The Professional Adventure Writing System

Contains

- Quick ref-card
- Intro guide
- Tech guide
- v.16+ supplement
- PAW essays
- Mag-published routines (compilation)

Programming Bible

v.1a 2014

Acknowledgement

Thanks to Howard and Pam for their forbearance. Phil for his 'comments', Graeme for his ideas, Dicon et al for the graphics and all our customers for their support and suggestions.

Function of P.A.W. Flags

The normal flags are free for use in any way in games. The auto decrement flags (2 to 10) are also free for use, but be sure you know in which situations they are reduced before using them. Other flags should mostly only be set using the appropriate action, but useful tests can be carried out on their contents.

Flag 0 When non zero indicates game is dark (see also object 0)
Flag 1 Holds quantity of objects player is carrying (but not wearing)

The following flags are decremented if non zero by PAW;
Flag 2 When a location is described
Flag 3 When a location is described and it's dark (Flag 0 not 0)
Flag 4 When a location is described, it's dark and object 0 is absent
Flags 5 to 8 Every time frame (i.e. every phrase/timeout)
Flag 9 Every time frame that it's dark
Flag 10 Every time frame that it's dark and object 0 is absent

Flags 11 to 28 are free for use in your own games

Flag 29 holds Picture Control flags
Bit 7 - Set this to force picture to be drawn (LOOK)
Bit 6 - Set this to always draw picture (PICS ON)
Bit 5 - Set this to never draw picture (PICS OFF)
this is set by using the GRAPHIC action.

Flag 30 Score flag
Flag 31/32 (LSB/MSB) holds number of turns player has taken (actually this is the number of phrases extracted from the players input).

Flag 33 holds the Verb for the current logical sentence
Flag 34 holds the first Noun in the current logical sentence
Flag 35 holds the Adjective for first Noun
Flag 36 holds the Adverb for the current logical sentence

Flag 37 holds maximum number of objects conveyable (initially 4)
Set using ABILITY action.
Flag 38 holds current location of player

Flag 39 holds current top line of screen
Set by the LINE action.
Flag 40 holds screen mode (range 0 to 4..) set with MODE action.
also Bit 7 - Forces no change of Border
Bit 6 - Produces "More.." when screen fills
Flag 41 holds line number for split (if not in range 4-24 then 12 used) this is set by the PROTECT action to be the current screen line.

Flag 42 holds prompt to use (a system message number - 0 selects

one of four randomly
Set by the PROMPT action.

Flag 43 holds the Preposition in the current logical sentence
Flag 44 holds the second Noun in the current logical sentence
Flag 45 holds the Adjective for the second Noun
Flag 46 holds the current pronoun ("IT" usually) Noun
Flag 47 holds the current pronoun ("IT" usually) Adjective

Flag 48 holds Timeout duration required
Flag 49 holds Timeout Control flags
Bit 7 - Set if timeout occurred last frame
Bit 6 - Set if data available for recall (not of use to writer)
Bit 5 - Set this to cause auto recall of input buffer on timeout
Bit 4 - Set this to print buffer on exit, (for use with Bit 3).
Bit 3 - Set this to take input from lower screen
Bit 2 - Set this so timeout can occur on ANYKEY
Bit 1 - Set this so timeout can occur on "More..."
Bit 0 - Set this so timeout can occur at start of input only
Set using INPUT and TIME (as is flag 48), TIMEOUT tests Bit 7 of this flag.

Flag 50 holds Objno. for DOALL loop. i.e. value following DOALL

Flag 51 holds last object referenced by GET/DROP/WEAR/WHATO etc. This is the number of the currently referenced object as printed in place of any underlines in text.

Flag 52 holds players strength (maximum weight of objects carried and worn - initially 10)
Set with ABILITY action.

Flag 53 holds object print flags
Bit 7 - Set if any object printed as part of LISTOBJ or LISTAT
Bit 6 - Set this to cause continuous object listing i.e. LET 53 64 will make PAW list objects on the same line forming a valid sentence.

Flag 54 holds the present location of the currently referenced object
Flag 55 holds the weight of the currently referenced object
Flag 57 is 128 if the currently referenced object is a container.
Flag 57 is 128 if the currently referenced object is wearable

Flag 58 when set to 128 causes PAW to match words in a sub-process (PARSE usage mainly). Cleared by Process 1/2.
Flag 59 should be avoided as it will be used for any expansion
Flag 60 to 255 are available for your own use.

The CondActs

Conditions:

```
AT          locno                   ;ensure player at specific location
NOTAT       locno                   ;or not
ATGT        locno                   ;higher location than specified
ATLT        locno                   ;lower...

PRESENT     objno                   ;specified object is HERE
ABSENT      objno                   ;or not
WORN        objno                   ;the object is WORN
NOTWORN     objno                   ;or not
CARRIED     objno                   ;etc
NOTCARR     objno
ISAT        objno   locno+          ;tests for an object at a location
ISNOTAT     objno   locno+          ;(or not) other than HERE

ZERO        flagno                  ;the flag contains 0!
NOTZERO     flagno                  ;guess what.... thats right NOT!
EQ          flagno  0-255           ;the flag has a value EQual to
NOTEQ       flagno  0-255           ;NOTEQual
GT          flagno  0-255           ;GreaterThan
LT          flagno  0-255           ;LessThan
SAME        flagno  flagno          ;The SAME as
NOTSAME     flagno  flagno          ;or NOT as the other number/flag

ADJECT1     word                    ;current LS adjective 1 is
ADVERB      word                    ;etc
PREP        word
NOUN2       word
ADJECT2     word

CHANCE      0-99                    ;random possibility of success
TIMEOUT                             ;players last input timed out

QUIT                                ;Are you sure?
```

Actions (Those marked { are type 4, } are type 3, | are type 1)

```
GET     {   objno                   ;GET specified object
DROP    |   objno                   ;what they say...
WEAR    |   objno
REMOVE  }   objno
CREATE      objno
DESTROY     objno
SWAP        objno   objno           ;exchanges position.
PLACE       objno   locno+          ;puts it at the location
PUTO        locno+                  ;puts the current (WHATO) object
PUTIN   {   objno   locno           ;for containers
TAKEOUT }   objno   locno
DROPALL                             ;all fall down...
AUTOG   {                           ;auto versions of above
```

AUTOD			;which do a WHATO etc
AUTOW			
AUTOR			
AUTOP	locno		
AUTOT	locno		
COPYOO	objno	objno	;make 2nd object be with 1st
COPYOF	objno	flagno	;copy position of object to flag
COPYFO	flagno	objno	
WHATO			;convert Noun1(Adjective1) to current object
WEIGH	objno	flagno	;weight of object is put in flag
SET	flagno		;flag becomes 255
CLEAR	flagno		;or 0
PLUS	flagno	0-255	;add value to flag
MINUS	flagno	0-255	;or take away
LET	flagno	0-255	;set to given value
ADD	flagno1	flagno2	;contents of flag1 added to flag2
SUB	flagno1	flagno2	;or subtracted from it
COPYFF	flagno1	flagno2	;duplicated
RANDOM	flagno		;set to random number from 0 to 99
MOVE	flagno		;Adjust contents of flag according to the LS Verb and the Connection table entry for location, that the contents specify. (allows movement in PSIs)
GOTO	locno		;put player at location
WEIGHT	flagno		;weight of objects carried & worn are put in flag
ABILITY	0-255	0-255	;set conveyable objects and strength
MODE	0-4	0-3	
LINE	0-20		;set screen line to split picture
GRAPHIC	0-3		
PROMPT	sysno		;prompt on input. 0 is random
INPUT	0-7		
TIME	0-255	0-7	
PROTECT			;protect text on screen to current print line from scrolling.
PRINT	flagno		;display contents of flag on screen
TURNS			;displays no of turns taken
SCORE			;the score
CLS			;clear the screen
NEWLINE			
MES	mesno		;message without a newline
MESSAGE	mesno		;message with a newline
SYSMESS	sysno		;system message without newline
PICTURE	locno		;Display picture (without CLS)
PAPER	0-9		
INK	0-9		

BORDER	0-7		
CHARSET	0-255		;select character set (if inserted)
SAVEAT			;save current print position
BACKAT			;return to it
PRINTAT	0-20	0-31	;set a new print position
LISTOBJ			;list objects at current location
LISTAT	locno+		;list objects at specified location
INVEN			
DESC			;restart main loop, describe HERE
END			;Type 2,exits table to restart game
DONE			;return to caller, done something
NOTDONE			
OK			;done with SM15
SAVE			
LOAD			
RAMSAVE			
RAMLOAD	flagno		
ANYKEY			
PAUSE	0-255		;delay program for n/50 of a second
PARSE			;convert input string to valid LS
NEWTEXT			;force the loss of remaining phrases
BEEP	0-255	0-255	;duration(1/100sec),pitch(BASIC+60/2)
PROCESS	procno		;execute sub-response/process
DOALL	locno+		;generate Noun(Adjective)1 for each object at Location locno.
RESET	locno		;move player and present objects, reset others to initially at - used to chain games with LOAD
EXTERN	0-255		;call external program

Where:
locno. is a valid location number.
locno+ also allows the use of; 252 (not-created), 253 (worn), 254 (carried) and 255 which is converted into the current location of the player.
mesno. is a valid message.
sysno. is a valid system message.
flagno. is any flag (0 to 255).
procno. is a valid sub-process number.
word; is a word of the required type, which is present in the vocabulary, or "_" which ensures no-word - not an anymatch as normal).

The system messages

The majority of system messages contain ESCC 7s to allow correct spacing on the screen.

SM0 - is used instead of the location description when it is dark.
SM1 - is printed by LISTOBJ if at least one object is present.
SM2 to SM5 - are the four input prompts which are selected randomly unless flag 42 is set to be a valid message number.
SM6 - is produced by the parser when no further phrase can be understood.
SM7 - is produced if no action was carried out (or NOTDONE was) in Response when the Verb is < 14
SM8 - is produced if no action was carried out (or NOTDONE was) in Response when the Verb is > 13
SM9 to SM11 - are printed by action INVEN.
SM12 - printed by QUIT
SM13 and 14 - are printed by the END action.
SM15 - the OK action message.
SM16 - the ANYKEY action message.
SM17 to SM20 - are the TURNS action messages.
SM21 and SM22 - are the SCORE action messages.
SM23 to SM29 - are the first of many messages produced by the object manipulating actions.
SM30 - the positive response expected by END and QUIT.
SM31 - the negative response expected by END and QUIT.
SM32 - produced when a screen full of text has appeared.
SM33 - the input marker.
SM34 - the cursor
SM35 - displayed when a timeout occurs
SM36 to SM45 - are more messages produced by the object manipulating actions.
SM46 - the link between objects when listing continuously
SM47 - the final link between the last two objects when listing
SM48 - the termination of a list of objects (printed by both LISTOBJ and LISTAT, so take care.)
SM49 and SM50 - yet more object messages
SM51 - the termination for a compound sentence on PUTIN/TAKEOUT (and AUTOP/AUTOT)
SM52 - a final object message.
SM53 - message for LISTAT action if no objects found.
SM54 - prompt for filename for game position save/load

SM55 onwards are free to be inserted for your own use. PAW on other machines may use more messages, so bear this in mind if you intend transferring the adventure to another version. E.g. PAW under CPM uses messages 55 to 60!

Graphics Editor Commands

Occasionally when moving the line at speed it will disappear, this is due to a beat frequency of update with screen flyback.

Moving the end of the rubber banded line is achieved with:

```
Q W E        at 1 pixel per move. Alternatively
A   D        use Interface 2/Plus 2 joystick port 2
Z X C        or Kempston™ interface (SYMBOL SHIFT & J)
```

Hold down the CAPS SHIFT key to accelerate to eight pixels per move.

The drawstring editing commands:

```
Cursor Right (CAPS SHIFT & 8) - Start of drawstring
Cursor Down  (CAPS SHIFT & 6) - Next command
Cursor Up    (CAPS SHIFT & 7) - Previous command

DELETE       (CAPS SHIFT & 0) - Delete previous command
GRAPHICS     (CAPS SHIFT & 9) - Delete next command
```

All the following commands require SYMBOL SHIFT to be held down unless otherwise specified.

The toggles:

```
I    Inverse Toggle
O    Over Toggle
Y    Grid Toggle
J    Kempston™ joystick toggle.
```

The next commands all insert into the database:

```
A    Absolute Move (a PLOT with I and O selected)
B    Block in rectangle attribute area. Diagonal defined by line
F    Fill area from end of line. (See footnote)
G    Gosub to location number with scale
X    Ink (produces prompt for Ink selection)
L    Line fix (draws rubber banded line)
C    Paper Select
P    Plot point at end of line
R    Relative Move point to end of line
S    Shade an area of screen from end of line with pattern
V    Flash Select
Z    Bright Select
T    Text character on screen, selects character and set no.
```

Note: Fill and Shade are not completely re-entrant. Fill is a shade with all pixels in pattern set.

ENTER on its own to finish editing session.

The Professional Adventure Writing System

Technical Guide

A Technical guide to:

The Professional Adventure Writer

A graphic adventure writing system for the Sinclair Spectrum computers.

(c) 1986/90 Gilsoft.
Program: T.J.Gilberts, G.Yeandle and P.Wade
Graphics: D. Peeke, K.Maddocks and A.Williams
Manuals: T.J.Gilberts

All Rights reserved. No part of this publication may be copied, loaned, hired or reproduced in any form whatsoever including electronic retrieval systems without the prior written consent of the authors and Gilsoft.

The above notice does not apply to the 'run time' routines appended to, and which form a part of, a saved game which you are free to distribute any way you wish in that form. All we would request is that you credit the use of the Professional Adventure Writer somewhere within the game.

Contents

Overview	Page 5
The Interpreter	Page 6
The CondActs	
Conditions	Page 11
Actions	Page 14
The Database	Page 33
The Editor	Page 38
The Graphics Editor	Page 51
Errors	Page 55

Overview

PAW can be divided into three main functional areas thus:

1/ The Database

The database is a collection of tables organized in a logical manner, which contain all the information to define an adventure game. This database is initially set up to contain a minimum of entries which will probably be needed in every game. There is usually one entry in a table to simplify the operation of the system. The database may utilize the additional sideways RAM pages of a 128K spectrum.

2/ The Editor

This consists of a two level menu system i.e. most functions on the main menu present a sub-menu from which all facilities can be selected. All facilities are set either from a command line as part of the sub-menu option or in response to a prompt. This combination of menu and command line has been found to be most efficient both for the beginner and the more experienced user.

3/ The Interpreter

The real heart of PAW is contained in this section of the program. The next chapter deals with the operation of the interpreter in detail, but it essentially fetches commands from the player and uses the information contained in the database to decode and respond to those commands.

The sheer size of PAW precludes it being held entirely within the main memory area. On a 128K spectrum the majority of the code is held as overlays on a sideways RAM page and transferred into the main area whenever required. A similar system is used on a 48K spectrum by holding three of the overlays in free memory until the area of memory used is overwritten. If the sideways RAM page is initialised on a 128K or the memory area overwritten on a 48K, an attempt to select an option not in the current overlay will result in PAW attempting to load the requisite overlay from cassette (or disc). The main menu and overlay system gives an overhead of approximately 4K which is unavailable to PAW for the database. About 10K is consumed by the interpreter and its workspace (e.g. Ramsave buffer, input and wordwrap buffers etc.). This leaves approximately 27K for the database, plus 5 pages of 16K each, on a 128K spectrum (a total of about 117K). This is maximised by the inclusion of a text compressor which introduces an overhead of 222 bytes (for an expansion dictionary) thus giving about 40% effective text compression. On an average text only game on the 48K spectrum this provides an effective addition of 8K of database space (i.e. a total of 35K!).

The Interpreter

The following description of the interpreter should be read in conjunction with the flowcharts provided overleaf.

Initialise

The background colours and character set are selected - the screen isn't cleared as this always occurs upon describing location zero. The flags are all zeroed except for; flag 37, the number of objects conveyable, which is set to 4; flag 52, the maximum weight of objects conveyable, which is set to 10; flags 46 & 47, the current pronoun ("IT" usually), which are set to 255 (no pronoun) and flag 1 which is set to be the number of objects carried but not worn. Note that clearing the flags has the effect that the game always starts at location zero. This is because flag 38, the current location of the player, is now zero.

Describe Current Location

If flag 2 is non zero it is decremented (reduced by 1). If it is dark (Flag 0 is non zero) and flag 3 is non zero then flag 3 is decremented. If it is dark, flag 4 is non zero and object 0 (the source of light) is absent, then flag 4 is decremented.

The screen is cleared if the current screen mode (contents of flag 40) is not 1.

If it is dark, and Object 0 is absent, then System message 0 (referred to as SM0 - "It's too dark to see anything") is displayed. Otherwise any picture present for the location is drawn and the location description displayed without a NEWLINE.

Search Process Table 1

Flowchart 2 and the next section describe the scanning of a process table. Process table 1 is used mainly to contain the entries to add extra information to the current location description. E.g. details of open doors, objects present etc. It will also contain a PROTECT action, when using screen mode 4, to set the screen line for text to scroll under.

We now enter the main loop of the interpreter which deals with each time frame (i.e. each phrase extracted or each timeout on input which occurs) and the response to players commands.

Search Process Table 2

This contains the main control for PAW's turn at the game. It is used to implement the movements and actions of PSI's, the uncontrolled events such as bridges collapsing, and so on.

Get Phrase

If flags 5 to 8 are non zero they are decremented. If it is dark (Flag 0 is non zero) and flag 9 is non zero then it is decremented. If it is dark and flag 10 is non zero it is decremented if object 0 is absent.

The parser is called to extract a phrase and convert it into a logical sentence - LS. If the input buffer is empty, a new input line is obtained from the player by printing a prompt and making a call to the input routine. The prompt will be the system message whose number is contained in flag 42 - a value of 0 will select one of system messages 2,3,4 or 5 in the ratio 30:30:30:10 respectively.

If the timeout option is selected, by making flag 48 contain a value greater than zero, then the input routine might time out. In this case SM35 ("Time passes...") is displayed and a return made to scan process 2 again.

A phrase is extracted and converted into the logical sentence; by converting any words present, that are in the vocabulary, into their word value and placing them in the required flags.

If no valid phrase can be found then SM6 ("I couldn't understand any of that") is displayed, and a return made to scan process 2.

Search Response Table

Turns (flags 31 and 32), which is the number of valid phrases extracted by the parser, is increased by one. Two flags are used to allow 256 lots of 256 turns (i.e. 65536).

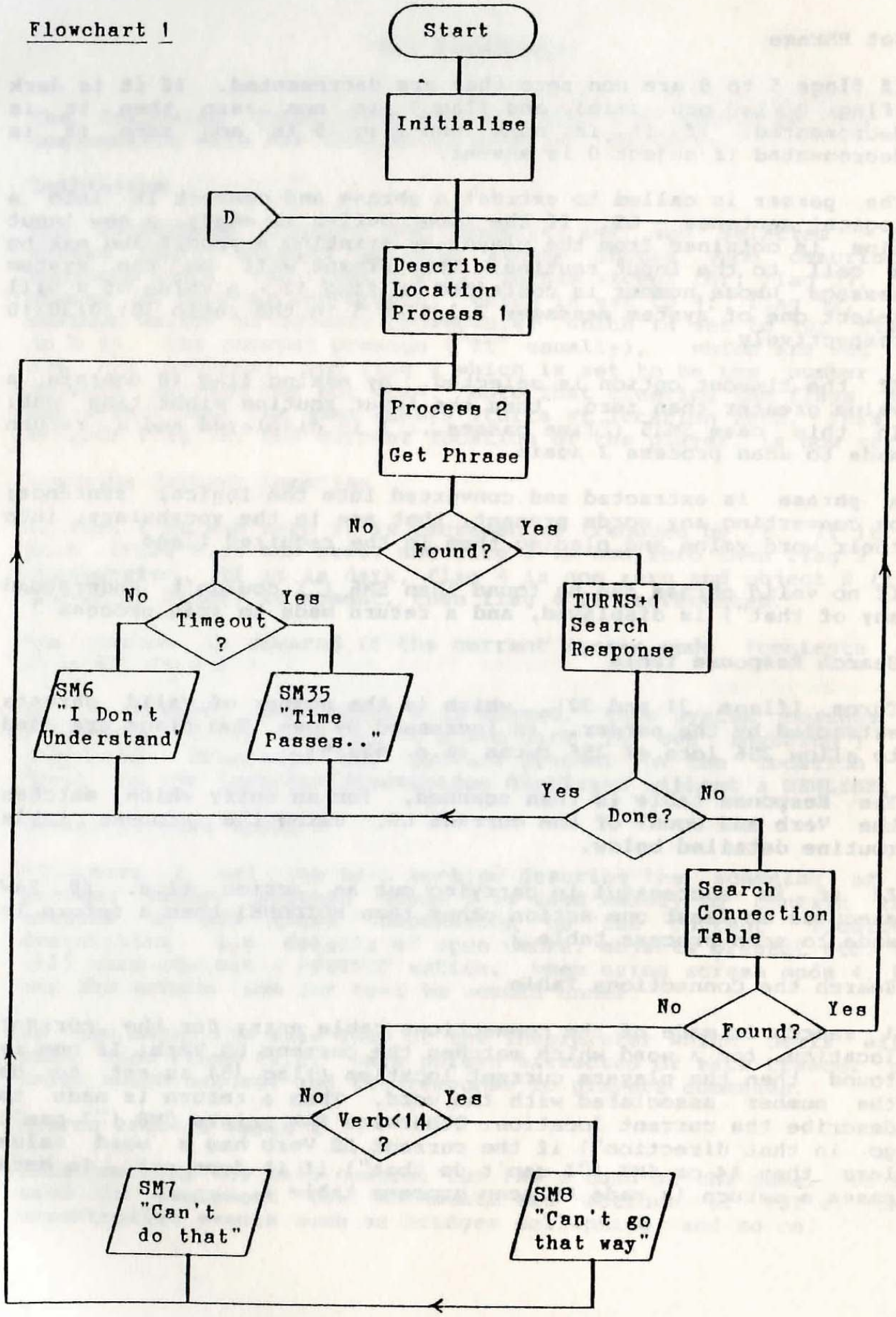
The Response table is then scanned, for an entry which matches the Verb and Noun1 of the current LS, using the process table routine detailed below.

If it is successful in carrying out an action (i.e. If PAW executes at least one action other than NOTDONE) then a return is made to scan process table 2.

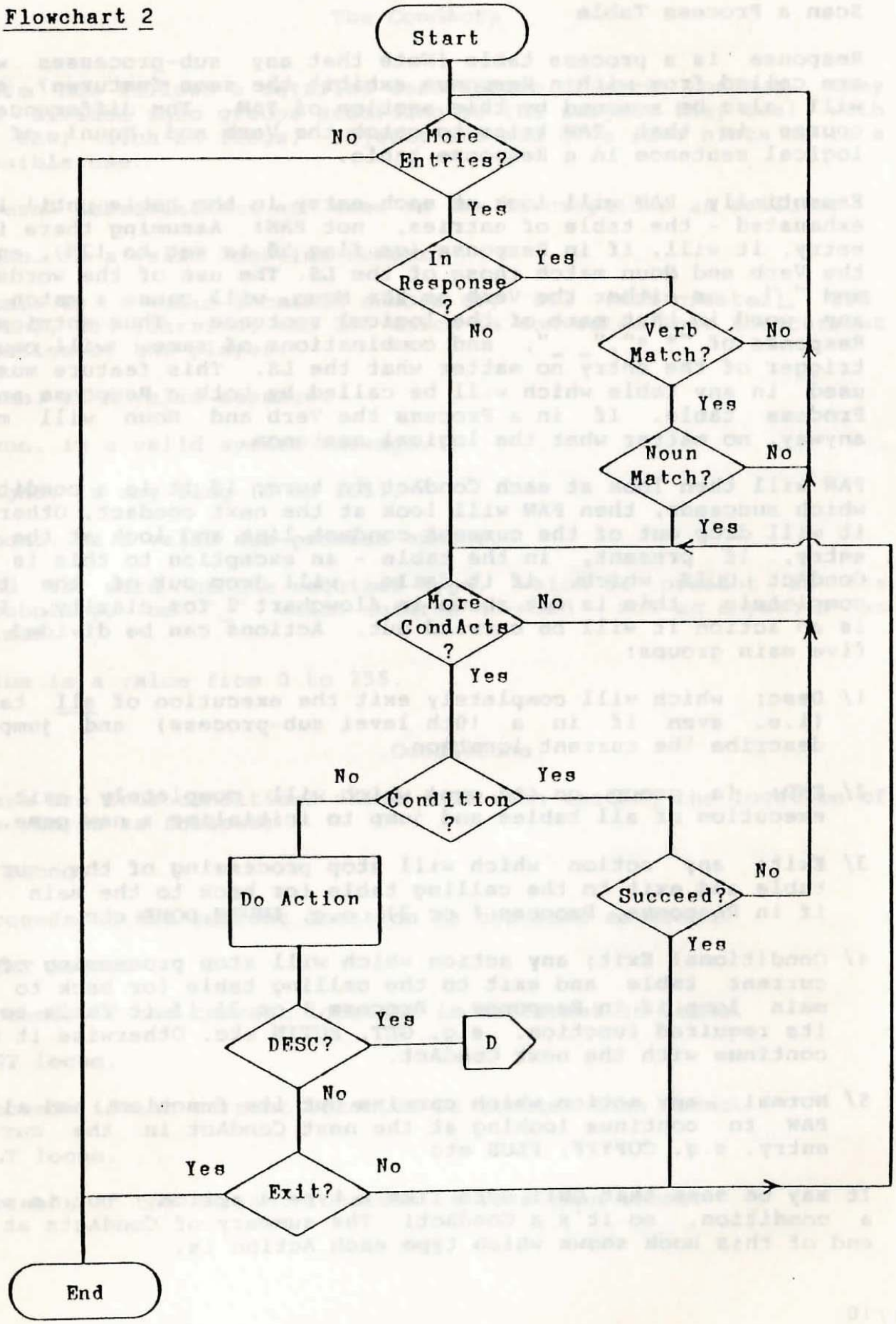
Search the Connections Table

A search is made of the connections table entry for the current location, for a word which matches the current LS Verb. If one is found then the players current location (flag 38) is set to be the number associated with the word. Then a return is made to describe the current location. Otherwise PAW prints SM8 ("I can't go in that direction") if the current LS Verb has a word value less than 14 or SM7 ("I can't do that") if it does not. In both cases a return is made to scan process table 2.

Flowchart 1


Start
Initialise
D
Describe Location Process 1
Process 2
Get Phrase
No
Found?
Yes
No
Timeout ?
Yes
TURNS+1 Search Response
SM6 "I Don't Understand"
SM35 "Time Passes.."
Yes
Done?
No
Search Connection Table
No
Found?
Yes
No
Verb<14 ?
Yes
SM7 "Can't do that"
SM8 "Can't go that way"

Flowchart 2


Start
More Entries?
No
Yes
In Response ?
Yes
No
Verb Match?
No
Yes
Noun Match?
No
Yes
More CondActs ?
No
Yes
Condition ?
No
Yes
Do Action
Succeed?
No
Yes
DESC?
Yes
D
No
Exit?
Yes
No
End

Scan a Process Table

Response is a process table (Note that any sub-processes which are called from within Response exhibit the same features), so it will also be scanned by this section of PAW. The difference of course is that PAW tries to match the Verb and Noun1 of the logical sentence in a Response table.

Essentially PAW will look at each entry in the table until it is exhausted - the table of entries, not PAW! Assuming there is an entry, it will, if in Response (or flag 58 is set to 128), ensure the Verb and Noun match those of the LS. The use of the words "*" and "_", as either the Verb or the Noun, will cause a match with any word in that part of the logical sentence. Thus entries in Response of "* *","_ _", and combinations of same, will cause a trigger of the entry no matter what the LS. This feature must be used in any table which will be called by both a Response and a Process table. If in a Process the Verb and Noun will match anyway, no matter what the logical sentence.

PAW will then look at each CondAct in turn; if it is a condition, which succeeds, then PAW will look at the next condact. Otherwise it will drop out of the current condact list and look at the next entry, if present, in the table - an exception to this is the CondAct QUIT which, if it fails, will drop out of the table completely - this is not shown in flowchart 2 for clarity. If it is an action it will be carried out. Actions can be divided into five main groups:

1/ Desc; which will completely exit the execution of <u>all</u> tables (i.e. even if in a 10th level sub-process) and jump to describe the current location.

2/ END; (a group on its own) which will completely exit the execution of all tables and jump to initialise a new game.

3/ Exit; any action which will stop processing of the current table and exit to the calling table (or back to the main loop if in Response, Process 1 or 2). e.g. INVEN,DONE etc.

4/ Conditional Exit; any action which will stop processing of the current table and exit to the calling table (or back to the main loop if in Response, Process 1 or 2) if it fails to do its required function. e.g. GET, PUTIN etc. Otherwise it will continue with the next CondAct.

5/ Normal; any action which carries out its function, and allows PAW to continue looking at the next CondAct in the current entry. e.g. COPYFF, PLUS etc

It may be seen that QUIT acts like a type 4 action, but is still a condition, so it's a CondAct! The summary of CondActs at the end of this book shows which type each Action is.

The CondActs

There now follows a detailed description of each CondAct. They are divided into groups according to the subject they deal with in PAW; such as flags, objects etc and give some hints as to a possible use.

Several abbreviations are used in the descriptions as follows;

locno. is a valid location number.

locno+ is a valid location number or; 252 (not-created), 253 (worn), 254 (carried) and 255 which is converted into the current location of the player.

mesno. is a valid message.

sysno. is a valid system message.

flagno. is any flag (0 to 255).

procno. is a valid sub-process number.

word is word of the required type, which is present in the vocabulary, or "_" which ensures no-word - not an anymatch as normal.

value is a value from 0 to 255.

Conditions

There are four conditions which deal with testing the location of the player as follows;

AT locno.

Succeeds if the current location is the same as locno.

NOTAT locno.

Succeeds if the current location is different to locno.

ATGT locno.

Succeeds if the current location is greater than locno.

ATLT locno.

Succeeds if the current location is less than locno.

The CondActs

There are eight conditions which deal with the current location of an object;

PRESENT objno.

Succeeds if Object objno. is carried, worn or at the current location.

ABSENT objno.

Succeeds if Object objno. is not carried, not worn and not at the current location.

WORN objno.

Succeeds if object objno. is worn

NOTWORN objno.

Succeeds if Object objno. is not worn.

CARRIED objno.

Succeeds if Object objno. is carried.

NOTCARR objno.

Succeeds if Object objno. is not carried.

ISAT objno. locno+

Succeeds if Object objno. is at Location locno.

ISNOTAT objno. locno+

Succeeds if Object objno. is not at Location locno.

There are eight conditions which deal with the value and comparison of flags;

ZERO flagno.

Succeeds if Flag flagno. is set to zero.

NOTZERO flagno.

Succeeds if Flag flagno. is not set to zero.

EQ flagno. value

Succeeds if Flag flagno. is equal to value.

NOTEQ flagno. value

Succeeds if Flag flagno. is not equal to value.

GT flagno. value

Succeeds if Flag flagno. is greater than value.

LT flagno. value

Succeeds if Flag flagno. is set to less than value.

SAME flagno flagno
1 2

Succeeds if Flag flagno has the same value as Flag flagno .
1 2

NOTSAME flagno flagno
1 2

Succeeds if Flag flagno1 does not have the same value as Flag flagno .
2

There are five conditions to check for an extended logical sentence. It is best to use these conditions only if the specific word (or absence of word using "_") is needed to differentiate a situation. This allows greater flexibility in the commands understood by the entry.

ADJECT1 word

Succeeds if the first noun's adjective in the current LS is word.

ADVERB word

Succeeds if the adverb in the current LS is word.

PREP word

Succeeds if the preposition in the current LS is word.

NOUN2 word

Succeeds if the second noun in the current LS is word.

ADJECT2 word

Succeeds if the second noun's adjective in the current LS is word.

The following condition is for random occurrences. You could use it to provide a chance of a tree falling on the player during a lightning strike or a bridge collapsing etc. Do not abuse this

facility, always allow a player a way of preventing the problem; such as rubber boots for the lightning, or similar.

CHANCE percent

Succeeds if percent is less than or equal to a random number in the range 1-100 (inclusive). Thus a CHANCE 50 condition would allow PAW to look at the next CondAct only if the random number generated was between 1 and 50, a 50% chance of success.

A single condition to test for a timeout;

TIMEOUT

Will succeed if the last request for input from the player was allowed to timeout. For example an entry in Process table 2 of; TIMEOUT MESSAGE 0 (where message 0 read "Come on sleepy") could be created, perhaps with a CHANCE or a count of time outs to make it less monotonous!

The true CondAct;

QUIT

SM12 ("Are you sure?") is printed and the input routine called. Will succeed if the player replies with the first letter of SM30 ("Y") to the prompt. If not then Actions NEWTEXT and DONE are performed.

Actions

There are nineteen actions to deal with the manipulation of object positions;

GET objno.

If Object objno. is worn or carried, SM25 ("I already have the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is not at the current location, SM26 ("There isn't one of those here.") is printed and actions NEWTEXT DONE are performed.

If the total weight of the objects carried and worn by the player plus Object objno. would exceed the maximum conveyable weight (Flag 52) then SM43 ("The _ weighs too much for me.") is printed and actions NEWTEXT DONE are performed.

If the maximum number of objects is being carried (Flag 1 is

greater than, or the same as, Flag 37), SM27 ("I can't carry any more things.") is printed and actions NEWTEXT DONE are performed. In addition any current DOALL loop is cancelled.

Otherwise the position of Object objno. is changed to carried, Flag 1 is incremented and SM36 ("I now have the _.") is printed.

DROP objno.

