ADVENTURE EDITOR SUPPLEMENT

The manual that accompanies the Adventure Editor is extremely difficult to understand. For this reason, I have decided to write a supplement which should help you with the manual.

Although Tex-Comp claims that a printer is not necessary, I found that it was. Programming an adventure game without this piece of equipment has to add countless hours to the task. Tex-Comp also claims that you do not need the Adventure Cartridge or Disk Program. I think it is an absolute must. The only way to debug a game is to play it. Incidentally, debugging a game is not difficult at all with a printed copy.

An adventure is written with lists, in column fashion, of nouns, verbs, objects, connections, references and situations. There is also a message list and then, of course, the Adventure Programming Language, or APL.

The best way to learn adventure programming is to print out an existing adventure game. I suggest printing out Pirate Island because it is not too involved and will be easier for you to understand. Copy Pirate Island onto another diskette because Scott Adams games must be converted before they can be printed. Then make certain you have an initialized diskette ready.

1. Turn on disk drive, monitor, console, then printer.
2. Insert Editor Assembler Cartridge.
3. Insert Adventure-Editor diskette in disk drive.
4. Press any key, then #2 for Editor-Assembler.
5. Program will load automatically.
6. Press #3 for 'load and run'.
7. Enter DSK1.CONVERT for file name.
8. When computer requests a second file name, just press enter.
9. Enter START for program name.

1. Remove Adventure Editor diskette.
2. Insert Pirate Island.
3. Enter I DSK1.PIRATE (the game will load.)
4. Enter G (the game will convert).
5. Enter !P PIO (or your printer name).
6. Enter !S (the statistics will print out).
7. Remove diskette and replace with initialized diskette.
8. Enter F DSK1.PIRATE (the converted game is saved).
9. Enter Q to quit (never Function Quit, this can partially destroy a game).

1. Remove converted program diskette.
2. Insert Adventure Editor diskette.
3. Press any key.
4. Press #2 for Editor Assembler.
5. Press #3 for 'load and run'.
6. Enter DSK1.EDITOR for file name.
7. When computer requests a second file name, just press enter.
8. Enter START for program name.
9. Remove Adventure Editor diskette.
10. Insert converted adventure.
11. Enter I DSK1.PIRATE (game will load if converted).
12. Enter !P PIO (or your printer name).

To print PIRATE, follow these simple directions:

The parameters for M,O,L,V, and N are taken from the Statistics listing printed with the Converter Program.

The parameters of R and of S are the same as of O.

The parameter of C is the same as of L.

E and G do not need parameters.

The parameter of A (APL) is the same as of V.

For example, to print the verbs of Pirate Island, type in !V,58. To print out the APL, you would type in !A,58. To print the messages, you would type !M,88.

To write your own game, you should first map out all the rooms in squares large enough so that you can make notations in each square of what will be visible during the game, as well as what you are hiding in these rooms. Above each square, jot down the name of each room, such as "bedroom", also number all these rooms starting with number 1, which is usually the starting room. The last room is the room where the player goes if he dies during the game. You should also note between squares, by means of arrows, in which directions you are allowing the player to move using "go north, south, east, west, up and down" commands. If the player must use commands like "go door" or "go coffin" to move to another room, make this notation between the squares. It is very important to make notes of your game plan before you start programming. It will save you a lot of time later on. Decide on the objects you want to hide as well as what the player has to do to find them. Once this work is done, you are ready to program your game. Don't be surprised if you get better ideas as you progress.

Load the Editor with the Editor Assembler using number 3 'Load and Run'. Type in EDITOR for the file name and just press enter for the second file name. The program name is START. When the screen changes, type in I DSK1.TEMPLATE. You are now ready to program your game. To save your work to diskette at any time, type in F DSK1.FILENAME.

You must decide if you will use three letter commands or four letter commands. I suggest that you use four letter commands for your first game. Even with four letters, you can run into trouble. For example, ball and balloon have identical first three as well as first four letters. You cannot use both of these words in a game, as the computer will not be able to differentiate between them.

