250,7        :rem 51
49728 DATA 173,0,208,201,132,208        :rem 44
49734 DATA 8,169,138,141,4,208         :rem 217
49740 DATA 76,132,194,169,218,141      :rem 112
49746 DATA 4,208,76,132,194,169         :rem 19
49752 DATA 0,141,41,208,173,0          :rem 145
49758 DATA 208,201,132,208,18,169      :rem 109
49764 DATA 127,141,4,208,169,230        :rem 55
49770 DATA 141,5,208,169,249,141        :rem 59
49776 DATA 250,7,76,132,194,169         :rem 22
49782 DATA 232,141,4,208,169,230        :rem 52
49788 DATA 141,5,208,169,248,141        :rem 67
49794 DATA 250,7,32,1,195,169          :rem 170
49800 DATA 200,141,68,3,96,201         :rem 198
49806 DATA 95,208,5,169,0,141          :rem 164
49812 DATA 64,3,96,32,141,196          :rem 165
```

```
49818 DATA 169,33,141,4,212,162          :rem 2
49824 DATA 255,142,1,212,142,37         :rem 251
49830 DATA 208,32,106,195,202,208        :rem 97
49836 DATA 244,169,10,141,37,208         :rem 57
49842 DATA 96,32,141,196,169,129         :rem 71
49848 DATA 141,4,212,162,255,142         :rem 50
49854 DATA 1,212,142,40,208,32          :rem 196
49860 DATA 106,195,202,208,244,96       :rem 110
49866 DATA 32,141,196,169,129,141       :rem 116
49872 DATA 4,212,162,0,142,1             :rem 94
49878 DATA 212,142,40,208,32,106         :rem 48
49884 DATA 195,232,224,50,208,242       :rem 108
49890 DATA 96,169,33,141,4,212          :rem 216
49896 DATA 162,15,142,1,212,32          :rem 203
49902 DATA 106,195,32,106,195,202        :rem 99
49908 DATA 224,5,208,242,169,0          :rem 211
49914 DATA 141,4,212,32,106,195         :rem 251
49920 DATA 96,169,33,141,4,212          :rem 210
49926 DATA 162,5,142,1,212,32           :rem 148
49932 DATA 106,195,32,106,195,232       :rem 105
49938 DATA 224,20,208,242,169,0            :rem 3
49944 DATA 141,4,212,32,106,195         :rem 254
49950 DATA 96,248,24,109,61,3           :rem 168
49956 DATA 141,61,3,169,0,109           :rem 161
49962 DATA 62,3,141,62,3,169            :rem 112
49968 DATA 0,109,63,3,141,63            :rem 111
49974 DATA 3,216,32,174,195,56          :rem 222
49980 DATA 173,62,3,237,71,3            :rem 113
49986 DATA 141,69,3,173,63,3            :rem 121
49992 DATA 237,72,3,13,69,3              :rem 69
49998 DATA 144,25,169,32,248,24          :rem 22
50004 DATA 109,71,3,141,71,3             :rem 85
50010 DATA 169,0,109,72,3,141           :rem 136
50016 DATA 72,3,216,238,64,3             :rem 97
50022 DATA 32,156,195,96,160,0          :rem 197
50028 DATA 200,208,253,96,169,0         :rem 250
50034 DATA 141,41,208,173,0,208         :rem 236
50040 DATA 201,132,208,16,169,248        :rem 87
50046 DATA 141,250,7,169,127,141         :rem 41
50052 DATA 4,208,169,222,141,5          :rem 193
50058 DATA 208,96,169,249,141,250       :rem 109
50064 DATA 7,169,232,141,4,208          :rem 199
50070 DATA 169,222,141,5,208,96         :rem 252
50076 DATA 162,0,160,35,24,32           :rem 139
50082 DATA 240,255,173,64,3,24          :rem 197
50088 DATA 105,48,32,210,255,96            :rem 0
50094 DATA 162,0,160,6,32,240           :rem 137
50100 DATA 255,173,63,3,41,240          :rem 186
50106 DATA 74,74,74,74,24,105           :rem 154
```

```
50112 DATA 48,32,210,255,173,63          :rem 243
50118 DATA 3,41,15,24,105,48              :rem 91
50124 DATA 32,210,255,173,62,3           :rem 188
50130 DATA 41,240,74,74,74,74            :rem 150
50136 DATA 24,105,48,32,210,255          :rem 241
50142 DATA 173,62,3,41,15,24              :rem 89
50148 DATA 105,48,32,210,255,173          :rem 41
50154 DATA 61,3,41,240,74,74              :rem 96
50160 DATA 74,74,24,105,48,32            :rem 149
50166 DATA 210,255,173,61,3,41           :rem 193
50172 DATA 15,24,105,48,32,210           :rem 187
50178 DATA 255,96,32,16,196,72           :rem 219
50184 DATA 32,71,196,32,125,196            :rem 2
50190 DATA 104,96,32,228,255,201          :rem 41
50196 DATA 0,208,3,76,70,196             :rem 108
50202 DATA 201,133,208,7,169,25          :rem 241
50208 DATA 141,60,3,169,133,201          :rem 238
50214 DATA 134,208,7,169,18,141          :rem 250
50220 DATA 60,3,169,134,201,135          :rem 236
50226 DATA 208,7,169,13,141,60           :rem 198
50232 DATA 3,169,135,201,136,208          :rem 37
50238 DATA 7,169,9,141,60,3               :rem 55
50244 DATA 169,136,96,173,68,3           :rem 218
50250 DATA 240,5,206,68,3,240            :rem 140
50256 DATA 21,173,70,3,201,0              :rem 83
50262 DATA 240,3,206,70,3,162            :rem 137
50268 DATA 90,202,208,253,169,0          :rem 252
50274 DATA 141,4,212,96,173,4            :rem 149
50280 DATA 208,201,0,240,5,169           :rem 189
50286 DATA 0,141,4,208,32,112            :rem 136
50292 DATA 195,169,255,141,70,3            :rem 3
50298 DATA 76,81,196,173,141,2           :rem 215
50304 DATA 41,1,201,1,208,6               :rem 28
50310 DATA 32,16,196,76,125,196             :rem 0
50316 DATA 96,162,0,169,0,157            :rem 154
50322 DATA 0,212,232,224,25,208          :rem 231
50328 DATA 248,169,15,141,24,212          :rem 45
50334 DATA 169,16,141,5,212,169          :rem 252
50340 DATA 240,141,6,212,169,100          :rem 26
50346 DATA 141,0,212,96,256               :rem 51
```

Beekeeper

Daniel Gray

64 Translation by Kevin Martin

You find yourself in the middle of a clover field doing battle with some rather nasty giant bees. Try maneuvering to the hive while avoiding the deadly stings and pinches of the bee and crab guardians. "Beekeeper" also contains some innovative programming techniques.

Giant bees, grotesque mutations, are taking over the world. Their enormous beehives are engulfing cities and their clover fields are spreading over the countryside, invading precious croplands. As Beekeeper, you must dodge worker bees and monster crabs in a desperate mission to locate and destroy the hive.

After the program is entered and run, you'll see a title screen. Next, instructions tell you the point values of the game targets and ask you to enter one of three difficulty levels.

Each level determines the speed and direction of the worker bees and crabs as they chase you around the clover field. In level 1, the bees and crabs are confined to vertical and horizontal movement, but in levels 2 and 3 they also move diagonally. At level 3, you must be very quick in order to avoid sting and claw.

The Bees Hunt You Down

Once you've selected a difficulty level, the screen clears and you find yourself in a clover field beside a giant beehive filled with drones. Your first ship appears just above the beehive, near the center of the screen. Pushing the joystick to the right rotates your ship clockwise; pushing the joystick to the left rotates the ship counterclockwise. Depressing the joystick button fires the ship's laser.

Use the Laser to Score

If you don't control the ship, it will run into the clover or the hive. If you are stung or pinched, your ship is destroyed. Take care, for you have only 8 ships available in each game. Defend yourself with your laser—each worker bee or crab you disable is worth 200 points.

Piercing a block of the hive gives you 50 points, while eliminating a drone within the hive awards you with 100 points. You can also fire at clover to get it out of your way. Clover is worth 50 points each. The best way to aim at the hive is by looping around the clover field until you are moving directly toward the target. Continue straight ahead while firing at the hive.

Once all 66 drones in the hive have been exterminated, the screen clears and another field is created, along with more ships. The game is over when all your ships have been destroyed.

Keyboard Control

If you don't have a joystick, or would rather use the keyboard to control your ship, you can make some simple changes to the program.

```
5Ø S=PEEK(2Ø3)
6Ø REM NOT NEEDED
7Ø IF S<>21THEN13Ø
13Ø IF S<>38THENIFS<>41THEN2ØØ
14Ø CC=1:IFS=41THENCC=-1
6ØØ PRINT"{CLR}"SPC(1Ø)"{RED}{DOWN}USE KEYBOARD TO
     PLAY"
```

The keyboard version plays exactly like the joystick version, except that three keys replace the joystick controls. The *P* key rotates the ship clockwise, the *O* key rotates it counterclockwise, and the *F* key is the fire button.

These keys are detected by PEEKing memory location 203. Each time a key is pressed, a unique number representing that key is stored in location 203 (and in location 197). For example, when *F* is pressed, a 21 is stored in that address. When *O* is pressed, location 203 contains 38, and a 41 is placed in this location when you press the *P* key.

Since this process requires fewer variables than the joystick routine, not as much RAM memory is used. However, the ship is slightly harder to control with the keyboard.

If you want to change the keys that control your ship's movement and laser fire, you'll need to know the number that represents your new key in location 203. To find the number, type in this one-line program:

```
1Ø PRINT PEEK(2Ø3):GOTO1Ø
```

This simple program is an infinite loop that displays the contents of location 203 onto the screen. When you run it, you'll see the number 64 scroll continuously up the screen; location 203 contains a 64 when no key is being pressed. To see the number representing any key, just hold down the key and note the number that scrolls up the screen. To substitute this new key for the fire button, replace the 21 in line 70 above with the new value. You can replace the other two keys (for clockwise and counterclockwise movement of the ship) in the same way.

More Manipulations

Other versions of "Beekeeper" can be created by manipulating the initial values of the variables. Here's a list of the most useful variables, found in lines 700–710:

Variable	Description
P1	Starting position of ship on screen
SH	Starting direction of ships
SQ	Starting number of ships provided in each level
AQ	Starting number of drone bees in hive for each level
SA	Highest point on the screen that the ship can reach
SE	Lowest point on the screen that the ship can reach

The IF-THEN statement in line 110 can be changed to give your ship's laser a greater range. For example, you can have the laser reach across the screen by changing this statement to:

```
IF I<39 THEN 90
```

The DATA in lines 840–885 control the shape of Beekeeper's custom characters. By changing the DATA in these lines, you can create your own character set, completely altering the game.

Beekeeper

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
1 POKE56,48:POKE55,0:CLR                    :rem 173
5 POKE53280,2:POKE53281,0                   :rem 140
10 DIMSP(8)                                 :rem 103
20 PRINT"{CLR}"                             :rem 198
30 V=1:SC=0:CM=54272                        :rem 111
40 GOSUB500:GOSUB800:GOSUB600                :rem 27
```

```
41 POKECM+24,15:POKECM+5,17:POKECM+6,241:POKECM,Ø:
   POKECM+12,17:POKECM+13,241                    :rem 12
42 POKECM+7,Ø                                    :rem 227
45 GOSUB7ØØ                                      :rem 128
5Ø S=PEEK(56320):SW=(SAND4)/4:F=(SAND16)/16
                                                 :rem 2Ø3
6Ø SR=(SAND8)/8                                   :rem 94
7Ø IFF=1THEN13Ø                                  :rem 1Ø9
8Ø I=1:A=SH-32:J=P1:POKECM+8,2ØØ:POKECM+11,129
                                                 :rem 174
9Ø J=J+SP(A):IFJ<SAORJ>2Ø23THEN12Ø                :rem 47
1ØØ IFPEEK(J)<>32THENPOKEBN,32:GOSUB31Ø:GOTO12Ø
                                                 :rem 146
11Ø POKEBN,32:POKEJ,42:POKEBBN+CM,3:BN=J:I=I+1:IFI
    <8THEN9Ø                                      :rem 8Ø
12Ø POKEBN,32:POKECM+11,128                       :rem 48
13Ø IFSW=1THENIFSR=1THEN2ØØ                      :rem 2Ø5
14Ø CC=1:IFSW=ØTHENCC=-1                         :rem 185
15Ø POKECM+1,5Ø:POKECM+4,33:IFP1=P2THENJ=P1:GOSUB3
    1Ø                                           :rem 186
16Ø IFSH=4ØTHENIFCC=1THENCC=-7                   :rem 1Ø6
17Ø IFSH=33THENIFCC=-1THENCC=7                   :rem 1Ø9
18Ø SH=SH+CC:POKEP1,SH:POKECM+4,32                :rem 87
2ØØ A=SH-32:MN=P1:P1=P1+SP(A):IFP1<SAORP1>SETHENP1
    =MN                                          :rem 67
21Ø IFPEEK(P1)<>32THENJ=P1:POKEMN,32:GOSUB3ØØ
                                                 :rem 17
22Ø POKEMN,32:POKEP1,SH:IFW=1THEN25Ø             :rem 244
23Ø P2=INT(RND(1)*21)+1783:MC=INT(RND(1)*6)+1:SX=4
    1:SY=32:BC=3:W=1                             :rem 35
24Ø IFMC=2THENSX=44:SY=43:BC=5                    :rem 94
25Ø A=SGN(P2-P1):MØ=P2:P2=P2-A*H:IFABS(P2-P1)>25TH
    ENP2=P2-A*4Ø                                 :rem 128
26Ø IFP2<SAORP2>SETHENP2=MØ                      :rem 221
27Ø IFA=ØTHENJ=P2:GOSUB31Ø:GOTO5Ø                 :rem 52
28Ø POKECM+P2,MC:POKEMØ,SY:POKEP2,SX:POKECM+MØ,BC:
    GOTO5Ø                                       :rem 1Ø8
3ØØ A=PEEK(J):IFA=43ORA<41THENP1=MN:GOTO48Ø
                                                 :rem 153
31Ø FORI=1TO5:POKEJ,42:POKEJ+CM,2:POKECM+7,5Ø:POKE
    CM+11,129                                     :rem 85
311 POKECM+1,6Ø:POKECM+4,33:POKEJ,32:NEXT        :rem 233
32Ø POKEJ+CM,3:N=SQ:IFJ=P2THENW=Ø:SC=SC+15Ø
                                                 :rem 186
33Ø IFP1=P2ORA=41THENSQ=SQ-1:P1=1764:SH=39        :rem 67
34Ø IFJ>19Ø3THENSC=SC+5Ø:AQ=AQ-1                 :rem 156
35Ø SC=SC+5Ø:IFSC>HITHENHI=SC                     :rem 82
37Ø PRINT"{HOME}{YEL}SCORE:"SC;TAB(32)"SHIPS:"SQ:P
    OKEP1+CM,3:POKEP1,SH:POKECM+11,128           :rem 219
```

```
380 IFSQ=NTHEN410                                   :rem 29
390 FORI=1TO3:FORS=10TO80STEP2:POKECM+1,S:POKECM+4
    ,33:FORA=1TO10:NEXT:NEXT                        :rem 31
400 POKECM+4,32:FORJ=1TO100:NEXT:NEXT               :rem 66
410 IFAQ=0THENV=V+1:PRINT"{CLR}{RED}{11 DOWN}
    {8 RIGHT}SWARM"V:FORI=1TO4000:NEXT:GOTO45
                                                    :rem 104
420 IFSQ<>0THEN480                                  :rem 62
430 PRINT"{HOME}{RED}{8 DOWN}"SPC(14)"* GAME OVER
    {SPACE}*"                                       :rem 169
435 PRINTSPC(11)"{4 DOWN}TRY AGAIN? [Y OR N]"
                                                    :rem 232
440 GETA$:IFA$=""THEN440                            :rem 83
450 IFA$="Y"THENRESTORE:GOTO20                      :rem 143
460 IFA$<>"N"THEN440                                :rem 97
470 PRINT"{CLR}":END                                :rem 16
480 RETURN                                          :rem 124
500 PRINT"{RED}{8 DOWN}{14 RIGHT}*{CYN}BEEKEEPER
    {RED}*"                                         :rem 55
510 PRINT"{GRN}{8 DOWN}{14 RIGHT}HI SCORE={YEL}"HI
                                                    :rem 28
520 FORI=1TO3000:NEXT:RETURN                        :rem 46
600 PRINT"{CLR}"SPC(9)"{RED}{DOWN}USE JOYSTICK IN
    {SPACE}PORT 2"                                  :rem 70
605 PRINTSPC(10)"{RED}{2 DOWN}CRAB"TAB(20)","SPC(5
    )"200"                                          :rem 150
610 PRINTSPC(10)"{5}{DOWN}WORKER"TAB(20)")"SPC(5)"
    200"                                            :rem 188
615 PRINTSPC(10)"{BLU}{DOWN}DRONE"TAB(20)")"SPC(5)
    "100"                                           :rem 229
618 PRINTSPC(10)"{YEL}{DOWN}HIVE"TAB(20)"@"SPC(6)"
    50"                                              :rem 7
620 PRINTSPC(10)"{GRN}{DOWN}CLOVER"TAB(20)"+"SPC(6
    )"50"                                           :rem 10
625 PRINTSPC(10)"{BLU}{2 DOWN}DIFFICULTY LEVELS...
    "                                               :rem 143
630 PRINTSPC(14)"{RED}{DOWN}[1] BEGINNER"           :rem 102
635 PRINTSPC(14)"{DOWN}[2] ADVANCED":PRINTSPC(14)"
    {DOWN}[3] MASTER"                               :rem 171
640 PRINTSPC(11)"{BLU}{DOWN}KEY IN YOUR LEVEL:"
                                                    :rem 181
650 GETA$:IFA$=""THEN650                             :rem 89
660 H=VAL(A$):IFH<1ORH>3THEN650                      :rem 92
670 RETURN                                          :rem 125
700 P1=1764:SH=39:SQ=8:SA=1064:SE=1823              :rem 171
710 CT=55296:CE=56295:AQ=66:BN=SA                   :rem 145
720 PRINT"{CLR}":FORI=CTTOCE:POKEI,3:NEXT           :rem 240
730 FORI=1TO50:A=INT(RND(1)*758)+SA:POKECM+A,5:POK
    EA,43:NEXT                                      :rem 163
```

```
740 FORI=1824TO1903:POKEI+CM,7:POKEI,0:NEXT
                                          :rem 158
750 FORI=1904TO2023:POKEI+CM,INT(RND(1)*6)+1:POKEI
    ,41:NEXT                              :rem 244
760 PRINT"{HOME}{YEL}SCORE:"SC;TAB(32)"SHIPS:"SQ:P
    OKEP1+CM,3:POKEP1,SH:RETURN           :rem 171
800 FORI=1TO8:READSP(I):NEXT:IFPEEK(251)=123THENPO
    KE53272,29:RETURN                     :rem 217
805 PRINT"{7 UP}{9 RIGHT}REDEFINING CHARACTERS"
                                           :rem 37
807 POKE251,123                            :rem 40
810 DATA 1,41,40,39,-1,-41,-40,-39        :rem 103
815 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND25
    1                                     :rem 191
816 FORI=0TO511:POKEI+12288,PEEK(I+53248):NEXT
                                          :rem 237
820 FORI=12552TO12623:READA:POKEI,A:NEXT:FORI=1264
    0TO12647:READA:POKEI,A:NEXT            :rem 80
830 FORI=12288TO12295:POKEI,255:NEXT:FORI=12632TO1
    2639:READA:POKEI,A:NEXT                :rem 34
835 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
                                          :rem 143
836 POKE53272,(PEEK(53272)AND240)+12      :rem 192
840 DATA0,96,112,120,207,120,112,96,8,92,60,108,24
    4,126,6,1                             :rem 123
850 DATA16,254,254,108,56,16,16,16,16,58,60,54,47,
    126,96,128                            :rem 210
860 DATA0,6,14,30,243,30,14,6,128,96,126,47,54,60,
    58,16                                 :rem 194
870 DATA16,16,16,56,108,254,254,16,1,6,126,244,108
    ,60,92,8                               :rem 95
880 DATA195,231,231,126,60,219,189,36,66,102,129,1
    89,126,219,189,36                      :rem 60
885 DATA24,24,102,102,24,24,60,0           :rem 71
890 RETURN                                :rem 129
```

Props

Philip I. Nelson

"Props" is a fast-paced game for the 64, complete with eight sprites, programmed characters, and all three SID voices for sound effects. Animated in machine language, there are six levels of play from which to choose.

Included is a detailed program discussion that offers a variety of excellent programming tips and techniques. Joystick required.

You are a lonely pigeon, lost in a dangerous sky filled with whirling propellers. You want to return to your coop and your mate, if only for a brief rest before flying away again. But to make matters even worse, every time you leave, and at other uncertain intervals, your mate moves to a new coop.

While in flight, you must avoid getting pulled into the propellers. If that happens, you lose points. Unless you escape quickly, the props may pull you back again and again. The props may start in an orderly formation, but every collision bumps one out of line. The worse you play, the more confusing things get.

After you've read Appendix C and saved a copy of "The Automatic Proofreader," you can begin typing in "Props." Since it's such a long program, and since there's a considerable amount of machine language data numbers to enter, the Proofreader will be especially helpful in insuring an error-free copy the first time. There are also short routines within Props to double check that the machine language data have been typed in correctly. If you've entered the wrong numbers, an error message will display on the screen when you try to run the program. Once you've found the bug, save the program again to tape or disk, preferably under a new filename.

To play, plug your joystick into port 2. The six skill levels range from leisurely to maniacal. Whenever you reach home and your mate, the score is displayed briefly. If you press the fire button during the score display, the game pauses to let you catch your breath. During the pause, you can change to a different skill level by pressing number keys 1 through 6. To quit, just pause and press the 0 key. If you score

well at any given level, the game pauses by itself and lets you pick a new skill level.

The Animation Subroutines

Two main machine language (ML) routines are responsible for virtually all the animation. The first one reads the joystick, moves the bird shape accordingly, and flaps the wings of both pigeons. The second rotates the eight propeller sprites and moves them up or down. Two additional small routines help program a new character set and fill color memory with white values for the new-ROM 64s.

Let's first look at the bird-moving routine *(Birdmove)*, which you could adapt for just about any graphics game. This routine begins at line 49400 and continues to line 49650. Since it's an ML routine, it's in the form of numbers, not in BASIC's syntax. Unless you know machine language, it may seem impossible to decipher it. However, we can describe what it does, and how it executes. For those of you who know ML programming, it shouldn't be difficult to modify the routine for your own use.

Birdmove animates the bird-shaped character. The routine keeps track of a variable, *BIRDLOC*, that represents the bird's current screen location. To move the bird around in screen memory (locations 1024–2023), a blank space is first put into the variable BIRDLOC to erase the character. Next, a check is done to see whether any movement has been requested through the joystick. If so, BIRDLOC changes to represent the new screen location. If not, BIRDLOC stays the same. In either case, a new shape appears in the updated BIRDLOC screen location.

Setting the Bird's Boundaries

To move the bird left or right, Birdmove subtracts or adds one to BIRDLOC. To move the bird up or down on the 40-column screen, 40 is subtracted or added. Before moving the pigeon around in memory, you need safeguards to prevent the bird figure from flying above screen memory into the BASIC program space, or below it into the sensitive zero page of memory, either of which could cause the computer to crash.

Birdmove uses two techniques to confine the bird. The first compares BIRDLOC to absolute upper and lower limits. If you try to move lower or higher than the bounds of screen

memory (1024 and 2023 respectively), Birdmove terminates without changing the BIRDLOC variable.

Collision Detection

The second safeguard is a collison check for horizontal movement. When you move left, for instance, Birdmove holds the updated BIRDLOC position in temporary storage. Before it moves a bird figure into the new location, the routine checks that spot to see which of the three possible characters is there.

If the location contains a space, the bird can move left. If the position holds the coop character, the old BIRDLOC is restored and the Birdmove routine is exited without changing position. If *neither* character is found, then the space must contain the mate character, so the routine sets a flag to show that the bird has reached home and ends with the wing-flapping display.

To modify Birdmove for your own games, just add more comparisons to check for as many possibilities as you need. For example, your game might check the desired location and then branch to appropriate routines to score if you've hit a treasure, faint if you've hit a troll, rejoice if you've bumped into a friend, and so on.

The Joystick Flags

The joystick reader at the beginning of Birdmove is from the *Commodore 64 Programmer's Reference Guide*. It stores flag values in a memory location which you can then PEEK to determine movement. In Props, the joystick flag values are in the cassette buffer, but you could put them in any safe memory area. The right/left flag is stored in location 832, and the up/down flag in 833. The value in 832 will be 255 for left, 1 for right, and 0 for no movement. For the up direction, 255 is stored in location 833; 1 for down and 0 for no movement are placed in the same address. Note that leftover flag values will remain in the computer's X and Y registers, so if your ML program goes from this routine to one that uses indirect addressing, you should clear the X and Y registers to 0 to keep things straight.

Programmed characters are used to make the birds' wings flap. In lines 62000–63000 of Props, the character set from the ROM chip is first copied into RAM memory beginning at location 14336. Then new shapes for characters 90–96 are created

by POKEing new values into the RAM character set. Character 90 is programmed to serve as our coop character; the other six characters consist of the various bird shapes.

Each time the Birdmove routine is called, it flips to the next character in this wing-flapping series to create the illusion of movement. To see all the programmed characters, first run the program and then press the RUN/STOP key while the instructions are displayed. Hold down the SHIFT key and press CLR/HOME to blank the screen. Now type in this line in direct mode (without line numbers). Use abbreviations for PRINT (?) and POKE (P-SHIFTed O), or part of it will be cut off.

```
PRINT"{CLR}"TAB(255):K=90:FORJ=1024TO1276STEP42:PO
  KE54272+J,1:POKEJ,K:K=K+1:NEXTJ
```

Press RETURN and you'll see the coop character and six bird shapes in the upper left of the screen.

Flapping from BASIC

Let's make the bird character flap its wings from BASIC in immediate mode. Type the following line and press RETURN:

```
FORK=1TO100000:FORJ=91TO96:POKE1024,J:FORL=1TO30:N
  EXTL,J,K
```

The bird should be flapping at the top left corner. Press the RUN/STOP key when you've seen enough. You can do the same thing with the ML routine. To set things up, enter this line and hit RETURN:

```
POKE 251,0:POKE252,4:POKE834,91
```

This puts information in memory locations that the ML routine uses to position the bird and start the wing-flapping character series. Now type in the next line and press RETURN:

```
FORJ=1TO100000:SYS49608:FORK=1TO30:NEXTK,J
```

Using the Routine's Modules

As before, press STOP when you've seen enough. The entire Birdmove routine starts at location 49408 in memory, with its flap portion toward the end of the routine (49608). At certain points during Props (the reunion or a pause), the birds flap their wings without moving.This is done by starting with the

flapping section at location 49608 and skipping the movement parts entirely. To place the mate somewhere, without any moving or flapping, the routine jumps even further, to 49615. By structuring an ML program in distinct modules like this, you're able to get maximum use out of what you've written.

Let's call the whole Birdmove routine to let the bird fly free. First, type this line and press RETURN:

```
POKE834,91:POKE835,0:POKE836,4:POKE837,230:POKE838
  ,6:POKE251,255:POKE252,5
```

You just positioned the bird and set limits to keep it on the screen. Now enter this as one line. Again, use abbreviations for PRINT and POKE to get this all on two screen lines.

```
PRINT"{CLR}":FORJ=1024TO2008STEP41:POKEJ,90:POKEJ+
  54272,1:NEXT:FORJ=1TO100000:SYS49408:NEXT
```

Using the joystick, you can move one of the bird characters you see on the screen. The bird will wrap around the screen when its way is clear, but stop when it hits a coop character. The up-and-down movement routine contains no collision check, however, so moving in those directions erases any character you encounter.

Vary the Difficulty with Delay Loops

Running at full ML speed, Birdmove is fun to play with, but too fast to be practical. Props uses a variable delay loop (connected to skill level) to slow things down to a manageable speed.

Spritemove, which runs from lines 49000 to 49350, is the second ML routine in Props. It handles the sprite animation, moving the eight propellers up or down at the correct speed and twirling them in unison.

Look at lines 2–6 of the program and you'll see something odd. The game works by cycling through these lines, calling the Birdmove routine over and over with the statement SYS 49408 in line 2. But Spritemove is called only once (SYS 49152) in line 1, while things are being set up. Yet the sprites move continuously as long as you're playing. How can Spritemove work all the time without calling it repeatedly? Easy—the computer just does it along with its other housekeeping.

Harnessing the Hardware Interrupt

In addition to executing your programs, your 64's processor chip has continual housekeeping to do, like updating timers and scanning the keyboard. But it can do only one thing at a time. So occasionally the computer stops doing your work and takes time out for its own tasks. You never notice these *interrupts*, because they happen about 60 times every second.

Like Birdmove, the 64's hardware interrupt routine is just another ML program, starting at location 59953 ($EA31 in hex) in memory. By changing one pointer (vector), the computer performs our ML routine first, then goes on to do its housekeeping as usual—but it does that 60 times a second!

Memory locations 788–789 ($0314–$0315) are specially reserved to hold the address where this interrupt routine begins. When you turn on your 64, it automatically sticks the normal (default) address in these locations. The first part of Spritemove just changes this vector to point the computer to the beginning of our ML program.

At the end of the ML routine, the computer is sent to its normal interrupt program at $EA31, rather than returning to the program as in a conventional ML subroutine.

Watch It in Isolation

Such an *interrupt-driven* ML routine will seem to run independently of BASIC. To watch Spritemove in isolation, first run Props and press the RUN/STOP key when the props move. You'll see the blinking cursor and READY signal, which shows the computer has quit executing the BASIC program. You're back in BASIC's immediate mode, but Spritemove is still working along with the interrupts, so the graphics and sound keep going.

You can do anything you'd normally do from BASIC, even call other ML subroutines as in the examples above, but there's a limit to how far you can take this technique. Grafting a lengthy ML routine onto interrupts will make those "time-outs" so long that they slow the BASIC operations down to a crawl.

To stop Spritemove, first clear the screen of character graphics by holding down SHIFT and pressing CLR/HOME. Now type SYS49152 and press RETURN. The props and sound should freeze.

To restart the props, move the cursor up to the same line

and press RETURN again. The interrupt vector now points to Spritemove, and things are moving. Spritemove is designed to alternately change and restore the interrupt vector every time the routine is called, letting you turn it on or off at will.

The Sprites Are Still There

Note that stopping Spritemove doesn't erase the sprites. If you want them to disappear at certain points in Props, you have to disable their display with the statement POKE SP+21,0. When that's done, the sprites are all still moving (in the sense that Spritemove keeps changing their location registers and shape pointers), but none of this is visible since the computer isn't showing it on the screen.

Compared to the interrupt routine, the rest of Spritemove is simple. The BASIC setup section of Props sets the eight sprites to fixed horizontal locations, giving each a path to move up or down in. Each prop always flies in the same direction—one space up or down on the sprite grid for every execution of Spritemove at skill level 1.

Each sprite has a register (memory location) containing its vertical location. To move the props, Spritemove increments or decrements every vertical location register one or more times, depending on the skill level.

This is simpler in ML than in BASIC. Assume that sprite 1 starts out at vertical location 100. If larger values are placed in its vertical location register, it moves down the screen.

Safe Increments Are Assured

In BASIC you'd have to insert a safeguard in the program to make sure a value larger than 255 couldn't be POKEd into the register, since that would crash the program with an ILLEGAL QUANTITY error.

But ML lacks the error-checking mechanics of BASIC and simply won't let you put a number bigger than 255 into any memory cell. Trying to increment a register from 255 to 256 will just flip its value back to 0. Increment that register again, and it'll contain the value of 1.

The same thing works in reverse—decrementing a register that contains 0 gives 255. This characteristic of ML, which might seem a limitation, is used to advantage in Spritemove, which just keeps incrementing and decrementing the vertical sprite registers blindly. You know ML won't let the program

exceed the safe 0–255 range, which conveniently enough the sprites also use for vertical location.

Animating the Propellers

You define a sprite's shape by pointing it to a block of shape information that you've placed in memory beforehand. To rotate the props, these related shapes are simply flipped through, much as the birds are made to flap their wings. Spritemove points all eight sprites to successive sets of shape data, which were stored when the program sets up. Since the props are bilaterally symmetrical, memory is saved by flipping through a series of only four shapes. Yet the effect is of an eight-position rotation.

Just as the computer looks in a special place to find the address of its interrupt routine, Spritemove checks and changes a special location for the current shape pointer, location 828 ($033C).

Other memory locations in the cassette buffer are used to store things for the ML routines. As you've seen, locations 832 and 833 hold values received from the joystick. Location 842 holds the home flag—the Birdmove routine stores a value of 1 here if the bird reaches home, a 0 otherwise.

Passing ML Values to BASIC

This is an example of how to use variables in machine language and pass information back and forth from ML to BASIC sections of your program. In BASIC, of course, you might use a variable, such as HOME, and say that HOME=1 when home is reached, making sure that HOME=0 at all other times. But ML doesn't recognize names—just numbers inside memory locations. In Spritemove, then, a memory location (842) is chosen to represent the condition of the home flag. All that needs to be done is to store a 1 in that location as a signal when home is reached.

Line 3 of the BASIC program uses the PEEK function to check that same memory location (HM=842) for a nonzero value, branching to the BASIC HOME subroutine starting at line 20 if that condition is satisfied. Once that subroutine is executed, the flag is set back to 0 in line 24, so that the pigeon can get lost again.

Synchronizing Sound and Action

Props also creates its filtered and ring-modulated sound effects by passing values from ML to BASIC. When the bird flies around the screen, a soft musical tone is heard; it changes constantly in relation to screen position. The sound is begun in line 2 by turning on voice 1 (POKEW1,17). In line 6, the pitch of voice 2 is changed by PEEKing into location 251, which, you'll recall, is used by Birdmove to store the bird's screen location. In this simple way, the bird's sound effect is linked to its action.

Voice 2 is always on during the game, set to the noise waveform to make a swooshing sound. The effect of fading in and out is created, not with the volume control (which affects all three voices equally), but with a filter, which can be set to affect any or all of the voices at a given moment. In line 1003, location 54296 is POKEd with a value of 47. Besides volume, this register lets you select the type of filter you want. Starting with a value of 15 (for maximum volume in all voices), 32 was then added to turn on bit 5 of that register to activate the *bandpass* filter. 15+32=47, the value POKEd into this address. This filter cuts out all but a narrow band of frequencies in the tone of the filtered voice.

Next, the voice to be filtered is selected. Also done in line 1003, 66 is POKEd into location 54295, which filters voice 2 and sets a moderate amount of resonance. (If you've never played with filter resonance, substitute 66 with 226 in line 1003; you'll hear the more pronounced effect of maximum resonance.)

A Swoosh Is Filtered Noise

Now the filter's ready to use. Picking the noise waveform for voice 2 gives a more or less random collection of all the audible frequencies to work with. Setting the *cutoff* frequency low will *pass through* a narrow band of low frequency tones for a roaring or rumbling sound, and suppress all other frequencies. A high cutoff value produces a narrow band of hissing, high-frequency tones. To make a swooshing sound, all you need to do is change the cutoff frequency at high speed, from low to high values.

To tie this sound to the graphics action, Spritemove is allowed to change the cutoff frequency at ML speed. At the very end of Spritemove is a small subroutine that stores a value

into the filter cutoff frequency register. This value is the same one used to control the number of spaces the sprites move each 1/60 second. At higher skill levels, larger numbers are added to the cutoff frequency register, to sweep the filter from low to high more rapidly.

As with sprite positioning, incrementing this register is simple and doesn't have the danger of causing ILLEGAL QUANTITY errors. The result is a repeated low-to-high sweep in the range of 0 to 255.

Filtering Voice Three

The echoing synthesizer tones heard while pausing, or when the bird's mate changes coops, are produced by applying similar bandpass filtering to voice 3. The technique is the same—the filter cutoff frequency is swept upwards, over and over. But instead of the noise waveform, the triangle waveform is used, ring-modulated by the pitch frequencies of voice 2 (line 51).

The pitch of voice 3 is linked to the bird's screen position by using the value found in location 251. And the pitch frequency of voice 2 is also swept down over and over, in the familiar 255 to 0 range, by the Spritemove routine.

Unlike the noise waveform, which contains tones at almost every audible frequency, the triangular waveform is rich in certain harmonic frequencies and totally lacking in others. So at certain frequencies the bandpass filter cuts out almost everything, creating silence. Adding ring modulation suppresses the flutelike tone you'd otherwise get from a triangle wave and adds new harmonics for an even stranger effect.

A Two-Voice Sound Effect

One final, important difference between this and the swooshing sound is in the ADSR (Attack/Decay/Sustain/Release) envelope. For the props' sound, voice 2's sustain value is set to the maximum of 240 (line 1082). The ADSR envelope is triggered only once, in line 11050.

With maximum sustain, the tone will never actually fade out—it only seems to reach silence when the filter is set to its lowest cutoff. In contrast, voice 3's ADSR envelope is used every time the synthesizer sound plays, causing the slow, ghostly fade-out.

But you can do fancy sound filtering without mastering

machine language. Take a look at lines 11050–11058, which govern the animation and sound of wings flapping during the instruction display. Here the filter frequency is controlled from an entirely different source.

A Special Number Generator

Location 54299 (VM+3) is a register that can be made to produce four different number sequences that are handy for controlling sound. It can generate a sweep of value from 0 to 255. Or it can sweep the range up and down. It can generate random numbers and can also flip back and forth from 0 to 255 at varying rates.

You choose *which* number sequence you want by selecting one of the four waveforms for voice 3. You control the rate at which the numbers change within the sequence by setting the frequency of voice 3.

For a convincing wing-flapping sound, the filter should sweep up and then back down again. Selecting the up-and-down number sequence by setting voice 3 to a value of 16 in line 11050 does this. To time it to the beating of the birds' wings, the various frequency values for voice 3 (H3 and L3) are manipulated until they're right. Note that you don't want to *hear* voice 3—you're using its pitch values only to control the output of voice 2. So W3 is POKEd to 16, which selects the triangle waveform *without* turning on the gate bit (which would make the voice audible).

Once you have Props running, you can learn a lot about the 64's SID chip just by changing the values used in this and other sound sections.

