

ARCADE GAMES FOR THE COMMODORE 64

Fanfare House, Inc.

Mary Ann Chapman and David Harper, *Editors*



IMPORTANT NOTICE

Directions for using this diskette are contained in the introduction of the accompanying book.

1. Insert the diskette in your Commodore Disk Drive.
2. After the C64 displays the ready prompt type:
LOAD "NAME OF THE GAME",8
3. After a few seconds the C64 will give you the ready prompt.
4. Type: RUN

You are now ready to play.

WARNING: DO NOT SAVE ANY OTHER PROGRAM ON THIS DISKETTE.

**Arcade Games
for the
Commodore 64**



Arcade Games for the Commodore 64

Fanfare House, Inc.

Edited by:
MaryAnn Chapman and David Harper

CBS Computer Books

HOLT, RINEHART AND WINSTON
New York Chicago San Francisco Philadelphia
Montreal Toronto London Sydney Tokyo
Mexico City Rio de Janeiro Madrid

Trademarks and Copyrights:

CP/M is a registered trademark of Digital Research Incorporated
Radio Shack and TRS-80 are registered trademarks of Tandy, Inc.
WORDSTAR is a registered trademark of MicroPro International Corporation
Commodore 64 is a trademark of Commodore Electronics, Limited

Copyright© 1985 Fanfare House, Inc.

All rights reserved.

Address correspondence to:

383 Madison Avenue, New York, NY 10017

First distributed to the trade in 1985 by Holt, Rinehart and Winston general book division.

Library of Congress Cataloging in Publication Data

FANFARE HOUSE, INC.

Arcade games for the Commodore 64.

1. Computer games. 2. Commodore 64 (Computer)--
Programming. I. Fanfare House, Inc.

GV1469.2.A73 1985 794.8'2 84-17632

ISBN 0-03-001049-7

Printed in the United States of America

Published simultaneously in Canada

5 6 7 039 9 8 7 6 5 4 3 2 1

CBS COLLEGE PUBLISHING

Holt, Rinehart and Winston

The Dryden Press

Saunders College Publishing

Table of Contents

	A	B	C
.....	170	130	170
.....	128	130	2
.....	143	135	242
.....	13	235	112
.....	13	235	112
.....	13	235	112
.....	15	235	240
.....	2	130	128
.....	34	170	136
.....	170	170	170
.....	170	170	170
.....	2	170	128
.....	2	170	128
.....	10	170	160
.....	10	170	160
.....	10	170	160
.....	10	170	160
.....	2	130	128
.....	2	130	128
.....	10	130	160
.....	10	130	160

BANK-# 2 PAGE-# 41

Preface ix

Acknowledgements xi

Introduction xiii

Playing the Games xiii

Information Provided for Each Game xvi

Modifying the Games xx

Try Programming Yourself! xxi

Rocket Lander 1

Sharpen your vehicle control skills with this rocket landing game.

Techniques Discussed: Sprites without DATA statements. Redefined characters.

Squirrel Away 7

Susie Squirrel needs to store her winter supply of nuts in her underground burrow.

Techniques Discussed: Moving sprites and characters together. Detecting sprite-to-character collisions.

Ghost Hunt 13

Three friends are out late at night trying to catch the slippery, tricky neighborhood ghost.

Technique Discussed: Detecting sprite-to-sprite collisions.

Firecracker Boy 19

Save the Fourth-of-July parade from interruption by a mischievous kid.

Techniques Discussed: Two-part design to handle large programs. Using variables to control processing sequence.

Into the Pot 27

Trap your dinner by moving boulders to guide chickens to your cooking pot.

Technique Discussed: Variables with fixed values.

Yorick's Revenge 33

Video pinball game, patterned after the action of a pinball machine.

Technique Discussed: Programming for speed.

Space Crash 39

Protect your home planet from swarms of evil aliens.

Technique Discussed: Redefined characters.

The Blobbis 45

Break through the force field that protects three evil Blobbis.

Technique Discussed: Animation using arrays.

Sound the Whistle 53

Two-player game similar to football, without a ball.

Techniques Discussed: Considerations in creating sounds. Upper/lower case mode.

Bumper Ball 59

A "pong" type of game for one player.

Techniques Discussed: Programming for speed. Use of variables to position cursor. Complications in collision detection.

Ride the Wind 65

Learn to land an experimental hot-air balloon.

Technique Discussed: Calculating effect of outside forces on moving objects.

Death Valley Patrol 71

Rescue paratroopers who have been stranded in the barren Death Valley.

Technique Discussed: Controlled random screens.

Mission: Tobor 79

Refuel automated atomic energy reactors on the planet Tobor by remote control.

Technique Discussed: Special use of the keyboard buffer.

Cosmo's Rescue 87

Free Cosmo from the hazardous cavern of the yellow monster Grunk-Snort.

Technique Discussed: Bank-switching to handle large programs.

Mazemaster 95

Find your way through complex mazes of three-dimensional corridors.

Techniques Discussed: Completely random screen generation. Simulated three-dimensional display.

Appendix A 101



Preface

Arcade Games For The Commodore 64 has been created for the C-64 owner who enjoys playing computer games and wants to learn more about the "magic" that makes them work. We have provided fifteen games which demonstrate the power of the Commodore 64 home computer. The games utilize most of the features of the C-64, including color, sound, music, animated character and sprite graphics, multicolored characters and sprites, and redefined characters. Some of the games produce multivoice sound effects. Some use more than thirty sprites! All offer colorful, animated fun.

All the games are written in BASIC and annotated with enough programming detail that you will be able to explore and experiment with many aspects of the programs. Even if you do not have a complete understanding of the BASIC language or any previous programming experience, you can exercise your own creativity by changing the games, using the modification hints given. You may find this to be even more fun than playing them!

For each game, in addition to play instructions and program listings, we have included a discussion of some of the programming techniques used in the game's creation. The explanation should satisfy a non-programmer's curiosity, or point out topics for further study by readers who want to program games themselves.

Our intent was to present enough information that you will feel you understand the general method by which the programmer achieved a certain result. If you wish to achieve the same result in a program of your own, studying our program as an example will be helpful. You will probably need to refer to other texts, however, to

obtain all the information you need. In many cases, we provide specific references for additional details.

Because we wanted to create games which would exploit the power of the Commodore 64 computer and be fun to play, each game program is elaborate and contains many different tricks and techniques. An attempt to thoroughly describe just one program could fill a book. The "Programmer's Notes" for each game highlight just one or two game programming topics. If you examine the programs in detail, you will uncover many more techniques which you might wish to study further.

Though we have made our explanations as clear as possible, you may find them easier to understand if you have some familiarity with the *Commodore 64 User's Guide* which came with your computer. In fact, we have often made reference to that guide for further details on topics which were beyond the scope of this book.

Whether you are a game player, a programming enthusiast, or just someone curious about computers, reading this book and looking over the program listings should give you a general understanding of the structure of a game program and some methods by which programmers achieve the effects which make games fun.

In addition to many hours of entertainment, we hope our games provide you with a learning experience as you discover how the various programmers have taken advantage of the Commodore 64's special features to implement their game ideas.

*Mary Ann Chapman and David Harper
Fanfare House, Inc.*

Acknowledgements

This book benefitted from the efforts of many of the staff and friends of Fanfare House, Inc., some of whom made contributions in more than one area.

Credit is due to Mitch Bunnell, John Carmody, Carl Dobbs, Beryl Fielder, Dennis Fleming, Bud Gressett, Mike Kupka, Charles Manry, Frank Mikulastik, Charles Mott, Tracy Powell, Dick Ramsell, Stanley Rosenthal, Rick Ryznar, Penny Smith, Igor Tulchinsky, Steve Watson, David Williams, and especially Vera Gonzales, for their participation in past projects which made this one possible.

We appreciate the assistance of Bob Behrstock, Emery Froelich, Marilyn Harper, Hoyet Hemphill, Dick Ramsell, John Wheeler, and David Williams, all of whom reviewed the manuscript and made helpful suggestions. Leonard and Kim Brusatori parent-tested the text, and Todd and Amy Ramsel and Jim Dwyer kid-tested the games.

Dave Dusthimer of CBS Technical and Professional Publications gave us heart with his enthusiasm and happily accepted our format suggestions. We appreciate the efforts of Chris Titus, of The Blacksburg Group, who got us together with CBS.

Special thanks to Emery Froelich and Fred Burchard, who lent their support in many ways; to Jesse Rothstein and Arthur Falconer, who have been generous with their wisdom and advice; to our secretary Beryl Fielder, who deciphered programmers' scribblings; and to Marilyn Harper, whose technical assistance has saved the day on many occasions.

The manuscript was prepared using WORDSTAR on a Radio Shack TRS-80 Model II computer running the CP/M operating system.



Introduction

The games in this collection are all "arcade-style", requiring skill, coordination and strategy to achieve high scores. In order for you to understand and modify the game programs, we chose to write them in BASIC, which is the language most commonly used by Commodore 64 owners.

Although BASIC is simple to demonstrate and understand, it does not result in extremely fast-running programs. As a result, the games do not have quite the lightning speed found in true arcade games, which are generally written in faster languages, but they are still fun to play. The methods used to achieve the "arcade" effects are the same as those used in the faster games.

Our programmers have created games with everything from squirrels to space ships, including helicopters, hot-air balloons, ghosts, aliens and robots. You will be playing a pinball game, putting out firecrackers, storing nuts for the winter, and trying to rescue, capture or escape from all sorts of strange creatures. There are two games that simulate sports: one is a "pong" or handball-type game, while the other is similar to football, for two players.

Playing the Games

Loading and Running the Games

All games are loaded and run in the same way. Place the disk which came with this book into your disk drive. Be sure it is inserted correctly, with the label side facing up and the edge farthest away from the label inserted first.

Consult your disk drive user's manual if you need more assistance.

If you wish to display a list of all the game programs on the disk, type **LOAD"\$",8** and press the **RETURN** key. This loads the disk's program directory into the C-64's memory. When the computer displays **READY**, type **LIST** and press the **RETURN** key, to see the directory of programs.

To run a particular game, determine the exact name of the game from either the disk directory or the description of the game in the book. Type **LOAD"game name",8** and press the **RETURN** key. After a delay of 15 seconds or more, depending on the size of the game, the computer will display **READY**. This indicates that the game has been found and loaded into the C-64's memory and is ready to be played. Type **RUN** and press the **RETURN** key, and the game's title screen will appear. Further instructions on how to play are in the description for each game.

For example, one of the games is called **Ghost Hunt**. To load this game, type **LOAD"GHOST HUNT",8** and press the **RETURN** key. Note: The space between *GHOST* and *HUNT* is part of the name and must be included, or the computer won't find the game on the disk.

Some of the games actually consist of two programs. To play a two-part game, you load and run one program, and that program automatically loads and runs the other. The second program always has a name beginning with **&&**. For example, the game called **Firecracker Boy** actually consists of the programs **FIRECRACKER BOY** and **&& FIRECRACKER**. If you attempt to load and run **&& FIRECRACKER**, it will run but it may not work correctly, and the game's characters will have a very strange appearance.

How to Stop a Game

When you are finished with one game and want to play a different one, you must stop the first game and get the computer ready to accept a new **LOAD** instruction.

Most of the games can be stopped by pressing and holding down the **RUN/STOP** key and then pressing the **RESTORE** key. The C-64 will respond by displaying

READY, meaning that it is ready for a new command. You may then load and run a different game or run the same one again by typing **RUN** and pressing **RETURN**. Even after pressing **RUN/STOP** and **RESTORE**, the original game is still in the computer's memory.

Games which use memory bank switching techniques (which will be discussed later) cannot be stopped quite so easily. The descriptions of those games direct you to this section for instructions on stopping them.

In those games, pressing **RUN/STOP** and **RESTORE** will cause the computer to behave very strangely. Instead, use the following procedures: Press **RUN/STOP**. Type **RUN20000** and press the **RETURN** key. Then press and hold down **RUN/STOP** and press **RESTORE**. The computer will then say **READY**, and you can load and run another game, or type **RUN** and press **RETURN** to run the same game again.

Ordinarily these procedures will be sufficient to stop a game, load another one, and run it. However, due to memory manipulation techniques used in some games, you may occasionally have difficulty starting the next game. If that occurs, or if you want to stop a game and get the computer back to **READY** without going through the procedures above, simply turn the computer off and on again. This will completely clear the C-64's memory. Then load the game you want to play.

Joysticks. Many of the games are played with one joystick. For those games, plug a joystick into control port 2 on the right-hand side of the Commodore 64, next to the ON/OFF switch. One game, **Sound the Whistle**, is played by two people and requires two joysticks, one plugged into each control port. Be sure you hold the joystick in the proper position.

Sound. All of the games are spiced with interesting sounds as they are played. Adjust the volume on your television or monitor to a comfortable level so that the sound effects will add to your enjoyment of the games. Of course, if someone in the room doesn't share your appreciation for electronic game sounds, you can turn the volume off completely.

Information provided for each game

The initial section of information on each game explains the basic idea of the game, describes the scene, introduces the characters, and explains how to load and play the game. You may wish to read just that section and then play the game, before proceeding to the program documentation which follows.

In the Programmer's Notes, the programmer of the game explains some of the methods used to make the game work as it does. Often this section includes a description of a programming challenge inherent to the game design, and how it was solved. Our programmers have used many techniques, only some of which are explained. If you carefully examine the program listings, you will uncover and begin to understand other techniques that can be useful if you start writing your own programs.

Hints for Modifying the Game include both simple changes which you can make to adjust the speed or difficulty of the game, and general suggestions for more complex modifications. As you play the games, other ideas may occur to you. When you begin to implement your own ideas, you will see how really creative the programming process can be!

Under Major Routines, you will find a synopsis, by line number, of the major functions of the game program. The significant variables used in the program are listed in the Major Variables section, along with a brief description of their use. Both of these sections will aid your understanding of how the program works.

For each game which uses sprites, a Sprites section provides information for each sprite used. The Line Number column, next to the description of a particular sprite, indicates the line numbers in the program where the DATA statements for that sprite may be found. The location in memory where each sprite is stored is represented by the number under the Page heading.

Each game which uses redefined characters has a Characters section which provides information for each redefined or custom character used. The Line Numbers

column, next to the description of a particular character, indicates the line numbers in the program where DATA statements used for the redefinition may be found.

Sprites

The sprite feature is the C-64's way of handling special shapes which are to be moved around on the screen. By creating moving objects as sprites, a programmer greatly simplifies the programming required to produce a game on the C-64.

Sprites are explained in Chapter 6 of the *Commodore 64 User's Guide*. Because this powerful feature is used extensively in most of the games in this book, a little further explanation here may be helpful. If your interest in technical details is minimal, this discussion is not essential to your understanding of the rest of the Introduction and the game descriptions.

To create a sprite, the programmer includes data in the program to tell the computer what size, shape, and color the sprite is to be and where it is to be located on the screen at any particular moment. The computer then automatically handles many functions that otherwise would require extensive programming.

The computer's VIC (Video Interface Controller) chip uses the programmer's data to display the sprite on the screen with the desired color, shape and size. By continuously redisplaying it at locations specified by the programmer, the VIC chip makes the sprite appear to move around. The VIC chip also detects collisions when the sprite is in the same place on the screen as another sprite or a background character.

The data which specifies the sprite's shape is usually coded into DATA statements in the program. With instructions which are executed very early when the program begins running, the sprite shape data is moved into an area of memory outside the program, where it is accessed later by the VIC chip.

The C-64's memory contains 64 Kilobytes of storage (1K = 1024 bytes or characters). It is organized into "banks" of 16K each, which are further divided into 256 "pages" of 64 bytes each. A sprite shape definition is 3 bytes wide by 21 bytes high, or a total of 63 bytes. The

64th byte of a page used to define a sprite always has a value of zero. In our documentation, the number under the Page heading is a reference to the page in memory, outside the program itself, where the shape data for a particular sprite is stored.

For example, if the page number for a rocket sprite stored in bank 0 is 192, the data that determines the shape of the rocket is stored in the C-64's memory beginning at page 192, or location 12288 ($192 * 64 = 12288$).

In most of these games, the sprites are stored along with the program in the first 16K bank of memory, which is known as bank 0. Some games use a technique called bank switching, which permits the sprites to be accessed by the VIC chip from a different bank. If a game uses a bank other than 0, its program documentation will so indicate. (For more discussion of bank switching, see the Programmer's Notes for the game **Cosmo's Rescue**.)

A maximum of eight sprites may be on the screen at any one time. Eight locations in memory, beginning at location 2040, are used as sprite page number pointers. When a game is running and the program calls for a sprite object to be displayed, the VIC chip obtains the page number to use from the sprite page pointer location assigned to that sprite.

When an object is to be animated, several different sprites are used, each representing a different step in the animation. Then a sequence of different sprite pages, one for each sprite, is used to create the animation effect. The page number stored in the sprite page pointer is continuously changed to the next page number in the animation sequence.

For each game, we have printed a few of the major sprites in graphic form. To the right of each sprite printout, you will find three columns of numbers. Each row contains the actual numbers used in the program's DATA statements to define the corresponding row of the sprite. The numbers below the sprite represent the location in memory (bank and page) where the sprite data is stored.

If you would like to learn how the numbers in DATA statements translate into a particular sprite shape, how sprites can be more than one color, and other details of using sprites, refer to the *Commodore 64 User's Guide* or to

some of the other excellent books now available about sprite programming on the Commodore 64.

Sprites are somewhat complicated to set up initially, but then they can be controlled and moved around with a minimum of programming. In most cases they offer the most advantageous method of creating moving objects. Their disadvantage is that no more than eight may be on the screen at one time.

Characters and Redefined Characters

The C-64 also offers another method of creating movable shapes in a program. You may have seen "graphics" programs in which characters such as *X* and *O* are used to draw the shapes. In addition to alphabetic and numeric characters, graphics characters with many different shapes are available on the C-64. These graphics characters can be arranged to form the desired object, and program instructions can be written to move the group of characters around on the screen.

It is possible to invent entirely new graphics shapes if the ones available aren't suitable for the desired effect. The programmer then includes instructions which redefine the standard characters to give them the specified new form.

Some of the game programs in this book use redefined characters to create animated figures and other special shapes. Once again, consult one of the books on graphics programming for the Commodore 64 for complete details on how this is done.

Since there are 256 characters in the C-64 character set, there may be up to 256 redefined character shapes in a program. Therefore this technique is sometimes used when a game design calls for a large number of movable objects, even though much more programming is required to move them around on the screen than is necessary with sprites. The most advantageous use of redefined characters is for complex background shapes that do not require movement.

The Program Listing

For readers who do not have a printer, and to save time for those who do, we have included a program listing. It is printed exactly as it would be if you loaded the game and

printed it on your own printer. A complete listing is invaluable if you want to learn how a program works or figure out how to modify it.

The many strange symbols in the listings are the Commodore 64's special graphics and control characters. When you list the program on your computer monitor, you will see the same symbols. To learn which keys to press to obtain these symbols and what the control characters do, consult Chapter 5 of the *Commodore 64 User's Guide* or one of the other books available on graphics programming for the Commodore 64.

Modifying the Games

If you want to follow our modification hints and experiment with changing the games, we suggest that you first set up a separate disk for your modified versions.

First, format a new disk as explained in the user's manual for your disk drive. Load the program into your computer from the original disk. Then remove the original disk, insert your new formatted disk, and save the program. You will still have the program loaded in the computer, and you can then begin making your changes.

Remember that when making changes, it is very easy to make errors that introduce "program bugs". The bugs can be severe enough to cause the computer to "lock up" when you try to run the game. If this happens, you will be unable to save your changes. Therefore, we suggest that each time you are ready to try running a newly modified program, you save it first. Even if it contains a bug, it may be easier just to correct the bug than to enter all your changes again.

Each time you save a version of the game, give it a different name, and keep each version until you are sure you will never want it again. For example, if you are working with the game **Ghost Hunt**, you may wish to name the modified versions **GHOST HUNT 1**, **GHOST HUNT 2**, and **GHOST HUNT 3**. After you are certain that **GHOST HUNT 3** works, you may decide to delete the other two versions from the disk. When you complete a modified version that you plan to save indefinitely, give it yet another name, such as **GHOST HUNT A** or **GHOST CHASE**.

NEVER TRY TO SAVE A PROGRAM ON THE DISK THAT CAME WITH THIS BOOK! You might accidentally damage the disk and lose all its games forever! As a safeguard, you could place a write-protect tab over the square notch on the original disk.

Try Programming Yourself!

Many computer owners begin to explore programming because they are intrigued by the inner workings of the programs they have purchased. As you begin to learn more about programming and begin to understand the game programs in this book, you may have modification ideas of your own. Go ahead and try them!

You may find yourself becoming more and more involved in programming, just because you enjoy it. Writing programs can be a lot of fun!

Programming Guidelines

If you decide to do much programming, you will want to refer to one of the many programming texts on the BASIC language. To supplement the information they provide on the features of the language, we would like to offer some hints on style and structure which can make your programs easier to debug and modify. These techniques can make your programming efforts more successful and less frustrating, whether you are writing original programs or simply modifying the game programs in this book.

1) Plan first – program later. Think through your design or proposed change before you begin to write any program instructions. Write down everything the program will do and the steps it will follow.

2) Don't be too clever. Use straightforward logic that will still make sense to you if you decide to work on the program again a month later.

3) Plan for speed, but don't make it your first priority. Get the program working correctly first. Then if some part of it seems to be running too slowly, you can concentrate your efforts on speeding up that routine.

4) Organize your program by dividing the program tasks into logical subroutines to be performed by one control section. Place the subroutines below the control

section. Then, when looking at or working on the program, you can see the sequence of tasks at a glance.

5) Make variable names as meaningful as possible. **SC** would be a better variable name than **R9** to keep track of the score. Use a name like **VI** for the base address of the VIC chip (53248). **VI + 16** is more easily remembered and meaningful than **53264**.

6) Document your programs internally with remarks. Identify the function of each subroutine with a remark at the beginning of the subroutine. Identify the usage of each group of DATA statements. Comment on any section of the program for which the function is not obvious. These remarks can be great timesavers later if you have to debug or modify the program.

As you examine the programs and their documentation in this book, ask yourself two questions: 1) What do you wish the programmers had done to make it easier for you to understand the program or to help you make changes? 2) What did they do which you found helpful? Your answers to these questions will provide useful guidance when you begin writing your own programs.

Compilers and Other Languages

If speed is important in your programs, you may find that the Commodore 64's BASIC interpreter cannot execute your program instructions quickly enough. Each time the program is run, an interpretive language must convert each statement in the program into machine language before it can be executed by the computer.

You may wish to acquire a BASIC compiler, which is a program used to convert your BASIC *source* program into machine language in a separate step, creating an *object* version of the program. No time is required for conversion when the object program is executed, and the program runs faster. You may find it runs so much faster that you actually have to make changes to slow it down!

Usually there are subtle differences between interpreter and compiler versions of the same language. Often some programming changes are required when moving from one to the other. You would be well-advised to wait until you are very comfortable with programming on the C-64 before you attempt to use a compiler.

You may also want to learn to use assembly language to speed up critical routines in your BASIC programs. Or, you might investigate other languages which are faster than BASIC, such as Pascal, FORTH, or 'C', all of which are compiled languages.

Programming References

For programming information beyond the scope of the *Commodore 64 User's Guide*, a number of excellent publications are available. The *Commodore 64 Programmer's Reference Guide* (Commodore Business Machines, Inc. and Howard W. Sams, Inc., 1982), available in bookstores, is a complete technical reference for the C-64 and is essential for the serious programmer.

For a step-by-step tutorial, however, the many other books available on C-64 graphics programming will be more appropriate. It is difficult to recommend specific titles because, while many of them are excellent, they vary widely in content and level of complexity. Find a bookstore with a good selection of Commodore computer books, and select some that are designed for your level of knowledge and include the topics of interest to you.



**Arcade Games
for the
Commodore 64**



Programmer's Notes

Sprites without DATA Statements. Five different sprites for the rocket in **Rocket Lander** are defined in DATA statements. A sixth sprite, for the landing pad, is defined using another technique instead of DATA statements.

The landing pad is simply a solid rectangular box with nothing showing on the screen except its top. If the sprite had been defined with DATA statements, all the numbers in the DATA statements would be 255. Instead, line 10 of the program contains a loop which stores the value 255 directly into all 63 bytes of memory set aside for the landing pad sprite definition.

Many sprites with simple designs can be created in a similar way. Instead of typing one value over and over again into DATA statements in the program, taking up valuable space in memory, let the computer do the work!

Redefined Characters. The futuristic landing control tower and the rugged landscape were created with redefined characters, defined in the eighteen lines of DATA statements at the end of the program. Line 170 replaces the standard characters with the values in the DATA statements. The Introduction to this book and the Programmer's Notes for the game **Space Crash** offer further discussion of redefined characters.

If you would like to see the standard characters which were redefined to obtain the characters that appear on the screen in **Rocket Lander**, do the following: Turn the computer off and on again. Load the game in the usual manner but do not type **RUN**. Type **170** and press **RETURN**. (This will delete line 170.) Now type **RUN** and press **RETURN**. The screen will be displayed with standard instead of redefined characters, but otherwise, the game will run as usual.

Rocket Lander

Hints for Modifying the Game

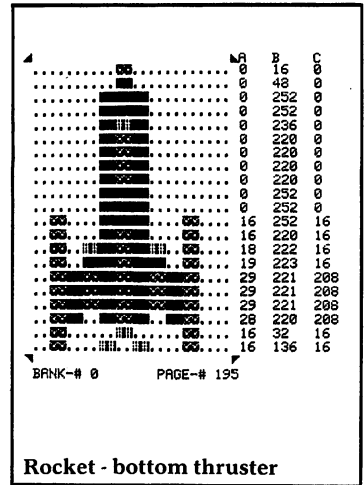
The variable **K1** represents the vertical speed of the rocket. In line 440, the expression **K1 < -.1** checks for the speed at landing time, **-.1** being the maximum speed,

allowed for a safe landing. Substituting other values for the .1 will make landing easier or more difficult.

The statement $K1 = K1 - .1$ in line 430 decreases the speed when the main engines are fired, and the statement $K1 = K1 + .1$ in line 437 simulates the force of gravity. Again, substituting other values for the .1 will change the difficulty of the landing.

The amount of fuel remaining is stored in the variable **PO**. The beginning fuel supply is set to 100000 in line 360 and can be changed to other amounts. Lines 410 and 420 decrease this amount by 100 when the side thrusters are fired. Line 430 decreases the amount by 500 when the main engines are fired. Changing these values will affect the fuel efficiency of the rocket.

You might want to modify the action of the side thrusters. The rocket moves sideways only when a thruster is fired and stops moving sideways when the thruster is released. Consider a method to keep the rocket's current horizontal momentum in effect until it is modified by further thruster firings.



Major Routines

0009-0025 INITIALIZATION

Generate landing pad sprite and set up array X\$() with radar antenna characters.

0030-0060 PRINT TITLE SCREEN

Display title screen until sprite data is loaded into memory.

0070-0086 PRINT INSTRUCTION SCREEN

0090-0170 LOAD CHARACTER DATA

Lower top of BASIC memory. Copy standard character set into main memory and replace 18 of them with redefined characters. Point VIC chip to new location of character set.

0200-0320 PRINT BACKGROUND

Print background using regular and redefined characters.

**Rocket
Lander**

0350-0370 INITIALIZE SPRITES

Set size, color, and location. Pick random location for landing pad. Set initial fuel supply.

0400-0470 MAIN GAME LOOP

Read current key pressed. If valid function key, select appropriate rocket sprite, decrease fuel supply, and adjust position of rocket. Prevent rocket from leaving screen to left or right. Add pull of gravity and check for good landing, crash, or out of fuel. Rotate radar antenna and print remaining fuel.

0500-0530 GOOD LANDING

Flash screen and make sound. Update good landing count and wait for any key pressed to begin again.

0600-0630 CRASH LANDING

Scroll screen, flash rocket sprites, and make sound. Update crash landing count, show "splat" rocket, and wait for any key pressed to begin again.

0700-0900 SOUND ROUTINES

Three sound routines.

1000-1020 WAIT FOR KEY PRESSED

5000-5190 SPRITE DATA

5200-5360 CHARACTER DATA

Redefined characters. One per line.