I find the VERB list the easiest to start with. There are rules and regulations governing this list as well as all other lists. You cannot use VERB 0. VERB 1 must be GO; VERB 10 must be TAKE and VERB 18 must be DROP. The program allows you to use synonyms. In other words, you can allow the player to give the command RUN ROOM instead of GO ROOM. Synonyms must be preceded by an asterisk. Examine the verb list of Pirate Island. The program will let you input the words automatically up to VERB 19. After that, you are adding to the existing computer list and must use an & before the V. For example &V19. This is important to remember, because all lists differ as to the number you can input automatically. Now let's get started. Type in V1. You will see on the screen (1)= GO. This verb has already been programmed in. Press enter and type in V2. Verb *WALK has also been programmed in. You can change this verb if you so desire, because it is a synonym. I suggest you leave in all verbs that are already programmed into the game for your first endeavor. When you reach VERB 4, a ? will appear on the screen. In uppercase letters, type in a verb of your choice; remember to use only the first four letters of the word. Continue inputting the verbs you need for the game and skipping those that have already been programmed in.

The NOUN list is composed of everything you want the player to refer to or manipulate during the game. Remember that all commands are composed of a verb

and a noun; for example: EAT CANDY. You cannot use NOUN 0, nor NOUN 1 through NOUN 7, nor NOUN 17 which are already programmed in. The program allows you to use synonyms in this list also. All synonyms must be preceded by an asterisk.

The OBJECT list differs from the NOUN list in that this list is composed of everything that is visible (that the player can see) during the game, as well as of items that can be manipulated. Also objects are written in lowercase, capitalizing only the first letter of the first word. You can use OBJECT 0. OBJECT 9 has a special meaning and can be used only if you have a lit torch or lit candle or some such object in the game. If you are not planning to use any such object, type in XYDZ or some such word that will never be used by the player for OBJECT 9. Also all objects that are treasures must be preceded by an asterisk. Scott Adams always used an asterisk before and after the treasure and capitalized the treasure.

The LOCATIONS list is made up of all the rooms in the adventure. You cannot use LOCATION 0. The program will automatically print out "I'm in a" during the game. For such rooms, just type in the name of the room. If you do not want to use this phrase, precede the location by an asterisk; for example, *I'm in an attic or *I'm on the beach and type out the entire sentence. The asterisk will not be printed. As you have already planned all your rooms and assigned a number to them, this list is quite easy to do. You access this list by typing L1 for location 1.

The CONNECTION list allows the player to move from room to room. You cannot use CONNECTION 0. Type in C1. Six L0's appear on the screen. In order, from top to bottom, they are: North, South, East, West, Up and Down. Refer to your room plan. Your rooms are numbered and you also have arrows denoting in which directions the player is allowed to move. This is easy to program. For example: L0, L4, L3, L2, L5, L0 would mean: from location one, the player cannot go north; he can go south to location 4, east to location 3, west to location 2 and up to location 5; he cannot go down. This listing cannot be used for situations where, for example the player can go north by using a command such as GO DOOR. Such a situation will be programmed later.

The SITUATION list is also quite easy to program and is accessed by typing S1 for SITUATION 1. Refer to your OBJECT list and have it in hand. To print it out, type in !P PIO (or your printer name) and press enter; type in !O,40 (or as many objects as you have listed). The situation numbers correspond to the object numbers. If, for example, OBJECT 0 is Table and this table is visible (the player can see it) at the beginning of the game in LOCATION 4, you would type in S(0)=L4. If, for example, OBJECT 5 if Five dollars and you want the player to carry it at the beginning of the game, you would type in S(5)=Carry. For all other objects that are not visible at the beginning of the game, leave in the L0 that is already programmed in.

The REFERENCE list is a little more difficult. Type R1 for REFERENCE 1. Have your object list in hand. The reference numbers also correspond to the object numbers. This list is used for automatic take/drop actions. This means that if the object is visible or if the player is carrying it at any time during the game, he can take it or drop it. It will not work, however, if you are setting up conditions before or after he can take or drop it: for example, if the player must carry a stepladder before he can take a box on a shelf, or if you want to give the player a message when he takes or drops an object. The reference list is programmed by typing in the noun number that corresponds to the object. For example, if OBJECT 1 is a Brown paper bag and NOUN 20 is BAG and NOUN 21 is *PAPE (paper), you would type in R(1)=N20. You cannot use synonyms when programming. If you don't want the player to pick it up automatically, leave in the R(1)=N0 which has already been programmed in.