Props

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
1 GOSUB1000:GOSUB80:GOSUB50:SYS49152        :rem 241
2 POKEW1,17:SYS49408:C=PEEK(CX):IFCTHEND=PEEK(SP+A
  (C)):GOSUB10                               :rem 122
3 IFPEEK(HM)THENGOSUB20                      :rem 222
4 FORJ=RTO20-DL:NEXT:BZ=BZ+R:IFBZ=MXTHENBZ=.:GOSUB
  50                                         :rem 247
5 IFSK>NTHENGOSUB7                           :rem 212
6 POKEH1,(PEEK(251)/5+9):POKEW1,16:GOTO2     :rem 90
7 KL=KL+R:IFKL>NTHENKL=.:POKEFAST,INT(RND(R)*M):PO
  KE845,(.)                                  :rem 238
8 IFRND(R)>PTHENPOKEFAST,V:POKE845,(.)       :rem 159
```

```
9 RETURN                                                    :rem 25
10 POKEBD,2:FORJ=RTO40:POKESP+A(C),D:POKEW1,129:SY
   S49608:POKEW1,128:NEXT                                   :rem 31
11 SC=SC-SK:IFSC<RTHENSC=.                                  :rem 179
12 POKEBD,.:POKECX,.:RETURN                                 :rem 234
13 REM LINES 7-9 = 'VARY SPEED' RTN                         :rem 146
14 REM LINES 10-12 = 'HIT PROP' RTN                         :rem 90
20 REM*** 'HOME'                                            :rem 59
21 SC=SC+SK+3:IFSC>199THENGOSUB70:GOSUB80                   :rem 107
22 POKESP+21,0:PRINTSC$"{6}SCORE:"SC                        :rem 202
23 FORJ=100TO1STEP-2:SYS49608:POKEW1,21:POKEH1,J:P
   OKEW1,20:NEXT                                            :rem 231
24 PRINTSC$"{13 SPACES}":POKEHM,0:POKECX,0:rem 171
50 REM*** 'NEW COOP'                                        :rem 48
51 POKESP+21,0:POKEW3,21:POKEH3,PEEK(251):PRINTSC$
   "{YEL}NEW COOP":PRINT"{HOME}{6}";                        :rem 50
52 FORJ=1TO24:PRINT"{RIGHT}Z{36 RIGHT}Z":NEXT
                                                            :rem 120
53 PRINT"{RIGHT}Z{36 RIGHT}Z{HOME}":POKEW1,16:POKE
   W3,20                                                    :rem 54
54 IFHF=0THENHF=1:GOTO58                                    :rem 254
55 HF=0:J=1062:FS=40*(INT(RND(1)*25))                       :rem 17
56 HI=INT((FS+J)/256):LO=(FS+J)-(HI*256):POKE843,L
   O:POKE844,HI                                             :rem 124
57 SYS49615:POKEJ+FS+L1,10:PRINTSC$"{10 SPACES}":P
   OKESP+21,255:GOSUB100:RETURN                             :rem 172
58 J=1025:FS=40*(INT(RND(1)*25))                            :rem 222
59 HI=INT((FS+J)/256):LO=(FS+J)-(HI*256):POKE843,L
   O:POKE844,HI                                             :rem 127
60 SYS49615:POKEJ+FS+L1,10:PRINTSC$"{10 SPACES}":P
   OKESP+21,255:GOSUB100:RETURN                             :rem 166
70 REM** NEXT LEVEL                                         :rem 86
71 PRINTCHR$(142):FORJ=1TO500:NEXT:PRINTCHR$(14):P
   OKESP+21,0:POKEW2,21                                     :rem 93
72 POKEW1,20:POKEFV,3 :FORK=5TO115STEP2:POKEW3,21:
   POKEBD,K:POKE646,K                                       :rem 0
73 POKEH1,K*1.4:PRINTSC$"NEXT LEVEL?":POKEH3,K/4:S
   YS49608:POKEW3,20:NEXT                                   :rem 253
74 POKEW1,20:POKEFV,66:FORJ=1TO3500:NEXT:POKEH3,10
   0:POKEBD,0:POKEW2,129:RETURN                             :rem 202
80 REM*** PICK SKILL LEVEL                                  :rem 232
81 POKESP+21,0:POKEFNA(0),32:KZ=0                           :rem 193
82 PRINTSC$"{YEL}{3 LEFT}PICK SKILL LEVEL":rem 136
83 PRINTSC$"{8}{2 DOWN}{RIGHT}(1 - 6)"                      :rem 181
84 GOSUB200                                                 :rem 126
85 IFKZ<>1THENSYS49608:GOTO84                               :rem 100
86 PRINTSC$"{3 LEFT}{16 SPACES}"                            :rem 208
87 PRINTSC$"{2 DOWN}{11 SPACES}"                            :rem 28
88 SC=0:RETURN                                              :rem 141
```

```
100 REM*** PAUSE ROUTINE                          :rem 151
101 IFFNB(.)THENRETURN                            :rem 133
102 POKEBD,13:PRINTSC$"{RVS}{YEL} PAUSING {OFF}"SC
    $"{8}{2 DOWN}SCORE:"SC                        :rem 253
103 PRINTSC$"{4 DOWN}FIRE=PLAY"SC$"{6 DOWN}ZERO=QU
    IT"                                           :rem 32
104 FORJ=1TO1500:NEXT:POKEBD,0                    :rem 97
105 SYS49608:S=S+1:IFINT(S/43)=S/43THENPOKEW3,20:P
    OKEH3,PEEK(SP+1):POKEW3,21                    :rem 170
106 IFFNB(.)=.THEN110                             :rem 167
107 GOSUB200                                      :rem 170
108 IFSK>5THENGOSUB7                              :rem 31
109 GOTO105                                       :rem 105
110 PRINTSC$"{12 SPACES}"SC$"{2 DOWN}{10 SPACES}"S
    C$"{4 DOWN}{9 SPACES}"                        :rem 127
111 PRINTSC$"{6 DOWN}{9 SPACES}"                  :rem 132
112 POKEW3,20:POKEBD,2:POKEBD,5:POKESP+31,0:POKEBD
    ,7:POKEBD,3:POKEBD,0:RETURN                   :rem 116
200 REM*** SKILL                                  :rem 115
201 G=PEEK(197)                                   :rem 46
203 IFG=35THEN300                                 :rem 210
204 IFG=56THENSK=1:DL=1:POKEFAST,1:POKEROT,4:POKE8
    29,0:POKE845,0:KZ=1:RETURN                    :rem 110
205 IFG=59THENSK=2:DL=5:POKEFAST,2:POKEROT,3:POKE8
    29,0:POKE845,0:KZ=1:RETURN                    :rem 119
206 IFG=8THENSK=3:DL=10:POKEFAST,3:POKEROT,2:POKE8
    29,0:POKE845,0:KZ=1:RETURN                    :rem 111
207 IFG=11THENSK=4:DL=15:POKEFAST,4:POKEROT,1:POKE
    829,0:POKE845,0:KZ=1:RETURN                   :rem 160
208 IFG=16THENSK=5:DL=19:POKEFAST,5:POKEROT,0:POKE
    829,0:POKE845,0:KZ=1:RETURN                   :rem 171
209 IFG=19THENSK=6:DL=19:POKEFAST,6:POKEROT,0:POKE
    829,0:POKE845,0:KZ=1:RETURN                   :rem 177
210 RETURN                                        :rem 115
300 REM*** QUIT                                   :rem 56
301 POKE49221,2:POKE49228,0:POKE829,0:POKE845,0:PO
    KE646,7                                       :rem 43
302 FORJ=SP+1TOSP+15STEP2:POKEJ,123:NEXT:POKE214,2
    4:PRINT:FORJ=1TO24                            :rem 112
303 PRINT:FORK=1TO20:NEXTK,J:POKEW3,20            :rem 189
304 POKEW3,21:PRINTSC$"BYE BYE . . .":FORJ=1TO2400
    :NEXT:POKESP+21,0                             :rem 159
305 FORJ=L1TOVM+3:POKEJ,0:NEXT:SYS64738           :rem 244
1000 REM*** INITIALIZE                            :rem 21
1001 REM**ML RTNS: SPRITEMOVE=49152:BIRDMOVE=49408
     :FLAPWING=49608:PUTMATE=49615                :rem 182
1002 POKE53272,21:POKE53281,0:BD=53280:POKEBD,0
                                                  :rem 150
```

```
1003 FORJ=54272TO54296:POKEJ,0:NEXT:POKE54296,47:P
     OKE54295,66                                  :rem 127
1004 GOSUB10000:GOSUB12000:FAST=49221:ROT=49228:R=
     1:N=5:M=7:P=(.98):V=28:MX=200                :rem 138
1006 REM*** SET UP SPRITES                        :rem 228
1008 POKE53275,255:SP=53248:CX=SP+31:POKECX,0
                                                  :rem 138
1012 FORJ=2040TO2047:POKEJ,221:NEXT               :rem 85
1014 B=80:FORJ=SP+1TOSP+15STEP2:POKEJ,B:B=B+20:NEX
     T                                            :rem 187
1016 POKESP+23,0:POKESP+29,24                     :rem 186
1018 POKESP,40:POKESP+2,70:POKESP+4,100:POKESP+6,1
     30:POKESP+8,188:POKESP+10,245                :rem 24
1020 POKESP+12,20:POKESP+14,48:POKESP+16,192
                                                  :rem 75
1022 POKESP+39,7:POKESP+40,3:POKESP+41,4:POKESP+42
     ,13                                          :rem 180
1024 POKESP+43,7:POKESP+44,3:POKESP+45,13:POKESP+4
     6,4                                          :rem 189
1025 REM ML VARBLS, ETC IN CASET BUFR             :rem 77
1026 POKE828,221:REM START SPRITE PNTRS           :rem 117
1028 POKE829,1:REM SPRITE ROTATE COUNTR           :rem 153
1030 POKE830,40:POKE831,0:REM LINE VAL            :rem 168
1032 POKE832,0:REM R/L JYSTK FLAG                 :rem 136
1034 POKE833,0:REM U/D JYSTK FLAG                 :rem 134
1036 POKE834,91:REM BIRD CHAR VAL                 :rem 110
1038 POKE835,1:POKE836,4:REM UP LIMIT             :rem 164
1040 POKE837,230:POKE838,6:REM DN LIMIT           :rem 244
1042 POKE841,1:REM SPRITE MOVE COUNTR             :rem 247
1044 HM=842:POKEHM,0:REM 'HOME' FLAG              :rem 66
1046 POKE843,95:POKE844,5:REM MATE'S FIRST LOCATIO
     N                                            :rem 61
1048 POKE251,144:POKE252,4:REM FIRST BIRD LOCATION
      - ZERO PAGE                                 :rem 101
1049 POKE845,0:REM FILTER CUT COUNTR              :rem 165
1050 DIMA(129):A(1)=1:A(2)=3:A(4)=5:A(8)=7:A(16)=9
     :A(32)=11:A(64)=13:A(128)=15                 :rem 235
1052 HF=0:BZ=0                                     :rem 6
1054 DEFFNA(X)=((PEEK(252)*256)+PEEK(251))         :rem 5
1056 DEFFNB(X)=(PEEK(56320)AND16)                  :rem 88
1058 SC$="{HOME}{15 RIGHT}{10 DOWN}"               :rem 121
1060 REM*** SOUND VRBLS                            :rem 59
1064 L1=54272:H1=L1+1:W1=L1+4:W2=L1+11             :rem 55
1068 L3=L1+14:H3=L1+15:W3=L1+18                    :rem 170
1070 FH=L1+22:FV=L1+23:VM=L1+24                    :rem 231
1074 REM*** SETUP SOUNDS                          :rem 155
1076 POKEL1+5,64:POKEL1+12,15:POKEL1+19,12:rem 128
1078 POKEL1+7,255:POKEL1+8,255                     :rem 183
1082 POKEL1+6,0:POKEL1+13,240:POKEL1+20,12:rem 109
1084 POKEFH,90                                     :rem 31
```

```
1100 GOSUB11000                                    :rem 52
3000 REM*** DRAW COOPS                             :rem 215
3004 FORJ=1984TO1024STEP-40:POKEJ+L1,10:POKEJ,90:P
     OKEJ+L1+1,10:POKEJ+1,90                       :rem 90
3006 POKEJ+L1+38,10:POKEJ+38,90:POKEJ+L1+39,10:POK
     EJ+39,90:NEXT                                 :rem 194
3008 FORJ=1024TO1984STEP40:POKEJ+L1,13:POKEJ+L1+1,
     13:POKEJ+L1+38,13                             :rem 204
3010 POKEJ+L1+39,13:NEXT:RETURN                    :rem 159
10000 REM*** TITLE                                 :rem 213
10001 PRINT"{CLR}{6}"CHR$(142);                    :rem 19
10002 PRINT"{18 RIGHT}{RVS}£ {*}"                  :rem 104
10003 PRINT"{17 RIGHT}{RVS}£{3 SPACES}{*}" :rem 76
10004 PRINT"{17 RIGHT}{RVS}{2 SPACES}P{2 SPACES}"
                                                   :rem 21
10005 PRINT"{17 RIGHT}{RVS}{2 SPACES}R{2 SPACES}"
                                                   :rem 24
10006 PRINT"{17 RIGHT}{RVS}{2 SPACES}O{2 SPACES}"
                                                   :rem 22
10007 PRINT"{17 RIGHT}{RVS}{2 SPACES}P{2 SPACES}"
                                                   :rem 24
10008 PRINT"{17 RIGHT}{*}{RVS} S {OFF}£"    :rem 54
10009 PRINT"{18 RIGHT}{RVS}{3 SPACES}{OFF}"
                                                   :rem 121
10010 PRINT"{18 RIGHT}{*}{RVS} {OFF}£"      :rem 249
10011 PRINT"{19 RIGHT}{RVS} {OFF}"          :rem 143
10012 PRINT"{17 RIGHT}{RVS}£{3 SPACES}{*}" :rem 76
10013 PRINT"{6 RIGHT}{RVS}£{8 SPACES}{*}{RIGHT} U
      CI {RIGHT}£{8 SPACES}{*}"                    :rem 49
10014 PRINT"{6 RIGHT}{RVS}{2 SPACES}PROPS
      {5 SPACES}- -{5 SPACES}PROPS{2 SPACES}"
                                                   :rem 105
10015 PRINT"{6 RIGHT}{*}{RVS}{8 SPACES}{OFF}£
      {RIGHT}{RVS} JCK {RIGHT}{OFF}{*}{RVS}
      {8 SPACES}{OFF}£"                             :rem 4
10016 PRINT"{17 RIGHT}{*}{RVS}{3 SPACES}{OFF}£"
                                                   :rem 226
10017 PRINT"{19 RIGHT}{RVS} {OFF}"          :rem 149
10018 PRINT"{18 RIGHT}{RVS}£ {*}"           :rem 111
10019 PRINT"{18 RIGHT}{RVS}{3 SPACES}{OFF}"
                                                   :rem 122
10020 PRINT"{17 RIGHT}{RVS}£ P {*}"         :rem 155
10021 PRINT"{17 RIGHT}{RVS}{2 SPACES}R{2 SPACES}"
                                                   :rem 22
10022 PRINT"{17 RIGHT}{RVS}{2 SPACES}O{2 SPACES}"
                                                   :rem 20
10023 PRINT"{17 RIGHT}{RVS}{2 SPACES}P{2 SPACES}"
                                                   :rem 22
```

```
10024 PRINT"{17 RIGHT}{RVS}{2 SPACES}S{2 SPACES}"
                                                :rem 26
10025 PRINT"{17 RIGHT}{*3}{RVS}{3 SPACES}{OFF}£"
                                               :rem 226
10026 PRINT"{18 RIGHT}{*3}{RVS} {OFF}£{HOME}{CYN}R
      EADING{2 DOWN}{3 LEFT}DATA..."            :rem 73
10027 Q$="{28 RIGHT}":POKE214,19:PRINT         :rem 121
10028 PRINTQ$"JOYSTICK":PRINTQ$"{2 DOWN}{2 RIGHT}I
      N PORT 2"                                :rem 155
10999 RETURN                                   :rem 236
11000 REM*** INSTRUCTIONS                       :rem 9
11002 PRINT"{CLR}":POKE53272,31                :rem 84
11004 R$="{4 RIGHT}"                           :rem 97
11006 PRINT"{6}{4 RIGHT}ZZZZZZZZZZZZZZZZZZZZZZZZZZ
      ZZZ"                                     :rem 136
11007 PRINT"{4 RIGHT}ZZ{25 RIGHT}ZZ"          :rem 123
11008 PRINTR$"ZZ{YEL} YOU ARE A PIGEON {WHT}X{YEL}
       LOST {6}ZZ"                             :rem 245
11010 PRINTR$"ZZ{YEL} IN A DANGEROUS SKY FULL {6}Z
      Z"                                       :rem 131
11012 PRINTR$"ZZ{YEL} OF WHIRLING PROPELLORS. {6}Z
      Z"                                       :rem 20
11013 PRINT"{4 RIGHT}ZZ{25 RIGHT}ZZ"          :rem 120
11014 PRINTR$"ZZ{CYN} BACK AT THE COOP, YOUR
      {2 SPACES}{6}ZZ"                         :rem 17
11016 PRINTR$"ZZ{CYN} MATE WAITS FAITHFULLY-- {3}X
      {6}Z"                                    :rem 245
11017 PRINT"{4 RIGHT}ZZ{25 RIGHT}ZZ"          :rem 124
11018 PRINTR$"ZZ{3} WATCH OUT FOR THE PROPS {6}ZZ"
                                               :rem 164
11020 PRINTR$"ZZ{3} WHILE YOU FLY FOR HOME. {6}ZZ"
                                               :rem 113
11021 PRINT"{4 RIGHT}ZZ{25 RIGHT}ZZ"          :rem 119
11022 PRINTR$"ZZ{YEL} HIT FIRE BUTTON DURING
      {2 SPACES}{6}ZZ"                         :rem 140
11024 PRINTR$"ZZ{YEL} SCORE DISPLAY TO PAUSE, {6}Z
      Z"                                       :rem 189
11026 PRINTR$"ZZ{YEL} OR CHANGE SKILL LEVEL.
      {2 SPACES}{6}ZZ"                          :rem 76
11027 PRINT"{4 RIGHT}ZZ{25 RIGHT}ZZ"          :rem 125
11028 PRINTR$"ZZ{CYN} HIT ZERO KEY DURING ANY {6}Z
      Z"                                       :rem 162
11030 PRINTR$"ZZ{CYN} PAUSE TO{2 SPACES}QUIT PLAYI
      NG. {6}ZZ"                               :rem 130
11032 PRINT"{4 RIGHT}ZZ{25 RIGHT}ZZ"          :rem 121
11038 PRINTR$"ZZ{YEL} PRESS FIRE BUTTON NOW
      {3 SPACES}{6}ZZ"                         :rem 102
11040 PRINTR$"ZZ{YEL} TO CHOOSE SKILL{2 SPACES}LEV
      EL. {6}ZZ"                               :rem 101
```

```
11042 PRINT"{4 RIGHT}ZZ{25 RIGHT}ZZ"          :rem 122
11044 PRINT"{4 RIGHT}ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
      "                                       :rem 241
11050 POKEW3,16:POKEH3,1:POKEL3,90:POKEW2,129
                                              :rem 187
11052 SYS49608:POKEFH,PEEK(VM+3)/2:IFFNB(0)THEN110
      52                                       :rem 70
11054 PRINT"{HOME}"CHR$(142):FORJ=1TO200:NEXT:PRIN
      TCHR$(14)                               :rem 205
11056 PRINT"{CLR}":POKEW3,20:SYS49920:REM WHITE CO
      L MEMORY                                :rem 121
11058 POKE843,254:POKE844,3:POKE251,220:POKEFH,2:R
      ETURN                                    :rem 44
12000 REM** SPRITE SHAPE DATA                 :rem 141
12001 IFPEEK(14081)=8ANDPEEK(15065)=195THENRETURN
                                              :rem 161
12287 CK=0:FORJ=14080TO14142:READQ:CK=CK+Q:POKEJ,Q
      :NEXT                                   :rem 162
12288 DATA 0,8,0,0,24,0,0,56,0,0,56,0,0,56,0,0,24,
      0,0,16,0,0,16,0,0,0,0                    :rem 58
12289 DATA 0,24,0,0,60,0,0,24,0,0,0,0,0,8,0,0,8,0,
      0,24,0,0,28,0,0,28,0,0,28               :rem 246
12290 DATA 0,0,24,0,0,16,0                    :rem 221
12351 FORJ=14144TO14206:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                              :rem 103
12352 DATA 0,0,0,16,0,0,28,0,0,14,0,0,15,0,0,7,128
      ,0,3,128,0,1,192,0,0,128                :rem 215
12353 DATA 0,0,24,0,0,60,0,0,24,0,0,2,0,0,3,128,0,
      1,192,0,1,224,0,0,240,0,0               :rem 238
12354 DATA 112,0,0,56,0,0,8,0,0,0              :rem 44
12415 FORJ=14208TO14270:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                              :rem 106
12416 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      ,0,0,0,0,0                              :rem 232
12417 DATA 0, 0,124,24, 62,255,189,255,124,24, 62,
       0,0, 0,0,0,0,0,0,0,0,0,0                :rem 37
12418 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0    :rem 82
12479 FORJ=14272TO14334:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                              :rem 118
12480 DATA 0,0,0,0,0,8,0,0,56,0,0,112,0,0,240,0,1,
      224,0,1,192,0,3,128,0,2                 :rem 152
12481 DATA 0,0,24,0,0,60,0,0,24,0,0,64,0,1,192,0,3
      ,128,0,7,128,0,15                       :rem 145
12482 DATA 0,0,14,0,0,28,0,0,16,0,0,0,0,0 :rem 157
12500 IFCK<>5053THENPRINT"{3 DOWN}ERROR IN DATA LI
      NES 12000-12482":END                    :rem 119
49000 REM*** POKE PROPMOVE RTN                :rem 250
49151 CK=0:FORJ=49152TO49310:READQ:CK=CK+Q:POKEJ,Q
      :NEXT                                   :rem 175
```

```
49152 DATA 120, 173, 21, 3, 201, 234      :rem 235
49158 DATA 208, 13, 169, 33, 141, 20      :rem 253
49164 DATA 3, 169, 192, 141, 21, 3        :rem 153
49170 DATA 76, 31, 192, 169, 49, 141       :rem 11
49176 DATA 20, 3, 169, 234, 141, 21       :rem 200
49182 DATA 3, 88, 96, 169, 0, 141         :rem 118
49188 DATA 73, 3, 238, 1, 208, 206        :rem 161
49194 DATA 3, 208, 238, 5, 208, 206       :rem 210
49200 DATA 7, 208, 238, 9, 208, 206       :rem 206
49206 DATA 11, 208, 238, 13, 208, 206      :rem 42
49212 DATA 15, 208, 238, 73, 3, 173       :rem 205
49218 DATA 73, 3, 201, 2, 208, 222        :rem 144
49224 DATA 173, 61, 3, 201, 5, 240        :rem 142
49230 DATA 6, 238, 61, 3, 76, 49           :rem 63
49236 DATA 234, 173, 60, 3, 201, 224      :rem 246
49242 DATA 208, 5, 169, 220, 141, 60      :rem 250
49248 DATA 3, 141, 248, 7, 141, 249       :rem 212
49254 DATA 7, 141, 250, 7, 141, 251       :rem 199
49260 DATA 7, 141, 252, 7, 141, 253       :rem 200
49266 DATA 7, 141, 254, 7, 141, 255       :rem 210
49272 DATA 7, 238, 60, 3, 169, 0           :rem 59
49278 DATA 141, 61, 3, 173, 69, 192       :rem 218
49284 DATA 10, 10, 10, 10, 10, 141        :rem 124
49290 DATA 78, 3, 24, 173, 77, 3           :rem 66
49296 DATA 109, 78, 3, 141, 77, 3         :rem 119
49302 DATA 141, 22, 212, 206, 8, 212      :rem 236
49308 DATA 76, 49, 234                    :rem 237
49350 IFCK<>17626THENPRINT"{3 DOWN}ERROR IN DATA L
      INES 49000-49308":END               :rem 206
49400 REM*** POKE BIRDMOVE RTN            :rem 222
49407 CK=0:FORJ=49408TO49643:READQ:CK=CK+Q:POKEJ,Q
      :NEXT                               :rem 192
49408 DATA 160, 0, 169, 32, 145, 251      :rem 253
49414 DATA 165, 251, 133, 253, 165, 252   :rem 150
49420 DATA 133, 254, 173, 0, 220, 162      :rem 35
49426 DATA 0, 74, 176, 1, 136, 74         :rem 110
49432 DATA 176, 1, 200, 74, 176, 1        :rem 151
49438 DATA 202, 74, 176, 1, 232, 74       :rem 210
49444 DATA 142, 64, 3, 140, 65, 3         :rem 102
49450 DATA 173, 64, 3, 201, 1, 240        :rem 142
49456 DATA 18, 201, 255, 240, 3, 76       :rem 208
49462 DATA 116, 193, 165, 253, 208, 2      :rem 54
49468 DATA 198, 254, 198, 253, 76, 73      :rem 85
49474 DATA 193, 230, 253, 208, 2, 230      :rem 47
49480 DATA 254, 160, 0, 177, 253, 201      :rem 45
49486 DATA 32, 208, 3, 76, 91, 193        :rem 171
49492 DATA 201, 90, 208, 14, 76, 108         :rem 3
49498 DATA 193, 165, 253, 133, 251, 165   :rem 166
49504 DATA 254, 133, 252, 76, 116, 193    :rem 105
```

```
49510 DATA 169, 1, 141, 74, 3, 96          :rem 109
49516 DATA 165, 251, 133, 253, 165, 252    :rem 153
49522 DATA 133, 254, 173, 65, 3, 201       :rem 250
49528 DATA 1, 240, 39, 201, 255, 240       :rem 250
49534 DATA 3, 76, 200, 193, 162, 0         :rem 151
49540 DATA 165, 253, 208, 2, 198, 254       :rem 59
49546 DATA 198, 253, 232, 224, 40, 208     :rem 107
49552 DATA 243, 24, 165, 253, 205, 67       :rem 56
49558 DATA 3, 165, 254, 237, 68, 3         :rem 172
49564 DATA 176, 34, 240, 32, 144, 38         :rem 6
49570 DATA 162, 0, 230, 253, 208, 2        :rem 195
49576 DATA 230, 254, 232, 224, 40, 208      :rem 98
49582 DATA 245, 24, 165, 253, 205, 69       :rem 63
49588 DATA 3, 165, 254, 237, 70, 3         :rem 168
49594 DATA 144, 4, 240, 2, 176, 8          :rem 112
49600 DATA 165, 253, 133, 251, 165, 254    :rem 149
49606 DATA 133, 252, 160, 0, 173, 66       :rem 253
49612 DATA 3, 145, 251, 173, 75, 3         :rem 155
49618 DATA 133, 253, 173, 76, 3, 133         :rem 5
49624 DATA 254, 173, 66, 3, 145, 253          :rem 8
49630 DATA 201, 96, 240, 4, 238, 66        :rem 209
49636 DATA 3, 96, 169, 91, 141, 66         :rem 176
49642 DATA 3, 96                           :rem 241
49650 IFCK<>33160THENPRINT"{3 DOWN}ERROR IN DATA L
      INES 49400-49642":END                :rem 205
49662 REM*** POKE COPYCHAR RTN             :rem 233
49663 CK=0:FORJ=49664TO49704:READQ:CK=CK+Q:POKEJ,Q
      :NEXT                                :rem 198
49664 DATA 169,0,133,4,169,208,133,5       :rem 252
49672 DATA 169,0,133,6,169,56,133,7        :rem 208
49680 DATA 162,0,160,0,177,4,145,6         :rem 140
49688 DATA 200,192,255,208,247,230,5,230   :rem 193
49696 DATA 7,232,224,16,208,236,160,0,96   :rem 196
49700 IFCK<>4894THENPRINT"{3 DOWN}ERROR IN DATA LI
      NES 49662-49696":END                 :rem 184
49918 REM*** POKE WHITEMEM RTN             :rem 244
49919 CK=0:FORJ=49920TO49939:READQ:CK=CK+Q:POKEJ,Q
      :NEXT                                :rem 207
49920 DATA 162, 0, 169, 1, 157, 0          :rem 101
49926 DATA 216, 157, 0, 217, 157, 0        :rem 209
49932 DATA 218, 157, 0, 219, 232, 208       :rem 54
49938 DATA 241, 96                          :rem 93
49950 IFCK<>2607THENPRINT"{3 DOWN}ERROR IN DATA LI
      NES 49918-49938":END                 :rem 184
62000 REM* COPY CHAR SET TO 14336          :rem 239
62002 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND
      251                                   :rem 27
62004 SYS49664                               :rem 8
62006 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
                                           :rem 237
```

```
62400 REM** POKE NEW CHARACTER DATA **      :rem 72
62500 CK=0:FORJ=15064TO15071:READQ:CK=CK+Q:POKEJ,Q
      :NEXT                                 :rem 160
62501 DATA 0,195,102,60,24,0,0,0            :rem 10
62502 FORJ=15072TO15079:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                            :rem 116
62503 DATA 0,0,195,126,24,0,0,0             :rem 220
62504 FORJ=15080TO15087:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                            :rem 116
62505 DATA 0,0,66,255,153,0,0,0             :rem 225
62506 FORJ=15088TO15095:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                            :rem 125
62507 DATA 0,0,0,90,255,129,0,0             :rem 227
62508 FORJ=15096TO15103:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                            :rem 116
62509 DATA 0,0,0,24,126,195,0,0             :rem 226
62510 FORJ=15104TO15111:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                             :rem 98
62511 DATA 0,0,0,24,60,102,195,129          :rem 119
62512 FORJ=15056TO15063:READQ:CK=CK+Q:POKEJ,Q:NEXT
                                            :rem 112
62513 DATA 255, 60, 24, 24, 24, 24, 60,255:rem 235
62600 IFCK<>3255THENPRINT"{3 DOWN}ERROR IN DATA LI
      NES 62000-62513":END                  :rem 132
63000 RETURN                                :rem 217
```

Worm of Bemer

Stephen D. Fultz

64 Translation by Kevin Martin

Nerm the Worm is lost in Bemer Castle and needs your help to get home. You must guide him through 11 rooms and help him find magic mushrooms to eat along the way. The journey is a navigator's nightmare, because you never know where the next mushroom will grow, and if Nerm hits a wall or gets trapped by his tail, he loses one of his lives. Optional keyboard controls.

"Worm of Bemer" is a fast-paced arcade game in which Nerm the Worm travels through rooms eating magic mushrooms. Nerm is lost in Bemer Castle and wants to return home. You're Nerm's guide, making sure he finds mushrooms so he can keep up his strength for the journey. A joystick plugged into port 2 gives Nerm directions. (An option to use the keyboard is provided below.) After eating five mushrooms in a room, Nerm can exit to the next room. But the trip's not over.

You must guide Nerm through ten more rooms before he finds home. The game starts with four lives for Nerm, but if he touches anything besides a mushroom, he loses one of those lives.

At the top of the screen you'll see the current score, what room Nerm is in, how many mushrooms he must eat to open the exits, and how many lives he has left (including his current life). You get 100 points, plus bonus points, for every mushroom Nerm eats. Nerm also gets a bonus life after completing the first two rooms and another for every third room thereafter.

Getting Nerm Home

Here are a couple of hints to help you get Nerm home. First of all, try to leave room between the walls and Nerm's tail, for if you block off the exits with his tail, you'll have trouble getting to the next room. Sometimes it will seem as if you've surrounded the mushroom with Nerm's tail; all you have to do is move Nerm to another section of the screen and wait for the tail to shorten enough so that you can get to the mushroom.

Nerm Can Use the Keyboard, Too

If you don't have a joystick or just want to use the keyboard instead, there are some changes to the program that allow you to use four keys to control Nerm's moves. By PEEKing location 197, you can read the keyboard. It's relatively simple, and you can even modify the changes to select your own keyboard pattern.

To switch from joystick to keyboard control, change these lines:

```
100 S=PEEK(197):FORD=1TOSP:NEXT
110 IFS=53THENDX=1:DY=0:DI=1:IFOD=2THENDX=-1:DY=0:
    DI=2
120 IFS=45THENDX=-1:DY=0:DI=2:IFOD=1THENDX=1:DY=0:
    DI=1
130 IFS=46THENDY=-1:DX=0:DI=4:IFOD=3THENDI=3:DY=1:
    DX=0
140 IFS=55THENDY=1:DX=0:DI=3:IFOD=4THENDI=4:DY=-1:
    DX=0
7736 PRINT"{WHT}{2 DOWN}{RIGHT}PRESS SPACE BAR TO
     {SPACE}PLAY AGAIN, Q=QUIT"
7780 S=PEEK(197):IFS=60THEN5011
10025 PRINT"{13}{3 DOWN}{10 RIGHT}USE KEYBOARD CONT
      ROLS"
10030 PRINT"{6}{6 DOWN}{10 RIGHT}HIT SPACE BAR TO
      {SPACE}START"
10045 S=PEEK(197):IFS=60THENRETURN
```

Once you've altered the program, you move Nerm up by pressing the @ key, down with the / key, to the right with the = key, and to the left with the : key.

If you want to change the keys that control Nerm's directions, decide which keys you'd like, then type in the following one-line program. Run it and press the keys you'd like to use.