Rocket Lander

Major Variables

General:

V Base address of VIC chip
SO Base address of SID chip
SL Address of hardware interrupt
SC Base address of screen memory

BA Base address of relocated character set
 AA Current key pressed
 V\$ String of cursor downs
 H\$ String of cursor rights

Radar Antenna:

X\$() Character strings for animation
 TU Counter for animation

Rocket:

RS Pointer to sprites
 K1 Vertical movement
 H1 Horizontal position
 V1 Vertical position
 CR Number of crash landings
 LN Number of good landings
 PO Fuel

Sprites

Line Numbers	Page	Description
See notes	13	Landing pad
5000-5030	192	Rocket - no thruster
5040-5070	193	Rocket - right thruster
5080-5110	194	Rocket - left thruster
5120-5150	195	Rocket - bottom thruster
5160-5190	196	Rocket - splat

Redefined Characters

Line Number	Description
5200	Solid block
5210	Mountain part - 1
5220	Mountain part - 2
5230	Mountain part - 3
5240	Mountain part - 4
5250	Mountain part - 5
5260	Mountain part - 6
5265	Mountain part - 7
5270	Mountain part - 8

**Rocket
Lander**

5280	Mountain part - 9
5290	Building part - 1
5300	Building part - 2
5310	Building part - 3
5320	Building part - 4
5330	Building part - 5
5340	Building part - 6
5350	Mountain part - 10
5360	Mountain part - 11

**Rocket
Lander**

Squirrel Away

Frank Mikulastik

Susie Squirrel has gathered her winter supply of nuts from the trees, and now she needs to get them stored safely away. By aiming very carefully, she can drop them into holes in the ground which lead to her underground burrow. However, her timing must be just right, because the birds flying by will snatch them from the air if they get half a chance!

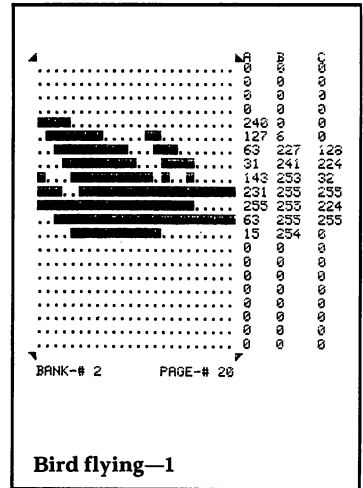
Control Susie's left and right movements with a joystick, and drop the nuts with the fire button. She has gathered 104 nuts – see how many you can successfully drop into her burrow.

Load the game by typing **LOAD"SQURREL AWAY",8** and pressing the **RETURN** key. After about 25 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. Susie will chatter at you from the title screen until you press the fire button and proceed to the instruction screen. Press the fire button again to start the game. When the game is over press the fire button to return to the instruction screen.

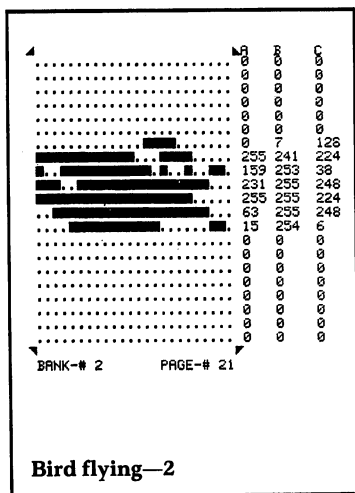
To stop the game, refer to the section in the introduction for stopping bank-switched games. This game uses memory bank 2 for sprites and graphics.

Programmer's Notes

Moving Sprites and Characters Together. This game design required 104 nuts, each of which must remain visible on the screen if it gets safely into the burrow.



Squirrel Away



Because the C-64 allows a limited number of sprites, the nuts were created as characters. The squirrel and birds, however, were created as sprites. (See the Introduction for a discussion of the differences between characters and sprites.)

Programming the squirrel carrying a nut presented an interesting problem. The squirrel and nut must move in unison, but the squirrel sprite is positioned with pixel coordinates, while the nut character is positioned with character coordinates. To make the squirrel appear to be carrying the nut as it moves, the squirrel's coordinates are changed by eight pixel units for each character unit by which the nut moves.

Making the motion appear natural when the squirrel turned around still remained a problem. This was solved by experimenting, which is part of the fun of developing video games. When you play the game, observe how this was worked out — when the squirrel turns in place, the nut jumps several character units to the other side of her.

Detecting Sprite-to-Character Collisions. When a moving sprite and a character come together on the screen and touch one another, a "sprite-to-character collision" occurs. Since the logic of a program often requires detection of these collisions, the Commodore 64's VIC chip has a special sprite-to-character collision register, located at memory location 52379 (sometimes referred to as the sprite-to-background collision register). The computer uses this register to reflect the current collision status of all the sprites.

Each sprite is monitored by one bit in the register — for example, bit 1 monitors sprite #1. Bit 1 ordinarily has a value of 0, but is set to a value of 1 when sprite #1 is involved in a collision. By reading the register as a binary number, the program can determine which sprites are currently involved in a collision. It is a good practice to save the register's contents in a variable, because once it is read, the register is automatically cleared.

In *Squirrel Away*, the sprite-to-character collision register is checked in line 2172 to detect a collision between any of the bird sprites and the falling nut.

Squirrel Away

Hints for Modifying the Game

Squirrel Away would be more difficult to play if the birds were larger or moved faster. Add a new line 305 reading **305 POKEV + 29,254** to make the birds longer. Currently the speed of the birds is calculated by adding $N * 2 + 8$ to **BX(N)** in line 3010. Try changing the 2 to other values.

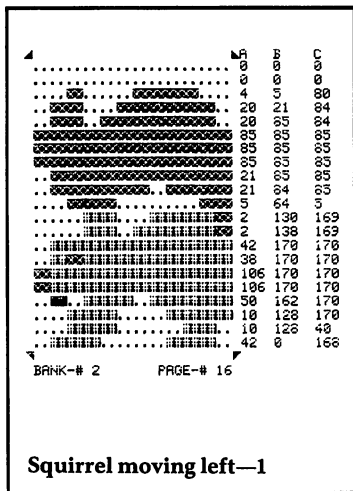
The birds could be made to fly across the screen in some fashion other than a straight line. Be careful, however, not to let them hit a background character other than the nut, because the program would then detect a sprite-to-character collision and proceed as if a nut had been eaten. The birds may hit each other safely, because they are all sprites and there is no check in this program for sprite-to-sprite collisions.

You can make it easier to aim the nut by letting the nut remain stationary when the squirrel changes direction. Instead of rotating the squirrel in place and making the nut jump from one side to the other, make the squirrel jump to the opposite side of the nut.

Major Routines

- 0030-0340 **INITIALIZATION**
Set VIC memory to bank 2. Read sprite data into memory. Initialize variables and sprites. Print title, instruction, and play screens.
- 0500-0550 **MAIN LOOP**
Move birds and read joystick. If fire button pressed, drop nut. If there are no nuts left to drop, end game. Move squirrel and nut.
- 0700-0790 **INITIALIZE SPRITES**
Set up starting positions for squirrel and birds.
- 0800-0890 **PRINT PLAY SCREEN**

**Squirrel
Away**



- 1500-1550 **MOVE SQUIRREL LEFT**
Calculate squirrel's new position, update sprite pointers, and make a sound.
- 1600-1650 **MOVE SQUIRREL RIGHT**
Calculate squirrel's new position, update sprite pointers, and make a sound.
- 2000-2020 **MOVE THE NUT**
Move nut with squirrel.
- 2100-2250 **DROP THE NUT**
Move nut down the screen and move birds. Check for nut hitting ground, birds, or burrow. Decrease nut count.
- 2300-2395 **NUT DROPS IN THE HOLE**
Move nut to the right of burrow and blink birds. If no nuts left, end game.
- 3000-3110 **MOVE THE BIRDS**
Update positions and sprite pointers. If bird moves off screen, relocate on left.
- 4000-4010 **NUT DROPS IN THE HOLE SOUND**
- 4100-4110 **NUT MISSED THE HOLE SOUND**
- 4200 **SQUIRREL MOVEMENT SOUND**
- 4300-4320 **NEW SCREEN SOUND**
- 4400-4440 **INITIALIZE SOUND**
- 5000-5090 **PRINT TITLE SCREEN**
- 5100-5130 **GAME IS OVER**
- 5200-5290 **PRINT INSTRUCTION SCREEN**
- 6000-6950 **SPRITE DATA**

Squirrel Away

Major Variables

General:

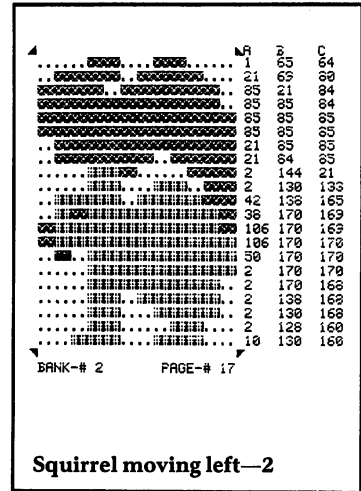
V	Base address of VIC chip
S1	Base address of SID chip
SM	Base address of screen memory
CM	Base address of color memory
M1	Base address of sprite pointers
BS()	Sprite page numbers for birds

Squirrel:

SP()	Sprite page numbers
D	Direction of movement
SN	Animation sequence counter
X	X position in graphics mode
X1	X position in sprite mode
Y	Y position in graphics mode
Y1	Y position in sprite mode

Nut:

T1	ASCII code for shape
T2	Color
TP	Position in screen memory (1-1000)
W1	Number left to drop
W2	Number dropped into burrow



Sprites

(NOTE: All sprites are in bank 2)

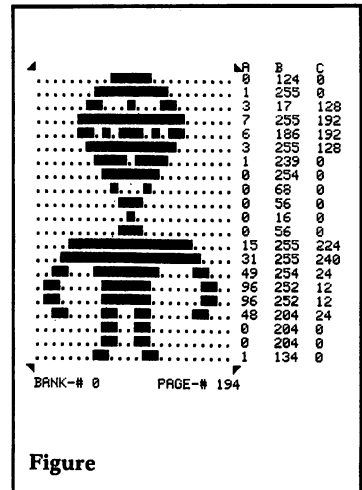
Line Numbers	Page	Description
6000-6130	16	Squirrel moving left - 1
6140-6270	17	Squirrel moving left - 2
6280-6400	18	Squirrel moving right - 1
6410-6540	19	Squirrel moving right - 2
6550-6680	20	Bird flying - 1
6690-6820	21	Bird flying - 2
6830-6950	22	Bird flying - 3

**Squirrel
Away**



Ghost Hunt

Igor Tulchinsky



Three friends are out late at night trying to catch the neighborhood ghost. She is a slippery, tricky ghost, and in order to capture her, all three figures must touch her at once without touching each other.

With a joystick, you control the movement of one figure while the other two stand still. The figure you are controlling is yellow, and you can switch to a different one by pressing the fire button. If you hold down the fire button while moving the joystick, all three figures will move in the same direction.

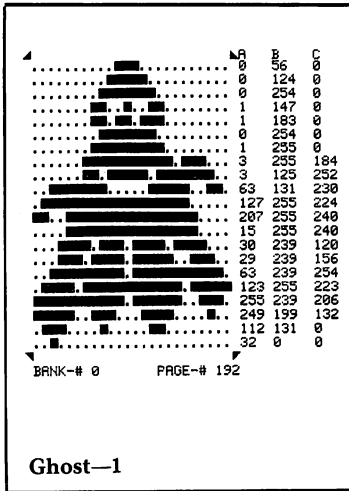
You are allowed a certain amount of time to catch the ghost. After you have caught her, you can play again, but each time she will be moving faster and the amount of time allowed to catch her will be shorter. You will accumulate points based on the length of time you take to catch each ghost. The game ends when the time runs out and the ghost has not been captured.

Load the game by typing **LOAD"GHOSH HUNT",8** and pressing the **RETURN** key. After about 20 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key.

Programmer's Notes

Detecting Sprite-to-Sprite Collisions. When two or more moving sprites come together on the screen and touch one another, a "sprite-to-sprite collision" occurs.

**Ghost
Hunt**



Since the logic of a program often requires detection of these collisions, the Commodore 64's VIC chip has a special sprite-to-sprite collision register at memory location 53278. The computer uses the register to store the current collision status of all the sprites.

Each sprite is monitored by one bit in the register – for example, bit 1 monitors sprite #1. Bit 1 ordinarily has a value of 0 but is set to a value of 1 when sprite #1 is involved in a collision. By reading the register as a binary number, the program can determine which sprites are currently involved in a collision. It is a good practice to save the register's contents in a variable, because once the register is read, it is automatically cleared.

In **Ghost Hunt**, the ghost is sprite #0. The figures are sprites #2, #3, and #4. When all of the sprites are involved in a collision, bits 0, 2, 3, and 4 will be set to 1, giving the collision register a value of $1 + 4 + 8 + 16$, or 29. In line 210, the sprite-to-sprite collision register is read, and its contents are stored in variable **Q1**.

The game design specified that for the ghost to be captured, none of the three figures could be touching each other. This presented a programming problem, because when each figure is touching the ghost, each is involved in a collision regardless of whether it is touching another figure. This problem is solved in Line 231 of the program, in the following way:

- 1) The ghost sprite is turned off. As long as it is off, the sprite collision register will not detect its collisions.
- 2) The sprite collision register is cleared.
- 3) A delay of at least 1/60th of a second is executed, during which the system can detect any collisions between the figures and update the register.
- 4) The sprite collision register is read again. If no collisions are detected, then no figures are touching each other and the capture is valid. However, if any figure is touching any other, then the sprite collision register will read greater than zero and the ghost will not be captured.
- 5) The ghost sprite is turned back on and the program continues.

Ghost Hunt

This all occurs so quickly that you never see the ghost disappear!

Hints for Modifying the Game

The statement `IFCT = 40` in line 52 controls how long the ghost will stay in one position before heading off in a new direction. Change the 40 to a larger number to have the ghost remain in one spot longer. A smaller value will create a very slippery ghost!

Each time the fire button is pressed, the next figure in sequence is selected for movement. To select the next figure randomly when the fire button is pressed, change line 12 to read:

```
12 IF(NOTBS%AND16)=16THENN1% = N%:N% = INT(RND(8)*3 + 1)
```

You might want to try changing the distance (in pixels) by which the figures and the ghost move, as the score increases. This would best be done in the `ADJUST SCORE` routine.

Major Routines

- 00010-00040 **MOVE FIGURE**
Check fire button. If it has been pressed, switch figures by changing colors. Check joystick and move figure selected.
- 00051-00110 **MOVE GHOST**
Make a sound. Check whether counter has reached a certain value. If so, pick new spot for ghost to move toward. Move ghost toward hidden spot. Switch to other ghost sprite, causing animation. Make a sound.
- 00200-00206 **INITIALIZATION**
- 00210-00300 **MAIN CONTROL LOOP**
Move figure and ghost, and read sprite collision register. Print remaining time on screen. If less than ten seconds left, flash screen and sound alarm. If no time left, end game. If ghost has not been cap-

**Ghost
Hunt**

tured, start over. If ghost has been captured, increase score and speed, and resume.

- 10000-10026 **SPRITE DATA**
- 11000-11090 **PRINT GAME SCREEN**
Draw earth, sky, and stars while making noises.
- 12000-12500 **PRINT TITLE SCREEN**
Initialize sound. Draw border dots while making musical sounds. Print title. Read in sprite data for ghost. Reinitialize sound. Make ghost fly around, while making ghostly whine and waiting for fire button to be pressed.
- 13000-13116 **SPRITE INITIALIZATION**
Read sprite data into memory. Set color and other sprite parameters. Initialize sprite-related variables.
- 13120-13180 **SOUND INITIALIZATION**
Clear sound chip. Define voices used. Print "Score" at top of screen.
- 15000-15040 **GAME OVER**
Make whine sound. Print farewell message. Wait until either fire button is pressed or 45 seconds have elapsed, then restart game.
- 16000-16020 **ADJUST SCORE**
Print score after converting it to string variable. Increase speed of game.
- 18000-18210 **PRINT INSTRUCTION SCREEN**
Print instructions and wait for player to press fire button before starting game.

Ghost Hunt

Major Variables

General:

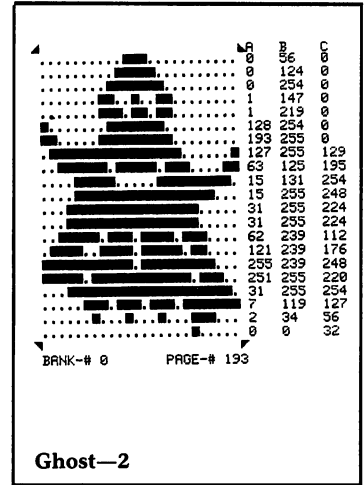
- V Base address of VIC chip
- S Base address of SID chip
- BS% Joystick status
- TZ Time left
- NP Score points awarded
- SC Score
- VZ Frequency of sound
- SI Increment of frequency
- MM Number of pixels moved by all sprites
- VL Time allowed to catch ghost

Figure:

- N% Number selected
- CL Color
- D3% Horizontal movement
- E3% Vertical movement
- X% () Horizontal positions
- Y% () Vertical positions
- PX Address of horizontal position
- PY Address of vertical position

Ghost:

- CT Counter for movement
- X8% Horizontal position to head toward
- Y8% Vertical position to head toward
- X3% Horizontal position
- Y3% Vertical position
- D% Horizontal change in position
- F% Vertical change in position
- DX% Horizontal movement
- DY% Vertical movement
- M% () Magnitude of movement



Sprites

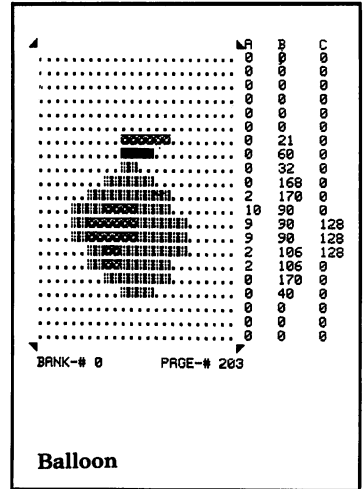
Line Numbers	Page	Description
10000-10006	192	Ghost - 1
10010-10016	193	Ghost - 2
10020-10026	194	Figure

**Ghost
Hunt**



Firecracker Boy

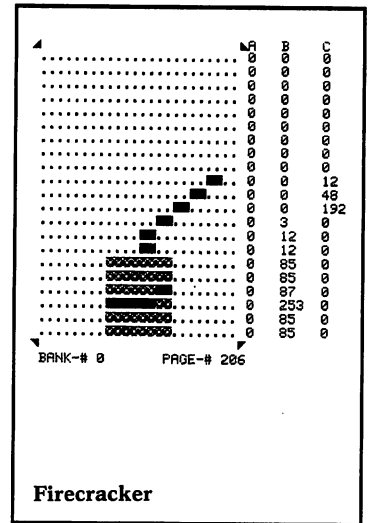
Charles Mott, Jr.



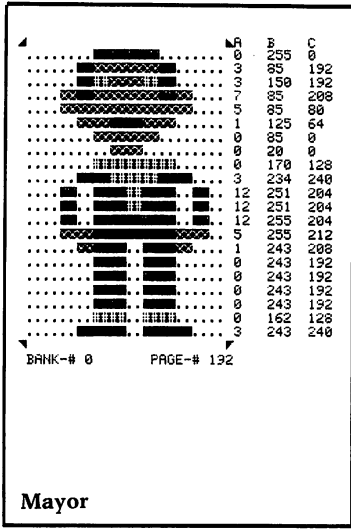
The Mayor's Fourth-of-July parade is in danger of interruption by a mischievous boy who is lighting firecrackers. The Mayor is on the reviewing stand watching the colorful floats and cars, and if he hears a firecracker explode, he will do a flip and stop the parade! You are a firefighter assigned to drop water balloons on the fuses of the firecrackers before they explode.

Control your movements left and right on the top floor of the blue Fire Station with a joystick, releasing the water balloons with the fire button. The net moving back and forth in front of the fire station may hamper your efforts by intercepting your balloons as they fall. The boy can trick you by hiding his firecrackers behind the door of the building, but you have your own tricks: you can drop balloons from behind the window pillars, and you can hit a firecracker even when it is behind a float. Points are scored for each firecracker you manage to put out.

Load the game by typing **LOAD"FIREFCRACKER BOY",8**. Then press the **RETURN** key. After about 25 seconds the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. After the title screen is displayed, the program will pause briefly before the game begins.



Firecracker Boy

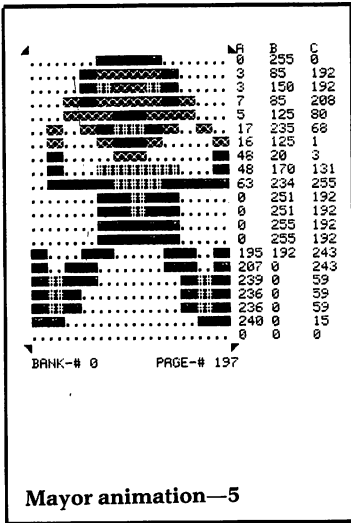


Programmer's Notes

Two-Part Design to Handle Large Programs. To achieve its animation effects, **Firecracker Boy** uses a total of 33 sprites, including 10 Mayor sprites, 10 firecracker sprites, and 2 sprites for the mischievous boy. In addition to the great number of sprites, the features of the game required a very large program. To accommodate all this, the program was written in two parts. Part 1 begins by displaying the title and instruction screens, while it loads the sprite data from the program into the sprite pages. It then automatically loads and runs Part 2 (the actual game) from disk.

Line 670 of Part 1 loads one byte of sprite data into memory. In a number of different places throughout the routines which display the title and instruction screens, **GOSUB 670** instructions gradually load all the sprite data. After the sprites are loaded, line 710 protects them from being over-written by specifying a smaller portion of memory to be used for variable storage in Part 2.

Lines 720 and 721 use the keyboard buffer to load Part 2 of the game. See the Programmer's Notes for the game **Mission: Tobor** for an explanation of this special use of the keyboard buffer.



Using Variables to Control Processing Sequence. Often the sequence of processing in a program depends on whether or not a certain condition exists. One method for controlling program sequence is the use of variables as "flags". At a logical place in the program, a test is made for the condition, and a variable is set to indicate whether or not the condition exists. When the course of the game at some later time depends on that condition, the variable is checked.

Part 2 of **Firecracker Boy** uses the array **FG()** for flags to monitor various conditions which may occur during the game. Each element of the array is designated to indicate a particular condition. When the condition does not exist, or is **FALSE**, the flag is cleared, or set to 0. If the condition then becomes **TRUE** during the course of the game, the flag is set to a positive number.

For example, **FG(2)** is the "firecracker-on-screen" flag. It is set to **FALSE** when the boy walks onto the screen carrying a firecracker and set to **TRUE** when he drops it.

Firecracker Boy

When a water balloon hits the firecracker, the flag is set back to *FALSE*. After the boy walks off the right side of the screen, the flag is continuously tested to see whether he should walk back onto the left side with a new firecracker. He will not reappear until a water balloon has put out the current firecracker and the flag is found to be *FALSE* once again.

The array **DR()** is used in a slightly different way to control sequence. The **DATA** statement at line 190 contains numbers coded to point to certain lines of the program. In lines 140-150, those numbers are loaded into **DR()**. Then in line 250, that array is used to determine what process is to be done next, based on the position of the joystick. The joystick port is read by a **GOSUB** to line 570, and the result is stored in the variable **JV**. If **JV** was set to 4, then the fourth element of array **DR()** is used to select the line in the program which is to be executed next.

For more information about **ON GOTO** and **ON GOSUB** instructions, see the *Commodore 64 User's Guide*.

Hints for Modifying the Game

Part 1. Lines 380-440 use **PRINT** instructions to display the contents of arrays **X\$()**, **Z\$()** and **C\$()**, which are the word *FIRE*, the word *CRACKER*, and a picture of a firecracker. For some ideas on other ways of printing these arrays, see the hints for modifying the game **Mission: Tobor**.

An alternative way of writing **Firecracker Boy** would be to load the sprite data from a disk file instead of having it in **DATA** statements. For a programming exercise, try creating a disk file of all the sprite data and turning **Firecracker Boy** into a one-part program. Move the title and instruction screen routines from Part 1 to Part 2, and add a routine in Part 2 to read the sprite data from a disk file. This could be an involved process, but it would be a good lesson in creating and working with disk files.

Part 2. The variables **PH** and **AP**, which control the number of pixels by which each of the main figures moves, are initially set to 9 in line 70. Smaller initial

```

A
..... 0 255 0
..... 3 85 192
..... 3 150 192
..... 7 85 208
..... 5 85 80
..... 1 125 64
..... 0 85 0
..... 0 20 0
..... 0 170 128
..... 3 234 240
..... 12 251 204
..... 12 251 204
..... 12 255 204
..... 5 255 212
..... 2 243 208
..... 8 243 224
..... 32 170 136
..... 128 170 130
..... 255 255 255
..... 59 0 59
..... 12 0 12
B
C

```

BANK-# 0 PAGE-# 224

Mayor float

```

A
..... 0 0 15
..... 0 84 3
..... 0 12 243
..... 0 12 251
..... 0 12 251
..... 4 15 240
..... 13 15 192
..... 53 79 192
..... 217 79 192
..... 213 91 132
..... 213 219 192
..... 213 219 128
..... 213 219 192
..... 217 75 192
..... 53 79 240
..... 13 15 255
..... 4 12 251
..... 0 12 251
..... 0 12 3
..... 0 84 15
..... 0 0 0
B
C

```

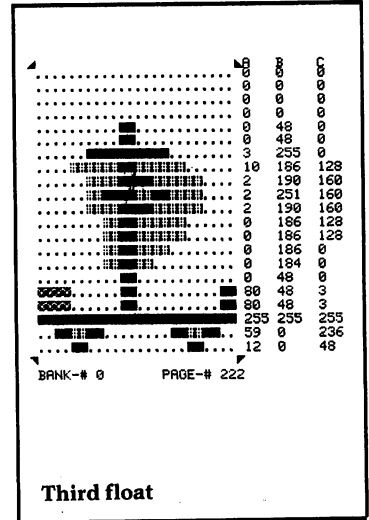
BANK-# 0 PAGE-# 200

Mayor flip—3

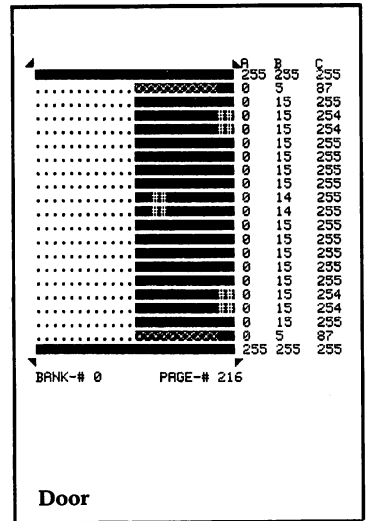
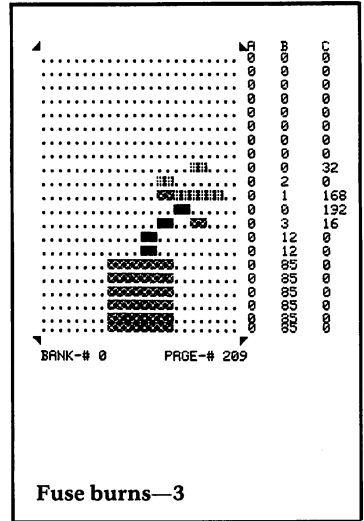
Firecracker Boy

PART 2

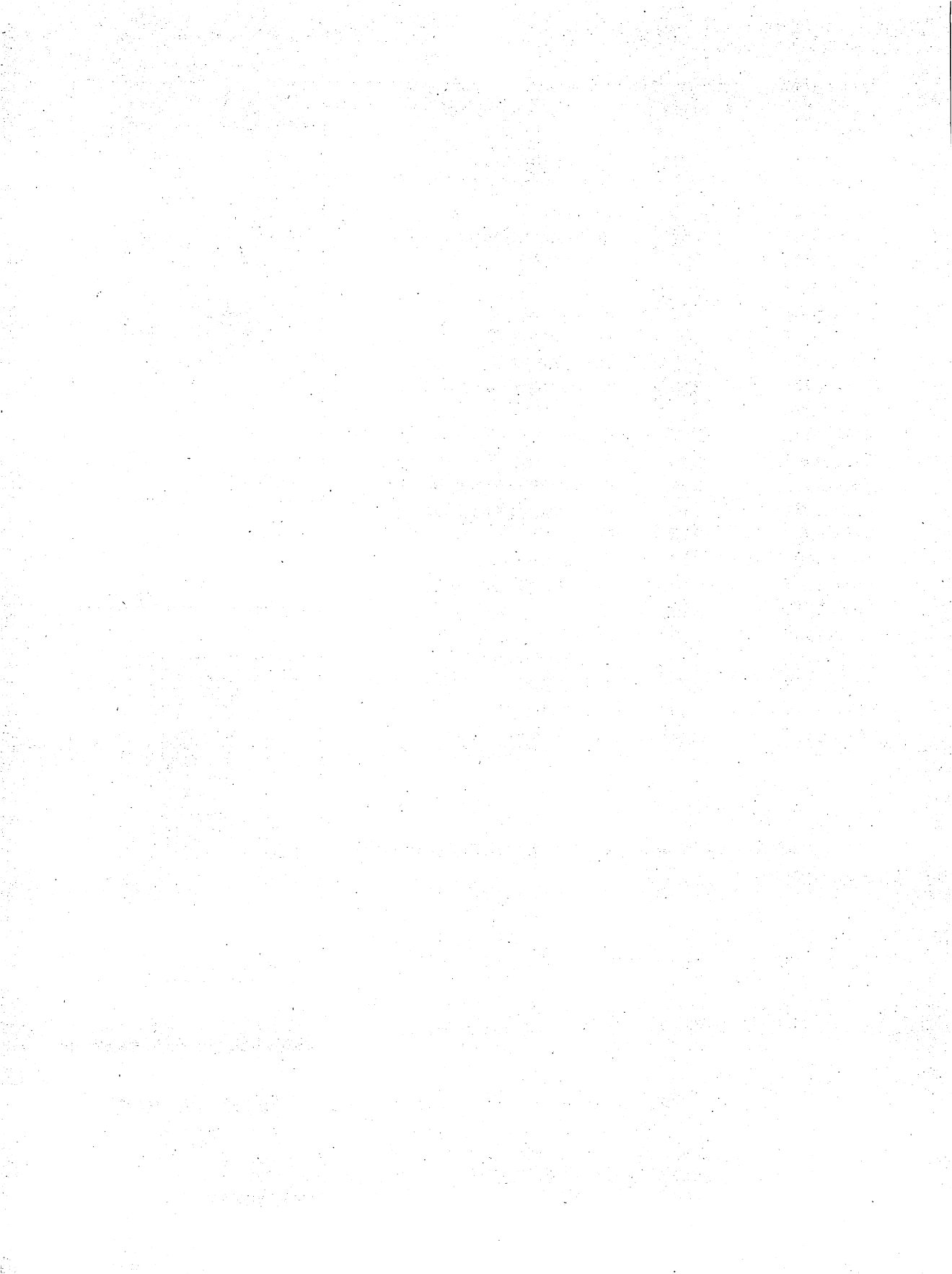
- 0050-0200 INITIALIZE VARIABLES
Set up joysticks and flag arrays. Initialize sprites.
- 0240-0530 MAIN GAME LOOP
Move boy and drop off firecracker at a random spot. Move firefighter, net, float, and water balloon. Check for balloon hitting firecracker. Check for end of game.
- 0570 READ JOYSTICK
- 0610 NEW BALLOON
Give a new balloon to firefighter: Turn off balloon sprite. Then, using firefighter's current position for new position of balloon, turn balloon sprite back on.
- 0650-0760 INITIALIZE SPRITES
Set position, color, and size.
- 0800-0960 GAME END
Firecracker burns down and explodes. Mayor does a flip.
- 1000-1240 PRINT GAME SCREEN
- 1280-1350 COLLISION DETECTION
Check for balloon hitting net or firecracker.
- 1410-1440 SOUND ROUTINES
Background sound, splash sound, and falling sound.
- 1480-1510 MOVE FLOAT ACROSS SEAM
Move float sprites past horizontal position 255.
- 1550-1590 PRINT SCORE
Speed up net, shorten fuse, and print score.



