TEX★WARE™

The TI-99/4A ADVENTURE EDITOR

WRITE AND EDIT YOUR OWN ADVENTURE GAMES FOR THE TI ADVENTURE MODULE!

This software package allows the user to accomplish the following:

1. Any existing adventure for use with the TI Adventure Module may be edited, altered, listed, or copied from either tape or disk and transferred to either tape or disk. (your choice).
2. New Adventure games for use with the TI Adventure Module may be created easily using a 'template' game as a start up step. These games can be played on a basic TI/99/4A since they use the full power and capabilities of the TI Adventure Module.
3. Either Editor Assembler or Mini Memory command modules can be used with this program with either cassette (M/M Version) or disk (E/A Version). The identical features are offered with each version.
4. Includes a full screen editor and an easy to use Adventure Programming Language (APL).

PUBLISHED BY

TEX★COMP

America's Number One TI computer retailer

P.O. BOX 33084 • GRANADA HILLS, CA 91344 • (818) 366-6631

Table of Contents

ADVENTURE EDITOR

1 INTRODUCTION AND OVERVIEW

1.1 Purpose of the program package

With the program package ADVENTURE EDITOR you own a tool for working on Adventure games for the 'Adventure' command module pertaining to the Texas Instruments TI99/4A Home Computer. You may wish to - and are able - to investigate in and/or alter all existing games as well as to write your own ones.

This offers all opportunities a 'TI-adventurer' has ever dreamed of.

1.2 What you need - equipment

You received the package according to your order in one of two possible versions: either for the Mini Memory or the Editor Assembler command module. Besides the console (TI99/4A; the old one -TI99/4- does't work) you will need the respective command module and - for Editor/Assembler only - memory expansion and Disk drive with controller or - for Mini Memory only - a suitable Cassette recorder.

The Adventure command module is NOT necessary but strongly recommended for testing your new games.

A printer with its TI-peripheral is recommended as well, but not a must. You can print out a full documentation of any game if you own one.

1.3 What you need - knowledge

You should already have played some adventure games with the TI99 before starting to develop your own ones in order to understand how it will work.

Some practical knowledge of any programming language (BASIC or the like) will help to understand the concepts and structures you will have to deal with in adventure programming. You will have to do a good bit of thinking in abstractions.

Knowledge of assembly language or GPL is NOT necessary!

You can ONLY write adventure games in english, since the command module will produce its own (english) messages which you cannot get rid of.

1.4 What you get

Depending on the version (MM=Mini Memory or EA=Editor/Assembler) you have received a Cassette in the first case or a diskette (5 1/4' SS/SD) in the second case. The software is functionally the same in both cases. Since the MM will be used up the last byte of its 4kB RAM, there will be a lot of room left if you run the EA-version.

You received two programs and one file in both versions:

```
- a. 'ADVENTURE EDITOR'
      for Editing/Printing games
         MM : Cassette side A Program No. 1
         EA : 'DSK1.EDITOR'
- b. 'ADVENTURE CONVERSION'
      for converting existing games
         MM : Cassette Side A Program No. 2
         EA : 'DSK1.CONVERT'
- c. 'ADVENTURE TEMPLATE'
      as a game template for your own ideas
         MM : Cassette Side B File No. 1
         EA : 'DSK1.TEMPLATE'
```

1.5 What you can do

The programs of this package will allow you to :
- prepare existing games for the ADVENTURE EDITOR using the ADVENTURE CONVERSION program
- in both old or new games (converted, if old) show, edit or print all texts and lists which make out the game.
- show, edit or print the logic of the game using a new mnemonic language (A.P.L.) (both with ADVENTURE EDITOR)
- load or save games from/to either Cassette or Diskette as desired.

The program(s) will in all cases take care of the internal data structures like pointers and tables. You may create a game freely without bothering about the internal stuff, entering text and structures as you like.

1.6 How the guide is conceived

There are two main purposes for this reference guide:
- You will get all information needed to understand the structure of an adventure game in order to create new ones which will react properly to the players commands.
- You will be advised in using the programs.

This results in a detailed description of adventure games with a separate chapter on all texts and lists you have to prepare.

A second chapter on the 'programming language' A.P.L. (which means Adventure Programming Language; don't confuse with APL!) will describe in detail this interesting tool. You will be given some examples in order to better understand its possibilities and to give you some ideas to start with.

You will possibly not understand everything in these chapters if you first read them; it is suggested that you skip those lines and reread them later on when you will be acquainted with the programs. Most probably the questions you will ask later on will find their answer here.

The next chapter will introduce you to the EDITOR program. You will have to learn the mnemonics used for command input when using the program. When you have had some training you will probably skip these lines.

This is the same for the chapter on using the CONVERT-program. One difference may be that you will use this program very rarely, namely when looking at your old games (Pirates Adventure or the like) which may be a good training to understand 'how to do it'. You will have to convert them before loading them with the EDITOR. For reasons of copyright these games are not discussed in the reference guide.

A short chapter on the contents of the game TEMPLATE will prepare you to start with your own games.

Some more remarks - technical and personal - of the author in the last chapter of the reference guide are intended to make you realize the problems and possibilities of adventure programming.

As you see this is a whole lot of information. Don't worry - you will get familiar with it, and then it looks like child's play ... and it is a play after all, isn't it? So I wish you lots of fun, and remember: You can always SAY YOHO ...

****

2 STRUCTURE OF AN ADVENTURE GAME

An Adventure game seems to be a pseudo-intelligent structure when it is used by a player. This is accomplished by means of the Adventure command module and seems very different to the game programmer.

Internally a game consists of several lists and reference data which - interpreted and interconnected by a special 'program' - give the effect of an intelligent entity.

This decription covers all relevant aspects of these lists and data in order to enable the game programmer to achieve the desired results. The very complex internal data structures (mainly pointer tables) will not be discussed since the ADVENTURE EDITOR programs will manage all this.

2.1 How to use text data

Text data make out the largest part of such a kind of adventure, as everybody who played those games can imagine. Therefore it is advisable to plan your texts in such a way that the whole plot of the game is consistant, and to find interesting and surprising solutions for any situation.

Your text data have to be entered as lists (messages, objects, locations, verbs and nouns), i.e. texts are ordered and numbered. Each numbering must begin with '0' (zero) and goes through to the maximum number of each list. The number '0' (zero) must exist, although it cannot be used for texts in some cases. The numbering must be continous.

The reference to any of these texts is achieved by giving the appropriate number preceded by the first letter of the list name (as a capital). For instance M2 means message #2 - whatever this text may be. The player of a game will of course never get in touch with these numbers; their only purpose is in programming the game.

Some numbers have predefined meanings in some cases which will be mentioned from case to case. The first element (number 0) has to be used with special care, as will be mentioned as well.

2.1.1 Messages

There is one list for all messages appearing throughout the game process. These can only be called from A.P.L. programs and will never appear automatically. - Don't confuse with the built-in messages given in several situations ('The light has run out' or the like) which can NOT be used from A.P.L.

Messages always appear in the lower part of the screen and are meant to describe the game or to give hints to the player.

Number 0 (M0) : can be used
Special meanings : none

2.1.2 Objects

There is one list of texts (names) for all the objects you can see and/or manipulate during the game. These objects are managed by automatic actions ('Visible...' , 'I'm carrying...' 'GET' , 'DROP' etc.) and by your program as well.

The object names will appear in the upper or lower part of the screen depending on the actions you take. The command module will add the 'I'm carrying...' or 'Visible...' headers.

Besides its number and the pertaining text any object always has a location where it can found ('Situation'). A 'Reference' to the nouns is given if it can be 'TAKEn or DROPped' automatically.

Treasures - if present - must have their name preceded by an asterisk ('*'). Everything else is optional (Capitals, asterisk at the end etc.). The number of treasures that equals '100%' MUST be given correctly within the 'general informations'.

Number 0 (O0) : can be used
Special Meanings : O9
There is a special meaning for object #9 (O9) : It must be used as a kind of 'lit torch' or the like. The reason for this is that there is a counter for this object which will be decremented every 'round' when this object (O9) exists somewhere in the game. When this counter ('Torch counter') will reach zero, the well known messages 'The light runs out in ... turns' or 'The light has run out' will appear automatically and cannot be suppressed. Therefore if you have to use this object for some other purpose be aware of this effect and initialize the 'torch counter' with a very large number (see 'general information').

2.1.3 Locations

All locations you can reach inside the game have their name in this list. Locations can be treated 'automatically' (by giving directions which must correspond to a list of 'Connections') or can be changed by A.P.L. program statemements if you wish to.

A location within the adventure game is identified by its number and appears to player as a short text. There is one 'actual location' - a number defining a location where the 'player puppet' is, and each object has got its 'location' defining where it can be found. There is one 'Starting location' and one 'Score location' - both defined in the 'general informations'

The location names will always appear at the top of the screen ('I'm in a...'). If you don't want this automatic header to appear begin the location name with an asterisk ('*'). In that case you must give the complete text ('*I'm at the beach'). The asterisk will not be printed!

Number 0 (L0) : DON'T USE
Special meanings : last location in the list
This location - the last element of the list independent of its actual number! - has the meaning of 'End of the game': Whenever you reach this place the screen colour switches to white/red. The game is not over however: Have you ever tried SAY YOHO in Pirate's Adventure when you're 'in Never Never Land' ?? This can be accomplished by taking actions in A.P.L. since only the A.P.L. command '!Over' will really end the game.

2.2 Vocabulary

You can only give your commands during the game using two word english sentences. Let's define the first word as a 'verb' and the second one - if it exists - as a 'noun'. This results in WITH being a verb (WITH HAND) or NICE being a noun (LOOK NICE).

Whenever you give a command the command module searches the 'verb' list for the first word. If it doesn't find it (and it isn't a direction or I),it gives the message 'I don't know how to ... something', otherwise it stores the number. The same is the done with the second word and the 'nouns' list, the message being of course altered to 'I don't know what a ... is'. If there is no second word (LOOK,LISTen etc.) actions are taken as normal. You can however construct A.P.L. program lines which will react to this as well.

The number of characters for abbreviations (see 'general information') will result in lists of verbs and nouns of that length. It's up to you to make abbreviations unique. If words are shorter than this length (GO etc.) enter only the full word. In all other cases enter the abbreviation (DRO or DROP depending on 3 or 4 characters for abbreviations!)

For equivalent words - i.e. words which will behave exactly like others - enter these equivalents immediately following the 'main' word and place an asterisk '*' at the beginning. This character must of course NOT be entered as a command; it just serves the purpose of designating synonyms. In programming only refer to the 'main' word; the duplicates are invisible to any reference.

