ATARI® PROGRAM exchange

AUTHOR'S GUIDE

MAY 1, 1981

TRADEMARKS OF ATARI

The following are trademarks of Atari, Inc.

ATARI
ATARI 400 Home Computer
ATARI 800 Home Computer
ATARI 410 Program Recorder
ATARI 810 Disk Drive
ATARI 820 40-Column Printer
ATARI 822 Thermal Printer
ATARI 825 80-Column Printer
ATARI 830 Acoustic Modem
ATARI 850 Interface Module

****

Distributed by

The ATARI Program Exchange
P. O. Box 427
155 Moffett Park Drive, B-1
Sunnyvale, CA 94086

To request an APX Software Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/745-5535.

****

ATARI PROGRAM EXCHANGE

AUTHOR'S GUIDE

Submitting your user instructions. When you submit your program to the ATARI Program Exchange (APX) for review, you also submit your program's user documentation. APX wants to offer ATARI Personal Computer owners not only well-written programs but also well-written user instructions! So, when we review your program, we also review your documentation. We'd like to have your manual on diskette, with the file formatted using either the FORMS program (available through APX) or the ATARI Word Processor, and accompanied by a printed copy of your manual. However, we'll also accept a typed copy of your user instructions.

The Author's Guide. This guide can help you organize and write your user documentation. It outlines suggested topics to cover for two kinds of programs: games (see pages 1 - 5) and all other programs (see pages 6 - 12). It also includes examples of ways to explain standard types of information, such as describing menu-driven, controller-driven, and command-driven programs. If you're so well-organized that you've already written complete user instructions, then check that they cover the topics mentioned in this guide. In addition to the user manual sections described in the following outlines, we'd welcome a table of contents and a list of figures and/or tables for your manual.

Finally, the guide also contains programming procedures and suggestions that can make your program easy for others to use (see pages 13 - 14). Consider these suggestions for your current program as well as for future programs you plan to submit to APX. If you submit your program on diskette, we'd prefer that it run under DOS II, as that version becomes available.

Retain copies for yourself. We won't be returning either your user manual or your program, regardless of the review decision. Therefore, keep copies of your program and all documentation.

Copyright information. ATARI won't copyright software distributed by APX. Therefore, if you want to copyright your software yourself, write to the Register of Copyrights, Copyright Office, Library of Congress, Washington, D.C. 20559.

When your program and user instructions are ready for review, fill out the Program Submission Form accompanying this guide and mail the following items:

Program cassette/diskette
Program Submission Form
Software Submittal Agreement (2 signed copies)
User instructions
Source code in machine-readable form

to:

THE ATARI PROGRAM EXCHANGE
P. O. BOX 427
1196 BORREGAS AVENUE
SUNNYVALE, CALIFORNIA 94086

GAME SOFTWARE OUTLINE

1 GENERAL DESCRIPTION

Game overview

This section orients the user. Try to keep your description under 200 words. Briefly mention your game's major features--such as the general type of game, how it's played (e.g., with joysticks), the object of the game, and any options and game variations--but save the details for later. An example is:

> AVALANCHE* is a paddle game of speed and dexterity. An avalanche of rocks is poised overhead, ready to pummel you. Score as many points as you can by absorbing the falling rocks with your shields before the rocks hit the ground. Use a paddle controller to maneuver your shields back and forth across the screen. Your chosen bonus level determines the number of misses you're allowed and the number of points you must obtain to earn an extra turn. Compete against your own best score or against another player.

Minimum RAM and accessories

A simple way to organize this information is to list it. Include here both required and optional accessories. For example:

```
REQUIRED ACCESSORIES
     16K RAM for cassette version
     24K RAM for diskette version
     ATARI 410 Program Recorder for cassette
     ATARI 810 Disk Drive for diskette
     ATARI BASIC Language Cartridge

OPTIONAL ACCESSORIES
     ATARI Joystick Controller
```

2 GETTING STARTED

A simple way to organize this information is in numbered steps, in the order a user would set up his or her equipment and load your program. First indicate what accessories and/or cartridge must be in place before loading your program. Then describe how to load your program.