5032-5035	200	Mayor flip - 3
5036-5039	201	Mayor flip - 4
5040-5043	202	Net
5044-5047	203	Balloon
5048-5051	204	Balloon - popped
5052-5055	205	Firefighter
5056-5059	206	Firecracker
5060-5063	207	Fuse burns - 1
5064-5067	208	Fuse burns - 2
5068-5071	209	Fuse burns - 3
5072-5075	210	Fuse burns - 4
5076-5079	211	Fuse burns - 5
5080-5083	212	Fuse burns - 6
5084-5087	213	Fuse burns - 7
5088-5091	214	Firecracker explosion - 1
5092-5095	215	Firecracker explosion - 2
5096-5099	216	Door
5100-5103	217	Boy animation - 1
5104-5107	218	Boy animation - 2
5108-5111	219	Car
5112-5115	220	First float
5116-5119	221	Second float
5120-5123	222	Third float
5124-5127	223	Fourth float
5128-5131	224	Mayor float



Firecracker Boy



Into the Pot

Frank Mikulastik

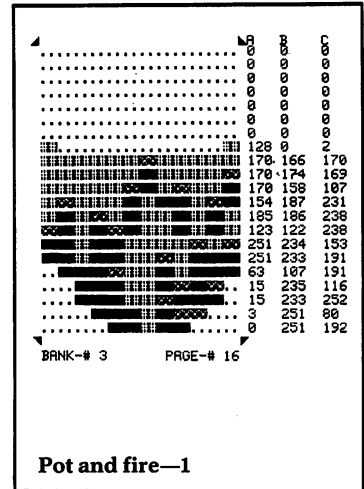
The cooking pot is boiling, but you haven't yet caught your dinner! You had planned to prepare your favorite recipe, which calls for two fat chickens, and two succulent-looking specimens are roaming around the rocky plain next to your camp. How quickly can you get them into your pot?

The blue blocks are heavy boulders which you cannot move, but you can push the lightweight patterned rocks around to create a path into the camp and steer the chickens toward the pot. Just in front of your camp there is a ready-made canyon entrance – if you can get them into the canyon, they're as good as in the pot!

Since it could take a while to round up the chickens, you might want to take a snack break and come back to the chicken chase a little later. Pressing the **F7** key will pause the game, saving your elapsed time. By entering **Y** to continue, you can pick up where you left off.

If you're not careful, you may trap yourself or a chicken in such a position that the game cannot possibly be won. In that case, pressing the **F7** key and entering **N** is the only way to quit and start a new game.

Load the game by typing **LOAD"INTO THE POT",8** and pressing the **RETURN** key. After about 20 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. After a delay of about 15 seconds, the title will flash on the screen. Press the fire button to proceed to the instruction screen. After you win a game, pressing the fire button will return you to the instruction screen.



Into the Pot

Programmer's Notes

Variables with Fixed Values. Experienced programmers often use variables to represent numbers that are fixed in the program and are not expected to change. One reason is to plan ahead for future program modifications. Although a particular number will never change in the current version of the program, assigning it a variable name makes it easier to find and change in future versions.

When a value is used in many different places in the program, assigning it to a variable name is even more important. Changing the value of a variable in one place in the program is simple, compared with the chore of tracking down and changing a numeric value which occurs throughout the program.

In **Into the Pot**, the number of blocks the cook can push at one time is set with the statement `NM = 2` in line 780. The variable `NM` does not change values as the game is played, and it is used only once, in line 1130. The statement `FORI = 1TONM` in line 1130 could have read `FORI = 1TO2` with the same result. You might decide later, however, to modify the game by enabling the cook to push a bigger stack of rocks as the game progresses. Having already defined the variable `NM` would simplify that change. The simplification would be even greater for a variable that is used several times throughout the program.

Often variable names are used simply to make the program instructions more meaningful. For example, it is common practice to include the instruction `V = 53248` to define the variable `V` as the base address of the VIC chip (53248). Then it can be referenced as `V`, rather than as `53248`.

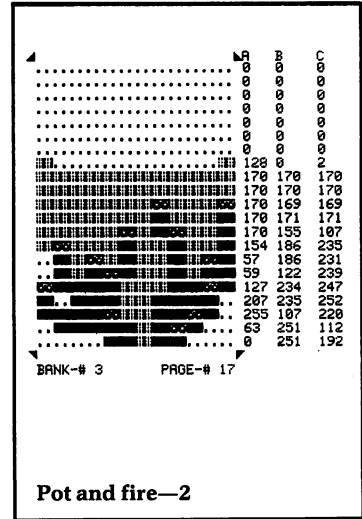
Into the Pot

Hints for Modifying the Game

To have the cook character start in a different position, change the number assigned to `P1` in line 700. A value of 0 would place it at the upper left-hand corner of the screen, and a value of 1000 would place it at the bottom right-hand corner. The starting positions of the chickens can be

changed in a similar way by assigning different values to **P2(1)** and **P2(2)** in line 720.

Try adding more chickens to the game. For each additional chicken, add a statement to line 720, setting the next element in the **P2()** array to the new chicken's desired starting position. For example, use **P2(3)** to establish the third chicken's initial position. Below line 740, add a line similar to lines 730 and 740 but using **P2(3)**. Then change the value of **NU**, set in line 780, to the new total number of chickens.



Major Routines

0013-0350 INITIALIZATION

Set up new character set. Set VIC memory to bank 3. Turn on new character set. Turn on multicolor character mode. Initialize variables. Print title screen and instruction screen.

0400-0420 START GAME

Print play screen. Initialize cook, chickens, and timer.

0500-0590 MAIN LOOP

Move cook and make a sound. Move chicken. Check to see if in no-win situation. If so, save timer and do NO WIN SITUATION. Repeat loop.

0700-0790 INITIALIZE CHARACTERS

0900-0980 PRINT PLAY SCREEN

Print borders, blocks and box. Blank out blocks in box. Print timer.

1000-1290 MOVE COOK

Read joystick. Set cook's direction accordingly and see if move possible. If no movement requested, return to MAIN LOOP. If location in front of cook is blank, move cook. Otherwise, return to MAIN LOOP. If cook is in front of a block, check to see if it can be moved. If it can, move block and cook, and increment the animation counter.

- 1500-1690 **MOVE CHICKEN**
Get a random number between 1 and 4 for direction. If direction selected in front of chicken is blank, move it and increment its animation counter.
- 2000-2091 **WIN SITUATION**
Make a sound. Check for fire button pressed to start a new game. If fire button pressed, print instruction screen and start a new game.
- 2100-2190 **NO-WIN SITUATION**
Check for keyboard input of Y or N to stop game or not. If game is to be stopped, print instruction screen to start another game. If game is to continue, reset timer and return to MAIN LOOP.
- 3000-3060 **PRINT TITLE SCREEN**
Make a sound and clear the screen. Switch character colors between red and yellow, while making a sound and checking for fire button to continue.
- 3100-3146 **PRINT INSTRUCTION SCREEN**
Clear screen and set background and border colors and make a sound. Wait for F1 or F7 to be pressed. If F1, continue with game. If F7, stop game.
- 4000-4048 **SOUND ROUTINES**
Five separate sound effects.
- 4100-4220 **CHARACTER DATA**
One redefined character per line.
- 5140-5400 **SPRITE DATA**

Into The Pot

Major Variables

Memory Pointers:

- V Base address of VIC chip
S1 Base address of SID chip

CM Base address of color memory
 SM Base address of character memory
 U Base address of sprite memory
 M1 Base address of sprite data

Cook:

S() Animation array
 N1 Animation counter
 P1 Position Counter
 T1 X direction movement
 T2 Y direction movement

Chickens:

S2() Animation array
 N2() Animation counters
 P2() Positions
 BN() Directions to move
 NU Number of chickens

Pot:

SP Animation counter

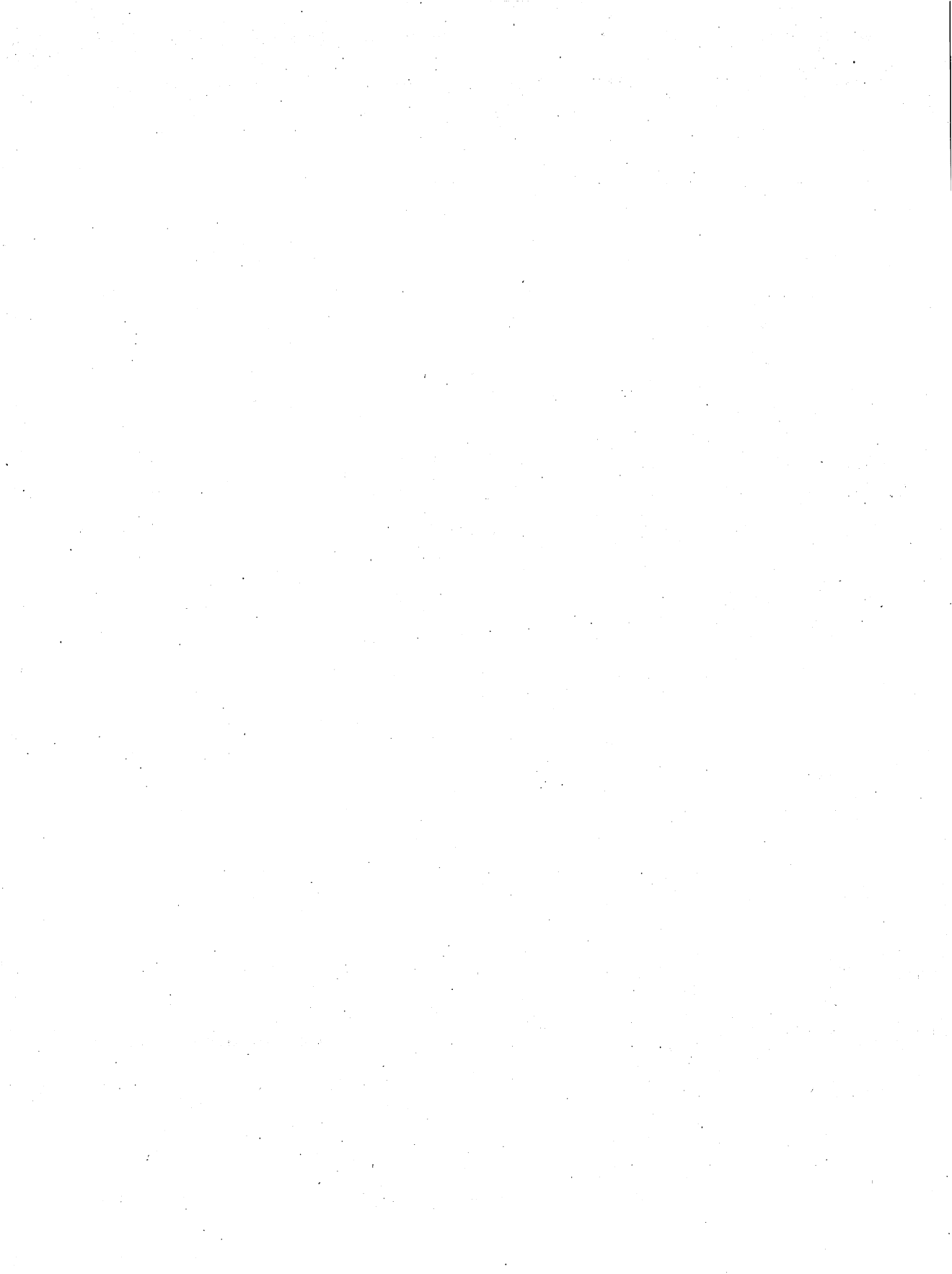
Sprites

(NOTE: All sprites are in bank 3)

Line Numbers	Page	Description
5140-5270	16	Pot and fire - 1
5280-5400	17	Pot and fire - 2

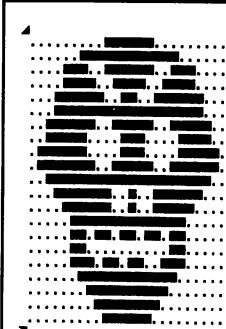
Redefined Characters

Line Number	Description
4100	Chicken - 1
4110	Chicken - 2
4120	Chicken - 3
4150	Cook - 1
4160	Cook - 2
4170	Cook - 3
4200	Block pattern - 1
4210	Block pattern - 2
4220	Block pattern - 3



Yorick's Revenge

Igor Tulchinsky



A	B	C
9	126	0
3	233	192
14	126	112
15	60	240
31	153	248
31	255	248
63	62	124
62	28	68
126	28	62
127	62	126
63	255	252
31	201	248
15	201	240
7	255	224
6	219	96
6	0	96
7	108	224
3	255	192
1	255	128
0	255	0
0	126	0

BANK-# 0 PAGE-# 199

Skull

Yorick's Revenge is a video game patterned after the action of a pinball machine. The object is to toss the ball into the high scoring areas, using the flippers located at the bottom of the screen, and to keep the ball in play for as long as possible.

Why is the game called **Yorick's Revenge**? When the ball crosses over the skull of Yorick, in the center of the screen, it departs in a new direction which appears to be random. But actually, as your score increases, Yorick will direct the ball closer to the gutter between the flippers, making it more and more difficult to keep the ball in play. Yorick will get his revenge!

The flippers are controlled by pressing any key on the keyboard. When the flippers are down, the ball bounces off them as if it had hit the bottom of the screen. When the flippers are up, the direction and speed of the ball's bounce depends on which part of the flipper hits the ball, as well as the ball's previous speed and direction.

Points are scored when the ball passes over the 100 and 500 point markers. The ball is lost if it drops in the gutter between the flippers. When three balls are lost, the game is over. Pressing the fire button will then start a new game.

Load the game by typing **LOAD"YORICK'S**

Yorick's Revenge

Hints for Modifying the Game

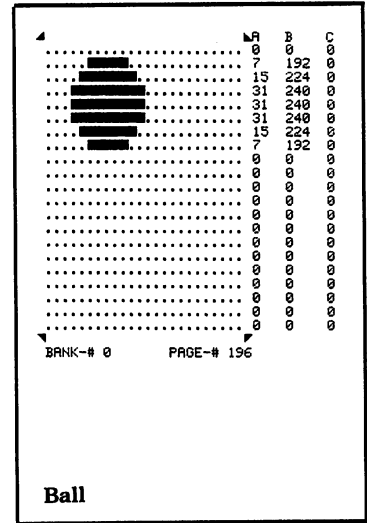
To make **Yorick's Revenge** more challenging by decreasing the size of the ball, change line 20070 to read:

```
20070 POKE V + 29,251:POKE V + 23,120:POKE V + 16,0
```

This will turn off vertical and horizontal expansion for the ball sprite. Or instead, test the score in the POINTS COLLISION routine, and make the ball become smaller when a certain score is reached. Similar approaches can be used on all the other sprites.

The force of gravity controlling the ball can be changed by adjusting the amount added to DY in line 42.

You could add more interest by making Yorick and/or the 100 and 500 point images move around on the screen. The instructions to make them move should be placed in the scoring routine or the sprite collision routines, to minimize loss of ball speed.



Major Routines

- 00010-00016 **INITIALIZATION**
Initialize all variables and prepare for new game.
- 00020-00051 **CONTROL LOOP**
Check to see if a key is pressed, and choose appropriate sprites for flippers. Advance ball and test if collision has occurred. If no collision, go back to beginning of loop.
- 00060-00140 **SPRITE-BACKGROUND COLLISION**
Determine whether ball has hit a bumper by checking if it is adjacent to a non-space character. If so, sound beep and bounce ball.
- 00150-00156 **SKULL COLLISION**
Sound hiss, then choose new direction for ball based on score.

**Yorick's
Revenge**

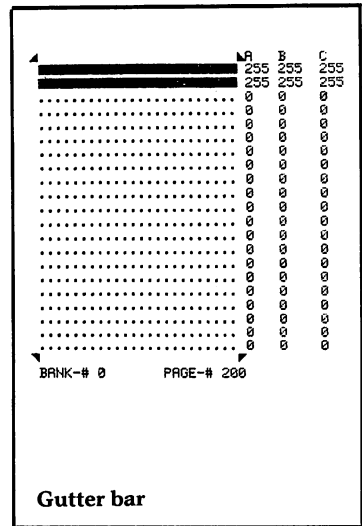
Major Variables

General:

SC	Score
NL	Number of balls left
S\$	String value of SC
N\$	String value of NL
F	Flag
Z	Sprite collision flag
V	Base address of VIC chip

Ball:

X	Horizontal pixel coordinate
Y	Vertical pixel coordinate
DX	Horizontal movement in pixels
DY	Vertical movement in pixels
T	Angular direction in radians
L	Speed



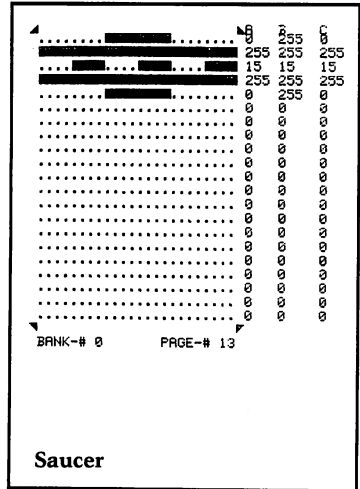
Sprites

Line Numbers	Page	Description
10000-10006	192	Left flipper - Down
10010-10016	193	Right flipper - Down
10020-10026	194	Left flipper - Up
10030-10036	195	Right flipper - Up
10040-10046	196	Ball
10050-10056	197	100 points
10060-10066	198	500 points
10070-10076	199	Skull
10080-10086	200	Gutter bar



Space Crash

Charles Mott, Jr.



Swarms of purple aliens are invading your planet, appearing without warning from hyper-space and floating to the ground. You must hold them off for as long as possible by intercepting and destroying them with your saucer. Each alien hit is worth ten points. When thirty-two aliens have landed, they will have taken over the planet by filling up the landscape, and the game will end.

If a falling alien encounters a tree or other object, it will be destroyed, taking with it whatever it hit. Eventually, this will clear the way for other aliens to reach the ground safely. To maximize your score, try especially hard to intercept those whose paths have already been cleared.

You will have maximum maneuvering room if you stay high in the sky, but this approach can be dangerous. The aliens are appearing from hyper-space in a band across the top half of the screen. If one comes in at the same spot where your saucer is at that moment, the saucer will be destroyed. You can also lose the saucer by flying it into the ground as you try to catch an alien low on the screen. Five saucers are available for your use. When they are all destroyed, the game is over.

Control the saucer with the cursor movement keys. Pressing the **CRSR- ↓** key will move the saucer down,

**Space
Crash**

while **SHIFT-CRSR-↑** will move it up. To move it left and right, use **CRSR-←** and **SHIFT-CRSR-→**.

Load the game by typing **LOAD"SPACE CRASH",8** and pressing the **RETURN** key. After about 20 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. The title screen will be displayed for about 30 seconds, before the saucer flies across the screen to begin the game.

Programmer's Notes

Redefined Characters. None of the graphics shapes available among the standard Commodore 64 graphics characters were suitable for some of the objects in this game, so special graphics shapes were customized by use of redefined characters. Setting up redefined or custom characters in a program involves many programming steps.

Every character or graphic symbol displayed on the screen is defined by a series of eight bytes. The eight-byte definitions of the C-64's standard character set are stored in the computer's read-only memory (ROM). To redefine them, first you must copy the regular character set into an area of memory where it can be modified. Then for each character that is to be redefined, the eight bytes in the appropriate place in the relocated character set must be replaced with other bytes that will create the desired new shape.

The method used in **Space Crash** to make the replacements can be a timesaver and can help maintain flexibility for future changes. The bytes used to redefine the characters are in **DATA** statements beginning at line 1380. Each line contains nine numbers, but only the last eight are used for character redefinition. The first number is the relative position, within the character set, of the character being replaced. (See appendix E of the *Commodore 64 User's Guide* for the numeric position of each character.)

Lines 310 and 320 actually do the redefinition, storing the last eight numbers in each **DATA** statement into the position in the relocated character set that was indi-

Space Crash

cated by the first number. The last DATA statement contains only a -1, which indicates the end of the redefinition data. In line 310, the statement `IFA = -1 THEN330` stops the moving of data when a -1 is found.

Using this method, new lines of DATA statements for new custom characters can be added or deleted, with no concern for their position within the other DATA statements, and existing DATA statements can be reassigned to different characters. The only requirement is that the last DATA statement contain a -1.

This is just a summary of the method used for character redefinition in this game. To learn all the details of the procedure, you should consult a book on C-64 graphics programming.

Hints for Modifying the Game

The total number of aliens on the screen at once is controlled by the variable `NA`, set to 5 in line 40. Changing the 5 to a larger number will increase the difficulty of the game. Or instead, increment `NA` as the score increases. Too large a number for `NA`, however, will make the game move very slowly.

The saucer's speed in all four directions is set in lines 520-550. The size of the saucer's moves, for any cursor key pressed, is set to 8 pixel units for each cursor key pressed. Change some or all of these 8's to make the saucer move at different speeds when it goes in different directions. For example, the saucer could move down quickly but rise up very slowly. In order for the saucer/alien collision routines to work correctly, all values must be multiples of 4.

Currently, no matter how many aliens are prevented from landing, more will keep appearing. You might want to consider having a fixed number of aliens to contend with, and award a bonus if they are stopped successfully.

If you prefer to control the saucer with a joystick instead of the cursor keys, change lines 520-550, using lines 1000-1050 in the game **Into the Pot** as a model.

**Space
Crash**

Major Routines

- 0030-0336 **INITIALIZATION**
Initialize variables and sound. Print title screen. Set color and position of saucer sprite and load its data into memory, setting unused bytes in the page to 0. Lower top of BASIC memory. Load new character set if not already present. Do title animation and print game screen.
- 0370-0380 **SPIN SAUCER**
Make saucer sound. Update sprite memory with new shape.
- 0510-0570 **MAIN LOOP**
Check for keyboard input and adjust saucer's position. Descend an alien and spin saucer. Check for saucer hitting ground or an alien. Move saucer.
- 0660-0711 **SAUCER CRASH**
Lower saucer as its shape data in memory is being randomly changed. Re-initialize saucer and check for end of game.
- 0740-0775 **SAUCER HIT ALIEN**
Determine which alien was hit. Erase alien and adjust score.
- 0780-0830 **NEW ALIEN APPEARS**
Pick random position, set position variables, and display alien.
- 0850-0890 **ALIEN DESCENDS**
If object at ground level was hit, erase it. If ground was hit, display landed alien and update score. Check for end of game. Make new alien appear.
- 0910-1120 **PRINT GAME SCREEN**
- 1160-1280 **PRINT TITLE SCREEN**

**Space
Crash**

- 1300-1315 END OF GAME
Move saucer and print game over message.
- 1330 SPRITE DATA
- 1380-1670 CHARACTER DATA

Major Variables

General:

- V Base address of VIC chip
 SO Base address of SID chip
 SC Base address of screen memory
 CL Base address of color memory
 BA Base address of relocated character set
 BR Address of border color
 BK Address of background color
 SR Score
 XP\$ String to position cursor on X axis
 YP\$ String to position cursor on Y axis
 IN Address for line of blocks at bottom of screen
 BL Blocks left to fill

Saucer:

- FS() Addresses of sprite data to change for spin
 GA Value for spin - 1
 AG Value for spin - 2
 U Pixels moved in Y direction
 VE Pixels moved in X direction
 OM\$ Saucer shape for score
 OM Pointer to score saucers

Aliens:

- NA Number allowed on screen at one time
 AA\$ First appearance shape
 WA() Redefined characters for animation
 AC Animation counter
 A() Current position in screen memory
 B() Equivalent X positions in pixels
 C() Equivalent Y positions in pixels
 RX Random X position
 RY Random Y position

**Space
Crash**

Sprites

Line Number	Page	Description
1330	13	Saucer

Redefined Characters

Line Number	Description
1380	Solid block
1500	Top of tree - 1
1510	Top of tree - 2
1520	Asterisk
1530	Alien - 1
1540	Alien - 2
1560	Alien on ground
1570	Alien - 3
1580	Alien - 4
1640	Score saucer - 1
1650	Score saucer - 2
1660	Score saucer - 3

Space Crash

The Blobbis

Frank Mikulastik

Equipped with the latest and most fashionable fighting tuxedo and shooting hat, you are out to break through the force field that protects three evil Blobbis. The force field has five levels, and each shot from your top hat will destroy a section of one level.

The different force fields move in opposite directions, so when you shoot through one and create a hole in the next one, the two holes will soon drift apart. Opening up a hole through all five levels just as a Blobbis passes over requires clever strategy!