2.2.1 Verbs

Verbs - the first words of the two-word commands - are used to select one special 'block' of A.P.L. lines, leading to the desired effects if you construct these lines accordingly. The number of the verb in its list is the number of the block selected. (Don't use synonyms as block numbers! They will never be executed.)

```
Number 0 (NO)      : DONT'T USE
Special meanings : V1  = GO   or the like
                   V10 = TAKE or the like
                   V18 = DROP or the like
```

2.2.2 Nouns

Nouns - the second words of the two-word commands - are used to select one or more lines within the block of A.P.L. program lines to be executed. You can take the actions you want to in these lines, depending on the command. It is possible to write lines for any noun (especially useful if no noun was given).

```
Number 0 (NO)      : DONT'T USE
Special meanings : N1  = NORT (North; for GO NORT = N )
                   N2  = SOUT (South; for GO SOUT = S )
                   N3  = EAST (East ; for GO EAST = E )
                   N4  = WEST (West ; for GO WEST = W )
                   N5  = UP   (Up   ; for GO UP   = U )
                   N6  = DOWN (Down ; for GO DOWN = D )
                   N17 = INVE
                         (Inventory; for TAKE INVE = I )
```

Since nouns often refer to objects (mentioned above) there is a list of references (for every object: which noun, if one ?). This list is only needed for 'automatic' TAKE/DROP actions.

2.3 Cross references

Some types of lists mentioned earlier are needed to give all necessary information for the game.

2.3.1 References

As mentioned above a reference has to be given in order to accomplish 'automatic' TAKE/DROP actions. So if you give a noun the module looks up in this list what objects you could have meant and takes the actions needed. It is possible to access several objects with the same noun: The first object that fits will be used.

It should be noted that all actions you take within the A.P.L can NOT make use of this table. You have to check for the existence of any object you want to refer to.

So if for example you give the command TAKE (N15) or DROP (N15) the command module will first check what objects can be referred to by N15. If there is (at least) one of them, it checks whether or not the player is carrying it or if it is visible, respectively. If he hasn't got too much to carry (if TAKE ... is entered) the appropriate actions are taken. So in this process the following 'automatic' messages could occur:

'I'm not carrying it' (DROP)
'I already have it' (TAKE)
'I don't see it here' (TAKE)
'I've got too much to carry' (TAKE)
'It's beyond my power to do that'
(No reference found !!)

Note: Special care has to be taken in your A.P.L. program in order to let this work properly!

2.3.2 Situation

As mentioned above there is a location for every object at any time. Since this list will vary very much during the game it is simply called 'situation'. There are by the way two lists which are identical at the beginning of the play: one of them will be altered during the game, whereas the other one will be used to initialize the situation after a game restart or at the beginning. It also serves for special A.P.L. statements checking whether or not an object has moved from its starting position. Note that you can enter only one list - the starting situation.

In the 'situation' table L0 has the special meaning of 'doesn't (yet) exist'. Furthermore an object can be 'Carried' which is also a valid location. All locations given are of course the starting locations for any object which will be altered during the game.

2.3.3 Connections

You have to define in which way the locations are connected by means of 'obvious exits'. If this is done you can 'GO' there by entering the direction.

This list consists of six location numbers in the following order: North(NORT) - South(SOUT) - East(EAST) - West(WEST) - Up(UP) - Down(DOWN). If there is no connection in one direction, enter L0 at that place. Don't confuse the player by giving connections which won't return him to his starting place when using the reverse directions unless you want to reach special effects (a maze for instance).

Example : Location #1 (L1) : L0 L2 L4 L0 L5 L6 means : From L1 SOUTH is L2 , EAST is L4 , UP is L5 , DOWN is L6

2.3.4 General Information

Some information which doesn't make up a list is gathered here. They are preceded by the neutral '#'-sign to indicate numerical values.

- Number of objects you can carry.
  This value is checked for 'automatic' TAKE/DROP actions and can be checked (optional) when picking up objects in your A.P.L. program.
  Error message :
  'I've got too much to carry.'
- Starting location. Location number where the player will start the game.
- Number of treasures that make out '100%'.
  Remember that the game is over for a Score 100%! If you enter '0' (zero) here, don't use treasures! (and vice versa)
- Length of abbreviations (3 or 4 in the most cases; max. value is 9!)
- 'Torch counter' starting value.
  This is a value of 2 byte length, max. 32767.
  For calculating the total multiply the first value (byte) by 256 and add the second one.
  If you don't use a lamp at all, enter 32767
  (#127 #255=127×256+255=32767)
- Score Location. Location number to gather your treasures (Remember? Objects beginning with '*').
  When the statement !'Score is executed, the number of treasures in this location is checked.

2.4 Internals

Some information within the game which is 'save'd and 'load'ed is not obvious to the player. This information is meaningful only to the A.P.L. programmer if he decides to use it. It cannot be 'printed' or 'listed' since it exists only as an abstract idea. This background information makes the game seem 'clever' or 'witty' however, increasing the joy of the player.

2.4.1 Flags (F)

You can use Bit-flags (set/reset or query) to indicate certain game states or to initialize certain actions. Flags are always initialized with '0' (reset) at the beginning or restart of a game.

You can use 32 flags (F0 through F31), with predefined meanings for F15 and F16.

F15 has the meaning of '(day-)light' (reset) or 'night/dark' (set) which will result in the screen colours black/green for 'light' or white/blue for 'dark'. If it's 'dark' (F15 Set) the command module will no longer display anything visible. This will be overridden by a visible or carried O9 if the 'torch counter' has not yet reached zero, which means that a 'lit torch' will illuminate the surrounding. This must be considered in the case that O9 is NOT used with that special meaning!

F16 is set by the command module if during the game the 'Lit torch' (which is O9) has run out of fuel (torch counter is zero). You can then take the appropriate actions. Have in mind the effects described above!

2.4.2 Counter (#)

You can use counters in the game to count anything you want to. All calculations are done with a 'current counter' which can be manipulated (display, add, subtract, set, compare) within the A.P.L. . The value of any of the counters may be from -32768 to +32767; the calculation constants within the A.P.L. may however only be positive values from 0 to 255! This may result in some problems but cannot be altered.

You can also exchange the 'current counter' with one of the six 'stored counters' #0 to #5. Note that these cannot be initialized at startup; you will have to initialize them yourself with an appropriate A.P.L. construction if this is necessary. It is however not possible to compare the value of two counters.

Don't confuse these counters with the 'torch counter' mentioned before which is under total control of the command module!

2.4.3 Locations stored

You can store up to seven Locations L0 through L6 which will be 'save'd and 'load'ed with the game. You can exchange them with the 'current location' any time you like from A.P.L. . The purpose of this is not so obvious but can be used by clever programmers to achieve special results.

Don't confus them with the locations in the list! L0 to L6 can mean one of the stored locations OR one of the list elements depending on the A.P.L. statement.

2.5 Entry screen

The entry screen appearing at initial startup is not of a big importance to the game. It serves however as an introduction to the plot and for copyright notes, so that it can be entered here. Note that the last two lines are reserved for the Editor Copyright and cannot be altered.

2.6 Adventure Programming Language - A.P.L.

A detailed description of this interesting 'programming language' will follow in the next chapter of the reference guide. Here are some general remarks:

The logic of the game is written and stored in intermediate language which has to interpreted during the game. This simple language designed for adventure programming is part of the adventure command module. To make it readable short mnemonics were developed which are very close to the actual meaning. Note that these mnemonics have not been found in the command module but are part of the EDITOR-Program which will convert it to the tokens for the adventure game.

2.6.1 Language structure

The main elements of this language are:
A) Conditional Statements
B) Actions (consequencs)
C) Logical Statements
   (IF,ELSE,RETURN)

The program you write is divided in 'blocks', each block consisting of 'lines'. This means that a block is first selected, and then all lines are checked for 'validity' and executed, if valid. This continues until the 'block end' condition is fulfilled. Only one block of lines can be executed at a time.

Each block is selected by a 'verb' (exception block 0). Each line owns a 'validity' header indicating the conditions for an execution of this line. If valid the statements of this line are executed one by one until a !RETURN is encountered or one one of the block ending statements (!Over, !'Score 100%) is executed.

The validity header is the number of noun or zero in all blocks but block zero, and a probability from 0 to 100 in block zero.

Within a line you can use nested IF (THEN)/ELSE structures to a level of six. Be cautious in using IF without a condition!

What you can NOT do in A.P.L. is - compared to BASIC -
- Jumps
- Subroutines
- REMs
- DATA Statements
- Variables
- and so on

2.6.2 continuous actions

Block #0 is executed automatically each time you enter a command after checking your command input. So in this program block you will place all actions that have to be executed without regard to the actual command.

The 'validity'-header at the beginning of each line means a probability from 0 to 100 (%), so that lines can be executed not every step. The internal RND function is used for this purpose.

Within block 0 all lines are executed (with their respective probability); this means that the only 'block ending conditions' are the execution of the statements '!Over' or '!'Score' with a result of 100%.

2.6.3 command dependant actions

All other blocks are executed depending on the number in the 'verb'-list. Remember that only 'main' verbs, no synonyms can have A.P.L. blocks! The EDITOR does not allow to build a block for such verbs, but you can trick him out by altering the verb list later on.

The 'validity' header for each line is a 'noun' number (without N!) so that a line is executed for a well defined combination of verb and noun, that is for one 'command'. You can have several lines for one command, but only one can be totally executed.

The 'block ending condition' for these blocks is the 'successful completion' of a line. This can be:
- a !RETURN
- a !'Carry which is NOT executed (!!)
- a Statement '!Over' or '!'Score' (100%)

A header of zero indicates a line which will be executed regardless of the noun given (especially no noun at all) so that you can use verbs intransitively.

If you combine noun-numbers and the number zero, be aware of the effect that any 'succefully completed' line ends the block! Therefore you will place those lines at the end of the block in most cases.

2.7 Actions taken by the command module

There is some 'intelligence' built in the command module which exceeds interpreting you A.P.L. program. This will result in 'automatic' actions in most cases, together with the appropriate messages.

You must be aware of this when writing your program. A 'sequence schedule' will be given at the end of this chapter which will explain in detail all possible combinations. Use that schedule whenever in doubt about the actions!

2.7.1 Directions

Directions can be entered as a single letter command or as GO (Direction). The command module will accept that and take the appropriate actions. The messages 'Please give me a direction also' , 'It's dangerous to move in the dark' or 'I fell down and broke my neck' may result from this.

The list of 'connections' serves as a 'map' for these actions.

If 'GO' is used with some other noun however you must care for all necessary actions in A.P.L. Block 1 yourself.

2.7.2 automatic TAKE/DROP

The automatic TAKE/DROP actions have been discussed earlier. Since these actions are taken AFTER completing the A.P.L. blocks some special effects have to be considered.

