LIMITED WARRANTY

RADIO SHACK Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

Except as provided herein, RADIO SHACK MAKES NO WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "SOFTWARE" LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES.

Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.
B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.
C. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.
D. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made.
E. All copyright notices shall be retained on all copies of the Software.

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

Robot Battle

Adventures in Programming

Radio Shack®

A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

10 9 8 7 6 5 4 3 2 1

Table of Contents

Introduction

You are entering a world where you have control over all events. Nothing occurs without your command. This is Robot Land and you are Person in Charge. Welcome to computer programming!

Required Equipment

- A Radio Shack TRS-80 Color Computer with at least 16K and connecting cable.
- A standard TV.
- A Color Robot Battle Program Pak™.
- A tape recorder and blank cassette, if you want to save your programs (optional).
- A compatible printer, such as the TRS-80 Line Printer VII or VIII, and connecting cables, if you want a printout of your programs (optional). **Warning**: Do not turn on your printer until you are ready to print. Never have your printer on when starting the Color Robot Battle program.

Note: In this manual, the tape recorder referred to is the Realistic® CTR-80A (or equivalent), available at your local Radio Shack store.

Overview

Color Robot Battle is an enjoyable introduction to the concepts and procedures of programming. The fun of video games and the thrill of discovery combine to make this program unique and rewarding.

Two robots are at your command. Tell them how and where to move, how to react to situations around them, when to fire lasers and missiles, when to stay and fight and when to run! You can even program them to dance!

The preferred approach to this program would be for two people to program one robot each, then challenge the other person's program in battle. Enjoy the challenges of computer programming while playing an exciting game with a friend, or by yourself.

Program offensive and defensive robots. Improve on your programs as you learn. However you play, with Color Robot Battle you'll finish a winner!

Guided Tour of Robot Land

You are embarking on a tour of Robot Land, where the language will be taught along the way. Follow this guide in sequence, and you will be traveling the Land and speaking fluent Robot in no time. Remember to stay on the tour's path; you don't want to get lost along the way!

Starting Out

To begin Color Robot Battle, insert the Program Pak and turn on your TRS-80 Color Computer. (Refer to your TRS-80 Color Computer Operation Manual for detailed information.) The opening screen appears:

COLOR ROBOT BATTLE

COPYRIGHT 1981

THE IMAGE PRODUCERS, INC.

PRESS <ENTER> TO BEGIN

PRESS D FOR DEMO

Normal entry into the program is achieved by pressing [ENTER] . To see a preprogrammed confrontation between two robots, type [D]. Then, to break out of demo mode, press [BREAK] . The menu for the two robots (Alpha and Omega) will appear. Everything you see will be explained in the course of this manual. To return to the opening screen, press the Reset button (located at the rear right of the Color Computer) and proceed as described below.

Press [ENTER] to see:

	LEFT EMPTY	RIGHT EMPTY
NEW	[NL]	[NR]
EDIT	EL	ER
SAVE	SL	SR
LOAD	[LL]	[LR]
COMPILE	CL	CR
BATTLE	B	

This menu shows the options (left column of terms) from which to choose as you set up your battle between the right and left robot. You can also later name the robots; names will be displayed where the words EMPTY appear now. (How to name the robots will be explained in the section "Naming the Robots" in Part II.)

The following sample programs have been designed to illustrate various aspects of the robots' capabilities.

Sample Program #1—Motion

Begin writing your first program by typing [N] [L], which informs the computer that you are about to write a new list of commands for the left robot. A green screen with a black bar across the center appears. This is Robot Battle's editor, where you enter all the commands your robot is to execute. Type the following commands, exactly as they appear here (you may use the left arrow key to backspace and type over any errors):

[F] [8] [:] [R] [8] [:] [B] [8] [:] [L] [8] [:] [H] [2]

The left robot is instructed in this sequence to go forward 8 steps, right 8 steps, back 8 steps, left 8 steps, then halt for 2 counts, or "clicks," as they are known in Robot Land. Each individual command is separated from the others with a colon (:). A line of commands can contain up to 32 characters (including spaces). See the Glossary of Robot Commands for more information on command abbreviations.

An automatic "loop" is built into the program. This means that as soon as the robot executes the last command, it goes back to the first command and repeats the sequence. Therefore, once the left robot stands still for two "clicks," it will go forward 8 steps, then right 8 steps, and so on.

Now enter some commands for the right robot. First press the [BREAK] key to exit the edit mode for the left robot; the previous command sequence is enough for the first program. You should now see the menu once again. The screen now shows "No Name" for the left robot. Type [N] [R] to indicate that you want to compose a new list of commands for the right robot. This returns you to the editing line (black line on green screen).