Standard ways to describe program loading

You load a program from diskette or from cassette. On each medium, the program loads one way if it's written in machine language and another way if it's written in BASIC. Here are examples for loading a program written in BASIC.

---

*Indicates trademark of Atari, Inc.

If you have the cassette version of the game:

1. Insert the game cassette in the program recorder, press REWIND, and then press PLAY.

2. Type CLOAD and press RETURN twice.

3. After the game loads into RAM, you'll see the READY prompt. Type RUN and press RETURN.

If you have the diskette version of the game:

1. Turn on your disk drive and insert the game diskette.

2. Power up your computer and turn on your video screen.

3. At the READY prompt, type RUN "D:filename" and press RETURN. The game will load into RAM.

The first display screen

Start your instructions for the game itself by describing the first display screen, which also tells the user what to expect if the program has loaded correctly. For example:

> You'll see a screen containing scoring information at the top; a four-layer avalanche; the text "AVALANCHE" and "BONUS PLAY FOR 300"; and four shields moving back and forth across the surface and absorbing falling rocks.

Game options and variations

If your program has variations chosen by pressing OPTION or SELECT or by responding to a prompt, describe each choice. An example of variations chosen by a function key is:

> Press SELECT to choose a one-player or two-player game. If only "#1" appears in the scoring area at the top of the screen, you've selected a one-player game. If "#1" and "#2" display, you have a two-player game. Players alternate turns.

An example of variations chosen by responding to a prompt is:

> You'll see the prompt DIFFICULTY LEVEL(0-9). Select the level you want, zero being the easiest and nine being the most challenging.

How to start

Finally, you want to be sure to tell the user how to start playing if your game doesn't start automatically! For example:

> After you've chosen your bonus level and number of players,

press START to begin playing. The avalanche will start falling in about 10 seconds, or you can press the red button on your paddle to start sooner.

3 PLAYING THE GAME

The playing field

If the game's playing field differs from the initial display, you'll want to explain any items on the screen that aren't self-explanatory. For example, a grid game using numbers as players can be described as follows:

> Your goal is first to locate an enemy unit and then to engage it in combat by moving your unit(s) into the hexagon it occupies. Each of your units displays as a number (0-9) in a hexagon. Enemy units visible to you display as numbers (also 0-9) in inverse video. Enemy units will suddenly appear whenever one of your units moves within three hexagons of them. You can move your units into any outlined hexagon. However, you can't pass through solid hexagons from any direction.

If sounds or color changes denote specific conditions players should be aware of--such as a flashing red to denote danger--you might want to describe these here as well. If the display screen changes under some circumstances, describe the screen variations and the conditions controlling them.

Game rules and moves/actions/commands

Organize this information to suit your game. A simply played game might need only one paragraph. That's all it took to explain this paddle game:

> By turning your paddle knob, maneuver your six shields to keep the rocks from hitting the ground. Each rock that gets by you counts as a miss (that is, a turn). As you absorb the rocks, your shields wear away--each one decreases in size and then disappears altogether, until you have only one small shield. At the same time, the rocks fall faster as you progress through the layers to the smaller rocks. If you're dexterous enough to absorb the entire mass, you face successive avalanches, but you start each new round with fewer shields.

On the other hand, if your game has many rules, or if it uses several keyboard commands, describe each. One approach is to explain rules and commands in the order a player typically needs to know the information while playing the game. Another approach is to group similar commands or rules. Your game might call for yet another approach. When you explain commands or actions, a "picture" of menus or data displays along with an explanation can be very helpful. An example is:

> You'll see the prompt INPUT COHORT # below the game grid. Enter one of two kinds of numbers: a number for one of your units or a number for an enemy unit. Entering a number for one of your

units (e.g., 6) results in a display like the following:

COHORT	EFF.ST.	DIS.ST.	MOVES
6	100	0	0 0 0 0

"COHORT" refers to your unit's number. "EFF.ST." (effective strength) and "DIS.ST" (disrupted strength) are explained below. Under "MOVES", you may replace as many of the zeros as you like with one-digit directional commands, which the program executes in real time.