If Object objno. is worn then SM24 ("I can't. I'm wearing the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is at the current location (but neither worn nor carried), SM49 ("I don't have the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is not at the current location then SM28 ("I don't have one of those.") is printed and actions NEWTEXT DONE are performed.

Otherwise the position of Object objno. is changed to the current location, Flag 1 is decremented and SM39 ("I've dropped the _.") is printed.

WEAR objno.

If Object objno. is at the current location (but not carried or worn) SM49 ("I don't have the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is worn, SM29 ("I'm already wearing the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is not carried, SM28 ("I don't have one of those.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is not flagged as Wearable (WR option in the object weight menu) then SM40 ("I can't wear the _.") is printed and actions NEWTEXT DONE are performed.

Otherwise the position of Object objno. is changed to worn, Flag 1 is decremented and SM37 ("I'm now wearing the _.") is printed.

REMOVE objno.

If Object objno. is carried or at the current location (but not worn) then SM50 ("I'm not wearing the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is not at the current location, SM23 ("I'm not wearing one of those.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is not flagged as wearable (and thus removeable) then SM41 ("I can't remove the _.") is printed and actions NEWTEXT DONE are performed.

If the maximum number of objects is being carried (Flag 1 is greater than, or the same as, Flag 37), SM42 ("I can't remove the _. My hands are full.") is printed and actions NEWTEXT DONE are performed.

Otherwise the position of Object objno. is changed to carried. Flag 1 is incremented and SM38 ("I've removed the _.") printed.

CREATE objno.

The position of Object objno. is changed to the current location and Flag 1 is decremented if the object was carried.

DESTROY objno.

The position of Object objno. is changed to not-created and Flag 1 is decremented if the object was carried.

SWAP objno objno
1 2

The positions of the two objects are exchanged. Flag 1 is not adjusted. The currently referenced object is set to be Object objno .
2

PLACE objno. locno+

The position of Object objno. is changed to Location locno. Flag 1 is decremented if the object was carried. It is incremented if the object is placed at location 254 (carried).

PUTO locno+

The position of the currently referenced object (i.e. that object whose number is given in flag 51), is changed to be Location locno. Flag 54 remains its old location. Flag 1 is decremented if the object was carried. It is incremented if the object is placed at location 254 (carried).

PUTIN objno. locno.

If Object locno. does not exist or is not flagged as a container (option C on the object weight menu) then a run time error is generated of "Illegal Argument".

If Object objno. is worn then SM24 ("I can't. I'm wearing the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is at the current location (but neither worn nor carried), SM49 ("I don't have the _.") is printed and actions

NEWTEXT DONE are performed.

If Object objno. is not at the current location, but not carried, then SM28 ("I don't have one of those.") is printed and actions NEWTEXT DONE are performed.

Otherwise the position of Object objno. is changed to Location locno. Flag 1 is decremented and SM44 ("The _ is in the"), a description of Object locno. and SM51 (".") is printed.

TAKEOUT objno. locno.

If Object locno. does not exist or is not flagged as a container (option C on the object weight menu) then a run time error is generated of "Illegal Argument".

If Object objno. is worn or carried, SM25 ("I already have the _.") is printed and actions NEWTEXT DONE are performed.

If Object objno. is at the current location, SM45 ("The _ isn't in the"), a description of Object locno. and SM51 (".") is printed and actions NEWTEXT DONE are performed.

If Object objno. is not at the current location and not at Location locno. then SM52 ("There isn't one of those in the"), a description of Object locno. and SM51 (".") is printed and actions NEWTEXT DONE are performed.

If Object locno. is not carried or worn, and the total weight of the objects carried and worn by the player plus Object objno. would exceed the maximum conveyable weight (Flag 52) then SM43 ("The _ weighs too much for me.") is printed and actions NEWTEXT DONE are performed.

If the maximum number of objects is being carried (Flag 1 is greater than, or the same as, Flag 37), SM27 ("I can't carry any more things.") is printed and actions NEWTEXT DONE are performed. In addition any current DOALL loop is cancelled.

Otherwise the position of Object objno. is changed to carried, Flag 1 is incremented and SM36 ("I now have the _.") is printed.

Note: No check is made, by either PUTIN or TAKEOUT, that Object locno. is actually present. This must be carried out by you if required.

DROPALL

All objects which are carried or worn are created at the current location (i.e. all objects are dropped) and Flag 1 is set to 0. This is included for compatibility with older writing systems. Note that a DOALL 254 will carry out a true DROP ALL, taking care of any special actions included.

The next six actions are automatic versions of GET, DROP, WEAR, REMOVE, PUTIN and TAKEOUT. They are automatic in that instead of needing to specify the object number, they each convert Noun(Adjective)1 into the currently referenced object - by searching the object word table. The search is for an object which is at one of several locations in descending order of priority - see individual descriptions. This search against priority allows PAW to 'know' which object is implied if more than one object with the same Noun description (when the player has not specified an adjective) exists; at the current location, carried or worn - and in the container in the case of TAKEOUT.

AUTOG

A search for the object number represented by Noun(Adjective)1 is made in object word in order of location priority; here, carried, worn. i.e. The player is more likely to be trying to GET an object that is at the current location than one that is carried or worn. If an object is found its number is passed to the GET action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM26 ("There isn't one of those here.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT DONE are performed

AUTOD

A search for the object number represented by Noun(Adjective)1 is made in object word in order of location priority; carried, worn, here. i.e. The player is more likely to be trying to DROP a carried object than one that is worn or here. If an object is found its number is passed to the DROP action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM28 ("I don't have one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT DONE are performed

AUTOW

A search for the object number represented by Noun(Adjective)1 is made in object word in order of location priority; carried, worn, here. i.e. The player is more likely to be trying to WEAR a carried object than one that is worn or here. If an object is found its number is passed to the WEAR action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM28 ("I don't have one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT DONE are performed

AUTOR

A search for the object number represented by Noun(Adjective)1 is made in object word in order of location priority; worn, carried, here. i.e. The player is more likely to be trying to REMOVE a worn object than one that is carried or here. If an object is found its number is passed to the REMOVE action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM23 ("I'm not wearing one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT DONE are performed

AUTOP locno.

A search for the object number represented by Noun(Adjective)1 is made in object word in order of location priority; carried, worn, here. i.e. The player is more likely to be trying to PUT a carried object inside another than one that is worn or here. If an object is found its number is passed to the PUTIN action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM28 ("I don't have one of those.") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT DONE are performed

AUTOT locno.

A search for the object number represented by Noun(Adjective)1 is made in object word in order of location priority; in container, carried, worn, here. i.e. The player is more likely to be trying to get an object out of a container which is actually in there than one that is carried, worn or here. If an object is found its number is passed to the TAKEOUT action. Otherwise if there is an object in existence anywhere in the game or if Noun1 was not in the vocabulary then SM52 ("There isn't one of those in the"), a description of Object locno. and SM51 (".") is printed. Else SM8 ("I can't do that.") is printed (i.e. It is not a valid object but does exist in the game). Either way actions NEWTEXT DONE are performed

Note: No check is made, by either AUTOP or AUTOT, that Object locno. is actually present. This must be carried out by you - if required.

COPYOO objno objno
1 2

The position of Object objno2 is set to be the same as the position of Object Objno1. The currently referenced object is set to be Object objno .
2

There are four actions which allow various parameters of objects to be; placed in flags, set from flags - for comparison or manipulation.

COPYOF objno. flagno.

The position of Object objno. is copied into Flag flagno. This could be used to examine the location of an object in a comparison with another flag value. e.g. COPYOF 1 11 SAME 11 38 could be used to check that object 1 was at the same location as the player - although ISAT 1 255 would be better!

COPYFO flagno. objno.

The position of Object objno. is set to be the contents of Flag flagno. An attempt to copy from a flag containing 255 will result in a run time error of "Invalid Argument". Setting an object to an invalid location will still be accepted as it presents no danger to the operation of PAW.

WHATO

A search for the object number represented by Noun(Adjective)1 is made in object word in order of location priority; carried, worn, here. This is because it is assumed any use of WHATO will be related to carried objects rather than any that are worn or here. If an object is found its number is placed in flag 51, along with the standard current object parameters in flags 54-57. This allows you to create other auto actions (the tutorial gives an example of this for dropping objects in the tree).

WEIGH objno. flagno.

The true weight of Object objno. is calculated (i.e. if it is a container, any objects inside have their weight added - don't forget that nested containers stop adding their contents after ten levels) and the value is placed in Flag flagno. This will have a maximum value of 255 which will not be exceeded. If Object objno. is a container of zero weight, Flag flagno. will be cleared as objects in zero weight containers, also weigh zero!

Now ten actions to manipulate the flags;

SET flagno.

Flag flagno. is set to 255.

CLEAR flagno.

Flag flagno. is cleared to 0.

LET flagno. value

Flag flagno. is set to value.

PLUS flagno. value

Flag flagno. is increased by value. If the result exceeds 255 the flag is set to 255.

MINUS flagno. value

Flag flagno. is decreased by value. If the result is negative the flag is set to 0.

ADD flagno flagno
1 2

Flag flagno2 has the contents of Flag flagno1 added to it. If the result exceeds 255 the flag is set to 255.

SUB flagno flagno
1 2

Flag flagno2 has the contents of Flag flagno1 subtracted from it. If the result is negative the flag is set to 0.

COPYFF flagno flagno
1 2

The contents of Flag flagno is copied to Flag flagno.
1 2

RANDOM flagno.

Flag flagno. is set to a number from the Pseudo-random sequence from 1 to 100. This could be useful to allow random decisions to be made in a more flexible way than with the CHANCE condition.

MOVE flagno.

This is a very powerful action designed to manipulate PSI's. It allows the current LS Verb to be used to scan the connections table for the location given in Flag flagno. If the Verb is found then Flag flagno. is changed to be the location number associated with it, and the next condact is considered. If the verb is not found, or the original location number was invalid, then PAW considers the next entry in the table - if present. Thus you could consider that PAW carries out the following imaginary entries on exit from Response if no action has been done;

```
_   _   MOVE     38        ;Attempt to move player
        DESC               ;Describe his new loc.

_   _   LT       33  14    ;Movement word?
        SYSMESS   7        ;"Can't go that way.."
        DONE
```

```
-    -    SYSMESS  8              ;"I can't do that"
```

This feature could be used to provide characters with Random movement in valid directions; by setting the LS Verb to a random movement word and allowing MOVE to decide if the character can go that way. Note that any special movements which are dealt with in Response for the player, must be dealt with separately for a PSI as well.

Three actions to manipulate the flags dealing with the player;

GOTO locno.

Changes the current location to locno. This effectively sets flag 38 to the value locno.

WEIGHT flagno.

Calculates the true weight of all objects carried and worn by the player (i.e. any containers will have the weight of their contents added upto a maximum of 255), this value is then placed in Flag flagno. This would be useful to ensure the player was not carrying too much weight to cross a bridge without it collapsing etc.

ABILITY value value
1 2

This sets Flag 37, the maximum number of objects conveyable, to value1 and Flag 52, the maximum weight of objects the player may carry and wear at any one time (or their strength), to be value2. No checks are made to ensure that the player is not already carrying more than the maximum. GET and so on, which check the values, will still work correctly and prevent the player carrying any more objects, even if you set the value lower than that which is already carried!

There are seven actions which deal with the manipulation of the flags for screen mode, format and input etc;

MODE mode option

There are five screen modes each controlled by Flag 40 and set using the MODE action thus:-

Mode 0

If a picture (other than 'subroutine') is present in the database, the screen is cleared to the default colours and the picture drawn. Once a key is pressed the original colours are restored, and the location is described.

Mode 1

Text Only. Full screen scroll. Text is output to the screen continuously. CLS is not active on describe.

Mode 2

If a picture (other than 'subroutine') is present in the database, the screen is cleared to the default colours and the picture drawn. The original colours are restored, the area below the line given in Flag 41 is cleared and text printed from the same line.

Mode 3

As Mode 2 above but the picture does not scroll away.

Mode 4

A text only mode which protects the location description as if it were a picture, all other text scrolls under. You will probably need a PROTECT action at a suitable point in Process 1 (e.g. after LISTOBJ?) as otherwise only as far as the last but one line of the text will be protected.

In all the graphic modes above; option 2 forces the border NOT to be set to the Paper colour - it's best to select this on modes 2 & 3 as the border is reset as soon as the picture is fully drawn causing an annoying flicker. Option 1 makes SM32 ("More...") appear when the screen area fills (unlike 'Scroll?' you cannot break out - well, in the editor you can!). Option 3 of course provides both options and option 0 neither! e.g. MODE 4 1 will select a fixed text window with a "More..." prompt.

LINE lineno.

This sets the line that text should be printed from in the split screen graphic modes, and in the case of mode 2 the top line of the area to be cleared. Be careful when using a lower screen input (see INPUT action) to allow at least four spare lines in the upper screen (not forgetting to account for the number of lines used by the prompt and marker) or a runtime error will occur if a full input line is entered.

GRAPHIC option

This allows the way in which pictures are dealt with by PAW to be modified, there are three valid options;

0 - Normal, any picture present for a location is drawn when the player first visits that location, and then only if pictures are turned ON or a temporary redraw is requested.

1 - OFF, any pictures are completely ignored by PAW, even if a temporary redraw is requested

2 - ON, any picture present for a location is drawn every time the player visits that location.

There is an option 3 (option = 2+1) but pictures OFF takes priority anyway, so it is redundant. The current picture for a location can be redrawn by adding 128 to Flag 29, this is a temporary redraw, which can be used for when the player requests the picture in Normal mode. i.e. PLUS 29 128 DESC would be the required entry in Response.

PROMPT sysno.

Causes System message sysno. to be displayed whenever PAW obtains a command line from the player. A value of 0 (default) will cause PAW to select one of SM2,SM3,SM4 or SM5 in the ratio 30:30:30:10 respectively. Note this does not affect the prompts displayed by the END or QUIT condacts.

TIME duration option

Allows input to be set to 'timeout' after a specific duration in 1.28 second intervals, i.e. the Process 2 table will be called again if the player types nothing for the specified period. 'option' allows this to also occur on ANYKEY and the "More..." prompt. In order to calculate the number to use for the option just add the numbers shown next to each item to achieve the required combination;

1 - While waiting for first character of Input only.
2 - While waiting for the key on the "More..." prompt.
4 - While waiting for the key on the ANYKEY action.

e.g. TIME 5 6 (option = 2+4) will allow 6.4 seconds of inactivity on behalf of the player on input, ANYKEY or "More..." and between each keypress. Whereas TIME 5 3 (option = 1+2) allows it only on the first character of input and on "More...".

TIME 0 0 will stop timeouts (default).

INPUT option

This action allows the way the input routine operates to be changed by selecting options. Some combinations are of no use but are a by product of the way PAW works. To calculate the number to use for the option just add the numbers shown next to each item to achieve the required combination;

1 - will cause the input prompt, marker and cursor to be displayed in the lower screen area. This is the preferred input mode for time out games, where timeout

can occur between keypresses, as the input line is not left partially printed if a timeout occurs part way through.

2 - will cause PAW to print a copy of the input line when ENTER is pressed, this is for use mainly with item 1 to allow the final input line to be displayed on screen so that the player knows what has been typed.

4 - This will cause the reprint of any text input so far, when input is resumed after a timeout. Again mainly for use with item 1 when timeout can occur between keypresses.

A default of INPUT 0 (no options) is assumed by PAW.

PROTECT

Used in conjunction with MODE 4 mainly. This action sets the current print line as the top of the scrolling text window. It should only be used within Process 1 (it will be allowed elsewhere but has no meaning) after any text you want protected has been printed. This is usually before/after a LISTOBJ action as objects are usually the last thing printed.

It also has uses in other modes to allow say a picture and part of the location description to be protected, with the remainder of the text scrolling underneath!

Three actions to deal with the printing of flag values on the screen;

PRINT flagno.

The decimal contents of Flag flagno. are displayed in the current temporary colours without leading or trailing spaces. This is a very useful action. Say flag 100 contained the number of coins carried by the player, then an entry in a process table of MES 10 ("You have ") PRINT 100 MESSAGE 11 (" gold coins."), could be used to display this to the player.

TURNS

SM17-20 "You have taken x turn(s)." is printed where x is Flag 31 + 256 * Flag 32.

SCORE

SM21-22 "You have scored x%" is printed where x is Flag 30.

The CondActs

Thirteen(!) actions to deal with screen output and control;

CLS

Clears the screen to current background colours. Also clears the current print position and SAVEAT position to 0,0.

NEWLINE

Prints spaces to the end of the current line and then resets the colours & character set to the current background colours.

MES mesno.

Prints Message mesno. in the current temporary colours.

MESSAGE mesno.

Prints Message mesno. in the current temporary colours, then carries out a NEWLINE action.

SYSMESS sysno.

Prints System Message sysno. in the current temporary colours.

PICTURE locno.

Draws graphics picture locno. regardless of whether it is a subroutine or main picture. Note: the screen and colours are not cleared first as with describe location. This could be used to add (or take away if you use PRINTAT and MES to remove with spaces) parts of a picture. The sample game TEWK uses this feature several times. Note that a picture will be drawn from the last point used by the previous picture drawn!

PAPER n

Where n ranges from 0 to 9. Changes background paper colour.

INK n

Where n ranges from 0 to 9. Changes background ink colour.

BORDER n

Where n range from 0 to 7. Changes screen border colour.

CHARSET value

Changes main character set to value given if valid, otherwise no action is taken.

The next three screen control actions allow a section of text to be printed on the screen away from the current print position. This could be used to provide an information line on top of the current picture, or a section of text in a small window on the picture etc etc. TEWK uses this facility to provide its drop down window inventory in response to the STATUS command.

SAVEAT

Flushes the current wordwrap buffer thus restoring the current background colours. Then saves the current print position, overwriting any previously saved position.

BACKAT

Flushes the current wordwrap buffer thus restoring the current background colours. Then restores the print position last saved by SAVEAT.

PRINTAT lineno. colno.

The current print position is changed to the specified value. Note this will also flush the wordwrap buffer and restore the temporary colours to the background colours.

Three actions dealing with listing objects on the screen. The first two are controlled by/set the value of flag 53 as described in the chapter on objects.

LISTOBJ

If any objects are present then SM1 ("I can also see:") is printed, followed by a list of all objects present at the current location. If there are no objects then nothing (as in null, not the word!) is printed.

LISTAT locno+

If any objects are present then they are listed. Otherwise SM53 ("nothing.") is printed - note that you will usually have to precede this action with a message along the lines of "In the bag is;" etc. It would be possible to create an alternative to the INVEN action described next by using 253 & 254 as parameters for LISTAT.

INVEN

This action is not affected by the continuous object list flag for compatability with older writing systems.

SM9 ("I have with me:-") is printed. If no objects are carried or worn SM11 ("Nothing at all.") is printed. Otherwise the object

text for each object that is carried or worn is printed on a separate line. If an object is worn its object text is followed by SM10 (" (worn)"), right aligned on the next line if it will not fit on the same one. Action DONE is then performed.

The two actions which completely exit Response/Process execution;

DESC

Will cancel any DOALL loop, any sub-process calls and make a jump to describe the current location.

END

SM13 ("Would you like to play again?") is printed and the input routine called. Any DOALL loop and sub-process calls are cancelled. If the reply does not start with the first character of SM31 a jump is made to Initialise. Otherwise a jump is made to the Editor (if it is present) or to the BASIC NEW command.

Three exit table actions;

DONE

This action jumps to the end of the process table and flags to PAW that an action has been carried out. i.e. no more condacts or entries are considered. A return will thus be made to the previous calling process table, or to the start point of any active DOALL loop.

NOTDONE

This action jumps to the end of the process table and flags to PAW that no action has been carried out. i.e. no more condacts or entries are considered. A return will thus be made to the previous calling process table or to the start point of any active DOALL loop. This will cause PAW to print one of the "I can't" messages if needed. i.e. if no other action is carried out and no entry is present in connections for the current Verb.

OK

SM15 ("OK") is printed and action DONE is performed.

Four actions to allow the current state of the game to be saved and restored;

SAVE

SM54 is printed to request a filename, and the input line presented. The standard start tape message for the spectrum is printed at the bottom of the screen. When a key is pressed the game position is saved to tape, then action DESC is performed. If BREAK is pressed during the save a jump is made to Initialise. The save includes all information required to allow the restoration of the game to the exact same state as it was before the save, including the values of flags, positions of objects, picture drawn flags etc.

LOAD

A filename is obtained using SM54 and the standard input line. The named file is serched for on the tape and the data loaded - which should be a game position. The action DESC is performed. If BREAK is pressed during the load or a tape error is detected a jump is made to Initialise. If data is loaded which is not a game position, a tape error will normally be detected.

RAMSAVE

In a similar way to SAVE this action saves all the information relevant to the game in progress not onto tape but into a memory buffer. This buffer is of course volatile and will be destroyed when the machine is turned off which should be made clear to the player. In addition it will also be cleared when you return to the editor section of PAW - in case you then change the design of the game!

RAMLOAD flagno.

This action is the counterpart of RAMSAVE and allows the saved buffer to be restored. The parameter specifies the last flag to be reloaded which can be used to preserve values over a restore, for example an entry of:

```
RAMLO _     COPYFF  30  255
            RAMLOAD 254
            COPYFF  255  30
            DESC
```

could be used to maintain the current score, so that the player can not use RAMSAVE/LOAD as an easy option for achieving 100%!

Note: unlike SAVE and LOAD the RAM actions allow the next Condact to be carried out. They should normally always be followed by a DESC in order that the game state is restored to an identical position.

The actions could be used to implement an OOPS command that is common on other systems to take back the previous move; by creating an entry in Process 2 (or Response) which does an automatic RAMSAVE every time the player enters a move.

Two actions to allow the game to be paused for a time or until a key is pressed;

ANYKEY

SM16 ("Press any key to continue") is printed at the bottom of the screen and the keyboard is scanned until a key is pressed or until the timeout duration has elapsed if enabled.

PAUSE value

Pauses for value/50 secs. However, if value is zero then the pause is for 256/50 secs. Note that the keyboard is disabled for the duration of the pause.

Two actions to deal with control of the parser;

PARSE

This action was designed to deal with speech to PSIs. Any string (i.e. a further phrase enclosed in quotes [""]) that was present in the players current phrase is converted into a LS - overwriting the existing LS formed originally for that phrase. If no phrase is present, or it is invalid, then PAW will look at the next condact, the text is not marked as illegal by a NEWTEXT action as you might expect, so you must explicitly include one if required. Otherwise the next entry is considered with the new LS of the speech made to the PSI. Because it overwrites the current LS it must be used only in a sub-response table, the table will have the form of:

```
*     *     PARSE                ;Always do this entry
            MESSAGE    x         ;"They don't understand"
            DONE

word  word  CondAct list         ;Any phrases PSI understands

_     _     MESSAGE    x         ;as above or different message
```

there will be two or more calling entries which will be similar to:

```
SAY   name  SAME     pos  38     ;Are they here?
            PROCESS    y         ;Decode speech..
            DONE                 ;LS destroyed so always DONE.
```

```
SAY    name  MESSAGE   z          ;"They are not here!"
             DONE
```

See the Notebook for details on using PARSE to deal with multiple statements in a string.

NEWTEXT

Forces the loss of any remaining phrases on the current input line. You would use this to prevent the player continuing without a fresh input should something go badly for his situation. e.g. the GET action carries out a NEWTEXT if it fails to get the required object for any reason, to prevent disaster with a sentence such as:

GET SWORD AND KILL ORC WITH IT

as attacking the ORC without the sword may be dangerous!

One action to deal with sound

BEEP duration pitch

Both duration and pitch may range from 0 to 255. Duration is in units of one hundredth of a second. The value of pitch is obtained by taking the value you would use in a BASIC BEEP command, adding 60 then dividing by 2.

Several actions which are more difficult to classify;

PROCESS procno.

This powerful action transfers the attention of PAW to the specified Process table number. This sub-process will exhibit the same features as the table which called it. i.e. if called by Response, PAW will match the Verb and Noun1 of the LS against the word entries as with the main table. Note that it is a true subroutine call and any exit from the new table (e.g. DONE, OK etc) will return control to the condact which follows the calling PROCESS action. A sub-process can call (nest) further process' to a depth of 10 at which point a run time error of "Limit reached" will be generated.

DOALL locno+

Another powerful action which allows the implementation of an 'ALL' type command.

1 - An attempt is made to find an object at Location locno. If this is unsuccessful the DOALL is cancelled and

action DONE is performed.

2 - The object number is converted into the LS Noun1 (and Adjective1 if present) by reference to the object word table. If Noun(Adjective)1 matches Noun(Adjective)2 then a return is made to step 1. This implements the "Verb ALL EXCEPT object" facility of the parser.

3 - The next condact and/or entry in the table is then considered. This effectively converts a phrase of "Verb All" into "Verb object" which is then processed by the table as if the player had typed it in.

4 - When an attempt is made to exit the current table, if the DOALL is still active (i.e. has not been canceled by an action) then the attention of PAW is returned to the DOALL as from step 1; with the object search continuing from the next highest object number to that just considered.

The main ramification of the search method through the object word table is; objects which have the Same Noun(Adjective) description (where the game works out which object is refered to by its presence) must be checked for in ascending order of object number, or one of them may be missed.

Use of DOALL to implement things like OPEN ALL must account for the fact that doors are often flags only and would have to be made into objects if they were to be included in a DOALL.

RESET locno+

This action is designed to allow the implementation of multi-part games where the objects which are not carried forward are reset to their starting location.

All objects which can be carried between parts must be present (with the same description) in each part. Any others may be reused within each part at will.

Any objects which are present at the current location are moved to Location locno. And the current location is set to be locno. Any other objects are set to the locations given by the Initially At table. No effect on flags. Action DESC is performed when complete.

The suggested method of its use is given in the chapter on multi-part games in the Notebook.

EXTERN value

Calls an external routine with parameter value. This command is dealt with in a section in the Notebook as its use is specialised, and not intended for the beginner.

Detailed Description of the Database

The database consists of a number of inter-related tables and also contains an area of miscellaneous information e.g. values of background colours, number of objects conveyable, character sets etc. On a 128K machine the main data areas and pointers are present for each RAM page. Effectively each page is an independent database. The tables present (in the order they appear in memory) are:-

UDG's

This table, which is 152 bytes long, contains the User Defined Graphics. The UDG's can be changed with the character editor described later. Codes 144-162.

Shades

This table is 128 bytes long and contains the graphics Shade patterns. These can be changed with the character editor as described later. Codes 0-15.

The above two tables are saved/loaded as a pair and are termed character set 0. (although the ROM set provides codes 32-127).

Miscellaneous

Several small tables and values, 50 bytes in total.

Hunks

Any misc. data blocks inserted by user overlays. May be empty.

Character Sets

This table is empty until a character set is inserted using the character editor. 768 bytes are used then by each character set inserted.

Dictionary

This table contains only one byte until the compressor is used for the first time on the database. After that it contains 222 bytes of expansion dictionary to allow the tokens 165-255 to be converted into the letter groupings they represent.

The Process tables

These tables form the heart of the database providing the main game control.

The Database

The Response Table

Each entry contains the word values of the Verb and Noun for the LS the entry is to deal with followed by any number of condacts. When the adventure is played, if there is an entry in the table which matches the Verb and Noun1 of the LS entered then the condacts are performed. The condacts that may be present and the effect that they have is fully specified in the description of the Interpreter. The order of entries in the table is in ascending order of the Verb value. Entries which have the same Verb value are held in ascending order of the Noun value. Entries with the same Verb and Noun value are held in the order they were inserted into the database. The word " _ " has a word of value 255 while the word "*" has a word value of 1. An example of the order of the table, with word value shown in brackets, is as follows:-

*	(1)	*	(1)
LOOK	(30)	UP	(10)
LOOK	(30)	DOWN	(11)
LOOK	(30)	_	(255)
GET	(100)	*	(1)
GET	(100)	KEY	(16)
GET	(100)	LAMP	(26)
GET	(100)	LAMP	(26)
_	(255)	_	(255)

The other process tables are held in the same format, but the words are ignored by PAW when they are scanned, unless the process is called from within Response using a PROCESS action.

Process 1

Is scanned by PAW after a location is described, to allow any additional information which forms a part of the location description to be displayed.

Process 2

Is scanned by PAW every time frame. That is after every phrase extracted from the player's input, or after every timeout on input.

Any further sub-processes follow the initial three when begun on the Process option of the main menu.

The Verb and Noun used for each entry in Process 1 and 2 (and any sub-processes called from them) have no meaning as they are ignored but can be used to provide a note as to the function of that entry within the table.

Every process table has an overhead of seven bytes (21 are used in a null database as three - including Response - are already

present.). Each entry subsequently uses 5 bytes and each condact uses 1, 2 or 3 bytes depending on the number of parameters.

Note: If a word is deleted from the vocabulary, and no synonyms of it are present, then all entries in the process tables (including Response) which contain that word, are also deleted.

The Object Text table

This table, which has an entry for each object, contains the text which is printed when an object is described. Each entry uses 3 bytes plus the length of the text. An object is anything in the adventure which may be manipulated and objects are numbered from 0 upwards. Object 0 is assumed by the Interpreter to be a source of light. Whenever a new object text is inserted an entry of not-created is made for that object in the Initially at table, an entry of "_ _" in object word and a weight of 1 unit without a container or wear/remove attribute is inserted into the object weight table

The Location Text table

This table, which has an entry for each location, contains the text which is printed when a location is described. Each entry uses 3 bytes plus the length of the text. The entries are numbered from 0 upwards and location 0 is the location at which the adventure starts. Whenever a new location is inserted a null entry for that location is also made in the connections table.

The Message Text table

This table contains the text of any messages which are needed for the adventure. The messages are numbered from 0 upwards and each one uses 3 bytes plus the length of the text.

The System Messages

This table contains the messages used by the Interpreter. Each entry uses 3 bytes plus the length of the text. The description of the Interpreter shows when these messages are used. In addition extra messages can be inserted by the writer to provide messages for the game if so required.

The Connections table

This table has an entry for each location and each entry may either be empty (null) or contain a number of 'movement pairs'. A movement pair consists of a word value of a Verb (or conversion Noun) from the vocabulary followed by a location number. This means that any Verb (or conversion Noun) with that word value causes movement to that location. A typical entry could be SOUTH 6 EAST 7 LEAVE 6 NORTH 5 which means that SOUTH or LEAVE or their synonyms cause movement to location 6, EAST or it's synonyms to location 7 and NORTH or it's synonyms to location 5. Each entry

uses 3 bytes plus 2 bytes for each movement pair.

Note 1. The movement pairs contain the word value not the actual word. If a word value is deleted from the vocabulary then all movement pairs which contain that word value are also deleted.

Note 2. When the adventure is being played it is only the LS Verb which will cause movement.

Note 3. If a movement is performed by an entry in the Response table using the GOTO action, then it may not be needed in the Connections table, unless that entry is required for a PSI who can move unconditionaly.

The Vocabulary

Each entry in the table uses 7 bytes and contains a word (or the first five characters, if the word is longer than five characters) a word type from 0 to 6 and a word value in the range 2-254. The types are:

0 - Verb
1 - Adverb
2 - Noun
3 - Adjective
4 - Preposition
5 - Conjugation
6 - Pronoun

Words with the same word value are called synonyms. The entries are held in ascending order of word value and within each word value, entries with more spaces come first e.g.

U
UP
CLIMB
ASCEN(D)

where entries with the same word value also have the same number of spaces the entry inserted first comes earlier e.g. CLIMB was inserted before ASCEN(D).

Note 1. Whenever the editor has to convert from a word value to a word it takes the first word with that value of the required type.

Note 2. Verbs and Conversion Nouns with values less than 14 should be reserved for movement words.

The Object Initially At table

This table has a 1 byte entry for each object, which specifies the location at which the object is situated at the beginning of

the adventure. An object can also start the adventure being worn, carried or not-created.

The Object Word table

This table has a 2 byte entry for each object, which holds the Noun and Adjective word values associated with that object.

The Object Weight and Attribute table

This table has a one byte entry for each object. Bit 7 is used to show if the object can be Worn/Removed (i.e. the WR option). Bit 6 is used to show that the object is a container (the C option) and Bits 0 to 5 define the weight of the object (giving a range from 0 -63).

The graphics area of the database grows DOWN from the top of memory:

The Location Flags

This table has a 1 byte entry for each picture; Bit 7 specifies if the picture can be drawn when that location is reached (i.e. if the location is not a 'subroutine'), bits 0 to 5 describe the start PAPER & INK for the picture while bit 6 is unused.

The Picture Table

Each entry in the table uses 3 bytes plus the length of the Drawstring. There are always the same number of entries as locations in the adventure. The Drawstring is encoded as a variety of various length commands which minimise the amount of memory needed to produce the drawing.

The pointers

The main database pointers.

Detailed Description of the Editor

Each menu option is described in the order it appears;

Vocabulary

Insert I word No. Type

No. is in the range 2-254 and Type is in the range 0-6.

If word is not already present in the vocabulary it is inserted with a word value of No. and a type of Type.

Delete D word

If word is present in the vocabulary, it, its type and its word value are deleted. If synonyms of the word deleted are present in the vocabulary no further action is taken. However, if no synonyms are present, then:-

a) all entries in the process tables which use this word value (in respect of type also) are also deleted.
b) if the word value is less than 14 then all movements in the connection table which use this word value are also deleted.
c) All entries in the Object Word Table which use this word value (in respect of type also) are set to null.

Show synonyms S word

If word is present in the vocabulary, it and all other words with the same word value and type are displayed. (Note that for Conversion Nouns - those less than 20 - any synonymous Verbs will also be shown and vice-versa).

Print P (Type) or L (Type)

Printing is either to the screen using P or to the printer using L. If Type is specified only words of that type are listed.

Note 1. Be careful using delete as it can also affect the process, connections and object word tables. It can also take a long time (minutes) if the database is large.

Note 2. Verbs and Nouns with a word value of less than 14 are assumed to be movement words by the Interpreter and cause SM7 ("I can't go in that direction.") to be printed instead of SM8 ("I can't do that").