The condition for the automatic TAKE/DROP action is that this very action has not been tried/and or completed in the A.P.L. block. If any one line (for that noun) has only been begun no further actions will be taken! In that case - if not yet successful - the built-in message 'I can't do that yet' will appear and nothing will happen.

2.7.3 Light

As mentioned above you can determine whether it is 'light' or 'dark' by resetting/setting F15 (Flag 15). The consequences of this are obvious. Keep in mind however that this Flag is dominated by the 'lit torch' 09: If this object is carried or visible, it will be light even though F15 is set, in case that the 'torch counter' has not yet reached zero.

Furthermore a 'lit torch' or the like is managed by the command module and will result in the messages 'The light runs out in ... turns' or 'The light has run out', if necessary. You cannot suppress these messages, but you can ask if the lamp is still lit (F16 Set). Otherwise it is up to you to take the actions required (let the 'torch' vanish or some other effect you have in mind).

2.7.4 The End of the game

When the player is 'dead' you can use the built-in feature of the 'white/red' screen by putting him in the last location. The command module will then switch the colors. This can be achieved by executing the command '!I'm Dead' in A.P.L.

This is however not the End of the game! The play can go on as normal, if you decide to let him out of 'limbo' in some way.

The real and immediate end of the game is achieved by executing the A.P.L. statement '!Over' . This will result in the message 'want to play this adventure again?' with all possible consequences.

The screen color and other actions depend on two flags that you can set (but not reset) from A.P.L :
- Flag Dead : will switch to white/red.
- Flag Well done : will switch to black/yellow.

Note that the same effects as with '!Over' can be accomplished when the statement '!'Score' is executed with a result of 100% !

2.7.5 Game sequence schedule

A game is a cyclic process which is passed through every time a command is entered. A grafical description helps best to understand how it works.

Possible messages are indicated by a (x) and mean:
(a) 'I don't know how to .... something'
(b) 'I don't know what a .... is'
(c) 'The light runs out in ... turns'
(d) 'The light has run out'
(e) 'Please give me a direction also'
(f) 'I can't go in that direction'
(g) 'It's dangerous to move in the dark'
(h) 'I fell down and broke my neck'
(i) 'I can't do that yet'
(j) 'It's beyond my power to do that'
(k) 'I'm not carrying it'
(l) 'I don't see it here'
(m) 'I already have it'
(n) 'I've got too much to carry'
(o) 'I don't understand your command'

```
***************************************************
*            Program start                        *
*            =============                        *
*                  I                              *
*            Entry screen                         *
*                  I                              *
*                  +-<-------------------------+  *
*                  I                           I  *
*            Initialization of                 I  *
*            Flags,Situation etc.              I  *
*                  I                           I  *
*                  +-<-----------------------+ I  *
*                  I                         I I  *
* +-<--Over----Block 0 - A.P.L.              I I  *
* I                I                         I I  *
* I            show Location/Objects         I I  *
* I                I                         I I  *
* I                +-<-------------------+   I I  *
* I                I                     I   I I  *
* I            'What shall I do?     (a)(b)  I I  *
* I                I                     I   I I  *
* I            Check verb/noun---Error-->-+  I I  *
* I                I                         I I  *
* I                count 09 (c)(d)           I I  *
* I                I                         I I  *
* I            (GO)N/S/E/W/U/D-------OK--(s)-->-+ I  *
* I                I-        IError          I I  *
* I                I         +----(e)(f)(h)-->-+ I  *
* I                I                         I I  *
* I            Block A.P.L.                  I I  *
* +-<----Over--- corr.verb -----RETURN-------->-+ I  *
* I                I                         I I  *
* I            Any one line                  I I  *
* I                  begun? ---------Yes--(i)-->-+ I  *
* I                I                         I I  *
* I                TAKE/DROP ?-----No---(o)-->-+ I  *
* I                I                         I I  *
* I            TAKE/DROP automat.---OK-------->-+ I  *
* I                I                         I I  *
* I                +-------Error--(j)(k)-->-+ I  *
* I                             (l)(m)(n)      I  *
* +----------------->-+                        I  *
*                  I                           I  *
*               play again ?------Yes--------->-+  *
*                  I                              *
*             Program End                         *
*             ===========                         *
***************************************************
```

3 ADVENTURE PROGRAMMING LANGUAGE 'A.P.L'

The 'programming language' A.P.L. (Adventure Programming Language) - don't confuse with APL, a mathematically oriented high level language - is a speciality of the Adventure command module.

Internally tokens - i.e. a byte code - are used to represent the commands in a similar way like in BASIC. Since these tokens are very confusing they will have to be translated to 'mnemonics' in order to make them readable. This has been done for the EDITOR program so that you can read and enter A.P.L. programs quite straightforward. - Most of the mnemonics will have to be completed by adding one or two 'parameters'.

Knowledge of A.P.L. was gathered in analyzing the built-in A.P.L. interpreter so that everything should work fine. It must however be stressed that there are some faults and mistakes within the Adventure command module which make programming somewhat confusing, especially when using IF/ELSE constructions.

Since A.P.L is a language designed specially for adventure games it can in no way be compared to any other existing language. You will simply have to learn and accept it as given.

3.1 Structures

It has been mentioned before that the structure of an A.P.L. program is 'blocks' and 'lines'. A diagram will show this clearly enough:

```
Block 0 : +----------------------------+
          ILine  (0,1) p:........     I
          ILine  (0,2) p:.............  I
          ILine  (0,3) p:...          I
          ILine  (0,4) p:............. I
          ILine  (0,5) p:.......      I
          I   ctd.                    I
          +----------------------------+
          (p = probability value)
Block 1 : +----------------------------+
          ILine  (1,1) N:........     I
          ILine  (1,2) N:............ I
          ILine  (1,3) N:...          I
          ILine  (1,4) N:............ I
          I   ctd.                    I
          +----------------------------+
          (N=Noun #)
Block 5 : +----------------------------+
          ILine  (5,1) N:.......      I
          ILine  (5,2) N:............ I
          ILine  (5,3) N:...          I
          I   ctd.                    I
          +----------------------------+
          (N=Noun #)
```

There are no blocks for verbs 2,3 and 4 in the example given. As can be seen it is possible to write blocks with different numbers of lines and to write lines with different lengths (number of statements). The order of lines in one block is normally of some importance for correct program execution; it is however determined at execution time which lines will be executed by checking the 'header' value of each line.

3.1.1 Block #0 - continuous actions

3.1.1.1 Tasks

Block #0 is executed once every round of the play so that you should program here all the actions which must happen independently of a command.

3.1.1.2 Probability values

The 'header' of each line is a number from 0 through 100 indicating the probability (in %) with which this special line will be executed. A value of 0 (zero) will result in no execution at all!

3.1.1.3 Ordering of lines

There is no definition as to the order in which you arrange you lines in this block. You must however consider the logical interconnections which may result in a well-defined ordering of your program lines. For instance if you use a flag be sure to have set or reset it before!

3.1.1.4 Finishing a line

You can end the execution of one line by executing the statement '!RETURN'. The next line will then be executed with its respective probability. The same effect will result from executing '!'Carry Ox' when this action results in an error!

!Over and !'Score (100%) will even stop the execution of the block and end the game.

3.1.2 Block #1 ff - command-initiated actions

The Block # is the number of an existing verb which may not be an 'equivalent' verb (indicated by an asterisk '*' at its beginning). There must not exist another verb with the same 'abbreviation' (the first four or so letters) since in that case the second occurence would never be detected.

3.1.2.1 Tasks

Since the lines in these blocks will be executed depending on the command given you will program the corresponding actions here. Pay special attention to a correct relation to block #0.

3.1.2.2 Noun

The 'header' of each line represents the number of a noun from the noun list, or it is 0 (zero). In the latter case this line will be executed independently of the noun given; i.e. it is always executed. Therefore such lines will be found - if they are used at all - at the end of a block. In all other cases the line is executed if the noun given (only the 'main' noun, not the synonym numbers!) corresponds to the line header value.

It is possible to use the same header value several times in one block. Be however aware that any line which executes 'successfully' ends the execution of the block!

3.1.2.3 Ordering of lines

Again the only restriction is that the results should make some sense.

3.1.2.4 Finishing a line

This will be the same as in Block #0 except that the next line will not in all cases be executed.

Any line will be executed either 'successfully' or 'not successfully'; in the first case the execution of the block will be terminated. The execution of the statements '!Over' or '!'Score' (100%) will immediately end the game.

The only way to end a line 'not successfully' is by executing a condition which is not fulfilled WITHOUT having passed an !IF-statement in that line before. It is however important for the 'automatic TAKE/DROP' action whether any one line has even begun executing; in that case that action will NOT be started and the message 'I can't do that yet' will appear if the line operated 'not successfully'.

3.2 A.P.L. Syntax

All A.P.L. Statements are given in both a long and short version. The short version serves mainly for fast entering of lines whereas the long version will - if possible - be displayed for sake of clarity.

The short version is - in most cases - the long version stripped of all blanks and non-capital letters.

It may happen that lines are too long to show them on one screen if the long version is used. In those cases the short version is used if in 'edit'-mode so that subsequent editing and reentering is possible. In 'print'-mode however the long version will be used and the screen will be divided in several subsequent screens till the end is reached.

A.P.L. statements ALWAYS begin with either an '!' or an '?'-sign which MUST be present; the question-mark '?' designates conditional statements whereas the exclamation-mark '!' is used for logical statements (including !IF) and for actions.

As far as parameters (numbers) are used in the following lists these are meant as examples only. The numbers 7 and 8 were used for indication of list elements and flags, the number 3 for 'stored values' and the number '9' as a numeric value.

In reality you will have to care for a correct use of these numbers. For instance if you use M65 (Message #65) without defining M65 the surprising results are up to you...

3.2.1 Conditional statements (?)

Conditions are raised at the moment they are executed and can either be fulfilled or not. Since a sequential execution of conditions equals logically ANDing them, you cannot make use of an OR or other logical connections.

If any one condition is NOT fulfilled program execution will continue at the !ELSE corresponding to the !IF statement executed last. (!!!) If no !IF has been encountered this means 'end of the line (not successful)'

3.2.2 Actions (!)

Action statements will result in their respective consequences as soon as they are executed. This may alter subsequent conditional statements!

The actions may be very different depending on the statement; see lists below.

3.2.3 Logical statements

Logical statements serve for structuring within one line. It is not possible to branch from one line to another.

3.2.3.1 !IF/!ELSE

You can use IF-structures up to a level of six (6). Display using the 'long' version will indent the levels appropriately.

If any one !IF statement has been executed any subsequent false condition will result in a branch to the pertaining !ELSE statement - even if this statement is BEFORE the condition that raised the fault! Keep this in mind clearly, since you can easily program loops that way which can end the same. The IF/ELSE differs from the BASIC version !!!