Scoring

If your game keeps scores, explain the scoring system. If scores display on the screen, explain what each piece of information represents. For example:

Each player's current score displays in the second line. You earn one point per rock in the first layer, two points per rock in the second layer, and so on, up to six points per rock in the sixth layer. The same scoring applies to each avalanche you work your way through.

4 PROBLEMS

Use this section to warn players about potential problems and to explain what happens when players do something invalid. An example of a program operation warning is:

Be sure to wait for the "beep" sound indicating the computer has read your command before you press another key. Otherwise, the program might lose some of your commands.

An example of a common invalid move with program response and recommended recovery is:

If you try to enter a move for a unit that already displays four moves (i.e., no zeros), the program will bump you back to the "INPUT COHORT #" prompt. You must either wait for the program to execute at least one move for that unit before entering another move, or erase one or more moves for the unit and then enter new moves.

5 SUGGESTED STRATEGY AND HELPFUL HINTS (optional)

You might want to give novice players some hints or strategy, especially if your game is complicated. An example is:

If you're a beginner, a simple strategy to follow is: (1) fight only one attacker; (2) pursue him (code 8) at warp factor 1; (3) lock on all phasers (code 4); (4) continuously take his position and watch his voyage; (5) when he gets within 1100 M, fire all phasers (code 1) and keep on firing while he is in range; and

(6) when the enemy is out of range, scan him (code 9). After a few trial games, you'll want to become as efficient as the enemy at firing photon torpedoes. Finally, when you master launching anti-matter probes, you can designate more than one opponent.

6 RESTARTING OR REPLAYING THE GAME

Describe here how players can both interrupt a game in progress to start over and play another round of a game they've finished. Examples are:

You can interrupt the game at any time and start over by first pressing BREAK, then SHIFT CLEAR, and typing RUN.

When you finish a game, responding Y to the prompt LIKE TO PLAY AGAIN (Y OR N)? will set you up to battle the barbarians yet another time!

7 ADVANCED TECHNICAL INFORMATION (optional)

If you think some users might want to modify your game or design another game based on some techniques you've used, you might want to include information here so users can study what you've done. Assume a reader of this section has fairly extensive experience programming ATARI Personal Computer Systems. Consider the following kinds of information, either for your complete program or for its particularly complex parts:

1. A fully commented program listing (in the case of a program written in machine language, this would be the source code, which you should make as readable as possible).

2. A system-level diagram or description of your program's activities.

3. A description of the data structures used in your program, including diagrams of diskette/cassette file formats, pointer arrays, and data arrays.

4. A list of the most important variables used and their function.

5. A cross-reference listing of the program.

6. Any helpful hints, descriptions of unusual PEEKS or POKES, and other information that would help someone understand your program.

8 SUMMARY OF USEFUL INFORMATION (optional)

Players will be grateful to see a one-page summary of the information needed to play your game! When writing this, assume they've read through your instructions and need just a brief reminder of commands, rules, and so on.

NON-GAME SOFTWARE OUTLINE

Whether to include some of the following sections and the amount of detail to use depends to a large extent on your program's intended user--taking into account both age and background. A program designed for preteenagers with no programming experience needs much more explanation and simplified vocabulary than does a program intended for adults with extensive programming experience. Form a mental portrait of your program's typical user and try to address that user in your instructions. In general, the more explanation you include and the simpler you keep your vocabulary, the wider the potential market for your software. Many of the examples below come from a manual written for preteenage and teenage users. Notice how the explanations assume limited computer experience and avoid using many specialized computer terms. Consider whether your program's documentation faces these or similar restrictions.