```
10 PRINT PEEK(197):GOTO 10
```

The number on the screen is the value stored in location 197. Place the value for your *up* key in place of the 46 in line 130 above. The *down* key's value should replace the 55 in line 140. Insert the *right* key value in line 110 (replacing the 53), and the *left* key value instead of the 45 in line 120. That's it.

Adding More Features

This game uses some of the special features of the Commodore 64, including custom characters, sound, and scrolling

screens. For instance, characters were redesigned for the walls, the mushroom, and the body of Nerm. The original character set was copied to a location in RAM not used by BASIC, and the new characters were added.

Since Worm of Bemer is written entirely in BASIC, it's relatively easy to modify. You can learn a lot about programming and games by changing the actions and settings of published programs such as Worm of Bemer. Some features you might add include a routine to save the high score to disk, adding more players, or having Nerm go to a different room depending on which exit he takes. Simpler enhancements would be changing the number of mushrooms that Nerm must eat to open the exits or changing his speed.

Worm of Bemer

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
1 POKE52,48:POKE56,48:CLR                          :rem 230
2 POKE53270,PEEK(53270)AND15                        :rem 62
5 POKE53280,0:POKE53281,0                          :rem 138
10 GOTO 5000                                        :rem 95
100 S=PEEK(56320)AND15:FORD=1TOSP:NEXT              :rem 98
110 IFS=7ORS=6ORS=5THENDX=1:DY=0:DI=1:IFOD=2THENDX
    =-1:DY=0:DI=2                                   :rem 18
120 IFS=11ORS=10ORS=9THENDX=-1:DY=0:DI=2:IFOD=1THE
    NDX=1:DY=0:DI=1                                :rem 108
130 IFS=14THENDY=-1:DX=0:DI=4:IFOD=3THENDI=3:DY=1:
    DX=0                                           :rem 122
140 IFS=13THENDY=1:DX=0:DI=3:IFOD=4THENDI=4:DY=-1:
    DX=0                                           :rem 123
145 PO=1024+XA+YA*40:OD=DI:POKEPO,42:POKEPO+SO,L1
                                                     :rem 3
150 XA=XA+DX:YA=YA+DY:L=LEN(XA$):XA$=XA$+CHR$(XA):
    YA$=YA$+CHR$(YA)                                :rem 0
155 Z=PEEK(1024+XA+YA*40):IFZ<>32THEN200           :rem 73
161 POKESO+1,40:POKESO+4,17                        :rem 83
162 PO=1024+XA+YA*40:POKEPO,42:POKEPO+SO,10:POKESO
    +4,16:IFL<WOTHEN100                             :rem 3
190 PO=1024+ASC(XA$)+40*ASC(YA$):LL=LEN(XA$)-1:XA$
    =RIGHT$(XA$,LL)                                :rem 238
191 POKEPO,32:POKEPO+SO,0                           :rem 43
195 YA$=RIGHT$(YA$,LL):GOTO100                      :rem 19
200 POKESO+1,20:POKESO+4,17:POKESO+4,16             :rem 72
201 PO=1024+XA+40*YA:POKEPO,42:POKEPO+SO,10:GOSUB6
    600:IFZ<>BUTHEN260                             :rem 202
210 WO=WO+15+3*LO:IFWO>240THENWO=240               :rem 187
```

```
220 XX=INT(RND(1)*36+2):X=INT(RND(1)*18+3):IFPEEK(
    1024+XX+40*X)<>32THEN220                    :rem 9
221 SC=SC+100+LO*7:POKESO+4,55:POKESO+4,17:rem 223
225 HI=HI-1:GOSUB6600:IFHI>0THEN229             :rem 112
226 PO=1024+20+40*2:POKEPO,160:POKEPO+SO,0:PO=1024
    +20+21*40:POKEPO,160                        :rem 2
227 POKEPO+SO,0:PO=1024+40*12:POKEPO,160:POKEPO+SO
    ,0:POKESO+1,100:POKESO+4,17                 :rem 169
228 PO=1024+40*12+39:POKEPO,160:POKEPO+SO,0:GOTO10
    0                                           :rem 242
229 PO=1024+XX+X*40:POKEPO,BUG:POKEPO+SO,13
                                                :rem 163
230 GOTO100                                     :rem 95
260 IFZ<>160ANDLI>1THENGOSUB7500:GOTO290        :rem 242
265 IFZ<>160THEN7500                            :rem 146
270 POKESO+1,90:POKESO+4,17                     :rem 89
271 GOSUB7000:PRINT"{HOME}{24 DOWN}"            :rem 151
275 FORDE=1TO24:PRINT:POKESO+1,DEL:POKESO+4,17:NEX
    T:POKESO+4,16                               :rem 40
280 LO=LO+1:WO=5:IFLO=12THEN1200                :rem 177
281 L1=L1+1:IFL1>15THENL1=11                    :rem 99
285 IFLO>EXTHENGOSUB9100                        :rem 29
287 PRINT"{CLR}":GOSUB 4100                     :rem 132
290 GOSUB6600                                   :rem 231
300 ONLO GOTO5020,400,500,550,600,700,800,450,550,
    1000,1100{A},1200                           :rem 129
399 GOTO5015                                    :rem 169
400 REM SECOND SCREEN                           :rem 244
410 FORI=1024+5+10*40TO1024+35+10*40:POKEI,35:POKE
    I+SO,9:NEXT                                 :rem 253
420 GOTO5020                                    :rem 150
450 REM SCREEN                                  :rem 61
460 FORI=1024+5+10*40TO1024+35+10*40:POKEI,35:POKE
    I+SO,9:NEXT                                 :rem 2
465 FORI=1024+18+5*40TO1024+18+20*40STEP40:POKEI,3
    5:POKEI+SO,9:NEXT                           :rem 177
470 GOTO5020                                    :rem 155
500 REM THE FOURTH SCREEN                       :rem 242
510 FORI=1024+5+5*40TO1024+35+5*40:POKEI,35:POKEI+
    SO,9:NEXT                                   :rem 166
520 FORI=1024+5+18*40TO1024+35+18*40:POKEI,35:POKE
    I+SO,9:NEXT                                 :rem 15
530 GOTO5020                                    :rem 152
550 REM FRAME 5                                 :rem 30
560 FORI=1024+7+6*40TO1024+33+6*40:POKEI,35:POKEI+
    SO,9:NEXT                                   :rem 173
575 FORI=1024+18+7*40TO1024+18+20*40STEP40:POKEI,3
    5:POKEI+SO,9:NEXT                           :rem 181
580 GOTO5020                                    :rem 157
```

```
600 REM FRAME 6                                    :rem 27
610 FORI=1024+1+10*40TO1024+18+10*40:POKEI,35:POKE
    I+SO,9:NEXT                                    :rem 252
615 FORI=1024+22+10*40TO1024+38+10*40:POKEI,35:POK
    EI+SO,9:NEXT                                   :rem 54
620 GOTO5020                                       :rem 152
700 REM FRAME 7                                    :rem 29
710 FORJ=6TO14:FORI=1024+6+J*40TO1024+12+J*40:POKE
    I,35:POKEI+SO,9:NEXT                           :rem 180
715 FORI=1024+20+J*40TO1024+32+J*40:POKEI,35:POKEI
    +SO,9:NEXT:NEXT                                :rem 122
720 GOTO 5020                                      :rem 153
800 REM FRAME 8                                    :rem 31
811 FORI=1024+1+8*40TO1024+18+8*40:POKEI,35:POKEI+
    SO,9:NEXT                                      :rem 173
812 FORI=1024+1+15*40TO1024+18+15*40:POKEI,35:POKE
    I+SO,9:NEXT                                    :rem 10
813 FORI=1024+15+12*40TO1024+38+12*40:POKEI,35:POK
    EI+SO,9:NEXT                                   :rem 60
890 GOTO5020                                       :rem 161
1000 FORJ=4TO19:FORI=1024+1+J*40TO1024+37+J*40:POK
     EI,35:POKEI+SO,9:NEXT:NEXT                    :rem 91
1005 FORJ=4TO19:FORI=1024+1+J*40TO1024+22+J*40:POK
     EI,32:POKEI+SO,9:NEXT:NEXT                    :rem 87
1010 GOTO5020                                      :rem 194
1100 FORJ=4TO19:FORI=1024+1+J*40TO1024+37+J*40:POK
     EI,35:POKEI+SO,9:NEXT:NEXT                    :rem 92
1105 FORJ=4TO19:FORI=1024+1+J*40TO1024+30+J*40:POK
     EI,32:POKEI+SO,9:NEXT:NEXT                    :rem 87
1110 GOTO400                                       :rem 144
1200 REM YOU WIN                                   :rem 146
1205 FORZZ=1TO3                                    :rem 167
1210 PRINT"{CLR}{8 DOWN}{14 RIGHT}NERM'S HOME"
                                                   :rem 27
1212 PRINT"{5 DOWN}{15 RIGHT}THANK YOU"            :rem 18
1215 FORG=1TO5                                     :rem 61
1220 FORI=1TO10                                    :rem 103
1229 POKESO+1,I+6:POKESO+4,17                      :rem 207
1240 NEXT:NEXT                                     :rem 127
1245 FORI=1TO200:POKESO+1,I:POKESO+4,17:NEXT
                                                   :rem 242
1250 NEXT:GOTO7700                                 :rem 72
4100 FORI=56216TO56295:POKEI,L1:POKEI-SO,36:NEXT:R
     ETURN                                         :rem 232
5000 REM UP THE GAME                               :rem 73
5005 GOSUB10000:GOSUB11100:BUG=33                  :rem 163
5011 SP=35:LI=4:SC=0:LO=1:GOSUB5500:HI=5:WO=5:EX=2
     :L1=11                                        :rem 126
5012 POKE53270,PEEK(53270)OR16                     :rem 163
```

```
5015 PRINT"{CLR}":GOSUB6500                    :rem 180
5020 XA$="":YA$="":XB$="":YB$="":XA=20:YA=19:DX=0
                                               :rem 161
5021 IFLO=3THENYA=18                           :rem 209
5025 DY=-1:T=0:IFHI<0THENHI=0                  :rem 242
5030 DI=4:IFHI>5THENHI=5                       :rem 197
5050 FORI=1024+40*2TO1024+39+40*2:POKEI,35:POKEI+1
     9*40,35:POKEI+SO,9                        :rem 104
5051 POKEI+SO+19*40,9:NEXT                      :rem 17
5055 FORI=1024+40*2TO1024+40*20STEP40:POKEI,35:POK
     EI+39,35:POKEI+SO,9                       :rem 26
5056 POKEI+SO+39,9:NEXT:IFHI>0THEN5060         :rem 76
5057 I=1024+20+40*2:POKEI,160:POKEI+SO,0:I=1024+20
     +21*40:POKEI,160:POKEI+SO,0               :rem 102
5058 POKE1024+12*40,160:POKE1024+12*40+SO,0:rem 28
5059 POKE1024+12*40+39,160:POKE1024+12*40+SO+39,0:
     GOSUB4100:GOTO150                         :rem 211
5060 XX=RND(1)*36+2:X=RND(1)*18+3:IFPEEK(1024+XX+X
     *40)<>32THEN5060                          :rem 255
5065 POKE1024+XX+X*40,BU:POKE1024+XX+X*40+SO,13
                                               :rem 238
5070 GOSUB4100                                  :rem 17
5085 GOTO150                                   :rem 161
5500 PRINT"{CLR}"                               :rem 46
5510 PRINT"{11 DOWN}{15 RIGHT}GET READY"       :rem 95
5540 FORX=1TO14:POKESO+1,NN(X):POKESO+4,17:FORD=1T
     O120:NEXT:NEXT                            :rem 84
5545 POKESO+4,16                               :rem 150
5550 RETURN                                    :rem 175
6500 REM REDEFINING SCREEN                      :rem 74
6510 POKESO+4,16                               :rem 143
6575 GOSUB 6600                                 :rem 35
6580 RETURN                                    :rem 179
6600 REM PRINT SCORE                           :rem 185
6605 PRINT"{YEL}{HOME}SCORE ";SC               :rem 160
6606 PRINT"{HOME}{30 RIGHT}ROOM ";LO            :rem 47
6610 PRINT"MUSHROOMS ";HI;"{17 SPACES}LIVES ";LI
                                                :rem 9
6620 RETURN                                    :rem 174
7000 REM CLEAN UP THE CENTIPEDE                 :rem 33
7002 SP=SP-5                                   :rem 174
7004 GOSUB 6600:HI=5                            :rem 84
7005 L=LEN(XA$)                                 :rem 66
7010 FORI=1TOL-1                               :rem 179
7020 POKESO+1,I:POKESO+4,17:FORQQ=1TO10:NEXT
                                               :rem 23
7190 PO=1024+ASC(XA$)+40*ASC(YA$):LL=LEN(XA$)-1:XA
     $=RIGHT$(XA$,LL)                          :rem 37
7195 YA$=RIGHT$(YA$,LL):POKEPO,32:POKEPO+SO,1
                                               :rem 17
```

```
7200 NEXT:POKESO+4,16                                  :rem 5
7210 RETURN                                           :rem 170
7500 REM OOPS                                         :rem 241
7510 PRINT"{CLR}{PUR}"                                :rem 205
7515 SP=SP-5                                          :rem 183
7520 PRINT"{12 DOWN}{18 RIGHT}OOPS"                   :rem 182
7521 LI=LI-1                                          :rem 148
7525 FORDE=1TO20:NEXT                                  :rem 47
7530 FORDE=1TO10:POKESO+1,DE*20:POKESO+4,17:FORQQ=
     1TO10:NEXT:NEXT:POKESO+4,16                      :rem 123
7550 FORDE=1TO20:NEXT                                  :rem 45
7560 IFLI<1THEN7700                                    :rem 96
7599 PRINT"{CLR}":RETURN                               :rem 92
7700 REM THE GAMES OVER                                :rem 60
7705 POKESO+4,16                                      :rem 150
7710 PRINT"{CLR}":POKE53270,PEEK(53270)AND15
                                                      :rem 121
7715 IF SC>HSTHENHS=SC:GOSUB9000:PRINT"{CLR}{8}"
                                                       :rem 43
7718 PRINT"{6 DOWN}{18 RIGHT}NERM"                     :rem 74
7720 PRINT"{YEL}{4 DOWN}YOUR SCORE ";SC                :rem 31
7730 PRINT"{6}{4 DOWN}HIGH SCORE ";HS                 :rem 241
7735 GOSUB 7800                                        :rem 37
7736 PRINT"{WHT}{2 DOWN}PRESS THE TRIGGER TO PLAY
     {SPACE}AGAIN, Q=QUIT"                            :rem 135
7740 FORX=1TO15:POKESO+1,PN(X):POKESO+4,17:FORD=1T
     O100:NEXT:NEXT                                    :rem 89
7745 POKESO+4,16                                      :rem 154
7780 S=PEEK(56320)AND16:IFS=0THEN5011                 :rem 149
7783 IFPEEK(197)=62THENPOKE198,0:SYS2048              :rem 135
7785 GOTO7780                                         :rem 234
7800 REM RANK THE GAMER                                :rem 44
7810 PRINT"{CYN}{2 DOWN}{7 SPACES}YOUR NEW RANK IS
      ";                                              :rem 158
7820 IFLO=1THENPRINT"ZERO"                            :rem 169
7830 IFLO=2THENPRINT"ROOKIE"                           :rem 52
7840 IFLO=3THENPRINT"NOVICE"                           :rem 49
7850 IFLO=4THENPRINT"AVERAGE"                         :rem 106
7860 IFLO=5THENPRINT"MASTER"                           :rem 61
7870 IFLO=6THENPRINT"GRAND MASTER"                    :rem 171
7880 IFLO=7THENPRINT"WIZARD"                           :rem 70
7890 IFLO=8THENPRINT"GRAND WIZARD"                    :rem 180
7900 IFLO=9THENPRINT"SUPER STAR"                       :rem 57
7910 IFLO>9THENPRINT"HALL OF FAME"                     :rem 65
7920 RETURN                                           :rem 178
9000 REM NEW HIGH SCORE                                :rem 51
9002 PRINT"{CLR}"                                      :rem 47
9003 PRINT"{CYN}{8 DOWN}{16 RIGHT}NEW HIGH"
                                                      :rem 158
```

```
9004 PRINT"{4 DOWN}{17 RIGHT}SCORE"          :rem 75
9005 FORY=1TO3                               :rem 82
9010 FORN=1TO5                               :rem 69
9020 FORD=1TO5:POKESO+1,D*20:POKESO+4,17:NEXT
                                             :rem 22
9030 POKESO+1,N*30:POKESO+4,17              :rem 254
9050 NEXT                                    :rem 13
9060 NEXT                                    :rem 14
9065 FORD=1TO30:NEXT                        :rem 236
9070 RETURN                                 :rem 176
9100 REM EXTRA LIFE                          :rem 82
9110 PRINT"{CLR}"                            :rem 47
9115 PRINT"{CYN}{12 DOWN}{15 RIGHT}BONUS LIFE"
                                            :rem 102
9120 FORJ=100TO200                            :rem 0
9140 POKESO+1,J:POKESO+4,17                 :rem 111
9150 NEXT                                    :rem 14
9160 POKESO+4,16                            :rem 147
9170 EX=EX+3                                :rem 166
9180 LI=LI+1                                :rem 149
9190 RETURN                                 :rem 179
10000 DIM PN(15),NN(18):PRINT"{CLR}{8}"     :rem 219
10005 SO=54272:POKESO+24,15:POKESO+5,17:POKESO+6,2
      41:POKESO,100                         :rem 253
10010 PRINT"{6 DOWN}{15 RIGHT}WELCOME TO"   :rem 139
10020 PRINT"{CYN}{4 DOWN}{14 RIGHT}WORM OF BEMER"
                                            :rem 130
10025 PRINT"{1}{3 DOWN}{13 RIGHT}USE JOYSTICK #2"
                                            :rem 168
10030 PRINT"{6}{6 DOWN}{11 RIGHT}HIT TRIGGER TO ST
      ART"                                   :rem 45
10045 S=PEEK(56320)AND16:IFS=0THENRETURN    :rem 210
10060 GOTO10045                              :rem 42
11100 PRINT"{CLR}{CYN}{12 DOWN}{9 RIGHT}REDEFINING
      {2 SPACES}CHARACTERS"                 :rem 130
11109 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND
      251                                    :rem 29
11110 FORI=12288TO12288+64*8:POKEI,PEEK(I+40960):N
      EXTI                                  :rem 146
11120 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
                                            :rem 228
11180 FORI=0TO39:READA:POKE12288+I+32*8,A:NEXT
                                            :rem 248
11185 FORI=0TO7:READA:POKE12288+I+42*8,A:NEXT
                                            :rem 201
11190 POKE53272,(PEEK(53272)AND240)+12       :rem 27
11200 FORI=1TO18:READNN(I):NEXT             :rem 163
11210 FORI=1TO15:READPN(I):NEXT             :rem 163
11240 DATA 0,0,0,0,0,0,0,0                  :rem 198
```

```
11250 DATA 0,20,85,85,255,40,40,0           :rem 71
11260 DATA 85,85,85,85,85,85,85,85         :rem 176
11261 DATA 170,190,190,190,190,190,170,170 :rem 19
11262 DATA 255,255,255,255,255,255,255,255 :rem 42
11263 DATA 0,20,255,255,255,255,20,0       :rem 223
11270 RETURN                               :rem 219
12000 DATA 33,44,0,44,50,0,50,56,67,56,67,56,44,0,
      33,44,0,44                           :rem 114
12100 DATA 44,0,0,33,31,33,37,37,33,0,0,0,42,42,44
                                           :rem 108
```

5
Arcade Games

Brunhilde Loves Bruno

Philip I. Nelson

In this game of love vs. evil, Brunhilde must traverse dangerous territory filled with goblins to free her friend Bruno from prison. But Bruno is entranced and must first be caught before the lovers can escape. There's also a detailed explanation of the game program and many valuable techniques you can use in your own creations. Joystick required.

"Brunhilde Loves Bruno" shows how some of the Commodore 64's more advanced graphics features can be used in a game. You'll find custom characters and all eight sprites here, as well as three-voice sound effects and music. And there's a handy machine language subroutine in the game that you can use in any program to quickly copy the 64's character set from ROM (Read Only Memory) into RAM (Random Access Memory).

The game is almost all in BASIC, so it's not difficult to type in. Make sure you read Appendix C and use "The Automatic Proofreader" as you type in Brunhilde Loves Bruno. You can be assured that the program will work correctly the first time.

Plug a joystick in port 2. Load and run the game and then select the skill level you'd like to play on. Level 1 is the easiest, level 4 the hardest. Your character, Brunhilde, will soon appear on the left side of the screen, near the bottom. You have five chances to sneak her through goblin territory to Valhalla Prison, where her friend, Bruno, has been trapped. Unfortunately, poor Bruno's in a trance, and can't recognize Brunhilde, so you have to catch him before the pair can travel back home, where a red heart beats faithfully. If your Brunhilde runs out of time (maximum of 200 seconds) or bumps into a goblin, Bruno goes back to prison and she has to start all over again. If the goblins prevail, they dance and sing a victory song. However, if you bring Bruno safely home, the goblins are discombobulated and Brunhilde sings her Valkyrie song of triumph.

Setting Things Up

Brunhilde Loves Bruno takes a lot of setting up, so the program lets you read instructions while the computer goes through its preliminaries. In lines 1390–1650, variables are initialized, sound registers are prepared, and the goblins get ready to move. Then instructions are displayed (by a GOSUB 1660), Brunhilde's theme song plays, and the custom characters are programmed (through a GOSUB 1910).

Five custom characters are used. Three create Brunhilde and Bruno, the fourth draws a miniature heart, and the fifth makes up the playfield border. In line 1910, locations 56334 and 1 are POKEd to turn off the keyboard, and to make the computer "look" at its permanent character set located in ROM. Once that's done, the SYS 49152 in line 1920 calls the machine language subroutine, which does the work of copying the standard character set from ROM into RAM where changes can be made. This copying takes nearly 30 seconds, if done in BASIC, so time is saved by using this machine language subroutine. (An assembly language listing appears at the end of this article, as Program 2, if you're interested in using the routine elsewhere.) Once the character set has been moved, location 53272 is POKEd to fool the computer into getting its character information from the new set in RAM. Five characters, with screen POKE codes from 90 to 94, are then changed by POKEing new DATA values into the correct RAM locations—and magically, the valiant Brunhilde appears in the instruction display.

Before the program leaves the instruction display, it prepares the color memory background for the playfield, so that the characters look yellow in the prison area, are green at home, and change colors in goblin territory (lines 2020–2040). Using a PRINT"{CLR}" at this point to erase the instructions would erase these colors, too, so when the player is finished reading, the entire screen is POKEd with a value of 32 (for blank spaces). The player picks a skill level, the playfield is drawn, and Brunhilde's off and running.

Bump and Run

Once play begins, the program simply cycles through line 120 over and over, calling subroutines to move Brunhilde (GOSUB 540), Bruno (GOSUB 610), and one goblin sprite (ON FNK(0) GOSUB 140, 190, and so on), each time through the loop. The

user-defined function FNK randomly picks which goblin sprite to move. If you'd rather move them in left-to-right order, substitute CF for FNK(0) in line 125.

Collision detection appears in the routines that move Brunhilde and the goblins. Register 53265 (V+31) can be used to check whether a visible (nonzero) portion of any sprite has bumped into the visible part of a character.

To check sprite 1 only, you could use the statement IF PEEK(V+31)AND1=1 THEN *(desired action).* By changing the value 1 to 2, 4, 8, and so on, you could check all eight sprites in order whenever you wanted. In this game, you can get by with the simpler statement IF PEEK(V=31)<>0 THEN *(action).* The only characters allowed into goblin territory are Brunhilde and Bruno, and you want to shift into the return home routine (GOSUB 740) whenever the lovers touch *any* of the eight sprites. Checking for any nonzero value in this collision register saves time.

Once a collision is detected, the character is immediately erased by POKEing its screen memory location with 32 (blank) to avoid an unwanted second collision. Then the collision register is cleared with POKE V+31,0 (lines 890 and 900).

Bruno's movement routine (GOSUB 610) simply sends him wandering through the prison. As long as Brunhilde hasn't caught him by moving precisely into his path, Bruno's horizontal (X) and vertical (Y) movement is limited, so he can't escape even if he accidentally hits the prison door. Line 630 checks whether Bruno's screen memory location (BU) is identical with Brunhilde's (BR). When that happens, the flag CT is set, and from that point onward, Bruno's location will copy Brunhilde's, to make him follow her.

Goblin Animation

At the start, two different sets of sprite shape DATA were stored in locations beginning at 832 and 896. By changing a sprite's pointer value, you can make it call its shape information from either location, and thus switch at will from one pattern to the other. In the movement routines for sprites 2 through 7, the pointer is switched to 14 when entering the routine, and back to 13 when exiting, so the goblin sprites take a different shape while they're moving. Sprites 1 and 8 look more frantic because they're switching shape pointers every time one cycle of the innermost movement loop is executed.

To make the sprites dance, all eight shape pointers are simply switched back and forth in lines 990–1020. By playing with these pointers, a different effect is achieved in the "discombobulation" routine that is used when you bring Bruno safely home. Lines 1280–1310 make the sprites appear to disintegrate; the program runs through *K* loop values from 6 to 14, flipping all the sprite pointers through nine successive memory locations. Only two of those values point sprites to an area of memory containing meaningful shape data, however—when pointed at other areas, the sprites are defined by garbage data contained there, which makes them appear to dissolve and rematerialize. For serious animation, of course, you can use this pointer-flipping technique to move sprites through a succession of different shapes, to make birds fly, horses run, and so on.

Besides changing shape, sprites can also be expanded vertically, horizontally, or in both directions at once. Brunhilde Loves Bruno uses the sprite expansion registers in two places—when Brunhilde bumps up against a goblin (GOSUB 740) and when the goblins do their victory song and dance (GOSUB 970). When a goblin touches Brunhilde, a sequence executes that successively POKEs location 53271 (V+23) with values of 1, 2, 4, 8, and so on. This expands each sprite vertically in order, making them seem to kick irritably (lines 800–850). When the game's lost, all goblins are expanded both vertically *and* horizontally, by POKEing a value of 255 into registers 53271 and 53277 (line 1030); this makes the shapes double-sized. To shrink them back to normal size, of course, both expansion registers are just POKEd with 0.

Brunhilde and Bruno—Alive at Last

Just as you can animate sprites by flipping back and forth between two alternate sets of shape information, so you can animate characters by switching between two alternate custom characters. Brunhilde does a little flip-flop every time her routine is executed, since the figure's screen memory location is first POKEd with 93, then with 92. Custom character 92 is the upright Brunhilde, and character 93 is the same shape, upside down. A third shape (character 91—an upright Brunhilde shape, minus legs) is used for Bruno's scrunched-down shape; he seems to hop up when character 91 (the same as Brunhilde) is POKEd into the proper screen location. To make

the heart appear to beat, the program switches from the standard heart character (83) to a custom-designed miniature heart character (90).

Given the Commodore 64's generous memory, these techniques can be used to create some mind-boggling graphic effects. Characters are smaller than sprites, so you must use simpler, bulky shapes, but in theory there's nothing to prevent you from programming dozens or even hundreds of characters. Likewise, the number of different sprite shapes you can use is limited only by the amount of available RAM. Remember, however, that only eight sprite shapes can appear on the screen at the same time.

Moving Arrays

Three arrays are used in this game to simplify character movement, color effects, and scorekeeping. The *M* array stores movement values for Brunhilde. In her routine, a value pulled from the joystick *(JK)* is used to pick a certain value from the movement array and add it to Brunhilde's present location in screen memory: BR=BR+M(JK). Adding 40 moves her down one space, adding 1 moves her right one space, subtracting 40 moves her up, and subtracting 1 moves her left. The *CH$* array stores strings that are PRINTed to the screen to show how many chances Brunhilde has left. The *Z* array stores values for colors that contrast well against the black background. When goblin territory is POKEd with colors, the statement Z(FNV(0)) lets the computer pick values randomly from this array. Likewise, each time a goblin's movement routine is entered, this same statement is used to switch the figure's color.

Ride of the Valkyrie

There's lots of sound in this game, using all three of the 64's voices and waveforms. The simplest, Brunhilde's beeping as she moves, is created by turning voice 1 (W1) on, and POKEing a value into its high frequency register. By tying the frequency value to her screen memory location (line 550), her beeps change pitch, sounding higher as she moves toward the top of the screen, and lower as she heads for the bottom of the playing field.

Each sprite has its own unique sound, made by mixing sound statements into each movement routine. To keep from slowing the movement down, these are one-voice effects. But

by varying the waveforms, or adding ring modulation (such as with sprite eight), you can still get a variety of sounds.

To play Brunhilde's theme song at the beginning or the goblins' tune at the end, note values are READ from the appropriate DATA statements and POKEd into the correct frequency registers. This program has a lot of DATA statements—for sprites, music, machine language, and programmed characters. Whenever you tell your computer to READ DATA, it starts at the lowest BASIC line number and moves on up, reading every item in order. Trying to READ after the DATA pointer hits the end of DATA gives you an OUT OF DATA error. That's no problem if you're just POKEing sprite DATA into memory—once you've used that DATA, there's no need to look at it again. But what if you want to play the same song more than once and your music DATA is stuck halfway down in your BASIC program, between sprite and machine language DATA?

One solution is to use a RESTORE statement to put your DATA pointer back at line 0, and then use a dummy loop to READ the correct number of DATA items (and do nothing with them), leaving the pointer at the right spot. In line 2650, the dummy loop READs through the 32 note values for the goblins' song, 63 items for each of the two sprite shapes, and 8 items for each of the programmed character shapes. In line 3020 all of the sprite, music, and character DATA are READ to move the pointer to line 2970, where the machine language DATA begin. You're really just using the DATA statements like a series of look-up tables located in the BASIC program itself. To find a certain piece of information, all you need do is send the pointer to the right place to pull the information from the correct table.

Prisoners of Love

Three different techniques are used to confine the various animated figures to the proper areas. It's always important to control such movement, to keep action on the screen, and to prevent characters from running past screen memory into the area where the BASIC program is stored. Brunhilde's movement is limited by the playfield boundaries themselves. Before moving her screen memory location, her routine PEEKs ahead to see whether the new location is already occupied by character 94 (remember, that's the custom character for the playfield

walls). If so, the subroutine RETURNs without moving her figure. Trying to move her left from her home area actually makes her bump up against the prison wall on the far right of the screen.

Bruno's routine also PEEKs ahead to check for character 94, which makes him bounce off obstacles inside his prison. But he's also limited by minimum and maximum X and Y values, which are defined in relation to his starting spot (upper left) in the prison. If he moves too far in any of the four possible directions, his X or Y value will be reversed in sign, making him rebound. Once Brunhilde captures him, he copies her location and RETURNs from his movement subroutine before getting into the X and Y scheme. This lets him crash out of jail.

Sprites, on the other hand, move independently of screen memory, on a totally separate display that's bigger than the visible screen. The visible sprite grid is about 180 × 266, compared to the 40 × 20 screen memory grid, so moving a sprite one space creates a barely perceptible movement. In this game the goblins have to be kept onscreen, moved in proportion to the characters, and prevented from bumping into playfield walls, which would set off the collision register.

To simplify things, each goblin sprite is confined to its own vertical pathway. This is done in lines 1630–1640 where each sprite's horizontal register is POKEd with a fixed value. Statements to check whether each sprite has moved too far up or down (vertical positions 75 and 220 on the sprite grid respectively) are included. Whenever those limits are crossed, the goblin wraps around to the opposite side. Finally, within these limitations, each goblin can be moved by changing the value of the variable assigned to its vertical position (S1 for sprite 1, S2 for sprite 2, and so on).

The amount of distance each sprite moves is determined by two things. First, the user-defined function FNG is employed to pick how many five-space leaps the sprite makes for every pass of its routine. Second, the maximum number of leaps FNG can choose is limited by PK, the variable representing skill level. At skill level 1, the maximum number of spaces a sprite can cover is 50, while at skill level 4, the maximum is 125. That's almost enough to wrap all the way around goblin territory to sneak up on Brunhilde from behind.

Cautions

When using sprite-to-character collision detection, you must take care with all PRINTs and POKEs, so as not to leave unwanted character data in screen memory where it would trigger an unwanted collision. And you must pay careful attention to the status of your collision register. Once a collision has been flagged, the value is stored in that register until you check it with another PEEK or clear it by POKEing the register with 0.

One final word of caution: As in any other BASIC program that uses a machine language subroutine, be sure to save the program before you run it, in case you made a typing error. Otherwise, your 64 could lock up, forcing you to turn it off, then on again, to regain control. That will erase any program in memory, including an unsaved version of Brunhilde Loves Bruno.

Program Lines

Here's a short summary of what the various sections and routines in Brunhilde Loves Bruno do:

Line	Function
100–110	Protect BASIC program space and call preparatory GOSUBs
120–125	Call subroutines for all game figures
140–170	Movement and sound for sprite 1
190–220	Movement and sound for sprite 2
240–270	Movement and sound for sprite 3
290–320	Movement and sound for sprite 4
340–370	Movement and sound for sprite 5
390–420	Movement and sound for sprite 6
440–470	Movement and sound for sprite 7
490–520	Movement and sound for sprite 8
540–590	Movement for Brunhilde character
610–720	Movement for Bruno character
740–950	Return home routine
970–1190	Lost game routine
1210–1370	Won game routine
1390–1650	Initialize variables; set up sound and sprite register
1670–1890	Display instructions for player
1910–2260	Program characters, select skill level, color playfield
2280–2380	Draw playfield
2420–2430	Sprite victory song DATA

Line	Function
2470–2520	Sprite shape DATA
2550	Small heart character DATA
2570	Small Bruno character DATA.
2590	Brunhilde character DATA
2610	Upside-down Brunhilde character DATA
2630	Playfield wall character DATA
2650–2710	Valkyrie song routine
2730–2760	Valkyrie song DATA
2780–2830	Ascending sound routine
2850–2890	Mighty Mouse song routine
2910–2920	Mighty Mouse song DATA
2930–2950	Descending sound routine
2970–3040	POKE machine language subroutine into memory beginning at location 49152

Program 1. Brunhilde Loves Bruno

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
100 PRINT"{CLR}":POKE52,100:POKE56,100:CLR:GOSUB29
    60:GOSUB1380:GOSUB1660                    :rem 198
110 GOSUB2640:GOSUB1910:GOSUB2270             :rem 232
120 GOSUB540:GOSUB610:GOSUB540:GOSUB610       :rem 161
125 ONFNK(0)GOSUB140,190,240,290,340,390,440,490:G
    OTO 120                                    :rem 23
130 REM***** MOVE SPRITE ONE ************      :rem 50
140 POKEW1,17:POKEV+39,Z(FNV(0))              :rem 120
150 FORJ=1TOFNG(0):POKE2040,14:S1=S1-5:IFS1<75THEN
    S1=220                                     :rem 78
160 POKE2040,13:IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH
    -1:GOSUB740:RETURN                        :rem 199
170 POKEV+1,S1:POKEH1,S1/6:NEXT:CF=2:RETURN
                                              :rem 229
180 REM**** MOVE SPRITE TWO ************      :rem 37
190 POKE2041,14:POKEW1,33                     :rem 164
200 POKEV+40,Z(FNV(0)):FORJ=1TOFNG(0):S2=S2+5:IFS2
    >220THENS2=75                              :rem 30
210 IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH-1:GOSUB740:
    RETURN                                      :rem 4
220 POKEV+3,S2:POKEH1,S2/17+FNB(0)::NEXT:CF=3:POKE
    2041,13:RETURN                            :rem 148
230 REM**** MOVE SPRITE THREE **********      :rem 75
240 POKE2042,14:POKEW1,129                    :rem 215
250 POKEV+41,Z(FNV(0)):FORJ=1TOFNG(0):S3=S3-5:IFS3
    <75THENS3=220                              :rem 40
260 IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH-1:GOSUB740:
    RETURN                                      :rem 9
270 POKEV+5,S3:POKEH1,S3/12:NEXT:CF=4:POKE2042,13:
    RETURN                                    :rem 222
```

```
280 REM**** MOVE SPRITE FOUR **********   :rem 62
290 POKE2043,14:POKEW1,17                 :rem 169
300 POKEV+42,Z(FNV(0)):FORJ=1TOFNG(0):S4=S4+5:IFS4
    >220THENS4=75                         :rem 41
310 IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH-1:GOSUB740:
    RETURN                                :rem 5
320 POKEV+7,S4:POKEH1,S4/5:NEXT:CF=5:POKE2043,13:R
    ETURN                                 :rem 178
330 REM**** MOVE SPRITE FIVE **********   :rem 40
340 POKE2044,14:POKEW1,33                 :rem 164
350 POKEV+43,Z(FNV(0)):FORJ=1TOFNG(0):S5=S5-5:IFS5
    <75THENS5=220                         :rem 51
360 IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH-1:GOSUB740:
    RETURN                                :rem 10
370 POKEV+9,S5:POKEH1,S5/10:NEXT:CF=6:POKE2044,13:
    RETURN                                :rem 233
380 REM**** MOVE SPRITE{2 SPACES}SIX **********
                                          :rem 247
390 POKE2045,14:POKEW1,129                :rem 224
400 POKEV+44,Z(FNV(0)):FORJ=1TOFNG(0):S6=S6+5:IFS6
    >220THENS6=75                         :rem 52
410 IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH-1:GOSUB740:
    RETURN                                :rem 6
420 POKEV+11,S6:POKEH1,S6/10:NEXT:CF=7:POKE2045,13
    :RETURN                               :rem 18
430 REM**** MOVE SPRITE SEVEN **********  :rem 86
440 POKE2046,14:POKEV+45,Z(FNV(0))        :rem 185
450 FORJ=1TOFNG(0):POKEW1,17:S7=S7-5:IFS7<75THENS7
    =220                                  :rem 46
460 IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH-1:GOSUB740:
    RETURN                                :rem 11
470 POKEV+13,S7:POKEH1,S7/5:POKEW1,129:NEXT:CF=8:P
    OKE2046,13:RETURN                     :rem 170
480 REM**** MOVE SPRITE EIGHT **********  :rem 75
490 POKEW1,21:POKEV+46,Z(FNV(0)):POKEH3,FNB(0)+5
                                          :rem 64
500 FORJ=1TOFNG(0):POKE2047,14:S8=S8+5:IFS8>220THE
    NS8=75                                :rem 112
510 POKE2047,13:IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH
    -1:GOSUB740:RETURN                    :rem 205
520 POKEV+15,S8:POKEH1,S8/10:NEXT:CF=1:POKE2047,13
    :RETURN                               :rem 23
530 REM******* MOVE BRUNHILDE **********  :rem 26
540 POKE1064,90:POKEBR,93:JK=PEEK(56320):JK=15-(JK
    AND15):POKEW1,17                      :rem 130
550 POKEH1,INT(120-(BR/20)):IFPZ<SKANDBU=1144ANDBR
    =1144THEN1200                         :rem 254
560 IFPEEK(BR+M(JK))=94THEN590            :rem 9
```

```
570 IFPEEK(V+31)<>0THENPOKEBR,32:CH=CH-1:GOSUB740:
    GOTO590                                    :rem 4
580 POKEBR,32:BR=BR+M(JK)                     :rem 238
590 POKEBR,92:POKEW1,16:POKE1064,83:RETURN    :rem 97
600 REM******* MOVE BRUNO ************       :rem 127
610 POKEBU,91:PRINT"{HOME}{2 RIGHT}"PZ:PZ=PZ+A
                                              :rem 44
615 IFPZ=SKTHENPOKEBR,32:CH=CH-1:GOSUB740:GOTO120
                                              :rem 121
620 BN=0:IFCT=ATHENPOKEBU,32:BU=BR:POKEBU,92:RETUR
    N                                          :rem 76
630 IFPEEK(BU)=PEEK(BR)THENBU=BR:POKEBU,91:POKEBU,
    81:POKEBU,91:CT=A:RETURN                   :rem 61
640 BN=BN+A:POKEBU,32:BU=1177+X+40*Y          :rem 103
650 POKEBU,92:IFBN=2THENRETURN                :rem 207
660 X=X+DX:IFX=0ORX=5THENDX=-DX                :rem 26
670 IFY<=0THENY=A:DY=-DY                       :rem 18
680 IFY=>20THENY=19:DY=-DY                    :rem 112
690 Y=Y+DY:IFDX=0THENDX=A                     :rem 113
700 BP=1177+X+40*Y:IFPEEK(BP)=32THEN640        :rem 7
710 IFPEEK(BP)=94THENDX=+DX:DY=-DY:GOTO660:rem 149
720 RETURN                                    :rem 121
730 REM***** BUMPED A SPRITE **********       :rem 243
740 POKEBU,32:CT=0:Y=1:X=1:BR=1864:BU=1177:JK=0
                                              :rem 41
750 POKEW1,16:POKEW1,21:POKEH1,2*FNB(0)+5:PZ=0
                                              :rem 216
760 POKEW2,21:POKEH2,FNB(0)+5:POKEL2,108       :rem 86
770 POKEW3,17                                 :rem 235
780 FORG=1TO3:POKEH3,14:POKE53280,Z(FNV(0)):rem 51
790 K=1:POKEW1,21                             :rem 217
800 FORJ=1TO8:POKEH1,4*J+20*G:POKEH3,40-(4*J):POKE
    2039+J,14                                 :rem 157
810 POKEV+23,K                                 :rem 37
820 K=K*2                                     :rem 201
830 FORI=1TO30:NEXT                           :rem 184
840 POKEV+23,0                                 :rem 13
850 POKE2039+J,13:NEXTJ                        :rem 98
860 FORJ=1TO5:PRINT"{HOME}"CHR$(142):FORK=1TO20:NE
    XT                                        :rem 116
870 PRINT"{HOME}"CHR$(14):FORK=1TO20:NEXTK:NEXTJ
                                              :rem 160
880 POKEW1,16:POKEW2,16:NEXTG                  :rem 47
890 POKE53280,0:POKEV+29,0:POKEW3,16:POKEH3,0:POKE
    W2,16:POKEV+31,0                          :rem 212
900 POKEH1,1:POKEH2,1:POKEV+31,0              :rem 136
910 FORJ=1TO1200:NEXT                          :rem 24
920 PRINT"{63}{HOME}{7 SPACES}{8 RIGHT}CHANCES:
    {RIGHT}{YEL}";CH$(CH):POKEBU,92           :rem 71
```

```
930 FORJ=1TO500:NEXT                                    :rem 236
940 IFCH<=0THENGOSUB970:RETURN                          :rem 201
950 RETURN                                              :rem 126
960 REM****{2 SPACES}LOSING MESSAGE **********
                                                        :rem 202
970 POKE53280,5:PRINT"{HOME}{YEL}{5 RIGHT}{RVS}
    {7 SPACES}TOO SLOW !!{6 SPACES}{OFF}{WHT}":PZ=
    0                                                   :rem 242
980 POKEV+23,0:POKEV+29,0                               :rem 195
990 FORJ=1TO16:POKEW1,17:READQ1:READQ2:IFQ1=0THENR
    ESTORE:GOTO1030                                     :rem 195
1000 POKEH1,Q1:POKEH2,Q2                                :rem 169
1010 FORB=0TO7:POKE2040+B,14:NEXTB                       :rem 26
1020 FORB=0TO7:POKE2040+B,13:NEXTB:POKEW1,16:NEXTJ
                                                         :rem 97
1030 PRINT"{HOME}{3}{5 RIGHT}{RVS} BRUNO IS OUR PR
     ISONER ! {OFF}{WHT}":POKEV+23,255:POKEV+29,25
     5                                                   :rem 87
1040 POKE53280,3:POKEH3,8:POKEL3,108                     :rem 74
1050 FORJ=1TO16:READQ1:READQ2:IFQ1=0THENRESTORE:GO
     TO1090                                              :rem 104
1060 POKEW1,17:POKEW2,17:POKEW3,21:POKEH1,Q1:POKEL
     1,Q2:POKEH2,Q1/2:POKEL2,Q2/2                        :rem 40
1070 FORB=0TO7:POKEH3,B*J+6:POKE2040+B,14:NEXTB:FO
     RB=7TO0STEP-1                                       :rem 139
1080 POKEH3,B*J+3:POKE2040+B,13:NEXTB:NEXTJ:POKEH1
     ,0:POKEH2,0:POKEH3,0                                :rem 26
1090 POKEW1,0:POKEW2,0:POKEW3,0:POKE53280,0:POKE53
     265,0:FORJ=1TO1500:NEXT                             :rem 198
1100 POKE53265,27:PRINT"{HOME}{WHT}{RVS}{RIGHT} PR
     ESS 'Y' TO PLAY AGAIN, 'N' TO QUIT "  :rem 248
1110 POKEW1,17:POKEW2,17:POKEW3,17:K=FNB(0):POKEH1
     ,K+8:POKEH2,K+13:POKEH3,K+3                         :rem 146
1120 GETA$:FORK=0TO7:POKE2040+K,14:NEXT    :rem 107
1130 IFA$=""THEN 1135                                    :rem 51
1132 GOTO 1140                                          :rem 198
1135 FORK=0TO7:POKE2040+K,13:NEXT:POKEW1,16:POKEW3
     ,16:POKEW2,16:GOTO1110                              :rem 182
1140 IFA$<>"N"ANDA$<>"Y"THENFORK=0TO7:POKE2040+K,1
     3:NEXT:GOTO1110                                     :rem 161
1150 IFA$="N"THEN 1155                                  :rem 133
1152 GOTO 1160                                          :rem 202
1155 PRINT"{CLR}":FORJ=53248TO53264:POKEJ,0:NEXT:P
     OKEW1,16:POKEW3,16:POKEW2,16:END                    :rem 173
1160 IFA$="Y"THENPOKEV+23,0:POKEV+29,0:CH=5:POKEW1
     ,16:POKEW2,16:POKEW3,16                             :rem 173
1170 FORJ=1024TO1063:POKEJ,32:NEXT                       :rem 40
1180 BR=1864:BU=1177:BP=1177:CT=0:PZ=0:POKE1144,32
     :GOSUB2180:RETURN                                   :rem 23
```