Type the following command sequence, exactly as it appears here:

[F] [8] [:] [L] [8] [:] [B] [8] [:] [R] [8] [:] [H] [2]

Press [BREAK] again to go to the menu. The screen now shows "No Name" for both robots. Now type [C] [L] to compile the program for the left robot, then type [C] [R] to compile the program for the right robot. The term "compile" means to translate the source program (the command sequence you just wrote) into the object code (language that the robots understand). In Robot Land, compiling must always be done before a program can be run; otherwise, the robots will not know what to do with your source programs.

Next, type the letter [B] to go to the battlefield. Your robots are not actually programmed to *battle* each other yet, but typing [B] takes you to the battlefield where the robots stand.

Action will not begin until one of the numeric keys along the top of the keyboard is pressed. Pressing the number [1] causes the robots to move fastest; pressing [9] makes them move very slowly. The slow speeds are good for seeing every step enacted as programmed; you may also find "bugs" (errors in the program) that should be ironed out. If you press [Ø] (zero) repeatedly, you can pace through the action step by step—and not miss a trick! Pick a speed and watch the robots move!

Your robots should now be marching on square tracks, with the left one moving clockwise and the right one moving counterclockwise. Your point of view is "overhead," looking at the tops of the robots. Each robot halts for two clicks upon completion of each square. The volume on your TV or monitor should be adjusted so you can hear the sound effects; sound can clue you in to details you may otherwise miss.

The direction a robot is facing is indicated by its firearm. If the weapon is pointing to the right, for example, the robot is facing right. This is an important fact to keep in mind because when you command the robot to turn left, it will turn to the left in relation to its firearm, *not* YOUR left (as you view the screen). The significance of keeping directions straight in your mind will become more clear as you become more involved with programming.

After watching the robots and experimenting with various speeds, press [BREAK] to return to the menu.

Sample Program #2— Direction & Weapon Control

Type [N] [L] to compose a new program for the left robot. You should now be back in the editor. This new program will replace the one you entered before.

Type:

[T] [1] [:] [X] [L]

This tells the left robot to turn once and fire a laser. Remember, the program will loop (repeat).

Press [BREAK], then type [N] [R] to enter a new program for the right robot. Once you are in the editor, type the following (press [ENTER] to start each new line). You can correct typing errors by using the [↑], [↓], [→] and [←] keys to move the cursor to the area you need to change. Then enter the information correctly.

[D] [Ø] [:] [X] [M]

[D] [2] [:] [X] [M]

[D] [4] [:] [X] [M]

[D] [6] [:] [X] [M]

The "D" commands indicate a specific direction for the robots to face. There are eight directions, or octants, in all. (Octant is defined in the Robot Programming Terminology section.) The octant facing straight up is Ø (zero). The other octants are numbered one through seven, going clockwise (see diagram on the next page). When you tell a robot to point in direction 2, for example, it will find that octant by taking the shortest route there. If the robot is facing D4 when it is told to go to D2, it will "back up" by way of D3 to reach its goal, rather than going clockwise all the way around the octants.

After the last command is executed, the program will loop back to the beginning.

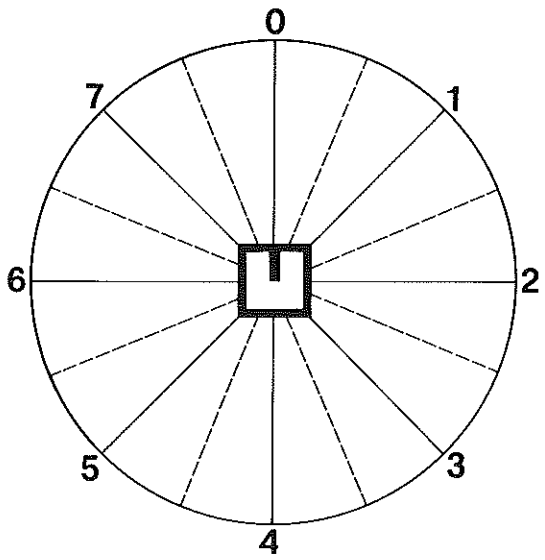

Figure 1. In this example, the robot is facing straight up (octant zero). If you programmed it to turn two octants clockwise (T2), it would turn twice and stop at octant 2. If, while facing zero, you programmed T-2, the robot would turn two octants counterclockwise and stop at octant 6. If you programmed the robot to face a specific direction (D2, for example), it would turn to octant 2 by way of the shortest route.