You can however make use of this effect when programming carefully. As a general rule you can say that any condition of that kind must be fulfilled some time when executed over and over again. Otherwise the loop will be endless. See the examples given at the end of this chapter.

3.2.3.2 !RETURN

A !RETURN somewhere within the line results in immediate termination ('successful'). - Whenever an !IF is NOT followed by a !RETURN execution will continue after the !ELSE statement; only by means of a !RETURN you can cancel execution. Keep in mind that a '!'Carry 0x' which results in the message 'I've got too much to carry' acts exactly like a !RETURN.

3.2.4 Statements using objects

long version	short	description
?Carry 07	?C07	Do I carry 07 ?
?Not Carry 07	?NC07	Don't I carry 07?
?Visible 07	?V07	Is 07 visible?
?Not Visible 07	?NV07	Isn't 07 visible?
?Carry Visible 07	?CV07	Do I carry 07 or is 07 visible?
?Not Carry Visible 07	?NCV07	Don't I carry 07 and isn't 07 visible?
?Carry	?C	Do I carry anything?
?Not Carry	?NC	Do I carry nothing?
?07	?07	Is 07 existing somewh.?
?Not 07	?N07	Isn't 07 existing ?
?Exchange 07	?E07	Has 07 been removed from its starting

		location defined in the 'Situation'-list?
?Not Exchange 07	?NE07	Is 07 at its starting location ?
!Carry 07	!C07	Carry 07 without check
!'Carry 07	!'C07	Carry 07 and check for maximum number. NOTE: This may react like a !RETURN in case of error!
!Visible 07	!V07	Put 07 to the current location.
!Not 07	!N07	Let 07 vanish. i.e. : to L0!
!07 to L8	!07L8	Put 07 at loc. L8
!07 to 08	!0708	Put 07 to the location where 08 is now.
!Exchange 07 and 08	!E0708	Exchange the current locations of 07 and 08.

3.2.5 Statements using locations

long version	short	description
?L7	?L7	Am I at location L7 now?
?Not L7	?NL7	Am I not at L7 now?
!07 to L8	!07L8	Put 07 at loc. L8
!L7	!L7	Goto location L7
!Exchange L3	!EL3	Exchange the current location with 'stored location #3'

The last statement refers to the 'stored locations' the meaning of which will be defined by the same programmer. NOTE: L3 is NOT L3 of the list!!

Keep in mind that this statement exchanges (and not duplicates) the two values of current location and 'stored location'.

3.2.6 Statements with counters

long version	short	description
?#9	?#9	Is the c. equal to 9?
?>#9	?>#9	Is the c. greater than 9
?Not >#9	?N>#9	Is the c. not gr. than 9
!#9	!#9	Set c. value to 9
!+#9	!+#9	Increment c. by 9
!-#9	!-#9	Decrement c. by 9
!Display #	!D#	Display counter (+/-!)
!Exchange #3	!E#3	Exchange current counter with stored counter #3

The last statement refers to one of the 'stored counters' whereas all the other work with the current counter. The numeric values range from 0 to 255, whereas the counter value may be between -32768 and +32767. This means that you cannot check for negative values of any counter!

Note that the last statement will result in exchanging (not duplicating) the current and the stored counter!

3.2.7 Statements with flags

long version	short	description
?F7	?F7	Is F7 set (=1)?
?Not F7	?NF7	Is F7 reset (=0)?
!F7	!F7	Set Flag 7 to 1
!Not F7	!NF7	Reset Flag 7 to 0
!F[redacted] Inventory	![redacted]I	Set Flag for the automatic 'Inventory' (upper half of screen!)
!Not [redacted] Inventory	!N[redacted]I	Reset Flag 'Inventory'
!Flag Dead	!FD	Set Flag 'Dead' for '!Over'
!Flag Well Done	!FWD	Set Flag 'successful' for '!Over'

Note that only the first 4 statements work with the user-defined flags #0 to #31 which include F15/F16 for the 'light' operation. All other flags are internal and will have effects on the screen display.

3.2.8 Statements involving 'macros'

long version	short	description
!I'm Dead	!I'D	Set to last location
!'Inventory	!'I	Display 'Carrying...' in the lower screen
!'Carry 07	!'C07	Carry 07 and check for maximum; this may result in !RETURN and 'too much to carry'
!Init 09	!I09	Carry 'Lit torch' and initialize torch counter
!'Score	!'S	Score; at a score of 100% - !Over !!
!Save Game	!SG	Save Game etc.
!Wait	!W	Wait app. 1.2 sec
!Over	!O	End of the game

3.2.9 Statements involving display

long version	short	description
!M7	!M7	Display Message M7
!'Inventory	!'I	Display 'Carrying...' in the lower screen
!Display Visible	!DV	Redisplay upper screen
!Display #	!D#	Display counter value
!Display Noun	!DN	Display Noun just entered
!Display Noun and Scroll	!DNS	ditto; new line
!Scroll	!S	new line

Inventory always means the 'I'm carrying ...' display. If you use the flag, it will be shown in the upper screen half automatically every round, if you give the command '!'I' it will be displayed in the lower scree once.

!DV will redisplay the upper screen half completely. This may be useful if you want to display what is visible for instance when a match is lie and extinguishes again. - Note that all subsequent messages will be displayed above the line 'What shall I do?'. You may find this useful in some cases.

3.2.10 List of statements (sorted by short versions)

long version	short	description
!ELSE	!ELSE	End IF
!RETURN	!EN	End of Line
!IF	!IF	Begin IF
?Carry	?C	Do I carry anything?
?Carry 07	?C07	Do I carry 07 ?
?Carry Visible 07	?CV07	Do I carry 07 or is 07 visible?
?Exchange 07	?E07	Has 07 been removed from its starting loc.?
?F7	?F7	Is F7 set (=1)?
?L7	?L7	Am I at loc. L7?
?Not Carry	?NC	Don't I carry anything?
?Not Carry 07	?NC07	Don't I carry 07 ?
?Not Carry Visible 07	?NCV07	Don't I carry 07 and isn't 07 visible?
?Not Exchange 07	?NE07	Is 07 at its starting location?
?Not F7	?NF7	Is F7 reset (=0)?
?Not L7	?NL7	Am I not at loc. L7?
?Not 07	?N07	Doesn't 07 exist?
?Not Visible 07	?NV07	Isn't 07 visible?
?Not >#9	?N>#9	Isn't the c. gr. than 9?
?07	?07	Is 07 existing somewh.?
?Visible 07	?V07	Is 07 visible?
?#9	?#9	Is the c. equal to 9?
?>#9	?>#9	Is the c. gr. than 9?

!Carry 07	!C07	Carry 07 without check
!'Carry 07	!'C07	Carry 07 and check for maximum
!Display Noun	!DN	Display noun just entered
!Display Noun and Scroll	!DNS	ditto, with new line
!Display Visible	!DV	Redisplay upper screen
!Display #	!D#	Display counter value
!Exchange L3	!EL3	Exchange current loc. with 'stored loc.'
!Exchange 07 and 08	!E0708	Exchange the current locations of 07 and 08.
!Exchange #3	!E#3	Exchange the current c. with 'stored counter #3'
!Flag Dead	!FD	Set Flag 'Dead' for '!Over'
!~~Flag~~ Inventory	!~~F~~I	Set Flag for automatic 'Inventory'
!Flag Well Done	!FWD	Set Flag 'successful' for '!Over'
!F7	!F7	Set Flag 7 to 1
!'Inventory	!'I	Display 'Carrying...' in the lower screen
!I'm Dead	!I'D	Put to last location
!Init 09	!I09	Carry Lit Torch (09) and start time
!L7	!L7	Goto Loc. L7
!M7	!M7	Display Message M7
!Not ~~Flag~~ Inventory	!N~~F~~I	Reset Flag 'Inventory'
!Not F7	!NF7	Reset F7 to 0
!Not 07	!N07	Let 07 vanish
!Over	!O	End of game!
!07 to L8	!07L8	Set 07 to loc. L8
!07 to 08	!0708	Set 07 to the curr. location of 08.
!Scroll	!S	Begin new line
!'Score	!'S	Score; for 100% = '!Over'
!Save Game	!SG	Save Game etc.
!Visible 07	!V07	Set 07 to current location.
!Wait	!W	Wait app. 1.2 sec
!#9	!#9	Set c. value to 9.
!+#9	!+#9	Increment c. by 9.
!-#9	!-#9	Decrement c. by 9.

3.3 Examples for A.P.L. program lines

Some examples will show you how to begin programming in A.P.L. The examples given are of course simple and are only meant as a starting point.

There wasn't enough room in this reference guide to discuss in depth one complete game with its listing. It is recommended that you CONVERT and EDIT one of the existing games you have already played in order to understand 'how to do it'. A printer will help you doing so!

3.3.1 Conventions for the examples

The examples of A.P.L. lines cannot of course function properly without the pertaining lists, which however have not been listed. So if you use 'Message M10' be sure to have it defined! This applies to all of the lists mentioned.

All line numbers or block numbers are examples; they will have to be changed in your programs.

If explanations are given behind the statements; DON'T enter these texts if you decide to try that line!

3.3.2 Messages

3.3.2.1 Messages every round

Let's suppose you want to have a message every round with 10% probability. Solution:

```
(0,20)=  10!M10
           !RETURN
```

If however this message is to be given only if the player is at location L5, you enter:

```
(0,20)=  10?L5
           !M10
           !RETURN
```

If M11 is to be given at L5 and M10 elsewhere, you could write:

```
(0,20)=  10!IF
            ?L5
            !M11
            !RETURN
           !ELSE
           !M10
           !RETURN
```

If M11 AND M10 are to be displayed at L5 and M10 elsewhere, you simply skip one statement:

```
(0,20)=  10!IF
            ?L5
            !M11
           !ELSE
           !M10
           !RETURN
```

All of these messages will be displayed every round with a probability of 10%. If you prefer to have a message M10 with 100% probability if the player is at L5 and O10 is visible (not carried), you write:

```
(0,20)=  100?L5
           ?Visible O10
           !M10
           !RETURN
```

3.3.2.2 Messages displayed at command input

It's very common to display messages as a reaction to some command, for instance LOOK. If verb #25 is LOOK, you can enter:

```
(25,1)= 0!M15          LOOK      : M15
         !RETURN
```

Now let's suppose that LOOK WIND is to produce a different message. This can be done using two lines: (N32 = WIND)

```
(25,1)=    32!M16          LOOK WIND: M16
             !RETURN
(25,2)=    0!M15           LOOK      : M15
             !RETURN
```

Another example for messages will show you how to break messages into segments which can help in saving memory. Let's suppose that M16 is to be displayed for LOOK WIND and different parts of the message shall follow depending on the current location.

```
(25,1)= 32!M16
          !IF
             ?L7          at loc.L7: M16+M17/M19
            !M17
          !ELSE
          !IF
             ?L8          at loc.L8: M16+M18/M19
            !M18
          !ELSE
          !Scroll         begin new line
          !M19
          !RETURN         else: only M16/M19
```

These examples have already shown a lot of the structure of A.P.L. program lines. Try to think of more complex examples yourself!