Note 3. Word values from 2 to 254 can be used for each word type. The words will not by synonymous because they are of a different type. This allows over 250 words of each type - even without synonyms - to exist in the vocabulary!

Location Text

Insert I

If the maximum number of locations has been inserted then the error "Limit Reached" is generated. Otherwise the next available location number on the highest used RAM page (shown on the Free Memory option) is allocated and a null entry is made for it in the graphic location flags, the picture table, connections table and the location text table. Processing then continues with an automatic call to the amend routine to allow the writer to amend the null text entry already set up in the location text table.

Begin new Page B

If all RAM pages have been used or the maximum number of locations have been inserted the error "Limit Reached" is generated. Otherwise the next available RAM page has a null database created on it and the next available location number is assigned. Processing then continues with an automatic call to the amend routine to allow the writer to amend the null text entry in the new page.

Amend A locno.

The existing text for Location locno. is copied to the input buffer and displayed at the bottom of the screen for amending. When ENTER is pressed the existing entry is replaced with the contents of the input buffer.

Print P (locno.) or L (locno.)

Printing is either to the screen using P or to the printer using L. Printing starts with the text for Location locno. or at the beginning if locno. is not specified.

Note 1. The start of an adventure is always at Location 0.

Note 2. There is a limit of 252 locations.

Note 3. You will be unable to begin a new page if all available location numbers have been previously allocated.

Connections

Amend A locno.

The existing entry for Location locno. is decoded, copied to the input buffer and displayed at the bottom of the screen for amending. When ENTER is pressed the input buffer is vetted to be empty or to contain word locno. repeated any number of times. word must be a Verb (or Conversion Noun) which is present in the vocabulary and locno. must be present in the location text table.

If there are no syntax errors the existing entry is replaced with an encoded copy of the input buffer (i.e. words changed to word values).

Print P (locno.) or L locno.

Printing is either to the screen using P or to the printer using L. Printing starts with the entry for Location locno. or at the beginning if locno. is not specified.

Note 1. A location text must be present for a Location before connections can be present.

Note 2. Any Verbs (or conversion Nouns) in the Vocabulary may be used in the Connections table not only movement words.

Note 3. When an entry is decoded (for Amend or Print) the word value is changed into the first Verb (or conversion Noun) in the Vocabulary with that word value.

Graphics

Amend A picno.

The graphic database is expanded to provide a gap at the end of the required picture. The main loop of the Graphic Editor described below is then entered. When return is pressed any gap still remaining is removed. n.b. unlike editing text the database itself is changed, thus you cannot abandon an edit with CAPS SHIFT & 6 (it does a NEXT command for a start!).

Size S

The number of bytes between the start of the drawstring and the start of the next is calculated and printed on the screen.

Print, Copy and Dump. P picno., C picno. and D picno.

picno. must be specified.

IF D was selected a filename is requested.

The required picture is drawn on the screen.

If C was selected the printer COPY is called (see printer section for details).

If D was selected a SCREEN$ file is saved with the filename given (i.e. a code block of the screen 16384,6912), this file could then be used as a loading screen for the game or edited using another art package.

Note 1. You cannot reload a SCREEN$ file into the database.

Default Colours

Amend A picno. (Paper Ink)

A flag is set to indicate that picture picno. is a subroutine unless PAPER and INK values are specified, in which case they are stored as the default colours for the picture. All locations in the adventure which do not require a picture should be Amended as a subroutine.

Print P or L

Printing is either to the screen using P or to the printer using L. If the location is not a subroutine the colours are printed.

Messages

Insert I

The next available message number is allocated and a null entry is made for it in the message text table on the highest used RAM page (shown on the Free Memory option). An automatic call to the amend routine is then made to allow the user to amend the null entry.

Begin new Page B

The next available message number is allocated to the null message on the next RAM page if it has been initialised on the location text menu. Otherwise the error "Page not initialised" is generated. Processing continues with an automatic call to the amend routine to allow the user to amend the null entry.

Amend A mesno.

The existing text for message mesno. is copied to the input buffer and displayed at the bottom of the screen for amending. When ENTER is pressed the existing text is replaced with the contents of the input buffer.

Print P (mesno.) or L (mesno.)

Printing is either to the screen using P or to the printer using L. Printing starts with the text for message mesno. or at the beginning if mesno. is not specified.

Note 1. There is a limit of 255 messages, but as system messages can be inserted and printed they can also be used in the game providing 510 messages (although some are used)!

Note 2. You cannot begin a new page of messages without first initialising the page by inserting a location on it.

System Messages

Insert

The next available system message number is allocated and a null entry made for it in the system message table on RAM page 0. An automatic call is then made to the amend routine to allow the writer to amend the null entry.

Amend A mesno.

The existing text for system message mesno. is copied to the input buffer and displayed at the bottom of the screen for amending. When ENTER is pressed, the existing text is replaced with the contents of the input buffer.

Print P (mesno.) or L (mesno.)

Printing is either to the screen using P or to the printer using L. Printing starts with the text for system message mesno. or at the beginning if mesno. is not specified.

Note 1. The description of the Interpreter shows where these messages are used. They may be changed to use "You" instead of "I" if you prefer, i.e. "You're not wearing it", or even into different languages, but be careful to maintain their meaning.

Note 2. Messages 30 and 31 are not really messages but contain the positive and negative replies used in the QUIT and END condacts. Therefore be very careful changing these as action END is the main way back to the Editor from the Interpreter. You shouldn't make them more than one character in length, or the compressor may tokenise them!

Note 3. SM10 (" (worn)") has its length calculated by INVEN. This means it should not include any control codes, and must be re-entered if the compressor is used to remove any tokens. In addition, for the calculation of screen position to work, it should start with a space.

Note 4. SM34 (the cursor) may contain any number of control characters, but, must not move the print position more than one character space or input will become confused. Any input characters will be in the temporary colours in force at the end of this message.

Note 5. A further 202 messages may be inserted for your own use, but as there is no equivalent of the MESSAGE action a NEWLINE action must be carried out explicitly if needed.

Note 6. CPM PAW uses messages 54 to 60. This should be borne in mind if you intend transferring a game to another system.

Object Text

Insert I

The next available object number is allocated and a null entry is made for it in the object text table. An entry of not-created is made for it in the object initially at table, an entry of "_ _" in the object word table and a weight of 1 unit without any attributes in the object weight table. Processing then continues with an automatic call to the amend routine to allow the user to amend the null text entry already set up in the object text table.

Amend A objno.

The existing text for Object objno. is copied to the input buffer and displayed at the bottom of the screen for amending. When ENTER is pressed the existing text is replaced with the contents of the input buffer.

Print P (objno.) or L (objno.)

Printing is either to the screen using P or to the printer using L. Printing starts with the text for Object objno. or at the beginning if objno. is not specified.

Note 1. Object 0 is considered by the Interpreter to be a source of light.

Note 2. There is a limit of 255 objects.

Object Initially At table

Specifies the location at which an object is situated at the start of the adventure.

Amend A objno. locno.

The existing entry for Object objno. is replaced with locno. which must either be present in the location text table or be one of the special locnos. 252 not-created, 253 worn or 254 carried.

Print P or L

Printing is to the screen using P or to the printer using L.

Note 1. An object text must be present for an object before a start location can be present.

The Object Word table

Amend A objno. Noun Adjective

The existing entry for Object objno. is replaced with the values of Noun and Adjective which must be present in the Vocabulary (or be an underline to set the entry to null).

Print P or L

Printing is to the screen using P or to the printer using L.

The Object Weight table

Amend A objno. Weight Option

The existing entry for Object objno, is replaced with the weight and option specified. The weight may range from 0 to 63. The option specifies the object attributes and may take the values:

0 - No attributes
1 - The object is a container.
2 - The object is wearable/removeable
3 - The object is a wearable/removeable container!

Print P or L

Printing is to the screen using P or to the printer using L. The weight can be followed by "C" to indicate a container and/or "WR" to indicate wearable/removeable.

Note 1. You may only select the container attribute for an object, if a location with the same number exists. This is because PAW treats this location as the 'inside'!

Note 2. A container of weight 0 will not have the weight of its contents added to any calculation. This could be used to create a magic sack or a levitation transporter etc.

The Process (including Response) tables

Two options are available only on the main Process sub-menu:

Begin new Table B

The next available process number is allocated and a null table is created. Processing continues with a call to the Select table option with the new table number.

Select table S No.

The specified table is made the currently selected table and the sub-menu is redisplayed.

The other options are available on both the Response and Process sub-menus:

Amend A Verb Noun (n)

If an entry number n is not specified the first entry in the table of Verb and Noun is copied to the input buffer and displayed at the bottom of the screen for amending. Otherwise the entry number specified is used (ranging from 0 to the number of entries inserted - 1). When ENTER is pressed the input buffer is vetted to be empty, in which case the existing entry is deleted, or to contain any number of valid condacts. If there are no syntax errors the existing entry is replaced with the contents of the input buffer. Any following entries in the table with the same word values (i.e. Verb and Noun) are then displayed in turn for amending in the same way.

Insert I Verb Noun (n)

Verb and Noun must be underline characters, asterisk characters or words which are in the vocabulary. The word values of Verb and Noun (underline has a word value of 255, asterisk a word value of 1) are used to find the correct place in the table for the new entry to be created. If any entries already exist for Verb Noun and no entry number n is specified then the new entry will be created after the existing entries. Otherwise the new entry will be created <u>before</u> the entry number specified. A null entry is created at the appropriate place and an automatic call made to the amend routine to allow the user to amend the null entry.

The condacts that may be used are shown in the description of the Interpreter and in the summary at the end of this guide.

Print P (Verb (Noun)) or L (Verb (Noun))

Printing is either to the screen using P or to the printer using L. Printing starts at the first entry with word values of Verb Noun If Verb or Noun are not specified then a word value of 0 is assumed. Thus P or L by itself starts at the beginning of the table.

Note 1. To delete an entry; amend it, so that no conditions or actions remain.

Note 2. There is a limit of 255 process tables. Calls can be nested to a maximum of 10, after which an error occurs.

Extra Options

Selects the other display for the main menu.

Test Adventure

"Do you require diagnostics?" is printed and any reply that doesn't start with "Y" is assumed to be negative. A jump is then made to the Interpreter. If diagnostics were requested then whenever the Interpreters' input routine is used, pressing ENTER without typing anything will result in the value of a flag being displayed in the lower screen along with an input prompt. You can at this point type in the number of another flag to display that flag's value or an "=" followed by a number to set the flag displayed to that value. Pressing ENTER returns to the input.

The main way back to the Editor from the Interpreter is by performing the action END in a process table.

Free Memory

The number of unused bytes on each page are printed. Pages which have not yet been initialised are shown as "unused" and if Page 7 currently contains the overlays this fact is also displayed. Following the unused memory list will be lists of the last message and last location used on each page to allow you to monitor them.

Note that on a 48K spectrum only page 0 will be displayed - as that's all there is!

Background Colours

The BORDER, PAPER & INK colours may be set to any valid values. INK 9 (i.e. contrast) is recommended but please note that INK 9 behaves differently in the bottom part of the screen.

In addition if any character sets have been inserted into the database you will be able to select them as the primary set.

Characters

Characters consist of an 8 by 8 pixel grid and define the way letters, numbers, punctuation, UDG's and shades look on the screen. Initially only one character set is present, the one that is contained within the spectrum ROM. This set is also considered to contain the 16 shade patterns (codes 0 to 15) and 19 UDGs (codes 144 to 162) and is termed Set 0.

Insert I

If the maximum number of character sets has not been reached the next available set number is allocated. A 768 byte entry is inserted in the characters table and a copy of the designs for the 96 characters (codes 32 to 127) in the ROM are copied into the entry. Otherwise an error of "Limit Reached" is generated.

Amend A set character

The character specified from the set specified is presented on screen for possible amending. Only characters 0 to 15 and 144 to 162 may be amended from set 0, whereas only characters 32 to 127 may be amended from any other set.

The editor provides three areas of screen:-

1/ The Grid — The bit patterns of a particular character are shown on a much enlarged (x8) grid of yellow and white squares, any set bits being shown by a black square. Also present on the grid is a flashing red square showing the current cursor position.

2/ The Test Patterns — To the right of the Grid are two test patterns of the current character as if it was a shade in both normal & Inverted forms. This does not change as you change the Grid but it can be updated by pressing key R for Redraw.

3/ Status Area — This shows the current character, set and a summary of the commands available.

In order to modify the pattern use the cursor keys (CAPS SHIFT 5 to 8 on a 48K spectrum) to move the flashing cursor. The state of the bit under the cursor can be changed at any time using the SPACE key. ENTER will store the amended character back in the database. Q will abandon the edit leaving the character unchanged.

Print and Copy sets P or C

The currently defined UDG's, Shades and inserted character sets are printed on the screen.

If C was selected then the printer COPY is called - see printer section for details.

Load and Save sets L set or S set

If set has been inserted a prompt is printed for a filename, the tenth letter is converted to a (c) sign and the required set loaded or saved as specified. Saving/Loading set 0 acts on the 286 bytes of UDG's and Shades.

Note 1. You cannot amend a character from a set or load a set until it has been inserted.

Note 2. The files for sets other than 0 are a standard 768 byte spectrum character set. They could thus be loaded into/defined with, a different designer as long as the filename ends in position 10 with a (c) symbol.

Compress text

"Compress database (Y/N)?" is displayed and any reply which doesn't start with "Y" is assumed to be negative and causes a return to the main menu. Otherwise a 222 byte dictionary is inserted into the database. The database is then scanned for occurrences of each of 90 common letter groupings which are replaced with a single byte token in the range 165 to 255. The on screen counter displays the number of tokens remaining. This method of compression reduces any text in the game by approximately 40% This could on a text only 128K game provide an equivalent of 160K of memory for the game!

Note 1. When editing text after compression the cursor will skip 2, 3, 4 or 5 characters due to the tokenisation. Corrections should be typed in full as they will be re-compressed the next time the compressor is used.

Save Database

The database area is saved to tape as a sequence of files, two per page. The tenth position of the filename is set by PAW as the letters A to L in sequence for each file.

Verify Database

The files for a database previously saved to tape are checked for differences against the database currently in memory.

Load Database

The files for a database previously saved to tape are loaded overwriting any database currently in memory.

Very Important

If BREAK is pressed or a tape error is detected during a load then the database held in memory will be corrupt and should not be used as it may corrupt the Editor and Interpreter. Under these circumstances the only Editor option which may be used safely is Load Database and this should be used until a database is loaded successfully.

Save Adventure

The Interpreter and database are saved to tape as a sequence of files using the filename specified. They are saved in such a way that the Adventure will auto run when loaded from BASIC using LOAD"" - without the PAW Editor being present.

Verify Adventure

Verifies that an Adventure has been saved correctly.

User Overlays

This allows any of 26 overlays (with the name PAWOVR x) where 'x' is a letter from A to Z to be loaded into the overlay area and executed. You will be prompted for which overlay to load. This can be any of the letters A to Z. PAW then searches the current device for an overlay with that extension.

This feature is designed to allow third party software producers to create products which integrate with the PAW system correctly. The products must be written in assembly language and can be up to about 5K in resident length. PAW PHOSIS/TEL/MEGA are an example of the many uses to which the feature can be put. More details are given in the User Overlay Writers Guide.

One user overlay is provided on the PAW cassette. This is PAWOVR H, the Hunk Manager. You will find this after the other overlays.

Hunk Management

The Hunk maager allows the manipulation of the data which may be inserted in the database by other user overlays. This data is inserted in a documented fashion by well behaved User Overlays using a system of memory Hunks (sections or areas of the database). The hunks of memory can be almost any size from 0 bytes (there is always a 3 byte overhead so a zero byte hunk will be three bytes long - it just won't have any room for information!) to the size of the free memory (although on a 128K Spectrum the maximum size of all hunks is limited to about 6K if you wish to use other character sets).

Each user overlay may own one (or more) hunks to contain information which will be preserved with the database. An example of this is the Direction Pointer Table (DPT) of PAW-TEL (one of the PTM overlays) which is used to describe how the various directions will be represented with the Map command. Thus it is related to the database and is included within it to save retyping it every time you load PAW-TEL.

The Sub-Menu

In the following description of each command, 'overlay' indicates the letter of the User Overlay which 'owns' the hunk. E.g. The DPT would be owned by overlay 'T' as it is used by PAW-TEL.

Insert I overlay size

Will insert a hunk of space (and initialize it to zero) of size (plus three byte overhead) belonging to User Overlay overlay. Thus to insert a DPT (for PAW-TEL) you would use I T 12, to insert the required space - This will of course insert 15 bytes, 12 of which are for data.

Delete D overlay (n)

Will allow the n(th) hunk belonging to User Overlay overlay to be deleted. It is possible (but not usual) for a User Overlay to own more than one Hunk, this allows you to delete the required one!

Load L overlay (n)

Allows a file to be loaded from the current device into the data area of the n(th) hunk belonging to User Overlay overlay. It must load exactly the right number of bytes (E.g. 12 for a DPT) to fill the data space of the hunk.

Save S overlay (n)

Allows the data area of the n(th) hunk belonging to User Overlay overlay to be saved to the current device.

Verify V overlay (n)

Will allow the data area of the n(th) hunk belonging to User Overlay overlay to be compared against a file on the current device. This is only of use if no change has occurred in the address of the hunk, i.e. soon after Saving it!

Print P

Will list any hunks present in the database, as the User Overlay which 'owns' them, which number they are and their true size - i.e. including the three byte overhead. There is no theoretical limit to the number of hunks belonging to a User Overlay, but a practical limit is set by free memory and the fact that Hunk Management can handle a maximum of 255!

Uses

The Hunk Management overlay will have no direct use immediately, but as more user overlays become available (or you write some yourself) you will find it useful to keep track of data being handled by the overlays. Some suggestions follow:

1/ Some user overlays may provide no way to Save and Load the data from their hunks to use in other databases. PAW-TELs' DPT is a trivial example. You could use the Hunk Management to do this using its Save and Load commands.

2/ Indeed if they are feature packed some overlays may provide no way of Inserting a data area for themselves - again this can be achieved with Hunk Management.

3/ Perhaps the most useful is to allow you to squeeze the last useful features into your game, by deleting all the unnecessary Hunks as you approach a full database!

Detailed description of the Graphic Editor

This section of PAW allows a variety of operations to be carried out on the drawstring for a location. When editing, the string is laid out in memory as follows;

END	The end of string marker
NEXT	Any commands still undrawn
SPARE	Available memory
END	Temporary end marker
DRAW	The main draw string

A rubber banded line is used for drawing; the base point of the line (Known as 'point') shows the last point plotted, moved to etc, the rubber banded end of the line shows the next position of point or the start point for a fill/shade etc.

The Editor provides four groups of commands. Any which insert a command into the drawstring require the SYMBOL SHIFT key to be held down;

1) Drawing Commands

ABS MOVE	A	Moves point to the x,y position of end of the line setting only the attributes. This is coded as a PLOT with Inverse and Over set on.
PLOT	P	Sets the pixel at the end of the line according to Inverse and Over, then moves point to that position.
REL MOVE	R	Moves point to the end of the line without affecting the screen. This is coded as a relative offset from the old point.
LINE	L	Draws (or fixes) a straight line from point to the end of line according to Inverse and Over, then moves point to end of the line. The line is coded as a relative offset from the old point.

FILL F The area from the end of line (relative) is filled using solid pixels. Fill works by passing a pattern to the SHADE routine so the notes on SHADE apply also

All the above use 3 bytes in the database.

SHADE S The area from the end of line (relative) is shaded with one of a large number of patterns. The database contains 16 patterns (0-15), which can be changed using the Character Editor.

The pattern used for shading is determined as follows:-
a) You are asked for 2 pattern numbers in the range 0 to 15. If you only want the one pattern then specify the same number for both patterns.
b) The 2 patterns specified are OR'd together i.e. they are placed on top of each other.
c) If INVERSE was 'on' the resultant pattern is inverted, i.e. SET/RESET pixels are swapped.

Note 1. The shade first works in a downward direction and then in an upward direction. For speed, when it is going down it doesn't look up and vice versa. Any areas the shade misses must be shaded separately, although careful choice of the start position for the shade will minimise this.

Note 2. If the area to be shaded is too complex then the shade will be abandoned. It has to do this to enable it to detect when it comes across an area which has already been shaded. Thus an area can only be shaded once as an already shaded area will be too complex to shade again. You should not shade an area and then try to fill in the background with a fill command, use the Inverse option!

TEXT T If any character sets have been inserted in the database, a set number is requested. Then a character code in the range 32 to 162 is requested. This character is placed on screen in the character square (as shown by the Grid command) that the tip of line is contained within. It is manly designed to allow "the fiddly bits" of as picture to be drawn using the character editor - thus using less memory than lots of lines!

Text and Shade use 4 bytes in the database each.

BLOCK B Causes a block of the currently selected colours to fill the rectangle of attribute squares which the line defines the diagonal of.

Block uses 5 bytes in the database.

2) Colour Commands

INK	X	The current ink is set to the value selected. INK 8 as in BASIC causes all ink to be taken from the existing screen attributes.
PAPER	C	Sets the current paper to the value selected. PAPER 8 as in BASIC.
FLASH	V	The new value of Flash is requested (0,1 or 8).
BRIGHT	Z	The new value of Bright is requested (0,1 or 8).

all the above use one byte in the database.

INVERSE	I	The state of Inverse (on/off) is toggled.
OVER	O	The state of Over (on/off) is toggled.

Neither Inverse nor Over use any memory but their state is encoded as part of each future instruction which is affected by them.

3) Subroutine Command

GOSUB G A picture number is requested which must be in the range 0 to locno. A scale value for the picture is then requested. This can be from 0 to 7 where the number indicates the size of the picture in eighths - 0 means 'no scale' (i.e. 8/8).

Please Note;

a) Scale only affects certain commands, these are MOVE RELATIVE,LINE,FILL and SHADE. MOVE ABSOLUTE, PLOT, BLOCK and TEXT commands will not be scaled or relocated and should generally not be used in subroutines (although they will work and can be used usefully sometimes).

b) You may only nest subroutine calls to a level of ten. (nesting means calling a subroutine from within a subroutine).

c) Scale does not affect GOSUB commands, i.e. if a GOSUB is used within a subroutine the string drawn will be at a fixed size and not scaled.

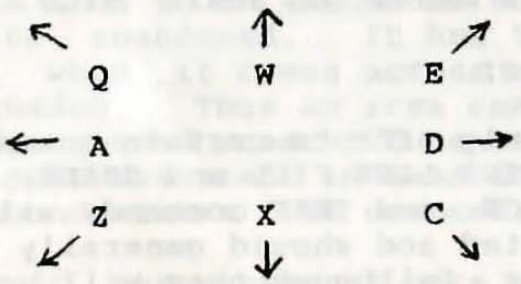
d) Calling the same routine you are drawing will cause a "Limit Reached" error as the limit of 10 subroutine levels will have been reached.

Gosub uses two bytes in the database.

4) Editing commands

START		Puts the Drawstring pointer at the start of the drawstring.
NEXT		Executes next available drawstring command: if there isn't one the command is ignored.
PREVIOUS		Moves the drawstring pointer back one command and updates the screen.
DELETE		(CAPS SHIFT & 0 on 48K) deletes the previous command in the drawstring and updates the screen.
DELETE NEXT		GRAPH(ICS) (CAPS SHIFT & 9 on 48K) deletes the next command if there is one.
GRID	Y	Has a toggle action for a character grid of INK 0, PAPER 7 and PAPER 6. This allows exact positions of colour boundaries to be taken into account while drawing.
JOYSTICK	J	Toggles the Kempston™ joystick option on and off.

The keys around S move the end of line around one pixel at a time (this can be accelerated to eight pixels a time by holding down the CAPS SHIFT key) thus:



The joystick should be plugged into Port 2 of Interface 2 or the Spectrum Plus 2. Alternatively it can be plugged into a Kempston™ interface in which case SYMBOL SHIFT & J should be used to enable PAW to read it. CAPS SHIFT will also accelerate the rate of movement on the joystick. The Fire button will act like SYMBOL SHIFT and L to draw a line.

Editor Error Messages and their meanings

Due to the complex nature of PAW many features can generate errors. Whenever an error is discovered a message describing the error is printed in the lower section of the screen. PAW then waits for a key to be pressed before returning to the menu on which that error occurred. There are two exceptions to this:

1/ After an error during editing of a drawstring; the Drawstring pointer is positioned just before the command which caused the error (i.e. a NEXT command will cause the error again). If you are unable to correct the problem then DELETE NEXT can be used to delete the erroneous command. Note that this may still leave further commands undrawn (which may cause another error).

2/ While an adventure is running; If the PAW editor is not present then a jump is made to initialise a new game after the error report. Otherwise a diagnostic line is printed on the bottom line of the screen. This displays the Process table, Verb and Noun of the entry and the condact in which the error occurred. You are then provided with the flag diagnostics to help confirm where the error occurred. When ENTER is pressed the normal error report is printed and a return made to the main menu of PAW.

When using your own programs in conjunction with the EXTERN command any of the standard BASIC errors can occur. Note though that the line and statement number are not printed - so test your program thoroughly without the adventure present!

PAW also generates several errors of its own which are:

BREAK	BREAK was pressed during a peripheral operation or while a game was running.
STOP in INPUT	CAPS SHIFT & 6 pressed.
Tape loading error	as BASIC. Note that a tape error during a load database means that the database is corrupt and only reloads should be attempted.
Database full	There is not enough room in the database for what you were attempting.
Limit reached	The maximum number of locations, messages, objects, process tables or RAM pages are already present. Alternatively the maximum subroutine or sub-process depth has been reached - 10. This can also occur if an attempt is made to DOALL while a DOALL loop is active.

Errors

Integer out of range	While drawing a picture a LINE command has gone out of range. This is usually due to a change of position of the starting point while editing.
Out of memory	This indicates that the entry for a process or connection entry was too large for the buffer (unlikely to occur). Also occurs when attempting to start a new RAM page on a 48K spectrum, loading a 128K game on a 48K spectrum or if insufficient workspace has been left e.g. by setting up fixed channels before loading or by writing too big a program for use with EXTERN.
Invalid Argument	This error is printed when a condact discovers an illegal value. e.g. An attempt to set flag 38 to an illegal location, an attempt to PUTIN/TAKEOUT on a non container object or to set an object at location 255 (COPYFO only). But see point 2 above for diagnostic information.

Note 1. During input the Spectrum will emit a RASP if the screen and/or input buffer is full.

Note 2. If an entry is present in the database which is too big to fit on one screen, the Spectrum will give out a RASP when the entry is displayed at the bottom of the screen for amending.

Note 3. If an abnormally large entry is inserted in the connections table using abbreviations e.g. N 1 W 6 S 4 etc and the abbreviations are deleted from the vocabulary, the movement entry (when decoded) i.e. NORTH 1 WEST 6 SOUTH 4 etc could be too big for the input buffer. If this happens an out of memory message will be produced. The remedy is to reinsert the abbreviations in the vocabulary.

The Professional Adventure Writing System

A Supplement for the

Spectrum — Version A 16

Additional Notes for PAW versions A16 and later.

The current version of PAW is slightly different from that described in the User Guides supplied. The main differences are described below.

User Overlays

Versions of PAW from version A16 are primarily intended to provide a well documented facility for third party software producers to create products which integrate with the PAW system correctly - User Overlays. The products must he written in assembly language and can be up to about 5K in resident length. A document is available from Gilsoft, to genuine interested writers, which describes the function calls and database structure, on provision of a large (AS) S.A.E.

The system is implemented as an extra menu option ('Z'). This is on the second half of the main menu, but can of course be selected whichever is displayed. You will he prompted for which overlay to load. This can be any of the letters A to Z. PAW then searches the current device for an overlay with that extension.

Current Device

PAW now maintains the idea of a 'current device', that is the device (Disc/Cassette/Microdrive), to which data is to be saved and loaded from. Only on the various Disc versions is there a way to change the current device (option Y, described in the additional notes for each drive) from Disc to Cassette and vice-versa. This allows user overlays to take advantage of disc versions to do any saving and loading without actually knowing about the types of drive available to PAW.

Printer Drivers

Printer drivers are now limited to 48 Bytes at address 29587 (PRTADD). The memory has been considerably reduced in order to provide the extra facilities.

The Parser

One or two subtle changes have taken place in the string handling section of the parser from Version A16 onwards. They are designed to provide a facility for multiple commands to he given to PSI's, a facility suggested by Gerald Kellett.

The three changes are as follows, and although they may not seem very major the logic changes they represent could affect some games if you weren't aware of them:

1/ The PARSE CondAct now maintains a 'current position' within the string in the current logical sentence. Thus a second PARSE CondAct will continue from where the last left off. Previously a subsequent PARSE would have given the same logical sentence as

the first. Thus SAY TO PSI "GET SWORD AND CLEAN IT" can now be made to work with some processing as described below

2/ The PARSE action does not now affect the 'command line empty/valid' flag - the one set by NEWTEXT. This means that a statement such as; SAY TO PSI "HGGHHG". GET SWORD - will now continue on to do the GET SWORD action. Prior to this version the NEWTEXT flag would have been set automatically. This was changed to allow multi-parsing to find the last command in a string without always setting the flag. You will thus need to add a NEWTEXT action to old games just after the PARSE CondAct - which is where processing goes if the string was invalid or empty - if the games are to operate identically with the new paws.

3/ The current verb and Adverb are not cleared (flags 33 and 36 to 255) when a string is parsed (i.e. the PARSE action). This means that if a Verb (or Conversion Noun) is omitted from the first phrase in the string then the current verb will be the one from the phrase which triggered the PARSE (usually SAY or TALK). This is a minor change which means that the current Verb is maintained when the string is multi-parsed. I.e. SAY TO PSI "GET SWORD AND SHIELD" will now work with the processing shown below.

Extra System Flag

Flag 58 has now achieved the status of a system flag... Don't say you've used it - it was marked as 'reserved for expansion'.

If you set this flag to 128, in a Process table, PAW will start to match words which it normally doesn't do except in Response. This allows the multi-parse facility to provide actions for a PSI during Process 2. It will also have other uses we are sure...

The effect is cancelled next time Process 1 or 2 are carried out - by PAW subtracted 128 from the flag. This ensures that Process I and 2 act as normal until specifically told to change. You can of course cancel the effect yourself by setting the flag back to zero.

Using Multi-Parse

The ability to give a PSI a list of commands to do has incredible possibilities for the creation of synchronized problems. Where both the PSI and the player must work together.

These sort of problems can add a whole new dimension to adventures and are well worth considering, here are some suggestions:

> Imagine a game with a room that is instant death for the player which contains an object that he requires. You could instruct a PSI to go in, get the object and come back out.
>
> Say that in order to kill a certain monster you needed a simultaneous attack from three characters. You could use

the following:
SAY TO PSI1 "WAIT, KILL MONSTER"
SAY TO PSI2 "KILL MONSTER"
KILL MONSTER
All three KILL MONSTERs would he carried out in the same time frame.

They say the best way to demonstrate something is by example. So here goes with a short listing of a game with only one problem:

In order to get out of a cavern you need to he lifted on a platform controlled from another room. This can only be achieved by giving a PSI (who happens to be hanging around) a list of things to do. I.e. Go to the cavern and pull the rope. While you in the meantime step onto the platform and wait...

Flag Usage

20 - Location of PSI
21 - copy of flag 20 during movement processing

60 - when 0 indicates platform is on floor, 1 - held by PSI and 2 - held by Player.

195 - Players Verb/Pronoun-Noun Saved
196 - Players Adverb/Pronoun-Adjective Saved

197 - Number of Logical Sentences waiting for PSI
198 - Next storage flag group to store LS in
199 - Next storage flag group to get a LS from
200 - 206 - Store 0 for LS
207 - 213 - Store 1 for LS
214 - 220 - Store 2 for LS

Notes

The principle of the multi-parse is that the entire string is broken down into a list of LSs that the PSI will he required to do. These LSs are then stored (saved if you like) in some flags to be doled out, one per timeframe (use of process 2).

The LSs waiting for the PSI to do are held in a 'queue' which is a computer term for an ordered list. They are actually held in a 'round robin fifo queue'. fifo stands for 'first in first out'. i.e. the first LS given to the PSI must he the first it carries out. While 'round robin' indicates that the LS storage used goes around the available storage flags in a circular motion. i.e. it goes back to the beginning when it falls off the end! Thus the groups of flags will be used in the order; Store 0, Store 1, Store 2, Store 0 etc. The use of only three storage areas means that only three commands can he queued for the PSI, there is no reason why this cannot be expanded upon. Indeed if you only needed Verb Noun commands to he given to PSI's you could save only those parts of the LS. Thus requiring only two flags per LS not 7!

The extraction of multiple phrases is done by a single process table which calls itself to get the next phrase. This is known as 'recursion' and is simpler than a sequence of entries doing PARSE and PROCESS calls etc. It does limit you to 9 phrases in a string though - Why? (Clue: you can only nest PROCESS calls to a depth of ten.)

Locations

Location 0
I am in a large cavern. On the East wall, high up, is an entrance from which a shaft of light descends. A lifting platform, obviously intended as a means of getting to the entrance, is linked via a series of pulleys on the roof to a steel cable which disappears into a hole in the North wall just above a tunnel.
Location 1
I am standing on a platform
Location 2
I am standing on a ledge overlooking a lush green valley. To the West is an entrance to a large cavern.
Location 3
I am in a small ante-room. A twisting tunnel leads South. A steel cable hangs from the ceiling.