The "X" commands, separated from the "D" commands in the preceding program with a colon, tell the robot to fire (shoot). In this particular program, the right robot will fire missiles (XM). Remember, a robot can only fire effective weapons if the firearm is facing the target.

Now that both robots have been programmed, press [BREAK] to return to the menu. Compile the programs by typing [C] [L] and then [C] [R]; then type [B] to go to the battlefield. If the program returns you to the edit line instead of the battlefield, see the section called "If the Program Won't Compile," which follows.

Choose the speed at which you would like to view the action. Slower speeds (higher numbers) are best to demonstrate what is happening. You can always switch to a faster speed after observing a few moments.

Keep in mind what your program directions were. You can review the programs you entered by re-reading the previous pages in this manual. The left robot should be making a one octant clockwise turn, then firing a laser and repeating this sequence. Notice the changes in direction as indicated by the position of the firearm.

Meanwhile, the right robot is turning to each programmed direction, firing a missile and repeating the sequence.

Every time the right robot fires a missile, it "rests" a few counts before turning and firing another missile. It takes a moment for a robot to regain its strength after it fires a missile. Lasers don't take up much energy, however. This is more noticable on slow speeds (and with more volume).

If the Program Won't Compile

If for any reason the computer does not understand what you are attempting to do, it can not compile your program. This almost certainly means an error has occurred in the writing of the program. In this event, the editor will appear, with the green cursor (position indicator) showing you where the problem is—or at least begins!

If your program does not compile, check your work carefully. Make sure commands are stated in a way the computer understands. For example, if you typed "FM" for Fire Missile rather than [X] [M] (the code for Fire Missile), the computer will not be able to execute a command it does not recognize. The more involved your programs become, the greater the possibility of error becomes. Don't worry! For every problem you may encounter, there is a solution!

Victorious Robot

If you let the programs run for a length of time, one robot will eventually triumph over the other. The robot with energy to spare is the winner. The energy levels are represented by the two lines on the bottom of the screen during a battle. The color of each energy level line matches the color of the corresponding robot. Energy levels are affected in various ways, as described in the following section.

Cost of Living in Robot Land

At the beginning of an encounter on the battlefield, each robot has 62 units of energy. Both energy levels on the bottom of the screen represent full energy capacity. The number of units is not printed anywhere; they are preset to 62 units at the start of any "battle."

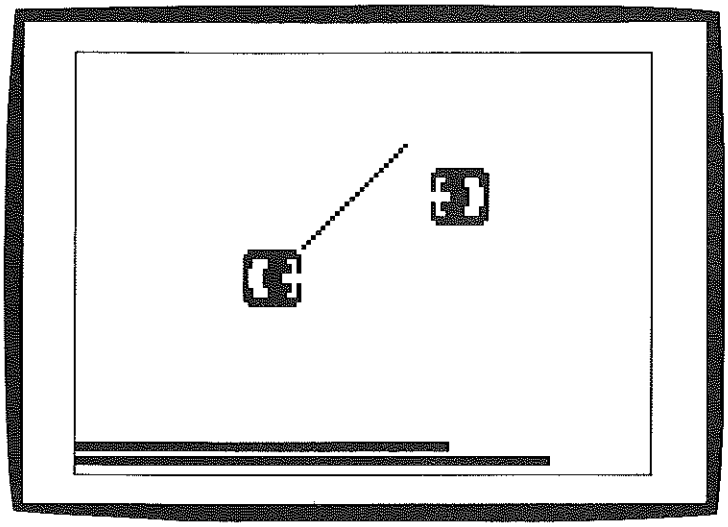

Figure 2. Robots in battle with varying energy levels.

Many activities during battle cost the robots a certain amount of energy. For example, running into a wall depletes one third of a robot energy unit. The "cost of living," recorded in energy units (or fractions of units), is as follows:

When a Robot:	Cost in Energy Units is:
Is hit by a missile	2.00
Is hit by a laser	0.20
Runs into a wall	0.33
Runs into a robot	0.40
Shoots a missile	0.33
Shoots a laser	0.10

Clearly, getting hit by a missile is the single most costly event in Robot Land. Notice how the energy levels decrease as the battle progresses.

Ultimately, one robot runs out of energy altogether. The screen changes color; the winning robot then springs into a "victory dance," firing lasers in all directions and spinning in circles. This celebration continues until the BREAK key is pressed.

Sample Program #3—Conditionals

The term "conditional" means that *if* a robot sees or senses a specified object, it *does* something in response. For example, you can issue a command stating that if a robot senses a wall nearby, it should make a turn to avoid bumping into the wall.