3.3.3 Objects/Locations

Let's suppose you want to achieve the following result: Whenever the player carries O8, it vanishes and O10 appears together with message M10. This must be done in Block #0:

```
(0,15)=  100?Carry O8          carried?
             !Not O8           vanished!
             !Visible O10      O10 instead
             !M10              Message
             !RETURN
```

If you want to program the following effect: When carried O8 'changes to' O10 and O8 disappears to the old location of O10 - this is quite simple:

```
(0,15)=  50?Carry O8
            !Exchange O8 and O10
            !M10
            !RETURN
```

Now a more complex example: At location L7 - where the player must NOT be at the moment - the objects O15,O16 and O17 will turn to a new object O18 with some probability:

```
(0,15)=  10!Exchange L0        Save location
           !L7                 new location
           !IF
              ?Visible O15
              ?Visible O16
              ?Visible O17
              !Not O15
              !Not O16
              !Not O17
              !O18
           !ELSE
           !Exchange L0        Back to the
           !RETURN             current location
```

Another example will show 'picking up' objects. If instead of a quiet 'automatic' TAKE a message has to be displayed when doing so, you will have to do so yourself. Keep in mind that the 'reference'-table won't work for that kind of action! Note: Verb #10 = TAKE

```
(10,1)=  33?Visible 024
           !'Carry 024
           !M35        Message
           !RETURN
```

Locations will mainly appear as conditions since most of the movement will be done by the 'automatic' GO option.

If however you enter GO WIND or something like that you will have to program that yourself:

```
(1,2)=  32?L7          right location?
          ?Visible 019  open window etc?
          !L8           new location!
          !RETURN
```

There was an example in exchanging locations before, but it was a side effect in testing for objects.

You can achieve interesting results by exchanging locations! As an example let's imagine a game where some magic word will move you not to one place, but to the place where you used that word last!

Let V15 be SAY and N45 the 'magic word'. Besides 08 must be carried for the effect to take place.

```
(15,1)=    45?Carry 08
             !Exchange L1
             !M86
             !RETURN
```

Remember to initialize the 'stored location' L1 for the time he uses the magic word for the first time!

3.3.4 Flags

You will have to give meanings to the different flags when writing your game (if you decide to use them). For instance it is advisable to check for certain complex situations in Block #0 and store the result using a flag. This simplifies the actions you will take.

Let's suppose you want to check for L17, nothing carried, and the existance of three objects:

```
(0,38)=  100!Not F6          reset flag
            ?Not Carry
            ?L17
            ?Visible O2
            ?Visible O7
            ?Visible O13
            !F6              set flag
            !RETURN
```

You can later on use F6 instead of all the conditions. But be cautious: Don't alter the situation later on in block #0 without updating F6!

```
(45,3)=    36?F6
             !M28
             !Scroll
             !Wait
             !M29
             !Scroll
             !RETURN
```

Another important meaning for (one) flag is 'startup' You will perhaps have to initialize 'stored counters' and 'stored locations' (which the command module cannot do automatically). So you use one flag (F1) which you set once and never reset :

```
(0,1)=    100?Not F1       Startup
             !F1           set
             !#10          Number
             !Exchange #1    stored
             !#40          Number
             !Exchange #2    stored
             !Exchange L1  Save curr.loc.
             !L8           new loc.
             !Exchange L1    stored
             !RETURN
```

There are so many possibilities for a use of flags that any description will lack completeness. So this is enough for that.

3.3.5 Counters

You can use counters for a lot of things: whether the player 'starves' the 'time' goes by or perhaps you count x-y-coordinates during a game: In all those cases you can't do without counters.

Store them to work with several counters. The initialization process has been described before.

Let's suppose you want to decrement a counter ('time') every round and end the game when it reaches zero:

```
(0,14)=  100!Exchange #2       Get counter
            !-#1               -1
            !IF
               ?#0             =0?
               !M56
               !Flag Dead      dead
               !Over           end
               !RETURN
            !ELSE              else
            !Exchange #2       Save counter
            !RETURN
```

When using several counters be sure always to exchange the right values so that stored counters represent only one thing! This is somewhat difficult when using IF/ELSE statements.

Let's have an example for an x-y-coordinate calculation. Counter #1 = x-value , Counter #2 = y-value. You want to check for both values being equal to 50 and then set F9:

```
(0,23)=  100!Not F9
            !Exchange #1
            !IF
               ?#50
               !Exchange #1
               !Exchange #2
               !IF
                  ?#50
                  !F9
               !ELSE
               !Exchange #2
```

```
                !RETURN
           !ELSE
           !Exchange #1
           !RETURN
```

3.3.6 IF/ELSE

There have been some examples for IF/ELSE constructions in the examples. In those cases no special problems would arise since the IF behaved 'logically'. An example for the 'illogical' behaviour of this statement will follow. NOTE : The following example is WRONG !! (endless loop)

```
(45,1)=    23!IF            <I1>
             ?L4            <B1>
             !IF            <I2>
               ?Carry 06    <B2>
               !M65
             !ELSE          <E2>
             ?Carry         <B3>
             !M66
          !ELSE             <E1>
          ?Exchange 07    — <B4>
          !M67
          !EN
```

Let's see what happens: If <B1> and <B2> are true, <B3> must be true as well (OK.); if <B4> is true the line is executed 'normally'. If however <B4> is FALSE execution continues at <E1> or <E2> depending on the result of <B1>!!! In any case this will result in an endlessly looping program.

Now let's have an example for a correct use of this special type of IF/ELSE statement. Suppose you want to decrement the 'stored counter #1' by the value of the current counter. This cannot be done with any simple statement! Let's suppose that the value of the current counter is equal to or greater than 0 (zero) and that it may be destroyed in that subtraction.

```
(13,5)=   78!IF
             ?#0          current ctr.=0:Exit
             !RETURN
          !ELSE           <E1>
          !Exchange #1    Counter #1
          !-#1            decrement by 1
          !Exchange #1
          !-#1            ditto current counter
          ?#0             =0? Exit; else  <E1>!
          !RETURN
```

This finishes the short introduction in examples for A.P.L. program lines. Anything more detailed would certainly boost this reference guide to its double volume. You are strongly encouraged to try and find out yourself how it works by writing examples and analyzing existing programs. Your success will help you to learn it!

4 HOW TO USE THE 'ADVENTURE EDITOR'

4.1 General

The ADVENTURE EDITOR program is written completely in TMS9900 assembly code. Two versions exist at the moment: MM-version for use with the 'Mini Memory command module' and EA-version for use with the 'Editor/Assembler command module'. Both versions own the same commands and features so that one reference guide is valid for both versions. In programming however the restrictions for MM were considered, especially the available memory space, so that the EA version uses the available memory only to a small degree.

The program will support keyboard data entry, screen display and file operations with all available peripherals (Cassette, Diskette, RS232 etc.)

4.2 Equipment

The programs will ONLY function properly on a TI99/4A and NOT on the old TI99/4 home computer!

The minimum configuration of your home computer needed for this program depends on the program version. In any case however you can use all peripherals you want.

4.2.1 MM- (Mini Memory-) version

The program will be furnished on cassette for this version. Therefore you will need the console with the MM command module and a suitable cassette recorder at the least.

All available standard peripherals are supported however, so that you can make use of diskette files and a printer, if you want to.

The MM pseudo file EXPMEM2 (Memory expansion) is supported as well. You can use this file for fast data storage as long as you use the MM command module. This means that this pseudo-file will of course not be available in the Adventure command module.

4.2.2 EA- (Editor/Assembler-) version

This program version is furnished on diskette storage medium so that you will need the console, EA command module, memory expansion and at least one disk drive.

Cassette files are supported as well so that games can be read from cassette or stored there if so desired. A printer port can be used.

4.3 Loading and starting the program

The startup procedure depends on the program version.

4.3.1 Mini-Memory version

First be sure to have your MM initialized (Option 3 / Proceed) Then the program can be loaded from cassette using the EASY BUG 'L' option. Note: the available memory space in Mini Memory will be used up to the last byte so that no other program can be loaded!

You can start the EDITOR program by entering the program name 'START' when using the 'RUN'-Option of the Mini Memory selection list.

Command procedure:

```
*******************************************
* Choice Mini Memory (3)                  *
*        Reinitialize (3)                 *
*        (Proceed)                        *
* <Quit>                                  *
* Choice Easy Bug (2)                     *
*        (any key)                        *
*        L                                *
*        (observe cassette commands)      *
* <Quit>                                  *
* Choice Mini Memory (3)                  *
*        Run (2)                          *
*        START                            *
*******************************************
```

4.3.2 Editor/Assembler version

The program will be started using the 'LOAD AND RUN' option of the Editor Assembler. It will be loaded on absolute addresses using the lower memory expansion. Be sure to have no other program in memory when loading the EDITOR.

You start the program by entering the program name 'START'.

Command procedure:

```
***********************************************
* Choice Editor Assembler (2)                 *
*        Load and Run (3)                     *
*        DSK1.EDITOR                          *
*        <Enter>                              *
*        START                                *
***********************************************
```

4.4 Screen decription

When started the standard screen is displayed. A title screen or selection list was impossible due to memory space limitations. As a program identification the word 'EDITOR' is displayed in the status line followed by the revision No., i.e. '------EDITOR--Rev1' in this case.

The screen is divided into TEXT AREA (Lines 1-22), STATUS LINE (Line 23) and ECHO LINE (Line 24). The screen is 40 characters/line, i.e. the program uses text mode as does the adventure command module.

The TEXT AREA serves for displaying and editing game data. You will be allowed to enter all displayable characters in this field.

The STATUS LINE will display a message from time to time indicating program status. Whenever keyboard entry is possible the above mentioned program ID will be displayed here. ('-----EDITOR--Rev1')

The ECHO LINE serves for entering commands and for echoing prompts which are error messages at the same time. A continuous '%-full' display is given here as well. - You are NOT allowed to enter lowercase letters here since they are converted to uppercase as you enter them. This serves for simplifying operation by overriding the 'alpha lock' key.

4.5 Keyboard

Entering text has been made comfortable. A blinking 'underline' cursor will appear on the screen like in the adventure command module.

There is no auto-repeat for any key, so that you have to push any key as many times as you want it to execute.