Each Blobbis has a different method of preventing you from penetrating the force field. The most dangerous is the Hunchback Blobbis, who drops a deadly pulsar from the center of the screen on each trip across the screen. If it hits you, it's all over! The Bug-eyed Blobbis frustrates you by rebuilding the top two layers of the force field on each trip. And the Flashing Blobbis makes a confusing and hypnotizing display.

Hitting any one of the Blobbis will win the game for you. As your strategy improves, you should be able to do it with fewer shots. Use the joystick to move the player sprite, and press the fire button to fire your hat.

A screenshot of the game 'The Blobbis' showing a score table. The table has three columns labeled 'B', 'E', and 'C'. The rows represent different levels or items, with scores listed in the 'B', 'E', and 'C' columns. The scores are: 0 48 0, 0 170 0, 2 170 128, 10 255 160, 11 235 224, 43 125 224, 171 190 224, 171 255 232, 170 195 170, 170 190 170, 250 170 170, 10 170 170, 10 170 175, 10 170 160, 10 170 160, 10 170 160, 2 130 128, 2 130 128, 10 130 160, 10 130 160. Below the table, it says 'BANK-# 2' and 'PAGE-# 20'. At the bottom, it says 'Hunchback Blobbis Walks - 1'.

	B	E	C
.....	0	48	0
.....	0	170	0
.....	2	170	128
.....	10	255	160
.....	11	235	224
.....	43	125	224
.....	171	190	224
.....	171	255	232
.....	170	195	170
.....	170	190	170
.....	250	170	170
.....	10	170	170
.....	10	170	175
.....	10	170	160
.....	10	170	160
.....	10	170	160
.....	2	130	128
.....	2	130	128
.....	10	130	160
.....	10	130	160

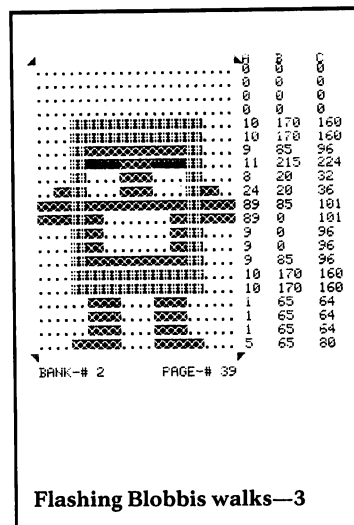
BANK-# 2 PAGE-# 20

Hunchback Blobbis Walks - 1

The Blobbis

sprite #0, or $M1+1$. At all times, that location will contain the page number to be used for the next Blobbis to be displayed.

Now look at lines 4150 and 4160 in the MOVE BUG-EYED BLOBBIS section. First $N3$, the current sequence step counter, is incremented. If it becomes larger than $B3$, the sequence element counter, it is reset to 1 to begin a repeat of the sequence. Then sprite pointer location $M1+1$ is loaded with the contents of the element of the animation array indicated by the current value of $N3$. $M1+1$ will now contain the page number of the next sprite in the sequence. Each time these instructions are executed, the next sprite in the walking sequence for the Bug-eyed Blobbis is displayed.



Hints for Modifying the Game

The speed of each figure in the game can be changed in the following way: The position of each Blobbis is incremented by 6 pixels at the beginnings of lines 4010, 4120, and 4210. Change the +6 in these lines to other values to make them move more quickly or slowly. Lines 4000, 4100, and 4200 determine when each Blobbis does its special trick by monitoring its location. The amount of movement for the player sprite is set to +9 or -9 in lines 1040 and 1050, depending on the position of the joystick. Change these values for a faster or slower player sprite.

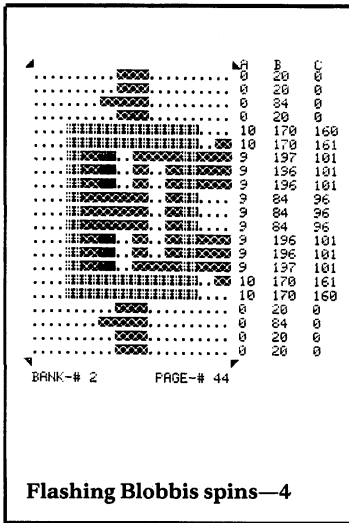
Try adding an additional Blobbis. Using lines 4100-4190 as a model, set up new animation arrays and major variables. You might consider having your new Blobbis rebuild different rows of the force field.

Major Routines

0010-0430 INITIALIZATION

Set up memory variables. Set VIC memory to bank 2. Read sprite data and make sound. Set up sprite animation arrays. Print title and instruction screens. Build force field and initialize sprites.

The Blobbis

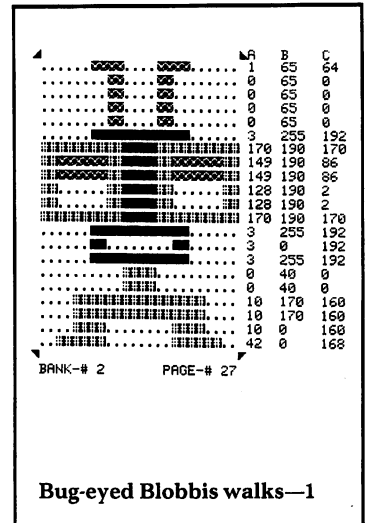


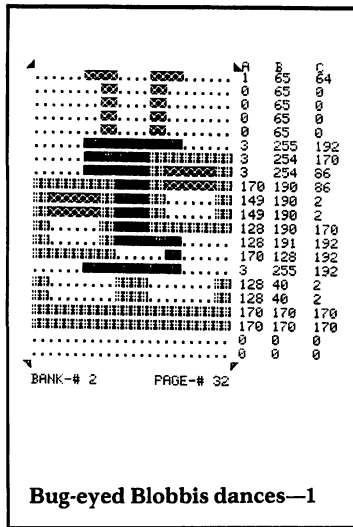
- 0500-0590 **MAIN LOOP - NOT SHOOTING**
Move player and current Blobbis. Check for game over. Select next force field row and move it.
- 0700-0790 **INITIALIZE SPRITES**
Set starting positions.
- 0800-0860 **BUILD FORCE FIELD**
- 0865-0890 **BUILD FLOOR**
- 1000-1090 **PLAYER MOVEMENT DRIVER**
Read joystick. If fire button pressed, begin MAIN LOOP - SHOOTING. If joystick moved, do MOVE PLAYER.
- 1100-1190 **MOVE PLAYER**
Change pointer to animation array. If player not at edge of screen, move player.
- 2000-2090 **MOVE ROW TO THE RIGHT**
Move one row of the force field to the right.
- 2100-2190 **MOVE ROW TO THE LEFT**
Move one row of the force field to the left.
- 3000-3090 **MAIN LOOP - SHOOTING**
Main in control loop for player shooting. Fire the bullet and move it until it hits something. Move a Blobbis and check for its being hit.
- 4000-4090 **MOVE HUNCHBACK BLOBBIS**
Do special action if in correct position. Change pointer to animation array. If at edge of screen, select next Blobbis.
- 4100-4190 **MOVE BUG-EYED BLOBBIS**
Do special action if in correct position. Change pointer to animation array. If at edge of screen, select next Blobbis.
- 4200-4290 **MOVE FLASHING BLOBBIS**
Do special action if in correct position.

The Blobbis

Change pointer to animation array. If at edge of screen, select next Blobbis.

- 5000-5091 PRINT TITLE SCREEN
- 5100-5180 PRINT INSTRUCTION SCREEN
- 5200-5290 GAME OVER
Make a sound while waiting for fire button to start a new game.
- 6000-6004 SCREEN CHANGE SOUND
- 6010-6013 BLOBBIS MOVEMENT SOUND
- 6020-6026 INITIAL BULLET SOUND
- 6030-6036 A BLOBBIS IS HIT
Flash and make a sound.
- 6040-6046 FORCE FIELD SOUND
- 6050-6056 PLAYER WALKING SOUND
- 6060-6069 MOVING BULLET SOUND
- 6080-6089 GAME OVER SOUND
- 6100-6119 BUG-EYE ACTION
Dances and fills in top 2 rows of force field.
- 6200-6290 FLASHER ACTION
Flashes the screen and does some flips.
- 6300-6390 HUNCHBACK ACTION
Drops the pulsar and does a dance. Check for pulsar hitting player.
- 6400-6490 PULSAR HITS PLAYER
Player flashes and sound.
- 7000-12310 SPRITE DATA





Major Variables

Memory Pointers:

- V Base address of VIC chip
- S1 Base address of SID chip
- M1 Base address of sprite pointers
- SM Base address of screen memory
- CM Base address of color memory

Player:

- S1() Animation array for walking
- B1 Length of walk sequence
- D Direction of movement
- X X position
- N1 Animation counter

Hunchback Blobbis:

- S2() Animation array for walking
- B2 Length of walk sequence
- N2 Walk animation counter
- D2() Animation array for dance
- C2 Length of dance sequence
- N6 Dance animation counter
- P2 X position

Bug-Eyed Blobbis:

- S3() Animation array for walking
- B3 Length of walking sequence
- N3 Walk animation counter
- D3() Animation array for dance
- C3 Length of dance sequence
- P3 X position

Flashing Blobbis:

- S4() Animation array for walking
- B4 Length of walking sequence
- N4 Walk animation counter
- D4() Animation array for dance
- C4 Length of dance sequence
- P4 X position

Pulsar:

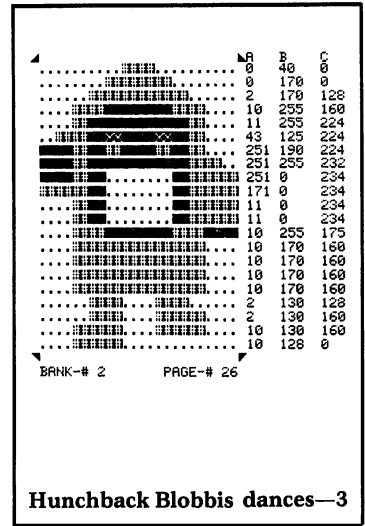
- S5() Animation array for pulsing
- B5 Length of pulse sequence

The Blobbis

N5 Animation counter
 P5 Y position

General:

WS Which Blobbis is on screen
 F Game status
 K Which row of force field to move
 BL Number of bullets fired

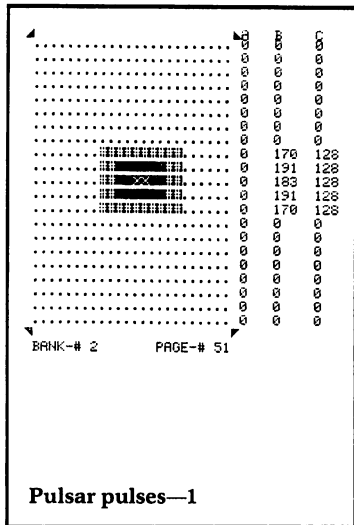


Sprites

(NOTE: All sprites are in bank 2)

Line Numbers	Page	Description
07550-07670	20	Hunchback Blobbis walks - 1
07680-07810	21	Hunchback Blobbis walks - 2
07820-07950	22	Hunchback Blobbis walks - 3
07960-08080	23	Hunchback Blobbis walks - 4
08090-08220	24	Hunchback Blobbis dances - 1
08230-08350	25	Hunchback Blobbis dances - 2
08360-08490	26	Hunchback Blobbis dances - 3
08500-08630	27	Bug-eyed Blobbis walks - 1
08640-08770	28	Bug-eyed Blobbis walks - 2
08780-08900	29	Bug-eyed Blobbis walks - 3
08910-09040	30	Bug-eyed Blobbis walks - 4
09050-09170	31	Bug-eyed Blobbis walks - 5
09180-09310	32	Bug-eyed Blobbis dances - 1
09320-09450	33	Bug-eyed Blobbis dances - 2
09460-09580	34	Bug-eyed Blobbis dances - 3
09590-09720	35	Bug-eyed Blobbis dances - 4
09730-09850	36	Bug-eyed Blobbis dances - 5
10000-10130	37	Flashing Blobbis walks - 1
10140-10270	38	Flashing Blobbis walks - 2
10280-10400	39	Flashing Blobbis walks - 3
10410-10540	40	Flashing Blobbis walks - 4
10550-10670	41	Flashing Blobbis spins - 1
10680-10810	42	Flashing Blobbis spins - 2
10820-10950	43	Flashing Blobbis spins - 3
10960-11080	44	Flashing Blobbis spins - 4
11090-11220	45	Flashing Blobbis spins - 5
11230-11350	46	Flashing Blobbis spins - 6
11360-11490	47	Player walks right - 1

**The
 Blobbis**

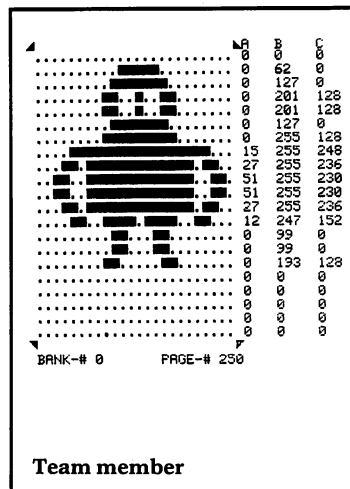


11500-11630	48	Player walks right - 2
11640-11760	49	Player walks left - 1
11770-11900	50	Player walks left - 2
11910-12030	51	Pulsar pulses - 1
12040-12170	52	Pulsar pulses - 2
12180-12310	53	Pulsar pulses - 3

The Blobbis

Sound the Whistle

Igor Tulchinsky



Sound the Whistle is a two-player game similar to football or soccer, but without a ball. The game requires strategy and quick response to the opposing player's actions. Each player controls a four-member team. The object is to score points by running one of your team members through the opponent's goal line and off the edge of the screen, while preventing a member of the opposing team from running past your own goal line. If you run off your own side of the screen, you earn a point for the other team. The first team to score ten points wins the game.

The currently selected team member, or runner, is displayed in a different color than the rest of the team, and is moved with the joystick in the desired direction of movement. The runner can be advanced toward the goal line or used to block the other team's runner. To change runners, hold down the fire button and point the joystick either up, down, left, or right. Each direction corresponds to a particular team member. If a runner bumps into another team member, the runner will be blocked and forced back.

Load the game by typing in **LOAD" SOUND WHISTLE",8** and pressing the **RETURN** key. After about 20 seconds, the computer will say **READY**, indi-

Sound the Whistle

cating that the game is loaded. Then type **RUN** and press the **RETURN** key.

Programmer's Notes

Considerations in Creating Sounds. Because sound is one of the basic means of communicating with a computer user, it is a very important element of most games. Clever use of sound effects can add interest to any game and improve even the best ones.

The sound chip in the Commodore-64 computer has three separate voices which may be used alone or together to achieve an amazing range of sound effects. However, synthesizing a particular sound can be very difficult.

Imagine the sound you want to create, and think about its "color" or texture. Is it random, or does it follow a pattern? Is it buzzy or smooth? Is it shrill and piercing, or is it soft? Gunshots, explosions, and wind are noise-like sounds and may be reproduced quite well by the noise waveform. Piercing sounds are best produced by the sawtooth waveform. The triangle waveform is effective for soft sounds, while the pulse waveform is best for brassy sounds.

The C-64 provides a tremendous assortment of interrelated methods to control sounds, and experimentation is often required when the desired sound doesn't fit any easily defined category. Refer to the *Commodore 64 User's Guide* for an introduction to the many possibilities for sound and music creation. Varied and novel examples of sound creation can be found in all the games in this book. Other books dedicated to C-64 sound and music programming are available in bookstores.

Upper/Lower Case Mode. Lines 20220 through 21220 of the **Sound the Whistle** program listing contain some symbols that appear, at first glance, to be errors. Actually they are the upper case characters of text that was entered in upper/lower case mode. When this data is printed to the screen, it will be converted correctly to upper and lower case.

Sound the Whistle

Hints for Modifying the Game

The statement `FL%(I) = 3` in line 675 sets to 3 the distance a player bounces back after hitting another player. Change the 3 to a larger number for greater bounce.

You might want to modify the way the game is played, by making the following changes.

- 1) Make a new rule that all team members on the team must cross the goal line before a point may be scored.
- 2) Remove from the game any team member who has been blocked more than three times.
- 3) Make the selected team member's figure invisible by changing it to the same color as the field.
- 4) Make each team member a different color, to simplify selecting them.

Major Routines

- 00196-00209 **INITIALIZATION**
 Initialize sprites and sound. Blow whistle and display the title screen. Wait for a key to be pressed while flashing title in different colors. Display instructions and again wait for a key to be pressed. Reinitialize sprites.
- 00215 **MAIN LOOP**
 Alternate between each game player.
- 00500-00680 **MOVE TEAM MEMBERS**
 Read joystick for one player. If fire button is pressed, select appropriate team member. If block flag for that player is not set, read joystick for team member direction. If there is movement, make sound. Determine which sprite to use, and move it. If border has been hit, bounce back. If score has been made, print score and check for end of game. Check for sprite collisions and set block flags.

**Sound
the Whistle**

10000-10006	SPRITE DATA
12000-12050	PRINT BACKGROUND Print the play field.
13000-13150	INITIALIZE SPRITES Initialize positions and colors. Blow whistle.
15000-15070	END GAME Turn everything off and make game ending sound. Wait until either a fire button is pressed or 45 seconds have elapsed, then restart game.
16000-16010	UPDATE SCORE Print each player's current score.
17010-17030	SELECT TEAM MEMBER Choose team member and change its color.
18000-18020	INITIALIZE SOUND
19000-19010	SCORE SOUND
20000-20040	PRINT TITLE Print title and wait for key to be pressed.
21000-21250	PRINT INSTRUCTION SCREEN Print instructions and wait for key to be pressed.

Major Variables

General:

V	Base address of VIC chip
S	Base address of SID chip
JS%	Joystick status
S%()	Score
Q1%	Sprite collision flag
CQ%	Block counter
P%()	Powers of 2

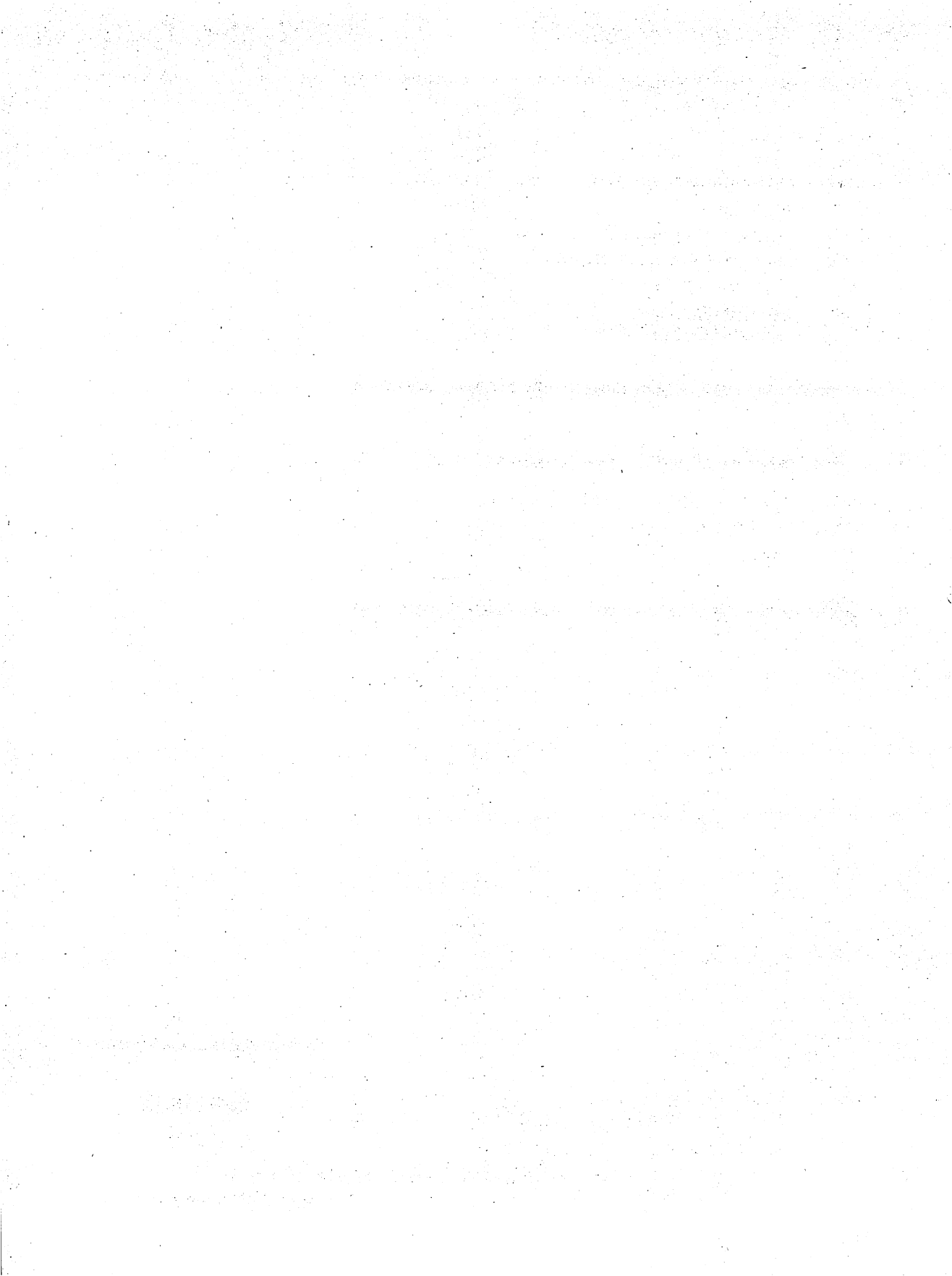
Sound the Whistle

Team members:

- CL%() Colors
- N%() Team member numbers
- FL%() Block flags
- DX%() Change in horizontal positions
- DY%() Change in vertical positions
- X%() Horizontal positions
- Y%() Vertical positions
- Z%() Sprite being moved

Sprites

Line Numbers	Page	Description
10000-10006	250	Team Member



Bumper Ball

Charles Mott, Jr.

A	B	C
3	255	192
7	255	224
15	255	240
31	255	248
63	255	252
127	255	254
255	255	255
255	255	255
255	255	255
255	255	255
255	255	255
255	255	255
255	255	255
255	255	255
127	255	254
63	255	252
31	255	248
15	255	240
7	255	224
3	255	192

BANK-# 0 PAGE-# 15

Ball and Bumper

Bumper Ball is a "pong" or handball-type game pitting one player against the computer. You score one point each time you hit the white ball, controlling the black paddle with the joystick. The computer gets one point each time you miss and let the ball go off the edge of the screen. To have the next ball served, press the fire button. The red bumper ball sometimes adds confusion by deflecting the white ball.

Hitting the ball into the white plunger on the right side of the screen will earn you a bonus of 30 points. Hitting the plunger, however, will freeze your paddle until the ball bounces back from the plunger and hits either the paddle or the red bumper ball, or is lost off the screen. If neither your paddle nor the red bumper ball is lined up to return the ball again, the computer will also get a 30-point bonus! Therefore it is good strategy always to return the paddle to the center of the screen after hitting the ball.

The white ball will start out moving slowly, but its speed will increase if your paddle is moving when it hits the ball. The speed of both the white ball and the red bumper ball are increased when bonus points are scored.

The game is over when you miss the fifth ball. The program will then subtract the computer's score from

Bumper Ball

yours to give you a final score. If you decide to quit while you still have balls left to play, press **F7** to get your total score. Then you can press **F7** to start a new game.

Load the game by typing **LOAD"BUMPER BALL",8** and pressing the **RETURN** key. After about 15 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. When the game screen displays the flashing message **YOUR SCORE**, press the fire button to have the first ball served.

Programmer's Notes

Programming for Speed. The **MAIN CONTROL LOOP** routine in this program is located at the beginning, where it will run most quickly. The initial setup routine, however, must be executed first. Therefore the **GOTO170** statement in line 10 jumps over the main loop to execute the setup routines, and then the **GOTO20** statement in line 1370 returns to execute the main loop. For further discussion on programming for speed, see the Programmer's Notes for the game **Yorick's Revenge**.

Use of Variables to Position Cursor. In programming screen displays, it is often necessary to position the cursor at certain locations on the screen in order to begin displaying at those locations with **PRINT** statements. This usually involves printing a series of **CURSOR-DOWN** and **CURSOR-RIGHT** commands, which appear in the program listing as special symbols, before printing what is to appear on the screen.

Bumper Ball accomplishes this without cluttering up the program with cursor symbols. In line 360, the variable **PY\$** is set to contain one **HOME-CURSOR** character code and twenty-five **CURSOR-DOWN**'s. In line 370, the variable **PX\$** is loaded with forty **CURSOR-RIGHT**'s. (The character codes for all keyboard commands are listed in appendix F of the *Commodore 64 User's Guide*.)

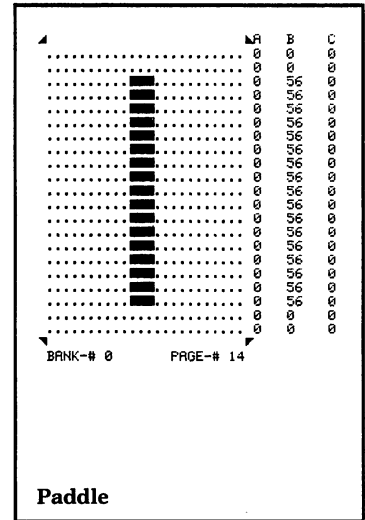
Calculations made elsewhere in the program then position the cursor within the C-64's twenty-five rows and forty columns. For example, when the ball hits the paddle, the player's updated score is displayed by the

Bumper Ball

PRINT statement in line 680. **LEFT\$(PY,6)** and **LEFT\$(PX,32)** cause the leftmost 6 characters of **PY\$** and the leftmost thirty-two characters of **PX\$** to be printed. The effect, therefore, is to position the cursor on row 6 at column 33. By simply looking at the **PRINT** instruction, you can easily tell where on the screen the displayed data will appear.

Complications in Collision Detection. When the game logic requires that collisions between two moving objects be detected, the size and speed of the objects always involve trade-offs. If the distance one object travels in each move is greater than the size of the other object, it is possible for them to appear to pass directly over one another without a collision being detected.

In **Bumper Ball**, for example, the ball's maximum move of 30 pixels may allow the ball to pass over the paddle occasionally. On the other hand, it does permit the game to build up to a fast pace. A lower maximum would permit more reliable collision detection but would make the game less exciting to play.



Hints for Modifying the Game

In implementing any of the following suggestions, bear in mind the preceding discussion regarding collision detection, and carefully observe the effects on the reliability of the collision detection as you experiment with different values for the variables.

The statements **POKEV + 23,0:POKEV + 29,0** in line 300 set the sizes of the paddle and ball sprites. For a larger red bumper ball, change both 0's to 4's. For a larger paddle, change the first 0 to 2. And to make both the bumper ball and the paddle larger, change the first 0 to 6 and the second 0 to 4.

As the white ball is accelerated, it is prevented from moving too quickly by the check **ABS(CM) < 30** in line 710. The ball is allowed to travel a maximum of 30 pixels per move. Experiment with different limits. A similar check for the speed of the red bumper ball is made in line 1170.

When the joystick is activated, the paddle travels up

Bumper Ball

or down the screen 20 pixels per move. For a paddle that is more or less responsive, adjust the + 20 in line 30 and the - 20 in line 40.

The white ball's initial horizontal and vertical speed are controlled by the variables **BM** and **CM**, respectively, which are set in line 220. The ratio of these two numbers determines the angle of bounce. Experiment with other values for these two variables.

The numbers after the **IFCX**> . . . portion of line 130 are the horizontal and vertical pixel coordinates of the range over which the red bumper ball can move. To alter the bumper ball's boundaries, change those numbers.

Since the main loop has to be as efficient as possible to maintain adequate speed, we suggest that any changes be limited to variable values, sprite sizes, and features outside the main loop.

Major Routines

- 0020-0150 **MAIN CONTROL LOOP**
Read joystick and move paddle. Check for ball hitting bonus plunger or being missed by paddle. Move ball and bumper ball. Bounce ball if it hits wall, bumper ball, or paddle.
- 0170-0200 **PRINT TITLE SCREEN**
- 0220-0370 **INITIALIZATION**
Initialize variables and load sprites into memory while flashing title screen and making sounds.
- 0390-0660 **PRINT GAME SCREEN**
Initialize game screen variables and print screen.
- 0680-0720 **BALL HIT PADDLE**
Update and print score. If paddle was moving, speed up white ball.
- 0740-0790 **BALL MISSED PADDLE**
Update and print computer score. Make sound and flash **COMPUTER SCORE** block.