1 INTRODUCTION

Overview

Use this section to orient the user. Describe generally your program's application, its most important features, and why and how one uses it. The idea is to give your readers a frame of reference for all the details that follow, but not to overwhelm, bewilder, or discourage them with too many details at this point. Other aspects you might want to mention briefly are your program's basic logic, any equations it uses, the kinds of activities users can do or the functions your program performs, whether your program is menu/prompt-driven, controller-driven, or command-driven, the kinds of output it produces, program limitations (if they're likely to discourage some users), and any other special considerations generally affecting your program. An example is:

> The NEWSPAPER ROUTE MANAGEMENT PROGRAM helps you manage a newspaper route by simplifying your record-keeping chores. It supports daily and Sunday routes of as many as 100 subscriptions. You use menu selections to enter, update, and delete customers' addresses and class of service to reflect subscription changes, and you keep track of your customers on a computer-displayed map. You use a joystick controller to position houses on the map and to update it when necessary. In addition, you can create and display several kinds of customer lists. If you have an ATARI 825(TM) Printer or an equivalent printer attached, you can print these lists, as well as payment collection lists and customer receipts.

Minimum RAM and accessories

A simple way to organize this information is to list it. The same section in the GAME SOFTWARE OUTLINE shows an example.

Special terms or notation

Note here any terms and symbolic notation you think most users will need explained, either because you use the term or notation in a special way or because the term or notation isn't known generally. Include terms and notation you use in either your program or your manual. Examples help to clarify notation descriptions. An example of a special terms explanation is:

The terms "START" and "ACTIVE" both refer to a customer who currently receives the paper, and the term "STOP" refers to a customer who is on vacation or has temporarily suspended service.

Special function keys

If your program uses function keys, such as the control or escape key or the directional arrow keys, note that information here, along with a brief description of their purpose.

References to related publications

If your program assumes a user is familiar with another ATARI publication, mention the document here. If your program depends on a user's having read some other, non-ATARI publication, cite the work as follows: author (last name first), full title, edition, volume, number of pages, publisher, and copyright year. If only some chapters or pages are important, mention these pages instead of the total number of pages for the work.

2 GETTING STARTED

Please see the discussion under GETTING STARTED in the GAME SOFTWARE OUTLINE for suggested ways to describe how users should load programs, depending on the software medium and the language.

The first display screen

Begin your detailed discussion of your program by describing the first display screen. At a minimum, tell the user what to expect, if the program loaded correctly. Better yet is to include an illustration of the first display screen. For example:

The first display screen looks like this:

```
FILE: PAPER30.LST
DATE: 2/22/81

13 MAP ELEMENTS
 6 STREET NAMES

40 HOUSES SAVED ON 02/28/81

     ACTIVE HOUSES . . . . . 38
     STOPS . . . . . .  2

TODAY'S DATE (MM/DD/YY) ?_
```

3 FUNCTIONS AND/OR COMMANDS AND/OR MENUS

This section is the heart of your user manual. These are the step-by-step instructions for using your program. Organize your information in a way that's logical for your program application. Some standard ways are by menu screens and by menu selections within each screen, or by each kind of program function, or by order of activity, if there's one usual order. For each unit, include the following information, as applicable:

1. Its name and purpose

2. The possible actions/options

3. The possible follow-up steps

Here is an example of instructions for a menu-driven program:

```
SELECTION 1:  DISPLAY MAP
```

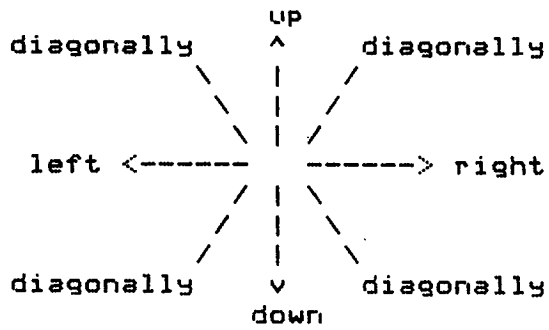
Use this selection to look at maps of the houses on your route. When you first select DISPLAY MAP, an empty street map displays in the graphics area of your video screen. The text window displays this Selection 1 Submenu:

```
SELECT
   1. ACTIVE HOUSES
   2. STOPS
   3. BOTH ?_
```

Enter 1 to display a map with house markers for only your active customers. Enter 2 to display one with markers for only your inactive customers. Enter 3 to display a map with markers for all your customers.