The ENTRY screen is where you enter what your game is about. Type in E and a model will appear on the screen. If you use Function Insert or Function Delete because you have made an error, the frame of dots will move. If you want to

keep this frame, it is better to use the Function Arrow keys. It's nice to add your name and address at the bottom, so that players can contact you if they encounter difficulties with your game. It's entirely up to you.

The GENERAL INFORMATION screen can be accessed by typing in G. Pages 11 and 60 of the manual describe this clearly.
1 = number of objects the player can carry.
2 = the starting location.
3 = the number of treasures.
4 = length of nouns and verbs.
5 = the starting value of the torch counter.
6 = how many steps for the torch counter.
7 = location where the treasures are to be dropped.

I find that the MESSAGE list is easier to compile while writing the Adventure Programming Language or APL. MESSAGE 0 can be used. I am in the habit of using MESSAGE 0 for I see nothing special and MESSAGE 2 for O.K. If you develop such habits, it will be easier for you when programming subsequent games. As I write the APL, I jot down the message or messages I need for each line and I number them consecutively. After I finish writing the APL, I enter the MESSAGE list into the computer.

ADVENTURE PROGRAMMING LANGUAGE

Adventure Programming Language is difficult to understand because it cannot be easily explained. With a little patience, you can master it.

All statements in APL must begin with either a ? or an !. These statements are listed on pages 31 and 32 of the manual. A ? is a conditional statement and an ! is a logical statement. Although this may help a little, it certainly doesn't make it clear. I found the manual very confusing with its explanation of these statements. Instead of thinking of the ? statements as "Don't I carry O7?" or "Do I carry O7?", I like to think of them in another way. This helped me and it may help you. I like to think of the ? statements as "IF" statements and the ! statements as "THEN" statements. We, therefore, have "If this is true", "Then this will happen".

Here is a sample of how I interpret the meaning of these statements (please refer to page 26 of the manual).

```
?Carry O7              If the player is carrying O7.
?Not Carry O7          If the player is not carrying O7.
?Visible O7            If O7 is visible.
?Not Visible O7        If O7 is not visible.
?Carry Visible O7      If the player is carrying O7 or if it's visible.
?Not Carry Visible O7  If the player is not carrying O7 or if it isn't visible.
?Carry                 If the player is carrying anything.
?Not Carry             If the player is not carrying anything.
?O7                    If O7 exists in the game.
?Not O7                If O7 doesn't exist in the game.
?Exchange O7           If O7 has been removed from it's starting location as
                          defined in the Situation list.
!Carry O7              Then the player carries O7 without check.
!'Carry O7             Then the player carries O7 with a check for a maximum.
!Visible O7            Then O7 becomes visible.
!Not O7                Then O7 disappears.
!O7 to L8              Then O7 is put at location 8.
!O7 to O8              Then O7 is put at the location where O8 is now.
!Exchange O7 and O8    Then exchange the current locations of O7 and O8.
```

All verbs, but not the synonyms, are used in APL and you are not allowed to use noun synonyms. Remember commands consist of a noun and a verb. Type in &A1,1. This means APL language; the first 1 denotes the verb #1 which is GO and the second 1 denotes that this is the first line for this verb. Actually a LINE consists of several lines. For example:

```
(1,1)  2?L8           (The 2 is noun 2 (door)). If the player is at location 8.
GO     !M2            Then display message 2 (O.K).
DOOR   !L6            Then the player goes to location 6.
       !RETURN        End of line.
```

The following line for the verb GO would be (1,2). Don't forget to use an & when adding elements to the list. All the lines listed for each verb are called a BLOCK.