4.5.1 Upper- and lowercase

Upper- and lowercase letters and other displayable characters have to be entered the same way as in other programs. The 'alpha lock' key will disable entering lowercase letters. All characters are displayed the same way as in the 'Adventure' command module, i.e. lowercase letters are 'real' lowercase, as soon as you have loaded a game.

The old TI99/4 is NOT able to enter lowercase letters, so for this reason you will not be able to work with that one.

As mentioned above entering of lowercase letters is allowed only in the text area. All input in the echo line will be displayed as uppercase. This feature has been added for simplifying command input since all commands and filenames are valid only as uppercase words.

4.5.2 Function keys

If you depress <FCTN> the following keys will have special meanings:

<FCTN> 1	= Delete Char	
<FCTN> 2	= Insert Mode	
<FCTN> 3	= Erase Field	
<FCTN> 5	= Begin	Move to top
<FCTN> 8	= Redo	Reexecute
<FCTN> 9	= Back	Abort entry
<FCTN> =	= QUIT	Caution!!!
<FCTN> E	= Up arrow	Line up
<FCTN> X	= Down Arrow	Line down
<FCTN> S	= Left arrow	Char right (wrap!)
<FCTN> D	= Right arrow	Char left (wrap!)

You can leave INSERT mode the same way as you do in BASIC.

The REDO key is important when reexecuting commands, since just pressing <ENTER> will be interpreted as 'no entry'.

The BACK key will be used mainly when aborting false inputs in the text area which otherwise would be stored.

4.6 Echo line

The ECHO LINE (24th line) serves for entering commands and for receiving (error-) messages.

4.6.1 Messages in the Echo Line

The input prompt generated by the program serves as error message at the same time. Whenever the command executed last was successful and without errors, the ' OK: ' prompt is displayed. If however there was some kind of error or a warning has to be given it will change to 'xx ER! '. The letters 'xx' indicate the type of error that occured last. In both cases the next command can now be entered.

4.6.1.1 Error messages

The letters 'xx' within the error-prompt can have the following values and meanings:

xx	Meaning	
CD	Command Error	- invalid command
FI	File Error	- I/O or File error for games
PR	Print Error	- I/O or File error for printer The I/O error will not be analyzed.
ID	Ident Error	- The program in memory is a) not an adventure game b) not yet converted
TX	Text Error	- lowercase etc.
MO	Memory Overfl.	- (VDP-)memory overflow
IO	Index Overfl.	- Number too big or too small (indices only)
NO	Numeric Overfl.	- Number too big
SO	Screen Overfl.	- Screen Overflow
SY	Syntax Error	- invalid A.P.L. syntax
LI	Line too long	- A.P.L. line too long or text after last !RETURN
LA	Last line too long	- last A.P.L. line too long. (converted game only)
IF	IF/ELSE Error	- A.P.L. error in IF/ELSE structures
RT	Return assumed	- Warning: !RETURN assumed on any level of A.P.L.

4.6.1.2 ... percent full message

At the end of the echo line a number is displayed before the execution of every command. This will indicate how much of the available (VDP-) memory is occupied by the game at present. This message will be given only if a (converted) game has been loaded.

This percentage is deriven from an 'available VDP memory space' which has been chosen as to enable data storage on diskette medium. (i.e. up to >3780V). The adventure command module however will accept files longer than that (up to >3A98V) which will ONLY be possible on cassette storage medium with no diskette controller connected.

Note: this can lead to the effect of not being able to load future games with this program and - by the way - not from diskette within the adventure command module. Games of that kind will however not have been written by means of the ADVENTURE EDITOR. All games presently available (12 games) do not exceed that size and can be loaded and edited.

Example:

```
***************************************
*-----------------------------EDITOR--Rev1*
*   OK: E                          95%*
***************************************
----------    95% of the memory occupied !
```

4.6.2 Commands in the echo line

Commands given in the echo line may be preceded by blanks which will be skipped. Within a command blanks are not allowed with the exception of all file commands.

Commands consist of an uppercase letter and extensions.

4.6.3 Ending program execution (Q)

In order to end program execution properly enter 'Q'. This will close any open print file unit. Game data stored in VDP memory are lost after execution of this command, so be sure sure to have them stored before doing so!

Example:

```
****************************************
*---------------------------EDITOR--Rev1*
*    OK: Q                          95% *
****************************************
```

4.6.4 File commands

All file commands require a valid filename which may be the filename of any valid peripheral (see instructions for your peripheral(s)). A blank must be entered following the command (I,F,P) and preceding the filename (for instance 'I CS1')

When using the MM version of this program you can make use of the pseudo-file EXPMEM2 if you have the memory expansion connected to your home computer. This enables fast data storage and retrieval as long as the MM is operating. Be sure to have stored your games on cassette before removing the MM or switching off your computer!

Whenever you use the cassette medium (CS1 or CS2) the usual operating instructions will appear and have to followed.

If during a file operation undefined characters will appear on the screen this is not fatal as long as they disappear at the end of that operation. If however you destroy the character set by loading a BASIC program for instance you will have to restart the EDITOR.

Note: Due to an error in the internal file routines it may come to a system crash when entering invalid file names. When this happens all data are lost and your computer has to be switched off and on again.

4.6.4.1 Loading adventure games (I)

The 'I' command followed by a filename will load a game from that peripheral to VDP memory. Any game currently stored there will be destroyed by that operation.

After loading the game the program checks for a correctly 'converted' game and returns and 'ID ER!' if this check fails. If a file or I/O error will occur during the loading operation the message 'FI ER!' will be returned. In that case check the game in memory - if you want to use it - since it may have been (partly) destroyed.

Examples:

```
****************************************
*-----------------------------EDITOR--Rev1*
*     OK: I CS1                          *
*   OK: I DSK1.ADVENTURE                 *
*   OK: I EXPMEM2                        *(1)
****************************************
------- (1) MM with mem. expansion only !!!
```

4.6.4.2 Storing adventure games (F)

You can store games on any peripheral using the command 'F' and a valid filename.

Any stored (and converted) game may be stored on any peripheral. If an 'ID ER!' is returned for that command the game was either not yet converted or there was no game at all. A 'FI ER!' (File error) indicates that something is wrong with the peripheral specified; the game has NOT been stored or changed in that case.

Examples:

```
****************************************
*-----------------------------EDITOR--Rev1*
*     OK: F CS1                          *
*     OK: F DSK1.ADVENTURE               *
*     OK: F EXPMEM2                      *(1)
****************************************
------- (1) MM with mem. expansion only !!!
```

4.6.4.3 Opening a printfile (P)

Using the command 'P' with any valid filename you will OPEN a printfile on that peripheral. Any display resulting from a '!' command (see below) will then be directed to that file in parallel to screen output. This will continue until you close that file unit.

The file is opened as a SEQUENTIAL,DISPLAY,VARIABLE 80,OUTPUT file. The first 40 characters of any one line will be used however for text output. - You are NOT allowed to specify CS1/CS2 or EXPMEM2 as a print file unit.

After opening this file unit the header '-Processed by WEIAND ADVENTURE EDITOR-' will be printed.

The print file unit will be CLOSEd in one of the following cases:
a) List File Error ('PR ER!')
b) 'I' or 'F' - command even if
   this will result in an error.
c) 'Q' command - but NOT when the
   <FCTN>= (Quit) key is used !!!
This means that you can issue the command 'P CLOSE' which will result in a 'PR ER!' but will close the print file unit.

Examples:
```
****************************************
*-----------------------------EDITOR--Rev1*
*    OK: P RS232.BA=600                 *
*    OK: P DSK2.ADV-LIST                *
*    OK: P CLOSE                        * (3)
****************************************
---------- (3) will result in a PR ER!
           thus closing the file unit.
```

4.6.5 Data manipulation

There are commands for each list you can access. These will have to be modified depending on the actions you want to take.

4.6.5.1 List selection

The data manipulation commands are single capital letter commands derived from the list name:

- Messages (M)
- Objects (O)
- Locations (L)
- Verbs (V)
- Nouns (N)
- References (R)
- Situation (S)
- Connections (C)
- A.P.L. (A)
- General Information (G)
- Entry screen (E)

4.6.5.2 Using indices

Directly following the letter you will generally have to specify one or two parameters. Two numbers have to be separated using a comma (',').

- Edit A.P.L.	: Block#, Line#
- Edit other lists	: Element#
- Display A.P.L.	: Block#(from),Block#(to)
- Display other lists	: Element#(from),Element#(to)

The commands 'E' (Entry screen) and 'G' (General Information) need no parameter at all.

Examples:

```
****************************************
*-----------------------------EDITOR--Rev1*
*     OK: E                           70%*
*     OK: MO                          70%*
*     OK: A1,2                        70%*
*     OK: 3M                          70%*
*     OK: !00,100                     70%*(5)
*     OK: !0,100                      70%*(6)
****************************************
---------- (5) and (6) are identical !
```

4.6.5.3 Adding elements to lists (&)

A leading '&'-sign allows adding elements to the list specified by the command letter. This can be done in lists M, O, L, V, N and A.

Any new element will be given the default value, i.e. '?' for text lists and 100!RETURN for an A.P.L. line. This will have to be changed to the desired value at once or later on.

Adding elements is generally possible only AT THE END OF THE LIST !!! This means that specifying a parameter is not necessary. Keep this in mind: It is NOT POSSIBLE to insert new elements within the lists!!

An exception from this occurs in editing A.P.L. programs since it is possible (and necessary) to insert lines between existing lines. If you specify a pair of numbers here the new line will be inserted BEFORE the line specified. A number zero or no number at all will add the new line to the end of that block.

Examples:

```
*****************************************
*-------------------------------EDITOR--Rev1*
*     OK: &M                          70%*
*     OK: &O                          70%*
*     OK: &A1,2                       70%*
*     OK: &A0,0                       70%*(4)
*     OK: &A                          70%*(5)
*****************************************
---------- (4) and (5) are identical !
```

4.6.5.4 Deleting elements

Deleting elements from the lists is achieved by returning an 'empty' text area for that element (not by altering the command).

This is possible for the lists specified above; only A.P.L. lines can really be deleted (purged), whereas other list elements will be replaced by the default value '?'. The reason for this is that purging list elements would result in the necessity to rewrite all of the A.P.L. program!

4.6.6 Displaying text (!)

The commands described above will call one element and display it in the text area so that it can be edited.

If however you want to have the elements displayed you can specify the '!'-character in the first place. In that case the element will only be displayed; it can't be edited. If a print file unit has been opened output will be directed to that peripheral in parallel.

If you specify more than one parameter the corresponding list elements will be displayed one by one automatically. This will help you finding a special list element and also printing parts or all of the same contents.

If during this display process you press any one key (except <BACK>) the automatic display will stop and wait for another key so that you can have a look at the current element displayed. The message '---Waiting ...' is displayed at that time. If the next key is <BACK>, the display process is aborted while any other key will continue the quick search.