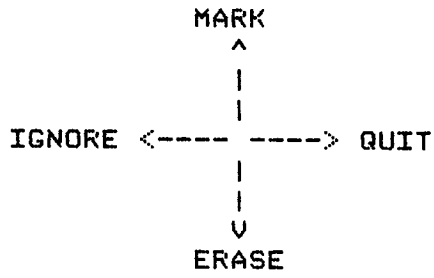
An example of explaining a program driven by controllers is:

As long as the map displays, all your commands are through the joystick controller. The map displays in the graphics area of your video screen. The small flashing cursor indicates your current map location. Hold the joystick with the red button to your upper left, toward the video screen. Position the cursor over an existing house marker or go to a new location by moving your joystick in these directions:



When the cursor is positioned where you want it, press the red button. Then move the joystick as shown below to add a new house marker (MARK), erase an existing house marker (ERASE), do

nothing with your current cursor position (IGNORE), or leave the map in its current state and return to the Customer Record display (QUIT):



Here is a sample instruction for a command-driven program. One way to arrange commands is alphabetically, which makes random searching fairly fast. Another way is by logical groups. In this example, the command's name, format, and one or more examples appear on separate lines, followed by a description of what it does and explanations of the examples:

JUMP TO NEXT PAGE

Format: .Jn

Example: .J10

The command ".Jn" promotes the text following it to the next page if fewer than "n" output lines remain on the current page. Use this command when you want a certain block of text to appear on the same page or when you want to leave room for tables, figures, or illustrations. For example, to avoid breaking up a figure occupying 10 output lines, enter ".J10" prior to this block of text. If at least 10 lines remain on the current output page, the figure will print on that page; otherwise, the program will leave the remainder of the current page blank and print the figure beginning at the top of the next page.

Refer to ATARI's BASIC REFERENCE MANUAL, pages 3 and 4, for the correct punctuation to use when a command format contains optional items or requires one of a choice of items. Notice the generous use of illustrations and examples in these sample instructions.

4 TROUBLESHOOTING

Describe what could go wrong in your program and how users can recover. Consider mistakes users can make on program loading, during data input, and at other stages of program use.

Error codes and/or messages

If your program contains error codes or error messages, list each one and explain the code or message, common causes for triggering the code or message, and how to recover from the error.

An example is:

```
CHECK PRINTER AND SELECT ONE
      1. TRY AGAIN
      2. RETURN TO MAIN MENU
    You've requested that customer receipts or a collection or
    house list be printed, but the program can't carry out your
    instruction. Make sure that your printer and interface are
    turned on, and then enter a 1 to print your data. If you've
    changed your mind about printing the information, enter 2
    to return to the Main Menu.
```

Program Bugs

If you know that your program contains some bugs, describe them here--the circumstances likely to trigger the bug, the bug itself, and a suggested recovery. An example is:

```
You might occasionally generate an error code if you're careless
with your responses to the prompts. For example, if you simply
press RETURN in response to the SELECT OPTION prompt, you'll see
ERROR -12 (Line Not Found). In these cases, you must rerun the
program.
```

Program operation limitations and warnings

Describe potential common problems users could encounter when using your program that aren't the result of a user error, but instead pertain to program operation ability or accuracy. Again, be sure to suggest how to recover or how to avoid the problem altogether. An example of a program limitation is:

```
Remember, your house numbers can be no longer than 4 characters
and your street names can be no longer than 15 characters.
```