Bumper Ball

- 0810 **RANDOM NOTE**
Sound a note of random frequency.
- 0840-0920 **PREPARE FOR NEXT BALL**
Check for no balls left to play. Sound random notes and flash **YOUR SCORE** block while waiting for fire button or F7 to be pressed.
- 0940-1020 **SPRITE DATA**
- 1040-1150 **BALL HIT PLUNGER**
Speed up ball and bumper ball. Capture ball with plunger. Sound random notes and flash **BUMPER BALL** block. Add and print bonus points one at a time. Release next ball.
- 1170-1180 **SPEED UP BUMPER BALL**
Check maximum speed before incrementing.
- 1200-1270 **PRINT FINAL SCORE**
Sound random notes as the difference between your score and computer score is calculated and printed. Run game again if F7 is pressed.
- 1290-1380 **RELEASE NEXT BALL**
Pick random spot for bumper ball in front of plunger. Move plunger in and out to deliver next ball into play. Move ball across screen until it hits bumper ball. Continue normal play.

Major Variables

General:

- JS Address of joystick port
JV Value at joystick port
V Base address of VIC chip
SO Base address of SID chip
CL Base address of color memory
SC Base address of screen memory
BR Address of border color

**Bumper
Ball**

BK Address of background color
MY Y position of paddle
MS Computer score
GG Your score
PO Points to add to score
PY\$ String of down cursors
PX\$ String of right cursors
BL\$() Balls left to play strings
XX Pointer to BL\$()

Ball:

BX X position
BY Y position
BM Pixels moved in X direction
CM Pixels moved in Y direction

Bumper ball:

CY Y position
CX X position
MC Pixels moved

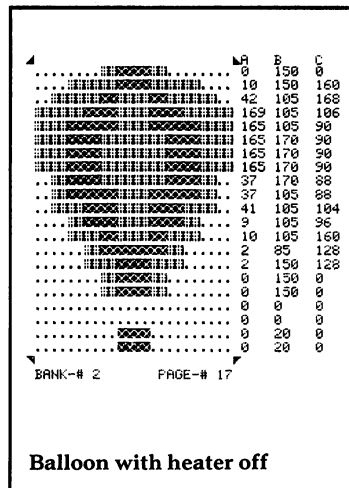
Sprites

Line Numbers	Page	Description
0940-0970	14	Paddle
0990-1020	15	Ball and Bumper

Bumper Ball

Ride the Wind

Frank Mikulastik



As a hot air balloon test pilot, your assignment is to test an experimental balloon by safely landing it on each of the eight landing pads at the balloon company's testing grounds.

The balloon's left and right movement is determined by tricky wind currents, which you read from the motion of the clouds. Your only control is the balloon's heater, activated by pressing the space bar. This causes an upward thrust, counteracting the downward pull of gravity and causing the balloon to rise. Using the heater to move up and down through the wind currents, you can control the sideways motion.

Points are scored by landing gently on each pad. Touching down at too great a descent speed will result in a crash. You must complete a round of testing by landing on each of the eight pads once, before you can receive more points by landing on any of them a second time. A color bar at the top left of the screen indicates the pads on which you have successfully landed. The game ends when you have crashed three times.

You will probably have to play this game several times before you begin to make successful landings. Becoming a skillful balloon test pilot requires persistence!

Ride the Wind

Hints for Modifying the Game

You can make **Ride the Wind** more or less difficult by changing the gravity, wind, heater lift values, and number of balloons. In the statement $Y1 = Y1 + 1.5$ in line 1010, change the 1.5 to increase or decrease the force of gravity. In the statement $Y1 = Y1 - 3$ in line 1020, change the 3 to increase or decrease the heater lift amount. To change the wind velocities, represented by variable **Q** in line 1050, change the 2.5 to multiply the sine function by a different value. For example, if you multiplied by 5.0, the winds would be twice as strong at each level. In line 780, set **M** to a value other than 3 to allow more or fewer crashes before the game ends.

Addition of a penalty for colliding with a cloud would be another way to add interest to this game. You might also want to use a different shape of sprite for the clouds, perhaps one which would be easier or harder to avoid with the balloon.

Major Routines

- 0009-0430 INITIALIZATION
Set VIC memory to bank 2, initialize variables, and load sprite data into memory. Print title, instruction, and game screens. Set up balloon and cloud sprites, and heater sound.
- 0500-0520 CONTROL LOOP
Move balloon and clouds. Check for good landing or balloon crash.
- 0700-0790 SET UP BALLOON AND CLOUDS
Initialize sprite pointers.
- 0800-0890 SET UP HEATER SOUNDS
- 0900-0990 PRINT GAME SCREEN
- 1000-1190 MOVE BALLOON
Add gravity pull. If space bar pressed, select balloon sprite with heater, add heater lift

**Ride
the Wind**

force, and set heater sound volume. If space bar not pressed, select regular balloon sprite. Calculate and add wind force to balloon's motion.

- 2000-2090 **MOVE CLOUDS**
Calculate new position for each cloud. If a cloud moves off edge of screen, reposition it to other side.
- 3000-3090 **COLLISION DETECTION**
If balloon hit edge of screen or platform supports, make explosion and sound. If more balloons to play, begin again. Otherwise, end game.
- 3100-3190 **LANDING DETECTION**
Check to see if position of balloon is same as a landing pad.
- 3200-3290 **LANDED**
If pad has not been landed on before, do cloud spin and sound.
- 4000-4190 **PRINT TITLE SCREEN**
- 4200-4390 **PRINT INSTRUCTION SCREEN**
- 5000-5290 **SOUND ROUTINES**
Three sound routines.
- 5500-5590 **GOOD LANDING**
Spin clouds and make a sound.
- 6300-8000 **SPRITE DATA**

Ride the Wind

Major Variables

General:

- V Base address of VIC chip
S1 Base address of SID chip
Q1 Wind direction at Y level of a cloud

SC Score
N() Frequency of notes

Balloon:

X1 Horizontal velocity
Y1 Vertical velocity
X X position
Y Y position
SP Sprite page number to use (17 or 18)
SV Sound volume of heater
M Balloons left to play
Q Wind direction at balloon's position

Landing Pads:

L Pad just landed on
LD All pads balloon has landed on
K1 Position
K2 Color

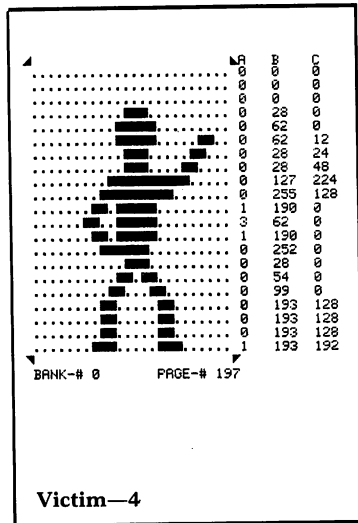
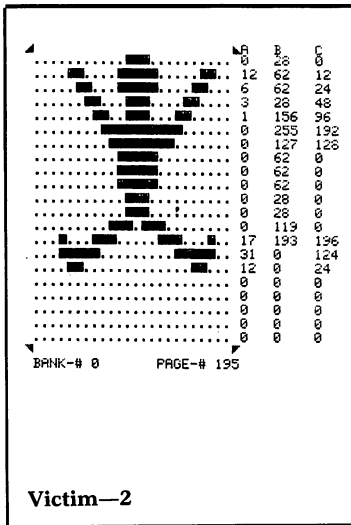
Sprites

(NOTE: All sprites are in bank 2)

Line Numbers	Page	Description
6300-6430	16	Cloud 1
6580-6710	17	Balloon with heater off
6720-6850	18	Balloon with heater on
6900-7030	19	Balloon explosion - 1
7040-7170	20	Balloon explosion - 2
7180-7300	21	Balloon explosion - 3
7310-7440	22	Balloon explosion - 4
7450-7570	23	Balloon explosion - 5
7600-7730	24	Cloud 2
7740-7870	25	Cloud 3
7880-8000	26	Cloud 4

**Ride
the Wind**





You are allowed to crash and try again four times in each canyon. The game will end on the fifth crash or if you run out of fuel. If your fuel gauge shows that you are getting low, you may refuel by returning to base and gently landing. However, you can eventually lose by using up all the fuel available.

Landing beside the victim is the preferred method of rescue, and is rewarded with the most points. In each canyon, the first victim rescued by landing is worth ten points; the second, a hundred; and the third, a thousand. When the rescue is made by touching the victim, you receive only five points. If any victim in the valley has been rescued by landing, a bonus is awarded, based on the time required to perform all the rescues in that valley.

Load the game by typing in **LOAD"DEATH VALLEY",8** and pressing the **RETURN** key. After about 35 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. The title screen will be displayed for about 20 seconds before the game begins. When the game is over, press the fire button to start a new game.

Programmer's Notes

Controlled Random Screens. *Death Valley Patrol* uses a number of different play screens as the game advances. There are three different ways that these could have been programmed: completely predefined screens, completely random screens, or controlled random screens.

Completely predefined screens can create an interesting game, but they have some disadvantages. The screen definitions use a large amount of memory and generally require extensive programming time. Once completed, the game lacks variety because the player will always advance through exactly the same screens in the same sequence each time the game is played.

Random screens are generated at the time the game is played. Their basic design is controlled by general rules or parameters, but the details are determined by a random number generator. Completely random screens are relatively quick to program and present an infinite vari-

Death Valley Patrol

- 30000-30100 **SPRITE INITIALIZATION**
Do checksum to see if sprites already loaded. If not, load sprites. Define sprite size, color, etc. Set all needed constants.

- 31000-31650 **CREATE BACKGROUND**
Create mountains, cave, and split screen. Drop victim. Set landing detectors. Draw score and fuel gauge. If subroutine is accessed from REFUEL, return. Create wind matrix.

- 31700-31710 **CLEAR A HOLE**

- 31900 **CREATE A PSEUDORANDOM NUMBER**

- 32000-32010 **SOUND INITIALIZATION**

- 33000-33150 **SPRITE-BACKGROUND COLLISION**
Check for base landing and landing speed. If too high, explode. If low enough, refuel. Check for cliff collision. If collision detected, explode.

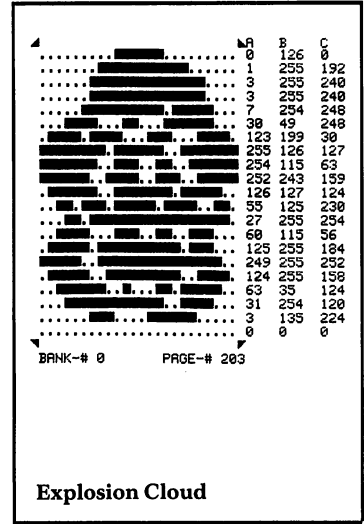
- 33300-33400 **VICTIM RESCUE**
If landing speed is too high, explode. Determine type of rescue and increment score.

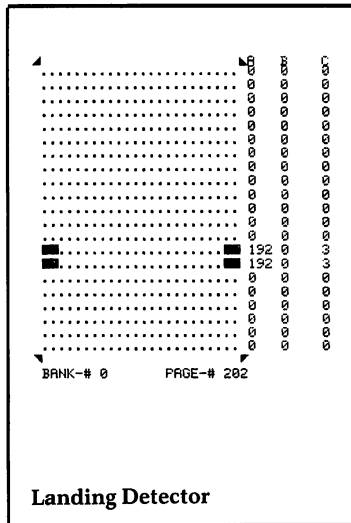
- 34000-34040 **SPRITE-SPRITE COLLISION**
Check to see which victim was rescued. Branch accordingly.

- 35000-35020 **PRINT SCORE**
Change score to a five-character string value and print it.

- 36000-36090 **PRINT TITLE SCREEN**
Print title and make sprites jump and wave.

- 40000-40030 **ENDING**
Make sound and end game.





Major Variables

General:

PK	Joystick status
V	Base address of VIC chip
C1	Game time counter
RP	Random number seed
SC	Score
C2	Random wind counter
DF	Difficulty
AG	Fuel left at base
E	Current mountain height
D	Last mountain height
A	Mountain position
R	Random number
F	Refueling flag
LV	Level
TP	Highest canyon number of last stage
DX()	Horizontal wind
DY()	Vertical wind

Helicopter:

S()	Sprite pointers
DX	Change in horizontal movement
DY	Change in vertical movement
X	Horizontal position
Y	Vertical position
NH	Number of crashes
GS	Fuel left

Victims:

T()	Sprite pointers
NK	Number rescued by touching
NM	Number rescued by landing

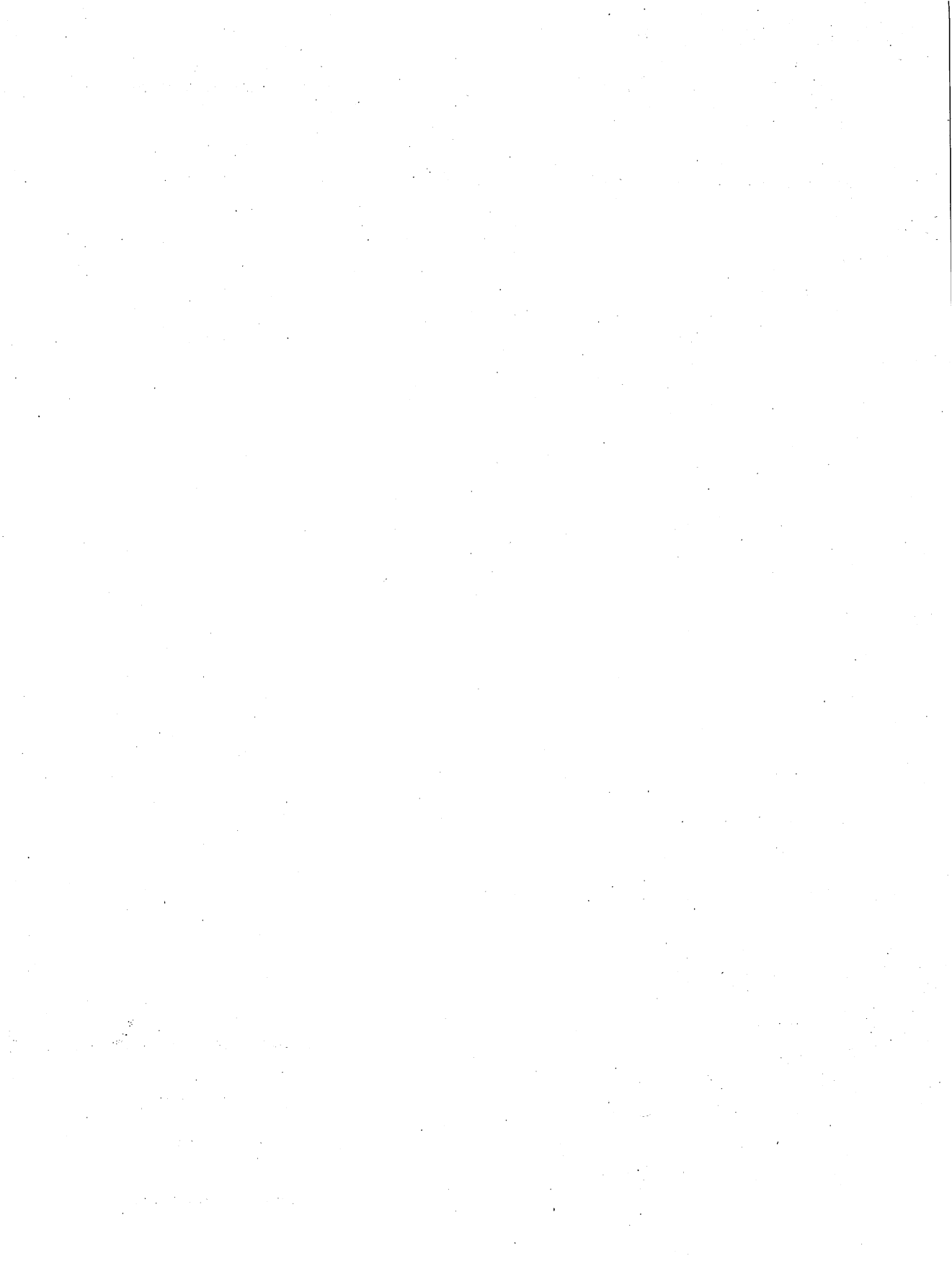
Death Valley Patrol

Sprites

Line Numbers	Page	Description
10000-10006	192	Helicopter - 1
10010-10016	193	Helicopter - 2
10020-10026	194	Victim - 1

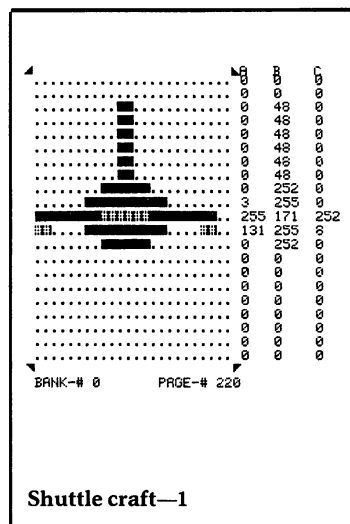
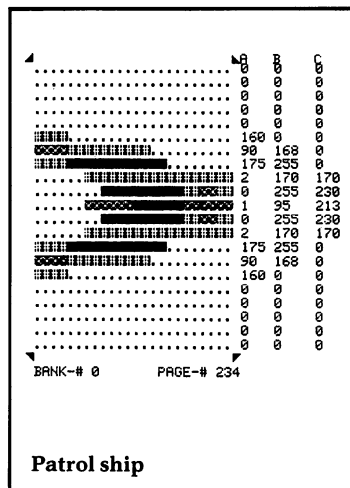
10030-10036	195	Victim - 2
10040-10046	196	Victim - 3
10050-10056	197	Victim - 4
10060-10066	198	Victim - 5
10070-10076	199	Victim - 6
10080-10086	200	Victim - 7
10090-10096	201	Victim - 8
10100-10106	202	Landing detector
10150-10156	203	Explosion cloud

**Death Valley
Patrol**



Mission: Tobor

Charles Mott, Jr.

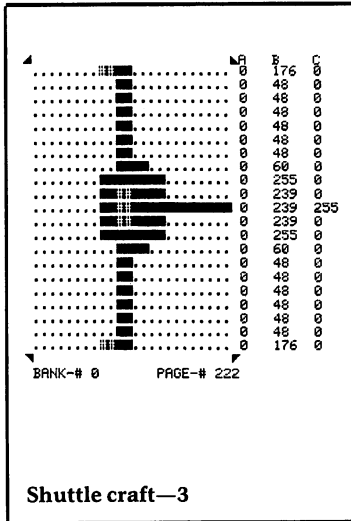


Automated atomic energy reactors provide power for operations on the planet Tobor. The reactors are shielded by a protective force field and guarded by an automated patrol ship. This protection system has developed a serious malfunction and is preventing delivery of essential coolant packs. As a result, the reactors are now in danger of overheating.

You have been dispatched to drop new coolant packs from your space shuttle to the robots below. These robots continuously move back and forth between each reactor and its supply area. If a coolant pack falls just in front of a robot, the robot will catch the pack and take it to the reactor. Keep an eye on the reactor's coolant level gauge. To save the reactor, you must successfully deliver enough coolant packs to fill it up.

The protective force field constantly pushes your shuttle up and away from the reactor. If you penetrate the shield, the defective protection system will strengthen the field and repel you, using more coolant in the process. If you drop a coolant pack from just above the shield, the protection system will dispatch a patrol ship to investigate. A collision with the patrol ship will not harm your shuttle, but it will cause the reactor to use still more coolant.

Mission: Tobor



When you have returned a reactor to normal operation, you will be moved on to the next one. The robot at each new reactor will move faster than the previous one, frantically trying to find coolant packs. If you allow any reactor's coolant level to drop below the critical point, the reactor will melt down and the game will end, displaying your rating as a shuttle pilot.

Use the joystick like the control stick in an airplane to move the shuttle. Push it forward to dive, pull it back to climb, and move it left or right to move sideways. To release a coolant pack, press the fire button.

Load the game by typing **LOAD"MISSION TOBOR",8** and pressing the **RETURN** key. After about 30 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. The title screen will be built on the screen, complete with instructions. Then after a delay of about 20 seconds, the game will begin.

Programmer's Notes

Special Use of the Keyboard Buffer. One way of handling a program which uses many sprites and requires a great deal of memory is to write it in two parts. Part 1 of this game is called **MISSION TOBOR** on the disk. It prints the title screen, scrolls the instructions, and loads the sprite data into memory in preparation for Part 2, the actual game, which is called **&& MISSION**. Part 2 is then run by making special use of the keyboard buffer.

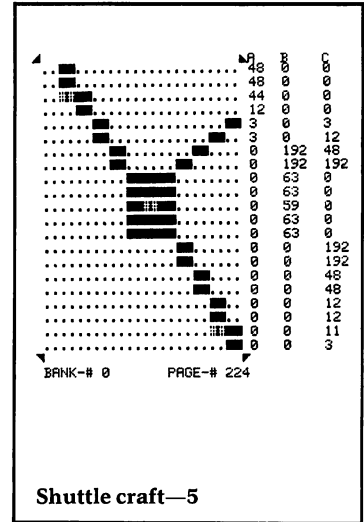
The Commodore 64 has a special section of memory called the keyboard buffer, beginning at memory location 631, for temporary storage of keyboard input. It can hold up to ten characters and is automatically cleared as its contents are executed. Any time a key is pressed, a special code for that key is stored in the keyboard buffer until it can be processed.

For the C-64 to load and run Part 2, it must be given instructions similar to those that were entered to load and run Part 1. To do this, Part 1 prints the instruction **LOAD"&& MISSION",8** onto the screen, in the same color as the background so that it will be invisible. Then, in lines 3610-3650, it stores the following directly into the keyboard buffer: two **CURSOR-UP** commands, a

Mission: Tobor

RETURN command, the characters *R*, *U*, *N*, and a colon, and then another RETURN command. (The character codes for all keyboard commands are listed in appendix F of the *Commodore 64 User's Guide*.)

When Part 1 ends, the C-64 executes the commands that are in the keyboard buffer. The two CURSOR-UP commands position the invisible cursor to the invisible LOAD line. The RETURN command causes the LOAD"&& MISSION",8 to be executed, loading Part 2. The next positions in the keyboard buffer contain RUN; and a RETURN. This is equivalent to typing in RUN and pressing RETURN, so Part 2 then begins running.



Hints for Modifying the Game

Part 1. To present the title screen in a different way, modify lines 700 and 730. The **RIGHT\$** instructions are string functions that control the point in the string variables **X\$** and **Z\$** at which the print statement is to begin, and how many characters after that point will be printed. Replacing the **RIGHT\$** instructions with **LEFT\$** or **MID\$** will create different effects.

The PRINT TITLE SCREEN routine is designed to gradually build the game name on the screen as the sprites are loaded into memory. The name is camouflaged until it is completely built. To have it presented more quickly, eliminate the **GOSUB2000** in line 720.

If you study the program carefully and learn to use the string commands, you can find other, more complicated ways of changing the title screen presentation.

Part 2. To change the initial speed of the robot, modify the value assigned to **PH** in line 103. The speed of the robot at each new reactor is increased in lines 3205-3209. Change the values assigned to **PH** to make the robot speed up by a different amount, or slow down instead. To change the speed of the shuttle, assign a different value to **AP** in line 103. To change the speed of the patrol ship, assign a different value to **U** in the same line.

The strength of the force field's push against the shuttle is set with the statements **MX=MX+1**; **MY=MY-1** in line 390. Change the +1 and -1 to other

**Mission:
Tobor**

```

  0 129 0
  0 66 0
  0 96 128
  0 16 0
  1 8 128
  1 24 128
  0 66 0
  4 66 32
  2 36 64
  1 36 128
  8 129 16
  8 18 34
  68 8 66
  33 96 132
  20 74 40
  2 145 64
  1 8 128
  0 0 0
  0 0 0
  0 0 0
  0 0 0
  BANK-# 0 PAGE-# 218

Broken reactor—2

```

values for the force field to have a different effect on the shuttle's behavior.

The time before a reactor's coolant runs out is controlled by the variable **PI**, which is set to 1 in line 90. Changing the 1 to a 2 will cause the reactors to melt down in half the time. The initial state of the next reactor, determined by its sprite pointer, is set with the statement **PS=202** in line 948. Changing the 202 to a higher number, not greater than 216, will make the next reactor start out in a more critical state.

Set a limit to the number of coolant packs available to be dropped during the game. Or set a limit for each reactor, and decrease it for each reactor serviced. You will have to add variables for the limit that is set and the number of packs dropped.

Major Routines

PART 1

- 0070-0320 **INITIALIZATION**
Begin loading sprite data into memory. Initialize array variables for title and instructions.
- 0510-0730 **PRINT TITLE SCREEN**
Manipulate and print title array variables with sound and extended background color, while loading sprite data.
- 0740-0840 **SCROLL INSTRUCTIONS MESSAGE**
Scroll instruction message in Y\$() across middle of screen while loading sprite data.
- 2000-2100 **READ SPRITE DATA INTO MEMORY**
Read one byte of sprite data into memory.
- 3500-3680 **RUN PART 2**
Lower top of BASIC memory and use keyboard buffer to run Part 2.

**Mission:
Tobor**

5000-5131 **SPRITE DATA**
 Data statements for sprites. Every four lines is a complete sprite.

PART 2

0040-0140 **INITIALIZATION**
 Read data statements for score ratings and joystick control array. Initialize variables and sprites. Print background screen.

0147-0480 **MAIN CONTROL LOOP**
 If joystick activated, move shuttle; otherwise let it drift. Move robot and pack, if one was dropped. Check for robot catching pack. If robot is at reactor with pack, switch to reactor sprite with more coolant. If reactor full, print score and start new mission. If reactor empty, end game.

0500 **READ JOYSTICK**

0600-0620 **MOVE SHUTTLE**

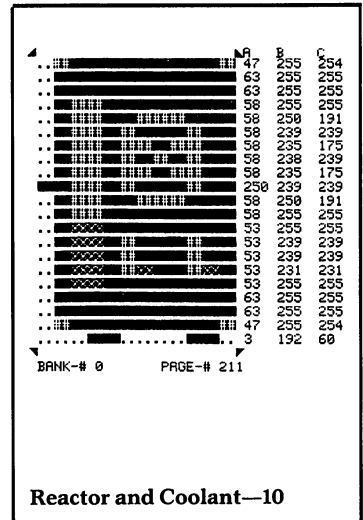
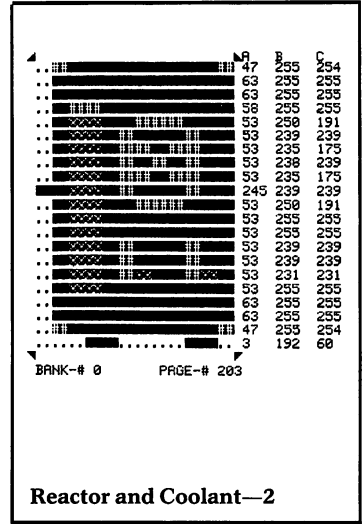
0800-0895 **INITIALIZE SPRITES**
 Sprite positioning and definitions.

0900-0940 **END OF GAME**
 Flash border and background while making melt-down noises. Print overall rating. Wait for fire button to start new game.

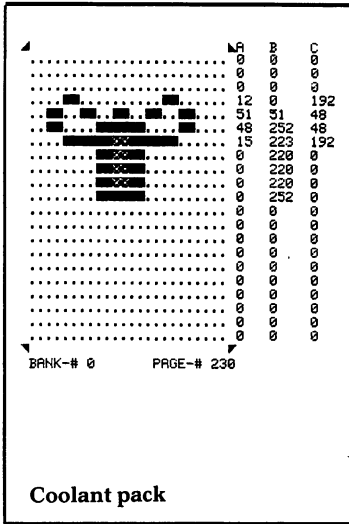
0955-1000 **CATCH COOLANT PACK**
 If pack is within 10 pixels of robot, switch robot sprite to one carrying a pack, and set flags.

1100-1155 **PRINT BACKGROUND SCREEN**

2000-2310 **SOUND ROUTINES**
 Three sound routines.



**Mission:
 Tobor**



3200-3250 PRINT SCORE

Print rating points and increase speed of robot.