Connections

```
Location   0  N    TO   3
Location   1
Location   2
Location   3  S    TO   0
```

Messages

Message 0
A PSI is here.
Message I
The PSI doesn't understand.
Message 2
You have said enough to the PSI.
Message 3
You speak to the PSI.
Message 4
The PSI cannot do that.
Message 5
The PSI pulls on the cable.
Message 6
The PSI releases his grip on the cable.
Message 7
The PSI stands on the platform.
Message 8
The PSI steps off the platform.
Message 9
The PSI leaves.
Message 10
A platform

```
Message    11
The platform
Message    12
 rests on the floor of the cavern.
Message    13
 hangs just inside the opening.
Message    14
 now
Message    15
 which
Message    16
jars into motion.
Message    17
A PSI arrives.
Message    18
The PSI can't go that way.
Message    19
You release your grip on the cable.
```

Response Table

```
*      *      EQ        60     2  ;Player holding cable?
              CLEAR     60        ;Release it
              MESSAGE   19
              PROCESS    8        ;Cancel DONE flag

I      _      INVEN

GET    PLATF  PREP     OFF        ;Movements on and off platform
              AT         1
              ZERO      60
              GOTO       0
              DESC

GET    PLATF  PREP    OFF
              AT         1
              GOTO       2
              DESC

GET    PLATF  PREP     ON
              AT         0
              ZERO      60
              GOTO       1
              DESC

GET    PLATF  PREP     ON
              AT         2
              NOTZERO   60
              GOTO       1
              DESC

R      _      DESC

QUIT   _      QUIT
              TURNS
```

```
                END

SAVE    _       SAVE

LOAD    _       LOAD

RAMSA   _       RAMSAVE

RAMLO   _       RAMLOAD  255

SAY     PSI     NOTSAME   20    38  ;Talk to PSI if in cavern
                ATLT       2        ;or on platform etc
                LT        20     2
                PROCESS    3
                DONE

SAY     PSI     SAME      20    38  ;otherwise have to be same
                PROCESS    3        ;location
                DONE

SAY     PSI     MESSAGE   20
                DONE

WAIT    _       OK

PULL    CABLE   AT         3        ;Allow player to hold cable
                ZERO      60
                LET       60     2
                OK

RELEA   CABLE   OK

STAND   PLATF   PREP      ON
                AT         0
                ZERO      60
                GOTO       1
                DESC
```

Process 1

```
*       *       EQ        31     0  ;Deal with start of game
                EQ        32     0
                MODE       1     1  ;Continuous scrolling text
                TIME       8     3  ;Timeouts
                INPUT      7        ;Input at bottom of screen

*       *       NEWLINE             ;Always start a fresh line
                ATLT       2        ;In cavern or on platform
                MES       11        ;"The Platform"

*       *       AT         2        ;Outside cavern
                NOTZERO   60        ;Platform is at top
                MES       10        ;"A Platform"

*       *       ATLT       2        ;In cavern or on platform
```

```
                ZERO       60        ;which is on floor
                MESSAGE    12        ;" rests on the floor."

*       *       ATLT        3        ;Anywhere except anti-room
                NOTZERO    60        ;Platform at entrance
                MESSAGE    13        ;" by the entrance."

*       _       ZERO        0        ;Standard PAW dark stuff
                ABSENT      0        ;for Object list
                LISTOBJ

*       _       PRESENT     0
                LISTOBJ

_       _       SAME       20    38  ;PSI where player is?
                MESSAGE     0        ; "There is a PSI here."
```

Process 2

```
*       *       NOTZERO   197        ;Any commands for PSI
                LET        58   128  ;Allow word matching
                PROCESS     5        ;extract next action for PSI
                CLEAR      58        ;Prevent word matching
```

Process 3 - Deals with speech to PSI

```
*       *       COPYFF     46   195  ;Save 'IT' for player
                COPYFF     47   196
                SET        46        ;No IT at mo!
                SET        47
                PARSE                ;Get a phrase
                MESSAGE     1        ;nOt one there
                COPYFF    195    46  ;Restore IT
                COPYFF    196    47
                DONE                 ;all over

*       *       MESSAGE     3        ;"You speak to PSI"
                PROCESS     4        ;extract and store phrases
                COPYFF    195    46  ;Restore IT
                COPYFF    196    47
```

Process 4 - This will extract and store up to three phrases although this could he expanded with a few simple changes/extra entries. Note that this is Recursive as it calls itself!

```
*       *       EQ        197     3  ;Max of three phrases in queue
                MESSAGE     2        ;"Said enough to PSI."
                DONE

*       _       ZERO      198        ;Use store 07
                COPYFF     33   200
                COPYFF     34   201
                COPYFF     35   202
                COPYFF     36   203
```

```
            COPYFF    43  204
            COPYFF    44  205
            COPYFF    45  206

*     _     EQ       198    1  ;Use store 1?
            COPYFF    33  207
            COPYFF    34  208
            COPYFF    35  209
            COPYFF    36  210
            COPYFF    43  211
            COPYFF    44  212
            COPYFF    45  213

*     _     EQ       198    2  ;Use store 2?
            COPYFF    33  214
            COPYFF    34  215
            COPYFF    35  216
            COPYFF    36  217
            COPYFF    43  218
            COPYFF    44  219
            COPYFF    45  220

*     _     PLUS     197    1  ;One more phrase stored
            PLUS     198    1  ;Next store
            EQ       198    3  ;reached the last?
            CLEAR    198       ;Go back round

_     _     PARSE               ;Get another phrase
            DONE                ;No more there so finished

_     _     PROCESS    4        ;Store it
```

Process 5 - Extracts the next phrase from store for the PSI

```
*     *     COPYFF    33  195  ;Save Verb/Adverb of player
            COPYFF    36  196

*     _     ZERO     199       ;Store 0?
            COPYFF   200   33
            COPYFF   201   34
            COPYFF   202   35
            COPYFF   203   36
            COPYFF   204   43
            COPYFF   205   44
            COPYFF   206   45

*     _     EQ       199    1  ;Store 1?
            COPYFF   207   33
            COPYFF   208   34
            COPYFF   209   35
            COPYFF   210   36
            COPYFF   211   43
            COPYFF   212   44
            COPYFF   213   45
```

```
*       _       EQ        199    2  ;Store 2?
                COPYFF    214   33
                COPYFF    215   34
                COPYFF    216   35
                COPYFF    217   36
                COPYFF    218   43
                COPYFF    219   44
                COPYFF    220   45

_       _       MINUS     197    1  ;One less in store
                PLUS      199    1  ;Extract next from one more
                EQ        199    3  ;Reached end?
                CLEAR     199       ;Back to start

_       _       PROCESS     6       ;Do the job
                COPYFF    195   33  ;Restore player Verb/Adverb
                COPYFF    196   36
```

Process 6 - Commands that can be given to PSI

```
*       *       EQ         60    1  ;Holding Cable?
                AT          3       ;Where player can see PSI?
                MESSAGE     6       ;"PSI Releases grip"

*       *       EQ         60    1  ;Holding cable?
                CLEAR      60       ;Release grip.
                ATLT        3       ;Can player see effect?
                MES        11       ;Describe "The platform"
                MESSAGE    16       ;" jars into motion."
                MES        11       ;"The platform"
                MES        14       " now"
                MESSAGE    12       ;" rests on the ground."

GET     PLATF   PREP      OFF       ;GET OFF PLATFORM
                EQ         20    1  ;PSI on it?
                ZERO       60       ;Platform on ground?
                CLEAR      20       ;Put PSI in cavern (loc 0)
                ATLT        2       ;Can player see it?
                MESSAGE     8       ;"PSI steps off."
                DONE

GET     PLATF   PREP      OFF       ;GET OFF PLATFORM
                EQ         20    1  ;PSI on it?
                LET        20    2  ;Platform by entrance?
                ATLT        2       ;Player see it?
                MESSAGE     8       ;"PSI steps off."
                DONE

GET     PLATF   PREP       ON       ;GET ON PLATFORM
                ZERO       20       ;PSI on ground?
                ZERO       60       ;along with platform?
                LET        20    1  ;Move PSI to platform
                ATLT        2       ;Can player see it?
                MESSAGE     7       ;"PSI steps on."
                DONE
```

```
PULL   CABLE EQ          20     3  ;PSI in anti-roan?
             ZERO        60        ;with no one holding cable?
             AT           3        ;Is player here as well?
             LET         60     1  ;PSI holding cable
             MESSAGE      5        ;"PSI grips cable."
             DONE

PULL   CABLE EQ          20     3  ;PSI in anti-room?
             ZERO        60        ;with no one holding cable?
             ATLT         2        ;Can player see result?
             LET         60     1  ;PSI holding cable
             MES         11        ;Describe "The platform"
             MESSAGE     16        " jars into motion."
             MES         11        ;"The platform"
             MES         14        " now"
             MESSAGE     13        ;" hangs by the entrance."
             DONE

RELEA  CABLE DONE                  ;Is done by any action!

STAND  PLATF PREP        ON        ;STAND ON PLATFORM
             AT           0        ;See above GET ON PLATFORM
             ZERO        60
             LET         20     1
             MESSAGE      7
             DONE

WAIT   _     DONE                  ;Do nothing for a time frame

_      _     LT          33    14  ;Movement?
             PROCESS      7        ;Deal with it
             DONE

_      _     CLEAR      197        ;Can't do it so cancel any
             CLEAR      198        ;waiting LS for PSI.
             CLEAR      199
             SAME        20    38  ;Is player where PSI is?
             MESSAGE      4        ;"PSI can't do it."
```

Process 7 - Deal with movement for PSI

```
*      *     COPYFF      20    21  ;Save current location
             MOVE        20        ;Try and move
             NOTSAME     20    21  ;Did location change?
             SAME        21    38  ;Was player there?
             MESSAGE      9        ;tell them "PSI leaves."

*      *     NOTSAME     20    21  ;Somewhere new?
             SAME        20    38  ;Where player is?
             MESSAGE     17        ;tell them "PSI arrives."

*      *     SAME        20    21  ;No change?
             CLEAR      197        ;Can't go that way so
             CLEAR      198        ;clear any outstanding LS
```

```
            CLEAR    199        ;for PSI
            SAME      20    38  ;Player here?
            MESSAGE   18        ;tell them.
```

Process 8

```
*     *     NOTDONE             ;Cancel the 'done' flag
```

Playing

If you do type this in you may like to try some of the following sequences from the starting position...

GET ON PLATFORM, SAY TO PSI "GO NORTH, PULL CABLE AND RELEASE IT" THEN GET OFF IT.

This shows the independence of IT for Player and PSI.

SAY TO PSI "N, PULL CABLE", STAND ON PLATFORM, GET OFF IT

Is the solution, although if you wished to lower the platform after.

SAY TO PSI "N, PULL CABLE & RELEASE IT",GET ON PLATFORM AND OFF IT

Would leave you outside without a platform, while...

SAY TO PSI "STAND ON PLATFORM, WAIT THEN GET OFF IT". N, PULL CABLE, RELEASE IT, S

Would leave you without a means of exit and the PSI outside!

A new overlay - Hunk Management

The latest versions of PAW are supplied with an extra overlay, implemented under the user overlay scheme. This is overlay H - Hunk Management. It is supplied for your convenience as a useful utility. It allows the manipulation of the data which may be inserted in the database by other user overlays. This data is inserted in a documented fashion by well behaved User Overlays using a system of memory Hunks (sections or areas of the database). The hunks of memory can be almost any size from 0 bytes (there is always a 3 byte overhead so a zero byte hunk will be three bytes long - it just won't have any room for information!) to the size of the free memory (although on a 128K Spectrum the maximum size of all hunks is limited to about 6K if you wish to use other character sets).

Each user overlay may own one (or more) hunks to contain information which will be preserved with the database. An example of this is the Direction Pointer Table (DPT) of PAW-TEL (one of the PTM overlays) which is used to describe how the various directions will be represented with the Map command. Thus it is related to the database and is included within it to save retyping it every time you load PAW-TEL.

The Sub-Menu

Hunk management is presented in the same format as other PAW menus. In the following description of each command, 'overlay' indicates the letter of the User Overlay which 'owns' the hunk. E.g.. The DPT would be owned by overlay 'T' as it is used by PAWTEL.

Insert **I overlay size**

Will insert a hunk of space (and initialize it to zero) of size (plus three byte overhead) belonging to User Overlay overlay. Thus to insert a DPT (for PAW-TEL) you would use I T 12, to insert the required space - This will of course insert 15 bytes, 12 of which are for data.

Delete **D overlay (n)**

Will allow the n(th) hunk belonging to User Overlay overlay to be deleted. It is possible (but not usual) for a User Overlay to own more than one Hunk, this allows you to delete the required one!

Load **L overlay (n)**

Allows a file to be loaded from the current device into the <u>data area</u> of the n(th) hunk belonging to User Overlay overlay. It must load exactly the right number of bytes (E.g. 12 for a DPT) to fill the data space of the hunk.

Save **S overlay (n)**

Allows the data area of the n(th) hunk belonging to User Overlay overlay to be saved to the current device.

Verify **V overlay (n)**

Will allow the data area of the n(th) hunk belonging to User Overlay overlay to be compared against a file on the current device. This is only of use if no change has occurred in the address of the hunk, i.e. soon after Saving it!

Print P

Will list any hunks present in the database, as the User Overlay with 'owns' them, which number they are and their true size - i.e. including the three byte overhead. Thus the DPT would be represented as:

```
T      0          15
```

There is no theoretical limit to the number of hunks belonging to a User Overlay, but a practical limit is set by free memory and the fact that Hunk Management can handle a maximum of 255!

Uses

The Hunk Management overlay will have no direct use immediately, but as more user overlays become available (or you write some yourself) you will find it useful to keep track of data being handled by the overlays. Some suggestions follow:

1/ Some user overlays may provide no way to Save and Load the data from their hunks to use in other databases. PAW-TELs' DPT is a trivial example. You could use the Hunk Management to do this using its Save and Load commands.

2/ Indeed if they are feature packed some overlays may provide no way of Inserting a data area for themselves - again this can be achieved with Hunk Management.

3/ Perhaps the most useful is to allow you to squeeze the last useful features into your game, by deleting all the unnecessary Hunks as you approach a full database!

The Professional

Adventure Writing System

Introduction

The Professional Adventure Writer

A graphic adventure writing system for the Sinclair Spectrum computers.

(c) 1986/90 Gilsoft.
Program: T.J.Gilberts, G.Yeandle and P.Wade
Graphics: D. Peeke, K.Maddocks and A.Williams
Manuals: T.J.Gilberts

All Rights reserved no part of this publication may be copied, loaned, hired or reproduced in any form whatsoever including electronic retrieval systems without the prior written consent of the authors and Gilsoft

The above notice does not apply to the 'run time' routines appended to and which form a part of a saved game which you are free to distribute any way you wish in that form, all we would request is that you credit the use of the Professional Adventure Writer somewhere within the game.

Introduction

Welcome to the world of adventure writing...

The Professional Adventure Writer (or P.A.W. as it is more commonly known) provides you with the facilities to produce high quality graphic adventures (in machine code) of equal or better quality than many commercially available.

PAW will provide you with the basic framework for writing a game, but it is still up to you to provide an imaginative storyline and original puzzles.

This manual provides a tutorial covering its use in constructing an adventure and we would recommend you work your way through this manual and its accompanying examples before attempting a game of your own.

Good luck...

A great deal of time and effort has been put into ensuring PAW deals with all conceivable situations in a logical and useful manner this has resulted in a complex program of some 20K in size, and it is entirely possible that somewhere deep within the code a few well hidden bugs remain, indeed a well known quote states that; "Testing only proves the presence of bugs, not their absence". If you should find a problem please tell us so that we can correct it if necessary.

All due care has been exercised in the preparation of these manuals and their accompanying programs, however the authors and Gilsoft International Ltd assume no responsibility for errors, omissions or suitability of their contents for any application. Nor do we assume any liability whatsoever for damages resulting from their use. This disclaimer does not affect your statutory rights.

Getting Started

If you have purchased a Disc or Microdrive version of PAW please see the extra instruction sheet supplied for loading details.

The cassette supplied contains the main PAW program on side 1, this program is the same for both 48K and 128K users.

Insert cassette in the recorder and type:-

LOAD "" then press ENTER and PLAY on tape deck.

or on 128K use the load option from the start up menu, note that loading PAW in 48K mode on a 128K spectrum will make it assume it is a 48K version, it is usually preferable to load PAW in 128K mode.

PAW will display a start up screen when loaded which shows its current version number (a letter followed by a two digit number e.g. A01) Also shown are two address' in decimal which will be required if you wish to write your own BASIC or machine code additions to PAW - details in the technical guide.

Pressing any key will cause the main menu to be displayed...

Within this tutorial any input that may be required by PAW is shown enclosed in brackets e.g. [A 1 BAG], if the surrounding text requires the entry to be made type exactly what is within the brackets - including any spaces - but not the brackets themselves. Several special keys are shown by using their name in capitals (upper case), e.g. ENTER,CAPS SHIFT etc do not type these in full, just press the required key (or combination of keys).

Note: If you have a cassette version of PAW and require a Disc or Microdrive version see the section on Upgrades at the end of this manual.

Any communication regarding PAW should be directed to:-

Gilsoft

2 Park Crescent,
Barry,
South Glamorgan,
CF6 8HD.

Concepts

It is probably a good idea at this stage to introduce some of the more important concepts which PAW embodies in its design.

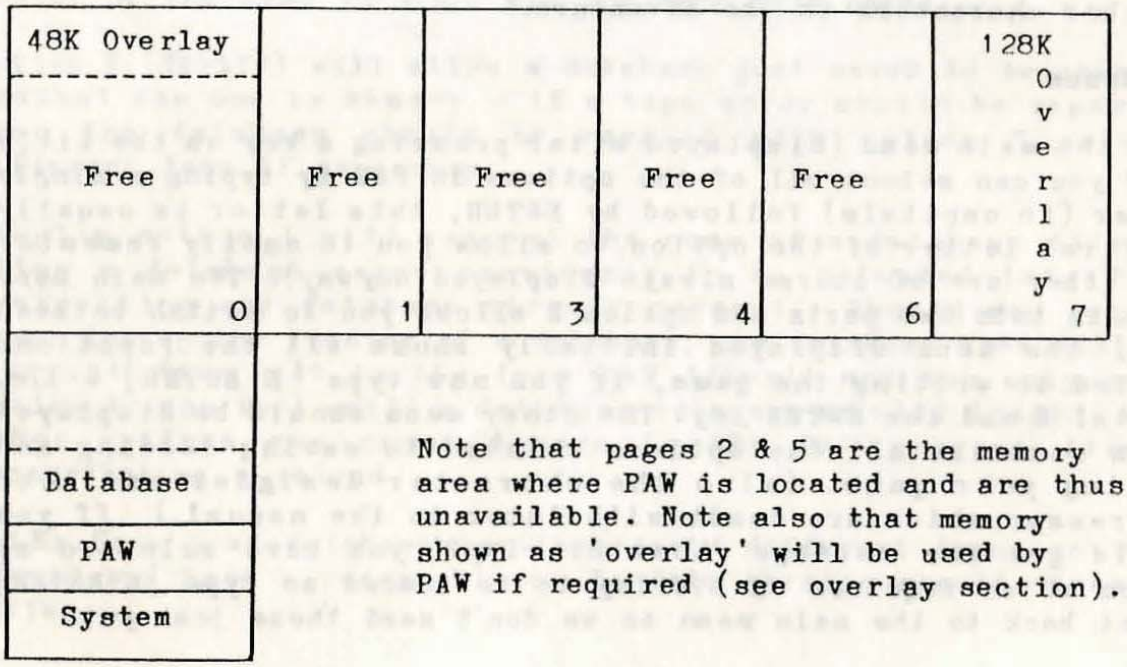
Overlays

For many years it has been common practice for very large programs on disc based machines to be split into two or more parts which are then loaded as required. This system has not been featured greatly on tape software due to the serial nature of tape storage. PAW uses a very simple form of overlay to allow maximum free memory on both 128K and 48K spectrums which is quick and easy to use. The system uses the free memory available on each machine to store the overlays until such time as that memory is required and/or overwritten, they are then pulled in from cassette as required. On a 128K spectrum you will not need to worry about overlays until the last 16K is required (i.e. after you have written a game about 92K long!). On a 48K the overlays will come into play when you need to use graphics or have entered about 16K of program. The tutorial in this manual does not need to use overlays so their use is deferred until a later chapter.

Memory Paging

The 128K spectrum uses a system known as paging to provide its extra memory, diagram 1 should help to visualize the arrangement where the top 16K of memory can be shuffled along like a slide in a projector to present one of five 'pages' to the computer which can only 'see' 64K at any one time.

Diagram 1



Note that pages 2 & 5 are the memory area where PAW is located and are thus unavailable. Note also that memory shown as 'overlay' will be used by PAW if required (see overlay section).

Concepts

Databases

PAW stores your game in a 'database' (a collection of tables and information which define the game you are writing). Initially the database is very small with only the words and commands common to every game already defined. This database gradually uses the area of memory shown as free on diagram 1. PAW can also make use of the other pages, but, on a 48K spectrum the extra pages are not available and you will thus not be allowed to try and use them. This also means that if you are writing on a 128K spectrum and you want your game to run on both a 48K and 128K spectrum you must not use any page other than the main page (page 0).

Parser

Back to school for this bit:-

parse v.t. to classify a word or analyse a sentence in terms of grammar. **parsing** n. (Minster English dictionary)

PAW features a fairly powerful parser to convert what the player types when playing your adventure into a series of simplified 'Logical Sentences' (LS's) to which you will have defined the responses. The parser does this by extracting 'phrases' from the input string one at a time and allowing the rest of PAW to interpret their meaning. Phrases are separated by any punctuation mark and the conjugations 'AND' or 'THEN' (Although you can change this if required), when it runs out of phrases in the current input string it will request another. A phrase consists of at least a Verb (a doing word!) and optionally two Nouns (words which name objects) possibly with associated Adjectives (which describe objects), an adverb (which modifies the verb), a preposition (shows the relation of one Noun with another word) and finally a string enclosed in quotes which is used for speech to other characters in the adventure.

The Menus

From the main menu (displayed after pressing a key on the title page) you can select all of the options in PAW by typing a single letter (in capitals) followed by ENTER, this letter is usually the first letter of the option to allow you to easily remember them (they are of course always displayed anyway). The main menu is split into two parts and option E allows you to switch between them, the menu displayed initially shows all the functions related to writing the game, if you now type [E ENTER] - i.e. Capital E and the ENTER key. The other menu should be displayed which contains all the options related to saving loading and testing your game. (also the character designer and text compressor which are dealt with later in the manual.) If you should get the message 'Load Overlay?' you have selected an option which requires an overlay to be loaded so type [N ENTER] to get back to the main menu as we don't need these just yet.

The Edit Line

This is very similar to the editor provided by the INPUT command in BASIC, you can use the cursor LEFT and RIGHT keys to move through anything you type, and DELETE to delete the character to the left of the cursor as usual. EDIT must be pressed twice (or held down until it repeats) to clear anything you have typed and cursor DOWN pressed twice (or held down!) to abandon the current text (this is different to clearing the text as it will leave any text you may be editing unchanged in the database).

Free Memory

Option F on the main menu will show how much memory remains free in each page you have available (i.e. only page 0 will be displayed on a 48K spectrum.) In addition it also shows the highest location and message used so far, the reason for this wonderfully useful bit of information will be explained in more detail later. Pressing any key will return you to the main menu.

Saving, Verifying and Loading the database

Obviously you will not be able to finish typing in a game in only one session, and indeed you should not attempt to enter large quantities of information in one go as any interruption to the power supply (or fault in the computer) can cause the loss of a lot of hard work! These options allow just the contents of the database of information to be stored on tape in a 'file' (collection of information or data) and recalled at any time.

Selecting option S will request a filename to save the database under, this should be meaningful and usefully could contain a version number, e.g. DEM001. The database will then be saved in several parts (PAW saves two files per page, changing the last letter of the name to A,B,C etc for each file used).

Option H (Verify) will allow a database just saved to be checked against the one in memory - if a tape error should be reported then the database should be resaved with option S onto a different tape if necessary.

Finally option J will request the name of a database file to allow a database saved previously to be reloaded into PAW, overwriting any database already present. Should you get an error during the loading of a database, the database area will be 'corrupt' (i.e. not in the form PAW likes!) and the only safe option to use is J until a database is successfully loaded, any other options may cause damage to the PAW program itself necessitating a reload.

RULE: Save your database regularly onto different tapes so that you always have a reasonably up to date version should disaster strike.

Writing an adventure

Now the fun starts...

Planning

Planning your game is very important if you want to create a professional result, it is no use sitting at the machine and typing away in fits and starts as you wait for inspiration! You will merely entangle yourself in a maze of numbers and words with no recourse but to start from scratch anyway.

To illustrate the recommended approach to writing an adventure we will consider the design and development of a simple game from initial idea to final testing.

Remember to save your database regularly!!!

Getting an Idea

This is always the hardest part of creating anything! An original storyline can provide a game with an interest which rescuing a princess will probably not evoke in a modern adventurer.

Subjects for adventures are all around in many day to day actions, in exotic places around the world and out of this world!

If you decide to base a game on a book or film you have enjoyed and intend using it commercially make sure you have obtained permission from the original author or copyright owners.

For our sample we will use a simple problem which besets a passenger on his/her way home:-

While standing on the bus stop the passengers' ticket blows away in the breeze and is carried away by a small bird into an adjacent park, the computer will play the part of the passenger who you must direct to find the ticket before the bus arrives.

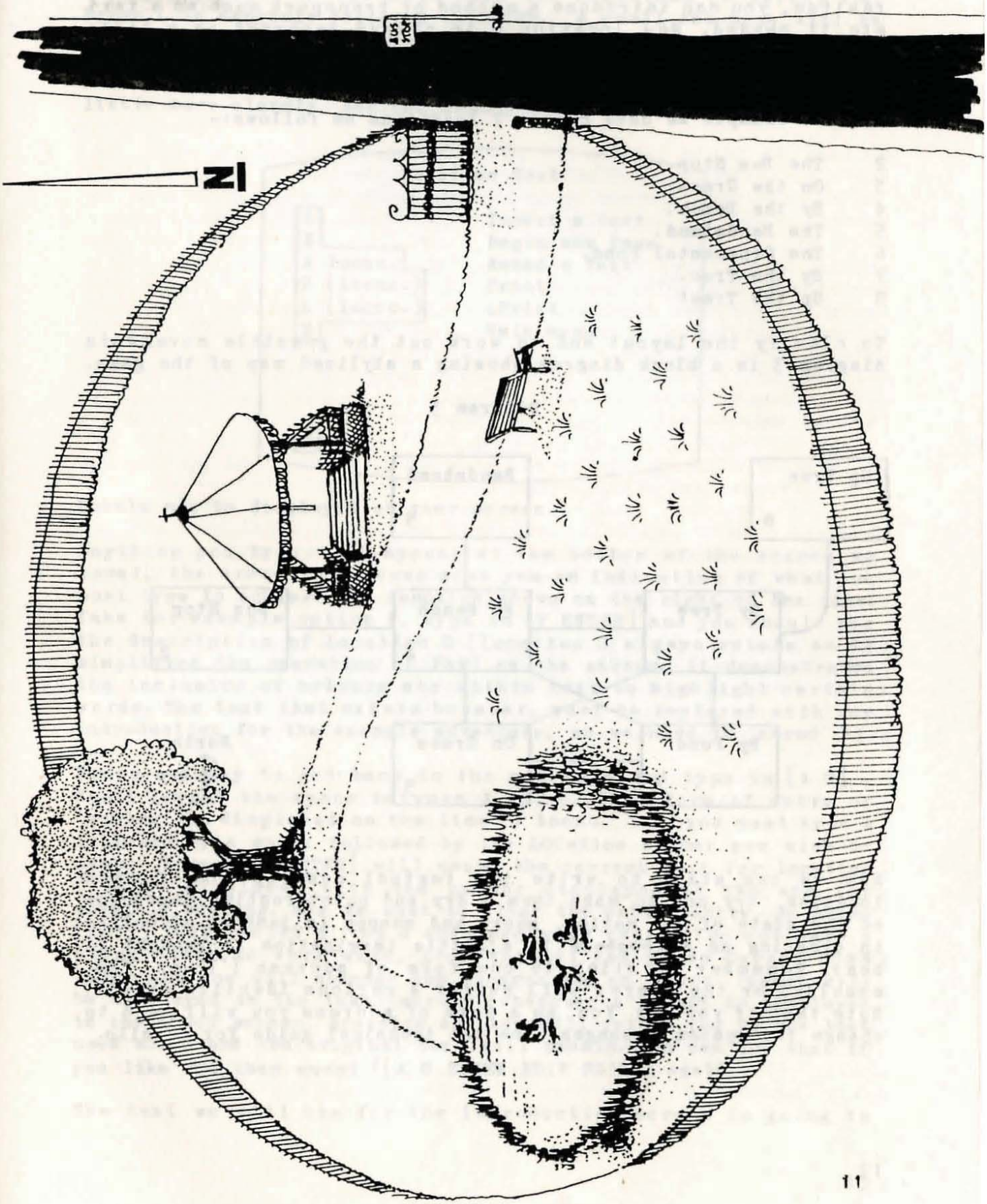
Game design

Now you have the idea, it is worth drawing a rough sketch of the area the game will take place within as we have done in diagram 2 (well actually our artist drew it...).

Note that any game design ought to be within a logically enclosed area or the player will wonder why they can't go in a direction when nothing appears to bar the way.

An adventure consists of a number of 'discrete' that is separate, 'locations' or places, the player can visit. You must now decide which areas can become a location and number them individually.

Diagram 2



Writing an adventure

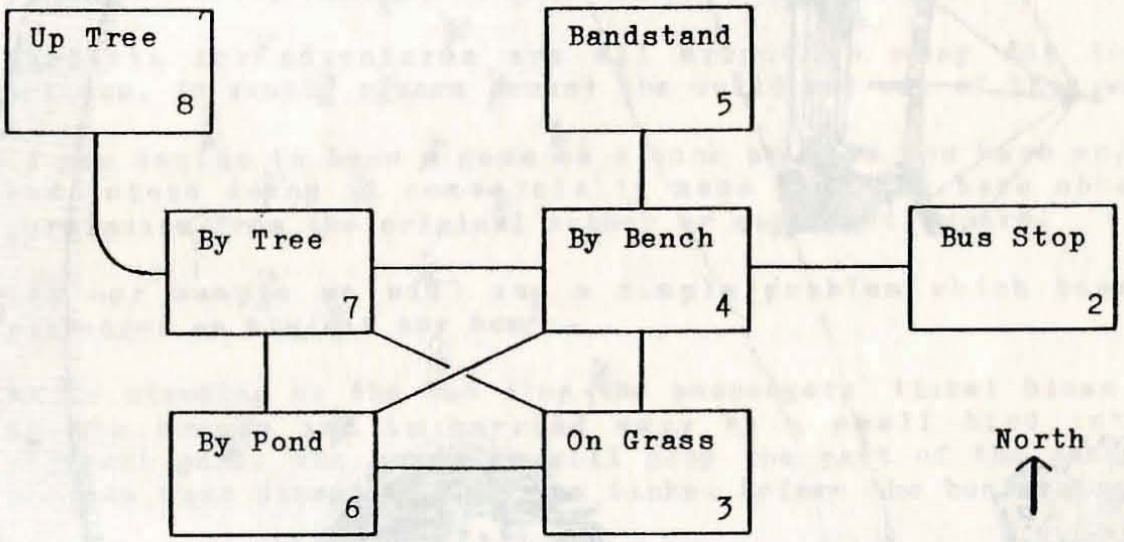
Try to make the scale consistent or logical (unless the game is illogical by intent!) as a single step to an airport earlier described as 10 miles away doesn't help the impression of realism, you can introduce a method of transport such as a taxi etc if needed. Now location 0 is always reserved as a title screen for a game and we want location 1 for a special use later so we number the locations from 2 upwards.

For our example we have chosen 7 locations as follows:-

2 The Bus Stop.
3 On the Grass.
4 By the Bench.
5 The Bandstand.
6 The Ornamental Pond.
7 By the Tree.
8 Up the Tree!

To clarify the layout and to work out the possible movements diagram 3 is a block diagram showing a stylized map of the game.

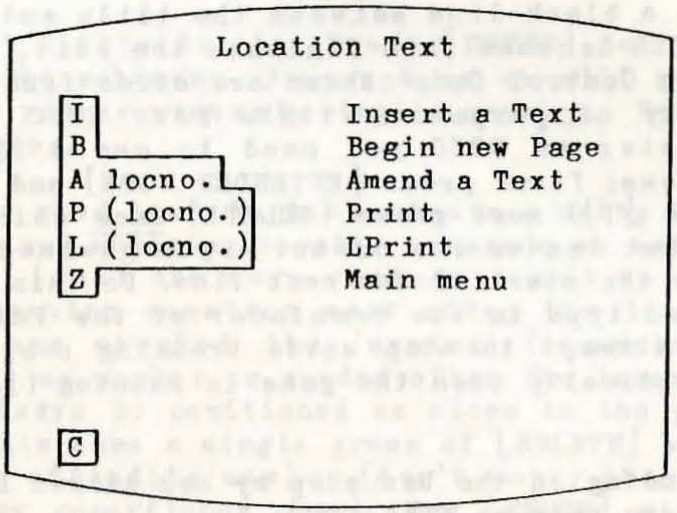
Diagram 3



Now we can start to write the textual description of each location, try not to make them a dry and uninteresting monologue on the state of the nation, short and snappy is just as effective in creating an atmosphere if a little imagination is brought to bear. Remember to stick to one form of address ('I' or 'You' usually) or the player will suffer a serious identity crisis. Note that if you use 'You' as a form of address you will need to change the system messages, see the technical guide for details.

Start Typing

Select option L on the main menu for Locations [L ENTER] i.e. Press L in capitals followed by ENTER. A small sub-menu will be displayed which shows the options available to deal with entering and amending location descriptions. All the menus in PAW are laid out in a similar way so we will examine these options a little more closely, see one menu you've seen them all!



Should now be displayed on your screen.