An example of a program warning is:

```
If you lengthen your variable names, you could cause some new
lines to become longer than the logical line limit of the Screen
Editor. When this happens, the program will still run, but you
might not be able to edit those lines because the Screen Editor
will truncate them at the logical line limit.
```

5 ADVANCED TECHNICAL INFORMATION (optional)

Use this section for information that advanced users would need to modify your program or to use its more advanced features. The section describing ADVANCED TECHNICAL INFORMATION in the GAME SOFTWARE OUTLINE lists some of the kinds of information advanced users will find helpful. A sample of a program listing explanation is:

```
Lines 100-125 contain the timing loop that checks for legal
input and waits for the user to press the correct key before
continuing.
```

Lines 130-220 contain the strategy loop for the computer's move. Depending on which level you're playing, the computer checks from one to five moves ahead. Specifically, lines 150 and 160 check ahead and convert the results into numerical form, line 170 stores the value into an array, and lines 180-210 compare and evaluate the data.

Lines 300-350 keep track of the score and display it on the screen. Line 340 achieves the fireworks effect over the scoreboard and line 345 calls the sound subroutine.

An example of a variable list is:

```
A$(342)    ARRAY FOR START TOKENS
B$(6)      STRING FOR BLANK FILL
C$(1024)   ARRAY FOR INPUT FILE
D$(1024)   ARRAY FOR NAME TABLE
E(256)     ARRAY FOR VARIABLE ENDS
F          POINTER TO A$
I          TOKEN COUNTER
L          LINE LENGTH
  :              :
```

The numbers in parentheses are the dimension sizes.

An example of advanced programming information is:

You can alter the program to sort records greater than 185 bytes. To help you, both the source code for SUPERSORT--with filename SUPER2.ASM--and the source code for AUTORUN--with filename DVMOD.ASM--are on the disk. The variable TREC is a temporary storage variable residing on page six. Because it shares this area with other variables, it's only 185 bytes, but you can move TREC to a more open area to increase its byte size. The program accepts single byte record lengths even if you move TREC. In addition, you can modify the program to accept double byte records since the program uses record length (RLENG) only during additions. Because the addition is already double byte (to handle propagation), you need only change hi byte additions from ADC #0 to ADC RLENGHI. However, these modifications tend to slow down the program.

6 SAMPLE APPLICATION(S) OR SESSION(S) (optional)

Use this section to describe typical applications, both the most basic and some logical extensions. An example of this approach is the following description, which first shows an application using program default settings, and then describes extensions of the simplest case:

EXAMPLE 1. To renumber a program using the default settings.

```
ENTER INPUT DEVICE?D:TEST
```

```
ENTER OUTPUT DEVICE?D:TEST.REN

STARTING NO.?0
```

The program renumbers your designated file using the default parameters described earlier in this manual.

EXAMPLE 2. To renumber an entire program, setting your own values (the numbers are samples; use your own values).

```
ENTER INPUT DEVICE?D:TEST
ENTER OUTPUT DEVICE?D:TEST.REN

STARTING NO.?5
INCREMENT?5
FROM?10
TO?30000
```

EXAMPLE 3. To move a block of code (you want to move lines 100 through 190 to lines 1000 through 1090, setting your own values).

```
ENTER INPUT DEVICE?D:TEST
ENTER OUTPUT DEVICE?D:TEST.REN

STARTING NO.?1000
INCREMENT?10
FROM?100
TO?190
```

EXAMPLE 4. To switch two blocks of code.

1. To switch block A with block B, move block A to some unoccupied line numbers, using the procedure in Example 3.

2. Next, using the file resulting from step 1 and the same procedure, move block B into block A's former location.

3. Finally, using the file resulting from step 2 and the same procedure, move block A into block B's former location. Now save this final version.

If one block is bigger than the other, use a smaller increment value to accommodate the larger block.