Major Variables

PART 1

General:

- S Base address of SID chip
- X\$() Title word "MISSION"
- Z\$() Title word "TOBOR"
- Y\$() Instructions
- A\$ Temporary variable for instruction scroll

PART 2

General:

- V Base address of VIC chip
- SI Base address of SID chip
- SC Base address of screen memory
- BR Border color
- BK Background color
- JS Address of joystick port
- DR() Joystick directions
- FG() Event flags
- S1-S8 Sprite pointers
- YL Number of reactors saved
- LP Pointer to ratings
- G\$() Ratings

Shuttle:

- P1 Sprite pointer
- MX X position
- MY Y position
- AP Pixels moved

**Mission:
Tobor**

Reactor:

- PS Sprite pointer
- PI Sprite pointer increment

Robot:

RB Sprite pointer
 R X position
 PH Pixels moved

Patrol ship:

BP X position
 U Pixels moved

A	B	C
3	255	0
3	171	0
0	252	0
12	48	192
51	51	48
48	252	48
15	223	192
0	220	0
0	220	0
0	220	0
0	252	0
9	48	0
0	48	0
0	48	0
0	48	0
0	48	0
0	48	0
3	255	0
3	255	0
3	255	0
0	136	0

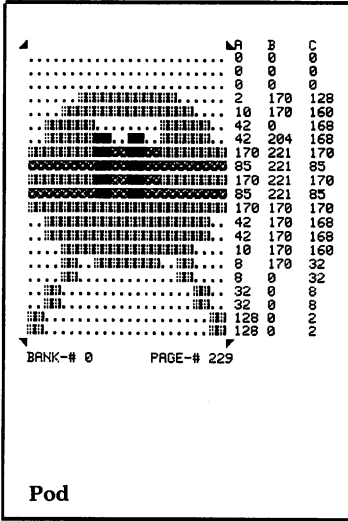
BANK-# 0 PAGE-# 232

Robot-2

Sprites

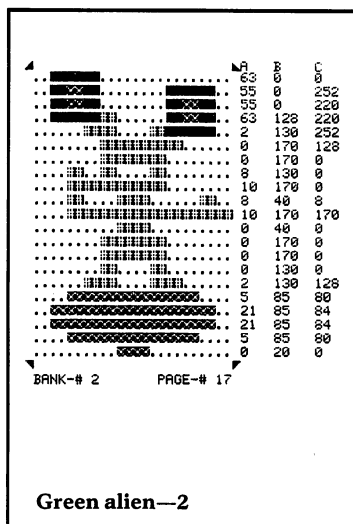
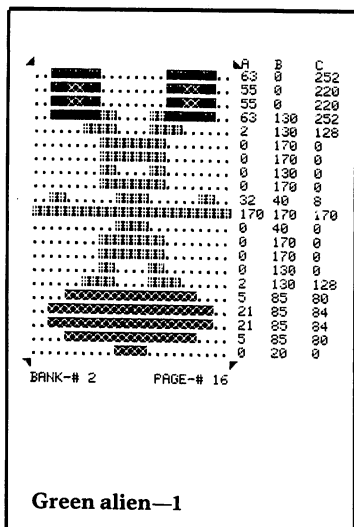
Line Numbers	Page	Description
5000-5003	202	Reactor and coolant - 1
5004-5007	203	Reactor and coolant - 2
5008-5011	204	Reactor and coolant - 3
5012-5015	205	Reactor and coolant - 4
5016-5019	206	Reactor and coolant - 5
5020-5023	207	Reactor and coolant - 6
5024-5027	208	Reactor and coolant - 7
5028-5031	209	Reactor and coolant - 8
5032-5035	210	Reactor and coolant - 9
5036-5039	211	Reactor and coolant - 10
5040-5043	212	Reactor and coolant - 11
5044-5047	213	Reactor and coolant - 12
5048-5051	214	Reactor and coolant - 13
5052-5055	215	Reactor and coolant - 14
5056-5059	216	Reactor and coolant - 15
5060-5063	217	Broken reactor - 1
5064-5067	218	Broken reactor - 2
5068-5071	219	Broken reactor - 3
5072-5075	220	Shuttle craft - 1
5076-5079	221	Shuttle craft - 2
5080-5083	222	Shuttle craft - 3
5084-5087	223	Shuttle craft - 4
5088-5091	224	Shuttle craft - 5
5092-5095	225	Shuttle craft - 6
5096-5099	226	Shuttle craft - 7
5100-5103	227	Barrel
5104-5107	228	Pipework
5108-5111	229	Pod
5112-5115	230	Coolant pack
5116-5119	231	Robot - 1

**Mission:
Tobor**



5120-5123	232	Robot - 2
5124-5127	233	Robot - 3
5128-5131	234	Patrol ship

**Mission:
Tobor**



Load the game by typing in **LOAD" COSMO'S RESCUE",8** and pressing the **RETURN** key. After about 45 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. After the title screen is displayed, the instruction screen will appear and pause for about 15 seconds. Then you will be asked to press the fire button to start the game. When the game is over, press the fire button to return to the instruction screen.

To stop the game, see the special instructions in the Introduction for stopping bank-switched games. This game uses bank 2 for sprites and graphics.

Programmer's Notes

Bank Switching to Handle Large Programs. Because *Cosmo's Rescue* is a very large game, and in addition, uses 26 sprites for animation effects, a special technique was required to load the game and all the sprite data into memory at the same time.

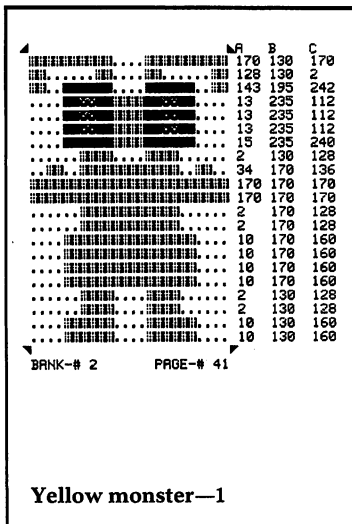
The "Sprites" section of the Introduction explains that the C-64's memory is divided into four banks of 16K each. The computer's VIC chip is designed to "see" and access only one 16K bank at a time. Since the VIC chip performs all sprite handling, it must be set to access the bank containing the pages where the sprite shape data is stored.

Bank 0 of the computer, the first 16K of memory, is ordinarily used for most purposes unless other provisions are made. When the computer is first turned on, the VIC chip is set automatically to bank 0, and when programs are loaded, they are stored there. When a program and its sprite data both fit into 16K, no further action is needed. The sprite data will be stored into pages in bank 0, where the VIC chip is set to find it, and processing will begin.

If the program and its sprite data require more than 16K, however, the sprite data must be stored into pages in another bank. The VIC chip must then be "bank-switched" or set to access the other bank for the sprite data. The program remains in bank 0 and is executed there.

In addition to sprites, the VIC chip also controls other graphics features. Bank switching must be imple-

Cosmo's Rescue



lated in line 700 using **T5**. To move the green figure's starting position, change the values added to **Y** and assigned to **X** in line 700.

In programming **Cosmo's Rescue**, care was taken to make the layout of the cavern easy to change. To design your own cavern, change the **PRINT** statements in lines 3112 through 3158. Getting the new cavern screen to look just the way you want can be tedious and frustrating, but after it is done, you will have a new game to play.

You might want to create several cavern screens, each in its own subroutine. Then program the game to move on to the next screen after each one has been played successfully. You would then have programmed an example of completely predefined screens (see the Programmer's Notes for the game **Death Valley Patrol**).

Major Routines

0010-0335 INITIALIZATION

Set VIC memory to bank 2. Print instruction screen and load sprite data. Set up sprite arrays for animation and turn on sprites. Print play screen.

0400-0490 MAIN LOOP

Move green alien and check remaining fuel. Fall, if out of fuel. Check for landing on platform, hitting cavern or monster, picking up purple alien and successful rescue. Move purple figure and yellow monster.

0700-0790 INITIALIZE GREEN ALIEN

Set starting position, size, and color of green alien and his ship.

1000-1490 MOVE GREEN ALIEN

Read joystick and add pull of gravity. If fire button pressed, subtract from gravity and adjust fuel. If joystick moved, update horizontal speed and adjust fuel. Calculate new position and select proper thruster sprite. Sequence to next green alien sprite and print remaining fuel.

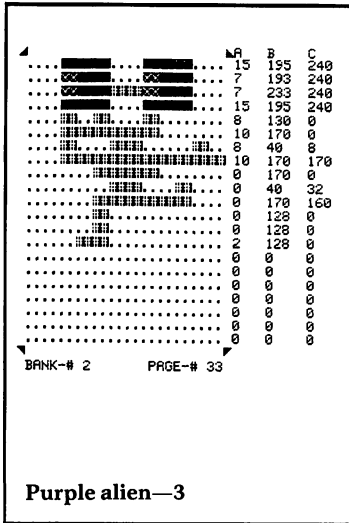
Cosmo's Rescue

- 3000-3090 INITIALIZE SPRITES
Set starting positions and colors.
- 3100-3190 PRINT PLAY SCREEN
- 3200-3290 GAME OVER
Do animation sequence for purple alien and yellow monster while waiting for fire button.
- 3300-3390 PRINT INSTRUCTION SCREEN
- 3500-3590 SCREEN CHANGE
Screen change sound and flash.
- 3600-3690 GREEN ALIEN LANDS ON PLATFORM
Fall if landing speed is too high. Check for purple alien being rescued. Refuel.
- 3700-3790 GREEN ALIEN FALLS
Do falling animation sequence and check for end of game.
- 4000-4090 MOVE PURPLE ALIEN
Sequence to next sprite in animation and calculate new position. Reverse direction if movement boundary reached.
- 4100-4190 MOVE YELLOW MONSTER
Sequence to next sprite in animation and calculate new position. Reverse direction if movement boundary reached.
- 4300-4390 PICK UP PURPLE ALIEN SOUND
Sound for green alien rescuing purple alien.
- 4500-4590 FALLING SOUND
- 4600-4690 SUCCESSFUL RESCUE
Calculate and print score. Make sound and continue game.
- 7000-10540 SPRITE DATA

	NA	B	C
.....	0	0	0
.....	15	195	240
.....	13	195	112
.....	15	235	240
.....	2	0	128
.....	2	0	128
.....	34	0	136
.....	42	0	168
.....	34	0	136
.....	42	0	168
.....	42	0	168
.....	2	170	128
.....	2	170	128
.....	10	170	160
.....	10	170	160
.....	10	170	160
.....	10	170	160
.....	2	130	128
.....	2	130	160
.....	10	130	160
.....	10	128	0

BANK-# 2 PAGE-# 46

Yellow monster—6



Major Variables

General:

FU	Fuel remaining
FS	Score counter
CT	Successful rescue flag
ME	Rescue attempts left
V	Base address of VIC chip
S1	Base address of SID chip
SM	Base address of screen memory
CM	Base address of color memory
M1	Base address of sprite data
T5	Landing platform Y coordinate

Green Alien:

MS()	Animation array for flying
A1()	Animation array for falling
T1	Length of flying sequence
T	Length of falling sequence
X	X position
Y	Y position
X1	Number of pixels to move in X direction
Y1	Number of pixels to move in Y direction
X2	New pixel count for X direction
Y2	New pixel count for Y direction
K1	Page number for bottom thruster sprite
K2	Page number for left thruster sprite
K3	Page number for right thruster sprite
SP	Animation counter

Purple Alien:

GS()	Animation array for walking
T3	Length of animation sequence
E1	Maximum left position
F1	Maximum right position
A3	Animation counter
G1	X position
D1	Direction of movement flag

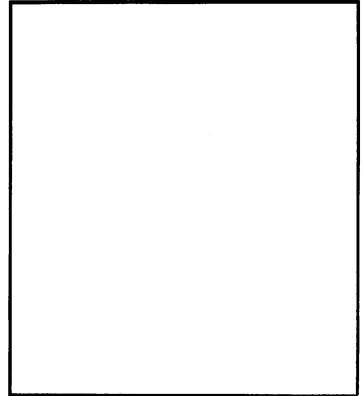
Yellow Monster:

MM()	Animation array for walking
T4	Length of walking sequence
A4	Animation counter

Cosmo's Rescue

Mazemaster

Igor Tulchinsky



Mazemaster tests your ability to maintain a sense of direction in a complex network of corridors. The game simulates 3-D by providing a perspective view of the corridor down which you are moving. The view shows the floor and ceiling, the right and left walls, and the surface facing you at the end of the corridor. The floor and ceiling are alternately red and white, each color panel marking the distance you will move with one step. Doors in the walls are indicated by black panels. In addition, a top view of the maze is available.

There are two different ways of playing this game. In either option, the object is to finish in the shortest time possible. A final score is awarded, based on your playing time and the complexity of the maze.

In option 1, **PAINT THE FLOOR**, you must paint the floor yellow by traveling through the entire maze. Whenever you retrace your steps, the floor in front of you will be yellow instead of red and white. (Since you will start at a random point within the maze, it may be necessary to retrace some of your steps in order to travel through the entire maze.

In option 2, **FIND THE TREASURE**, a treasure is hidden in the maze for you to find. The treasure's location is shown in the top view but invisible to you as you approach it in the corridor. Your distance from the treasure is displayed, however, so you can tell whether you are moving toward or away from it.

Mazemaster

When you select an option, you will also determine the complexity of the maze by entering a number between 1 and 200 and pressing **RETURN**. The program will then randomly generate the maze, displaying a map as it develops. After the maze is drawn, different areas will be colored to indicate the wall color in those areas. The wall colors will help you relate the corridor perspective view to the top-view map as you play.

Pressing the space bar moves you forward one step, and the **R** and **L** keys turn you right and left. To turn around and go back, turn in the same direction two times. In order for you to turn and go through a door, the door must be at the edge of the screen, indicating that you are standing next to it. After you turn, you will have a perspective view down the corridor you have turned into. If you turn and are not standing next to a door, you will just be looking at a wall.

If you become confused as you are playing and would like another top view of the maze, press **H** for help. This time, your position will be marked on the map with a *U*, *D*, *R*, or *L* (for up, down, right, or left), which indicates the direction you are facing within the maze as it is displayed. If you are playing **PAINT THE FLOOR**, the areas already painted will be yellow.

Load the game by typing in **LOAD "MAZEMASTER",8** and pressing the **RETURN** key. After about 20 seconds, the computer will say **READY**, indicating that the game is loaded. Then type **RUN** and press the **RETURN** key. Select an option from the game menu and enter the desired complexity. The program will then begin drawing the maze. After you finish a game, press the space bar to return to the menu screen.

Programmer's Notes

Completely Random Screen Generation. The mazes in **Mazemaster** are created by a completely random method. See the Programmer's Notes for the game **Death Valley Patrol** for a discussion of completely predefined, completely random, and controlled random screen generation.

Mazemaster

Simulated Three-Dimensional Display. One of the unusual features of this game is its three-dimensional display of a maze. Each interior view of a corridor shows the nearest five steps, with the floor, ceiling, and walls converging toward the center of the screen. Alternating red and white ceiling and floor panels mark the steps. If a wall in one of the five steps opens onto a side corridor, it is displayed as a black panel. When the player is within five steps of the end of a corridor, the center of the screen shows the color of the wall that is being approached.

One set of array variables is used for all the different interior views of the corridors. The string arrays $G\$()$, $C\$()$, $R\$()$ and $L\$()$ represent the colors of the floor, ceiling, left walls and right walls, respectively. Each array has five elements, one for each step in the view. As the end of the corridor is approached and there are fewer than five steps to display, the other array variables are assigned to be the color of the end wall, so that it appears to grow larger and larger with each step.

Other variables are used to indicate the direction the player is facing in the maze and to monitor the features of the corridor that are to be displayed on the next step. As the player moves forward in the maze, the contents of the string array variables are modified to reflect the new view.

Hints for Modifying the Game

The score is calculated and printed in lines 40040-40055, based on time and maze complexity. You might want to add a penalty for looking at the maze map. First, add the line **1005 NA = NA + 1** to count the number of times the maze map is displayed. Then add **40051 SC = SC - NA * 10** to include the new counter in the score calculations. The numbers **1005** and **40051** are line numbers which indicate the location in the program where you should place the new lines.

Mazemaster has instructions to end a game when a time limit is reached, but the maximum number of seconds is set with the statement **TM = 1E9** in line 22070. The expression **1E9** is BASIC notation for ten raised to the ninth power – a gigantic number! Change this line to set

Mazemaster

the time limit to a realistic number based on maze complexity.

You might want to try playing in the dark: turn off the three-dimensional display and just make a "bump" sound when you hit the end of a corridor. Or make the treasure periodically move around within the maze.

Major Routines

- | | |
|-------------|--|
| 00013 | INITIALIZATION
Set up sound. Print title screen. Offer and accept game and complexity choice. Create selected maze. Jump to selected main loop. |
| 00100-00410 | 3-D PRINTOUT OF MAZE
Print out 3-D maze as calculated in next section. |
| 00530-00720 | 3-D PRINTOUT CALCULATIONS
Set up direction variables. Calculate colors for ceiling, floor, left, and right walls for five squares in direction player is facing. |
| 01000-01060 | PRINT HELP MAZE
Display top view of maze. Display player and, if in FIND THE TREASURE option, display treasure. |
| 02000-02110 | PAINT FLOOR MAIN LOOP
Select starting point. Print out maze, squares painted, and time. Wait for key to be pressed, while incrementing time. React to key pressed. |
| 03000-03110 | FIND TREASURE MAIN LOOP
Select starting point. Print out maze, distance, and time. Wait for key to be pressed, while incrementing time. React to key pressed. |

Mazemaster

- 20030-20090 **CREATE RANDOM MAZE**
Create maze, by picking random direction and testing spot two units in that direction. If spot is empty, link it to maze. If filled, pick new direction. If all directions are filled, pick new starting point.
- 20100-20180 **COLOR THE MAZE**
Color all walls, ceilings and floors.
- 21010-21080 **PRINT TITLE**
Display heading.
- 22010-22080 **PRINT GAMES MENU**
Print menu and enter all inputs.
- 40005-40080 **GAME OVER**
Sound end of game sound. Calculate score. Wait for key to be pressed, then resume game.
- 50000-50010 **SOUND INITIALIZATION**
- 55010 **SOUND A BEEP**

Major Variables

General:

- M%() Maze
C\$() Ceiling color
G\$() Ground color
R\$() Right wall color
L\$() Left wall color
D() Directions
CM Complexity
F3 Space taken flag
G\$ Game number
DX() Horizontal component
DY() Vertical component
D1 Check for adjacent corridors (right)
D2 Check for adjacent corridors (left)

Player:

XX Horizontal distance "seen"
YY Vertical distance "seen"
X Horizontal position
Y Vertical position
D Direction

Treasure game:

A Horizontal position
B Vertical position

Paint game:

NY Number of squares painted yellow
NB Number of squares
P Position of next yellow square

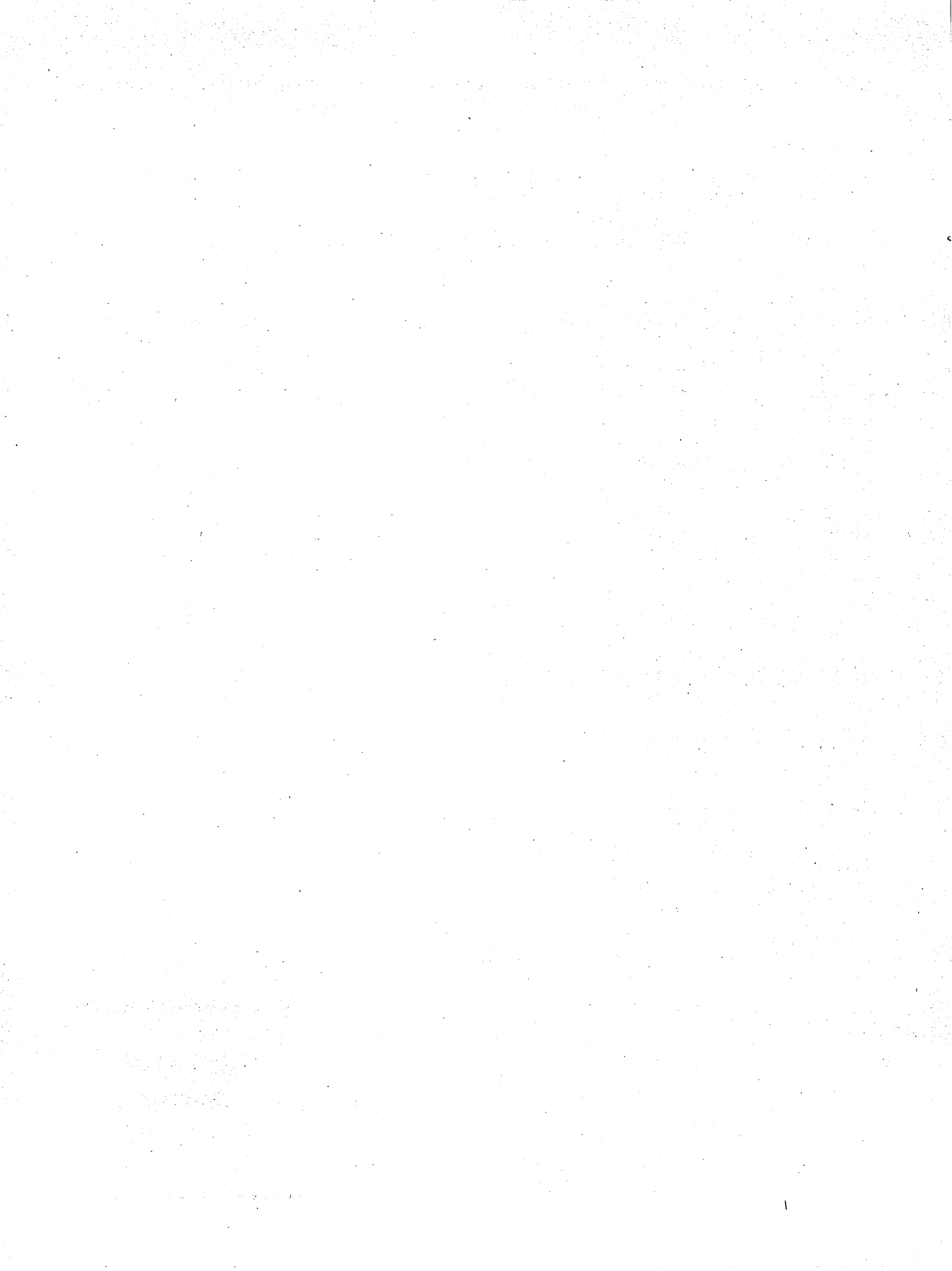
Mazemaster

Appendix A

Game Programs

```
5340 DATA255,255,255,0,255,255,255,4,55
5350 DATA128,192,192,24,25,25,25,5,4,55
5360 DATA3,15,15,31,63,127,127,255
```