```
1190 GOTO1110                                        :rem 199
1200 REM**** WINNING ROUTINE **********              :rem 51
1210 POKE1064,83:POKE1144,92                         :rem 31
1220 POKEW1,17:POKEW2,17:POKEW3,17:POKEH1,34:POKEL
     1,75:POKEH2,43:POKEL2,52                        :rem 0
1230 POKEH3,51:POKEL3,97:FORJ=1TO120:NEXTJ:POKEH1,
     68:POKEL1,149:POKEH2,86                         :rem 1
1240 POKEL2,105:FORJ=1TO750:NEXT                     :rem 199
1250 POKEW1,16:POKEW2,16:POKEW3,16                   :rem 29
1260 FORJ=1TO15STEP3:POKEW1,17:POKEW2,17:POKEW3,12
     9                                               :rem 166
1270 POKE53280,Z(FNV(0))                             :rem 61
1280 FORK=6TO14:POKEW2,21                            :rem 249
1290 POKEH1,K*J:POKEH2,(K*J)/3:POKEH3,55-(K*J)/4:P
     OKEW2,17                                        :rem 128
1300 FORI=2039TO2047:POKEI,K:NEXTI                   :rem 90
1310 POKEW3,17:NEXTK                                 :rem 214
1320 POKEW1,16:POKEW2,16:POKEW3,16:NEXTJ             :rem 222
1330 PRINT"{HOME}{YEL}{7 RIGHT}{RVS} LOVE TRIUMPHS
     ! {OFF} "                                       :rem 139
1340 FORJ=2039TO2047:POKEJ,13:NEXT                   :rem 48
1350 GOSUB2640                                       :rem 21
1360 GOSUB1090:GOTO120                               :rem 26
1370 REM                                             :rem 175
1380 REM*** INITIALIZATION ************              :rem 78
1390 S1=220:S2=75:S3=220:S4=75:S5=220:S6=75:S7=220
     :S8=75                                          :rem 135
1400 X=1:Y=1:DX=1:DY=1:BU=1177:BP=1177:CT=0:PZ=0:S
     K=201:A=1:CH=5:CF=1:BR=1864                     :rem 50
1410 V=53248:W1=54276:W2=54283:W3=54290:H1=54273:L
     1=54272:H2=54280:L2=54279                       :rem 82
1420 H3=54287:L3=54286:FORJ=54272TO54299:POKEJ,0:N
     EXT                                             :rem 108
1430 POKE54296,15:POKE54277,15:POKE54291,15:POKE54
     284,15                                          :rem 153
1440 CH$(0)=" ":DEFFNK(X)=INT(RND(1)*8)+1            :rem 168
1450 CH$(1)="{-} ":CH$(2)="{2 -} ":CH$(3)="{3 -} "
     :CH$(4)="{4 -} ":CH$(5)="{5 -}"                 :rem 26
1460 DIMM(11):M(0)=0:M(1)=-40:M(2)=40:M(3)=0:M(4)=
     -1:M(5)=-41                                     :rem 151
1470 M(6)=39:M(7)=0:M(8)=1:M(9)=-39:M(10)=41:rem 5
1480 POKEV+31,0                                      :rem 61
1490 DIMZ(9):Z(1)=10:Z(2)=8:Z(3)=7:Z(4)=13:Z(5)=3:
     Z(6)=4:Z(7)=1:Z(8)=12                           :rem 111
1500 DEFFNB(X)=INT(RND(1)*14)+1:DEFFNV(X)=INT(RND(
     1)*8)+1                                         :rem 23
1510 DEFFNG(X)=INT(RND(1)*PK):POKE53281,0:POKE5328
     0,0                                             :rem 77
1520 REM*** SPRTS ON * SIZE REGSTRS ***              :rem 112
```

```
153Ø POKEV+21,255:POKEV+23,Ø:POKEV+29,Ø      :rem Ø
154Ø REM*** SET SPRITE POINTERS *******    :rem 137
155Ø FORJ=1TO8:POKE2Ø39+J,13:NEXTJ          :rem 68
156Ø REM*** SKIP MUSIC DATA ***********    :rem 2Ø6
157Ø FORJ=1TO32:READQ:NEXT                 :rem 148
158Ø REM*** POKE SPRITE DATA **********    :rem 244
159Ø FORJ=ØTO62:READQ:POKE832+J,Q:NEXTJ    :rem 218
16ØØ FORJ=ØTO62:READQ:POKE896+J,Q:NEXTJ    :rem 22Ø
161Ø REM                                   :rem 172
162Ø REM*** SPRITE HORIZ LOCATIONS ****    :rem 226
163Ø POKEV,45:POKEV+2,75:POKEV+4,1Ø5:POKEV+6,135:P
     OKEV+8,165:POKEV+1Ø,195                :rem 93
164Ø POKEV+12,225:POKEV+14,255             :rem 186
165Ø RETURN                                :rem 172
166Ø REM***** INSTRUCTIONS ************     :rem 48
167Ø PRINT"{CLR}{13}{-}{3 RIGHT}SS{RIGHT}SS
     {5 RIGHT}{RVS}{YEL}{11 SPACES}{OFF}{13}
     {4 RIGHT}SS{RIGHT}SS{3 RIGHT}{-}"     :rem 179
168Ø PRINT"{13}{3 RIGHT}S{2 RIGHT}S{2 RIGHT}S
     {4 RIGHT}{RVS}{YEL} BRUNHILDE {OFF}{13}
     {3 RIGHT}S{2 RIGHT}S{2 RIGHT}S"       :rem 125
169Ø PRINT"{13}{-}{2 RIGHT}S{5 RIGHT}S{4 RIGHT}
     {RVS}{YEL}{3 SPACES}LOVES{3 SPACES}{OFF}{13}
     {3 RIGHT}S{5 RIGHT}S{2 RIGHT}{-}"     :rem 211
17ØØ PRINT"{13}{4 RIGHT}S{3 RIGHT}S{5 RIGHT}{RVS}
     {YEL}{3 SPACES}BRUNO{3 SPACES}{OFF}{13}
     {4 RIGHT}S{3 RIGHT}S"                 :rem 214
171Ø PRINT"{13}{-}{4 RIGHT}S{RIGHT}S{6 RIGHT}{RVS}
     {YEL}{11 SPACES}{OFF}{13}{5 RIGHT}S{RIGHT}S
     {4 RIGHT}{-}"                          :rem 67
172Ø PRINT"{13}{6 RIGHT}S{24 RIGHT}S"       :rem 4Ø
173Ø PRINT"{13}{-}{36 RIGHT}{-}"           :rem 233
174Ø PRINT"{CYN}{2 RIGHT}BRUNO'S STUCK IN
     {2 SPACES}VALHALLA PRISON,"           :rem 227
175Ø PRINT"{-}{RIGHT}GUARDED BY EVIL GOBLINS. ONLY
      THE{2 RIGHT}{-}"                     :rem 211
176Ø PRINT"{2 RIGHT}VALIANT BRUNHILDE CAN RESCUE H
     IM."                                   :rem 42
177Ø PRINT"{-}{36 RIGHT}{-}"               :rem 1Ø8
178Ø PRINT"{6}{2 RIGHT}SHE HAS 5 CHANCES TO GET HI
     M {13}HOMES"                           :rem 56
179Ø PRINT"{6}{-}{RIGHT}EACH CHANCE LASTS 2ØØ TICK
     S.{2 SPACES}IF A{RIGHT}{-}"           :rem 117
18ØØ PRINT"{2 RIGHT}GOBLIN TOUCHES HER, IT'LL BOOT
      HER"                                  :rem 36
181Ø PRINT"{-}{RIGHT}BACK HOME. IF TIME RUNS OUT,
     {SPACE}BRUNO{RIGHT}{-}"               :rem 165
182Ø PRINT"{2 RIGHT}STAYS IN HIS PRISON FOR ETERNI
     TY!!"                                 :rem 93
```

```
1830 PRINT"{C-}{36 RIGHT}{C-}"                    :rem 105
1840 PRINT"{2 RIGHT}{YEL}BRUNO'S BEEN ENCHANTED
     {2 SPACES}AND CAN'T"                         :rem 58
1850 PRINT"{C-}{RIGHT}RECOGNIZE BRUNHILDE, SO SHE'S
      GOT{2 RIGHT}{C-}"                           :rem 3
1860 PRINT"{2 RIGHT}TO CHASE HIM DOWN-- CAN YOU HE
     LP?"                                         :rem 136
1870 PRINT"{C-}{36 RIGHT}{C-}"                    :rem 109
1880 PRINT"{RIGHT}{C-}{RIGHT}{C-}{RIGHT}{C-}{RIGHT}
     {C-}{RIGHT}{RVS}{CYN} JOYSTICK IN PORT 2{OFF}
     {YEL}{2 RIGHT}{C-}{RIGHT}{C-}{RIGHT}{C-}{RIGHT}
     {C-}"                                        :rem 3
1890 PRINT"{DOWN}{WHT}STAND BY WHILE THE GOBLINS G
     ET NASTY--"                                  :rem 255
1900 RETURN:REM COPY CHAR ROM INTO RAM            :rem 41
1910 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND2
     51                                           :rem 236
1920 SYS49152                                     :rem 208
1930 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
                                                  :rem 188
1940 POKE53272,(PEEK(53272)AND240)+14             :rem 239
1950 RESTORE:FORJ=1TO(32+63+63):READQ:NEXT:rem 109
1960 REM*** POKE NEW CHAR DATA ********           :rem 211
1970 FORJ=15056TO15095:READQ:POKEJ,Q:NEXT:RESTORE
                                                  :rem 149
1980 POKEW1,16:POKEW2,16:POKEW3,16:POKE54278,64:PO
     KE54285,64:POKE54292,64                      :rem 60
1990 POKEW1,17:POKEH1,43:POKEL1,52:POKEW3,33:POKEH
     3,6:POKEL3,108                               :rem 132
2000 POKEW2,33:POKEH2,8:POKEL2,147                :rem 2
2010 REM***** POKE COLORS *************           :rem 156
2020 FORJ=1106TO1906STEP40:FORK=JTOJ+30:POKEK+L1,Z
     (FNV(0)):NEXTK,J                             :rem 83
2030 FORJ=1144TO1904STEP40:POKEJ+L1,13:NEXT
                                                  :rem 114
2040 FORJ=1177TO1937STEP40:FORK=JTOJ+5:POKEK+L1,7:
     NEXTK,J:POKEW1,32:POKEW2,32                  :rem 89
2050 REM***** PROMPT PLAYER ***********           :rem 250
2060 POKEW3,32:PRINT"{HOME}{23 DOWN}{YEL}";
                                                  :rem 144
2070 PRINT"{C-} {C-} {C-} {RVS}PRESS SPACE BAR TO CON
     TINUE{OFF} {C-} {C-} {C-}"                   :rem 60
2080 POKEW1,16:POKEW1,17:POKEH1,10:POKEL1,205
                                                  :rem 177
2090 GETA$:I=Z(FNV(0)):FORJ=1944TO1983:POKEJ+L1,I:
     NEXT:IFA$=" "THEN2140                        :rem 198
2100 POKEW1,16:POKEW1,17:POKEH1,12:POKEL1,216
                                                  :rem 174
```

```
2110 IFA$<>" "THENI=Z(FNV(0)):FORJ=1983TO1944STEP-
     1:POKEJ+L1,I:NEXT:GOTO2080                :rem 83
2120 IFA$=" "THEN2140                          :rem 48
2130 GOTO2080                                  :rem 201
2140 POKEW1,16:POKEW3,16:FORJ=1944TO1983:POKEJ,32:
     NEXT:POKE1944+L1,13                       :rem 185
2150 FORJ=1946TO1976:POKEJ+L1,Z(FNV(0)):NEXT:FORJ=
     1978TO1982:POKEJ+L1,7:NEXT                :rem 201
2160 FORJ=1024TO2023:POKEJ,32:NEXT             :rem 37
2170 REM**** SKILL LEVEL **************       :rem 153
2180 PRINT"{HOME}{6}{RVS}{2 SPACES}PICK SKILL LEVE
     L: 1 (EASY) TO 4 (HARD)":POKEV+31,0       :rem 87
2190 GETA$:IFA$<>"1"ANDA$<>"2"ANDA$<>"3"ANDA$<>"4"
     THEN2190                                  :rem 164
2200 IFA$="1"THENPK=10                         :rem 210
2210 IFA$="2"THENPK=15                         :rem 217
2220 IFA$="3"THENPK=20                         :rem 215
2230 IFA$="4"THENPK=25                         :rem 222
2240 FORK=1024TO1063:POKEK,32:NEXT             :rem 41
2250 PRINT"{HOME}{5 SPACES}{10 RIGHT}CHANCES:
     {RIGHT}{YEL}";CH$(CH);"{6}{2 RIGHT}LEVEL:
     {YEL}";(PK-5)/5                           :rem 14
2260 RETURN                                    :rem 170
2270 REM****** DRAW VALHALLA **********       :rem 194
2280 FORJ=1104TO1143:POKEL1+J,3:POKEJ,94:NEXT
                                               :rem 235
2290 FORJ=1984TO2023:POKEL1+J,3:POKEJ,94:NEXT:POKE
     1064+L1,10:POKE1064,83                    :rem 46
2300 FORJ=1145TO1825STEP40:POKEL1+J,3:POKEJ,94:NEX
     T                                         :rem 144
2310 FORJ=1905TO1945STEP40:POKEL1+J,3:POKEJ,94:NEX
     T                                         :rem 152
2320 FORJ=1143TO2023STEP40:POKEL1+J,3:POKEJ,94:NEX
     T                                         :rem 135
2330 FORJ=1176TO1216STEP40:POKEL1+J,3:POKEJ,94:NEX
     T:POKE1222+L1,3:POKE1222,94               :rem 145
2340 FORJ=1296TO1976STEP40:POKEL1+J,3:POKEJ,94:NEX
     T:POKE1497+L1,3:POKE1497,94               :rem 190
2350 POKE1977+L1,3:POKE1977,94:POKE1817+L1,3:POKE1
     817,94                                    :rem 198
2360 FORJ=1659TO1660:POKEJ+L1,3:POKEJ,94:NEXT:POKE
     1782+L1,3:POKE1782,94                     :rem 19
2370 FORJ=1338TO1339:POKEJ+L1,3:POKEJ?,94:NEXT
                                               :rem 251
2380 RETURN                                    :rem 173
2390 REM********************************       :rem 200
2400 REM***** MUSIC DATA **************        :rem 99
2410 REM********************************       :rem 193
2420 DATA51,97,51,97,51,97,51,97,43,35,43,35,43,35
     ,57,172,51,97,51,97,51,97                 :rem 203
```

```
2430 DATA51,97,43,35,43,35,43,35,0,0          :rem 18
2440 REM*******************************      :rem 196
2450 REM***** SPRITE DATA{7 SPACES}*******:rem 152
2460 REM*******************************      :rem 198
2470 DATA36,252,18,168,212,20,112,252,89,124,196,1
     26,48,252,56,96,112,96,126               :rem 233
2480 DATA253,192,31,255,128,3,254,0,1,252,0,0,252,
     120,0,249,248,0,255,152,31               :rem 178
2490 DATA255,48,31,222,48,24,0,51,28,0,126,12,0,12
     4,198,0,0,126,0,0,62,0,0                  :rem 59
2500 DATA72,63,36,40,43,21,154,63,14,126,35,62,28,
     63,12,6,28,6,3,223,126,1                  :rem 87
2510 DATA255,248,0,127,192,0,127,128,15,63,128,31,
     223,0,25,255,252,28,255,254              :rem 253
2520 DATA12,123,206,204,0,28,126,0,56,62,0,112,0,0
     ,227,0,0,254,0,0,252                      :rem 99
2530 REM**{2 SPACES}CHARACTER DATA ************
                                               :rem 161
2540 REM**{17 SPACES}********************** MINIA
     TURE HEART ************                   :rem 227
2550 DATA 0,0,40,124,124,56,16,0               :rem 14
2560 REM**{17 SPACES}********************** BRUNO
      CROUCHING ************                   :rem 235
2570 DATA 0,0,0,0,60,126,217,255               :rem 17
2580 REM**{17 SPACES}********************** BRUNH
     ILDE{7 SPACES}************                 :rem 98
2590 DATA 60,126,217,255,60,126,102,231       :rem 123
2600 REM**{17 SPACES}********************** RVS B
     RUNHILDE{3 SPACES}************             :rem 86
2610 DATA 231,102,126,60,255,217,126,60       :rem 116
2620 REM**{16 SPACES}********************** PRIS
     ON WALL{4 SPACES}*************             :rem 31
2630 DATA 255,255,195,195,195,195,255,255      :rem 5
2640 REM**** VALKYRIE THEME ***********        :rem 0
2650 RESTORE:FORJ=1TO(32+63+63+8+8+8+8+8):READQ:NE
     XT:POKE54277,63                           :rem 97
2660 POKE54274,97:POKE54275,5:POKEW1,32:POKEW2,32:
     POKEW3,32                                :rem 249
2670 READQ1:READQ2:READQ3                      :rem 32
2680 IFQ3=0THEN2770                            :rem 79
2690 POKEW1,65:POKEH1,Q1:POKEL1,Q2             :rem 68
2700 FORJ=1TOQ3:NEXTJ                          :rem 82
2710 POKEW1,64:GOTO2670                        :rem 89
2720 REM*** VALKYRIE SONG DATA ********       :rem 53
2730 DATA 10,205,187,10,205,93,14,107,139,14,107,2
     4,14,107,24,17,37,280                    :rem 154
2740 DATA 14,107,280,17,37,139,17,37,24,17,37,24,2
     1,154,280,17,37,280                       :rem 76
2750 DATA 21,154,139,21,154,24,21,154,24,25,177,28
     0,12,216,280                             :rem 230
```

```
2760 DATA 17,37,139,17,37,24,17,37,24,21,154,0
                                          :rem 217
2770 REM**** ASCENDING TONE **********    :rem 236
2780 POKE54277,15:POKEW1,17:POKEW2,21:POKEW3,17
                                          :rem 40
2790 POKEL1,108:POKEL2,108:POKEL3,108     :rem 156
2800 FORJ=8TO31STEP.4                      :rem 21
2810 POKEH1,J-3:POKEH2,J:POKEH3,J*2        :rem 88
2820 REM                                  :rem 176
2830 NEXT:POKEW1,16:POKEW2,16:POKEW3,16   :rem 155
2840 REM*** MIGHTY MOUSE THEME ********    :rem 78
2850 READQ1:READQ2:READQ3:READQ4:READQ5   :rem 215
2860 IFQ1=0THEN2930                        :rem 75
2870 POKEW1,17:POKEW2,17:POKEW3,21:POKEH1,Q1:POKEL
     1,Q2:POKEH2,Q3:POKEL2,Q4             :rem 116
2880 POKEH3,Q1/4:POKEL3,Q2/4              :rem 135
2890 FORJ=1TOQ5:NEXT:POKEW1,16:POKEW2,16:POKEW3,16
     :GOTO2850                            :rem 229
2900 REM**** MIGHTY M. SONG DATA ******   :rem 241
2910 DATA 102,194,12,216,280,86,105,10,205,93,68,1
     49,8,147,280                         :rem 247
2920 DATA 51,97,12,216,93,68,149,8,147,93,86,105,1
     0,205,93,0,0,0,0,0                    :rem 13
2930 POKEW1,21:POKEW2,33:POKEW3,129:POKEH1,102:POK
     EL1,194:POKEH2,6:POKEL2,108          :rem 149
2940 FORJ=25TO2STEP-.03:POKEH3,J:NEXT      :rem 70
2950 RESTORE:RETURN                        :rem 14
2960 REM***** POKE ML ROUTINE *********   :rem 239
2970 DATA 169,0,133,4,169,208,133,5       :rem 193
2980 DATA 169,0,133,6,169,56,133,7        :rem 151
2990 DATA 162,0,160,0,177,4,145,6          :rem 85
3000 DATA 200,192,255,208,247,230,5,230   :rem 113
3010 DATA 7,232,224,16,208,236,96          :rem 87
3020 RESTORE:FORJ=1TO(32+63+63+8+8+8+8+8+63+35):RE
     ADQ:NEXT                             :rem 121
3030 FORJ=49152TO49190:READQ:POKEJ,Q:NEXT  :rem 51
3040 RESTORE:RETURN                         :rem 5
```

Program 2. Assembly Listing—Copy Character Set From ROM To RAM (14336)

```
C000    A9 00     LDA #$00
C002    85 04     STA  $04
C004    A9 D0     LDA #$D0
C006    85 05     STA  $05
C008    A9 00     LDA #$00
C00A    85 06     STA  $06
C00C    A9 38     LDA #$38
```

```
C00E     85 07       STA  $07
C010     A2 00       LDX #$00
C012     A0 00       LDY #$00
C014     B1 04       LDA ($04),Y
C016     91 06       STA ($06),Y
C018     C8          INY
C019     C0 FF       CPY #$FF
C01B     D0 F7       BNE $C014
C01D     E6 05       INC  $05
C01F     E6 07       INC  $07
C021     E8          INX
C022     E0 10       CPX #$10
C024     D0 EC       BNE $C012
C026     60          RTS
```

SuperSprite

Nick Sullivan

Guide "SuperSprite," a falling superhero, safely through four kryptonite barriers. Uses the function keys as cursor controls.

"SuperSprite" makes use of two fascinating aspects of the Commodore 64's sprite graphics. First, the size of a sprite is doubled at the flip of a bit in either or both of its two dimensions. Second, it enables the computer to detect, by PEEKing a single register, collisions between sprites and other graphic data.

The SuperSprite character resembles a super-powered being with arms outstretched in flight. SuperSprite is *not* a steady flyer. And he expands and lengthens suddenly. These characteristics are unfortunate, as his flight path is blocked by barriers of kryptonite, impassible except for narrow gaps. The gaps are movable—luckily, for SuperSprite does not wear a helmet—but moving them requires a deft hand at the controls. And that's where you come in.

You're the keeper of the Spritely Gates, and you get 20 turns to manipulate the barriers so that SuperSprite can make his way to the bottom. If you make it, you increase your score and begin a new turn at the top of the screen.

You'll use the four function keys, each of which controls a gap in one of the four barriers. The f1 key controls the top barrier, the f3 key moves the second barrier, and so on. These keys work as cursors to align the gaps with SuperSprite's path so that he can fly through. If SuperSprite hits a barrier, you lose a turn, and he starts over at the top of the screen. Unshifted, a function key moves its gap to the right; a SHIFTed function key moves the gap to the left. Holding the keys down causes them to repeat.

The soothing SuperSprite soundtrack is created by feeding a slightly altered version of SuperSprite's Y-position data to the frequency registers in the SID (Sound Interface Device) chip. The swooshing sound gets deeper as SuperSprite descends through the barriers.

Special Scoring Technique

Scoring is based on several factors linked through the expressions on lines 210, 590, and 600. The program displays and saves the best score yet achieved on your computer. The record is stored in a location whose contents are displayed in line 10 between the REM keyword and the colon. When you type in the program, the character in this position is the letter *A*. After you have finished entering the program, but before you save it, you should type:

POKE PEEK(44)*256 + PEEK(43) + 5,1

This will properly initialize the high-score record for you.

If you break the record, a special message will remind you at the end of the session to save the program so you can preserve your high score. It's good practice to perform a VERIFY to make sure that the SAVE was successful.

SuperSprite

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
10 REMA:                                          :rem 192
100 REM SEED RANDOM NUMBER GENERATOR, DISABLE CHAR
    ACTER SET SHIFTS                               :rem 49
110 U=RND(-TI):POKE657,128                        :rem 245
120 REM SAVE OLD ENVIRONMENT, BUILD NEW ONE, BUILD
     SPRITE, PRINT INSTRUCTIONS                    :rem 35
130 SP=13:V=53248                                 :rem 171
140 POKE2041,PEEK(V+24):POKEV+24,22               :rem 244
150 POKE2042,PEEK(V+32):POKEV+32,7                :rem 199
160 POKE2043,PEEK(V+33):POKEV+33,7                :rem 203
170 POKE2044,PEEK(646):POKE646,0                   :rem 56
180 POKE2045,PEEK(650):POKE650,128                :rem 155
190 PRINT"{CLR}":GOSUB4000                        :rem 124
200 GOSUB6000:PRINT"{CLR}":POKEV+24,20             :rem 84
210 DEFFNSC(U)=INT(U+U*GL/10)                     :rem 205
220 FORI=1TO16:T$=T$+"{RIGHT}":NEXTI              :rem 241
230 B$="{3 DOWN}"+CHR$(13)+"{RVS}":FORI=0TO39:B$=B
    $+" ":NEXTI                                   :rem 107
240 V=53248:C=55296:S=1024:SD=54272                :rem 151
250 HS=PEEK(43)+PEEK(44)*256+5                    :rem 124
260 T1=135:T2=255:T3=230:T4=133                   :rem 181
270 T5=5:T6=9:T7=.992:T8=42:T9=32                 :rem 116
280 BS=PEEK(HS):CH=65508:AR=780                    :rem 48
290 REM INITIALIZE CURSORS, PRINT GAME SCREEN
                                                  :rem 53
300 E(1)=1269:E(2)=1475:E(3)=1682:E(4)=1888
                                                  :rem 164
310 POKEV+21,0:GOSUB2000                          :rem 127
```

```
320 REM MAKE SPACE FOR SCORE ON SCREEN, SET SOUND
    {SPACE}CHIP, CLEAR KEYBOARD BUFFER      :rem 38
330 FORI=1080TO1100:POKEI,32:NEXT           :rem 237
340 POKESD+4,16:POKESD+11,16:POKESD+18,16   :rem 150
350 POKESD+6,240:POKESD+13,240:POKESD+20,240
                                            :rem 33
360 POKESD+24,15                            :rem 130
370 GETA$:IFA$<>""THEN370                   :rem 148
380 :                                       :rem 213
390 REM BEGIN 20-TURN GAME LOOP,SET PART SCORE TO
    {SPACE}ZERO                             :rem 191
400 FORGL=1TO20:PS=0                        :rem 203
410 REM POSITION SPRITE, UNEXPAND IT, ZERO THE COL
    LISION DETECT REGISTER                   :rem 8
420 POKEV,23:POKEV+1,49                     :rem 96
430 POKEV+23,0:POKEV+31,0                   :rem 178
440 REM PRINT SCORE INFO, TURN ON SOUND     :rem 76
450 PRINT"{HOME}{DOWN}"T$"TURN:"GL"{2 SPACES}SCORE
    :"TS                                    :rem 1
460 PRINT"{DOWN}"T$"{5 RIGHT}BEST YET:"PEEK(HS)-1
                                            :rem 146
470 POKESD+4,17:POKESD+11,17:POKESD+18,17   :rem 157
480 REM TURN ON SPRITE, TAKE 1 TURN, GOTO 570 IF G
    AME ABORTED (FQ SET)                    :rem 135
490 POKEV+21,1:GOSUB1000:IFFQ=1THEN550      :rem 31
500 REM SCROLL SPRITE OFF SCREEN IF NO COLLISION,
    {SPACE}OTHERWISE SCREECH TO A HALT      :rem 130
510 IFPEEK(V+1)>T3THENFORI=PEEK(V+1)TO250:POKEV+1,
    I:NEXTI:GOTO550                         :rem 111
520 POKESD+4,129:POKESD+11,33:POKESD+18,33:rem 201
530 FOR I=72TO20STEP-2:POKESD+1,I:POKESD+8,I-3:POK
    ESD+15,I-2:NEXTI                        :rem 62
540 REM TURN OFF SOUND. JUMP TO EXIT IF FQ SET
                                            :rem 131
550 POKESD+4,16:POKESD+11,16:POKESD+18,16   :rem 153
560 POKESD+1,0:POKESD+8,0:POKESD+15,0       :rem 197
570 IFFQ=1THENFQ=0:GL=20:NEXT:POKEV+21,0:GOTO670
                                             :rem 1
580 REM CALCULATE PART SCORE FROM SPRITE Y POSITIO
    N, MODIFY, ADD TO TOTAL SCORE           :rem 39
590 PS=FNSC(INT((PEEK(V+1)-71)/40))         :rem 214
600 TS=TS+INT((PS↑1.4+PS)/2)                :rem 156
610 POKEV+21,0:POKEV+16,0                   :rem 179
620 NEXTGL                                  :rem 106
630 REM UPDATE HIGH SCORE RECORD, ZERO SOUND CHIP,
     GO TO EXIT (3000)                      :rem 103
640 IFPEEK(HS)<TS+1THENPOKEHS,TS+1          :rem 161
650 FORI=0TO24:POKE SD+I,0:NEXTI            :rem 211
660 FORI=1TO500:NEXTI                        :rem 52
```

```
670 GOSUB3000:CLR:GOTO210                                  :rem 1
680 :                                                      :rem 216
970 REM SUBROUTINE--ONE TURN OF THE GAME                   :rem 169
980 :                                                      :rem 219
990 REM DECIDE WHICH SPRITE POSITION REGISTER TO A
    LTER (W) AND BY HOW MUCH (U)                           :rem 58
1000 U2=RND(1):W=V+INT(U2*2):U=GL/7+2                      :rem 123
1010 REM Y-EXPAND SPRITE IF EXPRESSION TRUE
                                                           :rem 162
1020 IFU2>T7ANDPEEK(V+1)<T1THENPOKEV+23,1                  :rem 255
1030 REM SET MSB OF SPRITE X-POSITION IF NECESSARY
                                                           :rem 24
1040 IFPEEK(W)+U>T2THENPOKEV+16,1:POKEV,PEEK(V)+U-
     T2:GOTO1070                                           :rem 45
1050 REM UPDATE SPRITE POSITION.{2 SPACES}EXIT IF
     {SPACE}COLLISION DETECTED OR 4TH GAP CROSSED
                                                           :rem 250
1060 POKEW,PEEK(W)+U                                       :rem 198
1070 IF(PEEK(V+31)AND1)ORPEEK(V+1)>T3THE?NRETURN
                                                           :rem 164
1080 REM CHANGE FREQUENCIES ACCORDING TO SPRITE Y
     {SPACE}POSITION AND RANDOM NUMBER U2                  :rem 67
1090 U=232-PEEK(V+1):POKESD+1,U*2/3:POKESD+8,U+3*U
     2:POKESD+15,U+3                                       :rem 70
1100 REM POLL KEYBOARD. EXIT ON *; PAUSE ON ' '; G
     OTO1000 IF NOT A FUNCTION KEY                         :rem 70
1110 SYSCH:U=PEEK(AR):IFU=T8THENTS=0:FQ=1:RETURN
     {17 SPACES}:REM EXIT                                  :rem 253
1120 IFU=T9THENGOSUB8000:REM TAKE 5                        :rem 97
1130 IFU<T4THEN1000:REM INPUT<F1                           :rem 190
1140 REM MOVE LINE RIGHT IF KEY NOT SHIFTED (1120)
     , LEFT IF SHIFTED (1140)                              :rem 16
1150 U=U-132:IFU<T5THEN1180:REM INPUT F1-F4:rem 25
1160 IFU<T6THEN1200:REM INPUT F5-F8                        :rem 56
1170 GOTO1000:REM INPUT>F8                                 :rem 45
1180 E(U)=E(U)+(E(U)=1100+200*U-(U>2)):POKEE(U),22
     3:POKEE(U)-1,160                                      :rem 78
1190 POKE E(U)+4+(U>2),32:POKEE(U)+5+(U>2),95:E(U)
     =E(U)+1:GOTO1000                                      :rem 130
1200 U=U-4:E(U)=E(U)-(E(U)=1064+200*(U)):POKEE(U)+
     3+(U>2),95                                            :rem 227
1210 POKEE(U)+4+(U>2),160:POKEE(U)-1,32:POKEE(U)-2
     ,223:E(U)=E(U)-1:GOTO1000                             :rem 217
1970 REM SUBROUTINE--PRINT GAME SCREEN                     :rem 134
1980 :                                                     :rem 12
1990 REM LINE COLOURS SET IN LINE 2000. LAST COLOU
     R IS FOR PRINTED MESSAGES                             :rem 154
2000 PRINT"{CLR}{2 DOWN}{PUR}"B$"{GRN}"B$"{7}"B$"
     {3}"B$"{HOME}{BLK}"                                   :rem 125
```

```
2010 FORI=1TO4:POKEE(I)-1,223:POKEE(I)+3-(I<3),95:
     FORJ=E(I)TOE(I)+2-(I<3)                  :rem 179
2020 POKEJ,32:NEXTJ,I:RETURN{7 SPACES}         :rem 32
2030 :                                         :rem 255
2980 REM SUBROUTINE--EXIT OR RE-ENTRY          :rem 82
2990 :                                         :rem 14
3000 PRINT"{CLR}{DOWN}"T$"{2 RIGHT}YOUR SCORE:"TS
                                               :rem 218
3010 PRINT"{DOWN}"T$"{4 RIGHT}BEST YET:";PEEK(HS)-
     1                                         :rem 218
3020 PRINT"{3 DOWN}{RIGHT}PLAY AGAIN (Y/N)?
                                               :rem 192
3030 GETA$:IFA$=""THEN3030                     :rem 175
3040 IFA$="Y"THENRETURN                        :rem 164
3050 IFA$="N"THEN3070                          :rem 132
3060 GOTO3030                                  :rem 200
3070 IF NOT(BS<PEEK(HS))THEN3100               :rem 112
3080 PRINT:PRINT" CONGRATULATIONS, YOU BROKE THE R
     ECORD.                                     :rem 40
3090 PRINT:PRINT" BE SURE TO SAVE THE PROGRAM.
                                                 :rem 1
3100 PRINT:PRINT" SEE YOU AROUND.               :rem 11
3110 PRINT"{4 DOWN}"                           :rem 218
3120 REM RESTORE PREVIOUS ENVIRONMENT          :rem 160
3130 POKEV+24,PEEK(2041):POKEV+32,PEEK(2042):POKEV
     +33,PEEK(2043)                            :rem 186
3140 POKE646,PEEK(2044):POKE650,PEEK(2045):POKE657
     ,0                                        :rem 219
3150 END                                       :rem 160
3160 :                                           :rem 4
3980 REM SUBROUTINE--BUILD SPRITE AT PAGE 13 (LOCA
     TIONS 832-895 IN TAPE BUFFER)             :rem 103
3990 :                                          :rem 15
4000 FOR I=0TO41:READU:POKE(64*SP)+I,U:NEXT
                                               :rem 115
4010 FORI=42TO63:POKE(64*SP)+I,0:NEXT          :rem 222
4020 POKE2040,SP:POKEV+23,1:POKEV+29,1:POKEV+39,0:
     RETURN                                    :rem 180
4030 RETURN                                    :rem 167
4040 DATA 248,0,0,62,0,0,7,128,0,1,224          :rem 43
4050 DATA 0,0,120,0,0,62,0,0,15,112            :rem 132
4060 DATA 0,7,248,0,7,248,0,6,120,0            :rem 163
4070 DATA 6,12,0,6,6,0,3,6,0,1,128             :rem 103
4080 :                                            :rem 6
4980 REM DATA FOR INSTRUCTIONS PAGE            :rem 140
4990 :                                          :rem 16
5000 DATA"YOUR OBJECTIVE IS TO MANEUVER THE
                                               :rem 238
5010 DATA"HORIZONTAL LINES SO THAT SUPERSPRITE
                                               :rem 192
```

```
5020 DATA"MAY FLY SAFELY THROUGH THE GAPS.:rem 244
5030 DATA"THE FOUR LINES ARE CONTROLLED BY THE
                                          :rem 102
5040 DATA"FOUR FUNCTION KEYS ON YOUR RIGHT.
                                          :rem 123
5050 DATA"                                  :rem 6
5060 DATA"PRESSING ONE OF THESE KEYS WILL CAUSE
                                          :rem 199
5070 DATA"THE CORRESPONDING LINE TO SLIDE RIGHT.
                                          :rem 174
5080 DATA"THE SAME KEY SHIFTED WILL CAUSE ITS
                                           :rem 25
5090 DATA"LINE TO SLIDE LEFT.{2 SPACES}PRESS THE S
     PACE                                 :rem 249
5100 DATA"BAR TO PAUSE, '*' TO ABORT.     :rem 229
5110 DATA"                                  :rem 3
5120 DATA"A GAME CONSISTS OF 20 TURNS. POINTS ARE
                                           :rem 75
5130 DATA"AWARDED FOR EVERY GAP SAFELY TRAVERSED.
                                          :rem 233
5140 DATA"THE PAYOFFS INCREASE WITH THE NUMBER OF
                                           :rem 68
5150 DATA"GAPS TRAVERSED ON A TURN, AND WITH THE
                                           :rem 37
5160 DATA"NUMBER OF TURNS TAKEN.{2 SPACES}SUPERSPR
     ITE'S                                :rem 131
5170 DATA"FLIGHT SPEED AND NATURAL WAYWARDNESS
                                          :rem 122
5180 DATA"ALSO INCREASE AS THE GAME PROGRESSES."
                                          :rem 111
5190 DATA"                                  :rem 11
5200 DATA"THE MAXIMUM SCORE IS 253. GOOD LUCK.
                                          :rem 104
5210 DATA"                                  :rem 4
5220 DATA"{4 SPACES}PRESS{SHIFT-SPACE}SPACE
     {SHIFT-SPACE}BAR{SHIFT-SPACE}TO START
     {SHIFT-SPACE}GAME.                   :rem 204
5230 :                                      :rem 4
5980 REM SUBROUTINE--PRINT INSTRUCTIONS    :rem 102
5990 :                                      :rem 17
6000 POKEV,23:POKEV+1,49:POKEV+21,1         :rem 58
6010 GOSUB7000:GOSUB7000                   :rem 143
6020 FORW=1TO23:READU$:PRINTU$:GOSUB7000:NEXTW
                                          :rem 220
6030 POKEV+21,0:POKEV,23:POKEV+1,49:POKEV+23,0:POK
     EV+29,0                              :rem 152
6040 GETA$:IFA$<>CHR$(32)THEN 6040        :rem 103
6050 RETURN                               :rem 171
6060 :                                      :rem 6
```

```
6980 REM SUBROUTINE--FLY SPRITE DOWN 1 PRINT LINE
     {SPACE}(AHEAD OF INSTRUCTIONS)        :rem 243
6990 :                                      :rem 18
7000 FORI=1TO8:POKEV,PEEK(V)+1              :rem 83
7010 POKEV+1,PEEK(V+1)+1:NEXT              :rem 210
7020 POKEV,PEEK(V)+1:RETURN                :rem 188
7030 :                                       :rem 4
7980 REM SUBROUTINE--ANSWER THE PHONE       :rem 81
7990 :                                      :rem 19
8000 POKE SD+24,0                          :rem 123
8010 SYSCH:IFPEEK(AR)<>32THEN8010          :rem 252
8020 POKE SD+24,15                         :rem 179
8030 RETURN                                :rem 171
```

Olympiad

Kevin Woram and Mike Buhidar, Jr.