There is also a probability block. This is the 0 (zero) block. For example:

```
(0,1)  100?L4         100% of the time - If the player is at location 4.
       !Visible O20   Then object 20 will be visible.
       !RETURN        End of line.

(0,2)  10?L3          10% of the time - If the player is at location 3.
       !M24           Then display message 24.
       !RETURN        End of line.
```

Here are a few examples of APL. I strongly suggest that you study Pirate Island until you understand exactly how this works. It won't take long and once you understand it, you'll marvel at how simple it is.

```
(1,1)  2?L3           If the player is at location 3.
GO     ?Visible O12   If O12 (a blue car) is visible.
CAR    !M15           Then M15 is displayed (The door is locked.)
       !RETURN        End of line.

(10,1) 4?Visible O2   If O2 (gun) is visible.
TAKE   ?Not L4        If the player is not at location 4.
GUN    !'Carry O2     Then the player carries the gun but with a check.
       !M2            Then display message 2 (O.K.)
       !RETURN        End of line.
```

As you remember, you gave the player a maximum amount of objects he could carry at one time. In !'Carry O2, the program will check to see how many objects the player is carrying. If he is carrying the maximum, the automatic message "I have too much to carry" will show up on the screen. This is done by means of the ' before the word Carry. There will be occasions when you will let the player carry a certain object no matter how many things he is carrying. To do this, use !Carry O2. As you can see, the ' has been omitted.

```
(18,1) 15?Carry O40   If the player is carrying O40 (a pill).
DROP   ?L22           If the player is at L22 (bridge over lake).
PILL   !Not O40       Then the pill disappears.
       !M72           Then display M72 (It fell into the lake).
       !RETURN        End of line
```

Flags are frequently used not to repeat an action. Supposing when the player looks into a box, he finds a gold ring. Every time he looks into the box, he will find another gold ring. You can stop this action in two ways. You can use the statement ?Not O20 which means "If O20 (gold ring) doesn't exist in the game yet" or you can use a flag. Which one you use depends on what you plan in the game for the gold ring. If the ring is going to exist for the entire game, then you can use ?Not O20. However, if you are going to make the ring disappear in some way or other, like for instance, if you allow the player to wear it, then you must use a flag. Let me give you two examples to clarify this.

```
(31,1) 23?Not O20     If O20 (the ring) doesn't exist.
LOOK   ?Visible O30   If the box is visible.
BOX    !Visible O20   Then the ring becomes visible.
       !M2            Then display M2 which is O.K.
       !RETURN        End of line.
(31,1) 23?Not F6      If Flag 6 is not set.
LOOK   ?Visible O30   If the box is visible.
BOX    !Visible O20   Then the ring becomes visible.
       !M2            Then display message 2 (O.K.).
       !F6            Then set Flag 6.
       !RETURN        End of line.

(22,1) 31?Carry O20   If the player is carrying the ring.
WEAR   !M2            Then display message 2 (O.K.).
RING   !Carry O35     Then carry O35 which is "I'm wearing a ring".
       !Not O20       Then the ring disappears.
       !RETURN        End of line.
```

As you can see in the last example, the ring disappeared. If you had used ?Not O20 the next time the player looked into the box, he would have found another ring. However, with the flag, it is set and the program can never execute the line again.

In my game, Cinderella, I had a reason for wanting the doorbell to start ringing when the player, who is Cinderella, was in her bedroom and I wanted the doorbell to keep ringing until she answered the door. If I used the following:

```
(0,1)  100?L3          If the player is at location 3.
100%   ?Not F14        If Flag 14 is not set.
PROBA- !M92            Then display M92 (the doorbell is ringing).
BILITY !RETURN         End of line
```

The doorbell only rang in her room with this statement. I had planned to make the doorbell stop ringing when she answered the door by setting the flag at that location. I found a solution by using two flags.

```
(0,1)  100?L3          If the player is at location 3.
100%   ?Not F14        If Flag 14 is not set.
PROBA- ?Not F20        If Flag 20 is not set.
BILITY !F20            Then set Flag 20.
       !RETURN         End of line.

(0,2)  100?Not F14     If Flag 14 is not set.
100%   ?F20            If Flag 20 is set.
PROBA- !M92            Then display M92 (the doorbell is ringing).
BILITY !RETURN         End of line.
```