Conditional commands are preceded by an equal sign (=). At the menu (press BREAK to stop the last program if you haven't already done so), type N L to program the left robot. Once in the editor, type in the following program exactly, pressing ENTER after each line:

```
=R:XM

=W:T2

=?:T1

F8
```

This program instructs the left robot to fire a missile *if* it sees a robot and turn two octants clockwise *if* it sees a wall. The third command is interesting. It is a random command, indicated by the question mark. The computer executes a random command approximately half the times it finds one. Therefore, the =?:T1 command means that if the robot "feels like it," it can turn one octant clockwise. Random commands add an element of surprise on the battlefield. Any action the robots can perform can be programmed to occur randomly.

The last command in the above list is a simple one you probably recognize: move forward 8 steps. This is also the only command in this program that is a "sure thing." The others depend on certain factors to be present before they can be carried out. When there are no walls or robots in front of the left robot, it will continue to march ahead 8 steps, with, of course, an occasional random turn.

Here is the corresponding program for the right robot (remember to press BREAK to go to the menu, then type N R for a new program for the right robot). Type the following exactly, pressing ENTER at the end of each line:

```
=M:XL

=W:T-2

=R:XM

F6
```

This programs the right robot to fire a laser when it sees a missile (lasers can destroy missiles, but the reverse is not true); turn counterclockwise two octants when it sees a wall (counterclockwise turns are indicated by a T followed by a negative number; in this case, -2); fire a missile when it sees the other robot, and move forward six steps.

Press [BREAK] to go to the menu, then compile both robot programs. If you have trouble compiling the programs, check each command carefully. Then type [B] and select a speed to see the results.

Sample Program #4—CALL, GOTO & Labels

CALL and GOTO are commands used to call up or go to a particular part of a program. These commands allow control and versatility in your programming.

CALL and GOTO commands must have something to call or go to, so labels are used. A line of commands preceded by a label can be thought of as a program within the larger program, or a subroutine (offshoot of the main routine).

The following program should help illustrate these terms. First, go to the editor for the left robot. You should be able to find your way there now that you have done it a few times. Enter this program exactly as it is written here, pressing [ENTER] at the end of each line:

```
ROB> =R:XL:GROB

WAL> =W:T1:GWAL

START> C ROB:C WAL:F8:=?:T1

GSTART
```

Now, go to the editor for the right robot and type in this routine, pressing [ENTER] after each line:

```
ROB> =R:XM

WAL> =W:T-2

START> C ROB:C WAL:F8:=?:T1

GSTART
```

Always remember to put a [>] symbol immediately after any label in your programs. Labels must appear at the beginning of a line, and cannot excede six characters in length. In these last two programs, each line begins with a label, and each labeled line is "called" by another line in the program.

The main routines (the ones with the START> labels) are the same for both robots. The program differences lie in the subroutines that are called by the main routines.

Each robot's main routine says first to "call robot" (CROB). The program then looks for the ROB label and proceeds to carry out the commands contained in that subroutine. In the case of the left robot's current program, the commands in the ROB subroutine say to fire a laser if the other robot is in view, and to keep doing so until the other robot is out of range.

Once the robot has executed the ROB commands, it returns to the main program routine (START>) which says to next see (or call) the subroutine WAL. The line of commands labeled WAL tells the robot to turn one octant clockwise if there is a wall in its way. The robot will continue to turn until its path is clear of walls.

After the WAL routine has been accomplished, the robot is programmed to go forward eight steps, (occasionally) make a clockwise turn, and ultimately return to START.

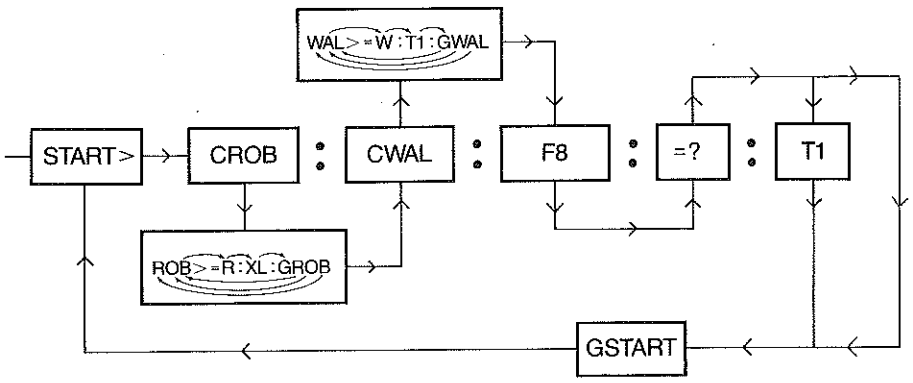

Figure 3. Flow chart of current routine for the left robot.