In this mythical struggle between a magician and a king, you decide the fate of the realm with your joystick. For two players and two joysticks.

Long ago, Admar, a magician of great power, served the king of Denbar as an advisor in matters of policy. Through the years, Admar's power grew so that the king began to fear him. Foolishly, the king decided that because of his power, Admar could no longer be trusted, and he plotted to kill the magician.

Admar, actually still loyal to the king, learned of the king's plot. Fearing for his own life, he fled the capital with a legion of his own warriors.

The king followed with an army of his own and attacked Admar's stronghold. There were heavy casualties on both sides. It took time, but finally the king and Admar realized that continued battle would result in nothing more than a general bloodletting of the entire countryside.

An Enchanted Arena

So it was agreed that an enchanted arena should be built where the king's black knights would do mock battle with Admar's red knights. Whoever's knights won would claim the kingdom as his own.

You and a friend control the actions of the knights as they fight for their masters. Movement in all eight directions is controlled by the joysticks. The red knight is controlled by the joystick in port 1, the black knight by the joystick in port 2.

The knights have also been given 20 magical arrows that stun on contact. The arrows are launched by pressing the fire button. When your knight has used all of his arrows, your only defense is to run.

Teleportation Grids

To add an element of randomness to the battle, three teleportation grids have been added to the arena. A large one is in the center of the arena, while the other two are in corners. Either knight may use any of the three grids. When any

warrior steps onto one of these grids, he is instantly teleported to a random position in the arena.

There are also two doors on each side of the arena, which allow you to move directly from one side to the other, in effect wrapping around the screen. You can even shoot arrows through these doors. If the opposing knight happens to be standing in front of the door on the left side, for instance, and you fire *through* the door on the right, you can stun him.

Typing Olympiad

"Olympiad" makes extensive use of keyboard graphics in drawing the arena display. To avoid confusion and possible typing errors, please refer to "How to Type In Programs," Appendix B, before you attempt to enter this program. Using "The Automatic Proofreader," found in Appendix C, will insure that you type Olympiad correctly the first time. Make sure you read the explanation and have a copy of the Proofreader program available before you start typing.

Pay close attention to lines 3010–3220 as you type them in—especially to the places where program lines are divided on the page. If any spaces are to be left after the characters on one line of the page, the correct number will be indicated in braces at the beginning of the next line. Unless you are specifically instructed to type spaces, do not do so. For instance, in the statement below, there should be only two spaces (as specified in the braces) between the SHIFTed characters on the first line and those on the second.

```
3180 PRINT"{UP}UCK UCCCCCCCI{2 SPACES}UCCCCCCCCI
     {2 SPACES}UCCCCCCCI{3 SPACES}B"
```

Note, however, that spaces are sometimes called for *within* a line. In the example above, there should be a space after the first SHIFTed *K*, before the next SHIFTed *U* is entered. Single spaces are not indicated by braces—there's just a gap. Whenever more than one space is to be inserted, you'll see the number in braces (as in the first line, where you see {2 SPACES}).

Olympiad

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
1 POKE53280,2:POKE53281,1:GOTO1000          :rem 189
2 DIM X(15),CS(15),D2(15),C2(15)             :rem 69
4 CLR:N=15:B=32:FJ=56320:FT=56321:CO=54272:JB=16:H
  P=102:GOSUB500                             :rem 223
```

```
10 RN=1:RO=1104:FB=1030:FO=1054:LB=1032:LR=1056:GO
   SUB3000                                    :rem 15
20 DX(0)=0:DX(1)=-40:DX(2)=40:DX(4)=-1:DX(5)=-41:D
   X(6)=39:DX(8)=1:DX(9)=-39                  :rem 54
22 D2(0)=0:D2(1)=-40:D2(2)=40:D2(4)=-1:D2(5)=-41:D
   2(6)=39:D2(8)=1:D2(9)=-39                   :rem 8
25 DX(10)=41:CS(0)=192:CS(1)=194:CS(2)=195:CS(4)=1
   93:CS(5)=198:CS(6)=197                     :rem 118
27 D2(10)=41:C2(0)=193:C2(1)=194:C2(2)=195:C2(4)=1
   93:C2(5)=198:C2(6)=197                     :rem 141
30 CS(8)=192:CS(9)=196:CS(10)=199             :rem 244
32 C2(8)=192:C2(9)=196:C2(10)=199             :rem 147
34 RESTORE:GOSUB400:FORNP=13824TO13983:READMD:POKE
   NP,MD:NEXT                                 :rem 166
36 PRINT"{2 UP}";SPC(JB);"{7 SPACES}"         :rem 217
50 OP=1105:O2=1982:POKEOP,195:POKEO2,194:POKEOP+CO
   ,0:POKE02+CO,5                             :rem 101
60 IF AT+NA=0 THEN POKEOP,B:POKEO2,B:RN=RN-1:GOTO3
   4                                          :rem 99
61 JV=N-(PEEK(FJ)ANDN):FR=PEEK(FJ)ANDJB:CS(0)=CS(J
   V):UP=OP+DX(JV)                            :rem 51
65 IFPEEK(UP)<>BTHENGOSUB4000                 :rem 68
70 POKEOP,B:POKEUP+CO,0:POKEUP,CS(JV):OP=UP:rem 70
75 IFFR<>JBTHENGOSUB100                       :rem 217
80 J2=N-(PEEK(FT)ANDN):F2=PEEK(FT)ANDJB:C2(0)=C2(J
   2):U2=O2+D2(J2)                            :rem 24
85 IFPEEK(U2)<>BTHENGOSUB4100                 :rem 41
90 POKEO2,B:POKEU2+CO,2:POKEU2,C2(J2):O2=U2
                                              :rem 111
95 IFF2<>JBTHENGOSUB110                       :rem 188
97 GOTO60                                      :rem 15
99 REM SHOOT ARROW                            :rem 110
100 IFNA=0THENRETURN                           :rem 43
101 NA=NA-1:BP=INT(NA/10):IFBP>1THENBP=1       :rem 168
102 PRINT"{HOME}{BLK}";NA:POKE1026+BP,B:D=DX(JV):J
    C=CS(JV):GOSUB200                          :rem 77
105 AP=UP+D:C1=0:GOTO115                       :rem 157
110 IFAT=0THENRETURN                           :rem 50
111 AT=AT-1:BT=INT(AT/10):IFBT>1THENBT=1       :rem 199
112 PRINT"{HOME}{RED}";SPC(36);AT:POKE1062+BT,B:D=
    D2(J2):JC=C2(J2):GOSUB200                  :rem 48
114 AP=U2+D:C1=2                               :rem 119
115 AD=JC+8:IFPEEK(AP)<>BTHENRETURN            :rem 228
120 FORA=1TO15:NP=AP+D                         :rem 71
125 AC=NP+CO                                   :rem 180
130 IFPEEK(NP)<>BTHEN300                       :rem 181
140 POKEAP,B:POKEAC,C1:POKENP,AD:AP=NP:NEXT:POKEAP
    ,B:RETURN                                  :rem 169
199 REM STILL CHECKER                           :rem 4
```

```
200 IFD<>0THENRETURN                                    :rem 30
210 IFJC=194THEND=-40:RETURN                            :rem 229
220 IFJC=195THEND=40:RETURN                             :rem 186
230 IFJC=193THEND=-1:RETURN                             :rem 179
240 IFJC=198THEND=-41:RETURN                            :rem 237
250 IFJC=197THEND=39:RETURN                             :rem 199
260 IFJC=192THEND=1:RETURN                              :rem 136
270 IFJC=196THEND=-39:RETURN                            :rem 245
280 D=41:RETURN                                         :rem 154
299 REM DEATH                                           :rem 238
300 IFPEEK(NP)<192THENPOKEAP,B:RETURN                   :rem 133
310 IFC1=0THEN330                                       :rem 201
312 POKELB,B:LB=LB-1:GOSUB600                            :rem 2
315 IFLB=FB-1THEN6000                                   :rem 206
317 GOTO34                                               :rem 59
330 POKELR,B:LR=LR-1:GOSUB610                            :rem 51
335 IFLR=FO-1THEN6010                                   :rem 238
340 GOTO34                                               :rem 55
400 NA=20:AT=20:PRINT"{HOME}{BLK}";NA;SPC(32);"
    {RED}";AT                                           :rem 234
410 PRINT"{BLU}{HOME}{2 DOWN}{RVS}";SPC(16);"ROUND
    ";RN;"{OFF}":RN=RN+1:RETURN                         :rem 197
500 REM SOUND INITIALIZATION                             :rem 42
510 S=54272:FORQ=STOS+24:POKEQ,0:NEXT                    :rem 66
520 POKES+24,15:POKES+5,17:POKES+6,248                  :rem 211
525 POKES,150:POKES+1,150                               :rem 186
530 RETURN                                              :rem 120
600 DP=UP:OM=U2:GOTO620                                 :rem 177
610 DP=U2:OM=UP                                         :rem 167
620 POKEAP,B:POKEOM,B:GOSUB7000:FORK=210TO208STEP-
    1:POKEDP,K                                          :rem 65
630 FORH=1TO100:NEXT:NEXT:POKEDP,211:POKEDP,B:POKE
    UP,B:POKEU2,B:RETURN                                :rem 120
1000 REM CHR. SET LOADER                                 :rem 83
1010 PRINT"{CLR}{BLK}LOADING CHARACTER SET INTO ME
     MORY. . .":PRINTCHR$(142)                          :rem 122
1020 POKE52,48:POKE56,48:CLR:G=56334                     :rem 59
1030 POKEG,PEEK(G)AND254                                :rem 145
1040 POKE1,PEEK(1)AND251                                 :rem 99
1045 IFPEEK(13983)=102THEN1060                          :rem 157
1050 FORI=0TO2047:POKEI+12288,PEEK(I+53248):NEXT
                                                        :rem 74
1060 POKE1,PEEK(1)OR4                                   :rem 207
1070 POKEG,PEEK(G)OR1                                   :rem 249
1080 POKE53272,(PEEK(53272)AND240)+12                   :rem 232
1090 FORNP=13824TO13983:READMD:POKENP,MD:DC=DC+MD:
     NEXT                                               :rem 158
1095 IFDC<>13392THENPRINT"ERROR IN DATA. . .":STOP
                                                        :rem 166
```

```
1100 GOTO2                                          :rem 45
1999 REDEFINED CHARACTERS                           :rem 66
2000 DATA102,227,241,159,159,241,227,102  :rem 216
2010 DATA102,199,143,249,249,143,199,102  :rem 235
2020 DATA126,219,153,24,60,231,231,126    :rem 113
2030 DATA126,231,231,60,24,153,219,126    :rem 114
2040 DATA60,6,207,253,201,201,124,60        :rem 6
2050 DATA60,62,147,147,191,243,96,60        :rem 36
2060 DATA60,96,243,191,147,147,62,60        :rem 37
2070 DATA60,124,201,201,253,207,6,60        :rem 9
2082 DATA0,132,66,63,66,132,0,0,0,33,66,252,66,33,
     0,0,16,56,84,16,16,16,40,68           :rem 233
2084 DATA68,40,16,16,16,84,56,16,7,3,5,8,16,224,32
     ,32,4,4,7,8,16,160,192,224            :rem 202
2086 DATA224,192,160,16,8,7,4,4,32,32,224,16,8,5,3
     ,7                                     :rem 39
2088 DATA0,0,8,16,4,16,0,0                  :rem 26
2090 DATA0,0,20,10,32,20,0,0                :rem 99
2092 DATA68,9,32,132,1,40,130,17            :rem 78
2094 DATA0,0,0,0,0,0,0,0                   :rem 157
2999 REM PLAYFIELD                          :rem 91
3000 PRINT"{CLR}{RED}{12 SPACES}{RVS}O L Y M P I A
      D{OFF}{14 SPACES}"                    :rem 70
3001 NA=20:AT=20:PRINT"{HOME}";NA;SPC(B);AT
                                           :rem 204
3002 FORL=1024TO1035:POKEL+CO,0:NEXT:FORL=1057TO10
     62:POKEL+CO,2:NEXT                    :rem 219
3004 FORL=FBTOLB:POKEL,194:NEXT:FORL=FOTOLR:POKEL,
     194:NEXT                              :rem 96
3010 PRINT"{UP}{BLK}UC{R}CCCCCCCCCCCCCCCCCCCCCCCCC
     CCCCCCCCC{R}CI"                       :rem 36
3020 GOSUB3990:POKE1106+CO,0:POKE1106,66:POKE1141+
     CO,0:POKE1141,66:POKE1142,HP          :rem 51
3030 POKE1142+CO,2:PRINT"{UP}B {X} UCCCCI
     {2 SPACES}UCCCCCI{2 SPACES}UCCCCCI{2 SPACES}U
     CCCCI {Z} B"                          :rem 144
3040 PRINT"{UP}B{3 SPACES}JCCCCK{2 SPACES}JCCCCCK
     {2 SPACES}JCCCCCK{2 SPACES}JCCCCK{3 SPACES}B"
                                           :rem 183
3050 GOSUB3990                              :rem 29
3060 PRINT"{UP}B{3 SPACES}UCCCCCCCI{2 SPACES}UCCCC
     CCCCI{2 SPACES}UCCCCCCCI{3 SPACES}B"  :rem 75
3070 PRINT"{UP}B{3 SPACES}JCCCCCCCK{2 SPACES}JCCCC
     CCCCK{2 SPACES}JCCCCCCCK UCK"         :rem 210
3080 PRINT"{UP}JCI";SPC(34);"JCC"           :rem 43
3090 PRINT"{UP}CCK{3 SPACES}UCCCCI{4 SPACES}UC{W}
     {2 SPACES}{Q}CI{4 SPACES}UCCCCI"      :rem 85
3100 PRINT"{6 SPACES}JCCCCK{2 SPACES}UCK{6 SPACES}
     JCI{2 SPACES}JCCCCK{3 SPACES}UCC"     :rem 235
```

```
3110 PRINT"{UP}CCI{11 SPACES}B{10 SPACES}B
     {11 SPACES}JCI"                          :rem 80
3120 PRINT"{UP}UCK UCCI UCCI B{4 SPACES}{BLK}{+}
     {RED}{+}{BLK}{4 SPACES}B UCCI UCCI{3 SPACES}B
     "                                        :rem 233
3130 PRINT"{UP}B{3 SPACES}JCCK JCCK B{4 SPACES}
     {RED}{+}{BLK}{+}{4 SPACES}B JCCK JCCK UCK"
                                              :rem 54
3140 PRINT"{UP}JCI{11 SPACES}B{10 SPACES}B
     {11 SPACES}JCC"                          :rem 84
3150 PRINT"{UP}CCK{3 SPACES}UCCCCI{2 SPACES}JCI
     {6 SPACES}UCK{2 SPACES}UCCCCI{6 SPACES}"
                                              :rem 137
3160 PRINT"{UP}{6 SPACES}JCCCCK{4 SPACES}JC{W}
     {2 SPACES}{Q}CK{4 SPACES}JCCCCK{3 SPACES}UCC"
                                              :rem 66
3170 PRINT"{UP}CCI{34 SPACES}JCI"             :rem 210
3180 PRINT"{UP}UCK UCCCCCCCI{2 SPACES}UCCCCCCCCI
     {2 SPACES}UCCCCCCCI{3 SPACES}B"          :rem 239
3190 PRINT"{UP}B{3 SPACES}JCCCCCCCK{2 SPACES}JCCCC
     CCCCK{2 SPACES}JCCCCCCCK{3 SPACES}B"     :rem 52
3200 GOSUB3990                                :rem 26
3210 PRINT"{UP}B{3 SPACES}UCCCCI{2 SPACES}UCCCCCI
     {2 SPACES}UCCCCCI{2 SPACES}UCCCCI{3 SPACES}B"
                                              :rem 218
3220 PRINT"{UP}B {S} JCCCCK{2 SPACES}JCCCCCK
     {2 SPACES}JCCCCCK{2 SPACES}JCCCCK {A} B"
                                              :rem 21
3230 PRINT"{UP}B":POKE1945,HP                 :rem 236
3245 FORL=56215TO56295:POKEL,0:NEXT           :rem 121
3250 POKE1983,93:POKE1984,74:FORL=1985TO2022:POKEL
     ,67:NEXT                                 :rem 237
3260 POKE1986,113:POKE1946,66:POKE2021,113:POKE198
     1,66:POKE2023,75:RETURN                  :rem 13
3990 PRINT"{UP}B";SPC(38);"B":RETURN          :rem 49
3999 REM HIT DATA                             :rem 193
4000 IFPEEK(UP)<>HPTHENUP=OP:RETURN           :rem 74
4010 RF=INT(RND(1)*879)+RO:IFPEEK(RF)<>BTHEN4010
                                              :rem 97
4020 UP=RF:POKEUP+CO,0:POKEOP,B:MP=UP:JP=JV:GOSUB5
     000                                      :rem 198
4100 IFPEEK(U2)<>HPTHENU2=O2:RETURN           :rem 241
4110 R2=INT(RND(1)*879)+RO:IFPEEK(R2)<>BTHEN4110
                                              :rem 59
4120 U2=R2:POKEU2+CO,2:POKEO2,B:MP=U2:JP=J2:GOSUB5
     000:RETURN                               :rem 51
5000 FORMN=208TO210:POKEMP,MN:FORW=1TO150:NEXT:NEX
     T:POKEMP,CS(JP):RETURN                   :rem 255
5999 REM END ROUTINE                          :rem 193
```

```
6000 WN$=" RED ":LS$=" BLACK ":T1=4:T2=4:GOTO6020
                                                :rem 98
6010 WN$=" BLACK ":LS$=" RED ":T1=3:T2=5    :rem 40
6020 PRINT"{CLR}";TAB(T1);"{DOWN}{BLK}THE";WN$;"KN
     IGHTS WERE VICTORIOUS!{DOWN}"           :rem 44
6030 PRINTTAB(T2);"THEY DEFEATED THE";LS$;"KNIGHTS
     {DOWN}"                                :rem 118
6040 PRINTTAB(14);"IN";RN-1;"ROUNDS"        :rem 141
6060 PRINT"{15 DOWN}{4 SPACES}PRESS <<SPACEBAR>> T
     O PLAY AGAIN{3 SPACES}"                :rem 151
6065 GETI$:IFI$<>" "THEN6065                 :rem 18
6070 GOTO4                                   :rem 58
7000 REM DEATH SOUND                        :rem 154
7010 POKES+4,129:FORQ=1TO30:NEXT:POKES+4,128
                                            :rem 178
7040 RETURN                                 :rem 171
```

Burn Rubber

Jonathan Cook

"Burn Rubber," a simple but exciting race game, pits you against the clock as you roar down straightaways, around hairpin turns, and through dangerous S-curves.

Racing your Formula-I car around the complex track, competing against the clock, you try to finish as many laps as possible before crashing. Although a relatively simple game to program and play, "Burn Rubber" takes advantage of the Commodore 64's sprite graphics and sound effects. You have to maneuver your car carefully, taking the turns and curves as fast as possible to post the best lap time. You can even slow and speed up your car, to pound down the straightaways or to slowly make the hairpin turns.

Not only are you racing against the clock, but you're also trying to collect as many points as possible. Your score is based on the average speed and the number of laps you complete. The faster you go and the more laps you finish before the eventual crash, the better your score will be. The computer keeps track of the points per race, the time, the lap total, and even the time per lap.

From the Starting Line

You control the car by pressing keys. The keys and their controls are:

Key	Controls
P	Move the car up (north)
.	Move the car down (south)
L	Move the car left (west)
;	Move the car right (east)
1	Slower speed
2	Faster speed

To start the race, just press the L key to move your car to the left until it's off the screen. You'll then see a display on the right side of the screen that tells you the points, time elapsed, laps completed, and time per lap. All except the time elapsed should read 0. Press the space bar, as the prompt at the bottom of the screen suggests, to restart the clock. Now your car again reappears on the screen, ready to begin.

Press the ; (semicolon) key to move the car to the right. You can race either clockwise or counterclockwise; the latter seems more natural, somehow. Immediately, you have to maneuver the car through a tricky S-curve. From there, the track twists and turns until it brings you back to the starting point. Use the 1 and 2 number keys to control the speed. Pressing repeatedly slows or speeds up your car's engine. You'll hear the difference in the sound. The straightaways are where you can open up the throttle; unless you're a peerless driver, you'll probably have to slow down for some of the turns. If you want to quit and find out your score, time, and laps completed, just drive off to the left near the bottom of the screen. The display changes and your new totals appear.

The race ends when your car crashes or is driven off the track. To start a new race, just press the space bar again. Your car reappears and is ready to take on the circuit again.

There's always a new challenge just around the next corner.

Burn Rubber

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
100 DIM F1(5),F2(5):PRINT"{CLR}"                :rem 48
900 PRINT"{CLR}":POKE53272,23                   :rem 250
1000 PRINT TAB(14) "{WHT}{RVS}BURN{SHIFT-SPACE}RUB
     BER{OFF}"                                  :rem 97
1010 PRINT"{YEL}{DOWN}{RIGHT}THE OBJECT OF THIS GA
     ME IS TO DRIVE"                            :rem 23
1020 PRINT"{RIGHT}AROUND THE TRACK AS FAST AS POSS
     IBLE"                                      :rem 135
1030 PRINT"{RIGHT}WITHOUT CRASHING. YOU CAN GO AS
     {SPACE}MANY"                               :rem 17
1040 PRINT"{RIGHT}LAPS AS YOU LIKE. YOU CAN RACE F
     OR"                                        :rem 24
1050 PRINT"{RIGHT}POINTS OR FOR TIME. TO END A LAP
      OR"                                       :rem 47
1060 PRINT"{RIGHT}TO RESET THE CLOCK YOU MUST DRIV
     E THE"                                     :rem 201
1070 PRINT"{RIGHT}CAR OFF THE SCREEN AT THE LOWER
     {SPACE}LEFT."                              :rem 96
1072 PRINT"{RIGHT}THERE IS NO TOP SPEED, BUT THE F
     ASTER"                                     :rem 10
1073 PRINT"{RIGHT}YOU GO, THE HARDER IT IS TO DRIV
     E.{2 DOWN}"                                :rem 181
1080 PRINT TAB(15) "UP={RVS}P{OFF}{DOWN}" :rem 143
```

```
1090 PRINT TAB(11) "LEFT={RVS}L{OFF}{3 SPACES}
     {RVS};{OFF}=RIGHT{DOWN}"                    :rem 40
1100 PRINT TAB(13) "DOWN={RVS}.{OFF}{DOWN}"
                                                 :rem 119
1110 PRINT TAB(11) "{RVS}1{OFF}=SLOWER"          :rem 12
1120 PRINT TAB(11) "{RVS}2{OFF}=FASTER{DOWN}"
                                                  :rem 8
2990 PRINT TAB(13) "{RVS}{WHT}SPACE{SHIFT-SPACE}TO
     {SHIFT-SPACE}START{OFF}{BLU}"               :rem 214
3000 GET CH$:IF CH$="" THEN 3000                 :rem 61
3010 PRINT"{CLR}":POKE53272,21                   :rem 35
4700 VL=54296:W=54276:A=54277:SR=54278:H=54273:L=5
     4272:TI$="000000"                           :rem 152
4710 FORCS=54272 TO 54296:POKECS,0:NEXT          :rem 11
4800 POKE53281,0                                 :rem 90
4805 REM SET SOUND,TRACK COLOR                   :rem 74
4810 FORK=1 TO 5:READF1(K):READF2(K):NEXTK:rem 220
4900 FOR I=55296 TO 56295:POKEI,14:NEXT          :rem 175
4905 :                                           :rem 12
4910 REM SET UP SPRITE 0 TO LOOK LIKE A          :rem 136
4920 REM CAR AND PUT IT AT START LOCAT-          :rem 181
4930 REM ION. ALSO SET UP 1ST TRACK              :rem 213
4940 REM WITH GOSUB 20000.                       :rem 145
4950 :                                           :rem 12
5000 V=53248:POKEV+21,1                           :rem 8
5100 FOR S1=12288 TO 12350:READQ1:POKES1,Q1:NEXT
                                                 :rem 253
5110 FOR S2=12352 TO 12414:READQ2:POKES2,Q2:NEXT
                                                 :rem 251
5120 FOR S3=12416 TO 12478:READQ3:POKES3,Q3:NEXT
                                                  :rem 11
5130 FOR S4=12480 TO 12542:READQ4:POKES4,Q4:NEXT
                                                   :rem 9
5140 FOR S5=12544 TO 12606:READQ5:POKES5,Q5:NEXT
                                                  :rem 16
5200 P1=192                                       :rem 33
5300 LX=24:LY=204:REM CAR START LOCATION         :rem 156
5310 GOSUB 20000                                  :rem 59
5320 POKEV+39,1:POKE 2040,P1                      :rem 31
5400 POKEV,LX:POKEV+1,LY                          :rem 10
5500 REM FREQUENCY DATA                          :rem 122
5510 DATA 2,204,3,134,4,112,5,152,7,12            :rem 51
14900 REM CAR U/D                                :rem 128
15000 DATA 0,0,0,0,0,0,0,0,0                      :rem 32
15010 DATA 0,0,0,0,0,0,0,0,0                      :rem 33
15020 DATA 0,0,0,0,0,0,15,207,192                 :rem 45
15030 DATA 3,3,0,15,255,192,3,3,0                 :rem 61
15040 DATA 15,207,192,0,0,0,0,0,0                 :rem 47
15050 DATA 0,0,0,0,0,0,0,0,0                      :rem 37
```

```
15060 DATA 0,0,0,0,0,0,0,0,0                    :rem 38
15065 REM CAR R/L                               :rem 136
15070 DATA 0,0,0,0,0,0,0,0,0                    :rem 39
15080 DATA 0,0,0,0,0,0,0,0,0                    :rem 40
15090 DATA 3,51,0,3,255,0,3,51,0                :rem 10
15100 DATA 0,48,0,0,48,0,0,48,0                 :rem 213
15110 DATA 3,51,0,3,255,0,3,51,0                  :rem 3
15120 DATA 0,0,0,0,0,0,0,0,0                    :rem 35
15130 DATA 0,0,0,0,0,0,0,0,0                    :rem 36
15140 REM EXPLOSION SPRITE DATA                 :rem 145
15160 DATA 0,0,0,0,0,0,0,0,0                    :rem 39
15170 DATA 0,0,0,0,0,0,0,0,0                    :rem 40
15180 DATA 12,0,0,3,3,0,0,195,0                 :rem 209
15190 DATA 0,63,48,3,255,192,0,63,48            :rem 239
15200 DATA 0,63,0,0,204,192,0,12,48             :rem 156
15210 DATA 0,0,0,0,0,0,0,0,0                    :rem 35
15220 DATA 0,0,0,0,0,0,0,0,0                    :rem 36
15230 REM                                       :rem 223
15240 DATA 0,0,0,0,0,0,0,204,12                 :rem 191
15250 DATA 48,48,12,48,240,48,12,48,192         :rem 139
15260 DATA 12,48,192,3,63,207,48,255,204        :rem 182
15270 DATA 12,255,240,3,255,192,0,255,240       :rem 219
15280 DATA 0,255,204,15,255,243,48,63,48        :rem 184
15290 DATA 0,204,48,3,207,12,3,3,15             :rem 168
15300 DATA 3,0,195,12,0,192,0,0,192             :rem 160
15310 REM                                       :rem 222
15320 DATA 12,0,207,3,3,15,195,207,63            :rem 14
15330 DATA 48,51,252,48,255,252,204,63,240      :rem 23
15340 DATA 12,255,192,195,255,207,51,255,204
                                                :rem 126
15350 DATA 15,255,252,243,255,195,51,255,240
                                                :rem 127
15355 DATA 51,255,204,15,255,243,51,255,240:rem 72
15360 DATA 243,204,48,207,207,12,15,3,15        :rem 168
15370 DATA 63,12,195,60,204,192,240,48,192      :rem 27
15400 :                                         :rem 52
15410 REM KEYBOARD CHECK FOR DIRECTION,         :rem 66
15420 REM SPEED,SOUND                            :rem 6
15430 :                                         :rem 55
15440 SP=1:FC=1:POKEVL,15:POKEA,136:POKESR,136
                                                :rem 33
15500 GET C1$:IFC1$="" THEN C1$=C3$             :rem 243
15520 IF C1$="P" THEN LY=LY-SP:C3$=C1$:POKE2040,P1
      +1                                        :rem 92
15540 IF C1$="." THEN LY=LY+SP:C3$=C1$:POKE2040,P1
      +1                                        :rem 58
15560 IF C1$="L" THEN LX=LX-SP:C3$=C1$:POKE2040,P1
                                                :rem 254
```

```
15580 IF C1$=";" THEN LX=LX+SP:C3$=C1$:POKE2040,P1
                                                :rem 237
15600 IF C1$="1" THEN SP=SP-1:C1$=C3$:FC=FC-1:IFFC
      <1 THEN FC=1                               :rem 90
15620 IF C1$="2" THEN SP=SP+1:C1$=C3$:FC=FC+1:IFFC
      >5 THEN FC=5                               :rem 99
15630 TS=TS+SP                                   :rem 88
15700 POKEW,33:POKEH,F1(FC):POKEL,F2(FC)        :rem 118
15800 GOSUB 17000                               :rem 118
16000 POKEV,LX:POKEV+1,LY                        :rem 56
16100 GOTO 15500                                 :rem 44
16900 :                                          :rem 58
16910 REM CHECK STATUS OF CAR                   :rem 146
16920 REM UPDATE TIME,SPEED,SCORE                :rem 29
16930 REM UPDATE SECTION OF TRACK               :rem 201
16940 :                                          :rem 62
17000 IF PEEK(V+31)AND1=1THEN GOSUB18000        :rem 252
17050 IF LY<100 AND LX>190 AND LP=0 THEN LP=1:SB=1
      :LC=LC+1                                  :rem 144
17060 IF LY>180 THEN LP=0                        :rem 65
17100 IF LX<6 THEN GOSUB 19000                   :rem 71
17200 RETURN                                    :rem 218
17900 :                                          :rem 59
17910 REM EXPLOSION SEQUENCE                      :rem 0
17920 :                                          :rem 61
18000 POKEV+39,14:POKE2040,P1+2                 :rem 223
18010 FOR I=1 TO 150:NEXT                        :rem 74
18015 POKEW,32:POKEW,129                        :rem 158
18020 POKEV+39,7:POKE2040,P1+3                  :rem 180
18030 FOR I=1 TO 200:NEXT                        :rem 72
18040 POKEV+39,2:POKE2040,P1+4                  :rem 178
18050 FOR I=1 TO 300:NEXT                        :rem 75
18060 POKE2040,P1+3                             :rem 255
18070 FORI=1 TO 200:NEXT                         :rem 76
18080 POKE2040,P1+2                               :rem 0
18090 FORI=1 TO 150:NEXT                         :rem 82
18100 LX=25:LY=204:SP=1:C1$="":C3$="":SB=0:LP=0:LC
      =0:TS=0                                   :rem 234
18110 POKEV+39,1:POKE2040,P1:POKEV,LX:POKEV+1,LY
                                                :rem 203
18120 POKEV+31,0:FC=1                           :rem 157
18140 RETURN                                    :rem 222
18900 :                                          :rem 60
18910 REM DISPLAY STATS ON THE RACE              :rem 37
18920 :                                          :rem 62
19000 PRINT"{HOME}":POKEW,32:IF SB=0 THEN TP=0:GOT
      O19020                                    :rem 169
19010 TP=INT((TS*(100/TI))*LC)                  :rem 161
19020 PRINT TAB(28)"TOTAL PTS"                  :rem 218
```

```
19025 PRINT TAB(28) "{7 SPACES}{UP}"          :rem 245
19030 PRINT TAB(28) TP                        :rem 192
19040 PRINT"{2 DOWN}" TAB(28) "TIME"          :rem 246
19050 PRINT TAB(29) TI$                       :rem 224
19060 PRINT"{2 DOWN}" TAB(28) "LAPS"          :rem 249
19065 PRINTTAB(28) "{6 SPACES}{UP}"           :rem 249
19070 PRINTTAB(28) LC                         :rem 175
19075 PRINT"{2 DOWN}" TAB(28) "TIME/LAP"       :rem 10
19077 IF LC=0 THEN LC=1                       :rem 191
19080 PRINT TAB(28) "{6 SPACES}{UP}"          :rem 246
19085 PRINT TAB(28) INT(TI/60)/LC              :rem 82
19090 LX=25:LY=204:SP=1:C3$="":C1$="":FC=1:SB=0
                                              :rem 93
19100 PRINT"{3 DOWN}" TAB(28)"PRESS {RVS}SPACE
      {OFF}"                                  :rem 114
19110 PRINTTAB(33) "TO"                       :rem 254
19115 PRINT TAB(28) "START CLOCK"              :rem 94
19120 GET KH$:IF KH$="" THEN 19120            :rem 193
19130 PRINT"{3 UP}" TAB(28) "{11 SPACES}"      :rem 88
19133 PRINT TAB(33) "{2 SPACES}"               :rem 96
19135 PRINT TAB(28) "{11 SPACES}"             :rem 102
19200 TI$="000000":TP=0:LP=0:LC=0:TS=0        :rem 112
19800 RETURN                                  :rem 226
19900 :                                        :rem 61
19910 REM TRACK ONE                            :rem 63
19920 :                                        :rem 63
20000 PRINT"{2 +}UCCCCCCCCI{6 +}UCCCCCCCCI{2 +}
                                               :rem 3
20010 PRINT"{2 +}B{7 SPACES}B{6 +}B{7 SPACES}B
      {2 +}                                   :rem 38
20020 PRINT"{2 +}B{7 SPACES}B{6 +}B{7 SPACES}B
      {2 +}                                   :rem 39
20030 PRINT"{2 +}B{2 SPACES}UCI{2 SPACES}B{6 +}B
      {2 SPACES}UCI{2 SPACES}B{2 +}           :rem 234
20040 PRINT"{2 +}B{2 SPACES}B{+}B{2 SPACES}B{6 +}B
      {2 SPACES}B{+}B{2 SPACES}B{2 +}         :rem 125
20050 PRINT"{2 +}B{2 SPACES}B{+}B{2 SPACES}JCCCCCC
      K{2 SPACES}B{+}B{2 SPACES}B{2 +}        :rem 61
20060 PRINT"{2 +}B{2 SPACES}B{+}B{12 SPACES}B{+}B
      {2 SPACES}B{2 +}                        :rem 23
20070 PRINT"{2 +}B{2 SPACES}B{+}B{12 SPACES}B{+}B
      {2 SPACES}B{2 +}                        :rem 24
20080 PRINT"{2 +}B{2 SPACES}B{+}JCCCCCCCCCCCCCK{+}B
      {2 SPACES}B{2 +}                        :rem 78
20090 PRINT"{2 +}B{2 SPACES}B{+}UCCCCCCCCCCCCCCCK
      {2 SPACES}B{2 +}                        :rem 120
20100 PRINT"{2 +}B{2 SPACES}B{+}B{17 SPACES}B{2 +}
                                              :rem 232
```

```
20110 PRINT"{2 +}B{2 SPACES}B{+}B{17 SPACES}B{2 +}
                                              :rem 233
20120 PRINT"{2 +}B{2 SPACES}B{+}B{3 SPACES}UCCCCCC
      CCCCCCCK{2 +}                           :rem 175
20130 PRINT"{2 +}B{2 SPACES}B{+}B{3 SPACES}JCCCCCC
      CCCCCCCI{2 +}                           :rem 163
20140 PRINT"{2 +}B{2 SPACES}B{+}B{17 SPACES}B{2 +}
                                              :rem 236
20150 PRINT"{2 +}B{2 SPACES}B{+}B{17 SPACES}B{2 +}
                                              :rem 237
20160 PRINT"{2 +}B{2 SPACES}B{+}JCCCCCCCCCCCCCCCI
      {2 SPACES}B{2 +}                        :rem 105
20170 PRINT"{2 +}B{2 SPACES}B{16 +}B{2 SPACES}B
      {2 +}                                   :rem 169
20180 PRINT"{2 +}B{2 SPACES}B{16 +}B{2 SPACES}B
      {2 +}                                   :rem 170
20190 PRINT"CCK{2 SPACES}JI{15 +}B{2 SPACES}B{2 +}
                                               :rem 25
20200 PRINT"{6 SPACES}JI{14 +}B{2 SPACES}B{2 +}
                                               :rem 26
20210 PRINT"CCCI{3 SPACES}JCCCCCCCCCCCCCCK
      {2 SPACES}B{2 +}                          :rem 3
20220 PRINT"{3 +}JI{20 SPACES}B{2 +}           :rem 56
20230 PRINT"{4 +}JCCCCCCCCCCCCCCCCCCCCCK{2 +}
                                               :rem 91
20240 PRINT"{28 +}{UP}"                       :rem 130
20470 RETURN                                  :rem 221
```

Haunted Mansion

Calvin Overhulser

64 Translation by Kevin Martin

Rescuing cats from a witch's clutches is only part of this colorful and imaginative game. Side-stepping ghosts and bats and evading evil spirits complete the action in this joystick-controlled adventure.

An evil witch has captured some friendly neighborhood cats and taken them to her haunted mansion, where she plans to turn them into "witch cats." The witch's mansion is a maze of corridors, and she has hidden the cats in nooks and crannies throughout the house. Even scarier, she has ghosts, bats, and evil spirits occupying the mansion. Your goal is to rescue the cats while avoiding the witch's minions.

Don't Drop That Cat!

Once you've typed in and saved "Haunted Mansion" (make sure you read and use "The Automatic Proofreader" found in Appendix C to enter an error-free version), load and run it. There's a short delay while the custom characters are being created. Instructions then display.

After selecting one of six skill levels, from easy (1) to difficult (6), use your joystick (plugged into port 2) to maneuver through the maze. When you reach one of the cats, return with it to the bottom row of the maze, the only safe spot for felines in this game. If you run into a ghost or bat on your return, you'll drop the cat and lose points. The frightened cat will then jump to another random location in the maze. The ghosts and bats aren't deadly. They're just as scared of you as you are of them. Bumping into one eliminates it, but you lose points. At the higher skill levels you'll have to sacrifice points by deliberately running into the ghosts or bats to clear a path so that you can get to a cat.

You'll notice that the cat catcher changes color from time to time. When he's searching for a cat, he's blue. After bumping into a ghost or bat, however, he momentarily turns white (scared himself!). When he's got a cat under his arm, he changes to yellow to let you know he needs to get rid of it.

And if a spirit manages to grab him, he's purple. There are also sound effects that give you information, ranging from the the background sound, which begins once the maze is drawn, to the distressing beeps, which occur when an evil spirit touches your character.

Your most dangerous enemies are the moving evil spirits, which look like disembodied faces. If you run into one of them, there's no second chance—the game ends, and your final score, skill level, and round are shown. You'll be given the option to play another game and choose a skill level. The higher skill levels award more points but are more difficult.

Building the Mansion

I've included several REMs for the major subroutines to show how the program is logically constructed. The main loop is in lines 210–240. Lines 300–307 are used to update the position of the cat saver (that's you) and the selected evil spirit. The ON-GOSUB in line 215 for the cat saver and line 630 for the randomly chosen evil spirit allows the new location for either to be calculated using the same subroutines.

The joystick is read with a standard routine in lines 210–214. Location 56320 (for a joystick in port 2) is PEEKed and the value ANDed with 15. Depending on the result, one of several lines is called by line 215.