Anything you type will appear at the bottom of the screen as usual, the areas in inverse give you an indication of what you must type to achieve the function shown on the right of the line. Take for example option P, type in [P ENTER] and you should see the description of location 0 (location 0 always exists as it simplifies the operation of PAW) on the screen. It demonstrates the inclusion of colours etc within text to highlight certain words. The text that exists however, must be replaced with our introduction for the example adventure, so we need to 'amend' it.

Press any key to get back to the sub-menu and type in [A 0] - don't forget the space between A and 0. This form of entry or 'syntax' is displayed on the line 'A locno.' i.e. you must type A followed by a space followed by the LOCation Number you wish to amend. Pressing [ENTER] will cause the current text for location 0 to be displayed with the cursor displayed at the end for possible editing. Now we want to clear the entire entry, so press [EDIT] (CAPS SHIFT & 1 on a 48K spectrum) twice. You now have a blank entry to work with, note that if you press Cursor DOWN twice now you will abandon the changes and an 'error' report will be displayed in the lower screen (whenever a report is displayed in the lower screen, pressing any key will return you to the last used menu) and the original text will remain. You can try that if you like and then amend ([A 0 ENTER EDIT EDIT]) again.

The text we will use for the introduction screen is going to

include some colour to brighten up the title, so type eleven spaces [ ]- which will put the title in the centre of the line. Next press [EXTENDED MODE] (SYMBOL & CAPS SHIFT on a 48K spectrum). The keys from 0 to 7 will now provide the paper (or background) colours which are available, we want the title on a Red background so press [2]. Now type in the title which is [The Ticket]. Next the original paper colour must be restored (Black) so press [EXTENDED MODE] again and type [0].

We need to put a blank line between the title and the text, we can't press ENTER because this finishes the edit, so we need an 'Extended Screen Control Code' these are codes from 0 to 7 which serve a variety of purposes within PAW. ESCC 7 provides a newline, to enter an ESCC you need to use a 'quirk' in the editor, as follows; first press [EXTENDED MODE] and select colour White (i.e. type [7]) next press [DELETE] once which deletes the paper control but leaves the number - you guessed it 7! so the cursor jumps to the start of the next line. Do this again to give a blank line and type in the remainder of the introduction as follows, don't attempt to stop words breaking over lines as PAW does this automatically when the game is running (i.e. It formats the text):

> [While standing on the bus stop my bus ticket has been blown away, can you help me find it?]

When you have finished press [ENTER] to finish the edit and then anykey as requested to return to the sub-menu. You can use option P to look at your text if you like, but note that it is not yet formatted, this is done only while playing the game.

If you have a printer you might like to try option L which 'Line Prints' the text, if this causes the computer to 'hang' or appear to do nothing press [BREAK] (CAPS SHIFT & SPACE on a 48K Spectrum) to get back to the menu then refer to the technical guide for further information on printers.

Option I on the menu allows a new location to be created within the game, it has no number following it because PAW automatically assigns the next available location number. Now we wanted location 1 for a special purpose which we will explain later (good these secrets aren't they!) so just type [I ENTER], you will now have a blank location available for use so just press [ENTER] again (and any key) to return to the sub-menu.

Now type [I ENTER] again to 'insert' location 2, and type in the following text to describe it. Note that the spaces between "..a road.." and "..South. To.." will occur exactly at the start of new lines, due to the text formatting you must still type the spaces or the formatter will think the two words are one long word - the spaces will be 'suppressed' (removed) by the formatter where required when the text is printed during the game.

RULE: Always type the space between two words or between a full stop and the start of the next sentence, even if that space is at the start of a new screen line. It will be suppressed by the formatter if required.

The text for location 2 is as follows:

> [I'm standing by a bus stop, on a road which runs North to South. To the West a park gate set in iron railings stands open.]

Right got that little lot in. Press [ENTER] to finish the edit and return to the sub-menu. From now on we will omit the 'ENTER' assuming that you are remembering it anyway. Besides it saves our typing finger.

Now might be a good time to demonstrate a natty little feature which prevents you entering invalid commands on menus. Try typing in [I 3 ENTER] (well one more reminder won't hurt). You should now have a flashing question mark after the 3 along with the cursor. PAW has checked the 'syntax' (remember that?) and discovered that no number is needed after the Insert option. The cursor will always be positioned as close to the problem as PAW can get - in this case a single press of [DELETE] will suffice to get rid of the offending number (don't worry about the space as PAW ignores any superfluous ones). Now [ENTER] will provide you with a blank entry for location 3. The text required for this and the remaining locations is shown below, so using your new found knowledge type in the description then insert and enter the remainder of the locations. Note that the []'s have been omitted.

> Location 3
>
> The grass on which I stand is neatly trimmed. To the North is a path and bench while to the West is an ornamental pond.
>
> Location 4
>
> I am on a gravel path running East to West, by a park bench. To the South is a grassy area while to the North I can see a bandstand.
>
> Location 5
>
> I am standing on the bandstand which appears to be made of ornate cast iron painted white. To the South is a path.
>
> Location 6
>
> The sun glitters on the surface of the ornamental pond, whose waters ripple in the gentle breeze. A path runs North towards a large tree, while to the East is a grassy area.

Start Typing

Location 7

The path curves South and East here beside a large tree.

Location 8

I am sitting on a branch in a broad leaved tree, the park is spread out before me, to the East I can see the bus stop through the gate in the railings.

Use [P] to check what you have typed. when the screen fills up with text you will see "More..." printed in the lower part of the screen, pressing any key except BREAK, SPACE or N at this point will cause another screen full of text to appear and so on until the final 'Press any key' is reached. BREAK, SPACE or N will cause a 'BREAK' error and allow you to exit the listing.

Great now we have a collection of locations, but no way to get from one to the other!

That leaves two options to discuss; B is to begin a new memory page on 128K machines - do not do that at the moment as the demo game will fit easily on page 0. The option is dealt with in the chapter on 128K considerations in the technical guide.

So we use the last option, Z which it should be obvious takes us back to the main menu, so type [Z ENTER] (oops another ENTER!).

Connections

Select option C from the main menu, type [C], and you will be presented with a sub-menu again similar to that for locations. Notice that entries can be Amended (A), Printed (P) or Line Printed (L) only, the reason is that PAW inserts a blank entry in the connections table every time you insert a location. If you use option P to look at the table, type [P], you will see the blank entries for locations 0 to 8.

Looking back at our map we can see the interconnections required between the locations, for location 2 (the bus stop) we need a single movement WEST which will take the player to location 4.

So type [A 2] to amend the entry for location 2 and type [WEST 4] which instructs PAW that when the player types the word West when in location 2 they are to be moved to location 4!

If you now use option P to examine the entry (note that [P 2] will print only the entries from location 2 onwards) you will discover that it looks something like:-

```
Location 2     W    TO   4
etc
```

This is because PAW knows that 'W' is synonymous with 'WEST' (A synonym is a word which means the same) and PAW will always use the shortest synonym it knows when printing (indeed it will also use it if you amend the entry a second time).

For location 3 (the grass) we need three connections:-

NORTH to 4, WEST to 6 and NorthWest to 7.

'NW' is the word paw understands for NorthWest so amend the entry for location 3 by typing [A 3] and type in the following exactly as printed (again we have omitted the []'s and will do so in future for entries that are displayed on a line of their own):

NORT 4 WEST 6 NW 77

when you press [ENTER], PAW will place the flashing 'syntax marker' after 'NORT' because it does not know the word, add the [H] to make it 'NORTH' and press [ENTER] again, this time the marker will be displayed after the 77 because PAW does not know of a location 77 (in fact there are no locations higher than 8 yet in our game) this is another example of the syntax checker at work, it will usually prevent you entering silly or illegal information. So delete one of the sevens by pressing [DELETE] and press [ENTER] to complete the edit.

The remaining connections are as follows (in abbreviated form to save you some typing):-

Location 4	N 5	E 2	S 3	SW 6	W 7
Location 5	S 4	SW 7			
Location 6	N 7	NE 4	E 3		
Location 7	U 8	NE 5	E 4	SE 3	S 6
Location 8	D 7				

'D' and 'U' are short for DOWN and UP respectively to allow the player to go up and down the tree (SW is SouthWest, NE is NorthEast and SE is SouthEast!).

Finally amend the entry for location 0 so that a movement will take us to location 2 where we start the game (there is a better way than this which needs a table we haven't come across yet so is best left till later). Typing [A 0 ENTER NORTH 2 ENTER] from the sub-menu will achieve the required effect.

Check these thoroughly against the map and the list and when you are happy that they are correct, reward yourself with a cup of tea after saving the database for safety. Just to remind you; use option Z to return to the main menu and option S from there to save the current database (you can also use option E to see the other main menu if you want.).

Playing the Game

A suitable time has now arrived to try out the game. The option to test the game is T on the main menu, if you now select it, by typing [T], you will be asked whether you require diagnostics, for the moment just type [N ENTER] (oops, another one of those ENTERs) for no, as we don't know what diagnostics are, and indeed don't need them yet.

Now you should have the title screen you typed in earlier along with a request for input displayed. The input line is used to enter the commands for PAW to interpret into things to do, according to the information you have entered when writing your game. So far we have only told it about where to take us when a certain direction is entered, so try starting the game properly by typing [NORTH] (or whatever direction you used in the connection table entry for location 0) and pressing [ENTER]. (DELETE on the input line will allow you to correct any mistakes).

The screen will clear and the description for location 2 will appear - if it doesn't you probably have the entry in Connections wrong, don't worry you can go back to the editor by typing [QUIT] (which is a command PAW knows to start with) and replying [Y]; you do want to quit and [N]; you don't want to try again. Use the connections table option to check and amend the entry as necessary.

You can now try moving between your locations, testing the possible moves (make a note of any which are wrong so that you can correct them upon returning to the editor).

You might also like to try some of the other commands which PAW knows, e.g. R or REDESCRIBE will display the location description again - which is useful if a lot of text has been output and the description lost. I or INVENTORY will list the 'objects' you are carrying, you will be carrying one object to start with but will be able to do nothing with it.

You are probably dying to see the parser in action by now (why not?) so if you work your way back to the bus stop and enter the following line you will get a flying visit round the game!

GO WEST THEN NORTH THEN SW THEN UP AND DOWN THEN SOUTH
AND EAST THEN NORTH THEN EAST. INVENTORY

By the way [W.N.SW.U.D.S.E.N.E.I] will have the same effect but doesn't look half as impressive...

Right back to the boring bit, QUIT from the game as shown previously so that we can deal with the next chapter in this saga.

Objects

Objects are anything which the player can manipulate within the game, for example; An apple which they could eat, a key which they could use to unlock a door, or a rucksack to contain the key and the apple!

In our simple game we will have the following objects (not all of which have a function in the final game).

Object 0	A lit torch.
Object 1	A bag.
Object 2	A sandwich.
Object 3	An apple.
Object 4	A ticket.
Object 5	A lead.
Object 6	An anorak.
Object 7	An unlit torch.

Note that the torch is in fact two separate objects which we will swap over when the player turns it on and off.

Option O from the main menu as you might have guessed is used to enter the descriptions of the objects in a very similar way to how we entered our location descriptions, select the option by typing [O]. You might not be surprised either to find that an object O already exists if you use option P to list them. Enter the above descriptions - remember to use [A O] for the first one as it already exists - perhaps using a different Ink colour, say Cyan (selected by [EXTENDED MODE] and holding down [CAPS SHIFT] while pressing key [5]). Don't forget to turn the colour back to white (press [EXTENDED MODE] then [CAPS SHIFT] & [7]) at the end of each text.

Back to the main menu so that we can tell PAW all about our new objects. Option I allows us to define where each object will initially be when the adventure starts, so select the option and the sub-menu to deal with initial position of objects will be displayed, similarly to connections there is no option to insert as this is done by PAW automatically when you insert an object. Notice that Amend has two 'parameters' the object number and it's position and that this position has several special values (which are non existent locations); 252 is 'not-created', i.e. does not yet exist within the game. 253 is used to 'contain' objects worn by the player, while objects carried by the player are contained in location 254.

For example we want to make the lit torch a 'not-created' object (it was 'carried' in the start database) so type [A O 252] and the message "Amended" will be printed to show that PAW has completed the task.

Objects

The remainder of the initial positions are as follows, so amend each in turn but don't try and type in the comments. Use option P to ensure the positions are correct when you finish.

```
Object 1   2      ;the bag starts off at the bus stop
Object 2  254     ;the player is carrying the sandwich
Object 3  254     ;and the apple
Object 4   8      ;the ticket is up the tree..
Object 5   3      ;the lead on the grass.
Object 6  253     ;the player is wearing the anorak.
Object 7  254     ;and carrying the unlit torch.
```

Next we need to tell PAW some more about the objects, their relative weight, if they are capable of containing other objects and if the player can wear them! Type [Z] to return to the main menu and select option X (xtra info??), by typing [X], this selects the object weight menu which also allows us to set the other two object 'attributes' (container/wearable). You may not be surprised by now to find that PAW has created an entry in this table for all the objects inserted earlier. It in fact inserts an entry which makes all objects weigh 1 unit, not be a container and not be wearable this is known as an objects' 'default' attributes. PAW allocates default values to everything even if that entry is 'null' (nothing).

We need to amend only the anorak and bag attributes as the player is able to 'wear' the anorak (in fact the player is doing this when the game commences) and the bag will be able to 'contain' other objects. In addition the bag and anorak are 'heavier' relatively than the other objects and will have their default weight changed to 3 units each.

The A to amend an entry is followed by three values; the object number, the unit weight of that object and finally it's attribute options which are; 0-none, 1-a container for other objects, 2-the player may wear (and remove) it and 3-a container which may be worn (and removed), (e.g. a pair of jeans which have a pocket).

So for our bag (object number 1) which is a container weighing three units we need to type [A 1 3 1] (not forgetting the spaces). The anorak has no pockets (well ours hasn't anyway!) but it can be worn/removed so the entry is [A 6 3 2].

Now use option P to examine the new entries which should be:-

```
Object   0 weighs   1
Object   1 weighs   3  C          ;C means a container
Object   2 weighs   1
Object   3 weighs   1
Object   4 weighs   1
Object   5 weighs   1
Object   6 weighs   3     WR      ;WR means Wear or Remove
Object   7 weighs   1
```

You might like to test the adventure again now to ensure that all the objects are where they should be, but note you will not be able to do anything with them yet, we still have to tell PAW what word describes each object. (it may be a suitable time to do another save of the database now but ensure it is on a fresh section of tape and use the next filename number if you are numbering the versions.).

Vocabulary

This is a major table and to reflect this the menu is rather more complex than any you have met so far. Note that most of the text is merely reminding you of various options that are available and thus isn't really as complex as it looks at first.

The vocabulary is a list of all the words which PAW is able to recognize in any input the player types in during the game. Thus any words which aren't in this table will have no effect at all! Initially the vocabulary contains about 70 common English words which will be required for most adventures.

Each entry for a word consists of up to five letters which will either be a complete word e.g. NORTH or the first five letters of a longer word e.g. ASCEN(D). a word value and a word type (e.g. Noun, Verb etc).

The use of only five letters to store a word reduces the amount of memory required to store the entire vocabulary, the amount of typing the player must do and makes PAW faster at looking up words when required. Five letters is also more than adequate to differentiate the majority of important words in the English language from each other.

The menu allows the insertion and deletion of words, the listing of entries for each word type and the inspection of 'synonyms' which we met earlier when we found out that PAW knew 'W' meant the same as 'WEST'. Try [S WEST] (Show synonyms) to see indeed that PAW knows 'W' as well.

Now try [P 2] to look at all the Nouns that PAW knows to start with (the numbers which represent each word type are given on the right of the menu, type 2 are Nouns, type 0 are Verbs etc.).

You will find all the major compass directions, 'I' (Which even though it is short for Inventory is a Noun.) and 'ALL' - plus their synonyms.

We need to increase the number of Nouns by inserting a word for each of our objects, now the first free number appears to be 15, but Noun values less than 50 have special meanings thus;

Nouns less than 50 are Proper Nouns, for example peoples names or places, but more specifically for PAW they are Nouns which will

not affect the subject of 'it', take the sentence:-

GET THE SWORD AND CLEAN IT

It (a word known as a pronoun) refers to the SWORD obviously and PAW will (as long as SWORD is a Noun in the vocabulary with a value greater than 49) know this and assuming you have dealt with the possibility of cleaning the sword, allow you to do so. But take the following sentence:-

GET THE SWORD AND KILL THE ORC WITH IT THEN DROP IT.

Normally PAW assumes 'it' to be the last used Noun but as long as ORC is a Noun in the vocabulary with a word value less than 50 then PAW will remember 'it' as being the sword and carry out the action correctly. This feature is noted on the left of the menu along with mention of word values less than 20; these are Nouns, which if PAW cannot find a Verb in a 'phrase' containing one, will convert temporarily (i.e. it does not change the vocabulary) into a Verb. The major use of this is for things like NORTH which may be typed on their own implying GO NORTH which in normal English is invalid but is common when playing adventures.

Finally words less than 14 are assumed to be movement words (any word which is a direction) and merely determine the message which will be printed if PAW cannot do anything with the phrase it has found (i.e. it determines if "I can't" or "I can't go in that direction" is displayed). Note that this tag of 'less than 14 is a movement' applies to both Verbs and conversion Nouns.

Since all our objects are 'its' we must give them word values greater than 49 as follows:-

TORCH	50
BAG	51
SANDW(ICH)	52
APPLE	53
BUS	54
TICKE(T)	54
LEAD	55
ANORA(K)	56

note that there are two words with value 54, this makes BUS and TICKET synonymous so if the player types GET BUS TICKET or GET TICKET, PAW will know they mean the same thing.

Use the I option to insert these 8 words as Nouns (word type 2), for example TORCH is inserted by typing [I TORCH 50 2] and so on. Use [P 2] to check that the extra Nouns are now in the vocabulary when you have finished.

We also need some words to describe the difference between our two torches to PAW. The words which describe a Noun are called

Adjectives, you can see which adjectives PAW knows already using [P 3] (adjectives are word type 3), we need two extra adjectives LIT and UNLIT so insert these as word values 138 and 139 respectively using [I LIT 138 3] and [I UNLIT 139 3]. Note that the adjective numbers start high as they are not used as often in sentences and it is pointless to search through them all the time when PAW mostly needs common Nouns or Verbs.

All word values from 2 to 254 are available for each type of word and there is no limitation on the number of words with the same word value (synonyms) so the vocabulary can become quite large if you want.

Just to familiarize yourself with the other options, try inserting a word which is already present e.g. [I GET 20 0] will result in the message "GET is already present", deleting a word which isn't present [D BANANA] (we don't have an banana in our game or the word in the vocabulary) will result in "BANANA is not present". Note that PAW takes only upto the first five letters whenever you refer to a word and ignores the rest.

We will come back to the vocabulary fairly soon, but are going to tell PAW which words describe our objects first. We may have described what each object is and how much it weighs and even where it starts in the game, but we haven't actually told PAW which word in its vocabulary refers to which object!

Return to the main menu now so that we can continue with yet another option.

Object Worls

Option W is the table where the words in the vocabulary are linked to a particular object, again you can only Amend, Print or LPrint the entries in the table as PAW inserts a blank entry for each object when you insert its description on the object text option. So [P] will reveal 8 blank entries for our objects.

The object word table allows both a Noun and an Adjective to be associated with each object number in the game. Our objects require the following entries:-

```
Object 0  TORCH  LIT
Object 1  BAG    _
Object 2  SANDW  _
Object 3  APPLE  _
Object 4  TICKE  _
Object 5  LEAD   _
Object 6  ANORA  _
Object 7  TORCH  UNLIT
```

This introduces a special word "_" (underscore or underline depending on your colonial bias) which means (in this case) no

word ("_" is [SYMBOL SHIFT] & [0]). You must always type it in if there is no adjective to describe the Noun. So for example the lit torch and the bag are amended using [A TORCH LIT] and [A BAG _] respectively (they are in reverse order to simplify their use within PAW). Amend these and the other objects entries now so that we can have a play again.

Play it again...?

This time we will examine the use of diagnostics, so from the main menu type [T] to select test game and [Y] to request diagnostics. The title and introduction will appear again along with a request for input. The request for diagnostics has apparently had no effect, but if you now press [ENTER] before you type anything in you should find the cursor dissapear and a line similar to:

Flag 38= 0 ?

appear in the bottom of the screen followed by a flashing cursor.

PAW contains 256 of what are known as 'flags', each flag can be used to contain a number from 0 to 255 and are used to indicate (or flag!) the state of some part of the game. e.g. You could decide that flag 11 when set to 1 meant that the park gate was closed and when set to 0 meant it was open, we will see examples of the way flags can be set and used in the next section.

PAW has set aside several of the flags to indicate specific things (flags 0 to 10 and 29 to 59 actually). The value displayed on the bottom of the screen is the contents of flag 38 which PAW knows is your current location (0 in this case). To see this go back to the input prompt by pressing [ENTER] (ENTER 'toggles' between diagnostics and input if you haven't typed anything else) and move to the start of the game properly using [NORTH] (or whichever direction you used in the connections table for location 0). Again before typing anything on the new input line press [ENTER] to get diagnostics and the line:

Flag 38= 2 ?

should be displayed, because you are now at location 2 (the bus stop). You can look at the values of other flags by typing their number before pressing ENTER, try [100 ENTER] to look at flag 100, which will display:

Flag 100= 0 ?

A very powerful feature allows you to set the value of a flag by putting = in front of the number, try [=10 ENTER] and the line should be redisplayed as:

Flag 100= 10 ?

Flag 100 does nothing in our game and its value is unimportant, but if you decide to practice on your own do not change the values of any other flags for the moment or you may get some funny effects if you happen on a flag which is important. Return to the input line when you have finished (press [ENTER]) so that we can see what else PAW can do.

We should now be able to manipulate the objects in the game, at the moment the bag will be at the bus stop with us, we will be carrying the sandwich, apple, unlit torch and wearing the anorak.

Use the diagnostics to look at the value in flag 1 ([ENTER 1 ENTER]) this has the value 3, which is the number of objects carried but not worn, return to the input line and type [GET BAG], PAW will print the message "I now have the bag.", which is known as auto-reporting (PAW automatically reports any action it has carried out). This command has caused the current position of the bag to be changed from location 2 (the bus stop) to 'location' 254 (carried), note that no change has occurred to the initially at table in the database only to a copy which was made when the game started. If you look at the value of flag 1 again (notice how the flag you looked at last is displayed when you reselect diagnostics) you should find it has been increased to 4.

Now try [REMOVE ANORAK] and the report "I can't remove the anorak, my hands are full" will be printed. This is because PAW initially (by default in other words) allows the player to carry only four objects at any one time, this logically must prevent the player from taking off clothing etc (actually removing is changing an objects position from location 254 to location 253!) Try [DROP BAG] and then [REMOVE ANORAK] again, this time you should be able to do so. Look again at flag 1 and you should discover it is still four - this is because removing the anorak has increased the number of things you have in your 'hands'.

Try the following and see if you can work out why they do what they do:

GET BAG

REMOVE ANORAK

WEAR ANORAK

GET APPLE

GET TICKET

Notice that all except the last report actually mentioned the objects by name, this is because they were in plain sight and thus the player would know they existed. But to a player who did not know the game, the ticket has not yet been found and to

mention it by name would imply that it existed or that there was only one in the whole game thus giving a clue!

If you try to put anything in the bag you will discover that PAW drops that object instead. This is because we haven't yet told PAW **what** can be put in the bag only that it is a container, the next chapter deals with this subject.

Finally we will find a 'bug' in our game; type [GET GATE] which will result in "There isn't one of those here". It shouldn't say that because the description says there is a gate here!

The problem arises because although we told PAW about the apple the sandwich, the torch and so on, we didn't tell it about the gate, if you use GET (or DROP,WEAR and REMOVE) with any word which is not in the vocabulary then PAW assumes it is an object which is 'not here'. Of course once the word is in the vocabulary, PAW will know it isn't an object (if there is no entry for the word in the object word table) and report "I can't do that." which is correct.

So go back to the editor main menu ([QUIT ENTER Y ENTER N ENTER]) and select the vocabulary option [V]. The extra Nouns we require are as follows:

GATE	57
RAILI(NGS)	58
GRASS	59
PATH	60
BENCH	61
POND	62
BANDS(TAND)	63
IRON	63
TREE	64
BRANC(H)	64
LEAF	64

Notice how all the ways the player can refer to the tree are catered for, we have no intention of allowing the manipulation of leaves or the branch, but if you did you would need to give them separate word values - this is an important design consideration.

You might like to test the game again to ensure that GET GATE does indeed produce the correct response.

We have now dealt with; creating locations and connecting them together, creating and describing objects, assigning them a word from the vocabulary, a starting point in the game, a relative weight, flagging if they are wearable (and removable) and if they are a container. The next chapter goes on to create problems and characters to make the game world a more interesting place by allowing the player to do things!

Process & Response

We now come to the section of PAW which allows the problems and characters in the game to be created.

The Response table

The response table is option R on the main menu and is a special form of what PAW terms a process table. A process table can be thought of as a simple sequential (it does each command in turn) programming language, the commands which are carried out are called 'CondActs' because they can be divided mainly into two groups; Conditions and Actions.

Earlier we mentioned that the parser in PAW breaks sentences down into phrases, which are then organized into what is known as a LS (logical sentence). In the case of directions like NORTH (which are LS's on their own) it uses the connections table to discover where (if at all) it should move the player to. Before it does that however it carries out a check against the response table to see if that table contains an entry which can deal with the LS, i.e. give a response to part of/entire command the player originally typed.

Every possible phrase the player types and therefore every LS that your game will respond to, will have a corresponding entry in the response table, except for most movements which you set in the connections table.

The most important part of a LS is the Verb, this shows the purpose of the LS, next most important is the first Noun which shows the subject of the LS; e.g. GET APPLE, GET is the purpose and APPLE is the subject.

If you now select the response table option from the main menu by typing [R] you will be presented with the sub-menu to deal with this table.

Type [P] to look at the table. For the moment ignore the other entries and consider the first entry only:

```
I    _    INVEN
```

the two words indicate the Verb and Noun respectively of the LS that this entry can deal with. Now I is a conversion noun (as we saw in the section on vocabulary) which means if it is the only word the player types in a phrase, it will become the Verb for the LS. The underline (_) indicates that the Noun is not important in this entry - a bit like 'no word' in the object word table. What this means in simple terms is that if the player types I on its own PAW will match it up with the first entry in the response table and carry out that entry as described next.

Process & Response

In order to carry out the entry, PAW will execute each of the condacts (commands) in the list which follows. Now the first entry contains only one condact;

INVEN is an action (the act part of the word condact!). It is an action because is carries out the act of listing the objects the player is carrying and wearing on the screen, you do not need to worry how INVEN does this it just does.

When you typed I (or INVENTORY which is synonymous remember) during testing the game, it was this entry in response that caused something to happen because a logical sentence of "I _" was created by the parser, which PAW then found matched the first entry in the response table.

INVEN once it has listed any objects you are carrying, instructs PAW it has 'done' something, when PAW discovers this it asks the parser for another LS, which the parser provides by decoding the next phrase in the players input, PAW gets this LS and checks it against the entries in response and so on. this 'loop' is shown in diagram 4 in the form of a flowchart which you should follow from the box marked 'start'. The loop is slightly more complex than the diagram might lead you to believe and a complete one is given in the technical guide, but diagram 4 will do for now.

We advise you reread the above paragraphs and study the diagram until you are happy with the way PAW operates on LS's before proceeding.

Let's consider the second entry in response:

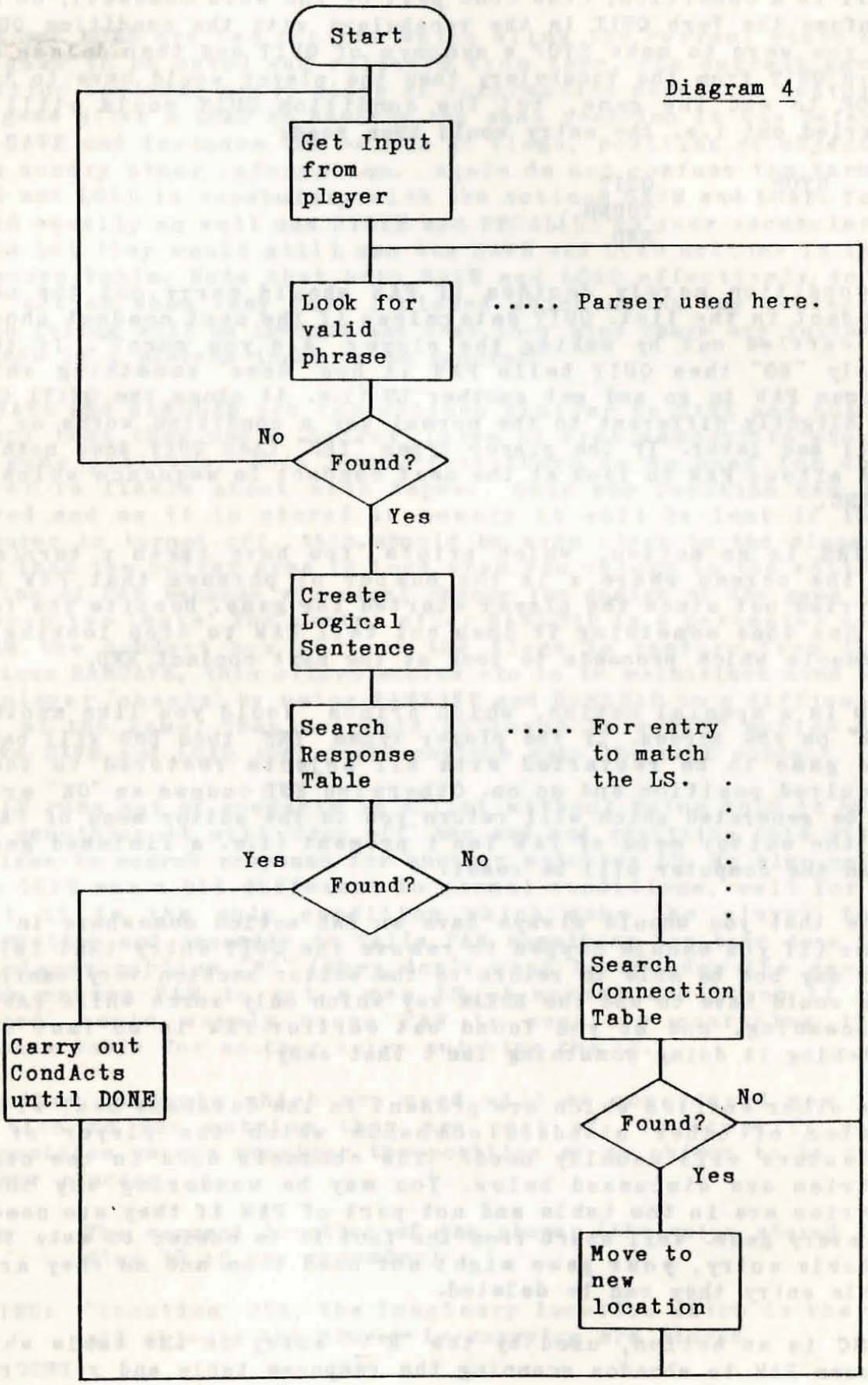
```
GET   I    INVEN
```

as you might have worked out this entry deals with the phrase TAKE INVENTORY (GET is a synonym of TAKE, I is a synonym of INVENTORY and PAW always prints the shortest synonym) this deals with another way the player might request a list of the objects he has with him.

We will skip the next few entries (you will have to press a key - except for BREAK, SPACE or N - to get the next screen full of text at this point) and move onto:

```
QUIT  _    QUIT
           TURNS
           END
```

Now, QUIT is a Verb in the vocabulary, so, as the minimum valid phrase is a Verb, if QUIT is typed on its own by the player then the parser will generate a LS of "QUIT _", on searching through the response table PAW will find the above entry and start to carry out the condacts which follow;


Start
Diagram 4
Get Input from player
Look for valid phrase
..... Parser used here.
No
Found?
Yes
Create Logical Sentence
Search Response Table
..... For entry to match the LS.
Yes
No
Found?
Search Connection Table
Carry out CondActs until DONE
No
Found?
Yes
Move to new location

QUIT is a condition, (the cond part of the word condact), do not confuse the Verb QUIT in the vocabulary with the condition QUIT, if you were to make STOP a synonym of QUIT and then delete the word QUIT from the vocabulary then the player would have to type STOP to end the game, but the condition QUIT would still be carried out i.e. the entry would then read:

```
STOP _      QUIT
            TURNS
            END
```

A condition merely decides if PAW should carry out the next condact in the list. QUIT determines if the next condact should be carried out by asking the player "Are you sure?". If they reply "NO" then QUIT tells PAW it has 'done' something which causes PAW to go and get another LS (i.e. it stops the QUIT) this is slightly different to the normal way a condition works as you will see later. If the player types "YES" then QUIT does nothing and allows PAW to look at the next condact in sequence which is TURNS.

TURNS is an action, which prints "You have taken x turn(s)." on the screen where x is the number of phrases that PAW has carried out since the player started the game. Despite the fact it has done something it does not tell PAW to stop looking at condacts which proceeds to look at the next condact END.

END is a special action, which prints "Would you like another go?" on the screen. If the player types "YES" then END will cause the game to be restarted with all objects restored to their required position and so on. Otherwise END causes an "OK" error to be generated which will return you to the editor menu of PAW. If the editor menu of PAW isn't present (i.e. a finished game) then the computer will be reset.

Note that you should always have an END action somewhere in the game (if you should happen to remove the QUIT entry that is) or you may not be able to return to the editor section very easily - you would have to use the BREAK key which only works while PAW is processing, and as you found out earlier PAW is so fast that catching it doing something isn't that easy!