These labels can be anything you find appropriate to indicate the routine that follows. WAL and ROB are just two simple examples of logical labeling. One steadfast rule, however, is that if you use labels, your main routine must always be labeled START>. The computer always hunts for this particular label to get the program started and keep it looping. (This only applies to programs using labels; you have already written programs that loop successfully without *any* labels.) More label examples are provided in the section called Robot Programming Terminology, under "label."

The order in which these labeled subroutines appear makes no difference as to the order in which they will be executed. As you can see in the example, the main routine is not at the top of the program; the computer knows to start at the START> line regardless of where it appears. Similarly, when the line labeled START is being executed, the computer will find the routines labeled ROB and WAL regardless of where they appear.

The sequence of labeled routines is totally up to you. You may, for example, decide first that you want a robot to fire a laser at the other robot if it's in view; therefore, that is the first thing you happen to write into the editor. As long as your call labels match the labels they are calling *exactly,* the program will flow in its proper sequence. (For example, to call the ROB routine, you *must* write [C][R][O][B], *not* CROBOT.) A program with mismatched labels will not compile.

The difference between CALL commands and GOTO commands may not seem obvious at first glance. CALL will call up a labeled routine, execute it, and return to the line from which it originally came to find the next command to activate. In the main routines for the two current programs, the robot is instructed to CALL the ROB routine, and return to finish the remainder of the main routine once the ROB commands have been carried out.

Now look at the ROB subroutine for the left robot; it ends with GROB. This means continue to execute the ROB line until no longer possible (when the other robot is no longer within range to fire at with lasers). If you put a GOTO directive in the middle of a line of commands, the program might never return to finish the rest of the line. CALL commands remind the program (in our case, the robot) where it came from and where it should return once it has executed other routines.

Be careful when using CALL, GOTO or conditionals; otherwise, you may find your robots stuck in a loop. Refer, for example, to the WAL subroutine in the last sample program for the left robot. If that line did not include the =W conditional, the robot would go to the WAL> label, make one clockwise turn and repeat endlessly. As it is, the the robot will turn only if a wall is in its way. Then it returns to the main routine (START>) to carry out the next command. So, if you notice your robot repeating the same limited moves, check these three important factors!

This concludes your basic guided tour through Robot Land. You now have an understanding of basic programming terms and how they interact to create a "living" robot. The next section provides further details that did not directly apply to the sample programs provided in these sections.

By the way—if your curiosity has caused you to venture into the program editor for the demo robots, you may have recognized Sample Program #4 as one and the same!

More About Robots and Program Control

Now that you have "toured" through the basic fundamentals of robot programming, you are ready to take on a few more details. The information included here will allow you to expand on what you already know; you may yet become a programming wizard!

Items on the Menu

Take another look at the program menu. Notice how many of the choices are in inverse video (light letters in black boxes). When an item on the menu is in inverse characters, that function is an available choice to make at that time. For example, the B will not appear inversed until the programs have been compiled because you cannot go to battle without first having problem-free programs to execute. (Remember, the program will not compile if there are errors in command or label usage.)

NL and NR, as explained in the sample programs, take you to the editor to write new programs for the left and right robots.

Edit

Beneath NL and NR on the menu are EL and ER. Type these letters to return to an existing program to make changes (edits). Once you've written a program and have seen it in action, you will probably see things on the screen that were unexpected during programming. This is when you'll want to add, remove or otherwise alter portions of your program.

Changing and adding to programs is the most valuable learning experience you can get. This is what Color Robot Battle is all about! A complete account of all you need to know about editing is contained in the section, "Editor Line Commands."

Save

The next option on the menu is Save. Only one program for each robot can be in memory at a time, which would be the most recently written program. No robot program will be in memory if the computer has been turned off or if other programs have been run since your last robot battle.

The Save feature is a great convenience. Save programs you consider to be especially effective, or start saving programs right away to keep track of your progress as you become more adept at programming.

When you are recording a program onto tape, be sure the tape in the cassette is past the "leader" tape (different colored cellophane at the beginning and end of each cassette). Leader tape does not accept a recording signal; only the dark (magnetic) tape is effective. Set the counter to ∅. The first program you record can start at counter number ∅. Subsequently saved programs should be started at counter numbers past the end of the last program.