If you've already looked at the game, you've probably noticed the custom characters. The first 64 standard characters are copied by line 110 from ROM (Read Only Memory) into RAM (Random Access Memory). Lines 130–150 then READ and POKE the DATA statements (lines 10000–10190) for the custom characters into RAM locations. Line 100 lowers the top of BASIC memory to protect the custom character set from being erased. Table 1 lists the custom characters and their screen POKE codes.

Note that there is a custom character "space" (screen code 62) in addition to the normal space (screen code 32). This allows the same character to be displayed on the screen with both codes, but lets the program tell the difference. The normal space is used inside the mansion, and the custom character space is used outside. This keeps the bats, ghosts, and evil spirits from appearing in the sky, since they can be placed only in a location containing a normal space.

Table 1. Custom Characters

Screen Code	Original Character	Custom Character
27	[	Witch
28	£	Witch
29	]	Witch
30	↑	Witch
31	←	Evil spirit
35	#	Solid block
36	$	Moon
37	%	Moon
38	&	Moon
39	'	Moon
40	(	Moon
41	(	Moon
42	*	Moon
43	+	Moon
44	,	Roof
45	-	Roof
58	:	Cat saver
59	;	Ghost
60	<	Cat
61	=	Bat
62	>	Space (outside house)

The game screen is built in lines 1000–1095, and the maze is generated in lines 1200–1292. (Note that the algorithm used to create the maze is Charles Bond's excellent BASIC version. Refer to *COMPUTE!'s First Book of Commodore 64 Games* for the article that explains the algorithm. It also includes a machine language version, which draws the maze at incredible speed.)

Variable names, listed in Table 2, are used more than once where possible to conserve memory. This isn't crucial when you're programming on the 64, but it is a good practice to get into, especially if you program occasionally on computers with smaller amounts of RAM.

Table 2. Program Variable Names

Variable	Description
A	Variable in READ statements.
I	Miscellaneous counters in FOR/NEXT loops + random numbers.
J	Random number.
N	Counter in FOR/NEXT loops.
O	Constant=0.
P	Constant=1.
Q	Constant=40.
V	Volume (54296).
X	Counter in FOR/NEXT loops + random numbers.
Z	Current location to be updated in subroutines 300–307.
AA	Skill level.
A$	String for GET statements.
BL	Flag to place character.
CC	Cat counter.
CF	Cat flag CF=4 means carrying cat.
CL	Current location for cat saver.
CM	Difference between color memory and screen memory.
DF	Dead flag.
HL	Constant=32.
JP	Joystick value.
RN	Number of current round.
SC	Screen RAM location.
SR	Current score.
SH	Sound High (54273).
SL	Sound Low (54272).
TL	Temporary storage for CL or A(I) during update.
WL	Constant=35.
A(0)–A(3)	Variables for maze generator.
A(1)–A(13)	Locations of evil spirits.

Haunted Mansion

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
100 POKE52,48:POKE56,48:CLR                          :rem 70
102 POKE53280,0:POKE53281,0                          :rem 232
105 PRINT"{CLR}{7}{3 DOWN}***********{RVS}HAUNTED
    {2 SPACES}MANSION{OFF}***********";               :rem 180
107 PRINT"{13 DOWN}{9 SPACES}REDEFINING{2 SPACES}C
    HARACTERS"                                        :rem 1
108 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND25
    1                                                 :rem 186
110 FORI=0TO511:POKE12288+I,PEEK(53248+I):NEXT
                                                      :rem 224
115 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
                                                      :rem 134
130 FORI=12288+35*8TO12288+45*8+7:READA:POKEI,A:NE
    XT                                                :rem 42
140 FORI=12288+27*8TO12288+31*8+7:READA:POKEI,A:NE
    XT                                                :rem 39
150 FORI=12288+58*8TO12288+62*8+7:READA:POKEI,A:NE
    XT                                                :rem 48
152 SC=1024                                           :rem 50
153 CM=54272                                          :rem 106
155 SH=54273:SL=54272:V=54296:WF=54276:O=0:P=1:Q=4
    0:DIMA(13):POKEV,15                               :rem 250
157 POKESL+5,17:POKESL+6,241                          :rem 139
160 GOSUB800                                          :rem 175
165 GOSUB900                                          :rem 181
170 POKE53272,(PEEK(53272)AND240)+12                  :rem 183
175 GOSUB1000:RN=RN+1                                 :rem 241
200 CL=SC+859                                         :rem 197
210 JP=15-PEEK(56320)AND15:IFJP=8THENJP=3:GOTO215
                                                      :rem 200
211 IFJP=2THENJP=5:GOTO215                            :rem 114
212 IFJP=4THENJP=7:GOTO215                            :rem 119
213 IFJP=1THEN215                                     :rem 244
214 JP=2                                              :rem 160
215 TL=CL:Z=TL:ONJPGOSUB301,300,303,300,305,300,30
    7,300                                             :rem 66
220 CL=Z:POKEV,15:POKESH,50:POKEWF,17:GOSUB400:POK
    EWF,16                                            :rem 33
225 IFDFTHEN500                                       :rem 118
230 POKETL,32:POKECL,58:POKECM+CL,3+CF                :rem 76
232 GOSUB700:IFCC=1THENFORX=1TO1500:NEXT:GOTO170
                                                      :rem 52
235 GOSUB600:IFDFTHEN500                              :rem 199
240 GOTO210                                           :rem 98
300 RETURN                                            :rem 115
301 Z=Z-Q:RETURN                                      :rem 29
```

```
303 Z=Z+P:RETURN                                    :rem 28
305 Z=Z+Q:RETURN                                    :rem 31
307 Z=Z-P:RETURN                                    :rem 34
400 REM COLLISION CHECK                            :rem 130
405 IFPEEK(CL)=44ORPEEK(CL)=45ORPEEK(CL)=35THENCL=
    TL:RETURN                                       :rem 30
407 IFCFANDCL<SC+873ANDCL>SC+845THEN2000   :rem 167
410 IFCFANDPEEK(CL)=60THENCL=TL:RETURN     :rem 221
415 IFPEEK(CL)=60THENCF=4:POKEWF,33:FORI=1TO100:NE
    XT:RETURN                                       :rem 49
420 IFPEEK(CL)=61ORPEEK(CL)=59THEN1800      :rem 23
425 IFPEEK(CL)=31THEN1900                          :rem 201
430 RETURN                                         :rem 119
500 REM GOTCHA!                                     :rem 80
510 PRINT"{CLR}{73}{DOWN}{12 RIGHT}ANOTHER VICTIM!"
                                                   :rem 255
520 POKE53272,21                                    :rem 88
525 PRINT"{DOWN}{13 RIGHT}SKILL LEVEL"AA   :rem 112
530 PRINT"{DOWN}{11 RIGHT}ROUND"RN"SCORE"SR:rem 70
540 PRINT"{DOWN}{10 RIGHT}PLAY AGAIN?{2 SPACES}
    {RVS}Y{OFF} OR {RVS}N{OFF}"                      :rem 2
550 GETA$:IFA$=""THEN550                            :rem 87
560 IFA$="Y"THENCF=0:GOTO570                       :rem 157
562 IFA$<>"N"THEN550                               :rem 102
565 SYS2048                                        :rem 109
570 RN=0:SR=0:DF=0                                  :rem 38
575 GOTO165                                        :rem 118
600 REM MOVE SPIRITS                               :rem 223
610 I=INT(RND(1)*(AA*2))+1                         :rem 116
620 TL=A(I):Z=TL:POKEA(I),32                       :rem 150
630 ONINT(RND(1)*4)+1GOSUB301,303,305,307 :rem 242
635 IFZ>SC+845ANDZ<SC+873THEN660           :rem 190
640 IFPEEK(Z)=58THEN1900                           :rem 156
650 IFPEEK(Z)=32THENA(I)=Z                          :rem 61
660 POKEA(I),31:POKECM+A(I),4:RETURN       :rem 175
700 PRINT"{HOME}{23 DOWN}{10 RIGHT}{WHT}ROUND"RN"S
    CORE"SR"{LEFT} ";:RETURN                   :rem 236
800 PRINT"{CLR}YOU WILL ENTER A WITCH'S HAUNTED HO
    USE. ";                                    :rem 223
815 PRINT"{DOWN}THE WITCH IS AWAY, FLYING ON HER B
    ROOM. ";                                        :rem 7
820 PRINT"{DOWN}SHE HAS CAPTURED YELLOW CATS AND W
    ILL{3 SPACES}";                            :rem 221
825 PRINT"{DOWN}TURN THEM INTO WITCH CATS UNLESS Y
    OU{4 SPACES}{DOWN}RESCUE THEM."            :rem 231
830 PRINT"{DOWN}GUIDE YOURSELF WITH A JOYSTICK. PI
    CK UP ";                                   :rem 134
832 PRINT"{DOWN}ONE CAT AT A TIME. BRING IT TO THE
    {6 SPACES}";                                :rem 58
```

```
835 PRINT"{DOWN}BOTTOM ROW.{2 SPACES}WHILE CARRYIN
    G A CAT, YOU{2 SPACES}";                    :rem 210
837 PRINT"{DOWN}WILL TURN YELLOW.{2 SPACES}YOU CAN
     PICK UP ONLY ";                            :rem 39
840 PRINT"{DOWN}ONE CAT AT A TIME, AND YOU GET POI
    NTS{3 SPACES}";                             :rem 49
845 PRINT"{DOWN}FOR EACH CAT YOU SAVE. WHEN YOU SA
    VE 10 {DOWN}CATS, YOU GET A NEW HOUSE."
                                                :rem 142
847 PRINT: PRINT"{13 SPACES}PRESS ANY KEY";
                                                :rem 212
850 GETA$:IFA$=""THEN850                        :rem 93
855 PRINT"{CLR}IF YOU RUN INTO A BAT OR GHOST, YOU
    'LL{2 SPACES}";                             :rem 34
860 PRINT"{DOWN}JUMP IN FEAR, DROP ANY CAT YOU ARE
    {6 SPACES}{DOWN}CARRYING AND LOSE POINTS."
                                                :rem 232
865 PRINT"{DOWN}THE HOUSE IS ALSO HAUNTED BY EVIL
    {7 SPACES}";                                :rem 196
866 PRINT"{DOWN}SPIRITS FLOATING ALONG THE HALLWAY
    S."                                         :rem 237
867 PRINT"{DOWN}IF YOU TOUCH A SPIRIT, YOU LOSE TH
    E{5 SPACES}{DOWN}GAME!"                     :rem 59
868 PRINT"{DOWN}(BOTTOM ROW IS SAFE) "          :rem 97
870 PRINT"{YEL}{DOWN}HIGHER SKILL LEVELS SCORE MOR
    E POINTS{3 SPACES}{DOWN}BUT ARE HARDER.{OFF}"
                                                :rem 246
890 RETURN                                      :rem 129
900 PRINT"{6}{DOWN}ENTER SKILL LEVEL 1-6"       :rem 29
920 GETA$:IFA$=""THEN920                        :rem 89
930 AA=VAL(A$):IFAA<1ORAA>6THEN920              :rem 13
940 RETURN                                      :rem 125
1000 POKE53280,0:POKE53281,0:PRINT"{CLR}"       :rem 180
1003 PRINT">{YEL}>.>.>>>>>>>>>>>>>>{BLU}-,{YEL}>.>
     >>>>>>>>.>>>>>>>";                         :rem 120
1005 PRINT">>>>>>>.>>.>>>>>>>{BLU}-##,{YEL}>.>>$%>
     .>>>>.>>>>>";                              :rem 99
1010 PRINT">>>>.>>>>>>>>>>.>{BLU}-####,{YEL}>>&[£
     '>>.>>>>>>>>";                             :rem 136
1015 PRINT">>>>>>>.>>>>>>>>{BLU}-######,{YEL}>(]↑)
     >.>>>.>>>>>";                              :rem 95
1020 PRINT">>>>>>>>>>.>>>>{BLU}-########,{YEL}>*+>
     .>>>>>>>>>>";                              :rem 250
1025 PRINT">>>>>.>>>>>>>>{BLU}-##########,{YEL}>>>
     .>>>>>>>.>>";                              :rem 224
1030 PRINT">>>>>>>>>>>>>{BLU}-###########,{YEL}>>
     >.>>>>>>>>>";                              :rem 198
1035 PRINT">>>.>>>>>.>>{BLU}-############,{YEL}>
     .>>>>>>.>>>";                              :rem 101
```

```
1040 PRINT">>.>>>>>>>>{BLU}-################,{YEL}
     >>>.>>>>>>>";                              :rem 75
1045 PRINT">>>>>>>>>>{BLU}-##################,
     {WHT}>>>>>>>>>>";                          :rem 161
1050 PRINT">>>>>>>>>{BLU}-####################,
     {WHT}>>>>>>>>>";                           :rem 103
1055 PRINT">>>>>>>>{BLU}-######################,
     {WHT}>>>>>>>>";                            :rem 54
1060 PRINT">>>>>>>{BLU}-########################,
     {WHT}>>>>>>>";                             :rem 252
1065 PRINT">>>>>>{BLU}-##########################,
     {WHT}>>>>>>";                              :rem 203
1075 PRINT">>>>>{BLU}-############################
     {WHT}>>>>>>";                              :rem 168
1080 PRINT">>>>>{BLU}############################
     {WHT}>>>>>>";                              :rem 154
1085 PRINT">>>>>{BLU}############################
     {WHT}>>>>>>";                              :rem 159
1090 PRINT">>>>>{BLU}############################
     {WHT}>>>>>>";                              :rem 155
1092 PRINT">>>>>{BLU}############################
     {WHT}>>>>>>";                              :rem 157
1093 PRINT">>>>>{BLU}############################
     {WHT}>>>>>>";                              :rem 158
1094 PRINT">>>>>{BLU}############################
     {WHT}>>>>>>";                              :rem 159
1095 PRINT">>>>>{BLU}############################
     {WHT}>>>>>>";                              :rem 160
1200 REM BUILD MAZE                              :rem 68
1205 A(0)=2:A(1)=-80:A(2)=-2:A(3)=80:WL=35:HL=32:A
     =SC+846                                     :rem 19
1210 POKEA,4                                     :rem 148
1220 J=INT(RND(1)*4):X=J                         :rem 101
1230 B=A+A(J):IFPEEK(B)=WLTHENPOKEB,J:POKEA+A(J)/2
     ,HL:A=B:GOTO1220                             :rem 8
1240 J=(J+1)*-(J<3):IFJ<>XTHEN1230               :rem 128
1250 J=PEEK(A):POKEA,HL:IFJ<4THENA=A-A(J):GOTO1220
                                                 :rem 34
1260 FORI=SC+216TOSC+856STEP40:POKEI,32:POKEI+7,32
     :NEXT                                       :rem 152
1270 FORI=SC+140TOSC+860STEP40:POKEI,32:NEXT
                                                 :rem 235
1280 FORI=SC+372TOSC+852STEP40:POKEI,32:POKEI+15,3
     2:NEXT                                      :rem 200
1290 FORI=SC+489TOSC+849STEP40:POKEI,32:POKEI+21,3
     2:NEXT                                      :rem 213
1291 FORI=SC+372TOSC+387:POKEI,32:NEXT           :rem 89
1292 FORI=SC+489TOSC+510:POKEI,32:NEXT           :rem 87
1293 FORI=SC+687TOSC+712:POKEI,32:POKEI+160,32:NEX
     T                                           :rem 97
```

```
1300 REM PLACE GAME CHARACTERS                    :rem 7
1310 REMPLACE GHOSTS                             :rem 230
1320 FORI=PTO3*AA                                 :rem 5
1330 X=INT(RND(1)*680)+SC+40                     :rem 22
1340 BL=0:GOSUB1700:IFBLTHEN1330                 :rem 146
1350 POKEX,59:POKECM+X,1:NEXT                    :rem 60
1400 REMPLACE BATS                               :rem 56
1420 FORI=PTO3*AA                                 :rem 6
1430 X=INT(RND(1)*680)+SC+40                     :rem 23
1440 BL=0:GOSUB1700:IFBLTHEN1430                 :rem 148
1450 POKEX,61:POKECM+X,5:NEXT                    :rem 58
1500 REMPLACE CATS                               :rem 58
1520 CC=11:FORI=PTO10                            :rem 232
1530 X=INT(RND(1)*680)+SC+40                     :rem 24
1540 BL=0:GOSUB1700:IFBLTHEN1530                 :rem 150
1550 POKEX,60:POKECM+X,7:NEXT                    :rem 60
1600 REMPLACE SPIRITS                            :rem 62
1620 FORI=PTOAA*2                                 :rem 7
1630 X=INT(RND(1)*680)+SC+40                     :rem 25
1640 IFPEEK(X)<>32THEN1630                        :rem 0
1650 POKEX,31:POKECM+X,4:A(I)=X:NEXT:RETURN
                                                 :rem 252
1700 IF(PEEK(X)<>32)OR(PEEK(X+P)<>32ANDPEEK(X+P)<>
     35)THENBL=1                                 :rem 152
1710 IF(PEEK(X-P)<>32ANDPEEK(X-P)<>35)OR(PEEK(X+Q)
     <>32ANDPEEK(X+Q)<>35)THENBL=1                :rem 24
1720 IF(PEEK(X-Q)<>32ANDPEEK(X-Q)<>35)THENBL=1
                                                  :rem 1
1730 RETURN                                      :rem 171
1800 REM SCARED!                                 :rem 128
1810 IFCF=OTHEN1840                              :rem 106
1820 X=INT(RND(1)*680)+SC+40                      :rem 26
1830 IFPEEK(X)<>32THEN1820                        :rem 2
1835 POKEX,60:POKECM+X,7:CF=0:SR=SR-2*AA↑2:IFSR<OT
     HENSR=0                                     :rem 205
1840 POKETL,32:POKECL,58:POKECL+CM,1             :rem 206
1843 FORI=15TO1STEP-1:POKESL,100:POKESH,100:POKEWF
     ,33:POKEV,I                                  :rem 99
1844 FORII=1TO25:NEXTII,I                         :rem 66
1845 FORI=1TO400:NEXT:POKEWF,32                  :rem 183
1850 TL=CL:Z=TL:ONINT(RND(1)*4)+1GOSUB301,303,305,
     307                                          :rem 62
1860 CL=Z:SR=SR-AA↑2:IFSR<OTHENSR=0               :rem 78
1870 GOTO400                                     :rem 157
1900 REM GOTCHA! SOUND                            :rem 14
1905 POKETL,32:POKECL,58:POKECM+CL,4:DF=1          :rem 5
1910 POKESL,100:FORI=1TO4:POKEWF,17:POKESH,25:FORX
     =1TO150:NEXT:POKEWF,16                      :rem 104
1911 FORX=1TO75:NEXT                              :rem 1
```

```
1913 IFI=1ORI=3THENPOKECL,31:GOTO1920       :rem 42
1914 POKECL,58                              :rem 38
1920 POKESL,0:FORX=1TO200:NEXT:NEXT:FORX=1TO500:NE
     XT:RETURN                             :rem 182
2000 REM LINE UP SAVED CAT                 :rem 190
2010 X=SC+985:POKEX-CC,60:POKECM+X-CC,7:CC=CC-1:SR
     =SR+10*AA↑2:CF=O                       :rem 16
2020 POKEWF,33:FORI=1TO50:NEXT:RETURN      :rem 149
10000 DATA255,255,255,255,255,255,255,255  :rem 31
10010 DATA0,0,0,0,0,15,63,255              :rem 155
10015 DATA0,0,0,0,0,240,252,255              :rem 0
10020 DATA1,1,3,3,7,7,7,7                  :rem 229
10030 DATA128,128,192,192,224,224,224,224  :rem 16
10040 DATA7,7,7,7,3,3,1,1                  :rem 231
10050 DATA224,224,224,224,192,192,128,128  :rem 18
10060 DATA255,63,15,0,0,0,0,0              :rem 160
10070 DATA255,252,240,0,0,0,0,0              :rem 1
10080 DATA128,192,224,240,248,252,254,255  :rem 26
10090 DATA1,3,7,15,31,63,127,255           :rem 76
10100 DATA255,255,255,255,250,246,244,224  :rem 21
10110 DATA255,255,191,63,15,15,7,63        :rem 233
10120 DATA240,249,240,228,0,252,255,255    :rem 170
10130 DATA255,255,127,35,1,112,63,255       :rem 71
10140 DATA0,102,0,24,0,0,28,0              :rem 151
10150 DATA56,84,56,16,124,186,40,108        :rem 33
10160 DATA62,42,62,28,28,28,60,120         :rem 179
10170 DATA40,124,85,125,57,57,61,127        :rem 32
10180 DATA0,16,124,254,214,130,0,0         :rem 156
10190 DATA0,0,0,0,0,0,0,0                  :rem 201
20000 JP=15-PEEK(56320)AND15                :rem 11
20010 PRINTJP:GOTO20000                    :rem 127
```

6
Machine Language Games

CUT-OFF!

Tom R. Halfhill

"CUT-OFF!" is a fast-paced, two-player game programmed entirely in machine language. With ten levels of difficulty—ranging in speed from moderately slow to impossibly fast—the game requires two joysticks.

Over the years, some computer games have become classics. Usually they are simple in concept, universal in appeal, yet general enough to be translatable for almost any computer. Some examples are *Pong,* the granddaddy of all videogames, *Breakout, Lunar Lander,* and the venerable *Space Invaders.* For legal reasons they may be disguised by different names, but there probably isn't a computer or videogame machine anywhere for which some version of these all-time favorites isn't available.

Another classic game is *Blockade.* Again, it goes by different names (sometimes *Surround*), but the basic concept remains the same. Two players steer a moving line around the screen, trying to head off each other, or force the opposing player to crash into a wall, or back up over an existing trail. This concept dates back to the early days of videogames. In fact, the very first videogame I ever played was a *Blockade*-style game. It was during the mid-1970s, and a friend and I encountered the machine in a dimly lit cafe. By today's standards the game was downright primitive. No color, crude sound effects, and slow action. Yet we had never played anything like it before. (We thought it would never catch on, because it cost 25 cents per play at a time when a quarter bought you three plays on most pinball machines.)

Years later, the basic concept of *Blockade* was revived and updated in the 1982 film *TRON.* In this Walt Disney production, humans trapped inside a bizarre computer world were forced to become gladiators on "light cycles"—space-age motorcycles that left walls in their wakes, counterparts of the lengthening trails in *Blockade.*

That's the story behind the latest incarnation of this popular game, now dubbed "CUT-OFF!". It preserves all the traditional concepts and includes color, sound, and the broad

range of speed levels possible only in a program written entirely in machine language.

Typing CUT-OFF!

Pure machine language programs are usually more difficult to enter than BASIC programs, because they consist of seemingly endless streams of numbers. To make typing CUT-OFF! easier, we've listed the programs in MLX format.

You may already be familiar with MLX if you've typed in some of the machine language programs published in other COMPUTE! books, or in COMPUTE! publications such as *COMPUTE!* magazine or *COMPUTE!'s Gazette* magazine. If you're not familiar with MLX, it's a utility designed by Charles Brannon to make typing errors almost impossible. To learn how to use MLX, see "Using the Machine Language Editor: MLX" in Appendix D at the back of this book. If you've previously typed in MLX, you can use it again for CUT-OFF!

Here's the information you'll need to enter CUT-OFF!:

Starting Address—49152 3
Ending address—50663 3
To run, enter SYS 49152 3
To stop, press RUN/STOP-RESTORE 3

Remember, to load a machine language program from disk or tape, you must use this special form of the LOAD command:

LOAD"*filename***",8,1 (for disk)**
LOAD"*filename***",1,1 (for tape)**

If you forget to append the *,1* to the command, the program loads into the wrong area of memory and will not work.

Youngsters to Superhumans

After you enter the SYS 49152 command, the game screen appears instantly. (One of the best things about machine language is that you don't have to wait around for programs to initialize.)

The opening screen allows you to select a skill level ranging from 0 (the slowest speed, suitable for youngsters) to 9 (recommended for superhumans only). The skill levels are spaced equally apart, so you might want to start at 3 or 4. The level you select remains the same for the entire game. To change levels in the middle of a game, press RUN/STOP–

RESTORE and restart the program with the SYS command. (Of course, this cancels the game in progress.)

To choose a skill level, move the joystick plugged into port 1 up or down. You'll see the number on the screen change and wrap around if you go below 0 or above 9. To lock in your choice and begin the game, press the fire button of joystick 1.

The game starts with the players ready to crash head-on. Joysticks 1 and 2 control the left and right players respectively. To steer, move the joystick up, down, right, or left. Diagonal motion is not allowed.

The joystick fire buttons toggle a pause feature. To freeze the action, quickly press and release the button (either joystick button works). This leaves you free to answer the phone or do other things. To restart the action, press and release the button again.

Heading Them Off at the Pass

There are four ways you can crash: hitting a wall, running into the other player's trail, crossing your own trail, or backing into yourself by trying to reverse your direction.

After a crash, the surviving player is awarded points equal to the number of segments in the crashed player's trail. This means that the longer the players last before crashing, the more points are at stake. Thus, it's possible to catch up even if you're way behind.

Each time you crash, you lose one "life." You start with ten lives, and the game ends when one player runs out of lives. After each crash, the screen updates the score and reminds you how many lives each player has left. To restart each round, press the joystick fire button.

When the game is over, you get a chance to change the skill level for the next game. Just to get a peek at how fast machine language can be, try a game at level 9. You'll be lucky if you can make one turn before crashing into a wall. Yet even this had to be slowed down with delay loops!

CUT-OFF!

For easy entry of this machine language program, be sure to read "Using the Machine Language Editor: MLX," Appendix D.

```
49152 :032,200,193,076,006,192,187
49158 :173,066,003,024,105,001,122
49164 :141,066,003,173,067,003,209
49170 :105,000,141,067,003,174,252
49176 :060,003,032,145,195,169,116
49182 :001,141,065,003,032,203,219
49188 :195,162,002,161,247,201,236
49194 :032,240,032,032,154,195,215
49200 :032,119,197,032,215,194,069
49206 :173,068,003,201,000,240,227
49212 :083,173,069,003,201,000,077
49218 :240,076,032,190,192,032,060
49224 :074,196,076,006,192,032,136
49230 :154,195,169,000,141,065,034
49236 :003,032,203,195,162,000,167
49242 :161,247,201,032,240,032,235
49248 :032,154,195,032,119,197,057
49254 :032,215,194,173,068,003,019
49260 :201,000,240,032,173,069,055
49266 :003,201,000,240,025,032,103
49272 :190,192,032,074,196,076,112
49278 :006,192,032,154,195,173,110
49284 :001,220,045,000,220,041,147
49290 :016,240,006,076,006,192,162
49296 :076,251,196,173,001,220,037
49302 :045,000,220,041,016,240,200
49308 :246,162,250,032,145,195,162
49314 :173,001,220,045,000,220,053
49320 :041,016,208,246,173,001,085
49326 :220,045,000,220,041,016,204
49332 :240,246,162,250,032,145,231
49338 :195,076,006,192,169,147,203
49344 :032,210,255,169,015,141,246
49350 :033,208,169,005,141,032,018
49356 :208,162,000,169,160,157,036
49362 :000,004,169,000,157,000,028
49368 :216,232,224,040,208,241,097
49374 :162,000,169,160,157,192,038
49380 :007,169,000,157,192,219,204
49386 :232,224,040,208,241,169,068
49392 :000,133,253,169,004,133,164
49398 :254,169,000,133,251,169,198
49404 :216,133,252,162,000,169,160
49410 :160,160,000,145,253,169,121
49416 :000,145,251,160,039,169,004
```

```
49422 :160,145,253,169,000,145,118
49428 :251,024,165,253,105,040,090
49434 :133,253,165,254,105,000,168
49440 :133,254,024,165,251,105,196
49446 :040,133,251,165,252,105,216
49452 :000,133,252,232,224,025,142
49458 :208,205,169,012,162,016,054
49464 :157,000,216,232,224,024,141
49470 :208,248,169,131,141,016,207
49476 :004,169,149,141,017,004,040
49482 :169,148,141,018,004,169,211
49488 :173,141,019,004,169,143,217
49494 :141,020,004,169,134,141,183
49500 :021,004,141,022,004,169,197
49506 :161,141,023,004,169,006,090
49512 :141,199,217,169,002,141,205
49518 :209,217,169,081,141,199,102
49524 :005,169,087,141,209,005,220
49530 :169,000,141,066,003,141,130
49536 :067,003,169,007,141,075,078
49542 :003,169,011,141,074,003,023
49548 :169,209,141,070,003,169,133
49554 :005,141,071,003,169,199,222
49560 :141,072,003,169,005,141,171
49566 :073,003,169,152,032,210,029
49572 :255,024,162,000,160,007,004
49578 :032,240,255,174,061,003,167
49584 :173,062,003,032,205,189,072
49590 :024,162,000,160,029,032,077
49596 :240,255,174,063,003,173,072
49602 :064,003,032,205,189,096,015
49608 :169,000,141,061,003,141,203
49614 :062,003,141,063,003,141,107
49620 :064,003,169,081,141,077,235
49626 :003,169,087,141,076,003,185
49632 :032,190,192,169,012,162,213
49638 :009,157,240,216,232,224,028
49644 :030,208,248,162,009,157,026
49650 :064,217,232,224,030,208,193
49656 :248,141,163,217,162,000,155
49662 :189,171,194,240,006,157,187
49668 :249,004,232,208,245,162,080
49674 :000,189,193,194,240,006,064
49680 :157,073,005,232,208,245,168
49686 :169,048,141,060,003,141,072
49692 :163,005,162,100,032,145,123
49698 :195,173,001,220,041,015,167
49704 :201,014,240,033,201,013,230
49710 :240,010,173,001,220,041,219
```

```
49716 :016,240,063,076,030,194,159
49722 :173,060,003,056,233,001,072
49728 :201,047,240,028,141,060,013
49734 :003,141,163,005,076,030,232
49740 :194,173,060,003,024,105,123
49746 :001,201,058,240,020,141,231
49752 :060,003,141,163,005,076,024
49758 :030,194,169,057,141,060,233
49764 :003,141,163,005,076,030,006
49770 :194,169,048,141,060,003,209
49776 :141,163,005,076,030,194,209
49782 :173,001,220,041,016,240,041
49788 :249,162,250,032,145,195,133
49794 :173,060,003,056,233,048,191
49800 :170,169,050,141,060,003,217
49806 :224,000,240,013,173,060,084
49812 :003,056,233,005,141,060,134
49818 :003,202,076,142,194,032,035
49824 :190,192,169,010,141,068,162
49830 :003,141,069,003,096,013,235
49836 :015,022,005,032,010,015,015
49842 :025,019,020,009,003,011,009
49848 :032,021,016,047,004,015,063
49854 :023,014,000,020,015,032,038
49860 :003,008,015,015,019,005,005
49866 :032,019,011,009,012,012,041
49872 :032,012,005,022,005,012,040
49878 :000,173,070,003,205,072,225
49884 :003,240,003,076,099,195,068
49890 :173,071,003,205,073,003,242
49896 :208,121,173,075,003,174,218
49902 :074,003,201,014,240,012,014
49908 :201,007,240,015,201,013,153
49914 :240,018,201,011,240,021,213
49920 :224,013,240,024,076,099,164
49926 :195,224,011,240,017,076,001
49932 :099,195,224,014,240,010,026
49938 :076,099,195,224,007,240,091
49944 :003,076,099,195,173,061,119
49950 :003,024,109,066,003,141,120
49956 :061,003,173,062,003,105,187
49962 :000,141,062,003,173,067,232
49968 :003,024,109,062,003,141,134
49974 :062,003,174,068,003,202,054
49980 :142,068,003,173,063,003,000
49986 :024,109,066,003,141,063,216
49992 :003,173,064,003,105,000,164
49998 :141,064,003,173,067,003,017
50004 :024,109,064,003,141,064,233
```

```
50010 :003,174,069,003,202,142,171
50016 :069,003,096,173,065,003,249
50022 :010,170,189,061,003,024,047
50028 :109,066,003,157,061,003,251
50034 :189,062,003,105,000,157,118
50040 :062,003,173,067,003,024,196
50046 :125,062,003,157,062,003,026
50052 :174,065,003,189,068,003,122
50058 :056,233,001,157,068,003,144
50064 :096,160,000,200,208,253,037
50070 :202,208,248,096,174,065,119
50076 :003,188,076,003,138,010,062
50082 :170,181,247,157,070,003,222
50088 :181,248,157,071,003,152,212
50094 :129,247,181,248,024,105,084
50100 :212,149,248,224,002,208,199
50106 :008,169,006,129,247,032,009
50112 :170,197,096,169,002,129,187
50118 :247,032,170,197,096,174,090
50124 :065,003,189,000,220,041,210
50130 :015,201,014,240,018,201,131
50136 :007,240,038,201,013,240,187
50142 :058,201,011,240,078,189,231
50148 :074,003,076,211,195,157,176
50154 :074,003,138,010,170,056,173
50160 :189,070,003,233,040,149,156
50166 :247,189,071,003,233,000,221
50172 :149,248,076,073,196,157,127
50178 :074,003,138,010,170,024,165
50184 :189,070,003,105,001,149,013
50190 :247,189,071,003,105,000,117
50196 :149,248,076,073,196,157,151
50202 :074,003,138,010,170,024,189
50208 :189,070,003,105,040,149,076
50214 :247,189,071,003,105,000,141
50220 :149,248,076,073,196,157,175
50226 :074,003,138,010,170,056,245
50232 :189,070,003,233,001,149,189
50238 :247,189,071,003,233,000,037
50244 :149,248,076,073,196,096,138
50250 :162,000,189,219,196,240,056
50256 :006,157,255,004,232,208,174
50262 :245,162,000,189,232,196,086
50268 :240,006,157,079,005,232,043
50274 :208,245,162,000,189,244,122
50280 :196,240,006,157,017,006,214
50286 :232,208,245,162,000,189,122
50292 :244,196,240,006,157,033,224
50298 :006,232,208,245,169,012,226
```

```
50304 :162,009,157,240,216,232,120
50310 :224,030,208,248,162,009,247
50316 :157,064,217,232,224,030,040
50322 :208,248,162,001,157,008,162
50328 :218,232,224,037,208,248,039
50334 :024,162,013,160,015,032,052
50340 :240,255,174,069,003,169,050
50346 :000,032,205,189,024,162,014
50352 :013,160,031,032,240,255,139
50358 :174,068,003,169,000,032,116
50364 :205,189,173,001,220,045,253
50370 :000,220,041,016,208,246,157
50376 :173,001,220,045,000,220,091
50382 :041,016,240,246,162,000,143
50388 :032,145,195,032,190,192,230
50394 :096,016,018,005,019,019,135
50400 :032,002,021,020,020,015,078
50406 :014,000,020,015,032,003,058
50412 :015,014,020,009,014,021,073
50418 :005,000,012,009,022,005,039
50424 :019,061,000,032,190,192,230
50430 :162,000,189,082,197,240,100
50436 :006,157,000,005,232,208,100
50442 :245,162,000,189,092,197,127
50448 :240,006,157,071,005,232,215
50454 :208,245,169,012,162,001,051
50460 :157,240,216,232,224,030,103
50466 :208,248,162,001,157,064,106
50472 :217,232,224,037,208,248,182
50478 :173,001,220,045,000,220,193
50484 :041,016,208,246,162,250,207
50490 :032,145,195,173,001,220,056
50496 :045,000,220,041,016,240,114
50502 :246,162,250,032,145,195,076
50508 :032,200,193,076,006,192,007
50514 :007,001,013,005,032,015,155
50520 :022,005,018,000,016,018,167
50526 :005,019,019,032,002,021,192
50532 :020,020,015,014,032,020,221
50538 :015,032,016,012,001,025,207
50544 :032,001,007,001,009,014,176
50550 :000,169,015,141,024,212,167
50556 :169,129,141,004,212,169,180
50562 :009,141,005,212,169,100,254
50568 :141,000,212,169,012,141,043
50574 :001,212,169,015,141,032,200
50580 :208,162,080,032,145,195,202
50586 :056,233,001,201,000,208,085
50592 :241,169,000,141,004,212,159
```

```
50598 :141,005,212,096,169,008,029
50604 :141,024,212,169,016,141,107
50610 :005,212,169,128,141,006,071
50616 :212,169,010,162,000,024,249
50622 :109,065,003,232,224,010,065
50628 :208,247,141,001,212,169,150
50634 :037,141,000,212,169,033,026
50640 :141,004,212,174,060,003,034
50646 :032,145,195,169,000,141,128
50652 :004,212,141,005,212,141,167
50658 :006,212,096,013,013,013,067
```

Astro-PANIC!

Charles Brannon

Written entirely in ultra-fast machine language, "Astro-PANIC!" is an arcade-style space game with multicolored sprites and 15 frantic levels of difficulty. Will you be the first human to make it to level 15? Joystick required.

"Astro-PANIC!" is a fast-paced, high-speed, all-machine-language game. The object is to defend your cannon, maneuvering it left and right as alien saucers dodge and dive in a relentless attack.

Plug a joystick into port 2 to play. After loading from tape or disk (see special instructions below), type SYS 49152 to run the program. The screen clears to black with a gray score window at the bottom. Press the f7 function key to start the action.

Swooping Saucers

Instantly, seven alien saucers begin to sweep about the screen. The saucers keep moving in their current direction until they hit a screen boundary, then rebound. Sometimes they change speed. Meanwhile, you move your cannon left and right to evade the erratic dives and swoops of the saucers. The slightest contact with an alien saucer spells destruction.

Fortunately, you have your super weapon, a laser/heat ray/particle beam/thermonuclear cannon. Simply press the trigger button on the joystick to unleash a bolt of this incredible power. The bolt continues until it hits a saucer, atomizing it, or until it reaches the top of the screen. Hold down the fire button to access the bolt's automatic fire mode.

You can pause the game at any time by pressing SHIFT or freeze it by depressing the SHIFT LOCK key. Press the SHIFT LOCK a second time to continue the game.

Scoring varies from time to time. It does depend on the saucer's postion, but it's not fixed. Sometimes you'll get a large score for dematerializing a saucer near the top of the screen, other times you'll be rewarded with many points if you blast one nearer your cannon. If you're concerned about a high score (rather than just surviving), there's just no rule for

where or when to shoot. Your best bet is to simply destroy as many as possible, as quickly as you can.

If you destroy all seven saucers, you advance to a new screen. Each level (1–15) is faster than the previous one and is indicated in the score window. Be warned—levels 10 and above are manic!

You lose a cannon whenever a saucer collides with your cannon. The game is over after you lose all three cannons to the marauding saucers. The scoreboard keeps track of the high score during the current session. Press f7 to start another game. Watch the time, though: Some people don't know when to quit!

PANICking

Keep moving. It's more important to protect your cannon than to make that tricky shot. Dodge the aliens first, shoot later. You won't always want to hold down the fire button to repeat, since sometimes a shot will be in the air when you'd rather shoot the alien right above you. Keep an eye on the movement of the saucers, so you can sometimes synchronize several wipe-out shots. Watch for the edges of the screen. Aliens will sometimes bounce off an edge right into you.

Typing the Program

To type Astro-PANIC!, use MLX, the Machine Language Editor, which virtually guarantees fool-proof entry of machine language programs. You'll find a complete description, as well as the MLX program itself, in Appendix D. (If you've previously typed in MLX, either for another machine language game in this book, or for other programs in *COMPUTE!'s Gazette* magazine or *COMPUTE!* magazine, you can use it again for Astro-PANIC!.) Here's the information you'll need to enter Astro-PANIC! with MLX:

Starting address—49152
Ending address—50777

After you are finished typing, MLX will let you save the program to tape or disk. To load Astro-PANIC!, enter LOAD"*filename*",8,1 for disk, or LOAD"*filename*",1,1 for tape. SYS 49152 to begin.

During our in-house testing of Astro-PANIC!, no one ever made it beyond level 12. We've since heard from two *COMPUTE!'s Gazette* readers who have made it to level 15 and

beyond. The game seems to slow down somewhat at those levels. It may be a moot point: Level 15 waits for the truly dedicated saucer-smasher.

Keep a sharp eye for any strange-looking lights in the sky!

Astro-PANIC!

For easy entry of this machine language program, be sure to read "Using the Machine Language Editor: MLX," Appendix D.