**Rocket
Lander
Program**




```
000000 REM PRINT TITLE SCREEN
000001 PRINT "FIRECRACKER BOY"; GOSUB 400:PRINT "BY";
000002 FORT#1T08:PRINTX#(T):GOSUB670:GOSUB5
000003 NEXT
000004 FORT#1T060:PRINT "FIRECRACKER BOY";
000005 FORTT#1T08:GOSUB670:PRINTLEFT#(Z#(T),
000006 FORT#1T050:GOSUB500:NEXTL
000007 FORT#1T050:PRINT "FIRECRACKER BOY";
000008 FORT#1T05:GOSUB670:PRINTLEFT#(C#(T),
000009 NEXT:GOSUB500:NEXTL
000010 FORT#1T050:PRINT "FIRECRACKER BOY";
000011 FORT#1T05:GOSUB670:PRINTLEFT#(V#(T),
000012 NEXT:GOSUB500:NEXTL
000013 PRINT "FIRECRACKER BOY";
000014 BY CHARLES MOTT, JR."
000015 PRINT "FIRECRACKER BOY";
000016 1984 FANFARE HOUSE, INC.;"
000017 FORT#STOS+24:POKET,0:NEXT:GOTOS30
000018
000019 REM INITIALIZE FALLING SOUND
000020
000021 POKEM+4,17
000022 ON M+5:RETURN
000023
000024 REM FALLING SOUND
000025
000026 IFOK<0OROV>N55THENO=1
000027 POKEM+1,0:O=O-1.5
000028 RETURN
000029
000030 REM PRINT INSTRUCTION SCREEN
000031
000032 POKEM+3,10:POKES+1,10:POKES+24,15:PO
000033 T#4,140:POKES+4,129
000034 FORT#1T0188:GOSUB670:NEXT
000035 PRINT "FIRECRACKER BOY";
000036 FORT#0T0359:PRINT "FIRECRACKER BOY";NEXT
000037 PRINT "FIRECRACKER BOY";
000038 FROM THE FIREFIGHTER'S
000039 WINDOW TO WINDOW, T
000040 HROW:PRINT "FIRECRACKER BOY";
000041 WATER BALLOONS ON THE LITT
000042 LE BOYS' HEADS.
000043 PRINT "FIRECRACKER BOY";
000044 FIRECRACKERS, SO THE MAYOR
000045 GET UPSET DURING THE FOURTH
000046 JULY
000047 PRINT "FIRECRACKER BOY";
000048 PARADE. "
000049 FORT#1T0400:GOSUB670:NEXT
000050 FORT#1T0650:GOSUB670:POKE1343+JU,105
000051 JU+1:GOSUB650:NEXT
000052 POKES+4,129:POKES+4,128
000053 RETURN
000054
000055 REM POKE 1 BYTE OF SPRITE DATA
000056
000057 READA:IFR<>256THENPOKELL,A:LL=LL+1:R
000058 EAD
000059 PRINT "FIRECRACKER BOY";
000060 AND NOW..."
000061 PRINT "FIRECRACKER BOY";
000062
000063 REM LOAD PART 2
000064
000065 TOPOKE#0+5,01:POKES+4,128:POKES+4,129
000066 FORT#1,50:POKES+56,75
000067 FORT#1,50:POKES+1,50:POKES+13,50
000068 FORT#1,50:POKES+13,50:POKES+13,50
000069 FORT#1,50:POKES+13,50:POKES+13,50
000070 FORT#1,50:POKES+13,50:POKES+13,50
000071 FORT#1,50:POKES+13,50:POKES+13,50
000072 FORT#1,50:POKES+13,50:POKES+13,50
000073 FORT#1,50:POKES+13,50:POKES+13,50
000074 FORT#1,50:POKES+13,50:POKES+13,50
000075 FORT#1,50:POKES+13,50:POKES+13,50
000076 FORT#1,50:POKES+13,50:POKES+13,50
000077 FORT#1,50:POKES+13,50:POKES+13,50
000078 FORT#1,50:POKES+13,50:POKES+13,50
000079 FORT#1,50:POKES+13,50:POKES+13,50
000080 FORT#1,50:POKES+13,50:POKES+13,50
000081 FORT#1,50:POKES+13,50:POKES+13,50
000082 FORT#1,50:POKES+13,50:POKES+13,50
000083 FORT#1,50:POKES+13,50:POKES+13,50
000084 FORT#1,50:POKES+13,50:POKES+13,50
000085 FORT#1,50:POKES+13,50:POKES+13,50
000086 FORT#1,50:POKES+13,50:POKES+13,50
000087 FORT#1,50:POKES+13,50:POKES+13,50
000088 FORT#1,50:POKES+13,50:POKES+13,50
000089 FORT#1,50:POKES+13,50:POKES+13,50
000090 FORT#1,50:POKES+13,50:POKES+13,50
000091 FORT#1,50:POKES+13,50:POKES+13,50
000092 FORT#1,50:POKES+13,50:POKES+13,50
000093 FORT#1,50:POKES+13,50:POKES+13,50
000094 FORT#1,50:POKES+13,50:POKES+13,50
000095 FORT#1,50:POKES+13,50:POKES+13,50
000096 FORT#1,50:POKES+13,50:POKES+13,50
000097 FORT#1,50:POKES+13,50:POKES+13,50
000098 FORT#1,50:POKES+13,50:POKES+13,50
000099 FORT#1,50:POKES+13,50:POKES+13,50
000100 FORT#1,50:POKES+13,50:POKES+13,50
000101 FORT#1,50:POKES+13,50:POKES+13,50
000102 FORT#1,50:POKES+13,50:POKES+13,50
000103 FORT#1,50:POKES+13,50:POKES+13,50
000104 FORT#1,50:POKES+13,50:POKES+13,50
000105 FORT#1,50:POKES+13,50:POKES+13,50
000106 FORT#1,50:POKES+13,50:POKES+13,50
000107 FORT#1,50:POKES+13,50:POKES+13,50
000108 FORT#1,50:POKES+13,50:POKES+13,50
000109 FORT#1,50:POKES+13,50:POKES+13,50
000110 FORT#1,50:POKES+13,50:POKES+13,50
000111 FORT#1,50:POKES+13,50:POKES+13,50
000112 FORT#1,50:POKES+13,50:POKES+13,50
000113 FORT#1,50:POKES+13,50:POKES+13,50
000114 FORT#1,50:POKES+13,50:POKES+13,50
000115 FORT#1,50:POKES+13,50:POKES+13,50
000116 FORT#1,50:POKES+13,50:POKES+13,50
000117 FORT#1,50:POKES+13,50:POKES+13,50
000118 FORT#1,50:POKES+13,50:POKES+13,50
000119 FORT#1,50:POKES+13,50:POKES+13,50
000120 FORT#1,50:POKES+13,50:POKES+13,50
000121 FORT#1,50:POKES+13,50:POKES+13,50
000122 FORT#1,50:POKES+13,50:POKES+13,50
000123 FORT#1,50:POKES+13,50:POKES+13,50
000124 FORT#1,50:POKES+13,50:POKES+13,50
000125 FORT#1,50:POKES+13,50:POKES+13,50
000126 FORT#1,50:POKES+13,50:POKES+13,50
000127 FORT#1,50:POKES+13,50:POKES+13,50
000128 FORT#1,50:POKES+13,50:POKES+13,50
000129 FORT#1,50:POKES+13,50:POKES+13,50
000130 FORT#1,50:POKES+13,50:POKES+13,50
000131 FORT#1,50:POKES+13,50:POKES+13,50
000132 FORT#1,50:POKES+13,50:POKES+13,50
000133 FORT#1,50:POKES+13,50:POKES+13,50
000134 FORT#1,50:POKES+13,50:POKES+13,50
000135 FORT#1,50:POKES+13,50:POKES+13,50
000136 FORT#1,50:POKES+13,50:POKES+13,50
000137 FORT#1,50:POKES+13,50:POKES+13,50
000138 FORT#1,50:POKES+13,50:POKES+13,50
000139 FORT#1,50:POKES+13,50:POKES+13,50
000140 FORT#1,50:POKES+13,50:POKES+13,50
000141 FORT#1,50:POKES+13,50:POKES+13,50
000142 FORT#1,50:POKES+13,50:POKES+13,50
000143 FORT#1,50:POKES+13,50:POKES+13,50
000144 FORT#1,50:POKES+13,50:POKES+13,50
000145 FORT#1,50:POKES+13,50:POKES+13,50
000146 FORT#1,50:POKES+13,50:POKES+13,50
000147 FORT#1,50:POKES+13,50:POKES+13,50
000148 FORT#1,50:POKES+13,50:POKES+13,50
000149 FORT#1,50:POKES+13,50:POKES+13,50
000150 FORT#1,50:POKES+13,50:POKES+13,50
000151 FORT#1,50:POKES+13,50:POKES+13,50
000152 FORT#1,50:POKES+13,50:POKES+13,50
000153 FORT#1,50:POKES+13,50:POKES+13,50
000154 FORT#1,50:POKES+13,50:POKES+13,50
000155 FORT#1,50:POKES+13,50:POKES+13,50
000156 FORT#1,50:POKES+13,50:POKES+13,50
000157 FORT#1,50:POKES+13,50:POKES+13,50
000158 FORT#1,50:POKES+13,50:POKES+13,50
000159 FORT#1,50:POKES+13,50:POKES+13,50
000160 FORT#1,50:POKES+13,50:POKES+13,50
000161 FORT#1,50:POKES+13,50:POKES+13,50
000162 FORT#1,50:POKES+13,50:POKES+13,50
000163 FORT#1,50:POKES+13,50:POKES+13,50
000164 FORT#1,50:POKES+13,50:POKES+13,50
000165 FORT#1,50:POKES+13,50:POKES+13,50
000166 FORT#1,50:POKES+13,50:POKES+13,50
000167 FORT#1,50:POKES+13,50:POKES+13,50
000168 FORT#1,50:POKES+13,50:POKES+13,50
000169 FORT#1,50:POKES+13,50:POKES+13,50
000170 FORT#1,50:POKES+13,50:POKES+13,50
000171 FORT#1,50:POKES+13,50:POKES+13,50
000172 FORT#1,50:POKES+13,50:POKES+13,50
000173 FORT#1,50:POKES+13,50:POKES+13,50
000174 FORT#1,50:POKES+13,50:POKES+13,50
000175 FORT#1,50:POKES+13,50:POKES+13,50
000176 FORT#1,50:POKES+13,50:POKES+13,50
000177 FORT#1,50:POKES+13,50:POKES+13,50
000178 FORT#1,50:POKES+13,50:POKES+13,50
000179 FORT#1,50:POKES+13,50:POKES+13,50
000180 FORT#1,50:POKES+13,50:POKES+13,50
000181 FORT#1,50:POKES+13,50:POKES+13,50
000182 FORT#1,50:POKES+13,50:POKES+13,50
000183 FORT#1,50:POKES+13,50:POKES+13,50
000184 FORT#1,50:POKES+13,50:POKES+13,50
000185 FORT#1,50:POKES+13,50:POKES+13,50
000186 FORT#1,50:POKES+13,50:POKES+13,50
000187 FORT#1,50:POKES+13,50:POKES+13,50
000188 FORT#1,50:POKES+13,50:POKES+13,50
000189 FORT#1,50:POKES+13,50:POKES+13,50
000190 FORT#1,50:POKES+13,50:POKES+13,50
000191 FORT#1,50:POKES+13,50:POKES+13,50
000192 FORT#1,50:POKES+13,50:POKES+13,50
000193 FORT#1,50:POKES+13,50:POKES+13,50
000194 FORT#1,50:POKES+13,50:POKES+13,50
000195 FORT#1,50:POKES+13,50:POKES+13,50
000196 FORT#1,50:POKES+13,50:POKES+13,50
000197 FORT#1,50:POKES+13,50:POKES+13,50
000198 FORT#1,50:POKES+13,50:POKES+13,50
000199 FORT#1,50:POKES+13,50:POKES+13,50
000200 FORT#1,50:POKES+13,50:POKES+13,50
```

Firecracker Boy Program


```

GOTO500
:
: REM      INITIALIZE CHARACTERS
:
N1=1:P1=10
POKECM+P1,0
N2<1>=1:P2<1>=10
POKESM+P2<1>,0:N2<2>=162
POKECM+P2<1>,0
40 POKESM+P2<2>,0:N2<2>=162:POKECM+P2<2>,0
M1=49:100+1016:SP=16
POKEV+M1,1:POKEM1,SP:POKEV+29,1:POKE
,1:POKEV+30,1
POKEV+31,7
16,1:POKEV+32,2:POKEV+33,0:POK
,1:POKEV+1,131
N1=N2:POKESM+M1,0
RETURN
:
: REM      PRINT PLAY SCREEN
:
PRINT" ":POKE53281,12:POKE53280,0:GO
40
FORI=0TO39
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=0TO999
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=999TO960STEP-1
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=960TO800STEP-40
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=800TO600STEP-200
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=600TO400STEP-200
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=400TO200STEP-200
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=200TO0STEP-200
POKEI+CM,14:POKEI+SM,167
NEXTI
FORI=0TO40
POKEI+CM,0:POKEI+SM,167
NEXTI
FORI=40TO600STEP40
POKEI+CM,0:POKEI+SM,167
NEXTI
FORI=711TO710
POKEI+CM-40,0:POKEI+SM-40,167
NEXTI
FORI=1TO40
X=INT(RND(0)*30+1):Y=INT(RND(0)*23+1
)
POKECM+X+Y*40,0
NEXTI
FORI=0TO4:FORJ=0TO6
POKEI+J*40+I*47+SM,32
NEXTI
PRINT"*****MIN:00*****SEC:00
*****"
PRINT"*****"
PRINT"*****"
RETURN
:
: REM      MOVE COOK
:
JV=15<PEEK(56328)>AND15
IFJV=1ORJV=9STHEN1=0:T2=1:GOTO1100
IFJV=10ORJV=16STHEN1=1:T2=0:GOTO1100
IFJV=4ORJV=5STHEN1=0:T2=1:GOTO1100
IFJV=2ORJV=3STHEN1=-1:T2=0:GOTO1100
RETURN
T3=T1+T2*40:T4=T3:Q1=PEEK(T3+SM)
IFQ1=0STHEN1=0
IFQ1<166>THENRETURN
FORI=1TONM
T4=T1+T2*40:Q2=PEEK(T4+SM)
IFQ2=0STHENGOTO1150
IFQ2<166>THENRETURN
NEXTI
RETURN
POKEI+SM,166:POKET4+CM,0
N1=N2+1:IFN1<4THENN1=1
POKEP1+SM,0:N1=1:T3=POKEP1+CM,0:POKE
M,0
RETURN
:
: REM      MOVE CHICKEN

```

Into the Pot Program


```

IF FT=1 THEN GOSUB 6030 : GOSUB 5200 : GOTO 400
IF FT=2 THEN GOSUB 6400 : GOSUB 5200 : GOTO 400
GOTO 500
REM INITIALIZE SPRITES
RX=0 : POKEY+16,0
POKEY,100 : POKEY+1,200 : POKEY+2,0 : POKE
RX=14
DX=100 : D=1 : N=1
RX=150 : D=1 : D=20
RETURN
REM BUILD FORCE FIELD
PRINT "J" : POKE 53281,0 : POKE 53280,5
FOR I=0 TO 150
FOR J=0 TO 150
POKE I+J*16,0
NEXT J
NEXT I
REM BUILD FLOOR
FOR I=1984 TO 2023 : POKE I+54272,4 : POKE I+
1984,160 : NEXT I
PRINT " " : BL=0
RETURN
REM PLAYER MOVEMENT DRIVER
JV=PEEK<56320>
IF JV=127 THEN 1090
JV=JV AND 16
IF JV<15 AND 15
IF JV=40 OR JV=50 OR JV=6 THEN D=1 : T=X-9 : GOT
O 1110
IF JV=80 OR JV=90 OR JV=10 THEN D=2 : T=X+9 : GO
TO 1110
N1=1 : POKE M1,S1<D,N1>
RETURN
REM MOVE PLAYER
N1=N1+1 : IF N1>81 THEN N1=1
IF T<TV OR T<OR OR T<12 THEN X=N1=1
X=X+RX : X=X AND 254 : IF T=255 THEN T=T-25
POKE M1,S1
GOSUB 6000
RETURN
REM MOVE ROW TO THE RIGHT
GOSUB 6010
FOR I=0 TO 40 : TOP+<K-1>*40 STEP -1
NEXT I
POKE M+1,M+1 : T1=T2
NEXT I
POKE M1+4,0 : RETURN
REM MOVE ROW TO THE LEFT
GOSUB 6010
FOR I=0 TO 40 : TOP+K*40-1
NEXT I
POKE M+1,M+1 : T1=T2
NEXT I
POKE M1+4,0
RETURN
REM MAIN LOOP - SHOOTING
T=1783+INT<<X+4>/8> : POKE T+54272,4 : P
OKE T,S1 : S3=70
BL=BL+1 : PRINT " "
G=PEEK<V+31> : IF <Q AND 2>=2 THEN F=1 : GOT
O 3000
IF F=0 THEN 3000
POKE T+SM-1024,32 : POKE T+54272,2 : T=T-
400 : IF T=0 THEN 3000
IF T=0 THEN 3000
IF <K+SM-1024>=160 THEN 3000
S3=INT<S3+1> : GOSUB 6000
POKE T+4,2 : POKE T+SM-1024,S1
FOR I=0 TO 100 : NEXT I
ON S3 GOSUB 1000,4000,4100,4200
GOTO 3000

```

The Blobbis Program


```

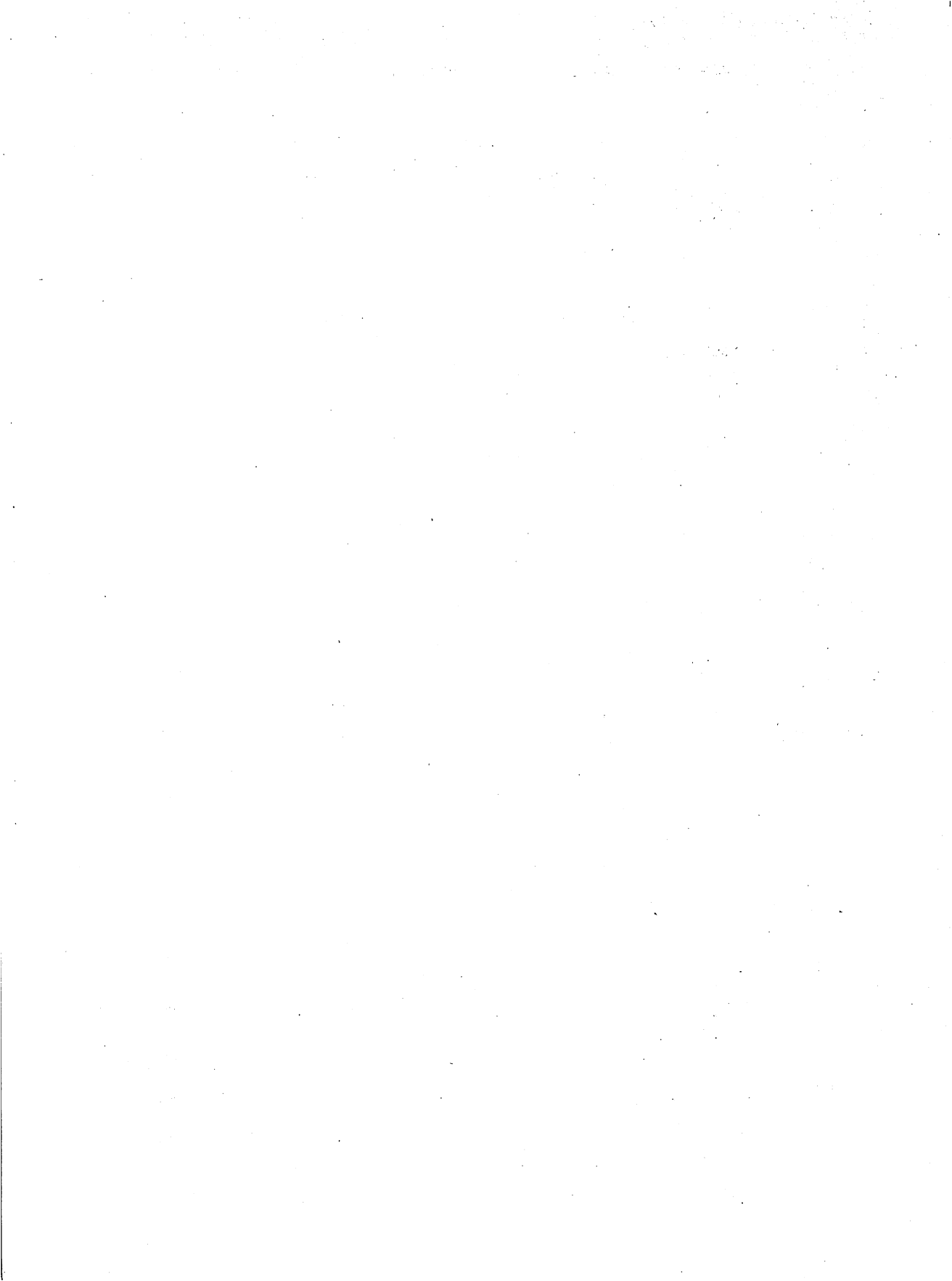
N1070 PRINT" * OUTSIDE OF THE SCREEN ARE
N1080 PRINT" *
N1090 PRINT" *
N1100 PRINT" *
N1110 PRINT" * LOVE A TEAM MEMBER WIT
N1120 PRINT" * JOYSTICK. *SELECT A DIFFER
N1130 PRINT" * MEMBER BY PRESSING THE FI
N1140 PRINT" * AND POINTING THE JOYSTICK
N1150 PRINT" * DIRECTION ASSIGNED TO THA
N1160 PRINT" *
N1170 PRINT" *
N1180 PRINT" *
N1190 PRINT" *
N1200 PRINT" *
N1210 PRINT" *
N1220 PRINT" * PRESS THE FIRE BU
N1230 PRINT" *
N1240 PRINT" *****
N1250 RETURN

```

Sound the Whistle Program

```
4000000 P:PRINT " (C) 1984 FANFARE HOUSE
4100000 P:PRINT "
4110000 POKE POKKEY,255:IFFX=1THENRETURN
4111000 POKE POKKEY,100
4112000 POKE POKKEY,155
4113000 POKE POKKEY,100
4114000 POKE POKKEY,0:POKE I,0:NEXT
4115000 GOTO 0646:POKE I,0:NEXT
4116000 GOTO 0646:POKE I,0:NEXT
4117000 GOTO 0646:POKE I,0:NEXT
4118000 RETURN
4119000 REM PRINT INSTRUCTION SCREEN
4200000 POKE POKKEY+I*2+1,0:NEXT
4210000 PRINT "YOU MUST LAND THE BALLOON
4220000 PRINT "
4230000 PRINT "THE LANDING PADS (10 POIN
4240000 PRINT "
4250000 PRINT "SPACE BAR CONTROLS THE BA
4260000 PRINT "
4270000 PRINT "WHICH WILL MAKE IT START
4280000 PRINT "
4290000 PRINT "WIND BLOWS THE BALLOON LE
4300000 PRINT "
4310000 PRINT "
4320000 PRINT "
4400000 GOTO 0646:POKE I,0:NEXT
4410000 GOTO 0646:POKE I,0:NEXT
4420000 GOTO 0646:POKE I,0:NEXT
4430000 GOTO 0646:POKE I,0:NEXT
4440000 GOTO 0646:POKE I,0:NEXT
4450000 GOTO 0646:POKE I,0:NEXT
4460000 GOTO 0646:POKE I,0:NEXT
4470000 GOTO 0646:POKE I,0:NEXT
4480000 GOTO 0646:POKE I,0:NEXT
4490000 GOTO 0646:POKE I,0:NEXT
4500000 GOTO 0646:POKE I,0:NEXT
4510000 GOTO 0646:POKE I,0:NEXT
4520000 GOTO 0646:POKE I,0:NEXT
4530000 GOTO 0646:POKE I,0:NEXT
4540000 GOTO 0646:POKE I,0:NEXT
4550000 GOTO 0646:POKE I,0:NEXT
4560000 GOTO 0646:POKE I,0:NEXT
4570000 GOTO 0646:POKE I,0:NEXT
4580000 GOTO 0646:POKE I,0:NEXT
4590000 GOTO 0646:POKE I,0:NEXT
4600000 GOTO 0646:POKE I,0:NEXT
4610000 GOTO 0646:POKE I,0:NEXT
4620000 GOTO 0646:POKE I,0:NEXT
4630000 GOTO 0646:POKE I,0:NEXT
4640000 GOTO 0646:POKE I,0:NEXT
4650000 GOTO 0646:POKE I,0:NEXT
4660000 GOTO 0646:POKE I,0:NEXT
4670000 GOTO 0646:POKE I,0:NEXT
4680000 GOTO 0646:POKE I,0:NEXT
4690000 GOTO 0646:POKE I,0:NEXT
4700000 GOTO 0646:POKE I,0:NEXT
4710000 GOTO 0646:POKE I,0:NEXT
4720000 GOTO 0646:POKE I,0:NEXT
4730000 GOTO 0646:POKE I,0:NEXT
4740000 GOTO 0646:POKE I,0:NEXT
4750000 GOTO 0646:POKE I,0:NEXT
4760000 GOTO 0646:POKE I,0:NEXT
4770000 GOTO 0646:POKE I,0:NEXT
4780000 GOTO 0646:POKE I,0:NEXT
4790000 GOTO 0646:POKE I,0:NEXT
4800000 GOTO 0646:POKE I,0:NEXT
4810000 GOTO 0646:POKE I,0:NEXT
4820000 GOTO 0646:POKE I,0:NEXT
4830000 GOTO 0646:POKE I,0:NEXT
4840000 GOTO 0646:POKE I,0:NEXT
4850000 GOTO 0646:POKE I,0:NEXT
4860000 GOTO 0646:POKE I,0:NEXT
4870000 GOTO 0646:POKE I,0:NEXT
4880000 GOTO 0646:POKE I,0:NEXT
4890000 GOTO 0646:POKE I,0:NEXT
4900000 GOTO 0646:POKE I,0:NEXT
4910000 GOTO 0646:POKE I,0:NEXT
4920000 GOTO 0646:POKE I,0:NEXT
4930000 GOTO 0646:POKE I,0:NEXT
4940000 GOTO 0646:POKE I,0:NEXT
4950000 GOTO 0646:POKE I,0:NEXT
4960000 GOTO 0646:POKE I,0:NEXT
4970000 GOTO 0646:POKE I,0:NEXT
4980000 GOTO 0646:POKE I,0:NEXT
4990000 GOTO 0646:POKE I,0:NEXT
5000000 GOTO 0646:POKE I,0:NEXT
5100000 REM SOUND ROUTINES
5110000 POKE POKKEY,15
5120000 POKE POKKEY,15
5130000 POKE POKKEY,15
5140000 POKE POKKEY,15
5150000 POKE POKKEY,15
5160000 POKE POKKEY,15
5170000 POKE POKKEY,15
5180000 POKE POKKEY,15
5190000 POKE POKKEY,15
5200000 POKE POKKEY,15
5210000 POKE POKKEY,15
5220000 POKE POKKEY,15
5230000 POKE POKKEY,15
5240000 POKE POKKEY,15
5250000 POKE POKKEY,15
5260000 POKE POKKEY,15
5270000 POKE POKKEY,15
5280000 POKE POKKEY,15
5290000 POKE POKKEY,15
5300000 POKE POKKEY,15
5310000 POKE POKKEY,15
5320000 POKE POKKEY,15
5330000 POKE POKKEY,15
5340000 POKE POKKEY,15
5350000 POKE POKKEY,15
5360000 POKE POKKEY,15
5370000 POKE POKKEY,15
5380000 POKE POKKEY,15
5390000 POKE POKKEY,15
5400000 POKE POKKEY,15
5410000 POKE POKKEY,15
5420000 POKE POKKEY,15
5430000 POKE POKKEY,15
5440000 POKE POKKEY,15
5450000 POKE POKKEY,15
5460000 POKE POKKEY,15
5470000 POKE POKKEY,15
5480000 POKE POKKEY,15
5490000 POKE POKKEY,15
5500000 POKE POKKEY,15
5510000 POKE POKKEY,15
5520000 POKE POKKEY,15
5530000 POKE POKKEY,15
5540000 POKE POKKEY,15
5550000 POKE POKKEY,15
5560000 POKE POKKEY,15
5570000 POKE POKKEY,15
5580000 POKE POKKEY,15
5590000 POKE POKKEY,15
5600000 POKE POKKEY,15
5610000 POKE POKKEY,15
5620000 POKE POKKEY,15
5630000 POKE POKKEY,15
5640000 POKE POKKEY,15
5650000 POKE POKKEY,15
5660000 POKE POKKEY,15
5670000 POKE POKKEY,15
5680000 POKE POKKEY,15
5690000 POKE POKKEY,15
5700000 POKE POKKEY,15
5710000 POKE POKKEY,15
5720000 POKE POKKEY,15
5730000 POKE POKKEY,15
5740000 POKE POKKEY,15
5750000 POKE POKKEY,15
5760000 POKE POKKEY,15
5770000 POKE POKKEY,15
5780000 POKE POKKEY,15
5790000 POKE POKKEY,15
5800000 POKE POKKEY,15
5810000 POKE POKKEY,15
5820000 POKE POKKEY,15
5830000 POKE POKKEY,15
5840000 POKE POKKEY,15
5850000 POKE POKKEY,15
5860000 POKE POKKEY,15
5870000 POKE POKKEY,15
5880000 POKE POKKEY,15
5890000 POKE POKKEY,15
5900000 POKE POKKEY,15
5910000 POKE POKKEY,15
5920000 POKE POKKEY,15
5930000 POKE POKKEY,15
5940000 POKE POKKEY,15
5950000 POKE POKKEY,15
5960000 POKE POKKEY,15
5970000 POKE POKKEY,15
5980000 POKE POKKEY,15
5990000 POKE POKKEY,15
6000000 POKE POKKEY,15
6100000 FOR J=0TO24:POKES1+J,0:NEXT:POKES1+2
6110000 POKE POKES1+5,15:POKES1+6,15
```