To save a program currently in the Robot Battle editor, first make sure your cassette recorder is properly connected to the Color Computer. (See your TRS-80 Color Computer Operation Manual for detailed instructions.) Type [S] [L] to save your left robot's program or [S] [R] to save the right robot's program. The screen will show:

```
INSERT SOURCE TAPE
REWIND TAPE
PRESS PLAY AND RECORD
PRESS <ENTER>
```

After you have followed these instructions, the program will be stored on tape and the screen will return to the menu. You will want to keep track of where programs start and end on the cassette for future reference.

Load

To run battles that you have saved onto cassette, activate the load feature. The loading process always overrides any other program in the editor; you may want to save the existing program before loading another.

For your own convenience, keep track of the counter number where each program has been saved. Check the counter on the recorder and jot down the program and its location for future reference. Build a portfolio of programs!

Type [L] [L] to load the left robot program into the computer; [L] [R] to load the right robot. The screen will show:

```
INSERT SOURCE TAPE
REWIND TAPE
PRESS PLAY
PRESS <ENTER>
```

Fast forward the tape to the counter number of the program you want to load. Follow the rest of the screen instructions; the program will be loaded. The screen will return to the menu when the loading is done.

Comments

Words or statements can be included in command lines that will *not* be acknowledged by the computer as something to be carried out. Comments are usually prompts to remind you what a program, or portion thereof (such as labels) is about.

To write a comment in a program, type an asterisk ([*]) either at the end of a command line or on a line of its own, followed by the comment (a key word or phrase). The computer interprets anything written after an asterisk as a note for you, and will not attempt to read it or deal with it in any way. (Do not put comments in mid-command line because any commands following a comment on a line will be ignored, with the exception of robot names, as described next.)

Naming the Robots

Names are handled similarly to comments. They are words typed into the editor, but are not commands for the robots to execute.

When you enter the editor, type an asterisk (*) before anything else. Immediately following the asterisk, type the name of your robot, such as:

* R E D

The rule here is: to name a robot, enter the desired name as you would enter a comment, but it must be the *first eight characters* (or less) on the *first line* of the program for that robot. The name will then appear at the top of the menu where "empty" appears otherwise. In the event that *more* than eight characters follow an asterisk on the first line, only the first eight will appear as the name. If you have an existing program which has not been named, you can name it by putting the program (in the editor) on the screen, creating a new first line with the @ command (see Editor Line Commands section in this manual) and typing an asterisk followed by the name.

Scan Sensors

All robot sensing, scanning and activity occurs in an octagonal pattern: robots turn in octant segments, face an octant's direction, walk down octant avenues, and fire lasers and missiles down octant paths.

Scanning is the only talent a robot performs in all directions at once. Scanning is a conditional command similar to those mentioned in the Conditionals section of Part I, page 10 of this manual (if there is a robot, fire a missile, etc.). The nature of the scan conditional command (S) is much more general, however.

An S in a line of commands tells the robot to activate its sensors and see if anything is there, in all directions within range. If it finds something, the program may then proceed to have the robot methodically comb through the conditionals:

=S Is something there? Sensors decide yes or no (= represents true; # represents false). If yes (true), proceed with a sequence such as:

=R If it's a robot... (insert next command)

=W If it's a wall... (insert next command)

#R If it's not a robot... (what would you like it to do?) and so on.

In other words, the S conditional is a qualifier or signal for the program to proceed to determine WHAT object is in sensory range, and then react to that object (or those objects) in whatever manner you have indicated in your program (fire a laser, turn and run, etc.).

Firing Ranges

Like all other robot activities, octants provide the guidelines for ranges in which ammunition may be fired.

Lasers

When a robot is programmed to fire a laser (XL) at some target within sensor range, the ammunition fires down an octant, but will veer off its straight-line course to strike its target as close to center as possible. Veering is limited, however; it can only move halfway between its original octant and the octant beside it.

When program processing encounters XL in the command sequence, it will automatically deal with XL in accordance with a predetermined priority. Upon reading any XL command (such as =R:XL), the program first asks, "Is a missile in this octant?" If a missile is in that octant range, a laser will be fired at the missile, before dealing with the robot that is also in the octant. This built-in self defense mechanism deals with the most immediate threat (a missile heading straight for the robot) before it executes =R:XL. Even though the robot is programmed to fire a laser at the other robot (if it's in range), the laser will deal with the most immediate problem *first,* which would be to get the missile before the missile gets *it.* It will then deal with any robots within its octant range.

There is one potential problem with the automatic search for a missile in an octant. You may at some point command a missile to be fired, followed by a laser. What may happen in this event is that your missile will be blasted by your own laser! Be cautious!