The other entries which are present in the database deal with a number of other standard commands which the player of an adventure will usually need. The condacts used in the other entries are discussed below. You may be wondering why these entries are in the table and not part of PAW if they are needed in every game. Well apart from the fact it is easier to make them a table entry, **your** game might not need them and as they are a table entry they can be deleted.

DESC is an action, used by the "R _" entry in the table which causes PAW to abandon scanning the response table and reDESCribe

the current location of the player.

SAVE and LOAD are two actions which allow the current state of the game to be saved and reloaded from tape, the current game position includes every piece of information needed to restore the game after a LOAD to exactly the same position it was before the SAVE and includes the values of flags, position of objects plus sundry other information. Again do not confuse the Verbs SAVE and LOAD in vocabulary with the actions SAVE and LOAD. You could equally as well use STORE and RECAL(L) as your vocabulary Verbs but they would still use the SAVE and LOAD actions in the response table. Note that both SAVE and LOAD effectively do a DESC action when they have finished which means any condacts which follow will be ignored and that they also cause any further phrases in a players input to be ignored.

RAMSAVE and RAMLOAD are two actions similar to SAVE and LOAD, except that they use a 'buffer' (area of free memory) to store the game position, this means that there is no need for the player to fiddle about with tapes. Only one position can be stored and as it is stored in memory it will be lost if the computer is turned off, this should be made clear to the player. Note that the buffer area is lost when you return to the editor section of PAW because you might change the design of the game in between two tests! The number after RAMLOAD is a parameter and tells the condact how many of the flags to restore from the previous RAMSAVE, this allows scores etc to be maintained even if the player 'cheats' by using RAMSAVE and RAMLOAD in a difficult part of the game. They are both followed by a DESC action as unlike SAVE and LOAD they just continue onto the next condact.

If PAW runs out of condacts in a list without being told it has DONE something it will 'drop off' the end and realizing this will continue to search response for another matching LS. We also said that QUIT was a bit different to normal conditions, well for a start it is the only condition which asks the player for information and secondly it tells PAW something has been done if the player replies "NO" (they don't want to abandon the game) which causes PAW to get a new LS. A normal condition if it 'failed' would merely cause PAW to continue searching the response table for another entry matching the LS.

The other condacts which are used will be considered now in relation to the entries they are part of. To simplify our explanations we can consider the position of an object to be one of four places;

HERE: The current location of the player (the value stored in flag 38 if you remember).

CARRIED: 'location' 254, the imaginary location which is where all objects the player is carrying are stored.

WORN: 'location' 253, the imaginary location which is where all objects the player is wearing are stored.

NOTHERE: Anywhere else! This may also include 'location' 252 which is the imaginary location where any objects which do not yet 'exist' within the game are stored.

Take the following two entries in the response table:

```
GET   ALL   DOALL   255

GET   _     AUTOG
            DONE
```

these two entries allow the player to GET an object. GETting an object involves changing its location from HERE to CARRIED.

Ignoring the GET ALL for a moment let us look at the GET _ entry, as we said earlier underline means 'any word' so no matter what Noun the player types in, in the phrase containing the GET the GET _ entry will match (this is called triggering the entry). Take the phrase GET THE APPLE; THE will be ignored because it is not in PAW's vocabulary, so the LS will be "GET APPLE", this will 'trigger' the GET _ entry resulting in PAW looking at condact;

AUTOG is an action which AUTOmatically Gets the object specified by the Noun. This is where the object word table comes into effect, AUTOG looks through the object word table for an entry which matches the Noun in the LS, when it finds one (APPLE in the example) it then knows the number of the object it refers to (the apple is object 3), it then ensures that the current location of that object number is HERE and if so changes it to CARRIED and prints the message "I now have the _." where the underline is replaced with the description of the current object. i.e. the one AUTOG just looked up. If it does not succeed in finding an entry then there are five possibilities;

1/ The player has tried to get an object which they are already carrying or wearing in which case "I already have the _." is displayed.

2/ The player has tried to get an object which is NOTHERE in which case "There isn't one of those here." is displayed.

3/ The player has tried to get something which is not an object but does have a word in the vocabulary (e.g. GATE in the demo game) this results in "I can't do that.".

4/ The player has used a word which is not in the vocabulary which causes the parser to create a LS of "GET _" which triggers our GET _ entry anyway. AUTOG assumes this to be a Noun describing an object (which may or may not exist) and displays "There isn't one of those here.".

5/ The player is unable to carry any more objects or this object would cause the weight limit to be exceeded in which case a suitable message is displayed.

If AUTOG succeeds then PAW looks at the next condact DONE;

DONE merely tells PAW that this entry is finished and it should go and get another LS.

Next we will look at the GET ALL entry, you may have guessed what this does (it attempts to GET all objects at the current location), so we shall explain the mechanism;

Should the player type the phrase GET ALL, the parser will create a logical sentence of "GET ALL", which will match the entry and cause PAW to look at the DOALL action;

DOALL is an action which is followed by a parameter which gives a location number to use. DOALL looks through the current location list for each object looking for entries that are at the same location as the parameter (which in this case is 255, a special location which means that the current location of the player should be used instead), when it finds one it looks in the object word table to find the vocabulary word which describes that object number, this is placed in the current LS (thus replacing the Noun ALL), a flag is set to indicate that DOALL is active and the rest of response is scanned by PAW for an entry which matches the newly modified LS. This will be the GET entry discussed earlier, which will GET that object. Once this has been done PAW will discover that DOALL is active and go back to the GET ALL entry (actually it goes direct to the DOALL action) and allows DOALL to look for another object which generates a new LS and so on for all objects at the specified location. When DOALL runs out of objects it resets the flag to show it is not active and tells PAW to get a new LS.

This may seem a rather roundabout way to approach this task, but if you examine the very similar DROP,WEAR and REMOVE entries you will see that the same mechanism is used to create all four commands. AUTOD, AUTOW and AUTOR work in a very similar way to AUTOG while DOALL merely uses location; 254 (CARRIED) as the parameter for DROP and WEAR (i.e. DOALL searches all the CARRIED objects when you try to DROP or WEAR ALL!) and 253 (WORN) when you try to REMOVE ALL.

If the above seemed a bit heavy going don't worry about it for now as DOALL is one of the two most complex condacts in PAW and hopefully the penny will drop as we continue. At this point you might like to use the test adventure option and try out the 'all' commands which may make the mechanism clearer.

Messages

Before we continue with the response table we will insert some entries in another table which will be needed. So return to the main menu and select option M for messages.

Messages should be a breath of fresh air after that discussion of the response table, the sub-menu provides options very similar to the location and object description menus, the purpose of messages is to contain all the text which will be displayed to describe what is happening in the game to the player, excluding the messages that PAW itself displays (like "I can't do that." etc). If you use option P you shouldn't be surprised to discover that an entry already exists.

We are going to deal with the player wanting to examine things in the game, e.g. EXAMINE APPLE. Now examining an object merely requires the writer to provide a message which gives more information about the specified object, so in the case of the apple we could say "The apple is crisp and green.". Change the text of message 0 to read that ([A 0 ENTER]), and then insert the following messages to deal with some of the other things in the game;

Message 1

It's a cheese and pickle sandwich.

Message 2

The ticket has "City Bus Company" printed on it.

Message 3

The bench is firmly screwed to a concrete base.

We are going to deal with only four items in the demo game but in a large game you would usually provide detail for most things, even if they serve no purpose it provides a touch of realism which always makes the player feel involved.

So back to the response table (option R from the main menu) and start work. Let's take the apple first; the phrase which the player will type will be EXAMINE THE APPLE (or EXAMI APPLE if they are lazy!) producing a LS of "EXAMI APPLE". So we need to insert an entry with these two words;

First type [I EXAMINE APPLE ENTER], PAW will ignore the extra letters and print the entry at the top of a clear screen and wait for you to type in the list of condacts for that entry. Now we must only allow the player to examine the apple if it is actually HERE, CARRIED or WORN (most normal people have a distinct disability at looking round corners or through walls!), this is

collectively known as present and can be checked for using the condition PRESENT followed by the number of the object we are considering (the apple is object 3). If the object is indeed present then we can display our message (0) which describes the object, using the action MESSAGE which is followed by the number of the message you want to display. Finally we top it off with a DONE action to tell PAW that we have completed the task.

So type [PRESENT 3 MESSAGE 0 DONE ENTER] which will result in the message "Inserted." being displayed by PAW. Now press a key to get back to the sub menu and use [P EXAMINE] to examine our new entry (the brackets about the parameters show they are optional, so you can use P to see the table from the start, P followed by a Verb to see entries from that Verb on, or P followed by a Verb and Noun to see all entries from that Verb and Noun on). The entry should look like;

```
LOOK  APPLE PRESENT   3
            MESSAGE   0
            DONE
```

PAW has found the synonym LOOK and printed it because it is shorter than EXAMI(NE). This is a classic example of a response table entry because if the condition PRESENT 3 fails then PAW will continue to look for an entry to match the LS, in this case it will find the next entry displayed which is LOOK _, this entry will trigger and describe the current location (the DESC action), after the PLUS action has added 128 to flag 29, which causes PAW to redraw any picture at the location as well as display its description this will be covered in more detail in the section on graphics. Assuming that the APPLE is indeed present then PAW will continue with the condacts and display our description of the apple (MESSAGE 0) the DONE action tells PAW to go and get another LS because we have done something - this prevents the LOOK _ entry triggering as well.

If the entry should be incorrect you can amend it by typing A LOOK APPLE, with process tables however there may be more than one entry with the same word values, these will be presented in turn for possible amending - just press ENTER to leave an entry as it is. To delete an entry entirely from the table remove all its condacts. i.e. Amend the entry and press EDIT twice to clear the buffer and then press ENTER.

So at the moment if the player tries to EXAMINE anything except the apple (or tries to EXAMINE APPLE when it isn't present) they will be rewarded with a fresh description of their current location. Let's insert the entries to deal with the sandwich, ticket and bench - the entries are listed as you would see them if you used P after they are typed in, along with some comments for your reference only, you must still enter them as you did the EXAMINE APPLE entry earlier.

Process & Response

```
LOOK  SANDW PRESENT  2      ;The sandwich is here
            MESSAGE  1      ;Describe it
            DONE

LOOK  TICKE PRESENT  4      ;The ticket is here
            MESSAGE  2
            DONE

LOOK  BENCH AT       4      ;The bench isn't an object
            MESSAGE  3      ;so check location
            DONE
```

AT is a condition which is followed by a location number which will succeed (i.e. allow PAW to continue onto the next condact) if the player is at the same location, this was used because the bench was not an object, but as it is part of the description for location 4 it will always be there!

Use test adventure to check that you can examine these four items correctly and that the location is described at any other time.

The Process Tables

We shall now turn our attention to the most powerful writing option on the main menu; the Process table.

It was stated earlier that the response table was a special form of process table, and indeed it is, if you select option P from the main menu you will be presented with a similar sub-menu to that for response except it has two extra options. Note that the title says "Process 2", this is because there is more than one process table in PAW, indeed there can be upto 254 process tables as we shall see.

There are two process tables in the database to start with, just like response PAW scans through them, but, unlike response, it scans them not after obtaining a LS, but;

Process 1 is scanned immediately after PAW has described a location. This allows information to be printed only once when the player first arrives at a location or when he requests a redescribe.

Process 2 just before requesting a new LS from the parser. This is used to provide PAWs 'turn' at the game.

The main difference being that it does not attempt to match the LS against each entry looking for a match, it does every single one!

So far while playing our demo game we have had to end the game by typing QUIT. Now the original storyline (if you can remember that

far back!) was to help the passenger find the ticket before the bus arrived. Now we obviously could have an entry in response which if the player said GET TICKET (and it was present) could trigger the end of the game e.g.

```
GET   TICKE PRESENT   4        ;The ticket is here
            TURNS
            END                ;That's all folks!
```

but wouldn't it be much better to finish the game when the player gets back to the bus stop?

We shall do so, but, first we need a message to describe the arrival of the bus, so return to the main menu and select messages (option M) and insert the following message;

Message 4

The bus arrives. I hand the ticket to the driver who smiles and says "Sorry I'm late, hope you haven't been standing too long?".

Back we go to the Process menu. Now although the words have no meaning to PAW they can usefully be used as a comment on what the entry does, the entry that determines the end of the game will be called "_ BUS" (we must start with _ as PAW only allows the Noun BUS in the Noun position). So type [I _ BUS ENTER] to insert the entry (PAW has actually inserted a null entry now and if you press [CURSOR DOWN CURSOR DOWN] to abandon the entry, you will have to [A _ BUS ENTER] to complete the entry). The conditions for the end of the game are that the player is at the bus stop (location 2) and is carrying the ticket (object 4). The first condition of course will be AT 2, the other can be checked with CARRIED 4 (pretty unusual names these conditions have...) so the final entry will be;

AT 2 CARRIED 4 MESSAGE 4 TURNS END

pressing [ENTER] will complete the insert/amend. Use [P] to ensure the entry looks as follows;

```
_     BUS   AT        2
            CARRIED   4
            MESSAGE   4
            TURNS
            END
```

this entry will now be scanned just before PAW gets a new LS and as soon as both conditions are met the game will end independent of the commands the player uses to get to the bus stop with the ticket!

Select process 1 by typing [S 1 ENTER] and use [P] to examine the

entries that are present, they are;

```
*     _     NEWLINE
            ZERO      0
            ABSENT    0
            LISTOBJ

*     _     PRESENT   0
            LISTOBJ
```

The asterisk '*' is an 'any-word' word like '_' with a subtle difference; Whenever PAW inserts entries in a process table (including response) it inserts them in order of word value of the Verb and then the Noun (i.e. all entries dealing with one type of Verb will follow each other in ascending order of Noun value). PAW considers underline "_" to be a word of value 255 (it will always be the last entry) and asterisk "*" to be a word of value 1 (it will always be the first entry). The position of entries in process tables can be important for example the two entries shown must always be done soon after a location description has been printed so we use an asterisk to ensure they will be close to the start of the table (the use of underline as the Noun allows entries to be inserted before them as we will do in a moment).

Back to why these two entries are present (always getting sidetracked, so much to tell!). Because PAW does every entry in Process 1 and 2 (you might spot that it would do anyway even if not forced to as the * _ entries would match any LS the player typed!) the first action NEWLINE will always be executed;

NEWLINE prints spaces to the end of the current line as opposed to just starting a newline as CHR$(13) does on a Spectrum. This allows areas of Paper colour to continue to the end of the line without having to type the spaces. It's main purpose here is to ensure that any text displayed will be on a new line because PAW does not start one at the end of displaying a location description, the technical guide shows how to use this to good effect to modify the location description to reflect changes in the location.

From now on the two entries must be considered as a pair, their ultimate purpose is to list the objects at the current location, first a bit of background information;

PAW uses flag zero to determine if there is light for the player to see by (this feature is not used at the moment in our demo game), if there is no light the flag will have a value other than zero and PAW will say "It's too dark to see anything." instead of the description for the location. In this case the objects that are present must not be listed.

Object 0 is assumed by PAW to be an object which provides light

which is why object 0 in our demo is lit torch. If this is present while the game is 'dark' (flag 0 is non zero) then it will override the darkness and so the objects must be described.

The two entries provide an example of using PAW to create an OR situation i.e List the objects if it is light OR if object zero is present;

ZERO is the first condition we have met which tests the state of a flag. ZERO 0 will succeed if flag zero contains 0 which means there is light.

ABSENT ensures that Object 0 is not present (opposite of PRESENT condition - all conditions have an opposite, e.g. AT has an opposite of NOTAT and so on.), the next * _ entry lists the objects if object 0 (the source of light) is present so we do not want this entry to succeed as well (i.e. This deals with the situation of it being light and object 0 being present which would otherwise list the objects twice!).

LISTOBJ lists any objects that are present at the players current location, if none are present it does nothing - it would look a bit silly saying "I can also see nothing."!

Think about the above as it represents a fairly useful feature of PAW which you may well need to adapt for use in your own games.

Right, now we shall reveal the better way of getting from the introduction screen to the start of the game at the bus stop:

```
*    *    AT      0
          ANYKEY
          GOTO    2
          DESC
```

Insert this into Process 1 (ensure you still have it selected) using [I * * ENTER] and [AT 0 ANYKEY GOTO 2 DESC]. This uses two new condacts ANYKEY and GOTO which are both actions;

ANYKEY prints "Press any key to continue." in the bottom section of the screen and waits for you to press a key, it then allows PAW to continue onto the next condact.

GOTO is followed by a location number and moves the player to that location, it effectively sets flag 38 (players current location) to the value given, it does nothing else so it is followed by a DESCribe to get PAW to display the new description.

This entry thus causes the title screen to be displayed (when PAW displays the first location description), a wait for a key and then the game itself is started at the correct location.

You might like to go to the connections table and remove the

entry for NORTH in location 0 as this is not needed now.

Use test game to see the above two entries in action. The following input while at location 2 (the bus stop) will 'solve' the game in one go:

GO WEST, WEST AND UP. GET THE TICKET. GO DOWN,EAST AND EAST.

You should then get the finishing message and an option to play again. If not check the entries in Process tables 1 and 2 thoroughly.

Let's have a break and go back to deal with the ability of the bag to contain objects. Thought we had forgotten about that, didn't you? Well we nearly did. This will require some entries in the response table and we are going to allow the player to LOOK IN BAG, so we need a new message "In the bag is:", select the messages option and insert this (it should be message 5) then select the response table. We are going to provide the player with the option of saying PUT ALL IN BAG as well as PUT object IN BAG. We can use exactly the same system as GET/DROP ALL discussed earlier. PUT is a synonym of DROP (which takes care of DROP TICKET IN BAG and such similar phrases), so the LS we must check for will be PUT _ (i.e. player is trying to put or drop something), now if the player includes IN BAG as part of the phrase we want PAW to put the object in the BAG. This means we must override the PUT _ entry already present, and if the extra words are included in the LS put the specified object in the bag. To insert this entry <u>before</u> the one already present requires the use of an extra option. Normally PAW would insert another entry with the same word values <u>after</u> any already present, it is possible to force this by specifying a number after the insert, try [I PUT _ 0 ENTER], this instructs PAW to place the entry before entry number 1 (which is the existing PUT _ entry). Now the condacts we need are:

PREP IN NOUN2 BAG PRESENT 1 AUTOP 1 DONE

This shows how we check for an extended LS (i.e. ensuring certain parts of the phrase were what we need).

PREP is a condition which is followed by a preposition from the vocabulary. Prepositions are words used before a Noun to show its relation to another word in the phrase, in this case the condition will succeed if the player has used IN as part of the phrase.

NOUN2 is a condition which is followed by a Noun from the vocabulary. This will succeed if the player has used BAG in the phrase. Combined with the previous entry it effectively stops PAW looking at the condacts unless the LS was PUT _ IN BAG where the underline is any object.

AUTOP is followed by a location number. Now we set aside location 1 for a special purpose early on in the tutorial, this is it, it is used as the inside of the bag! So AUTOP just like AUTOD scans the object word table for a Noun which matches the current first Noun in the LS, when it has found one it places it at the location given, reporting "I have put the _ in the bag."!

The DROP ALL entry which exists will also work to deal with PUT ALL IN BAG, because it does not ensure that IN BAG is part of the LS and will trigger on both occasions, and in both cases 254 is the location the objects will be coming from.

Now for a GET object OUT OF BAG type command we need an entry similar to the above to override the GET _ entry which is present so insert the following using [I GET _ O ENTER]:

```
PREP OUT NOUN2 BAG PRESENT 1 AUTOT 1 DONE
```

AUTOT is followed by a location number which shows where the object to TAKEOUT will come from.

The implementation of an ALL version of the command needs an entry of its own, at the moment GET ALL causes a DOALL 255 which is the current position of the player to be carried out, in order to get all from the bag we need to generate all the objects that are inside it (location 1), so insert a GET ALL entry to override the existing one thus; [I GET ALL O ENTER]:

```
PREP OUT NOUN2 BAG DOALL 1
```

Before you test the game we will insert the entry that allows the player to LOOK IN THE BAG the entry needed is as follows (note it will be positioned in a suitable place anyway so there is no need to specify a number after it when inserting).

```
LOOK  BAG   PREP     IN
            MESSAGE   5
            LISTAT    1
            DONE
```

LISTAT is followed by a location number and lists any objects present at that location, note that if no objects are present it will print "nothing." so the above would result in:

```
In the bag is:
nothing.
```

which is correct, unlike LISTOBJ which because of its main use does not print anything at all in that situation.

So use test adventure to ensure that you can indeed PUT ALL IN BAG and GET object OUT OF BAG etc.

The Bird

The tutorial game is a little bit simple to solve so we shall add some complexity in the form of puzzles by creating two characters to wander round our little world. These are termed 'Pseudo-Intelligences' (PSIs for short) because they cannot obviously think, but must appear to do so to the player. A PSI consists mainly of a collection of messages, flags and process table entries, but even a few simple entries can create a surprisingly realistic effect. Creating a complex PSI of say a human can take a fair bit of thought, but follows the same principles as we will take with our two PSIs; a bird and a dog.

The bird will complicate the scenario as follows; The bird will have the ticket at the start of the game (normally you would assign an unused location to contain the birds objects, but we will use location 252 - object does not exist in game - as we have only one PSI that can have an object). This means you must persuade the bird to drop the ticket, trying to GET it will result in an "I can't do that" message and the bird flying away. The bird also flies between the Bandstand and the Tree Branch at regular intervals. The way to get the bird to drop the ticket will be to drop the sandwich at the same location. So lets get that little lot working first.

To save flicking back and forth between tables change Object 4 to be not-created using [I] Initially at now. This makes the ticket a does not exist object which we are using to indicate it is in the birds beak. Insert the Nouns DOG and BIRD in the vocabulary with word values 21 and 22 respectively - ensure DOG is 21 and BIRD is 22 as their word values will be used to position the entries in process correctly. Then insert the following messages which will be needed. Make them use green ink (EXTENDED MODE, CAPS SHIFT & 4) - not forgetting to reset ink white (EXTENDED MODE, CAPS SHIFT & 7) at the end of each one.

Message 6

The bird drops the ticket to peck at the sandwich.

Message 7

The bird snatches the ticket.

Message 8

The bird ignores me.

Message 9

A small bird is here.

Message 10

The bird has a ticket in its beak.

Message 11

A small bird settles on the ground.

Message 12

A small bird lands on the branch.

Message 13

The bird sees the dog and flutters away quickly.

Message 14

The bird flies away.

We will insert the messages to deal with the dog later. So select the Process tables option and get ready for an ear bending on yet another feature of PAW.

In a large game which contains several PSIs and a lot of background action, Process tables 1 and 2 soon become so full of entries it is nigh on impossible to work out what they do. Enter stage left the other process tables to the rescue, these can be 'called' from Process 1,2 or Response and used as an extension of the table they are called from. Calling a process causes PAW to save where it is at the moment and shift the action to the indicated table. i.e. if called from Response PAW will try and match the LS against each entry and if called from Process 1 or 2 PAW will do each entry. Note that when something is DONE in the called process, then PAW will still shift back to the original table, so some very powerful things can be achieved with thoughtful use of these sub-process'. Users who program in other languages will recognize this as a 'subroutine'.

While PAW is in a sub-process it is quite possible for it to be asked to call yet another sub-process - a sub-sub-process? and so on down to a sub-sub-sub-sub-sub-sub-sub-sub-sub-sub-process! That is 10 levels of subroutine calls may be carried out, this is called 'nesting' a call, attempts to go further will result in an error "Limit Reached" although you will be provided with diagnostic information as well if you are testing the game from the PAW's editor section - see technical guide for details.

We are not going to use anything like that here, only a sedate sub-process. This will contain all the entries to deal with the birds activities.

The Bird

Use [B] to begin a new process table, PAW will allocate the next available number as in other menu options. You should be starting process 3. As there will only be a few entries in the table we will use the same word pair ("_ BIRD") although when writing your own games you may find using other words useful to remind you of each entries function.

Flag 11 will be a 'working' flag to contain a value for use in a comparison.

Flag 12 will contain the current location number of the bird.

Flag 5 is a special flag which if it has a value other than zero PAW will reduce by one whenever it scans process 2 - this is called an auto decrement flag. In this case it is used to count the number of 'time frames' that have passed in the game, a time frame is a single time round the big loop shown in diagram 4, and at the moment this is done before every phrase the player types. The bird will change location every three phrases on behalf of the player which will create the appearance of action in the game independent of the players input.

Now insert the following entries, without the comments as before, each entry is preceded by an explanation of its purpose and any new condacts it uses:

First determine if the bird is going to fly away this time through the table, this is indicated by flag five being zero (as it counts down from 3), if the ticket is at the same location as the bird it will be destroyed (i.e. put at location 252 so the bird 'has' it) and if the player is at the same location as the bird they will be told that the bird has snatched the ticket. Note that the bird will continue its cycle of movement even if the player does not see it, a tree certainly does fall even if there is no one to see it in PAW!

```
-     BIRD  COPYOF    4   11  ;Copy location of object 4
                                (ticket) to flag 11.
            SAME      11  12  ;and see if it is at the
                                same location as the bird.
            ZERO      5       ;Bird going to fly?
            DESTROY   4       ;Bird 'GETS' the ticket
            SAME      12  38  ;Bird at same location as
                              ;player?
            MESSAGE   7       ;Tell player about it.
```

Note there is no DONE action as we want PAW to do each entry in turn, the above entry shows how conditions and actions can be mixed together to create new conditions.

COPYOF is an action followed by an object number and a flag, it copies the current location of the specified object to the specified flag. We use it in this situation to see if the ticket

is at the same location as the bird by following it with;

SAME is a condition which compares the contents of the two flags and succeeds if they are the same.

DESTROY is an action which places the specified object at location 252, the not-created location.

Now deal with the two possible movements of the bird. If the bird is at the bandstand and flag five has reached zero then move the bird, set flag 5 to 3 again and tell the player the bird is gone if they were at the same location. Vice Versa if the bird is on the branch.

```
_       BIRD  EQ       12   8  ;Bird on branch?
              ZERO     5      ;Time to fly?
              LET     12   5  ;Move bird to bandstand
              LET      5   3  ;Three phrases 'till move
              AT       8      ;Player here as well?
              MESSAGE 14      ;tell them bird has flown

_       BIRD  EQ       12   5  ;Bird on bandstand
              ZERO     5      ;Time to fly?
              LET     12   8  ;Move to branch
              LET      5   3  ;Three phrases 'till move
              AT       5      ;Player here as well?
              MESSAGE 14      ;tell them...
```

EQ is a condition which is followed by a flag number and a value and will succeed if the flag contains the value, in this case it is checking if the bird is at a specific location.

LET is an action which is followed by a flag and a value. It sets the flag to the value.

Now we have dealt with the birds departure, next we must deal with its arrival, and if it arrives in a location which contains the player tell them about it.

```
_       BIRD  EQ        5   3  ;Bird just flown?
              SAME     12  38  ;Now at players location?
              AT        5      ;On bandstand?
              MESSAGE  11      ;landed on ground

_       BIRD  EQ        5   3  ;Bird just flown?
              SAME     12  38  ;Now at players location?
              AT        8      ;On branch?
              MESSAGE  12      ;landed on branch
```

Now if the bird has the ticket in its beak we must tell the player.

```
_       BIRD  EQ        5   3
              SAME     12  38
              ISAT      4 252  ;Ticket not-created?
              MESSAGE  10      ;Has a ticket in beak..
```

ISAT is a condition followed by an object and a location number and succeeds if the object is at the specified location.

Finally if the sandwich is at the same location as the bird the bird will drop the ticket to peck at the sandwich. This entry does not rely on flag 5 so it will be checked for every time PAW checks process 2, so even if the player drops the sandwich after the bird has arrived the correct sequence will still be carried out.

```
_       BIRD  COPYOF     2  11  ;Sandwich
              SAME      11  12  ;at same location as bird?
              ISAT       4 252  ;Ticket in beak?
              COPYFO    12   4  ;Put ticket down
              SAME      12  38  ;Player here as well?
              MESSAGE    6      ;tell them...
```

COPYFO is an action which copies the contents of the specified flag to the current location of the specified object. There are also COPYFF and COPYOO actions which you can probably guess the purpose of.

That completes the control routine for the bird, but we need an entry in Process 2 to call this table every time frame, so select Process table 2 and insert an entry:

```
_       BIRD  PROCESS 3
```

which will cause PAW to execute our bird control table every pass round its main loop.

We must ensure the bird starts at the correct location and that the player knows the bird is there when the location is described (or they will see messages about a bird arriving and flying off, with the description containing no mention of it). So select Process 1 which is called after a location is described and amend the existing * * entry we made earlier to contain a LET 12 8, which will ensure the bird is on the branch at the start of the game. The modified entry should read thus:

```
*       *     AT        0
              ANYKEY
              LET      12   8  ;Bird is on branch (locno. 8)
              GOTO      2
              DESC
```

Insert the following entry in the same Process table which tells the player the bird is present and if it has the ticket.

```
_       BIRD  SAME     12   38
              MESSAGE   9
              ISAT      4  252
              MESSAGE  10
```

Finally select the Response table option and insert the entry:

```
GET   TICKE SAME     12   38  ;Bird at same location?
            ISAT      4  252  ;with ticket in beak?
            CLEAR     5       ;Force it to fly away
            NOTDONE           ;"I can't do that"
```

This will trigger before the GET _ entry and prevent the "There isn't one of those here" message being produced if the bird is present with the ticket.

CLEAR is an action which is followed by a flag number and sets the flag to have the value 0. This will cause the bird to fly away (which it might have been going to anyway) simulating its fright at having a great hand descend on it to get its prized new possession.

NOTDONE is an action similar to the DONE action but it fools PAW into thinking that nothing was done and thus causes it to print the "I can't do that" message.

Now the moment of truth, upon testing the game you should be able to watch the bird fly in and out of the bandstand and the branch, play with the game for a while to see the fact that the bird does indeed continue its roving existence. Then try dropping the sandwich at the same location. Note that if you do not pick up the ticket before the bird flies away it will snatch the ticket back.

The Dog

The dog will be added to complicate the game a bit more. The dog will simply follow the player everywhere (being a very obedient dog) and frighten the bird off. Now a dog would not be able to climb the tree so we must prevent the player from tempting the bird with the sandwich on the branch. To do so we will arrange for any object dropped while on the branch to fall to the ground. The player will be able to get rid of the dog by putting the lead on it and then tying the lead to the bench. In addition the player will be able to 'speak' to the dog which will provide another way of getting rid of the dog by asking it to SIT or STAY.

Before we examine the entries in Process and Response needed to control the dog insert the following words (into vocabulary) and messages (into the messages table) which will be needed.

The Dog

Verbs		Noun	
TIE	34		
UNTIE	35		
SIT	36		
STAY	36		
COME	37	HERE	37

Message 15

The _ falls to the ground at the foot of the tree.

Do not change the colour of this message and ensure you include the underline as it serves a special purpose we will discuss later. Most of the remainder of the messages deal with the dog and should be entered in magenta (EXTENDED MODE, CAPS SHIFT & 3) not forgetting to reset white at the end of them. do not do this for messages 21,22,23 and 25. This colour coding allows the player to see exactly what each message is referring to.

Message 16

The dog's bright eyes stare at me with mindless love.

Message 17

A dog is here.

Message 18

The dog follows me wagging his tail.

Message 19

A lead trails behind the dog.

Message 20

The dog is tied to the bench by a lead.

Message 21

Trustingly the dog lets me put the lead around its neck.

Message 22

I've tied the lead to the bench.

Message 23

Who should I say it to?

Message 24

The dog is sitting quietly.

Message 25

I've untied the dog from the bench.

There is no real need to make the control routine for the dog a separate process table as it is only one entry, but we shall do so in case you wish to expand the game later.

Flag 13 will contain the current location of the dog.

Flag 14 will contain: 0 - the dog is free to roam, 1 - the dog has the lead around its neck, 2 - the dog is tied to the bench, 255 - the dog is sitting quietly.

From the Process table menu; Begin a new process table (this should be table 4) and insert the single entry:

```
_    DOG   NOTSAME  13   38  ;Dog not where player is?
           LT       14    2  ;Still able to move?
           NOTAT     8       ;Player isn't up the tree?
           COPYFF   38   13  ;Move dog to players locno.
           MESSAGE  18       ;tell them its followed...
```

You should be able to work out what **NOTSAME**, **NOTAT** and **COPYFF** do but the technical guide will help you out if you have problems.

LT is a condition which succeeds if the flag specified contains a value Less Than the specified value.

Insert into process table 2;

```
_    DOG   PROCESS   4
```

which if you use P to look at the table should come before the entry for the bird (if not you entered the two vocabulary words the wrong way round). This ensures the dog will be moved to the players new location before the bird is checked.

Similarly to the bird entries are required in process table 1 to inform the player of the dogs presence:

```
_    DOG   SAME     13  38  ;Dog at same location
           MESSAGE  17      ;tell player
           EQ       14   1  ;with lead?
           MESSAGE  19      ;yes so tell player

_    DOG   SAME     13  38
           EQ       14   2  ;Dog tied to bench?
           MESSAGE  20
```

```
_       DOG   SAME     13  38
              GT       14   2  ;255 is greater than 2 so
              MESSAGE  24      ;tell player dog is sitting
```

while you are in process 1 modify the * * entry to contain a LET 13 2 (before the GOTO) to make the dog start at the bus stop.

Now in order for the bird to be frightened away by the dog we need an extra entry in process table 3. Now the entry must go before the entry which decides to drop the ticket and after the entries which make the bird fly. This will ensure that the bird will fly away with the ticket if it has it and leave it if it does not. So we need to insert before the sixth entry, use [I _ BIRD 6] to achieve this and type in the condacts for the entry from its listing below.

```
_       BIRD  SAME     12  13  ;Bird and dog at same location
              LET      12   8  ;Only ever on bandstand so
              LET       5   3  ;move to branch, three phrases
              AT        5      ;Player on bandstand?
              MESSAGE  13      ;tell them bird is gone..
```

The number after insert/amend has a maximum value of 255 so do not insert more than 256 entries of the same word values (that would be pretty unmanagable anyway) if you want to retain the ability to insert anywhere as well as on the end of the list.