Note: If a print file has been opened the keyboard is scanned only after printing out one screen completely.

If you specify A.P.L. to be displayed the numbers following the '!A.' both indicate blocks of A.P.L. lines; i.e. you can't specify single lines in this mode. The display however will show the current block and line number as it does in edit mode.

If during display of any A.P.L. line it won't fit on one screen the first screen is displayed (long versions of A.P.L. commands) and printed if a print file has been opened, then the next screen will continue that line and so on. In that case the block/line number shown in the upper left corner will be followed by the plus ('+') sign in place of the usual '=' sign to indicate that the rest follows on the next screen.

Examples:

```
****************************************
*--------------------------EDITOR--Revi*
*   OK: !M10                       70%*
*   OK: !M0,100                    70%*(2)
*   OK: !M,100                     70%*(3)
*   OK: !A2                        70%*(4)
*   OK: !A,100                     70%*(5)
****************************************
---------- (2) and (3) are identical !
---------- (4) displays block #2 (all lines)
---------- (5) displays blocks #0 through 100
```

4.7 Editing text

Editing text data in the text area is possible only if the element has been called from edit mode (not display/quick search mode).

In the upper left corner the element number is shown followed by the equal ('=') sign if the element has been found.

If the element specified has been found the cursor will move to the first screen position within the text area. You can edit the text data as you like. Pressing <ENTER> or <REDO> will store the current display whereas <BACK> will abort data entry. This is important if you changed the values in some way and don't want them to be stored like that.

Whenever you ENTERered a screen the program checks for validity. If no error is found the results are stored. This happens word by word or (in A.P.L.) command by command so that your data may be truncated if an error is found!

The message '---Working ...' will be displayed in the mode line when this 'checking and storing' process is active. Note: It may take some time till the process is finished especially when entering long lines and/or when memory is filling!

4.7.1 Messages (M)

Messages are text elements used from within the A.P.L. . You can enter this text freely. When redisplayed all multiple blanks are removed and replacec by single blank signs.

The maximum word length is 32 characters. The total length of one message is only limited by the adventure display. Note: Specifying messages which - when displayed on one adventure screen at a time - will produce a screen overflow may cause a system crash within the adventure command module!

When displayed by the adventure command module words are never wrapped around the end of a line; a new line is begun instead. This will NOT happen within the ADVENTURE EDITOR where you can write words wrapping around the screen border. Have in mind the effect that display will look different during the game.

Example:
```
****************************************
*(10)= This is a sample message        *
*                                      *
*                                      *
*                                      *
*                                      *
*                                      *
*-----------------------------EDITOR--Rev1*
*  OK: M10                          70%*
****************************************
```

4.7.2 Objects (O)

The texts for objects are entered the same way as described for messages. Remember that objects beginning with the asterisk ('*') are 'treasures'!

4.7.3 Locations (L)

The texts for locations are entered the same way as described for messages. Have in mind that locations beginning with an asterisk ('*') will not be preceded by 'I'm in a ...' when displayed during the game!

4.7.4 Verbs (V)

The method for entering verbs is the same as entering other texts. They may however only be made out of uppercase letters. Using lowercase results in a 'TX ER!'. The length of any verb must be less than or equal to the length of 'abbreviations' specified within the 'general information' (see below). This covers NOT the asterisk ('*') which indicates that this verb is a synonym for the preceding one.

4.7.5 Nouns (N)

Nouns are entered exactly like verbs.

4.7.6 References (R)

References can be given for any object, i.e. the parameter given is the number of an object. Consequently the number displayed in the text area is a noun number (N..). If you enter 'NO' this will mean: there is no reference to TAKE/DROP this object.

Example:

```
*****************************************
*(10)= N5                               *
*                                       *
*                                       *
*                                       *
*                                       *
*                                       *
*-----------------------------EDITOR--Rev1*
*   OK: R10                         70%*
*****************************************
```

4.7.7 Situation (S)

The situations are as well given for objects so that the parameter of this command is the number of an object as well. The screen will display the number of a location (L..). If you enter 'L0' this means: the object doesn't exist (at the moment); if you enter 'Carry' it means exactly that.

Example:
```
****************************************
*(78)= Carry                           *
*                                      *
*                                      *
*                                      *
*                                      *
*                                      *
*-----------------------------EDITOR--Rev1*
*  OK: S78                          70%*
****************************************
```

4.7.8 Connections (C)

You enter connections between locations so that the parameter of this command is a location number. The results displayed are the location numbers of the six other location which can be reached by entering a direction command. If you enter 'L0' in one of the positions this means that there is no exit in this direction.

Example:
```
****************************************
*(14)= L0                           *(1)
*      L1                           *(2)
*      L0                           *(3)
*      L3                           *(4)
*      L11                          *(5)
*      L0                           *(6)
*                                      *
*--------------------------------EDITOR--Rev1*
*  OK: C14                          70%*
****************************************
---------- (1) North  : No connection
---------- (2) South  : Location #1
---------- (3) East   : No connection
---------- (4) West   : Location #3
---------- (5) Up     : Location #11
---------- (6) Down   : No connection
```

4.7.9 A.P.L. (A)

The game structure has to be entered using the 'program language' A.P.L. which has been described in the last chapter. In order to know an A.P.L. command the EDITOR requires only the uppercase letters, numerics and the characters "!","?","#",">","'" if they belong to that command. Lowercase letters and blanks are for 'decorative' purposes and clarity only.

The display is 'structured' if possible; i.e. !IF/!ELSE levels are suitably indented. This is however only done to make structures clear and can be altered or completely forgotten when entering data.

If the line is longer than one screen it will be redisplayed using the 'short command versions' and without indenting. This makes the line somewhat hard to read but it enables editing and reentering. Structured display may later be achieved using the '!' option.

The first number shown is the line 'header'. It may be edited but not deleted.

Example:

```
****************************************
*(1,5)=  20?L3                         *
*          ?Carry 017                  *
*          !M33                        *
*          !IF                         *
*            ?F7                       *
*            !M34                      *
*          !ELSE                       *
*          !RETURN                     *
*                                      *
*----------------------------EDITOR--Rev1*
*  OK: A1,5                        70%*
****************************************
```

4.7.10 General information (G)

Several numeric data are gathered here which are preceded by the '#' sign for that reason. - This command has no parameter.

Example:

```
*****************************************
*(0) = #6                               * (1)
*      #1                               * (2)
*      #10                              * (3)
*      #3                               * (4)
*      #0                               * (5)
*      #100                             * (6)
*      #15                              * (7)
*                                       *
*------------------------------EDITOR--Rev1*
*  OK: G                             70%*
*****************************************
---------- (1) Number of objects to carry=6
---------- (2) Starting location = L1
---------- (3) Max. number of treasures=10
---------- (4) Length of nouns/verbs =3
---------- (5) 0 x 256 +
---------- (6) 1 x 100 = 100 Steps for Torch
---------- (7) 'Score'-Location= L15
```

4.7.11 Entry Screen (E)

This command has no parameter.

The so-called entry screen will be shown once when the game is started. It can be edited here. The text should be written in a way as to identify the game for the future player.

The last two lines of the entry screen are 'Copyright' lines for the ADVENTURE EDITOR and can't be altered.

5 HOW TO USE 'ADVENTURE CONVERSION'

5.1 General

The ADVENTURE CONVERSION program is very similar to the EDITOR program as to its operation. It will however be used only rarely; i.e. only when you want to edit existing games which were not written with the ADVENTURE EDITOR.

Why do you need a special program for that purpose? Well, the games available from Texas Instruments contain all the texts and lists needed to play the game, but some internal data neede for the editor are missing or given inconsistently. The CONVERT program will generate or update this information from the existing lists and store them invisibly with the game. This means that a conversion process is needed only once for any one game. It will however not do any harm to a game if you 'convert' it several times by accident.

Especially the game contents will not be altered. There are two minor exceptions to this rule: A. The Entry screen will be changed by adding the copyright lines. B. Due to a false number of 'objects' given in some of the available games it will be impossible to load previous 'game situations' ('SAVE GAME') after conversion, since this number has to be updated by the CONVERT program.

5.2 Equipment

All of the remarks for the EDITOR program apply to ADVENTURE CONVERSION as well.

5.3 Loading and starting the program

The startup procedure is the same as for the EDITOR with the exception of the diskette filename which is 'DSK1.CONVERT' or the second program on cassette side A, respectively.

5.4 Screen, Keyboard and Echo line

All of this is same as for the EDITOR with the exception of the program ID given in the mode line which is '---CONVERT-Rev1'.

Error messages 'xx ER!' are the same but restricted to the following types:

```
xx   Meaning
--------------------------------------------------
CD   Command Error - invalid command
FI   File Error    - I/O or File error for games
PR   Print Error   - I/O or File error for printer
                     The I/O error will not be
                     analyzed.
ID   Ident Error   - The program in memory is
                     a) not an adventure game
                     b) not yet converted
--------------------------------------------------
```

5.5 Ending program execution (Q)

In order to end program execution properly enter 'Q'. This will close any open print file unit. Game data stored in VDP memory are lost after execution of this command, so be sure sure to have them stored before doing so!

Example:

```
***************************************
*---------------------------CONVERT-Rev1*
*  OK: Q                          95% *
***************************************
```

5.6 File commands

File commands are the same for CONVERT as for the EDITOR.

5.6.1 Loading/Saving games

When loading a game ('I') the VDP memory will be cleared first. This takes some time so that the message 'Working...' will be shown in the mode line. This results in destroying any game in memory even if no loading takes place due to a 'FI ER!'.

Keep in mind that you can 'load' ('I') any program file (i.e. any game even if not yet converted) but that a 'save' operation ('F') will only work for converted games since the game length is determined during conversion. If you try to save a game which is not yet converted an 'ID ER!' will be returned.

Have in mind the possibility of using the pseudo-file EXPMEM2 when working with the MM program version and having your memory expansion connected. This enables fast data storage for handing over games to the EDITOR.

Examples:

```
******************************************
*-------------------------------CONVERT-Rev1*
*   OK: I CS1                              *
*   OK: I DSK1.ADVENTURE                   *
*                                          *
*    OK: F CS1                             *
*    OK: F DSK2.ADVENTURE1                 *
*    OK: F EXPMEM2                         *(1)
******************************************
------ (1) ONLY MM with memory expansion !
```

5.6.2 Opening a printfile (P)

You can open a print file unit the same way as you do it in the EDITOR program. This print file will however only print 'statistics' if you enter the command to do so.