Laser fire is instantaneous in its connection with its target. A missile is slower and more limited in its applications, as described in the following section.

Missiles

Missiles do not have the capacity to intercept lasers or other missiles. A missile is slower to reach its target than a laser. This improves the chance for the targeted robot to save itself by responding with a laser, if it had been programmed =M:XL. Moreover, a robot can actually step out of the way of a missile, given enough space between robots at the time the missile is fired.

Click, Bang, Boom and Beep

The best way to familiarize yourself with the sounds these robots make is to program various activities, turn up the volume and press a high number on the keyboard to see and hear each command executed individually. The sound effects in Color Robot Battle coordinate with the action to make each movement clear to the programmer—and of course more fun to watch!

Robot activities are accompanied by the following sounds:

For each step, you hear ..... "Chcka"

When a robot turns, you hear ..... "Tch"

When a robot halts ..... "Click"

Scanning sounds like ..... "Blip-blip"

Firing a laser ..... "Beep"

Firing a missile ..... "Bang"

Robot collisions or ammunition hit ..... "Boom"

A good battle can generate a lot of commotion!

Printing Your Programs

To obtain a printed copy of a program, you must have a compatible printer connected to your TRS-80 Color Computer. When the program to be printed is displayed on the screen, turn on your printer. (Do not have it on until you are ready to use it.) Type [“] [P] (by simultaneously pressing the [SHIFT] , [“] and [P] keys) on the keyboard. When printing is complete, turn the printer off. This is necessary since some printers respond to the robot sounds by printing meaningless characters.

Reference Section

Editor Line Commands

These are the commands to use when you are in the editor (creating or changing robot programs).

→ or ←	These keys move the cursor right or left on the line without deleting.
SHIFT → or ←	The SHIFT key pressed simultaneously with the right or left arrow moves the cursor to the right or left end of a line.
↑ or ↓	Press the ↑ or ↓ to move text up or down one line.
SHIFT ↑ or ↓	The SHIFT key pressed simultaneously with ↑ or ↓ moves text up or down 1/2 page.
CLEAR	The CLEAR key deletes all characters from the current cursor position to the end of the line.
SHIFT & CLEAR	These keys, pressed simultaneously, delete the entire line upon which the cursor is positioned.
ENTER	Press ENTER to move to the next line and to create one if none exists.
@	Press @ to create a blank line above the current line, indicated by the cursor position.
BREAK	Press BREAK to exit the editor.
" P	This combination of keys (SHIFT , " , P) will start a printout of the program displayed on the screen. To print your programs onto paper, a compatible printer must be turned on and properly connected to your computer before you press " P.

Glossary of Robot Commands

:	Commands on the same line must be separated by a colon. Example—F2:R8.
F(n)	Forward n steps (Robots may take from one to eight steps.). Example—F2.
B(n)	Backward n steps. Example—B2.
L(n)	Left n steps. Example—L3.
R(n)	Right n steps. Example—R8.
H(n)	Halt n "clicks" (robot time units). Example—H8.
T(n)	Turn clockwise n octants. Example—T2.
T(-n)	Turn counterclockwise n octants. Example—T-3.
D(0-7)	Face direction (octant) ∅ through 7. Example—D6.
C	Call a line. Example—CLINE.
G	Go to a line. Example—GLINE.
START>	Unique label which always marks the beginning point of a program using labels.
XM	Fire missile.
XL	Fire laser.
=	True

#	Not true
=? or #?	Randomly evaluates a condition to be true (=?) or not true (#?) and carries out the command approximately half of the time. Neither the programmer nor the opponent knows when the condition will be true or false.
S	Sensory mechanism scans in all directions.
=S	Is anything out there?
R	Robot
=R	If there is a robot in this octant...
#R	If there is not a robot in this octant...
M	Missile
=M	If there is a missile in this octant...
#M	If there is not a missile in this octant...
W	Wall
=W	If there is a wall within sensory range...
#W	If there is not a wall within sensory range...
= (conditional)	Continue processing this line if the conditional is true.
# (conditional)	Continue processing this line if the conditional is false.