When the player, Cinderella, is in her bedroom, flag 14 is not set but flag 20 is. The next statement reads that when flag 14 is not set and flag 20 is, then message 92 is displayed. I stopped the doorbell from ringing with the following statement.

```
(58,3) 31?L2           If the player is at location 2.
OPEN   ?Not F21        If Flag 21 is not set.
DOOR   !F14            Then set Flag 14.
       !Visible O86    Then the open door is visible.
       !Not O54        Then the door is not visible.
       !Visible O27    Then the Duke is visible.
       !F21            Then set Flag 21.
       !RETURN         End of line.
```

This statement works because the player cannot leave the house except through one door, the front door. Flag 21 is used so that the line will never be repeated again when the player opens the door. The Duke only makes one appearance in the game. I had to write another statement for when the player opened the front door again.

```
(58,4) 31?L2           If the player is at location 2.
OPEN   ?F21            If Flag 21 is set.
DOOR   !M2             Then display M2 (O.K.).
       !Visible O86    Then the open door is visible.
       !Not O54        Then the door disappears.
       !RETURN         End of line.
```

Keep track of the flags you use on a piece of paper, because if you use the same ones for different things, you can really play havoc with your game. You have to remember that Flag 15 and Flag 16 are reserved for the "torch" and cannot be used for anything else.

I found IF/ELSE statements very confusing. If you have trouble with them, remember that you can write adventure games without them. I wrote my first three games without using any IF/ELSE statements. They are not like IF/ELSE statements in Basic. I finally figured them out by studying example after example. When I first tried them, they just didn't do what I had planned. It was only through trial and error that I finally understood them.

Here is an example of an IF/ELSE statement that doesn't work.

```
(26,1) 75?Carry O30   If the player is carrying O30 (matches).
LIGHT  !Exchange #1   Then get the stored counter #1.
MATCH  !IF
       ?>#0           If the counter is greater than #0.
       !-#1           Then decrease the counter by 1.
       !M111          Then display M111 (the match went out).
       !Exchange #1   Then save the counter.
       !RETURN        End of line
       !ELSE
       !M81           Then display M81 (I have no matches left).
       !Not O30       Then the matches disappear.
       !Exchange #1   Then save the counter.
       !RETURN        End of line
```

What happens in this example is that when the player lights a match, the counter is decreased by 1 and the statement ends with the return. It never branches out to the ELSE statement. Omitting the Exchange #1 and the RETURN listed under the IF doesn't help either because then the If statement is executed if the player is carrying the matches and the ELSE is also executed. The statements will keep executing until a fault (a line it cannot execute is reached) at which time it will branch to the ELSE statement. The proper way to write this line is:

```
(26,1) 75?Carry O30   If the player is carrying matches.
LIGHT  !Exchange #1   Get the counter.
MATCH  !IF
       ?>#0           If the counter is greater than #0.
       !-#1           Then decrease the counter by 1.
       !M111          Then display M111 (the match went out).
       !ELSE
       IF
       ?#0            If the counter is equal to 0.
       !M81           Then display M81 (I have no matches left).
       !Not O30       Then the matches disappear.
       ELSE
       !Exchange #1   Save the counter.
       !RETURN        End of the line.
```

An important thing to remember about IF/ELSE statements is that when a fault is encountered, it will branch to the ELSE statement even if this statement is before the fault.

Here is another example:

```
(36,1) 23!IF
BUILD  ?Carry O24     If the player is carrying O24 (hammer).
TABLE  ?Carry O25     If the player is carrying O25 (lumber).
       ?Carry O26     If the player is carrying O26 (nails).
       !M2            Then display message 2 (O.K.).
       !Not O24       Then O24 disappears.
       !Not O25       Then O25 disappears.
       !Not O26       Then O26 disappears.
       !Visible O41   Then the table is visible.
       !RETURN        End of line.
       !ELSE
       !M25           Then display M25 (I don't have everything I need).
       !RETURN        End of line.
```

This line works because if any one necessary object is not carried, a fault is encountered and the program branches to the ELSE statement.