Ride the Wind Program

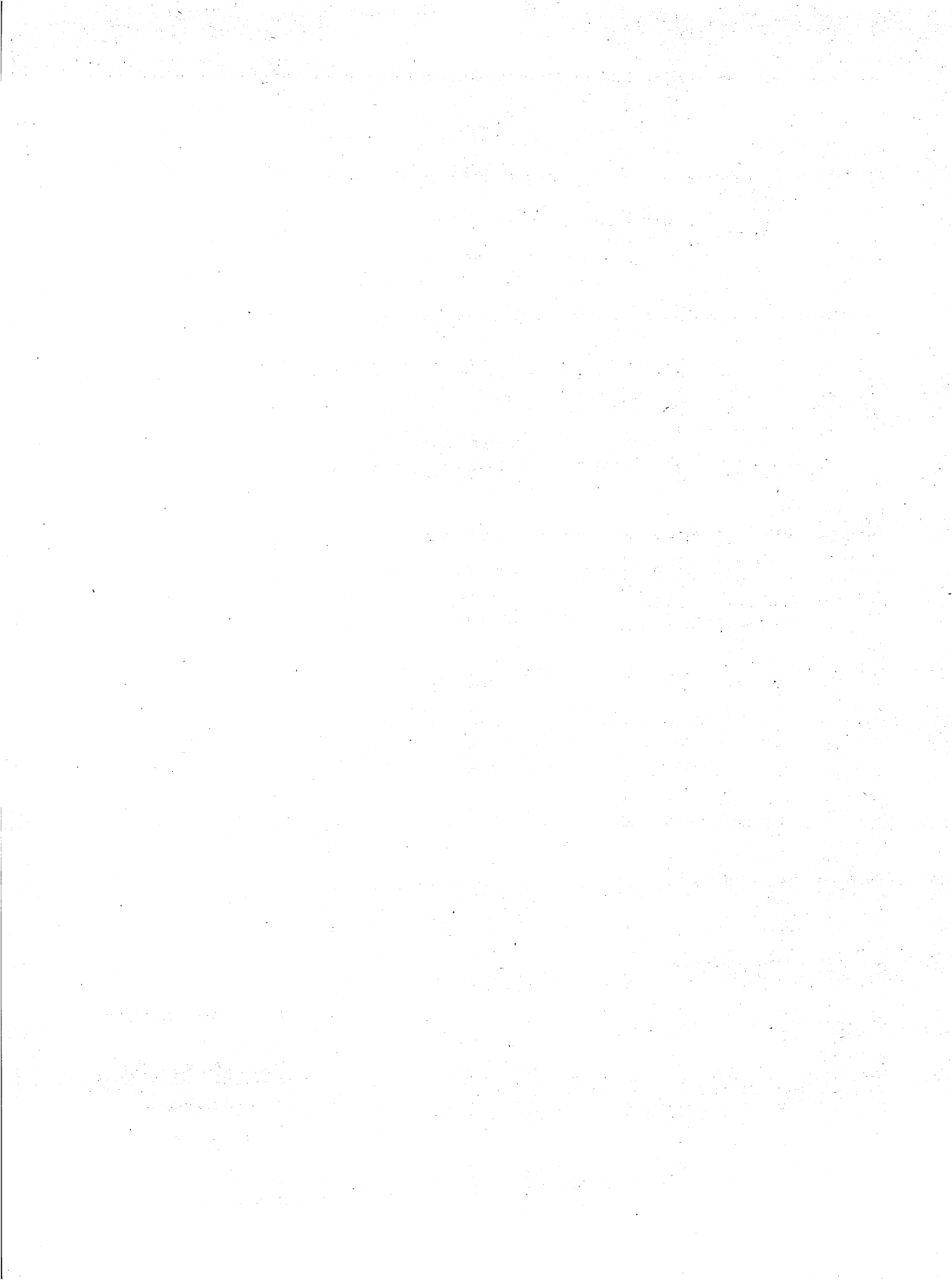



```

00000110 Q
00001100 P
00001110 R
00001120 S
00001130 T
00001140 U
00001150 V
00001160 W
00001170 X
00001180 Y
00001190 Z
00001200 _
00001210 `
00001220 a
00001230 b
00001240 c
00001250 d
00001260 e
00001270 f
00001280 g
00001290 h
00001300 i
00001310 j
00001320 k
00001330 l
00001340 m
00001350 n
00001360 o
00001370 p
00001380 q
00001390 r
00001400 s
00001410 t
00001420 u
00001430 v
00001440 w
00001450 x
00001460 y
00001470 z
00001480 {
00001490 }
00001500 ~
00001510 !
00001520 @
00001530 #
00001540 $
00001550 %
00001560 &
00001570 *
00001580 +
00001590 =
00001600 -
00001610 _
00001620 .
00001630 /
00001640 :
00001650 ;
00001660 <
00001670 >
00001680 [
00001690 ]
00001700 ^
00001710 *
00001720 +
00001730 -
00001740 /
00001750 :
00001760 ;
00001770 <
00001780 >
00001790 [
00001800 ]
00001810 ^
00001820 *
00001830 +
00001840 -
00001850 /
00001860 :
00001870 ;
00001880 <
00001890 >
00001900 [
00001910 ]
00001920 ^
00001930 *
00001940 +
00001950 -
00001960 /
00001970 :
00001980 ;
00001990 <
00002000 >
00002010 [
00002020 ]
00002030 ^
00002040 *
00002050 +
00002060 -
00002070 /
00002080 :
00002090 ;
00002100 <
00002110 >
00002120 [
00002130 ]
00002140 ^
00002150 *
00002160 +
00002170 -
00002180 /
00002190 :
00002200 ;
00002210 <
00002220 >
00002230 [
00002240 ]
00002250 ^
00002260 *
00002270 +
00002280 -
00002290 /
00002300 :
00002310 ;
00002320 <
00002330 >
00002340 [
00002350 ]
00002360 ^
00002370 *
00002380 +
00002390 -
00002400 /
00002410 :
00002420 ;
00002430 <
00002440 >
00002450 [
00002460 ]
00002470 ^
00002480 *
00002490 +
00002500 -
00002510 /
00002520 :
00002530 ;
00002540 <
00002550 >
00002560 [
00002570 ]
00002580 ^
00002590 *
00002600 +
00002610 -
00002620 /
00002630 :
00002640 ;
00002650 <
00002660 >
00002670 [
00002680 ]
00002690 ^
00002700 *
00002710 +
00002720 -
00002730 /
00002740 :
00002750 ;
00002760 <
00002770 >
00002780 [
00002790 ]
00002800 ^
00002810 *
00002820 +
00002830 -
00002840 /
00002850 :
00002860 ;
00002870 <
00002880 >
00002890 [
00002900 ]
00002910 ^
00002920 *
00002930 +
00002940 -
00002950 /
00002960 :
00002970 ;
00002980 <
00002990 >
00003000 [
00003010 ]
00003020 ^
00003030 *
00003040 +
00003050 -
00003060 /
00003070 :
00003080 ;
00003090 <
00003100 >
00003110 [
00003120 ]
00003130 ^
00003140 *
00003150 +
00003160 -
00003170 /
00003180 :
00003190 ;
00003200 <
00003210 >
00003220 [
00003230 ]
00003240 ^
00003250 *
00003260 +
00003270 -
00003280 /
00003290 :
00003300 ;
00003310 <
00003320 >
00003330 [
00003340 ]
00003350 ^
00003360 *
00003370 +
00003380 -
00003390 /
00003400 :
00003410 ;
00003420 <
00003430 >
00003440 [
00003450 ]
00003460 ^
00003470 *
00003480 +
00003490 -
00003500 /
00003510 :
00003520 ;
00003530 <
00003540 >
00003550 [
00003560 ]
00003570 ^
00003580 *
00003590 +
00003600 -
00003610 /
00003620 :
00003630 ;
00003640 <
00003650 >
00003660 [
00003670 ]
00003680 ^
00003690 *
00003700 +
00003710 -
00003720 /
00003730 :
00003740 ;
00003750 <
00003760 >
00003770 [
00003780 ]
00003790 ^
00003800 *
00003810 +
00003820 -
00003830 /
00003840 :
00003850 ;
00003860 <
00003870 >
00003880 [
00003890 ]
00003900 ^
00003910 *
00003920 +
00003930 -
00003940 /
00003950 :
00003960 ;
00003970 <
00003980 >
00003990 [
00004000 ]
00004010 ^
00004020 *
00004030 +
00004040 -
00004050 /
00004060 :
00004070 ;
00004080 <
00004090 >
00004100 [
00004110 ]
00004120 ^
00004130 *
00004140 +
00004150 -
00004160 /
00004170 :
00004180 ;
00004190 <
00004200 >
00004210 [
00004220 ]
00004230 ^
00004240 *
00004250 +
00004260 -
00004270 /
00004280 :
00004290 ;
00004300 <
00004310 >
00004320 [
00004330 ]
00004340 ^
00004350 *
00004360 +
00004370 -
00004380 /
00004390 :
00004400 ;
00004410 <
00004420 >
00004430 [
00004440 ]
00004450 ^
00004460 *
00004470 +
00004480 -
00004490 /
00004500 :
00004510 ;
00004520 <
00004530 >
00004540 [
00004550 ]
00004560 ^
00004570 *
00004580 +
00004590 -
00004600 /
00004610 :
00004620 ;
00004630 <
00004640 >
00004650 [
00004660 ]
00004670 ^
00004680 *
00004690 +
00004700 -
00004710 /
00004720 :
00004730 ;
00004740 <
00004750 >
00004760 [
00004770 ]
00004780 ^
00004790 *
00004800 +
00004810 -
00004820 /
00004830 :
00004840 ;
00004850 <
00004860 >
00004870 [
00004880 ]
00004890 ^
00004900 *
00004910 +
00004920 -
00004930 /
00004940 :
00004950 ;
00004960 <
00004970 >
00004980 [
00004990 ]
00005000 ^
00005010 *
00005020 +
00005030 -
00005040 /
00005050 :
00005060 ;
00005070 <
00005080 >
00005090 [
00005100 ]
00005110 ^
00005120 *
00005130 +
00005140 -
00005150 /
00005160 :
00005170 ;
00005180 <
00005190 >
00005200 [
00005210 ]
00005220 ^
00005230 *
00005240 +
00005250 -
00005260 /
00005270 :
00005280 ;
00005290 <
00005300 >
00005310 [
00005320 ]
00005330 ^
00005340 *
00005350 +
00005360 -
00005370 /
00005380 :
00005390 ;
00005400 <
00005410 >
00005420 [
00005430 ]
00005440 ^
00005450 *
00005460 +
00005470 -
00005480 /
00005490 :
00005500 ;
00005510 <
00005520 >
00005530 [
00005540 ]
00005550 ^
00005560 *
00005570 +
00005580 -
00005590 /
00005600 :
00005610 ;
00005620 <
00005630 >
00005640 [
00005650 ]
00005660 ^
00005670 *
00005680 +
00005690 -
00005700 /
00005710 :
00005720 ;
00005730 <
00005740 >
00005750 [
00005760 ]
00005770 ^
00005780 *
00005790 +
00005800 -
00005810 /
00005820 :
00005830 ;
00005840 <
00005850 >
00005860 [
00005870 ]
00005880 ^
00005890 *
00005900 +
00005910 -
00005920 /
00005930 :
00005940 ;
00005950 <
00005960 >
00005970 [
00005980 ]
00005990 ^
00006000 *

```

Death Valley Patrol Program

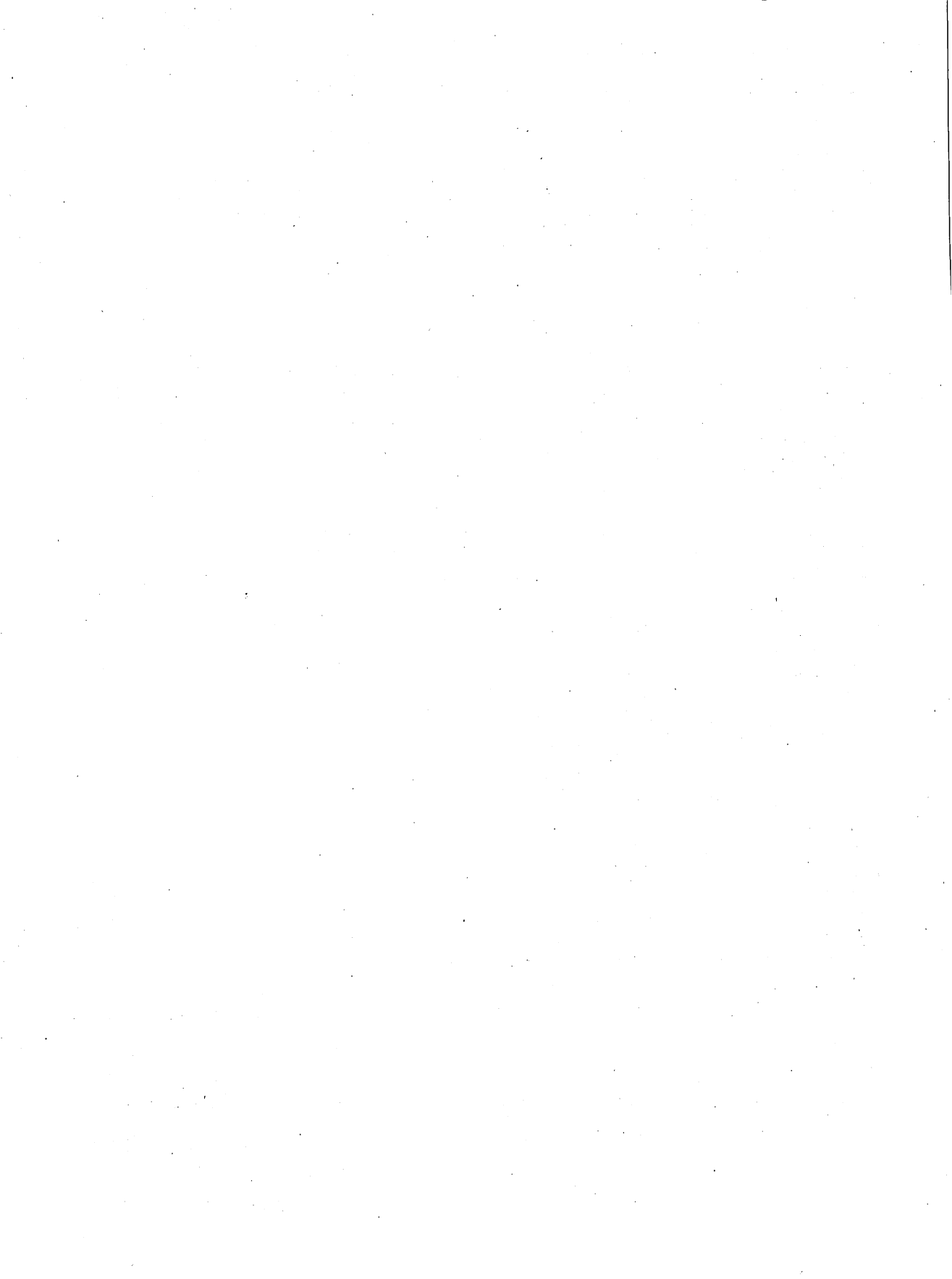



```

000070 DATA 16,68,18,34,66,8,66,33,96,132,
000071 DATA 0,0,1,0,120
000072 DATA 0,0,0,0,0,0,0,48,0,0,48,0,0,48
000073 DATA 48,0,0,48,0,0,48,0,0,252,0,3,2
000074 DATA 252,131,255,0,0,252,0,0,0,0,0,
000075 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
000076 DATA 0,12,0,0,14,0,0,14,0,0,12,0,0,
000077 DATA 12,0,0,12,0,0,60,0,0,255,0,0,2
000078 DATA 0,0,251,0,0,255,0,0,60,0,0,12,
000079 DATA 0,12,0,0,12,0,0,14,0,0,14,0,0,
000080 DATA 0,176,0,0,48,0,0,48,0,0,48,0,0,
000081 DATA 48,0,0,48,0,0,60,0,0,255,0,0,2
000082 DATA 255,0,239,0,0,255,0,0,60,0,0,4
000083 DATA 0,48,0,0,48,0,0,48,0,0,48,0,0,
000084 DATA 0,0,12,0,0,12,0,0,56,0,0,56,19
000085 DATA 0,192,12,3,0,3,3,0,0,252,0,0,2
000086 DATA 0,0,252,0,0,252,0,3,0,0,3,0,0,
000087 DATA 12,0,0,48,0,0,48,0,0,224,0,0,2
000088 DATA 48,0,0,48,0,0,44,0,0,12,0,0,3,
000089 DATA 0,12,0,192,48,0,192,192,0,63,0,
000090 DATA 0,0,63,0,0,63,0,0,0,192,0,0,19
000091 DATA 0,0,48,0,0,12,0,0,12,0,0,11,0,
000092 DATA 0,0,3,0,0,11,0,0,12,0,0,12,0,0,
000093 DATA 0,48,0,0,192,0,0,192,0,63,0,0,
000094 DATA 0,0,63,0,0,255,0,0,192,192,3,0,
000095 DATA 12,0,3,44,0,0,48,0,0,48,0,0,0,
000096 DATA 224,0,0,48,0,0,12,0,0,12,0,0,3
000097 DATA 0,0,0,192,0,0,192,0,0,63,0,0,6
000098 DATA 0,0,63,0,0,63,0,0,192,192,3,0,
000099 DATA 12,0,48
000100 DATA 48,0,48,0,0,12,0,0,14,0,0,3,0,
000101 DATA 0,0,0,0,20,0,2,170,120,2,170,1
000102 DATA 170,120,1,85,64,2,170,120,14,1
000103 DATA 176,254,170,191,14,170
000104 DATA 176,170,170,120,1,85,64,2,170,12
000105 DATA 170,120
000106 DATA 170,120,0,20,0,0,0,0,0,0,0,0,0,0
000107 DATA 15,2055,7054,172,0,1,140,15,241,
000108 DATA 106,177,1405,177,7054,177,172
000109 DATA 79,177,12,64,161,12,176,33,40,
000110 DATA 170,161,110,0,100
000111 DATA 100,120,100,106,100,4,240,64,2,255,255,254
000112 DATA 100,05,100,05,100,104,140,0,245,143,255,1
000113 DATA 100,0,100,0,100,170,0,0,2,170,120,10
000114 DATA 170,160,42
000115 DATA 100,160,42,170,160,42,170,221,170,0
000116 DATA 100,160,170,170,170,170,221
000117 DATA 100,160,170,170,170,170,42,170,160,42,1
000118 DATA 0,0,160,320,0,0,320,0,0,320,0,0,2,1
000119 DATA 0,0,0,0,0,0,0,0,0,12,0,192,51,
000120 DATA 0,0,48
000121 DATA 0,52,48,15,223,192,0,220,0,0,22
000122 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
000123 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
000124 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
000125 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
000126 DATA 3,255,0,3,171,0,0,252,0,0,48,0,
000127 DATA 0,48,0,0

```

Mission: Tobor Program




```

0100 PRINT"
0102 PRINT"
0104 PRINT"
0106 PRINT"
0108 PRINT"
0110 PRINT"
0112 PRINT"
0114 PRINT"
0116 PRINT"
0118 PRINT"
0120 PRINT"
0122 PRINT"
0124 PRINT"
0126 PRINT"
0128 PRINT"
0130 PRINT"
0132 PRINT"
0134 PRINT"
0136 PRINT"
0138 PRINT"
0140 PRINT"
0142 PRINT"
0144 PRINT"
0146 PRINT"
0148 PRINT"
0150 PRINT"
0152 PRINT"
0154 PRINT"
0156 PRINT"
0158 PRINT"
0160 POKEM1+1000,100:POKE56295,2
GOSUB3000
RETURN
REM
GAME OVER
PRINT"
FORI=0TO4:POKEM1+I,40:NEXT:FORI=1TO
OKM1+1,0:NEXT
FORJ=0TO3:POKE
FORJ=1TO500:NEXT
NEXTJ:GOSUB4000:FORJ=1TO
NEXTJ:POKE56300,VAND16:IFFR=16THENGOTO
PRINT"Q":RETURN
REM
PRINT INSTRUCTION SCREEN
PRINT"
IFFGTHENGOSUB3500
POKE53000,1:POKE53000,0
PRINT"
COSMO'S RESCUE
BY FRANK MIKULAST
(C) 1984 FANFARE HOUS
MOVEMENT : USE THE JOYST
BEGIN MOVING THE
ALIEN LEFT AND R
PRESS THE FIRE B
TO BEGIN MOVING
OBJECT : YOU MUST TRAVE
THE DANGEROUS CA
RESCUE THE PURPL
IF YOU CAN RETUR
TO THE LANDING P
YOU WILL BE SCOR
AMOUNT OF REMAIN
FUEL."
IFFGTHENFG=0:RETURN
PRINT"
PRESS THE
FIRE BU
START"

```

**Cosmo's
Rescue
Program**

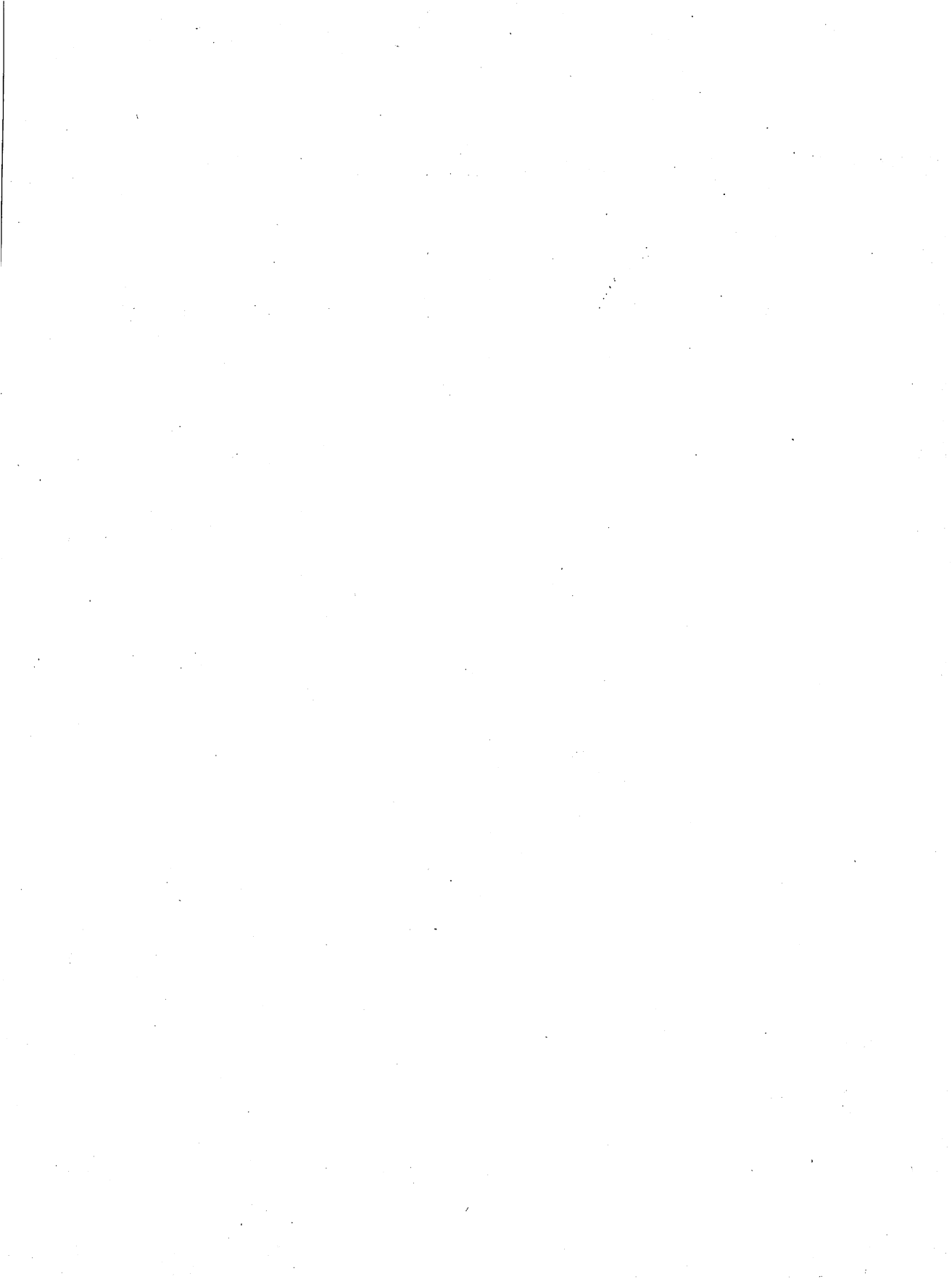



```

00000160 POKER,0:GX(J-7,I-2)=0
00000170 NEXT:GOTO 10
00000180 PRINT:PRINT"#####"TAB(6)"          YOU AR
00000190 HEAR#E
00000200 POKES53281,9:RETURN
00000210
00000220 REM PRINT TITLE
00000230
00000240 PRINT"[]":POKES53281,1:POKES53280,8:P
00000250 TTY,33
00000260 PRINT"#####"TAB(15)"MAZEMASTER
00000270
00000280 PRINT"#####"TAB(19)"BY"
00000290 PRINT"#####"TAB(12)"IGOR TULCHINSKY"
00000300 PRINT"#####"TAB(5)"AKC> 1984 FANF
00000310 CLR:INZ:
00000320 POKES54276,0:FORI=1TO1000:NEXT:POKE
00000330 TTY,33
00000340 FORI=1TO100:POKES53281,RND(8)*15:PO
00000350 KE54276,0:POKES54277,96:POKES5427
00000360 OK:RND(8)*100+100:NEXT
00000370 POKES54276,0:POKES54277,96:POKES54278,20:RETURN
00000380
00000390 REM PRINT GAMES MENU
00000400
00000410 POKES53281,1
00000420 PRINT"#####MAZEMAS
00000430 TTY,33:PRINT:PRINT
00000440 PRINT"
00000450 PRINT"
00000460 PRINT:PRINT:PRINT"#####ENTER
00000470 CHOICE"
00000480 GETCHOICE:
00000490 IF#="1"AND#("<"2"THEN22050
00000500 PRINT:PRINT"#####":INPUT#
00000510 IF#="1"THENCM=10
00000520 IF#="2"THENCM=4000
00000530 RETURN
00000540
00000550 REM GAME OVER
00000560
00000570 POKEXX
00000580 POKEXX
00000590 POKEXX
00000600 POKEXX
00000610 POKEXX
00000620 POKEXX
00000630 POKEXX
00000640 POKEXX
00000650 POKEXX
00000660 POKEXX
00000670 POKEXX
00000680 POKEXX
00000690 POKEXX
00000700 POKEXX
00000710 POKEXX
00000720 POKEXX
00000730 POKEXX
00000740 POKEXX
00000750 POKEXX
00000760 POKEXX
00000770 POKEXX
00000780 POKEXX
00000790 POKEXX
00000800 POKEXX
00000810 POKEXX
00000820 POKEXX
00000830 POKEXX
00000840 POKEXX
00000850 POKEXX
00000860 POKEXX
00000870 POKEXX
00000880 POKEXX
00000890 POKEXX
00000900 POKEXX
00000910 POKEXX
00000920 POKEXX
00000930 POKEXX
00000940 POKEXX
00000950 POKEXX
00000960 POKEXX
00000970 POKEXX
00000980 POKEXX
00000990 POKEXX
00001000 POKEXX
00001010 POKEXX
00001020 POKEXX
00001030 POKEXX
00001040 POKEXX
00001050 POKEXX
00001060 POKEXX
00001070 POKEXX
00001080 POKEXX
00001090 POKEXX
00001100 POKEXX
00001110 POKEXX
00001120 POKEXX
00001130 POKEXX
00001140 POKEXX
00001150 POKEXX
00001160 POKEXX
00001170 POKEXX
00001180 POKEXX
00001190 POKEXX
00001200 POKEXX
00001210 POKEXX
00001220 POKEXX
00001230 POKEXX
00001240 POKEXX
00001250 POKEXX
00001260 POKEXX
00001270 POKEXX
00001280 POKEXX
00001290 POKEXX
00001300 POKEXX
00001310 POKEXX
00001320 POKEXX
00001330 POKEXX
00001340 POKEXX
00001350 POKEXX
00001360 POKEXX
00001370 POKEXX
00001380 POKEXX
00001390 POKEXX
00001400 POKEXX
00001410 POKEXX
00001420 POKEXX
00001430 POKEXX
00001440 POKEXX
00001450 POKEXX
00001460 POKEXX
00001470 POKEXX
00001480 POKEXX
00001490 POKEXX
00001500 POKEXX
00001510 POKEXX
00001520 POKEXX
00001530 POKEXX
00001540 POKEXX
00001550 POKEXX
00001560 POKEXX
00001570 POKEXX
00001580 POKEXX
00001590 POKEXX
00001600 POKEXX
00001610 POKEXX
00001620 POKEXX
00001630 POKEXX
00001640 POKEXX
00001650 POKEXX
00001660 POKEXX
00001670 POKEXX
00001680 POKEXX
00001690 POKEXX
00001700 POKEXX
00001710 POKEXX
00001720 POKEXX
00001730 POKEXX
00001740 POKEXX
00001750 POKEXX
00001760 POKEXX
00001770 POKEXX
00001780 POKEXX
00001790 POKEXX
00001800 POKEXX
00001810 POKEXX
00001820 POKEXX
00001830 POKEXX
00001840 POKEXX
00001850 POKEXX
00001860 POKEXX
00001870 POKEXX
00001880 POKEXX
00001890 POKEXX
00001900 POKEXX
00001910 POKEXX
00001920 POKEXX
00001930 POKEXX
00001940 POKEXX
00001950 POKEXX
00001960 POKEXX
00001970 POKEXX
00001980 POKEXX
00001990 POKEXX
00002000 POKEXX

```

Mazemaster Program





CBS
CBS
CBS
CBS
CBS
CBS



Games Disk for
FANFARE HOUSE:
ARCADE GAMES FOR THE C64

ISBN 0-03-003664-X

CBS Educational & Professional Publishing
HRW 383 Madison Ave., New York, NY 10017

ARCADE GAMES FOR THE COMMODORE 64

Fanfare House, Inc.

While most other books promise you *hours* of fun on your Commodore 64, here's a book/disk package that promises you *years* . . .

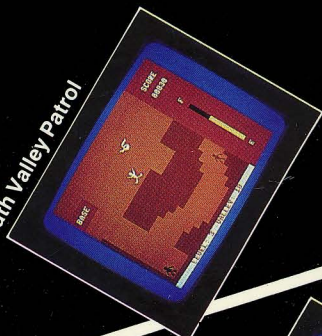
Play all 15 games or select just a few — you can change the rules in scores of ways — creating new games for years to come!

Whatever type of game you play — adventure, "space monsters," war games, or swords and sorcery — you can make them harder or easier; change the scoring mechanism; or modify the sprite characters.

All the games are coupled with individual programmer's notes to help beginners and pros learn each game, and programming hints to show quick methods and fast shortcuts.

DISK INSIDE! Includes 15 dynamite arcade games that you can turn into 100's more.

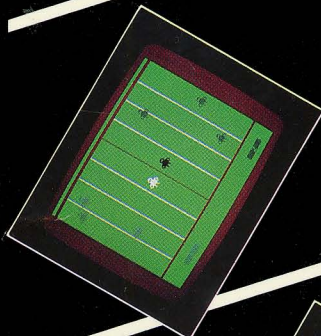
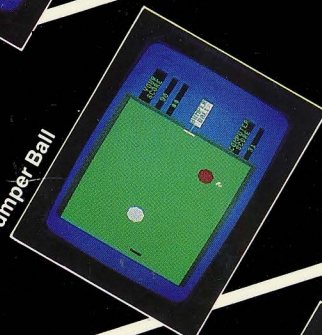
Death Valley Patrol



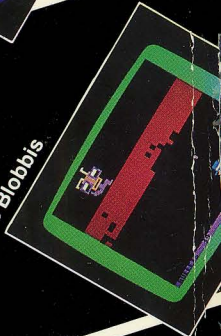
Ghost Hunt



Bumper Ball



The Blobbis



Space Crash