Robot Programming Terminology

Call	This directs the flow of events to a labeled subroutine (offshoot) of the main program. Once the subroutine is executed, the computer looks back to the spot where the call command occurred and proceeds to finish executing that command line. It is designated by C (subroutine label).
Comment	Word or phrase which is not related to the command sequence of the program; usually a key word describing a routine or portion thereof. It is designated by a * (comment).
Compile	Computer translation of information you write (source program) into information the computer understands and can work with (object code). It is designated by CL (Compile Left Robot program) or CR (Compile Right Robot program).
Conditional	If a certain condition is determined to be true, a particular (programmed) response will ensue. For example, =M:XL states that if a robot sees a missile in the octant it is currently viewing, it will fire a laser (down that octant). It is designated by =.
Editor	The Color Robot Battle editor is where communication between you and your robots occurs. All commands are first typed into the editor. Then you compile them to bring a program "to life."
Goto	Instructs processing to go to a particular label and execute the following command line. Goto, unlike Call, does not return to the point in the program where Goto occured; the program carries on from the last command on the line it "went to." (See flow chart on page 12.) It is designated by G (label).
Inverse Video	Characters or images on a TV or monitor display that are light on a dark background; reversed from the usual dark characters on a light background.

Label

Labels mark or name a subroutine within the program. The Call and Goto directives must have labels to search for in order to execute that line of the program. Labels can be anything you want them to be, up to a maximum of six characters. Here is a sample list of labels to help you formulate ideas:

LOOK> SCAN> SHOOT> WALL> TURN>

KILL> RUN> LOOP> UP> DOWN>

Labels don't have to be real words. However, it's less confusing if your labels do actually indicate something, such as the type of action programmed on the labeled line. If you do venture into "creative" labeling, you have the option of using comments to remind you how a line of commands performs.

Loop

A program's process of returning to a previously executed line of commands and repeating the command sequence automatically.

Menu

List of items from which to choose.

Object Code

The result of your "English" robot program translated into a language (code) that the computer understands and follows.

Octants

The pattern in which the robots move, sense and shoot, resembling a pie cut in eighths (see diagram on page 7).

Program

A set of instructions (commands) which dictate action.

Routine

Program

Source Program

Line or lines of commands written in "English" robot language. The computer can translate them into an object code and execute the program.

Inhabitants of Robot Land

You've toured Robot Land, learned some Robot language and played Robot games. It's only fair that you have an opportunity to meet more Robots! Here is the "welcoming committee" of Robot Land, pleased to perform for your amusement and entertainment. They can do more than shoot lasers and missiles, as you'll soon see!

Here's a fun challenge: below are some robot characters anxiously awaiting the chance to "come alive." Read the following stories, then see if you can write a program for each story. Make the robots chase each other, play hide and seek, trip the light fantastic—whatever your imagination creates!

These suggested programs are "just for fun." Perhaps they'll give you, a potential programming champion, ideas for bigger and better robot programs. Discover more about robot behavior and how you can better control it.

Program your robots to act like the characters in the stories. Each robot performs a simple action, using a few commands. Remember, you must give these robots "brains." To become a super programmer, all an ordinary person needs is a Color Computer and a basic knowledge of Robot Language.

You have the power to make these robots do almost anything, except perhaps step out of the screen into the three-dimensional world (but one day, who knows . . . ?).

At the Rink . . .

The two robots are enjoying a leisurely skating party . They are great at figure eights, together as a pair or skating independently, weaving in and out of each other's path. Are they coordinated enough to avoid bumping into each other?

The Chase . . .

The Red robot is quietly minding its own business when its sharp scanners sense an intruder. The Blue robot is sneaky, but Red escapes in the nick of time. The chase begins! Will Blue get Red?

Can you write a program to start a fast and frantic chase around the screen? You'll need a few commands to get those robots zipping around. Ready, set, GO!

Hide and Seek...

The robots' "eyes and ears" are their super-sensors. Hide and seek robot-style is more exciting—when *you* write the program! Create a new robot game and watch their adventures as Blue tries to stay clear of Red's scan range. If Blue gets caught, he's "it!" and the game begins anew.

Imagine the robots playing hide and seek; then, see if you can make your ideas work in a program. You'll need to tell your robots to use their scanners and make plenty of quick turns. Charge up those sensors and don't bump into any walls!

Trip the Light Fantastic in Robot Land...

Transform an ordinary TV or monitor into a ballroom and - voila - two ordinary robots become dancing dynamos! Can they dance? These robots can swing, tango, disco... they can even do the Bunny Hop! Put on your favorite record and get on down to the Robot Hop!

Clowning Around...

After a heavy battle, robots like to relax and horse around a bit. Red and Blue are darting around, turning cartwheels and blasting a few lasers for fun. Make them try to outdo each other in crazy antics!

How funny can your robots be? What hijinks can you program for them to perform? Try writing a program to turn those robots into real zanies. Robots are rough and ready in battle, but your program will show that they're really just a couple of clowns!

—HAPPY PROGRAMMING—