Here is another example:

```
(31,1) 22?Visible O30 If Mary is visible.
LOOK   !M20           Then display message 20.
MARY   !IF
       ?L10           If the player is at location 10.
       !M21           Then display message 21.
       !ELSE
       !IF
       ?L11           If the player is at location 11.
       !M22           Then display message 22.
       !ELSE
       !M23           Then display message 23.
       !RETURN        End of line.
```

In this example, if Mary is visible, message 20 and message 23 will always be displayed when the player gives the command LOOK MARY. However, if she is visible at location 10, messages 20, 21 and 23 will be displayed and at location 11, messages 20, 22, and 23 will be displayed. Remember that every statement is executed unless a fault is encountered.

We have touched on counters in the IF/ELSE examples. They are a bit confusing. Counters are initialized first. The manual gives a very good example at the bottom of page 38. This example sets counter #1 at 10 turns and counter #2 at 40 turns. Here is another example of initializing a counter.

```
(0,1)  100?Not F0     If Flag 0 is not set.
100%   !F0           Then set Flag 0.
PROBA- !#5           Then set counter at 5 turns.
BILITY !Exchange #1  Then save counter number 1.
       !RETURN       End of line.
```

In this example, counter #1 represents matches and is set at 5 turns and every time you use one, you will have one less. Refer to previous examples under IF/ELSE statements. Notice how you must always get the counter and then save it.

```
(31,6) 75?Carry O30   If the player is carrying O30 (matches).
LOOK   !Exchange #1   Then get counter #1.
MATCHES!M68           Then display message 68 (I have).
       !Display #     Then display counter.
       !M69           Then display message 69 (left).
       !Exchange #1   Then save the counter.
       !RETURN        End of line
```

In this example, the player looks at the matches to see how many he has left. The line is read and the message "I have (number of matches) left" appears on the screen.

Following are a few more tips that may help you.

At the end of a block, you can use the following:

```
(9,30) 0!M0           Display M0 (I see nothing special).
LOOK   !RETURN        End of line.
ANY-
THING
```

This line must be the last one in the block, otherwise the player would receive message 0 for everything he looked at. This line will be executed as long as the noun chosen by the player is listed in the noun list.

The !WAIT statement gives effect. It will create a 1.2 second pause before executing the next statement. You can use more than one for longer pauses.

A !SCROLL statement is also very effective. Sometimes it's better not to have consecutive messages in a line read in paragraph form. A scroll will start a new line.

Remember that the first line that can be executed will be executed and subsequent lines will be ignored. Therefore, you must have some kind of order in your APL. For example:

```
(31,2) 24?Visible O12 If O12 (hat) is visible.
LOOK   !M7            Then display message 7.
HAT    !RETURN        End of line.

(31,3) 24?Visible O12 If O12 (hat) is visible.
LOOK   ?L3            If the player is at location 3.
HAT    !M8            Then display message 8.
       !RETURN        End of line.
```

In this example, line 3 will never be executed because line 2 will always be executed whenever the player gives the command LOOK HAT. However, if line 3 is placed before line 2, it will be read first and if the player is not at location 3, the program will jump to the next line.

Also, if you want such commands as DRIVE EAST, or JUMP DOWN to be executed, you must not make the verbs drive and jump synonyms to Verb #1, which is GO. Go north, south, east, west, up and down are controlled by the program. A separate entry must be made for these verbs in the verb list.

These few pages were not meant to replace the manual. My aim was to clarify some areas of adventure programming which I found vague in the manual such as flags. I spent a day and a half trying to figure out how to use them. Once I figured out how, I couldn't believe how simple it was. My first game is being distributed by a company as a bonus gift when a certain purchase is made. Because of this, I have been receiving a lot of mail as well as telephone calls concerning my game and adventure programming. I wasn't the only one who experienced a lot of difficulty with the manual. This is what lead me to write these few pages. I have shared with you the sequence I use to write my games and how I do it. I hope this will help you. After you have written your first game, I'm certain you'll agree with me that the most difficult part of writing an adventure is coming up with good ideas.

LUCILLE F. ROCK
463 SOUTH MAIN STREET
WOONSOCKET, R.I. 02895