The last change to the process tables is to insert a sub-process which we will be calling from Response to deal with speech to the dog. The mechanism works very simply. If the player includes a phrase in double quotes ("") in the input sentence, then the parser will save where it was and carry on with decoding the phrase. There is an action called PARSE which instructs PAW to use the parser to decode the string the player typed in, this then becomes the LS. It is only sensible to do this in a sub-process as PAW will try to match the new LS against the rest of the table. Begin a new Process table (table 5 should be next) and insert the following entries:

```
*       *     PARSE            ;Convert string to LS
              MESSAGE  16      ;Not valid phrase so
              DONE             ;dog does not understand!

SIT     _     ZERO     14      ;Dog not partially tied up?
              SET      14      ;now sitting quietly
              MESSAGE  24      ;tell player (always at same
              DONE             ;place as dog) Then DONE

COME    _     EQ       14  255 ;Dog must be sitting
              CLEAR    14      ;Now normal
              MESSAGE  18      ;Dog follows
              DONE
```

```
_       HERE  EQ       14  255
              CLEAR    14
              MESSAGE  18
              DONE

_       _     MESSAGE  16        ;Anything else.
```

We get around the limited vocabulary that the dog understands by making him wag his tail for most things!

PARSE will allow PAW to continue looking at condacts if it fails to find a valid phrase, be careful here as the current LS may be a bit jumbled up (i.e. the parser managed to get some sense out of the phrase) so you should normally only print a message like "They didn't seem to understand" or some such similar and DONE to return to your calling action. If it does form a valid LS PAW will start to search the following entries for a match as with Response. PARSE should only be used in a sub-process called from Response it has no meaning in any other table.

Notice how the COME and HERE entries deal with a variety of phrases that the player might try to call the dog again having made it sit.

The _ _ entry catches all the valid LS which may have been in the string and the dog has no specific response to.

Select the Response table now to allow us to insert the extra entries to control speech and the dropping of objects in the tree.

First off the mark is the entry which causes all objects dropped in the tree to fall to the ground, now this must go between the entry which deals with putting objects in the bag and the normal DROP _ entry (actually printed as PUT!). [I PUT _ 1] will achieve this, the entry is:

```
PUT     _     AT        8          ;Player on branch
              WHATO                ;I say old boy!
              LT       51  255     ;Valid object?
              EQ       54  254     ;Object carried?
              MESSAGE  15          ;its now bottom of tree.
              PUTO      7          ;put it there
              DONE
```

This is an example of creating an automatic action of your own, like AUTOG and so on.

WHATO is an action which looks up the first Noun in the current LS in the object word table, converting it into an object number. This number is then placed in flag 51. Flag 51 always contains the number of the last object referenced by PAW and whenever it is set the associated flags 54 to 57 are also set. Flag 54

contains the current location of the object.

PUTO is an action which changes the location of the currently referenced object to be the one specified.

Message 15 contained an underline. Whenever PAW meets an underline in text (be it message or location) it replaces it with the current object hence the message is changed to suit the object currently being dealt with.

Next a relatively simple entry to deal with PUT LEAD ON DOG:

```
PUT   LEAD  PREP     ON        ;Ensure not a DROP LEAD
            NOUN2    DOG
            CARRIED   5        ;Player has the lead
            SAME     13   38   ;is at same location as dog
            LET      14    1   ;Dog now has lead on
            DESTROY   5        ;so player hasn't
            MESSAGE  21        ;tell them so.
            DONE
```

The entries which follow deal with a new concept again, the modification of the current LS. We want the game to understand both TIE DOG TO BENCH and TIE LEAD TO BENCH as the the same thing, now LEAD and DOG are separate word values, so the TIE DOG entry which will come first in the table (as its word value is lower that LEAD) converts the Noun into LEAD (55) and allows PAW to carry out the TIE LEAD entry! A similar system is used for UNTIE. Insert the entries:

```
TIE   DOG   LET      34   55   ;Flag 34 is Noun for LS

TIE   LEAD  PREP     TO
            NOUN2    BENCH
            AT        4        ;Where bench is.
            SAME     13   38   ;dog is here
            EQ       14    1   ;with lead on
            PLUS     14    1   ;now tied to bench
            MESSAGE  22        ;tell player about it
            DONE

TIE   _     NOTDONE            ;Ensure an I can't

UNTIE DOG   LET      34   55   ;Flag 34 is Noun for LS

UNTIE LEAD  AT        4        ;Where bench is
            EQ       14    2   ;dog tied to it
            CLEAR    14        ;Now free
            MESSAGE  25        ;Tell player
            CREATE    5        ;Recreate lead
            GET       5        ;Try and get it.
            DONE
```

```
UNTIE _      NOTDONE              ;Ensure an I can't
```

The NOTDONE makes sure PAW reports "I can't do that" if you try and TIE or UNTIE anything other than the lead/dog.

CREATE is an action which is followed by an object number. It causes that object to be at the position where the player is.

GET is an action which is followed by an object number. It attempts to get the specified object.

We use these actions instead of just placing the object at 254 so that any weight and/or number of objects carried problems are reported.

Finally the the entries to allow speech to the dog, we have also included the entry necessary to allow you to speak to the bird - it just ignores you!

```
SAY   DOG   SAME     13   38   ;It's here
            PROCESS   5       ;Someone else to do the work
            DONE

SAY   BIRD  SAME     12   38
            MESSAGE    8
            DONE

SAY   _     MESSAGE  23       ;Who?
            DONE
```

Notice that we do not ensure the preposition TO is specified - this allows the player to shorten their input if required. As a general guide don't check for an extended LS unless it is required to differentiate two similar phrases.

As a final test the following inputs should now work in the indicated situations, they show some of the power which the parser can provide your games with.

When on the path by the park bench with the lead and dog try;
PUT LEAD ON DOG AND TIE IT TO THE BENCH

then to untie it;
UNTIE DOG

When up the tree with the bag try;
PUT ALL IN BAG AND DROP IT. GO DOWN AND LOOK IN BAG

To make the dog sit down;
SAY TO DOG "SIT"

and get back up;
ASK DOG TO "COME HERE"

Do it yourself

Before we move onto a discussion of the graphics here are a few points that you might like to tidy up in the demonstration game as practice on using the system (after the graphics if you like).

1/ EXAMINE should respond to all objects even if it is with a general reply such as "I see nothing special about the _.". Hint: so as not to lose the use of LOOK on its own you could use a condition LT 34 255 before triggering (i.e. ensure a Noun was actually specified).

2/ The bird should really fly away if you GET SANDWICH while the bird is present. i.e. it will be pecking at the sandwich and any normal bird would fly...

3/ UNTIE _ and TIE _ should have a message something along the lines of "Tie what to what?", NOTDONE was an easy copout!

4/ How might you deal with the player typing PUT object IN BAG when the bag is not present? at the moment the game will drop the object instead, why?

5/ Nothing was ever done with the torch, the following entries will allow it to be turned on and off (you will also need TURN as a verb in the vocabulary):

```
TURN  TORCH PREP    ON
            CARRIED  7
            SWAP     7    0
            OK

TURN  TORCH PREP    OFF
            CARRIED  0
            SWAP     0    7
            OK
```

Lookup the extra condacts in the technical guide and read the chapter on light and dark - perhaps a cellar could be created below the bandstand? The movement would have to be checked in the Response table with an entry such as: (assuming 9 is the new location).

```
DOWN  _     AT      5      ;Player on bandstand?
            SET     0      ;Flag 0=255=Dark!
            GOTO    9      ;New location
            DESC
```

Not forgetting an entry for UP which clears the flag!

6/ What happens if the player types CLIMB TREE or CLIMB UP TREE and what is the best way to check for this? Hint: there is only one thing you can climb in that location.

Overlays

The 128K user will still not need to use overlays yet, but may find it useful to read this chapter anyway.

The idea of overlays was explained in concepts, in order to proceed with the graphics and text compression system the 48K user will have to load an overlay.

PAW will do most of the work for you, if you just select the main menu option you require you will be asked to confirm you want to load an overlay, any key other than Y will return you to the main menu. If you do proceed PAW will print the name of the overlay it is searching for on the screen.

The five overlay files are at the end of the main program (which is where your tape will be after loading PAW). If you have a tape counter it is worth setting it to zero at this point and then noting down the readings for each overlay, fast forward can then be used to go to just before the required file. The five files are in the order shown below, also described are the main menu options contained within each (note that the selection of an option present in the current overlay is automatic):

PAWOVR	1	Interpreter, Test Game, Save/Verify Adventure.
PAWOVR	4	Process/Response, Vocabulary, Connections, Words.
PAWOVR	5	Messages, Locations, Objects, Initially At, Object Weight and Background colours.
PAWOVR	2	Compressor.
PAWOVR	3	Character Editor/Graphics Editor.

Note that Save, Verify and Load database along with Free memory are always available as they are part of the main menu.

Once the file has loaded you will be presented with the sub-menu as normal. If an error occurs you will be returned to the main menu, just reselect the option and try again. (Note that any overlay loaded previously will be erased by a tape error, so unless you still have sufficient memory to hold the main overlays you will be able to do nothing but load a new overlay - or save/load database which is always available).

If you do not have a tape counter or wish to make things even simpler you might like to transfer each of the files to a separate tape, they are just normal CODE files.

Text Compression

Option K on the main menu (48K users will have to load the overlay at this point) will ask you if you want to compress the database, any key other than Y will return you to the main menu. Otherwise the text compressor will reduce the amount of memory needed for the text in your game by grouping common letters into a single 'token', this can take anything from one minute to an hour depending on the size of your game. On the demo this should take about a minute and save about 900 bytes!

The only difference you will notice is when editing existing text where the cursor will jump two, three, four or even five characters at a time - including deleting. Just retype all the letters separately if you make a correction which requires them to be deleted, they will be compressed the next time you use the compressor. Note that the compressor uses the normal spectrum tokens, which will produce letter groupings and not the keywords after you use the compressor, so do not use the tokens if you intend compressing the database.

The Character Editor

Just quickly we will take a look at the character editor. Select option Q from the main menu - 48K users will need to load an overlay. This sub-menu allows you to change the way the characters which are displayed on the screen look. You can have upto five different character sets in memory, and change between them at will using ESCC 0-5 or a CHARSET action in Process or Response. The sets are numbered 0 to 5, set 0 is the normal set which can not be changed, except for character values 0 to 15 which are the shade patterns and 144 to 165 which are the normal spectrum UDGs. If you use [P] to look at the table you will find only these characters displayed. Note that you have to Insert a blank set before you can change or load it which conserves memory in the database if you are not changing the character set.

At the moment we are just going to use the editor to change one of the shade patterns. These are just normal characters which the graphics system can use to colour in an area of screen with.

[A 0 15 ENTER] will allow you to edit character 15 of set 0. This is an unimportant shade pattern which we will be altering to represent the iron work on the bandstand.

Each character in PAW is defined on an 8 by 8 pixel grid, the top left box on the screen will be showing an enlarged version of the pattern as it is at the moment, the top centre and right boxes show how it will look when used as a shade (both normal and in inverse) while the bottom gives a summary of the commands available and current character under edit. Use the cursor keys (CAPS SHIFT 5 to 8 on 48K) to move the red flashing cursor around the grid and the SPACE key to 'toggle' the pixel it is over on/off (that is; if the pixel is on - black - it will be turned off, and if off - yellow/white - it will be turned on. Try it and you will soon see what we mean!).

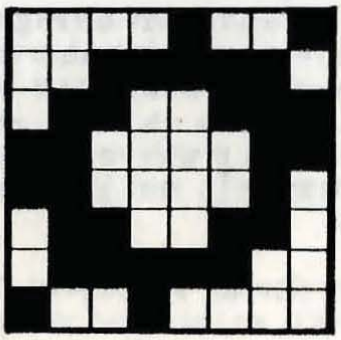


Diagram 5

The pattern we require is shown in Diagram 5. When you have finished use [R] to redraw the two shade boxes to see what the new pattern looks like. And finally press ENTER to end the edit.

Side two of the supplied cassette contains 22 different character sets which can be loaded into character sets 1 to 5, after you have inserted them of course. After insertion of a set the option to select that set as the default will be offered on the Background colours option of the main menu.

The Graphics Editor

The graphics system on PAW uses a method of drawing called Line and Fill which is very efficient on memory usage for the type of pictures included in adventure games. Instead of storing an image of the screen that you have drawn like many commercial art packages, it stores a list of the commands you used to draw it. Even the most complex of pictures will only consume 2K of memory as compared to 6K for a standard screen, and indeed you should find that effective designs can be drawn using as little as 100 bytes!

The list of commands stored is called a drawstring, and there is a drawstring for every location you insert using the locations option on the main menu. If you are illustrating only a few of your locations the other drawstrings will be empty.

Every picture (and therefore every location) has a Paper and Ink colour defined for it. Select option D from the main menu and you will be presented with a sub-menu to deal with amending these values, use [P] to see that an entry exists for our 9 locations in the demo game. They are all marked as a subroutine which tells PAW two things:

1/ Do not draw this picture when you describe a location.

2/ This is a sub-picture which can be used in other pictures.

The sub-picture facility is similar to the sub-process idea discussed previously and an example of its use is given later.

For our example of using the graphics we shall be drawing a picture of the bandstand as seen from location 4 (on the path). So we want to make PAW draw the picture when we visit the location. You do this by assigning the picture a Paper and Ink value, we will use a black background and Yellow ink so type [A 4 0 6 ENTER], if you now use [P] the entry for location 4 should be:

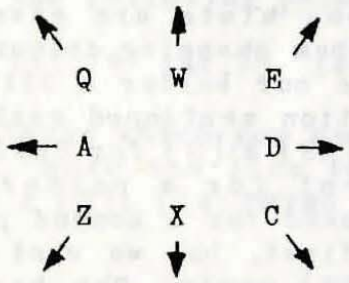
Location 4 Paper: 0 Ink: 6

You could change it back if you wanted by leaving the paper and ink values out of the amend (e.g. A 4). Leave it as it is for the moment and select option G from the main menu. 48K owners should have found that they did not need to use overlays again - this is because Characters, Default colours and Graphics are all in the same overlay, the use of the amend option in a moment will wipe out the overlays which PAW holds in memory and if you wish to use any other options except Save/Load or Free memory you will need to load the overlay containing the option.

So on with the graphics; several options are provided on the menu

to allow you to amend, print (on the screen), copy (to a printer), calculate the size of and dump a screen image of each picture in the game. We will be creating a picture for location 4 so type [A 4 ENTER]. The screen will clear and two lines of information will appear in the lower screen, this shows from left to right; On the top line the current drawing ink colour, the current background paper colour, the status of the flash and brightness options and on the lower line the current x,y coordinate of the drawing position and the location number under edit. Sundry other information is also displayed at times which will be explained as necessary.

If you look carefully you should also see a single flashing pixel in the bottom left, this is known as 'the point' and indicates the start position for any drawing. If you press key E you should see a line start to appear, the one end of the line is always at point and the other end of the line can be moved using the keys around S as follows:



Alternatively if you have a joystick you can plug it into port 2 (Plus 2 and Interface 1). Kempston™ interface users should press [SYMBOL SHIFT] and [J] to activate the driver for it - a letter J will appear on the bottom line to indicate it is active. The joystick will now move the end of line around. Movement will be by single pixels, this can be accelerated to eight pixels at a time by holding down the CAPS SHIFT key at the same time as one of the eight keys (or while pushing the joystick in a specific direction).

The line is 'rubber banding' (a term which arrives from the fact it acts like a taut rubber band) and will allow you to position lines accurately before you draw them.

Our sample game is going to have 'split screen' graphics so we want to leave several lines spare on the screen below the picture for text. Press [SYMBOL SHIFT] and [Y] to activate a grid which shows the character boundaries and move the line to X=0,Y=47. We are going to move 'point' to the end of the line so that any drawing starts at that pixel, press [SYMBOL SHIFT] and [P] for PLOT to achieve this, the current point will now be where the end of the line was. Next move the line to X=255,Y=47 using [A] once which demonstrates the 'wraparound' action of the line (i.e. moving off one side of the screen brings it back on the other side). This time we want to actually draw the line (this is called fixing the line) so press [SYMBOL SHIFT] and [L] for LINE

- or use FIRE on the joystick which acts like SYMBOL SHIFT and L.

If you make a mistake you can delete the previous command by pressing DELETE (CAPS SHIFT and 0 on 48K) - all the way back to the start of the picture if you like!

All graphics commands which insert in the drawstring (like PLOT and LINE) require SYMBOL SHIFT to be held down so we shall shorten it to SS, and any co-ordinates given will be in the form X,Y e.g. 255,47 instead of X=255,Y=47.

Borders around pictures seem to be the fashion at the moment so ours shall have one! Move the end of line to 248,55 and PLOT the point, now draw a box by moving to 248,168 and fixing the line, then onto 7,168 and fixing the line and so on for points 7,55 and back to 248,55. Note that these lines just skirted the outside of each character 'cell'. Because of the spectrums limitation of only two colours in each cell you must be careful in your positioning of lines (some hints are given in the technical guide), or you will find them changing colour later when you draw near them. Finally to make our border a little more interesting we shall use the shade option mentioned earlier. Move the end of line to 248,53 and press [SS] & [S] for SHADE; the lower screen will change to a request for a pattern number, type in [12 ENTER], you will be asked for a second pattern, this pattern would be overlaid on the first, but we want to use pattern 12 on its own so type [12 ENTER] again. The border area should be magically shaded with fine diagonal lines. Note that point has not moved and the line will grow from same same place as before the shade.

The shade command is as you will have noticed very fast, it is also very good at shading unusually shaped areas of screen, including worming its way through single pixel 'holes' in your picture. Of course you can delete an errant shade. The shade area is defined by at least a single pixel line or the edges of the screen as in the border detailed above. It will not always shade the entire empty area, but careful positioning will allow most of it to be shaded in one go, any unshaded areas can be completed by using the shade command again with a start point within the empty area. Shade is provided with sixteen possible default patterns which you can change using the character editor as detailed earlier. Note that if you change a pattern all uses of that pattern in pictures will be changed as well, so it is best to choose useful general patterns for the majority and define only a few special patterns where absolutely necessary. Patterns can of course be mixed together by specifying different pattern numbers when prompted to provide a wide variety of useful designs.

Now we shall lay down the main sky and grass areas. We are merely going to set down a paper colour so move the line to 8,56 and press [SS] & [A] for ABSOLUTE MOVE, this does not affect the

pixel unlike PLOT. Press [SS] & [Y] to get rid of the grid or you will not be able to see the colours. The grass will be green so press [SS] & [C] for PAPER, green is colour 4 so type [4 ENTER], note that the current paper colour changes to be 4. Now move the line to 246,87 and press [SS] & [B] for BLOCK which will colour in the rectangle of character cells which the line forms the diagonal of, with the current Ink and Paper colours. The sky will be blue so ABSOLUTE MOVE the line to 247,88 ([SS] & [A]), select blue paper ([SS] & [C], [1 ENTER]), then BLOCK the rectangle to 8,167 (move to 8,167 and press [SS] & [B]).

Now we shall draw the base of the bandstand in red brick. In order that we avoid the colour boundary problems the base will be exactly three character cells high and sixteen wide. Select red paper ([SS] & [C], [2 ENTER]), and black ink using [SS] & [X] for INK, 0 is black so type [0 ENTER]. Now PLOT the point at 191,72 (i.e. move the line to 191,72 and press [SS] & [P]). Then fix lines between each of the following points;

64,72 64,95 111,95 111,75 144,75 144,95 191,95 191,72

One of the shading patterns which you may have noticed earlier is a brick type pattern, move the line to 189,73 and SHADE using pattern 14 (i.e. [SS] & [S], [14 ENTER 14 ENTER]) to create an effective brick base.

Your picture should look like diagram six by now. To create the steps up to the bandstand move the line to 144,78 and press [SS] and [R] for RELATIVE MOVE which moves point like PLOT and ABSOLUTE MOVE but to a pixel a fixed distance from the current point instead of an absolute x,y position. It is used to keep groups of commands which draw a single object in the picture together, the reason will be demonstrated in a moment. Fix a line to 112,78 and then use RELATIVE MOVE to move to 111,81, fix a line to 143,81 and so on for each of these coordinate groups:

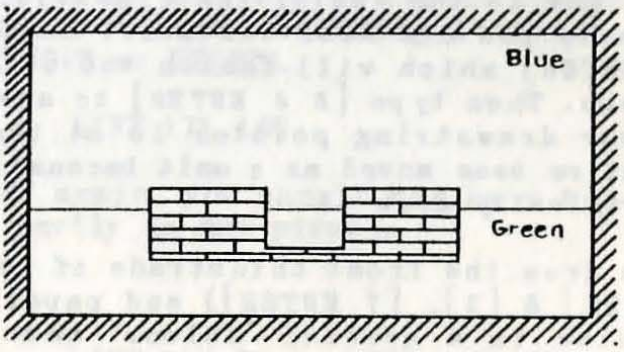


Diagram 6

REL MOVE to 144,84 LINE to 112,84
REL MOVE to 111,87 LINE to 143,87
REL MOVE to 144,90 LINE to 112,90
REL MOVE to 111,93 LINE to 143,93

We will now examine the editing facilities that are available to correct mistakes in addition to DELETE. As an example we will move the entire brick base of the bandstand two character cells

further to the right. To DELETE all the way and redraw seems a bit too much like hard work!

As you draw your picture PAW adds each command to the drawstring, where it adds them is called the drawstring pointer and at the moment the drawstring pointer is at the end of the drawstring. It is quite feasible for PAW to backtrack along the commands you have entered so far to any point along the drawstring. Press [CURSOR RIGHT] once, don't worry the picture is still there, but PAW only draws the picture as far as the drawstring pointer, which is now at the START of the drawstring, Diagram 7 might help you to visualize the way the drawstring works in memory. You can step forward one command in the drawstring by pressing [CURSOR DOWN] for NEXT command, and back one command using [CURSOR UP] for PREVIOUS command - note that this does not delete the command it merely moves the drawstring pointer back one command.

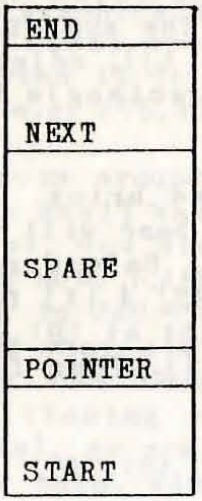


Diagram 7

Use NEXT (CURSOR DOWN) until the PLOT (at 191,72) command which starts the brick base is carried out. Use DELETE to remove it from the drawstring and PLOT 207,72 which PAW will insert in the drawstring at the pointer. If you use NEXT now the base line should be drawn, rather than use NEXT all the time to get to the end of the drawstring a useful trick especially where you are near the start on a long drawstring is to press [ENTER] which will finish the edit and return you to the sub-menu. Then type [A 4 ENTER] to amend the picture again, presto your drawstring pointer is at the end again. Notice how the entire base moved as a unit because we used RELATIVE MOVE when we originally drew it.

To draw the front balustrade of the bandstand select ink white ([SS] & [X], [7 ENTER]) and paper 8 ([SS] & [C], [8 ENTER]) - which is a special 'colour' meaning do not affect the paper colour. Then:

PLOT 206,96 LINE 206,109 LINE 162,109 LINE 162,96

move the line to 163,97 and SHADE using the pattern you designed earlier 15, ([SS] & [S], [15 ENTER 15 ENTER]).

PLOT 125,96 LINE 125,109 LINE 81,109 LINE 81,96

move the line to 82,97 and SHADE in pattern 15 again.

Now the upright poles for the pagoda:

PLOT 103,96 LINE 103,136 LINE 106,136 LINE 106,96

move the line to 104,121 and press [SS] and [F] for FILL, this fills the defined area completely in set pixels in a similar way to shade.

REL MOVE 184,96 LINE 184,136 LINE 181,136 LINE 181,96

move the line to 183,122 and FILL, ([SS] & [F]).

The top of the pagoda:

REL MOVE 207,143 LINE 206,140 LINE 202,136 LINE 86,136
LINE 83,139 LINE 80,143 LINE 207,143

move the line to 205,141 and SHADE in pattern 15.

REL MOVE 144,166 LINE 220,139
REL MOVE 65,139 LINE 144,166

move the line to 144,164 and FILL.

To create a rounding effect on the pagoda select OVER by pressing [SS] and [O], a letter O will appear on the top status line to indicate that over is active. Normally every PLOT and LINE command sets the pixels it affects, but those inserted while over is active will set pixels that are reset and reset pixels that are set, much like the toggle action of SPACE on the character editor. Note that the state of Over (and Inverse introduced later) is encoded as part of the command, to get the effect you have to insert the command while it is active, you cannot change a PLOT or LINE inserted previously without deleting it first. Over is cancelled by START, PREVIOUS or DELETE.

LINE 115,140 REL MOVE 144,166 LINE 171,140

and [SS] and [O] to turn over off again, you should now have two lines drawn partly in reset and partly in set pixels.

The last upright on the pagoda:

REL MOVE 142,136 LINE 142,96 LINE 145,96 LINE 145,135

move line to 143,133 and FILL.

The back balustrade of the bandstand is drawn slightly smaller:

REL MOVE 125,107 LINE 161,107
REL MOVE 162,96 LINE 126,96

move line to 128,98 and SHADE in pattern 15, and to 150,98 to SHADE in pattern 15 again. To improve the look we shall make the centre upright stand out a bit by removing a line of pixels either side of it. Press [SS] and [I] for INVERSE and a letter I should appear on the bottom line to show Inverse is active.

Inverse causes any PLOT and LINE commands to reset pixels instead of setting them. So:

```
REL MOVE 141,95     LINE 141,107
REL MOVE 146,108    LINE 146,96
```

then Inverse off ([SS] & [I]).

Now to construct the railings around the perimeter of the park we shall use a useful technique of undrawing the surround of a shade pattern. Draw in black ink ([SS] & [X], [0 ENTER]):

```
PLOT 8,104     LINE 79,104     LINE 79,88     LINE 8,88
PLOT 247,88    LINE 208,88     LINE 208,104   LINE 247,104
```

SHADE pattern 7 at 246,102 and 11,102. Undraw the top of the railings to create spikes by turning Inverse on ([SS] & [I]) and:

```
PLOT 247,104   LINE 208,104   PLOT 79,104   LINE 8,104
```

then Inverse off. To finish the effect draw:

```
PLOT 8,101     LINE 79,101     PLOT 208,101   LINE 247,101
```

The main picture is finished but we are going to add some tufts of grass using the subroutine feature mentioned earlier, this will save memory and the time taken to draw four tufts of grass. Press [ENTER] to finish the edit and return to the sub-menu. Now location 0 is the title screen for the demo so we shall use its drawstring to contain our tuft of grass. Using [A 0 ENTER] amend the picture for location 0. Note that the location number is followed by a letter S to indicate that this is a subroutine.

Draw the tuft of grass by temporarily plotting 72,72 and fixing lines between the following points:

```
68,85    77,73    74,89    81,73    79,94    87,72
87,94    92,70    94,86    95,70    99,79    98,69
```

now return to the START of the drawstring ([CURSOR RIGHT]) and use NEXT ([CURSOR DOWN]) to step past the PLOT, then use DELETE to remove it. This strange action means that the start of the first line is at 0,0 and allows us to position the picture accurately. If you try and amend the picture again you will get an "Out of range" error because PAW cannot draw a line 'off' screen. Your drawstring pointer will be positioned just before the first LINE command so insert the PLOT again while you edit the drawstring, deleting it again at the end.

Amend picture four again ([A 4 ENTER] from the sub-menu) and PLOT point 217,69. Now press [SS] and [G] for GOSUB, you will be prompted for a location number to use, type [0 ENTER] to use our newly defined tuft of grass. Next you will be prompted for a

scale, this defines how big the picture will be in eighths of its original size, type [4 ENTER] to draw it at half its original size. Do the same for the following:

```
PLOT 21,58     GOSUB 0 scale 5
PLOT 60,74     GOSUB 0 scale 3
PLOT 103,61    GOSUB 0 scale 4
```

And to demonstrate errors:

```
PLOT 128,170   GOSUB 0 scale 0
```

Scale 0 actually means full size, not zero eighths! The error which was generated has left the drawstring pointer before the command which caused the error, in this case the GOSUB. There would appear to be no way to delete this without plotting further down and so on. In fact [GRAPH] ([CAPS SHIFT] & [9] on a 48K) will DELETE the NEXT command. You might as well [DELETE] the PLOT as well.

Note that at the end of an edit it is possible for you to still have commands above the drawstring pointer that you do not want. You can ensure these are removed by holding down DELETE NEXT ([GRAPH]) for a while.

Return to the main menu and select Process table 1 (48K owners will need to load an overlay). Amend the * * entry to contain MODE 3 3 LINE 16 before the GOTO 2. The MODE action selects the way the screen operates. Mode 3 is a fixed graphic area (any text displayed will not remove it) - the second 3 tells PAW not to change the border colour and to print "More.." when a screenfull of text has been displayed. LINE 16 tells PAW where the first line of text is to be displayed.

Finally use test adventure (again 48K owners will need to load an overlay) to see your picture in action. It will be displayed the first time you visit the path, but not on subsequent visits. This is known as normal mode for graphics. It is also possible to select On and Off which always draw and never draw the graphics respectively. These options can be selected using PICS NORM, PICS ON and PICS OFF - those entries we didn't explain in Response!

You must make the decision as to whether to allow the player to switch between options during the game or to force a single method at the start.

End of the road

We hope that the above tutorial has provided an insight into som of the many powerful facilities of the Professional Adventur Writer. Now it is time for you to expand your knowledge of th system by using it! You will find a complete list of CondActs an flags in the quick reference guide. The Technical Guide an Notebook supplied will provide an exact specification o everything that PAW contains, plus many hints and tips on variety of subjects. These will form pretty essential - if little heavy - reading when writing more advanced games.

Finally you will find a small game in database form on th cassette after the overlays called "TEWK", which should be loade using option J on the main menu (after saving <u>your</u> database c course!). Looking through this should provide you with some mor ideas on giving your game an individual look.

What should I do next?
HAVE FUN!
OK

Tim Gilberts - November 1990

Upgrades

If you intend to make a more serious use of PAW then you may be interested in the following upgrades available only from Gilsoft by direct mail order:-

User Overlay Writers Guide £1.99

For the m/c enthusiasts a complete set of documentation for those wishing to penetrate the depths of the PAW database. Allowing them to write their own User overlays. This includes a full listing of the Hunk Manager as an example.

PHOSIS/TEL/MEGA £7.95

These were the first three commercially available user overlays from KELSOFT. Supplied on cassette with an accompanying manual they provide a host of utilities for the writing and debugging of adventures. Including the ability to; save, load, move and copy process tables/entries. Full conditional search facilities to extract data on flags/messages and process usage. A graphical map display. Overlays 4 and 5 in a single load to save 48K users the hassle of changing overlays! Plus many more...

PLUS 3 Disk Upgrade £5.00 + 50p P&P

A version of PAW especially written to take advantage of the disk drive on the +3. Including the facility to save/load databases, games, positions, character sets and overlays.

N.B. If you return your PHOSIS/TEL/MEGA cassette (or order the disk upgrade and PTM's at the same time) we will supply the special disk version of PHOSIS/TEL and MEGA on the same disk.

DISCiPLE/PLUS D £3.99 + 50p P&P

Similar to the PLUS 3 upgrade this allows the use of the DISCiPLE/PLUS D. This is supplied on cassette with a transfer program to avoid problems with the large number of disk types and sizes in use on these systems.

Microdrive upgrade £3.99 + 50p P&P

Again similar to the other two upgrades this provides a version to use the Sinclair microdrive to save/load files. Again this upgrade is provided on cassette due to the reliability and availability of cartridges.

To order any of the above items send a cheque or a postal order made payable to Gilsoft to the address below. Or alternatively quote your Mastercard/Access/Visa number, ensuring you sign the order and state clearly the cards expiry date.

We can also take credit card orders by telephone. The telephone is manned from 10am until 5.30pm each day except Mondays,Wednesdays and Sundays.

Our address and telephone number is:-

Gilsoft (PAW Upgrades)

2 Park Crescent
Barry
South Glamorgan
CF6 8HD

Tel: 0446 732765

The Professional Adventure Writing System

Notebook

Essays on the use of:

The Professional Adventure Writer

A graphic adventure writing system for the Sinclair Spectrum computers.

(c) 1986/90 Gilsoft.
Program: T.J.Gilberts, G.Yeandle and P.Wade
Graphics: D. Peeke, K.Maddocks and A.Williams
Manuals: T.J.Gilberts

All Rights reserved. No part of this publication may be copied, loaned, hired or reproduced in any form whatsoever including electronic retrieval systems without the prior written consent of the authors and Gilsoft.

The above notice does not apply to the 'run time' routines appended to, and which form a part of, a saved game which you are free to distribute any way you wish in that form. All we would request is that you credit the use of the Professional Adventure Writer somewhere within the game.

Contents

Whats all this then Page 5

PAW Features Page 6

These discuss in greater detail a number of important aspects of the PAW system which must be dealt with as a whole.

The Parser Page 6

How it works and the way it forms the LS.

Objects Page 8

Important aspects including '_' and containers.

Screen Output Page 9

Compound messaging explained.

Wordwrap Page 10

Beware for hidden monsters dwell here...

The input line Page 10

Spectrum Specifics Page 11

A number of items which deal with the implimentation of PAW on the Spectrum.

128K Considerations Page 11

This deals with the Page nature of a PAW database, when to change pages etc.

Colour Boundaries Page 11

The inescapable problem of colour clash in graphics has a number of solutions.

Flash and Bright Page 12

Character sets Page 13

Many people seem to suffer problems with the concepts, here are some pointers...