7 COMMAND SUMMARY AND OTHER USEFUL INFORMATION (optional)

Users will be grateful to see a one- or two-page summary of the information needed to use your program. When writing this, assume they've read through your manual and need just a brief reminder of menu selections, commands, options, and so on.

1 PROCEDURES

1. When you first send your program to the ATARI Program Exchange, mark it as REVISION 1.0 in the first remark statement of your program. Then, if you submit modified versions of your program at later dates, change this number to reflect the kind of revision: increase the integer value by one for each major change (e.g., adding more menu selections); increase the decimal value by one for each minor change (e.g., a bug fix). A major change modifies your program's functional capabilities; a minor change doesn't.

2 SUGGESTIONS

How usable your program is depends to a great extent on how carefully you've designed its user interface. A programmer who has considered the end user is much more likely to produce an easy-to-use program. The following techniques can greatly increase the usability of a program. Consider whether these approaches will improve your software, as well.

1. Make your program flexible by offering alternatives to required accessories, when this is possible. For example, if you design a game playable either from the keyboard or with a joystick, chances are your potential market will be broader than if your game is playable only with a joystick. By the same token, if your software can run on either cassette or diskette, or if its output can print on any of ATARI's printers (or equivalent printers), you'll also increase your potential market size.

2. Let the user know at every step what the program expects him or her to do. Ways to show the user what to do include:

a) menus for three or more choices
b) yes/no prompted questions
c) one-word answers to prompted questions
d) HELP commands and/or short instruction frames
e) Instructions to press the OPTION, SELECT, or START key when these are active

Approaches to avoid using include:

a) undescribed choices or options
b) CNTRL or SHIFT user inputs, unless really necessary
c) one-letter commands, except for menu selections
d) computer jargon in displayed information or prompts

3. Try to trap and handle internally all system error messages so that a user doesn't have to cope with these while running your program.

4. Give the user plenty of feedback while he's running your program. Avoid hiding in a visual and audio "hole" longer than three seconds. In general, the longer the wait, the more feedback you should give a user. Some ways to tell a user that the program is computing include:

a) an echo of characters the user input
b) messages asking the user to stand by

c) task completion countdowns
d) flashing displays
e) displays of the estimated computation time

5. Make the user interface as easy and enjoyable to use as possible. Consider these techniques:

a) joystick or paddle controllers (assign these from left to right on the computer console)
b) rotating fields, changed by cursor position or option keys
c) meaningful mnemonics and keywords for commands
d) graphical presentations of information instead of strictly numerical ones
e) color-coded fields, messages, and screens
f) flashing cursor when current location might be difficult to see

6. Use various colors for different levels of your program and as a cue, such as red for an error screen. When creating colors for your program, keep these restrictions in mind:

a) they must be legible on black and white screens
b) they shouldn't be essential to your program's operation
c) they should minimize color bleeding ("artifacting") on color screens
d) their usage should be consistent throughout your program.

7. Use sounds to enliven your program, but be sure they convey the same meaning throughout. For example, use the same sound throughout to signify an input error. Sound is also useful for regaining the user's attention after a long delay or when an unexpected message displays. However, too many sounds can be distracting to users.

8. Code your program to ignore or reset lower case and ATARI invert modes in user responses to help inexperienced users who might get into these modes and be unaware of it.

9. For games, use the START key to begin the game and to restart it with current options intact if pressed during the game.

10. Use the OPTION key to select the number of players in a multi-player game and the SELECT key to choose game variations. The display screen should clearly indicate when users select a different game or option.

11. As space permits, use tree structures for menu-driven programs deeper than one level. Users should be able to return to the current menu level or return to the next higher level at any point.

12. In most cases, terminate input from users by a RETURN. If you choose instead to use a self-terminating single keystroke, design your program so that (1) if the user hits RETURN anyway, the program ignores it, and (2) if the user makes a mistake, returning to the upper level is easy.