```
49152 :076,011,193,120,169,127,184
49158 :141,013,220,169,001,141,179
49164 :026,208,169,233,141,018,039
49170 :208,169,027,141,017,208,020
49176 :169,036,141,020,003,169,050
49182 :192,141,021,003,088,096,059
49188 :173,018,208,201,233,208,053
49194 :031,169,000,141,018,208,097
49200 :169,022,141,024,208,169,013
49206 :200,141,022,208,169,012,038
49212 :141,033,208,141,032,208,055
49218 :169,001,141,025,208,076,174
49224 :005,193,169,233,141,018,063
49230 :208,169,030,141,024,208,090
49236 :169,216,141,022,208,169,241
49242 :000,141,032,208,141,033,133
49248 :208,169,001,141,025,208,080
49254 :230,162,032,159,255,173,089
49260 :141,002,013,137,198,240,071
49266 :003,076,005,193,173,089,141
49272 :198,141,000,208,173,016,088
49278 :208,041,254,013,090,198,162
49284 :141,016,208,173,000,220,122
49290 :041,004,208,029,173,090,171
49296 :198,208,007,173,089,198,249
49302 :201,025,144,017,056,173,254
49308 :089,198,233,002,141,089,140
49314 :198,173,090,198,233,000,030
49320 :141,090,198,173,000,220,222
49326 :041,008,208,029,173,090,211
49332 :198,240,007,173,089,198,061
49338 :201,064,176,017,024,173,073
49344 :089,198,105,002,141,089,048
49350 :198,173,090,198,105,000,194
49356 :141,090,198,173,000,220,002
49362 :041,016,208,047,173,088,015
49368 :198,208,042,056,173,089,214
49374 :198,233,024,133,180,173,139
49380 :090,198,233,000,074,102,157
49386 :180,070,180,070,180,238,128
```

```
49392 :088,198,024,165,180,105,232
49398 :033,133,251,133,253,169,194
49404 :007,105,000,133,252,105,086
49410 :212,133,254,104,168,104,209
49416 :170,104,064,032,003,192,061
49422 :169,004,133,252,160,000,220
49428 :185,226,196,153,000,056,068
49434 :200,192,008,208,245,160,015
49440 :000,152,153,000,057,200,082
49446 :192,008,208,248,160,000,086
49452 :185,234,196,153,000,058,102
49458 :200,208,247,169,232,141,223
49464 :248,007,160,007,169,233,112
49470 :153,248,007,136,208,250,040
49476 :169,255,141,028,208,169,014
49482 :000,141,029,208,141,023,104
49488 :208,141,016,208,169,003,057
49494 :141,037,208,169,008,141,022
49500 :038,208,169,212,141,001,093
49506 :208,160,000,185,236,197,060
49512 :153,040,208,200,192,007,136
49518 :208,245,169,006,141,039,150
49524 :208,169,147,032,210,255,113
49530 :160,039,169,160,153,152,187
49536 :007,169,005,153,152,219,065
49542 :136,016,243,160,005,162,088
49548 :024,024,032,240,255,160,107
49554 :000,185,244,197,240,006,250
49560 :032,210,255,200,208,245,022
49566 :160,000,162,023,024,032,047
49572 :240,255,160,000,185,025,005
49578 :198,240,006,032,210,255,087
49584 :200,208,245,169,004,141,119
49590 :035,208,169,014,141,036,017
49596 :208,160,024,169,000,153,134
49602 :000,212,136,016,250,169,209
49608 :255,141,015,212,169,128,096
49614 :141,018,212,169,143,141,006
49620 :024,212,169,015,141,139,144
49626 :198,169,003,141,136,198,039
49632 :169,000,141,088,198,141,193
49638 :137,198,170,142,090,198,141
49644 :169,184,141,089,198,138,131
49650 :010,168,010,010,010,010,204
49656 :024,105,031,153,091,198,082
49662 :169,000,153,092,198,153,251
49668 :003,208,169,060,157,105,194
49674 :198,032,186,195,232,224,053
49680 :007,208,222,169,255,141,250
```

```
49686 :021,208,173,030,208,173,067
49692 :031,208,173,030,208,041,207
49698 :001,240,003,076,230,195,011
49704 :173,141,002,208,251,032,079
49710 :228,255,201,136,208,009,059
49716 :169,032,160,000,145,251,041
49722 :076,188,196,173,088,198,209
49728 :208,003,076,027,195,160,221
49734 :000,169,032,145,251,056,211
49740 :165,251,233,040,133,251,125
49746 :133,253,165,252,233,000,094
49752 :133,252,024,105,212,133,179
49758 :254,173,027,212,009,008,009
49764 :145,253,169,000,145,251,039
49770 :173,031,208,041,254,240,029
49776 :103,133,167,141,138,198,224
49782 :162,000,070,167,070,167,242
49788 :144,071,169,032,141,005,174
49794 :212,169,246,141,006,212,092
49800 :169,129,141,004,212,169,192
49806 :234,157,249,007,160,010,191
49812 :173,027,212,157,040,208,197
49818 :140,001,212,165,162,197,007
49824 :162,240,252,136,208,238,116
49830 :189,236,197,157,040,208,169
49836 :169,233,157,249,007,169,132
49842 :128,141,004,212,138,072,105
49848 :189,105,198,073,255,074,054
49854 :074,074,032,201,196,104,103
49860 :170,232,224,007,208,176,189
49866 :173,138,198,073,255,045,060
49872 :021,208,141,021,208,076,115
49878 :226,194,238,088,198,173,051
49884 :088,198,201,021,208,057,225
49890 :160,000,140,088,198,169,213
49896 :032,145,251,173,031,208,048
49902 :173,030,208,173,021,208,027
49908 :041,254,208,035,173,139,070
49914 :198,240,003,206,139,198,210
49920 :160,038,162,023,024,032,183
49926 :240,255,173,139,198,073,060
49932 :015,170,169,000,032,205,091
49938 :189,169,100,032,201,196,137
49944 :076,224,193,174,139,198,004
49950 :160,000,200,208,253,202,029
49956 :208,250,138,010,168,185,227
49962 :091,198,153,002,208,189,115
49968 :105,198,153,003,208,185,132
49974 :092,198,133,168,056,138,071
```

```
49980 :168,200,200,169,000,042,071
49986 :136,208,252,133,167,073,011
49992 :255,045,016,208,164,168,160
49998 :240,002,005,167,141,016,137
50004 :208,232,224,007,208,204,143
50010 :162,000,138,010,168,189,245
50016 :105,198,024,125,112,198,090
50022 :201,210,176,004,201,050,176
50028 :176,006,032,186,195,076,011
50034 :178,195,157,105,198,024,203
50040 :185,091,198,121,120,198,009
50046 :133,167,185,092,198,121,254
50052 :121,198,133,168,208,014,206
50058 :165,167,201,031,240,002,176
50064 :176,006,032,186,195,076,047
50070 :178,195,165,168,240,012,084
50076 :165,167,201,064,144,006,135
50082 :032,186,195,076,178,195,000
50088 :165,167,153,091,198,165,083
50094 :168,153,092,198,232,224,217
50100 :007,208,165,076,030,194,092
50106 :134,169,132,170,173,027,223
50112 :212,041,005,170,189,070,111
50118 :198,166,169,157,112,198,174
50124 :173,027,212,041,005,010,160
50130 :168,185,076,198,166,170,149
50136 :157,120,198,185,077,198,127
50142 :157,121,198,164,170,166,174
50148 :169,096,169,235,141,248,006
50154 :007,169,001,141,137,198,119
50160 :169,009,141,005,212,169,177
50166 :160,141,006,212,169,033,199
50172 :141,004,212,162,100,142,245
50178 :001,212,160,000,173,027,063
50184 :212,141,039,208,141,000,237
50190 :212,136,208,244,202,208,200
50196 :236,169,234,141,248,007,031
50202 :169,001,141,029,208,141,203
50208 :023,208,169,032,141,004,097
50214 :212,169,168,141,006,212,178
50220 :169,129,141,004,212,162,093
50226 :100,142,001,212,160,000,153
50232 :140,000,212,173,027,212,052
50238 :141,039,208,136,208,244,014
50244 :202,208,236,169,232,141,232
50250 :248,007,169,006,141,039,172
50256 :208,169,000,141,029,208,067
50262 :141,023,208,169,128,141,128
50268 :004,212,162,100,160,000,218
```

```
50274 :136,208,253,202,208,250,075
50280 :169,000,141,137,198,168,149
50286 :153,002,208,200,192,014,111
50292 :208,248,141,016,208,160,073
50298 :000,169,032,145,251,173,124
50304 :030,208,206,158,007,206,175
50310 :136,198,173,136,198,240,191
50316 :003,076,224,193,160,000,028
50322 :185,203,007,217,222,007,219
50328 :240,005,176,011,076,178,070
50334 :196,200,192,006,208,238,174
50340 :076,178,196,160,006,185,197
50346 :202,007,153,221,007,136,128
50352 :208,247,032,159,255,032,085
50358 :228,255,201,136,208,246,176
50364 :160,006,169,048,153,202,158
50370 :007,136,208,250,076,158,005
50376 :193,170,160,006,056,185,202
50382 :202,007,105,000,201,058,011
50388 :144,002,169,048,153,202,162
50394 :007,136,208,239,202,208,194
50400 :233,096,048,016,032,048,185
50406 :016,032,048,016,000,000,086
50412 :000,000,000,000,000,032,012
50418 :000,000,236,000,000,236,202
50424 :000,003,255,000,015,255,008
50430 :192,015,087,192,015,255,242
50436 :192,063,087,240,255,255,072
50442 :252,255,087,252,255,255,086
50448 :252,239,087,236,239,255,044
50454 :236,236,220,236,236,220,126
50460 :236,236,220,236,252,000,184
50466 :252,000,000,000,000,000,030
50472 :000,000,000,000,000,000,040
50478 :000,000,000,000,000,000,046
50484 :000,000,003,085,192,013,089
50490 :085,112,063,255,252,234,035
50496 :170,171,226,034,043,058,254
50502 :170,172,013,085,112,003,113
50508 :255,192,000,000,000,000,011
50514 :000,000,000,000,000,000,082
50520 :000,000,000,000,000,000,088
50526 :000,000,000,000,000,000,094
50532 :000,000,000,000,000,000,100
50538 :128,032,008,032,160,032,242
50544 :000,168,160,034,162,160,028
50550 :138,168,130,162,197,042,187
50556 :040,115,008,131,190,224,064
50562 :046,188,162,011,127,064,216
```

```
50568 :130,201,088,010,186,074,057
50574 :035,190,232,000,141,032,004
50580 :010,038,010,038,166,160,058
50586 :000,168,168,130,138,040,030
50592 :010,003,128,008,032,000,085
50598 :000,002,032,000,000,000,200
50604 :000,000,000,000,000,016,188
50610 :000,000,118,000,000,118,158
50616 :000,001,255,128,007,255,062
50622 :224,007,171,224,007,255,054
50628 :224,031,171,248,127,255,228
50634 :254,127,171,254,127,255,110
50640 :254,119,171,246,119,255,092
50646 :246,118,110,118,118,110,010
50652 :118,118,110,118,126,000,042
50658 :126,000,000,000,000,000,096
50664 :000,000,000,013,006,004,255
50670 :001,007,008,012,013,014,037
50676 :008,031,211,067,079,082,210
50682 :069,058,158,048,048,048,167
50688 :048,048,048,032,032,149,101
50694 :200,073,071,072,032,211,153
50700 :067,079,082,069,058,028,139
50706 :048,048,048,048,048,048,050
50712 :000,018,149,204,073,086,042
50718 :069,083,058,051,029,029,093
50724 :029,029,029,029,029,028,209
50730 :193,083,084,082,079,045,096
50736 :208,193,206,201,195,033,060
50742 :029,029,029,029,029,029,228
50748 :031,204,069,086,069,076,083
50754 :058,048,032,000,001,255,204
50760 :002,254,003,253,001,000,073
50766 :255,255,002,000,254,255,075
50772 :003,000,253,255,013,013,109
```

Nessie

Tom R. Halfhill

64 Version by Charles Brannon

In "Nessie," a nonviolent action game written entirely in machine language, you're posing as a photographer trying to snap a clear photograph of the Loch Ness monster. You need a joystick to aim your camera.

For decades, fans and believers of Scotland's Loch Ness monster have affectionately referred to the mysterious creature as Nessie—hence the title of this game.

Inspired by a TV documentary on Loch Ness that recounted the hundreds of attempts to photograph the monster, this game tries to re-create the difficulties of capturing the creature on film. Whether in the game or in reality, it's not easy. Almost all of the photographic attempts have ended in failure; there exist only a few controversial photos showing parts of fins, shadowy shapes, and blurred figures. Maybe your steady hand and sharp eye will succeed where others have failed.

Starting Nessie

Nessie is an all-machine-language game. Normally, it's quite difficult to type in a machine language (ML) program. That's because ML is simply a series of numbers. To make it easier to enter programs like Nessie, we've developed MLX, the Machine Language Editor. With MLX, you can almost guarantee that you'll type the program in correctly the first time. Before you begin entering Nessie, then, make sure you read and understand Appendix D. You'll find the MLX program listed there. You need a copy of it on tape or disk before you can start typing this game. (You may already have a copy if you used it to type in one of the other ML games in this book or if you typed in a copy from an issue of *COMPUTE!'s Gazette* magazine. If you have an older version of MLX, you may want to retype the listing in Appendix D, for it includes a recent addition that turns part of the keyboard into a numeric keypad. It makes entry even easier.)

To type in Nessie, load and run MLX. You'll have to supply two addresses. They are:

Starting address: 49152
Ending address: 52169

After you've saved Nessie to tape or disk, you can load it at any time by entering:

LOAD"*filename*",8,1 (for disk)
LOAD"*filename*",1,1 (for tape)

Type NEW after Nessie has loaded. This will not erase the program, but will reset some pointers that otherwise might give you an ?OUT OF MEMORY ERROR or cause other problems with the game. To start Nessie, type:

SYS 49152

Instantly, the game screen appears, with a boxed-in crosshair in the center of the screen. There are also prompts to select game options. The keys to press for each are:

f1: Telephoto
f3: Wide Angle
f5: Speed
f7: Easy Game
f8: Hard Game

There are two types of camera lenses: *Telephoto* and *Wide Angle*. By far the easiest is the wide angle, obtained by pressing the f3 key. A wide angle lens allows you to cover more area from your camera position. You'll see the viewfinder expand when you press the f3 key. Since its larger size makes it easier to enclose Nessie, you may want to begin play with this. The telephoto lens, which is the default selection (in other words, if you don't make a choice, this lens is used automatically), is about half as large, making it harder to catch Nessie within the border of the lens. In fact, the telephoto viewfinder barely frames Nessie. Press the f1 key to switch back to the telephoto lens if you previously selected the wide angle lens.

Function key f5 is used to increase or decrease the speed. The default speed is 5. Speed 1 is very slow, 9 almost too fast. Hitting the f5 key repeatedly cycles through the levels from 1 to 9. Make sure your joystick is plugged into port 2. Now you're ready to begin. Press f7 for an easy (default) game, or f8 for a more difficult level of play. (Pressing the f7 key during play restarts the game, something you might find handy.) If you chose the hard game, the playfield will be littered with

black squares that interfere with your photography. It's better to begin with the easy version.

The photo session has started, and the clock is moving.

Getting the Whole Picture

At the top left of the screen is your camera's film counter, which shows how many pictures remain on your roll of film. You start with a 20-exposure roll. Each time you snap a picture, the film counter decrements. The current game speed also displays.

Your camera viewfinder starts in the center of the screen. You can move it in any direction with the joystick. Pressing the fire button releases the shutter. The viewfinder frame itself is blue, with a red aiming crosshair in the center. To take a properly centered photo, you must position the crosshair over Nessie. If any part of Nessie is touching the viewfinder frame when you snap the shutter, it will register as a cropped photo when the film is developed at the end of the game. A picture of a piece of Nessie is better than nothing, but it's not nearly as valuable as a photo of the whole monster. (Let's face it, wouldn't you feel better walking into *The New York Times* with an indisputable picture of Nessie instead of a doubtful snapshot of a dorsal fin?)

For the same reason, you must be careful not to include any other objects in the viewfinder while photographing Nessie. This isn't as easy as it sounds. When you start the game, you'll find that Loch Ness is alive with turtles and eels. If you photograph one of these instead, you've been "fooled"—and your photo is worth only a handful of points. (The reason you get any points at all is that you might be able to sell the photo to *Field & Stream* or an airline magazine.) The eels are particularly troublesome. They bear an uncanny resemblance to Nessie, which is why so many hopeful photographers over the years have been fooled.

Make sure your camera's viewfinder doesn't touch any text (such as the score line) or any of the black squares in the hard game, or you'll just end up with a cropped picture.

Another hazard to beware of is jittery hands. Nessie is not an easy target—the creature swims around the Loch in random directions, staying still for only a moment before slipping away. Meanwhile, you're trying to center the monster in the viewfinder. If you snap the shutter while moving the finder,

the picture will be blurred. And that's worth zero points.

Pressing any key (except for the CTRL, RESTORE, SHIFT, and Commodore keys) freezes the screen. Hitting another key restores motion. You can do this if you want to take a short break from the game. Or if you get particularly frustrated with Nessie's darting behavior, you can use this feature to "freeze" the Loch and capture her on film. It's actually cheating, but....

When you get down to your last five pictures on the roll of film, the viewfinder frame automatically turns from blue to bright yellow as a warning. This is in case you're too busy to pay attention to the film counter.

Developing the Film

After you snap your last shot, the film instantly develops (machine language—faster than Polaroid) and is displayed. Each of the 20 finished prints shows what you photographed when you snapped the shutter. They are arranged in the order you shot them. (Programmers may want to note that each picture is a sprite. Using raster interrupts permits you to have up to 64 sprites simultaneously. In this program, we have 20 sprites displayed on one screen.) The frames show a whole Nessie, a cropped Nessie, a colored blur, another Loch creature, or emptiness, depending on your photographic skill. At the bottom of the screen is your final score, adjusted for the amount of time that elapsed.

To restart Nessie, just snap the shutter button. This returns you to the setup screen, where you can change lenses, if you wish, before playing again.

Pictures for Points

Since the telephoto lens is harder to use, it scores more points. A clear, properly framed photo of Nessie taken with the wide angle lens scores 2000 points, but a whopping 20,000 points if you used the telephoto lens. If Nessie is not completely enclosed (cropped), the photo scores only 100 points (1000 with the telephoto). Snapping a picture of a turtle or eel means you've been fooled, and you receive only 50 points, or 500 if you were using the telephoto. Moving the camera when you press the shutter button gives you a blurred picture (shown by the horizontal lines across the developed print). You get no points for this, just as you receive no reward for taking a photo of the Loch.

Another important factor in the scoring is time. One point (or ten points with the telephoto) is subtracted from your final score for every four seconds you took to shoot all your pictures. You have to balance your pace between reckless photography and careful shooting in order to get the best pictures in the least amount of time.

Anyone Can Use a Camera

Nessie is a game that can be enjoyed by almost anyone. Children especially will delight in capturing the creature on instant film. Setting the speed level allows you to slow down or speed up the creatures' movements. Slower speeds are appropriate for smaller children. Nessie is easy to snap, and there's none of the usual frustration with action games.

But set the speed to 8 or 9, the game selection to hard, and Nessie becomes a challenge to even the most agile photographer. You'll need a quick shutter and lots of luck to see the monster on your developed prints. In fact, it's almost as hard as getting the real Loch Ness monster on film.

Nessie

For easy entry of this machine language program, be sure to read "Using the Machine Language Editor: MLX," Appendix D.

```
49152 :169,071,133,251,169,199,224
49158 :133,252,169,000,133,253,178
49164 :169,056,133,254,162,004,022
49170 :160,255,177,251,145,253,235
49176 :136,192,255,208,247,230,012
49182 :252,230,254,202,048,007,255
49188 :208,238,160,127,076,020,097
49194 :192,198,254,160,128,169,119
49200 :000,145,253,200,208,251,081
49206 :169,255,141,014,212,141,218
49212 :015,212,169,128,141,018,231
49218 :212,169,000,141,024,212,056
49224 :169,252,141,027,208,162,007
49230 :005,032,189,193,169,128,026
49236 :157,205,203,157,211,203,196
49242 :169,000,157,199,203,189,239
49248 :024,199,157,041,208,189,146
49254 :032,199,157,250,007,138,117
49260 :010,168,169,000,153,005,101
49266 :208,202,016,217,173,022,184
49272 :199,141,039,208,173,023,135
49278 :199,141,040,208,173,030,149
```

```
49284 :199,141,248,007,173,031,163
49290 :199,141,249,007,169,000,135
49296 :141,236,203,169,160,141,170
49302 :235,203,141,000,208,141,054
49308 :002,208,169,128,141,237,017
49314 :203,141,001,208,141,003,091
49320 :208,169,255,141,021,208,146
49326 :141,028,208,169,003,141,096
49332 :029,208,141,023,208,169,190
49338 :006,141,032,208,141,033,235
49344 :208,169,005,141,037,208,192
49350 :169,004,141,038,208,169,159
49356 :001,141,246,203,032,231,034
49362 :193,120,169,118,141,020,203
49368 :003,169,196,141,021,003,237
49374 :088,160,000,132,007,152,249
49380 :010,170,185,199,203,074,045
49386 :102,007,185,205,203,157,069
49392 :004,208,185,211,203,157,184
49398 :005,208,201,050,144,007,093
49404 :201,229,176,003,076,014,183
49410 :193,185,229,203,073,255,116
49416 :024,105,001,153,229,203,211
49422 :185,199,203,240,010,185,012
49428 :205,203,201,064,176,013,114
49434 :076,062,193,185,205,203,182
49440 :201,024,144,003,076,062,030
49446 :193,169,000,153,223,203,211
49452 :185,217,203,073,255,024,233
49458 :105,001,153,217,203,201,162
49464 :255,208,003,153,223,203,077
49470 :185,250,007,073,001,153,219
49476 :250,007,185,217,203,170,076
49482 :185,250,007,009,002,153,168
49488 :250,007,224,001,240,005,039
49494 :073,002,153,250,007,024,083
49500 :185,205,203,121,217,203,202
49506 :153,205,203,185,199,203,222
49512 :121,223,203,153,199,203,182
49518 :024,185,211,203,121,229,059
49524 :203,153,211,203,200,192,254
49530 :006,240,003,076,227,192,098
49536 :173,016,208,041,003,005,062
49542 :007,141,016,208,174,245,157
49548 :203,240,008,160,000,136,119
49554 :208,253,202,208,250,173,160
49560 :027,212,201,128,144,011,107
49566 :173,027,212,201,006,176,185
49572 :249,170,032,189,193,032,005
```

```
49578 :228,255,240,012,201,136,218
49584 :208,003,076,208,197,032,132
49590 :228,255,240,251,076,223,175
49596 :192,169,000,157,223,203,108
49602 :173,027,212,041,003,168,050
49608 :185,060,199,157,217,203,197
49614 :201,255,208,003,157,223,229
49620 :203,173,027,212,041,003,103
49626 :168,185,060,199,157,229,192
49632 :203,029,217,203,240,215,051
49638 :096,169,000,141,239,203,054
49644 :141,240,203,141,241,203,125
49650 :141,242,203,141,243,203,135
49656 :141,076,204,169,020,141,231
49662 :244,203,169,005,141,245,237
49668 :203,169,191,160,194,032,185
49674 :030,171,169,240,160,194,206
49680 :032,030,171,032,228,255,252
49686 :240,251,201,133,208,018,049
49692 :169,224,141,248,007,169,218
49698 :225,141,249,007,169,000,057
49704 :141,243,203,076,019,194,148
49710 :201,134,208,018,169,226,234
49716 :141,248,007,169,227,141,217
49722 :249,007,169,001,141,243,100
49728 :203,076,019,194,201,135,124
49734 :208,030,238,245,203,173,143
49740 :245,203,016,005,169,009,211
49746 :141,245,203,201,010,144,002
49752 :005,169,000,141,245,203,083
49758 :009,048,141,006,004,076,122
49764 :019,194,201,136,240,051,173
49770 :201,140,208,165,238,076,110
49776 :204,032,157,194,162,050,143
49782 :160,040,173,027,212,201,163
49788 :167,176,249,133,253,173,251
49794 :027,212,041,003,024,105,030
49800 :004,133,254,169,160,145,233
49806 :253,165,254,105,212,133,240
49812 :254,169,000,145,253,202,147
49818 :208,220,096,169,191,160,174
49824 :194,032,030,171,173,245,237
49830 :203,009,048,141,006,004,065
49836 :056,169,009,237,245,203,067
49842 :141,245,203,014,245,203,205
49848 :014,245,203,014,245,203,084
49854 :096,147,146,014,159,211,195
49860 :080,069,069,068,058,158,186
49866 :053,032,032,032,032,032,159
```

```
49872 :032,032,032,154,206,069,221
49878 :083,083,073,069,032,032,074
49884 :032,032,032,032,155,198,189
49890 :073,076,077,032,204,069,245
49896 :070,084,058,158,050,048,188
49902 :013,000,153,018,070,049,029
49908 :146,058,212,069,076,069,106
49914 :080,072,079,084,079,032,164
49920 :018,070,051,146,058,215,046
49926 :073,068,069,032,193,078,007
49932 :071,076,069,032,018,070,092
49938 :053,146,058,211,080,069,123
49944 :069,068,013,018,070,055,061
49950 :146,058,197,065,083,089,156
49956 :032,199,065,077,069,032,254
49962 :018,070,056,146,058,200,078
49968 :065,082,068,032,199,065,047
49974 :077,069,000,173,000,220,081
49980 :072,041,016,141,238,203,003
49986 :104,041,015,073,015,170,228
49992 :024,173,235,203,125,049,113
49998 :199,141,235,203,141,000,229
50004 :208,141,002,208,173,236,028
50010 :203,125,038,199,141,236,008
50016 :203,074,008,173,016,208,010
50022 :074,074,040,008,042,040,124
50028 :042,141,016,208,024,173,200
50034 :237,203,125,060,199,141,055
50040 :237,203,141,001,208,141,027
50046 :003,208,173,238,203,240,167
50052 :013,169,000,141,247,203,137
50058 :169,001,141,246,203,076,206
50064 :160,195,173,247,203,240,082
50070 :023,173,246,203,240,004,015
50076 :206,246,203,096,169,006,058
50082 :141,032,208,141,033,208,157
50088 :169,128,141,004,212,096,150
50094 :206,038,004,173,038,004,125
50100 :201,048,176,020,169,057,083
50106 :141,038,004,206,037,004,104
50112 :173,037,004,201,049,176,064
50118 :005,169,032,141,037,004,074
50124 :169,031,141,005,212,141,135
50130 :024,212,169,003,141,006,253
50136 :212,169,255,141,000,212,181
50142 :141,001,212,141,004,212,165
50148 :169,004,141,074,204,162,214
50154 :006,172,244,203,136,169,140
50160 :000,153,032,204,173,074,108
```

```
50166 :204,009,002,141,021,208,063
50172 :173,030,208,032,127,196,250
50178 :173,030,208,041,253,025,220
50184 :032,204,153,032,204,014,135
50190 :074,204,202,208,225,169,072
50196 :001,141,032,208,141,033,064
50202 :208,141,247,203,169,004,230
50208 :141,074,204,162,006,169,020
50214 :000,153,052,204,173,074,182
50220 :204,009,001,141,021,208,116
50226 :173,030,208,032,127,196,048
50232 :173,030,208,041,254,025,019
50238 :052,204,153,052,204,014,229
50244 :074,204,202,208,225,173,130
50250 :031,208,032,127,196,173,073
50256 :031,208,153,248,203,173,072
50262 :000,220,041,015,073,015,194
50268 :153,012,204,169,255,141,002
50274 :021,208,206,244,203,173,129
50280 :244,203,240,038,201,005,011
50286 :208,005,169,007,141,040,168
50292 :208,096,032,137,196,032,049
50298 :057,195,076,049,234,032,253
50304 :130,196,173,018,208,201,030
50310 :255,208,249,238,241,203,248
50316 :208,003,238,242,203,096,106
50322 :169,147,032,210,255,169,104
50328 :000,141,032,208,141,033,195
50334 :208,120,169,015,141,020,063
50340 :003,169,198,141,021,003,187
50346 :169,001,141,026,208,169,116
50352 :027,141,017,208,169,050,020
50358 :141,018,208,169,127,141,218
50364 :013,220,169,255,141,027,245
50370 :208,169,000,141,075,204,223
50376 :169,031,141,021,208,169,171
50382 :255,141,029,208,141,023,235
50388 :208,169,016,141,016,208,202
50394 :162,000,138,010,168,010,194
50400 :010,010,010,010,024,105,137
50406 :029,153,000,208,169,009,030
50412 :157,039,208,232,224,005,077
50418 :208,232,088,162,019,160,087
50424 :000,189,248,203,041,003,164
50430 :029,012,204,240,010,169,150
50436 :241,153,101,198,169,000,098
50442 :076,086,197,189,052,204,046
50448 :041,064,208,044,189,052,102
50454 :204,208,010,169,242,153,240
```

```
50460 :101,198,169,000,076,086,146
50466 :197,142,072,204,189,052,122
50472 :204,162,000,074,176,004,148
50478 :232,076,043,197,189,030,045
50484 :199,153,101,198,174,072,181
50490 :204,169,005,076,086,197,027
50496 :189,032,204,240,010,169,140
50502 :240,153,101,198,169,010,173
50508 :076,086,197,169,236,153,225
50514 :101,198,169,200,024,109,115
50520 :239,203,141,239,203,173,006
50526 :240,203,105,000,141,240,255
50532 :203,202,200,192,020,208,101
50538 :142,032,121,198,173,239,243
50544 :203,013,240,203,240,029,016
50550 :056,173,239,203,237,242,244
50556 :203,141,239,203,173,240,043
50562 :203,233,000,141,240,203,126
50568 :173,076,204,240,006,014,081
50574 :239,203,046,240,203,162,211
50580 :024,160,000,024,032,240,116
50586 :255,169,236,160,197,032,179
50592 :030,171,174,239,203,173,126
50598 :240,203,032,205,189,169,180
50604 :048,032,210,255,173,243,109
50610 :203,208,005,169,048,032,075
50616 :210,255,162,120,032,127,066
50622 :196,202,208,250,173,000,195
50628 :220,041,016,240,249,173,111
50634 :000,220,041,016,208,249,168
50640 :120,169,000,141,026,208,104
50646 :169,255,141,013,220,169,157
50652 :049,141,020,003,169,234,068
50658 :141,021,003,088,162,255,128
50664 :154,076,000,192,159,208,253
50670 :076,065,089,032,065,071,124
50676 :065,073,078,058,211,078,039
50682 :065,080,032,083,072,085,155
50688 :084,084,069,082,032,154,249
50694 :211,067,079,082,069,058,060
50700 :158,018,000,173,018,208,075
50706 :024,105,002,141,001,208,243
50712 :141,003,208,141,005,208,218
50718 :141,007,208,141,009,208,232
50724 :141,011,208,174,075,204,081
50730 :189,097,198,168,162,000,088
50736 :185,101,198,157,248,007,176
50742 :200,232,224,005,208,244,143
50748 :238,075,204,173,075,204,005
```

```
50754 :201,004,208,005,169,000,141
50760 :141,075,204,170,189,093,176
50766 :198,141,018,208,169,001,045
50772 :141,025,208,104,168,104,066
50778 :170,104,064,049,097,146,208
50784 :194,000,005,010,015,224,032
50790 :225,226,227,228,228,227,183
50796 :225,226,228,225,227,226,185
50802 :228,224,225,227,225,224,187
50808 :229,162,000,169,007,157,076
50814 :000,216,157,000,217,157,105
50820 :000,218,157,000,219,232,190
50826 :208,241,169,000,133,253,118
50832 :169,004,141,073,204,133,100
50838 :254,169,005,141,072,204,227
50844 :160,000,169,112,145,253,227
50850 :200,162,006,169,064,145,140
50856 :253,200,202,208,250,169,170
50862 :110,145,253,200,206,072,136
50868 :204,173,072,204,208,228,245
50874 :162,004,032,008,199,160,239
50880 :000,024,169,093,145,253,108
50886 :152,105,007,168,169,093,124
50892 :145,253,200,192,040,144,154
50898 :239,202,208,230,032,008,105
50904 :199,169,005,141,072,204,238
50910 :160,000,169,109,145,253,034
50916 :200,162,006,169,064,145,206
50922 :253,200,202,208,250,169,236
50928 :125,145,253,200,206,072,217
50934 :204,173,072,204,208,228,055
50940 :032,008,199,206,073,204,206
50946 :173,073,204,208,144,096,132
50952 :165,253,024,105,040,133,216
50958 :253,165,254,105,000,133,156
50964 :254,096,010,014,009,009,156
50970 :001,001,007,009,224,225,237
50976 :228,228,232,232,236,228,136
50982 :000,000,000,000,255,255,036
50988 :255,000,000,000,000,000,043
50994 :000,000,000,255,255,255,047
51000 :000,001,001,001,000,255,058
51006 :001,000,000,255,001,000,063
51012 :000,255,001,000,000,000,068
51018 :000,000,000,000,000,000,074
51024 :000,128,000,000,128,000,080
51030 :002,160,000,000,128,000,120
51036 :000,128,000,000,000,000,220
51042 :000,000,000,000,000,000,098
```

```
51048 :000,000,000,000,000,000,104
51054 :000,000,000,000,000,000,110
51060 :000,000,000,000,000,000,116
51066 :000,000,000,000,000,000,122
51072 :000,000,000,000,000,000,128
51078 :147,170,170,128,128,000,109
51084 :128,128,000,128,128,000,140
51090 :128,128,000,128,128,000,146
51096 :128,128,000,128,128,000,152
51102 :128,128,000,128,128,000,158
51108 :128,128,000,128,170,170,120
51114 :128,000,000,000,000,000,042
51120 :000,000,000,000,000,000,176
51126 :000,000,000,000,000,000,182
51132 :000,000,000,000,000,000,188
51138 :000,000,000,000,147,000,085
51144 :000,000,000,000,000,000,200
51150 :000,000,000,000,000,000,206
51156 :000,000,000,008,000,000,220
51162 :008,000,000,008,000,000,234
51168 :170,128,000,008,000,000,018
51174 :008,000,000,008,000,000,246
51180 :000,000,000,000,000,000,236
51186 :000,000,000,000,000,000,242
51192 :000,000,000,000,000,000,248
51198 :000,000,000,000,000,000,254
51204 :000,000,048,042,170,170,178
51210 :032,000,002,032,000,002,078
51216 :032,000,002,032,000,002,084
51222 :032,000,002,032,000,002,090
51228 :032,000,002,032,000,002,096
51234 :032,000,002,032,000,002,102
51240 :032,000,002,032,000,002,108
51246 :032,000,002,032,000,002,114
51252 :032,000,002,042,170,170,212
51258 :000,000,000,000,000,000,058
51264 :000,000,000,000,000,000,064
51270 :048,000,000,000,000,000,118
51276 :000,000,000,000,000,000,076
51282 :000,000,000,000,000,000,082
51288 :000,000,000,000,000,000,088
51294 :000,000,000,000,080,170,088
51300 :000,118,170,160,086,174,040
51306 :168,022,234,186,006,174,128
51312 :170,011,170,233,010,170,108
51318 :149,005,064,020,001,000,101
51324 :080,000,064,064,000,000,076
51330 :000,000,000,000,233,000,107
51336 :000,000,000,000,000,000,136
```

```
51342 :000,000,000,000,000,000,142
51348 :000,000,000,000,000,000,148
51354 :000,000,000,000,000,000,154
51360 :000,000,000,170,000,082,156
51366 :170,160,118,174,168,086,018
51372 :234,186,022,174,170,011,201
51378 :170,233,010,170,148,001,142
51384 :064,020,000,080,005,000,097
51390 :016,000,000,000,000,000,206
51396 :000,000,233,000,000,000,173
51402 :000,000,000,000,000,000,202
51408 :000,000,000,000,000,000,208
51414 :000,000,000,000,000,000,214
51420 :000,000,000,000,000,000,220
51426 :000,170,005,010,170,157,226
51432 :042,186,149,174,171,148,078
51438 :170,186,144,107,170,224,215
51444 :086,170,160,020,001,080,249
51450 :005,000,064,001,001,000,065
51456 :000,000,000,000,000,000,000
51462 :233,000,000,000,000,000,239
51468 :000,000,000,000,000,000,012
51474 :000,000,000,000,000,000,018
51480 :000,000,000,000,000,000,024
51486 :000,000,000,000,000,170,200
51492 :000,010,170,133,042,186,065
51498 :157,174,171,149,170,186,025
51504 :148,107,170,224,022,170,121
51510 :160,020,001,064,080,005,128
51516 :000,000,004,000,000,000,064
51522 :000,000,000,000,022,000,088
51528 :000,000,000,000,000,000,072
51534 :000,000,000,000,000,000,078
51540 :000,000,016,000,000,116,216
51546 :000,000,020,000,000,005,115
51552 :000,000,001,000,000,001,098
51558 :068,064,001,149,144,001,017
51564 :089,080,000,068,100,000,189
51570 :000,020,000,000,005,000,139
51576 :000,000,000,000,000,000,120
51582 :000,000,000,000,000,000,126
51588 :000,000,000,000,000,000,132
51594 :000,000,000,000,000,000,138
51600 :000,000,000,000,000,000,144
51606 :000,000,000,000,000,000,150
51612 :000,000,000,000,000,000,156
51618 :016,000,000,116,145,001,184
51624 :021,086,085,009,101,148,106
51630 :000,017,064,000,000,000,255
```