Examples:

```
******************************************
*-------------------------------CONVERT-Rev1*
*    OK: P RS232.BA=600                    *
*    OK: P DSK2.ADV-LIST                   *
*    OK: P CLOSE                           * (3)
******************************************
---------- (3) will result in a PR ER!
           thus closing the file unit.
```

5.7 Command 'GO'

G or GO will start the conversion process. This will work only if you have 'loaded' an adventure game, otherwise an 'ID ER!' will be returned.

The message 'Working...' will appear in the mode line together with some numbers indicating the actual program step. These numbers have no special meaning to you; they just show that the program is still working. After completion the message 'Processed by Weigand Adventure Editor' will appear in the first lines of the text area.

```
****************************************
** Processed by WEIAND ADVENTURE EDITOR **
*(c)M.Weiand Feldgärtenstr.50 D5 Köln 60 *
*                                      *
*                                      *
*                                      *
*                                      *
*                                      *
*-----------------------------CONVERT-Rev1*
*  OK: G                           70%*
****************************************
```

5.8 Command 'Statistics'

Entering S will result in displaying some useful information on the game in memory which must be a converted one. These are clear enough so that they have not to commented here. You can make use of these data when printing or displaying an overall documentation of a game since you will get to know how many items there are in any one list.

!S will print the screen on a print file unit in parallel to the display if the 'P' command has been issued before.

```
****************************************
** Processed by WEIAND ADVENTURE EDITOR **
*(c)M.Weiand Feldgärtenstr.50 D5 Köln 60 *
*                                      *
*       Number of Messages : 0         *
*       Number of Objects : 10         *
*       Number of Verbs : 20           *
*       Number of Nouns : 20           *
*       Number of Locations : 2        *
*       You can carry 6 objects        *
*       Starting Location : 1          *
*       Number of Treasures : 0        *
*       Location for Treasures : 1     *
*       Length of Nouns/Verbs : 4      *
*                                      *
*       Bytes occupied : 2512 of 13184 *
*                                      *
*------------------------------CONVERT-Rev1*
*   OK: !S                          19%*
****************************************
```

6 EMPTY 'ADVENTURE TEMPLATE' FOR YOUR OWN ADVENTURE GAMES

Together with the EDITOR you received a file named TEMPLATE which is designed to enable writing games 'from the start'. A short decription follows.

The lists shown below are the same as generated by the EDITOR program when printed out, so this is an example for the documenation you will be able to have when using the EDITOR.

6.1 Why you should use the 'template'

A template will offer to the EDITOR program the basic structures it needs for generating an adventure game. This 'template' structure had to be 'hand-made' using a debugger and stored for your convenience. It is necessary to use the template since you cannot delete list elements from existing games with the EDITOR.

Furthermore the pre-defined values and some more information needed for any adventure game have been inserted as a start to your work.

6.2 What you can do with the 'template'

You can expand the lists given in the 'template' in any way you like. You can even alter the information stored in it if you like so; but be aware of what is pre-defined in the adventure command module (see the chapter 'Structure of an adventure game' if in doubt).

6.3 Loading the TEMPLATE

6.3.1 Mini-Memory version

If you own the MM (Mini-Memory) version the TEMPLATE file will be on cassette medium like the programs.

So you start the EDITOR program as usual and enter the command 'I CS1'. The file is located on cassette(each side).

6.3.2 Editor-Assembler version

Using the EA (Editor Assembler) version the TEMPLATE file is located on the diskette together with the programs EDITOR and CONVERT.

So after the standard startup procedure for the EDITOR you enter the command 'I DSK1.TEMPLATE'. When saving it change your diskettes since the one you got is write protected!

6.4 TEMPLATE Contents

When executing the 'Statistics' command from within the CONVERT program and having the TEMPLATE loaded before you will receive the following results (display/printout) :

```
Number of Messages : 0
Number of Objects : 10
Number of Verbs : 20
Number of Nouns : 20
Number of Locations : 2
You can carry 6 Objects
Starting Location : 1
Number of Treasures : 0
Location for Treasures : 1
Length of Nouns/Verbs : 4

Bytes occupied : 2512 of 13184
```

The following printouts can all be generated using the EDITOR by issuing the commands printed out on top of each list:

6.4.1 Entry Screen (E)

```
------------------------------EDITOR--Rev1
  OK: !E                              19%
            * My own Adventure *
 ........................................
 .                                      .
 .                                      .
 . You may construct your own ADVENTURE .
 .GAME with this template. You will have.
 .to add: -L-ocations                   .
 .        -O-bjects                     .
 .        -V-erbs                       .
 .        -N-ouns                       .
 .        -M-essages                    .
 .        -C-onnections                 .
 .        -S-ituations                  .
 .        -R-eferences                  .
 .        -E-ntry screen                .
 .        -G-eneral Information         .
 .     and-A-dventure Program Statements.
 .                                      .
 . Read carefully the instructions for  .
 .   the ADVENTURE EDITOR program!      .
 ........................................
                         now press ENTER...
```

6.4.2 General Information (G)

```
------------------------------EDITOR--Rev1
  OK: !G                              19%
(0)=      #6
          #1
          #0
          #4
          #0
          #100
          #1
```

6.4.3 Messages (M)

```
---------------------------------EDITOR--Rev1
   OK: !M,10                             19%
```

6.4.4 Verbs (V)

```
---------------------------------EDITOR--Rev1
   OK: !V,20                             19%
(0)=       ?
(1)=       GO
(2)=       *WALK
(3)=       *RUN
(4)=       ?
(5)=       ?
(6)=       ?
(7)=       ?
(8)=       SAVE
(9)=       QUIT
(10)=      TAKE
(11)=      *GET
(12)=      *PICK
(13)=      ?
(14)=      ?
(15)=      ?
(16)=      ?
(17)=      ?
(18)=      DROP
(19)=      *GIVE
(20)=      ?
```

6.4.5 Nouns (N)

```
---------------------------------EDITOR--Rev1
   OK: !N,20                             19%
(0)=       ?
(1)=       NORT
(2)=       SOUT
(3)=       EAST
(4)=       WEST
(5)=       UP
(6)=       DOWN
(7)=       GAME
(8)=       ?
(9)=       ?
(10)=      ?
(11)=      ?
(12)=      ?
(13)=      ?
```

```
    (14)=   ?
    (15)=   ?
    (16)=   ?
    (17)=   INVE
    (18)=   ?
    (19)=   ?
    (20)=   ?
```

6.4.6 Objects (O)

```
    ------------------------------EDITOR--Rev1
      OK: !O,10                         19%
    (0)=    ?
    (1)=    ?
    (2)=    ?
    (3)=    ?
    (4)=    ?
    (5)=    ?
    (6)=    ?
    (7)=    ?
    (8)=    ?
    (9)=    Lit torch
    (10)=   ?
```

6.4.7 References (R)

```
    ------------------------------EDITOR--Rev1
      OK: !R,10                         19%
    (0)=     NO
    (1)=     NO
    (2)=     NO
    (3)=     NO
    (4)=     NO
    (5)=     NO
    (6)=     NO
    (7)=     NO
    (8)=     NO
    (9)=     NO
    (10)=    NO
```

6.4.8 Situation (S)

```
--------------------------------EDITOR--Rev1
    OK: !S,10                            19%
     (0)=      LO
     (1)=      LO
     (2)=      LO
     (3)=      LO
     (4)=      LO
     (5)=      LO
     (6)=      LO
     (7)=      LO
     (8)=      LO
     (9)=      LO
     (10)=     LO
```

6.4.9 Locations (L)

```
--------------------------------EDITOR--Rev1
    OK: !L,10                            19%
     (0)=      ?
     (1)=      ?
     (2)=      ?
```

6.4.10 Connections (C)

```
--------------------------------EDITOR--Rev1
    OK: !C,10                            19%
     (0)=      LO
               LO
               LO
               LO
               LO
               LO
     (1)=      LO
               LO
               LO
               LO
               LO
               LO
     (2)=      LO
               LO
               LO
               LO
               LO
               LO
```

6.4.11 A.P.L. (A)

```
-----------------------------------EDITOR--Rev1
   OK: !A,20                                19%
(0,1)=    100!RETURN
(8,1)=    7!Save Game
           !RETURN
(9,1)=    0!Over
           !RETURN
(10,1)=   17!'Inventory
           !RETURN
```

7 ADDITIONAL INFORMATION

7.1 On the history of ADVENTURE EDITOR and ADVENTURE CONVERSION

The program package ADVENTURE EDITOR results from days (and nights) of a private computer hobby. There was no backup from Texas Instruments or Adventure International so the author asks you for some understanding.

As any TI99/4A owner who is interested in the 'internals' of his home computer will know the command modules available will result in an 'easy-to-use' program startup but are likely to lock the internals to anyone who wants to go a little deeper.

This is valid for the adventure command module as well : You can only buy the command module and the available games; writing or editing them is simply impossible.

When - hunting for all bits of information - you open the command module (you did it, i know...) you will see a little chip which will seem curious at the first look and even at the second one.

Well, this is one of the ingenious GROMs (Graphic ROMs) used so often by TI. It has the incredible advantage of not being available and thus not being interesting for 'pirates'. GROMs are by the way 'byte oriented serial ROMs' which makes replacement by other chips impossible.

The content of this little 'program safe' can be read from within machine language - but the next problem is arising soon: What the hell is that!? No machine language, no BASIC, but texts embedded in curious bytes ..

You have just come across the famous GPL (Graphics programming language) which is the 'native language' of your TI99/4! For some very understandable reasons TI is not willing to hand out information on that one - it simply seems to be 'top secret'.

So the only way to get along was to have a - very - close look at the internal ROM on addresses >0000 through >2000 which contains among other parts of the system the GPL interpreter. Analyzing that part of the system ROM helped to understand how GPL is structured and to get to know most of its commands. By the way GPL has nothing to do with graphics! It might be interesting to know why it was called so in order to understand what TI had in mind when planning the TI99/4 ...

GPL is a very interesting language with several very mighty addressing modes. It is quite well suited for writing complex programs requiring relatively few amounts of memory space, especially when making use of VDP, GROMs and RAM/ROM in turn. One great disadvantage is known to all those who have compared GPL (command modules) to pure assembly language : It is comparatively slow due to the interpreter overhead.

There can be no discussion of GPL here which would be of no use to most of the readers.

Anyway, with that GPL stuff in mind an analyzis of the adventure command module GROM was an easy thing.

A new type of structure came into view: the adventure game. Finally it was possible to write an assembly language program to edit adventure games and to write your own ones.

A special problem arouse in finding a name and command descriptions for the internal 'program language': It was called 'A.P.L. = Adventure Programming Language', and the command names seem to be suitable for fast data entry and readable display as well.

As you can see from all that there is a lot of work hidden behind the surface of the ADVENTURE EDITOR. But a new type of program has been born - one that makes use of the 'program safes' named command modules. This may seem simple to the user, but it is really the result of many a night spent by a computer freak.