```
51636 :000,000,000,000,000,000,180
51642 :000,000,000,000,000,000,186
51648 :000,000,000,000,000,000,192
51654 :000,000,000,000,000,000,198
51660 :000,000,000,000,000,000,204
51666 :000,000,000,000,000,000,210
51672 :004,000,000,029,000,000,249
51678 :020,000,000,080,000,000,066
51684 :064,001,017,128,005,149,080
51690 :064,009,089,064,021,017,242
51696 :000,036,000,000,080,000,100
51702 :000,000,000,000,000,000,246
51708 :000,000,000,000,000,000,252
51714 :000,000,000,000,000,000,002
51720 :000,000,000,000,000,000,008
51726 :000,000,000,000,000,000,014
51732 :000,000,000,000,000,000,020
51738 :000,000,000,000,000,000,026
51744 :000,000,000,000,004,064,100
51750 :069,029,101,101,084,022,188
51756 :086,064,001,068,000,000,007
51762 :000,000,000,000,000,000,050
51768 :000,000,000,000,000,000,056
51774 :000,000,000,000,000,000,062
51780 :000,000,000,000,000,000,068
51786 :000,000,000,000,000,000,074
51792 :000,000,000,000,000,000,080
51798 :020,000,000,093,000,000,199
51804 :037,000,000,005,000,000,134
51810 :005,064,000,005,068,064,048
51816 :001,085,080,002,086,080,182
51822 :005,101,084,005,085,149,027
51828 :001,085,080,001,064,080,171
51834 :000,080,020,000,000,000,222
51840 :000,000,000,000,000,000,128
51846 :255,000,000,000,000,000,133
51852 :000,000,000,000,000,000,140
51858 :000,020,000,000,093,000,003
51864 :000,037,000,000,005,000,194
51870 :000,005,000,000,005,064,232
51876 :000,001,068,064,001,085,127
51882 :080,002,086,084,005,101,016
51888 :085,005,085,144,001,085,069
51894 :080,001,001,064,001,001,074
51900 :064,000,064,080,000,000,140
51906 :000,000,000,000,255,000,193
51912 :000,000,000,000,000,000,200
51918 :000,000,000,000,000,000,206
51924 :000,000,000,000,020,000,232
```

```
51930 :000,117,000,000,088,000,167
51936 :000,080,000,001,080,001,130
51942 :017,080,005,085,064,005,230
51948 :149,128,021,089,080,086,021
51954 :085,080,005,085,064,005,054
51960 :001,064,020,005,000,000,082
51966 :000,000,000,000,000,000,254
51972 :000,000,000,000,000,000,004
51978 :000,000,000,000,000,000,010
51984 :000,000,000,000,000,020,036
51990 :000,000,117,000,000,088,227
51996 :000,000,080,000,000,080,188
52002 :000,001,080,001,017,064,197
52008 :005,085,064,021,149,128,236
52014 :085,089,080,006,085,080,215
52020 :005,085,064,001,064,064,079
52026 :001,064,064,005,001,000,193
52032 :000,000,000,000,000,000,064
52038 :000,000,000,000,000,000,070
52044 :000,000,000,000,000,000,076
52050 :000,000,000,000,000,000,082
52056 :000,000,000,000,000,000,088
52062 :000,000,000,000,000,000,094
52068 :000,000,000,000,000,000,100
52074 :000,000,000,000,064,000,170
52080 :000,080,000,000,081,016,033
52086 :000,085,084,000,089,100,220
52092 :000,085,085,000,086,089,213
52098 :000,085,084,000,125,000,168
52104 :000,000,000,000,000,000,136
52110 :042,160,000,000,000,000,088
52116 :000,000,255,255,255,000,145
52122 :170,165,015,255,255,000,246
52128 :000,080,042,129,064,001,220
52134 :017,080,005,090,160,240,246
52140 :063,255,021,089,080,086,254
52146 :170,170,005,085,064,255,159
52152 :255,064,020,005,000,170,186
52158 :040,000,000,000,000,000,230
52164 :000,000,000,013,013,013,235
```

Appendices

A Beginner's Guide To Typing In Programs

What Is a Program?

A computer cannot perform any task by itself. Like a car without gas, a computer has *potential*, but without a program, it isn't going anywhere. Most of the programs published in this book are written in a computer language called BASIC. BASIC is easy to learn and is built into all Commodore 64s.

BASIC Programs

Computers can be picky. Unlike the English language, which is full of ambiguities, BASIC usually has only one right way of stating something. Every letter, character, or number is significant. A common mistake is substituting a letter such as O for the numeral 0, a lowercase l for the numeral 1, or an uppercase B for the numeral 8. Also, you must enter all punctuation such as colons and commas just as they appear in the book. Spacing can be important. To be safe, type in the listings exactly as they appear.

Braces and Special Characters

The exception to this typing rule is when you see the braces, such as {DOWN}. Anything within a set of braces is a special character or characters that cannot easily be listed on a printer. When you come across such a special statement, refer to Appendix B, "How to Type In Programs."

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program. Some DATA statements contain actual programs (called machine language); others contain graphics codes. These lines are especially sensitive to errors.

If a single number in any one DATA statement is mistyped, your machine could lock up, or crash. The keyboard and STOP key may seem dead, and the screen may go blank. Don't panic—no damage is done. To regain control, you have to turn off your computer, then turn it back on. This will erase

whatever program was in memory, *so always save a copy of your program before you run it.* If your computer crashes, you can load the program and look for your mistake.

Sometimes a mistyped DATA statement will cause an error message when the program is run. The error message may refer to the program line that READs the data. *The error is still in the DATA statements, though.*

Get to Know Your Machine

You should familiarize yourself with your computer before attempting to type in a program. Learn the statements you use to store and retrieve programs from tape or disk. You'll want to save a copy of your program, so that you won't have to type it in every time you want to use it. Learn to use your machine's editing functions. How do you change a line if you made a mistake? You can always retype the line, but you at least need to know how to backspace. Do you know how to enter reverse video, lowercase, and control characters? It's all explained in your computer's manuals.

A Quick Review

1. Type in the program a line at a time, in order. Press RETURN at the end of each line. Use backspace or the back arrow to correct mistakes.
2. Check the line you've typed against the line in the book. You can check the entire program again if you get an error when you run the program.

How to Type In Programs

To make it easy to know exactly what to type when entering one of these programs into your computer, we have established the following listing conventions.

Generally, Commodore 64 program listings will contain words within braces that spell out any special characters: {DOWN} would mean to press the cursor down key. {5 SPACES} would mean to press the space bar five times.

To indicate that a key should be *shifted* (hold down the SHIFT key while pressing the other key), the key would be underlined in our listings. For example, S would mean to type the S key while holding the SHIFT key. This would appear on your screen as a heart symbol. If you find an underlined key enclosed in braces (e.g., {10 N}), you should type the key as many times as indicated (in our example, you would enter ten shifted N's).

If a key is enclosed in special brackets, [ ] , you should hold down the *Commodore key* while pressing the key inside the special brackets. (The Commodore key is the key in the lower-left corner of the keyboard.) Again, if the key is preceded by a number, you should press the key as many times as necessary.

Rarely, you'll see a solitary letter of the alphabet enclosed in braces. These characters can be entered by holding down the CTRL key while typing the letter in the braces. For example, {A} would indicate that you should press CTRL-A.

About the *quote mode:* You know that you can move the cursor around the screen with the CRSR keys. Sometimes a programmer will want to move the cursor under program control. That's why you see all the {LEFT}'s, {HOME}'s, and {BLU}'s in our programs. The only way the computer can tell the difference between direct and programmed cursor control is the quote mode.

Once you press the quote (the double quote, SHIFT-2), you are in the quote mode. If you type something and then try to change it by moving the cursor left, you'll only get a bunch of reverse-video lines. These are the symbols for cursor left. The only editing key that isn't programmable is the DEL key;

you can still use DEL to back up and edit the line. Once you type another quote, you are out of quote mode.

You also go into quote mode when you INSerT spaces into a line. In any case, the easiest way to get out of quote mode is to just press RETURN. You'll then be out of quote mode and you can cursor up to the mistyped line and fix it.

Use the following table when entering cursor and color control keys:

When You Read:	Press:	See:	When You Read:	Press:	See:
{CLR}	SHIFT CLR/HOME		{ 1 }	COMMODORE 1	
{HOME}	CLR/HOME		{ 2 }	COMMODORE 2	
{UP}	SHIFT ↑ CRSR ↓		{ 3 }	COMMODORE 3	
{DOWN}	↑ CRSR ↓		{ 4 }	COMMODORE 3	
{LEFT}	SHIFT ← CRSR →		{ 5 }	COMMODORE 5	
{RIGHT}	← CRSR →		{ 6 }	COMMODORE 6	
{RVS}	CTRL 9		{ 7 }	COMMODORE 7	
{OFF}	CTRL 0		{ 8 }	COMMODORE 8	
{BLK}	CTRL 1		{ F1 }	f1	
{WHT}	CTRL 2		{ F2 }	SHIFT f1	
{RED}	CTRL 3		{ F3 }	f3	
{CYN}	CTRL 4		{ F4 }	SHIFT f3	
{PUR}	CTRL 5		{ F5 }	f5	
{GRN}	CTRL 6		{ F6 }	SHIFT f5	
{BLU}	CTRL 7		{ F7 }	f7	
{YEL}	CTRL 8		{ F8 }	SHIFT f7	

The Automatic Proofreader

Charles Brannon

"The Automatic Proofreader" will help you type in program listings without typing mistakes. It is a short error-checking program that hides itself in memory. When activated, it lets you know immediately after typing a line from a program listing if you have made a mistake. Please read these instructions carefully before typing any programs in this book.

Preparing the Proofreader

1. Using the listing below, type in the Proofreader. Be very careful when entering the DATA statements—don't type an l instead of a 1, an O instead of a 0, extra commas, etc.
2. Save the Proofreader on tape or disk at least twice *before running it for the first time.* This is very important because the Proofreader erases part of itself when you first type RUN.
3. After the Proofreader is saved, type RUN. It will check itself for typing errors in the DATA statements and warn you if there's a mistake. Correct any errors and save the corrected version. Keep a copy in a safe place—you'll need it again and again, every time you enter a program from this book or from *COMPUTE!'s Gazette* magazine or *COMPUTE!* magazine.
4. When a correct version of the Proofreader is run, it activates itself. You are now ready to enter a program listing. If you press RUN/STOP–RESTORE, the Proofreader is disabled. To reactivate it, just type the command SYS 886 and press RETURN.

Using the Proofreader

All listings in this book have a checksum number appended to the end of each line. An example is ":rem 123". *Don't enter this statement when typing in a program.* It is just for your information. The rem makes the number harmless if someone does type it in. It will, however, use up memory if you enter it, and it will confuse the Proofreader, even if you entered the rest of the line correctly.

When you type in a line from a program listing and press RETURN, the Proofreader displays a number at the top of your screen. *This checksum number must match the checksum number in the printed listing.* If it doesn't, it means you typed the line differently from the way it is listed. Immediately recheck your typing. Remember, don't type the rem statement with the checksum number; it is published only so you can check it against the number which appears on your screen.

The Proofreader is not picky with spaces. It will not notice extra spaces or missing ones. This is for your convenience, since spacing is generally not important. But occasionally proper spacing *is* important, so be extra careful with spaces, since the Proofreader will catch practically everything else that can go wrong.

There's another thing to watch out for: if you enter the line by using abbreviations for commands, the checksum will not match up. But there is a way to make the Proofreader check it. After entering the line, list it. This eliminates the abbreviations. Then move the cursor up to the line and press RETURN. It should now match the checksum. You can check whole groups of lines this way.

Special Tape SAVE Instructions

When you're done typing a listing, you must disable the Proofreader before saving the program on tape. Disable the Proofreader by pressing RUN/STOP–RESTORE (hold down the RUN/STOP key and sharply hit the RESTORE key.) This procedure is not necessary for disk, *but you must disable the Proofreader this way before a tape SAVE.*

A SAVE to tape erases the Proofreader from memory, so you'll have to load and run it again if you want to type another listing. A SAVE to disk does not erase the Proofreader.

Hidden Perils

The Proofreader's home in the 64 is not a very safe haven. Since the cassette buffer is wiped out during tape operations, you need to disable the Proofreader with RUN/STOP–RESTORE before you save your program. This applies only to tape use. Disk users have nothing to worry about.

Not so for 64 owners with tape drives. What if you type in a program in several sittings? The next day, you come to your computer, load and run the Proofreader, then try to load

the partially completed program so you can add to it. But since the Proofreader is trying to hide in the cassette buffer, it is wiped out!

What you need is a way to load the Proofreader after you've loaded the partial program. The problem is, a tape load to the buffer destroys what it's supposed to load.

After you've typed in and run the Proofreader, enter the following lines in direct mode (without line numbers) exactly as shown:

```
A$="PROOFREADER.T":B$="{10 SPACES}":FOR X=1 TO 4:A
  $=A$+B$:NEXTX
FOR X=886 TO 1018:A$=A$+CHR$(PEEK(X)):NEXTX
OPEN1,1,1,A$:CLOSE1
```

After you enter the last line, you will be asked to press RECORD and PLAY on your cassette recorder. Put this program at the beginning of a new tape. This gives you a new way to load the Proofreader. Anytime you want to bring the Proofreader into memory without disturbing anything else, put the cassette in the tape drive, rewind, and enter:

```
OPEN1:CLOSE1
```

You can now start the Proofreader by typing SYS 886. To test this, PRINT PEEK (886) should return the number 173. If it does not, repeat the steps above, making sure that A$ ("PROOFREADER.T") contains 13 characters and that B$ contains 10 spaces.

The Proofreader will load itself into the cassette buffer whenever you type OPEN1:CLOSE1—and PROOFREADER.T is the next program on your tape. It does not disturb the contents of BASIC memory.

Replace Original Proofreader

If you typed in the original version of the Proofreader from the October 1983 issue of *COMPUTE!'s Gazette* magazine, you should replace it with the improved version below.

Automatic Proofreader

```
100 PRINT"{CLR}PLEASE WAIT...":FORI=886TO1018:READ
    A:CK=CK+A:POKEI,A:NEXT
110 IF CK<>17539 THEN PRINT"{DOWN}YOU MADE AN ERRO
    R":PRINT"IN DATA STATEMENTS.":END
120 SYS886:PRINT"{CLR}{2 DOWN}PROOFREADER ACTIVATE
    D.":NEW
```

```
886 DATA 173,036,003,201,150,208
892 DATA 001,096,141,151,003,173
898 DATA 037,003,141,152,003,169
904 DATA 150,141,036,003,169,003
910 DATA 141,037,003,169,000,133
916 DATA 254,096,032,087,241,133
922 DATA 251,134,252,132,253,008
928 DATA 201,013,240,017,201,032
934 DATA 240,005,024,101,254,133
940 DATA 254,165,251,166,252,164
946 DATA 253,040,096,169,013,032
952 DATA 210,255,165,214,141,251
958 DATA 003,206,251,003,169,000
964 DATA 133,216,169,019,032,210
970 DATA 255,169,018,032,210,255
976 DATA 169,058,032,210,255,166
982 DATA 254,169,000,133,254,172
988 DATA 151,003,192,087,208,006
994 DATA 032,205,189,076,235,003
1000 DATA 032,205,221,169,032,032
1006 DATA 210,255,032,210,255,173
1012 DATA 251,003,133,214,076,173
1018 DATA 003
```

Using the Machine Language Editor: MLX

Charles Brannon

Remember the last time you typed in the BASIC loader for a long machine language program? You typed in hundreds of numbers and commas. Even then, you couldn't be sure if you typed it in right. So you went back, proofread, tried to run the program, crashed, went back and proofread again, corrected a few typing errors, ran again, crashed again, rechecked your typing.... Frustrating, wasn't it?

Until now, though, that has been the best way to get machine language into your computer. Unless you happen to have an assembler and are willing to tangle with machine language on the assembly level, it is much easier to enter a BASIC program that reads DATA statements and POKEs the numbers into memory.

Some of these "BASIC loaders" use a checksum to see if you've typed the numbers correctly. The simplest checksum is just the sum of all the numbers in the DATA statements. If you make an error, your checksum does not match up with the total. Some programmers make your task easier by including checksums every few lines, so you can locate your errors more easily.

Now, MLX comes to the rescue. MLX is a great way to enter all those long machine language programs with a minimum of fuss. MLX lets you enter the numbers from a special list that looks similar to DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255 (forbidden in ML). It will prevent you from entering the numbers on the wrong line. In short, MLX makes proofreading obsolete.

Tape or Disk Copies

In addition, MLX generates a ready-to-use copy of your machine language program on tape or disk. You can then use the LOAD command to read the program into the computer, as with any other program. Specifically, you enter:

LOAD "*program name*",1,1 (for tape)

or

LOAD "*program name*",8,1 (for disk)

To start the program, you need to enter a SYS command that tranfers control from BASIC to your machine language program. The starting SYS is always listed in the article which presents the machine language program in MLX format.

Using MLX

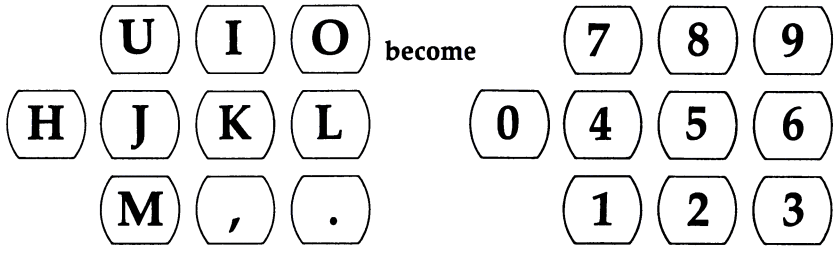
Type in and save MLX (you'll want to use it in the future). When you're ready to type in the machine language program, run MLX. MLX asks you for two numbers: the starting address and the ending address. These numbers are given in the article accompanying the ML program you're typing. For example, "Astro-PANIC!'s" addressses should be 49152 and 50777 respectively.

You'll see a prompt. The prompt is the current line you are entering from the MLX format listing. It increases by six each time you enter a line. That's because each line has seven numbers—six actual data numbers plus a checksum number. The checksum verifies that you typed the previous six numbers correctly. If you enter any of the six numbers wrong or enter the checksum wrong, the 64 sounds a buzzer and prompts you to reenter the line. If you enter the line correctly, a bell tone sounds and you continue to the next line.

A Special Editor

You are not using the normal 64 BASIC editor with MLX. For example, it will only accept numbers as input. If you make a typing error, press the INST/DEL key; the entire number is deleted. You can press it as many times as necessary, back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes on to accept the next number. If you enter less than three digits, you can press either the space bar or RETURN key to advance to the next number. The checksum automatically appears in reverse video for emphasis.

To make it even easier to enter these numbers, MLX redefines part of the keyboard as a numeric keypad (lines 581–584).



When testing it, I've found MLX to be an extremely easy way to enter long listings. With the audio cues provided, you don't even have to look at the screen if you're a touch-typist.

Done at Last!

When you get through typing, assuming you type your machine language program all in one session, you can then save the completed and bug-free program to tape or disk. Follow the instructions displayed on the screen. If you get any error messages while saving, you probably have a bad disk, or the disk is full, or you made a typo when entering the MLX program. (Sorry, MLX can't check itself!)

Command Control

You don't have to enter the whole ML program in one sitting. MLX lets you enter as much as you want, save it, and then reload the file from tape or disk later. MLX recognizes these commands:

SHIFT-S:Save
SHIFT-L:Load
SHIFT-N:New Address
SHIFT-D:Display

Hold down SHIFT while you press the appropriate key. MLX jumps out of the line you've been typing, so I recommend you do it at a prompt. Use the Save command to store what you've been working on. It will save on tape or disk as if you've finished, but the tape or disk won't work, of course, until you finish typing. Remember what address you stopped on. The next time you run MLX, answer all the prompts as you did before, then insert the disk or tape containing the stored file. When you get the entry prompt, press SHIFT-L to

reload the partly completed file into memory. Then use the New Address command (SHIFT-N) to resume typing.

New Address and Display

After you press SHIFT-N, enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksums won't match up. You can use the Display command to display a section of your typing. After you press SHIFT-D, enter two addresses within the line number range of the listing. You can abort the listing by pressing any key.

Tricky Stuff

The special commands may seem a little confusing, but as you work with MLX, they will become valuable. For example, what if you forgot where you stopped typing? Use the Display command to scan memory from the beginning to the end of the program. When you reach the end of your typing, the lines will contain a random pattern of numbers, quite different from what should be there. When you see the end of your typing, press any key to stop the listing. Use the New Address command to continue typing from the proper location.

You can use the Save and Load commands to make copies of the complete machine language program. Use the Load command to reload the tape or disk, then insert a new tape or disk and use the Save command to create a new copy. When resaving on disk it is best to use a different filename each time you save. For example, I like to number my work and use filenames such as ASTRO1, ASTRO2, ASTRO3, and so on.

One quirk about tapes made with the MLX Save command: when you load them, the message "FOUND program" may appear twice. The tape will load just fine, however.

I think you'll find MLX to be a true labor-saving program. Since it has been tested by entering actual programs, you can count on it as an aid for generating bug-free machine language. Be sure to save MLX; it will be used for future applications in COMPUTE! Books, *COMPUTE!* magazine, and *COMPUTE!'s Gazette* magazine.

Machine Language Editor: MLX

For mistake-proof program entry, be sure to read "The Automatic Proofreader," Appendix C.

```
100 PRINT"{CLR}{6}";CHR$(142);CHR$(8);:POKE53281,1
    :POKE53280,1                                  :rem 67
101 POKE 788,52:REM DISABLE RUN/STOP              :rem 119
110 PRINT"{RVS}{39 SPACES}";                      :rem 176
120 PRINT"{RVS}{14 SPACES}{RIGHT}{OFF}{*}£{RVS}
    {RIGHT} {RIGHT}{2 SPACES}{*}{OFF}{*}£{RVS}£
    {RVS}{14 SPACES}";                            :rem 250
130 PRINT"{RVS}{14 SPACES}{RIGHT} {G}{RIGHT}
    {2 RIGHT} {OFF}£{RVS}£{*}{OFF}{*}{RVS}
    {14 SPACES}";                                 :rem 35
140 PRINT"{RVS}{41 SPACES}"                       :rem 120
200 PRINT"{2 DOWN}{PUR}{BLK}{9 SPACES}MACHINE LANG
    UAGE EDITOR{5 DOWN}"                           :rem 6
210 PRINT"{5}{2 UP}STARTING ADDRESS?{8 SPACES}
    {9 LEFT}";                                    :rem 143
215 INPUTS:F=1-F:C$=CHR$(31+119*F)                :rem 166
220 IFS<256OR(S>40960ANDS<49152)ORS>53247THENGOSUB
    3000:GOTO210                                  :rem 235
225 PRINT:PRINT:PRINT                             :rem 180
230 PRINT"{5}{2 UP}ENDING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTE:F=1-F:C$=CHR$(31+119*F)
                                                  :rem 20
240 IFE<256OR(E>40960ANDE<49152)ORE>53247THENGOSUB
    3000:GOTO230                                  :rem 183
250 IFE<STHENPRINTC$;"{RVS}ENDING < START
    {2 SPACES}":GOSUB1000:GOTO 230                :rem 176
260 PRINT:PRINT:PRINT                             :rem 179
300 PRINT"{CLR}";CHR$(14):AD=S:POKEV+21,0 :rem 225
310 A=1:PRINTRIGHT$("0000"+MID$(STR$(AD),2),5);":"
    ;                                             :rem 33
315 FORJ=ATO6                                     :rem 33
320 GOSUB570:IFN=-1THENJ=J+N:GOTO320              :rem 228
390 IFN=-211THEN 710                               :rem 62
400 IFN=-204THEN 790                               :rem 64
410 IFN=-206THENPRINT:INPUT"{DOWN}ENTER NEW ADDRES
    S";ZZ                                         :rem 44
415 IFN=-206THENIFZZ<SORZZ>ETHENPRINT"{RVS}OUT OF
    {SPACE}RANGE":GOSUB1000:GOTO410               :rem 225
417 IFN=-206THENAD=ZZ:PRINT:GOTO310               :rem 238
420 IF N<>-196 THEN 480                           :rem 133
430 PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO";:INPUTT
                                                  :rem 234
440 IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAST";S;"
    {LEFT}, NOT MORE THAN";E:GOTO430              :rem 159
450 FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("0000"+MID$(S
    TR$(I),2),5);":";                             :rem 30
```

```
451 FORK=ØTO5:N=PEEK(I+K):PRINTRIGHT$("ØØ"+MID$(ST
    R$(N),2),3);",";                          :rem 66
460 GETA$:IFA$>""THENPRINT:PRINT:GOTO31Ø      :rem 25
470 NEXTK:PRINTCHR$(2Ø);:NEXTI:PRINT:PRINT:GOTO31Ø
                                              :rem 5Ø
480 IFN<Ø THEN PRINT:GOTO31Ø                  :rem 168
490 A(J)=N:NEXTJ                              :rem 199
500 CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT                        :rem 2ØØ
510 PRINTCHR$(18);:GOSUB57Ø:PRINTCHR$(146);:rem 94
511 IFN=-1THENA=6:GOTO315                     :rem 254
515 PRINTCHR$(2Ø):IFN=CKSUMTHEN53Ø            :rem 122
520 PRINT:PRINT"LINE ENTERED WRONG : RE-ENTER":PRI
    NT:GOSUB1ØØØ:GOTO31Ø                     :rem 176
530 GOSUB2ØØØ                                 :rem 218
540 FORI=1TO6:POKEAD+I-1,A(I):NEXT:POKE54272,Ø:POK
    E54273,Ø                                  :rem 227
550 AD=AD+6:IF AD<E THEN 31Ø                  :rem 212
560 GOTO 71Ø                                  :rem 1Ø8
570 N=Ø:Z=Ø                                    :rem 88
580 PRINT"[£]";                                :rem 81
581 GETA$:IFA$=""THEN581                       :rem 95
582 AV=-(A$="M")-2*(A$=",")-3*(A$=".")-4*(A$="J")-
    5*(A$="K")-6*(A$="L")                     :rem 41
583 AV=AV-7*(A$="U")-8*(A$="I")-9*(A$="O"):IFA$="H
    "THENA$="Ø"                               :rem 134
584 IFAV>ØTHENA$=CHR$(48+AV)                  :rem 134
585 PRINTCHR$(2Ø);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N67Ø                                      :rem 229
590 IFA>128THENN=-A:RETURN                    :rem 137
600 IFA<>2Ø THEN 63Ø                           :rem 1Ø
610 GOSUB69Ø:IFI=1ANDT=44THENN=-1:PRINT"{OFF}
    {LEFT} {LEFT}";:GOTO69Ø                    :rem 62
620 GOTO57Ø                                   :rem 1Ø9
630 IFA<48ORA>57THEN58Ø                       :rem 1Ø5
640 PRINTA$;:N=N*1Ø+A-48                      :rem 1Ø6
650 IFN>255 THEN A=2Ø:GOSUB1ØØØ:GOTO6ØØ      :rem 229
660 Z=Z+1:IFZ<3THEN58Ø                         :rem 71
670 IFZ=ØTHENGOSUB1ØØØ:GOTO57Ø                :rem 114
680 PRINT",";:RETURN                          :rem 240
690 S%=PEEK(2Ø9)+256*PEEK(21Ø)+PEEK(211)      :rem 149
691 FORI=1TO3:T=PEEK(S%-I)                     :rem 67
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT       :rem 2Ø5
7ØØ PRINTLEFT$("{3 LEFT}",I-1);:RETURN         :rem 7
71Ø PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}"     :rem 236
```

```
715 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL SAVE){DOWN}"                          :rem 106
720 F$="":INPUT"{DOWN} FILENAME";F$:IFF$=""THENPRI
    NT:PRINT:GOTO310                              :rem 71
730 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                              :rem 228
740 GETA$:IFA$<>"T"ANDA$<>"D"THEN740              :rem 36
750 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$:OPEN15,8,
    15,"S"+F$:CLOSE15                             :rem 212
760 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                       :rem 3
762 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                           :rem 109
763 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 69
765 K=S:POKE254,K/256:POKE253,K-PEEK(254)*256:POKE
    780,253                                       :rem 17
766 K=E+1:POKE782,K/256:POKE781,K-PEEK(782)*256:SY
    S65496                                        :rem 235
770 IF(PEEK(783)AND1)OR(191ANDST)THEN780          :rem 111
775 PRINT"{DOWN}DONE.{DOWN}":GOTO310              :rem 113
780 PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN720                               :rem 171
781 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO720                                   :rem 103
790 PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}"         :rem 212
795 PRINT"{2 DOWN}(PRESS {RVS}RETURN{OFF} ALONE TO
     CANCEL LOAD)"                                :rem 82
800 F$="":INPUT"{2 DOWN} FILENAME";F$:IFF$=""THENP
    RINT:GOTO310                                  :rem 144
810 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                              :rem 227
820 GETA$:IFA$<>"T"ANDA$<>"D"THEN820              :rem 34
830 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$          :rem 157
840 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                       :rem 2
841 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                           :rem 107
845 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 70
850 POKE780,0:SYS65493                            :rem 11
860 IF(PEEK(783)AND1)OR(191ANDST)THEN870          :rem 111
865 PRINT"{DOWN}DONE.":GOTO310                    :rem 96
870 PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}TRY AGAIN.
    {DOWN}":IFDV=1THEN800                         :rem 172
880 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO800                                   :rem 102
```

```
1000 REM BUZZER                                    :rem 135
1001 POKE54296,15:POKE54277,45:POKE54278,165
                                                   :rem 207
1002 POKE54276,33:POKE 54273,6:POKE54272,5 :rem 42
1003 FORT=1TO200:NEXT:POKE54276,32:POKE54273,0:POK
     E54272,0:RETURN                               :rem 202
2000 REM BELL SOUND                                 :rem 78
2001 POKE54296,15:POKE54277,0:POKE54278,247
                                                   :rem 152
2002 POKE 54276,17:POKE54273,40:POKE54272,0:rem 86
2003 FORT=1TO100:NEXT:POKE54276,16:RETURN  :rem 57
3000 PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":GOTO1000
                                                   :rem 89
```

Index

access array (text adventure) 29
action routines (text adventure) 36–40
ADSR envelope 148–49
animation 14, 140–42, 173–75
arrays 175
  text adventure 29–31
Asteroids 5
"Astro-PANIC!" 232–39
attack (sound) 148
"Automatic Proofreader, The" 261–64
automatic routines (text adventure) 34–35
balloon collisions 12
bandpass filter 147
BASIC 15
  combined with machine language 124
"Beekeeper" 133–38
billiard cushion collisions 12
Blockade 223
bomb collisions 12
Breakout 223
"Brunhilde Loves Bruno" 171–89
"Burn Rubber" 204–10
"Campaign Manager" 4, 82–119
  command summary 92–93
  formulae 88, 89, 91
  strategy 88–93
  typing in 84
cassette buffer 141
characters, programmed. *See* custom characters
collision detection 141
collisions 11–12, 141, 177–78, 190–91
Commodore 64 Programmer's Reference Guide 141
communication, game to player 13
copying character set 172
  assembly listing 188–89
custom characters 141–42, 172, 174–75, 213
"CUT-OFF!" 223–31
cutoff frequency 147
decay (sound) 148
delay loop 143
delete character 65
Donkey Kong 5, 14
flags, machine language 146
"Frantic Fisherman, The" 123–32
game design 3–21, 45–46
  in text adventure 25–43
hardware interrupt, modifying 144–45
"Haunted Mansion" 211–20
ideas 4–5
IF/THEN statement 46
improvisation 22
item description array (text adventure) 31
item flag array (text adventure) 31
item location array (text adventure) 31
jiffy 66
Joust 4, 14
joystick, reading 141–42
Jump Man 5
levels of play 14
line numbering 1
Lode Runner 5
Lunar Lander 223
machine language 15, 83, 139–49
  combined with BASIC 124
  games 221–53
main loop 15–16
  in text adventures 32–34
mapmaking (text adventure) 26–27
Mario Bros. 4
maze generation 212
milieu 11
ML. *See* machine language
"MLX" 83–84, 265–72
"Nessie" 240–54
object array (text adventure) 30
object token array (text adventure) 31
"Olympiad" 197–203
ON/GOTO statement 37, 46
Oregon Trail 63
parsing (text adventure) 35–36, 47–48
pitch 148
play mechanic 4, 5–11
Pong 4, 223
PRINT statement 15
"Props" 139–58
punishment 12–13
"Quatrainment" 76–81
quests 41–42
quote mode 259–60
raster interrupt 243
registers, sound 147–49
release (sound) 149
RESTORE statement 176
reward 12–13
ring modulation 176
room (text adventure) 25–26
room description array (text adventure) 30
room flag array (text adventure) 30
screen memory 140, 176
"Sea Route to India" 63–75
SID chip 149
simplicity, advantages of 7
simulation 11, 63–64, 94
sound 147–49

sound-effects 147–49, 175–76
Space Invaders 5, 223
Space Panic 5
sprites 124, 143–46, 172–78, 190–91, 243
  collision 177–78, 190–91
  pointer 173–74
  shape 146
story 5–6, 26
strategy games 61–119
subroutines 16–21
"Supersprite" 190–96
surround 223
sustain (sound) 149
tar baby collisions 11
text adventure v, 23–59
"Time Capsule" v, 44–59
  transparent collisions 11
treasure hunt 41–42
Tron 223
typing in programs 259–60
variables 21, 29, 32, 135
verb token array (text adventure) 30–31
verbs (text adventure) 28, 37–41
video memory, mapping 15
voices 147–49
wall collisions 11
Westward Ho 63
win-lose conditions 13–14
"Worm of Bemer" 159–67

Notes

Notes

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!** Magazine. Use this form to order your subscription to **COMPUTE!.**

For Fastest Service,
Call Our **Toll-Free** US Order Line

800-334-0868

In NC call 919-275-9809

COMPUTE!
P.O. Box 5406
Greensboro, NC 27403

My Computer Is:
☐ Commodore 64 ☐ TI-99/4A ☐ Timex/Sinclair ☐ VIC-20 ☐ PET
☐ Radio Shack Color Computer ☐ Apple ☐ Atari ☐ Other _____
☐ Don't yet have one...

☐ $24 One Year US Subscription
☐ $45 Two Year US Subscription
☐ $65 Three Year US Subscription

Subscription rates outside the US:

☐ $30 Canada
☐ $42 Europe, Australia, New Zealand/Air Delivery
☐ $52 Middle East, North Africa, Central America/Air Mail
☐ $72 Elsewhere/Air Mail
☐ $30 International Surface Mail (lengthy, unreliable delivery)

Name _____

Address _____

City _____ State _____ Zip _____

Country _____

Payment must be in US Funds drawn on a US Bank; International Money Order, or charge card.

☐ Payment Enclosed ☐ VISA
☐ MasterCard ☐ American Express
Acct. No. _____ Expires ___/___

COMPUTE! Books

P.O. Box 5406 Greensboro, NC 27403

Ask your retailer for these **COMPUTE! Books**. If he or she has sold out, order directly from **COMPUTE!**

For Fastest Service
Call Our **TOLL FREE US Order Line**

800-334-0868

In NC call 919-275-9809

Quantity	Title	Price	Total
_____	Machine Language for Beginners	**$14.95***	_____
_____	Home Energy Applications	**$14.95***	_____
_____	COMPUTE!'s First Book of VIC	**$12.95***	_____
_____	COMPUTE!'s Second Book of VIC	**$12.95***	_____
_____	COMPUTE!'s First Book of VIC Games	**$12.95***	_____
_____	COMPUTE!'s First Book of 64	**$12.95***	_____
_____	COMPUTE!'s First Book of Atari	**$12.95***	_____
_____	COMPUTE!'s Second Book of Atari	**$12.95***	_____
_____	COMPUTE!'s First Book of Atari Graphics	**$12.95***	_____
_____	COMPUTE!'s First Book of Atari Games	**$12.95***	_____
_____	Mapping The Atari	**$14.95***	_____
_____	Inside Atari DOS	**$19.95***	_____
_____	The Atari BASIC Sourcebook	**$12.95***	_____
_____	Programmer's Reference Guide for TI-99/4A	**$14.95***	_____
_____	COMPUTE!'s First Book of TI Games	**$12.95***	_____
_____	Every Kid's First Book of Robots and Computers	**$ 4.95†**	_____
_____	The Beginner's Guide to Buying A Personal Computer	**$ 3.95†**	_____

* Add $2 shipping and handling. Outside US add $5 air mail; $2 surface mail.

† Add $1 shipping and handling. Outside US add $5 air mail; $2 surface mail.

Please add shipping and handling for each book ordered. _____

Total enclosed or to be charged. _____

All orders must be prepaid (money order, check, or charge). All payments must be in US funds. NC residents add 4% sales tax.

☐ Payment enclosed Please charge my: ☐ VISA ☐ MasterCard
☐ American Express Acc't. No. _____ Expires ___/___

Name _____

Address _____

City _____ State _____ Zip _____

Country _____

Allow 4-5 weeks for delivery.

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!'s Gazette** for Commodore.

For Fastest Service
Call Our **Toll-Free** US Order Line
800-334-0868
In NC call 919-275-9809

COMPUTE!'s GAZETTE
P.O. Box 5406
Greensboro, NC 27403

My computer is:
☐ Commodore 64 (01) ☐ VIC-20 (02) ☐ Other (03) _____

☐ $20 One Year US Subscription
☐ $36 Two Year US Subscription
☐ $54 Three Year US Subscription

Subscription rates outside the US:

☐ $25 Canada
☐ $45 Air Mail Delivery
☐ $25 International Surface Mail

Name _____

Address _____

City _____ State _____ Zip _____

Country _____

Payment must be in US Funds drawn on a US Bank, International Money Order, or charge card. Your subscription will begin with the next available issue. Please allow 4–6 weeks for delivery of first issue. Subscription prices subject to change at any time.

☐ Payment Enclosed ☐ VISA
☐ MasterCard ☐ American Express

Acct. No. _____ Expires ____/____

The *COMPUTE!'s Gazette* subscriber list is made available to carefully screened organizations with a product or service which may be of interest to our readers. If you prefer not to receive such mailings, please check this box ☐.

Worlds Inside Your 64

COMPUTE!'s Second Book of Commodore 64 Games gives you 16 new worlds to explore. From photographing the Loch Ness monster to running a presidential campaign, these games put you in unique worlds where your strategy, skill, and knowledge are tested to the limit.

You'll find that each of the games is of the same high quality you've come to expect from COMPUTE! Publications. Some of the best games from *COMPUTE!* magazine and *COMPUTE!'s Gazette* are collected here, along with several others which have never before appeared in print. All are ready to type in and play.

Among the worlds you'll find in this book are:

- "Campaign Manager," where you're running on the Democratic or Republican ticket, hoping to become president.
- "Brunhilde Loves Bruno," a dazzling game inhabited by goblins, Valkyrie music, and lovers struggling to reunite.
- "Olympiad," a two-player game where knights fight a life or death battle in a labyrinth.
- "Time Capsule," a text adventure game that tests your courage as you try to escape a dangerous prison and return to your own time.
- "Astro-PANIC!," with flitting aliens that try to destroy your base.
- "The Frantic Fisherman," where charging sharks and deluges of rain threaten your peaceful fishing trip.
- And 10 more games that carry you far into the future or deep into the past.

Other articles show you how to write your own text adventure games and how to design a videogame from start to finish. There are even special-purpose programs included which make error-free program entry painless.

You expect excellent, playable games from COMPUTE! Publications. *COMPUTE!'s Second Book of Commodore 64 Games* is no exception. From beginning to end, this book will give you countless hours of challenging entertainment.

ISBN 0-942386-64-7



This was brought to you

from the archives of

http://retro-commodore.eu