Printers Page 14

A discussion of the use of external printers.

Contents

The RAMPRINT interface Page 15

EXTERN and Loading screens Page 16

More on the workings of EXTERN and BASIC loader program writing.

Common game features Page 18

Several features are found in most adventures. These short extracts give the method required to impliment them with PAW.

Light and Dark Page 18

Time Page 18

Exit lists Page 20

Secret exits Page 21

Examine objects Page 22

Invisible objects Page 22

Password protection Page 22

Multi-Part adventures Page 23

Comparing flags Page 24

PSI's revisited Page 25

PSI's are little more than a collection of process tables and flags. But they can be made to do a great variety of things.

Sanec created Page 25

Better directions Page 27

Following Page 27

Fighting Page 28

Multi-parsed commands for PSI's Page 30

Selling your games Page 40

Some notes to help those intending to sell their work.

Overview

Two of the greatest problems with any programming language, and indeed we must consider PAW as such, are:

i) Gaining an insight into WHY a particular feature was included in the language, and what possible use it might have.

ii) Concieving the method necessary to impliment a wanted function with the language.

The easiest way to solve these problems is by example. This book contains a series of notes and essays on a variety of topics related to PAW which give useful examples of the functioning of the system.

Some contain details that are not provided elsewhere in the manuals of PAW, purely because they relate to a number of features taken together. Thus there was no logical way of placing them in the other manuals.

Some are detailed descriptions on the implimentation of a specific function, while others a general guide as to the approach to take.

Several of the sections herin have arisen in response to direct questions from other owners of PAW over the years. We hope it is of use...

The level of expertise assumed in each section varies, so you will find it far more useful to 'dip' into this book to read about topics that interest (or puzzle!) you. As opposed to reading it from start to finish.

The Parser

The parser works by scanning an input line (up to 125 characters) for words which are in the vocabulary, extracting 'Phrases' which it can turn into Logical sentences.

When a phrase has been extracted, the Response and Connections tables are scanned to see if the Logical Sentence is recognised. If not then system message 8 ("I can't do that") or system message 7 ("I can't go in that direction") will be displayed - depending on the Verb value. i.e. if less than 14 then system message 7 will be used. Then a new text input is requested.

A new text input will also be requested if an action fails in some way (e.g. an object too heavy) or if the writer forces it with a NEWTEXT action. The results might otherwise be catastrophic for the player. e.g. GET AXE AND ATTACK TROLL, if you don't have the axe you wouldn't really want to tackle the Troll!

If the LS is successfully executed then another phrase is extracted or new text requested if there is no more text in the buffer.

Phrases are separated by conjugations ("AND" & "THEN" usually) and by any punctuation.

A Pronoun ("IT" usually) can be used to refer to the Noun/Adjective used in the previous Phrase - even if this was a separate input. Nouns with word values less than 50 are Proper Nouns and will not affect the Pronoun.

The Logical Sentence format is as follows:-

(Adverb)Verb(Adjective1(Noun1))(preposition)(Adjective2(Noun2))

where bracketed types are optional. i.e. the minimum phrase is a Verb (or a Conversion Noun - which is a Noun with a word value <20 - which if no Verb is found in a phrase will be converted into a Verb e.g. NORTH). If the verb is omitted then the LS will assume the previously used verb is required. i.e. GET SWORD AND SHIELD will work correctly! - the pronoun will be the first object in a list like this, SWORD in the example.

Note that the phrase does not strictly have to be typed in by the player in this format. As an example:

GET THE SMALL SWORD QUICKLY
QUICKLY GET THE SMALL SWORD
QUICKLY THE SMALL SWORD GET

are all equivalent phrases producing the same LS. Although the third version is rather dubious English.

A true sentence could be:-

GET ALL. OPEN THE DOOR AND GO SOUTH THEN GET THE BUCKET AND LOOK IN IT.

which will become five LS's:-

```
GET ALL
OPEN DOOR    (because THE is not in the vocabulary)
SOUTH        (because GO is not in the vocabulary)
GET BUCKET
LOOK BUCKET (from IT) IN (preposition)
```

Note that DOALL will not generate the object described by Noun(Adjective)2 of the Logical sentence. This provides a simple method of implementing EXCEPT. e.g. GET ALL EXCEPT THE FISH, it has the side effect of not allowing PUT ALL EXCEPT THE FISH IN THE BUCKET, as this has three nouns!

The main parser also stores the address of any string - section of the phrase enclosed in quotes - when scanning for the LS. This is the text examined by the PARSE CondAct.

One or two subtle changes have taken place in the string handling section of the parser from Version A16 onwards. These are designed to provide a facility for multiple commands to be given to PSI's, a facility suggested by Gerald Kellett. The three changes are as follows, and although they may not seem very major the logic changes they represent could affect some games if you weren't aware of them:

1/ The PARSE CondAct now maintains a 'current position' within the string in the current logical sentence. Thus a second PARSE CondAct will continue from where the last left off. Previously a subsequent PARSE would have given the same logical sentence as the first. Thus SAY TO PSI "GET SWORD AND CLEAN IT" can now be made to work with some processing as described in the section on PSI's.

2/ The PARSE action does not now affect the 'command line empty/valid' flag - the one set by NEWTEXT. This means that a statement such as; SAY TO PSI "HGGHHG". GET SWORD - will now continue on to do the GET SWORD action. Prior to this version the NEWTEXT flag would have been set automatically. This was changed to allow multi-parsing to find the last command in a string without always setting the flag. You will thus need to add a NEWTEXT action to old games just after the PARSE CondAct - which is where processing goes if the string was invalid or empty - if the games are to operate identically with the new paws.

3/ The current Verb and Adverb are not cleared (flags 33 and 36 to 255) when a string is parsed (i.e. the PARSE action). This means that if a Verb (or Conversion Noun) is omitted from the

first phrase in the string then the current verb will be the one from the phrase which triggered the PARSE (usually SAY or TALK). This is a minor change which means that the current Verb is maintained when the string is multi-parsed. I.e. SAY TO PSI "GET SWORD AND SHIELD" will now work with the processing shown below.

Flag 58 wasn't used in earlier versions of PAW. Now if you set this flag to 128, in a Process table, PAW will start to match words. It normally doesn't do so except in Response. This allows the multi-parse facility to provide actions for a PSI during Process 2. It will also have other uses we are sure...

The effect is cancelled next time Process 1 or 2 are carried out - by PAW subtracted 128 from the flag. This ensures that Process 1 and 2 act as normal until specifically told to change. You can of course cancel the effect yourself by setting the flag back to zero.

* * * * * *

Objects

Underlines in text will be converted during gameplay into a description of the last object referenced by GET,DROP,DESTROY etc. This is mainly to deal with the fact that GET,DROP etc report their success (or failure!) but can be used usefully for examining objects and other automatic reports.

Flag 53 is used to control the way objects are displayed when the LISTOBJ and LISTAT actions are used. If the flag is set to 64 (i.e. Bit 6 is set) then objects will be listed without newlines between them, forming a valid English sentence - compound listing.

The formats are as follows:

```
SM53 ("nothing.")  - can only occur with LISTAT
object SM48(".")
object SM47(" and ") object SM48(".")
object SM46(", ") object SM47(" and ") object SM48(".")
```

In addition, Bit 7 of flag 53 will be set (i.e. flag will be greater than 127) if any objects were printed. This allows you to determine whether or not a NEWLINE is required to reset colours. A LISTAT action will usually be preceded by a message.

The description of object is constructed from the full description given in the object text table. The preferred format for an object description is:

```
indefinite.article (adjective) noun . extra text
```

where; the indefinite article is "A" or adjectives "An" or

"Some". The Adjective and the Noun should have a lower case letter e.g. 'A small key', 'Some sand' or 'An orange. Rather mouldy'. PAW extracts a description of the object in two ways:

1/ For GET, DROP etc (i.e. "_") the indefinite article is skipped and the description printed upto (but not including) the first full stop. e.g. "I now have the small key.".
2/ For a compound list of objects the indefinite article is forced to start with a lower case letter and the description printed upto (but not including) the first full stop. e.g. "In the bag is a small key.".

Obviously if you don't use underline or compound listings, then you are free to describe objects any way you like.

Important: If an object is to be a container; there must be an unused location with the same number for PAW to use as the 'inside'! i.e. Object objno. 1 would need Location locno. 1 - Not forgetting to mark it as a container ("C") on object weight! This is the major cause of problems with objects as containers.

* * * * * *

Screen Output

Character sets are selected in text by selecting a colour 0-5 then a single DELETE to generate ESCCs 0-5, and will only take effect if a corresponding set has been inserted.

ESCC 7 is a true newline (but does not reset colours as it is part of a message).

ESCC 6 works as a TAB but should be used with care as it will often be suppressed by the text formatter when printing.

Colours and character sets selected within text will stay in force until a NEWLINE action. e.g. To print the number of coins carried by the player (held in flag 100 say); If message 1 contains a RED PAPER control code then an entry in Response (or a Process table) of:-

```
COINS   _   MES     1     ;"<RED>You are carrying"
            PRINT   100
            MESSAGE 2     ;" coins."
            DONE
```

will result in the entire message being printed in red. This is called a compound message and can be used successfully to create neat displays.

Beware the wordrap, for hidden monsters dwell there..

Wordwrap can cause problems with screen output in four situations if you are not careful:

1/ SM14 ("Goodbye...") must end in a Space, control code or newline (as in the start database), or the wordwrap system will never print it!

2/ A similar thing applies for SM16 the ANYKEY message. The last word will not be printed until after the key has been pressed otherwise.

3/ If PAUSE is used to control the speed of text output, you must ensure that a word is not left unprinted. This is automatic if you only use PAUSE after a MESSAGE or NEWLINE action (which print a newline and thus the current word under consideration for wordwrap). Or after a MES which ends in control code. Remember that SCORE and TURNS will suffer the same fate as they are effectively a combination of PRINT and MES actions.

4/ If you use different PAPER colours in text which you rely on the wordwrapping to deal with. Because the change of paper is not wordwrapped, but output immediatly, the word which follows may not fit on the line and the entire end of that line will be a block of the new paper colour.

* * * * * *

Input line

On PAW the state of the players input is controled by the colours in force at the end of SM34 (the cursor) as stated on page 42 (Note 4.). What might not be clear is that because the codes which select a character set (ESCC 0-5) are treated like colour control codes by PAW, they also stay in force. This allows the players input to be in any character set you want!

128K Considerations

When loaded, only page 0 is in use and has about 25500 bytes spare. Pages 1, 3, 4, 6 & 7 are unused but can hold 16K each.

Message & Location text, Connections, Graphics & Default Colours can use pages 1-7, all the other tables only use page 0. All table entries for a location will be held within the same page.

When a location text is inserted null entries are made in the Connection, Graphic & Default Colour tables for that location. The I option on the Location Text menu always inserts in the highest used page. Option B on that menu is used to begin a new page i.e. it inserts on the next page. You have to decide when to use B to start a new page but we would suggest that you use the compressor to gain all memory available before starting another page as you cannot insert extra messages etc onto a page once a new one has been started - only amend existing entries.

The start database supplied has location 0 on page 0 and you will need to consider whether location 1 should be on a new page bearing in mind that all the other tables can only use page 0. We would advise all messages and locations are on pages 1 and up, as page 0 can soon become filled with vocabulary etc.

A similar thing applies to the message text table; I inserts on the highest selected page, B begins another page if it has been initialised by putting a location on it!

If you want your game to run on a 48K spectrum then you must not use any page other than 0.

* * * * * *

Colour Boundaries

Probably one of the most difficult aspects of using graphics on the Spectrum is the problem of the attribute colour system.

This essentially means that only two colours can be used in any one 8x8 pixel group. The areas are shown by the GRID option in the graphics editor. The two colours are the PAPER and INK, the colours of reset and set pixels respectively. Both Paper and Ink can be any of eight values for each 8x8 pixel group.

The fill and shade routines fill in a defined area with a pattern of set and reset pixels. So if you use different INK and/or PAPER colours for two adjacent fills or shades, you will get a coarse stepping effect. This is due to the colours for the second fill affecting pixels in the groups which were set by the first.

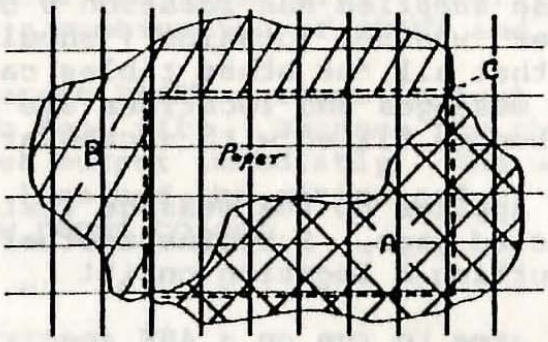
This effect can be overcome in one of four main ways:

1/ Ignore it! This system is used on some pictures in the TEWK game. The effect is minimised by not having shaded or filled areas directly adjacent to each other.

2/ Use shade fully. The shade system can be used along with just two colours to create very effective pictures using varying densities of shade.

3/ Careful positioning as used in the tutorial manual. All areas are defined on exact attribute boundaries. This has the effect of creating very cubist pictures.

4/ Alternate colour. This allows two areas of solid colour (not shades) to be adjacent to each other. It sounds complicated but in principle is fairly straightforward: The diagram shows this far more simply than reams of text:



The outlines of area A and B are drawn. Area A is filled using an INK of the first colour required. Area B is filled using an INK of the second colour required. The block command is used on the rectangle (C) with the PAPER set to be the second colour required and an INK of 8. This leaves area A and B unchanged. The colouring of the PAPER makes the join look continuous.

* * * * * *

Flash and Bright

In order to obtain Flash and Bright print codes on the Spectrum:

From EXTENDED MODE: 8 gives normal brightness.
9 gives bright.
CAPS SHIFT & 8 gives no flash.
CAPS SHIFT & 9 gives flashing.

Character sets

Let's settle this FONTS/CHARACTER SET/UDGs/TYPEFACE lark once and for all! The idea comes down essentially to redefining what those 8x8 pixel blobs the Spectrum calls characters look like on screen.

Normally the Spectrum uses a table of information contained in its ROM to work out what the 96 characters from Space (Code 32) to Copyright (Code 127) look like. This takes 8 bytes per character thus using 8*96 = 768 bytes. It is possible to make the Spectrum look anywhere for this information by changing the system variable CHADD to point 256 bytes below the 768 bytes of data for the character set (256 bytes below? Well that saves having to subtract 32 from the character number when it looks up the data i.e. 32*8 = 256!).

PAW takes care of CHADD for you when you select a new set using Background colours (having loaded it into a previously inserted space.).

To reiterate the method to insert a set is: Use insert (I) to insert space for a new character set - if neccessary. Then Load (L) followed by the set number from 1 to 5 to obtain the filename prompt. You may type your filename with or without the copyright symbol as PAW adds it if needed!

With a blank database the Insert option will insert the 768 bytes of space in the database required for character set 1 (subsequent Inserts will insert 768 bytes for character set positions 2,3,4 and 5). You can then use [L 1] to load the character set by typing in the appropriate name at the character set prompt. After the required Inserts, L 2, L 3 etc will load that set as well. The character sets currently loaded are saved to tape whenever you save the database.

The character sets on side 2 are designed so that you select the set you require when you start writing your game (or during the development). These are loaded as described above.

You can use ANY previously defined set with PAW as long as it is in the form of a 768 byte code block (at any address as PAW loads it into a suitable place). The trick is to ensure that the last (i.e. Tenth) character position contains a copyright symbol (EXTENDED MODE, SYMBOL SHIFT & P), it is also useful to make the name capitals (upper case) as this is the default mode when PAW requests a filename. Then simply follow the procedure for loading a set into PAW.

PAW and printers

PAW prints to the printer using channel #3 (usually "P").

Should this be a ZX thermal printer (or the Alphacom equivalent) then all printout is directed to the required ROM routines. The Thermal printers will only work in 48K mode on a 128K spectrum and will print exactly what is on screen including new character sets and so on. The "C" option on the characters and graphics menus will COPY the screen as normal.

On a 128K spectrum the printer channel defaults to the RS232 port. On the +3 the printer port can be either the Centronics or the RS232. Just use the relevent FORMAT command before loading PAW. If you have Interface 1, it is possible to use the RS232 to print to a printer by opening the "t" channel with the command 'OPEN #3,"t"' before loading the PAW program.

All printout is directed through the port having filtered the character set thus:

0 - 31 are suppressed except for 7 & 13 which print a CR (13) and 6 (TAB) which prints as a space.
32 - 127 are printed as normal.
128 - 162 are printed as a question mark ("?").
163 and 164 are expanded to be the keywords PLAY & SPECTRUM.
165 - 255 are expanded either to the standard keywords in an uncompressed database, or to the correct letter groupings in a compressed one.

In general it should be possible to use any printer interface which redirects channel #3 correctly - with one proviso; it does not mind having its output address redirected. Failing that if you are nifty with the old m/c you can write a specific printer driver (one for the RAMPRINT interface is given below. PAW prints an address when it loads called PRTADD (normally 29587). This gives the address of the vectors within PAW for the printer system defined as follows (char is in A):

```
PRTADD    DEFW PRTRET        ;Address of character output
COPVEC    JP   DOCOPY        ;Make this your screen copy
PRTRET    RET                ;Put your driver here..
```

There are only 48 bytes available for this code, but this should be sufficient for some simple OUTs. Save your driver code from the correct address with a filename with character position 10 set to "A" along with a file of one byte (at address 0?) with the same name ending in "B". You must redirect channel #3 (to anything) to fool PAW that there is a non thermal printer connected before you load it. Then use the Load database option from the main menu to load your driver (PAW thinks it's a database and loads both files, one of which is your driver).

The RAMPRINT interface

After investigation it would appear that several of the tables, (notably the System messages) do not print correctly with the RAMPRINT interface. So the only solution appears to be a printer driver. The listings below save a printer driver suitable for any A17 PAW (with PRTADD at 29587).

These create a machine code program which address the interface directly. Thus the values in the "(c)SET" command will have no effect. This is the reason there are two programs. One creates a driver which assumes the switches in your printer are set to do a newline on carriage return. This driver is called "RP". The other saves a driver, called "RPNL", which sends a code 10 (NL,newline) for every carriage return (13,CR). Basically if you find that the printer keeps printing all the characters on the same line then use the RPNL driver - or change the switches in your printer. Conversely if there is a blank line between every line of text then use the "RP" driver!

You must type in the programs and save a copy somewhere. Now with a clean blank tape run the program and allow it to save the driver (there are two files so you must press enter twice). We advise you to use a new cassette and save the driver (or several copies if you wish) at the start. This is because it must be loaded every time you load PAW.

The process is simple and can be carried out even if there is a database or game already in memory. Select the Load Database (J) option from the main menu and type the name of the driver you wish to load; "RP" or "RPNL". Although you are using the Load Database option you will not affect the database in memory. It merely loads the driver into the correct position. Now all the LPRINT options will work as expected.

Due to space restrictions the COPY command on graphics and characters will not work with the Ramprint interface from PAW. You can of course Dump the picture to cassette and then load it using BASIC to do the COPY.

RP

```
 10 CLEAR 29591
 20 LET a=29592: LET l=0
 30 READ n: IF n=-1 THEN GO TO 50
 40 POKE a,n: LET a=a+1: LET l=l+1: GO TO 30
 50 SAVE "RP        A"CODE 29592,l: SAVE "RP        B"CODE 1,1:STOP
100 DATA 245,245,219,59,230,1,32,250,241,211,59,62,1,211,
         123,62,0,211,123,241,201,-1
```

RPNL

```
 10 CLEAR 29591
 20 LET a=29592: LET l=0
 30 READ n: IF n=-1 THEN GO TO 50
 40 POKE a,n: LET a=a+1: LET l=l+1: GO TO 30
 50 SAVE "RPNL     A"CODE 29592,l: SAVE "RPNL     B"CODE 1,1: STOP
100 DATA 245,245,219,59,230,1,32,250,241,211,59,62,1,211,123,62,
    0,211,123,241,254,13,192,62,10,205,152,115,62,13,201,-1
```

* * * * * *

EXTERN and Loading screens

The EXTERN command can be used to call either your own machine language programs or a section of a BASIC program. This feature can be utilized only in a final game, as the 4K of memory previously occupied by the main menu becomes available (the other 2K is PAWs workspace so hands off!). The important address is the value printed when PAW loads called EXTVEC. You must use the value given on the copy of PAW you save the final game from, or chaos will ensue.

For machine coders only: In order that EXTERN can execute machine code routines you must POKE the three bytes at EXTVEC with a JP to your start address. Register IX must be preserved as it points at Flag 0. All others may be destroyed, but SP must be balanced. Register A on entry contains the parameter to the EXTERN command. If you save the two bytes at EXTVEC+1 they point to the routine which calls a BASIC program - the line number must be in A.

For BASIC users: a POKE extvec,195 - after loading the interpreter file - will enable any EXTERN actions to execute the BASIC line (100+parameter). e.g. EXTERN 10 will goto line 110 in BASIC. To return to the next condact a STOP command in BASIC is required.

You cannot use CLS in BASIC (or m/c), but address EXTVEC+3 contains a routine to do this. i.e. RANDOMIZE USR extvec+3 will clear the screen. In addition EXTVEC+12 contains the line number to print the filenames of the database files on and EXTVEC+9 is the load database and start game entry point.

As an example the following program (without the REM's) must be saved to a fresh tape with SAVE "gamename" LINE 10:

```
10 REM load game
15 BORDER 0:PAPER 0:INK 0:REM so filenames are invisible.
20 CLEAR 28799:REM this is maximum value for RAMTOP.
25 LET extvec=?????:REM whatever it is...
30 LOAD "" SCREEN$:PRINT AT 19,0;:REM stops damage to pic!
40 LOAD "" CODE:REM load interpreter file
50 POKE extvec,195:REM enable EXTERN
```

```
 60 POKE extvec+12,20:REM filenames on line 20
 70 RANDOMIZE USR (extvec+9):REM load database and start game
100 REM line that is called by EXTERN 0
110 PRINT "hello from BASIC":STOP
355 REM line that is called by EXTERN 255!
```

Lines 100-355 could contain GOTO statements if required.

Save your SCREEN$ immediately after the program on the tape. Then from within PAW save the Adventure (using option A) after that.

Important there must be at least 2000 bytes spare for PAWs workspace or your BASIC program will start to disappear! RAMTOP may be lowered to contain your machine code but 2K must be spare!

The flags can be accessed from BASIC by using the PEEK function. The address of flag 0 is (extvec-544). Thus flag 1 is at (extvec-543) and so on. If you want to pass a value back to PAW you can POKE a value into a flag and use the EQ Condition to test it.

If music be the food of love...

Several people have enquired about using music from stand alone music generators such as Wham-The Music Box. It should be quite possible to do this, the only problem that occurs is where to put the code! PAW is a very big program but about 4K is regained on a final adventure where the main menu was. This unfortunately is below 32768 which is no use to music programs due to a timing problem on the Spectrum.

The only suggestion we can make is that you leave sufficient memory free in the database on page 0 to contain the data file. Then use a header reader to discover the start address and length of the "........A" file on the finished game. Adding the length to the start address will give you the start address of the free space in the database, this can then be loaded with the music file (assembled at the correct address of course) from you own BASIC loader as described above.

For fastloaders etc. you may need to stop PAW loading the database itself in a finished game. This can be achieved by adding the following line to your loader:-

```
65 POKE 34884,0:POKE 34885,0:POKE 34886,0
```

The entry address of (extvec+9) in the RANDOMIZE USR statement remains the same. You must also add a line 45 which loads your database file! Of course you could assemble a single block with the interpreter included and allow line 40 to do the load.

Light and Dark

Darkness is becoming something of a cliche in adventures these days, but used correctly it can add to the sense of realism considerably.

Within PAW, darkness is created by setting flag 0 to a value other than 0. This must be done whenever the player moves into and out of darkness. i.e. the move must be done with a GOTO in the Response table, to allow the SET or CLEAR action to occur.

If the player is being provided with a source of light then object 0 is the easiest way of implementing it. A source of light does not have to be a torch or candle, with a little imagination it can be infra-red glasses or a wide beam laser!

Take for example the creation of a night and day cycle, over 24 time frames which we will assume are equivalent to 1 hour.

The entries required in Process 2 are:

```
_       CYCLE EQ      5   0     ;End of cycle
              LET     5  24     ;Start the counter again

_       NIGHT EQ      5  18     ;Nightfall
              SET     0
              MESSAGE x

_       DAY   EQ      5   6     ;Daybreak
              CLEAR   0
              MESSAGE y
```

Importantly if part of the game is underground, or inside a building, don't forget to determine if the player can actually see nightfall and daybreak from where they are, before printing the messages.

* * * * * *

Tick, Tock ow about a Clock cock!

If you want to create a simple clock and calendar system in an adventure, the following method may be of use:

This clock will use our standard 7 days of 24 hours of 60 minutes, but of course your game can use any method of representing time. In addition using a similar technique you could provide a true calendar giving Months and Years.

We will use three flags, 150,151 and 152 to represent minutes, hours and days respectively. In addition flag 153 will be used as described later. An entry in Process 2 will allow the clock to be incremented correctly:-

```
*   *   PLUS    150   1    ;One more minute (or perhaps ten?)
        EQ      150  60    ;One hour done?
        CLEAR   150        ;Zero minutes
        PLUS    151   1    ;Another hour
        EQ      151  24    ;One day over?
        CLEAR   151        ;Zero hours
        PLUS    152   1    ;One more week..
        EQ      152   7    ;All done?
        CLEAR   152        ;And so on if required.
```

You now have flag 150 containing 0 to 59 giving the minutes within the hour. Flag 151 containing 0 to 23 giving the time as a Twenty four hour clock and Flag 152 containing 0-6 as a day of the week!

If you set up several messages thus:

```
Message 1
The clock shows the time as
Message 2
:
Message 3
pm.
Message 4
am.
```

And a sub process (which we will call x) of the following:

```
_   _   COPYFF  151 153    ;Make a copy of the hours
        GT      153  11    ;Is it PM?
        MINUS   153  12    ;Yes so make a 12 hour clock

_   _   PRINT   153        ;Print hour
        MES       2        ;":"
        PRINT   150        ;the minutes
        GT      151  11    ;Is it PM?
        MESSAGE   3        ;Yes so print that.
        DONE               ;All finished

_   _   MESSAGE  4         ;Must be AM. No need for a DONE
```

along with a simple entry in response of:

```
LOOK  CLOCK MES       1
            PROCESS   x
            DONE
```

will allow a clock to show the time. Using another sub-process you could print the day of the week (using seven messages and seven entries!). And so on for the month and year!

A problem can be linked to this clock by testing for specific

times. E.g. if the player can only enter a shop when it is open, say 9am to 5pm every day except Sunday (which we will call day 0) then the conditional movement entries would be similar to:

```
ENTER SHOP  AT       loc        ;Ensure outside shop
            NOTZERO  152        ;Not a Sunday
            GT       151   8    ;At least 9 o clock
            LT       151  18    ;Must be less than six pm
            GOTO     ...        ;Etc to move player to shop

ENTER SHOP  AT       loc        ;Obviously shut
            MESSAGE  "The shop is shut."
            DONE
```

The same system can be used to trigger events at specific times of day, e.g. trains etc. It is probably worth making time pass faster for the player by stepping the clock at 5 or ten minute intervals, or perhaps adding a few extra minutes (or hours) when the player carries out an action which would take longer in real life. Time provides a whole new dimension in an adventure.

* * * * * *

Proper exit listings

In order to create an exits list in keeping with the continuous format object listing you will need to create several messages which are the various directions and several tables thus:

```
Process   4

_   _       CLEAR    200        ;No of messages printed
            CLEAR    199        ;Indicate "Nowhere is the message"
            COPYFF    33 201    ;Save current Verb as we use it

_   _       COPYFF    38 202    ;Use a copy of current location
            LET       33   2    ;See if North a valid move
            MOVE     202        ;Try it
            PROCESS    5        ;Yes so use common routine

_   _       COPYFF    38 202    ;Us a copy (as may have been moved!)
            LET       33   3    ;Try all other directions
            MOVE     202
            PROCESS    5        ;etc..

;Repeat this entry for all direction word values

_   _       GT       200   1    ;If more than one direction printed
            SYSMESS   47        ;" and "

_   _       PROCESS    6        ;Print outstanding message
```

```
-     -     SYSMESS  64        ;full stop
            NEWLINE
            COPYFF   201  33   ;Restore current Verb

Process 5

-     -     GT       200   1   ;At least one already printed
            SYSMESS   65       ;", "

-     -     NOTZERO  200       ;Not the first to be printed
            PROCESS    6       ;Print description

-     -     PLUS     200   1   ;Increment number printed
            COPYFF    33 199   ;Set up direction flag for message

Process 6 ;Useful to convert Movement value into a Message

-     -     ZERO     199       ;No directions matched
            MES      "Nowhere"

-     -     EQ       199   2
            MES      "North"

-     -     EQ       199   3
            MES      "South"
;Repeated for each direction value...
```

Just call this from your response table entry preceded by a message along the lines of "Possible exits are ". This will result in outputs of:

```
Possible exits are nowhere.
Possible exits are north.
Possible exits are north and south.
Possible exits are north, south and east.
etc...
```

With a little imagination you should be able to take care of the special case of only one exit. (i.e. 'is north' etc.).

* * * * * *

Secret exits

To conceal an exit, just don't put it in the Connections table. Then two entries in Response:

```
PUT   CARD  PREP     IN
            NOUN2    SLOT
            AT     locno        ;Where slot is
            SET      200        ;'Flag' that player has done this
            MESSAGE    x        ;"A hidden door appears eastward."
            DONE
```

```
E    _    AT      locno      ;Where slot is
          NOTZERO 200        ;Player has put card in slot
          GOTO        ..     ;wherever
          DESC
```

* * * * * *

Examine

An examine command can be implimented simply by creating a message which decribes an object, we will can it message number x. Then inserting an entry in Event of:-

```
EXAM KEY  PRESENT objno      ;Whatever object number the key is
          MESSAGE x          ;Describe it
          DONE
```

* * * * * *

Invisible Objects

Objects can be 'invisible' by making them not created and using an entry such as:-

```
SEARC DRAW  AT      locno      ;Where draw is?
            ISAT    objno 252  ;Object to 'find' is still uncreated
            CREATE objno       ;create that object here
            MESSAGE    x       ;"In the draw is an _."
            DONE
```

in Response, to create them when the player finds them! Note that the use of underline allows you to use only a single message as the currently referenced object is set by CREATE.

* * * * * *

Password protection

The simplest way to create a password protection system within PAW is to use anymatch ("* *") entries at the very start of the Response table. In addition a flag (we will use 200 for our example) will be used to indicate that the password has been entered (when set). Say the password is defined as a Verb of word value 100. Then the entries needed will be:

```
*    *    EQ      33    100  ;Correct password entered?
          SET     200        ;Password has been entered
          OK

*    *    ZERO    200        ;Password entered yet?
          MESSAGE "Invalid Password"
          DONE
```

Note that we check the current Verb directly (by examining Flag 33), this ensures that all inputs by the player will produce an "Invalid Password" message until he/she enters the correct word.

* * * * * *

Creating Multi-Part Adventures

In order to create a larger (and thus more interesting) play area in an adventure, without sacrificing the quality of the description, you can split the game into smaller sections. It is best to do this with a game that lends itself to having several areas, with only one join between each, this is called a bottleneck. e.g. a game where setting sail on a boat is the final task in the first part.

To allow the score, turns taken and other information to be carried forward into the next game you must use the LOAD/SAVE game position actions. In order to load a game position into a different game to that which it was saved from, you need the same number of locations and objects in each part. In addition, all objects which may possibly be carried forward by the player, must have the same description in all parts.

Let's take a game with 120 locations, that is to be split in half, thus requiring 60 locations in each part. Actually location 60 will exist in both games as the transition location (where the player starts and finishes) and a spare flag (say 26) will be used to indicate which part of the game a position is from. So when the player completes part 1 they are moved to location 60 and flag 26 is set to 1 to show it.

The setup for part 1 would be:

Location 60
End of Part 1 - Prepare a tape to save your position.
(You may save more than one copy if you like).
Please LOAD part 2 and follow the onscreen prompts.

Process 1

```
END   -   AT    60      ;End of game?
          LET   26   1  ;Valid position from part 1.
          SAVE
```

And in part 2:

Location 0
Part 2 - Prepare to load tape with saved position.

Location 60
Any introduction wanted for Part 2.

Process 1

```
START _     AT      0          ;Just starting?
            LOAD               ;Will then be at another location.

START _     NOTEQ   26    1    ;Not a valid position from part 1.
            GOTO    0          ;So request another load.
            DESC

START _     AT      60         ;Just loaded a valid position
            ANYKEY             ;Wait until introduction read
            RESET   1          ;Start game properly at location 1
```

The RESET action does a DESC of the new start location automatically, after setting all objects that aren't carried, worn or at location 60 to their starting position. Note that you should insert any CLEAR actions for flags between the ANYKEY and the RESET as the flags are not affected by the RESET.

Less than

An interesting omission from the comparison facilities of the PAW CondActs is the ability to compare two flags for relative sizes. With a bit of thought a useful sub-process can be created which does the same job - we will call it process z! Using two working flags the routine can be used with any two values - and won't damage the original values either:

```
_     _     CLEAR   252        ;Flag 252 indicates the comparison
            SAME    250 251    ;Are the working flags the same?
            NOTDONE            ;Don't set the done flag

_     _     LET     252   1    ;A value of One indicates 250>251
            SUB     250 251    ;Subtract the working flags
            ZERO    251        ;If the result is zero then 250
            NOTDONE            ;is greater than 251.

_     _     LET     252   2    ;Otherwise flag 250<251
            NOTDONE
```

Pseudo-Intelligences

The main thing to remember is that a character (or PSI) is a word in the Vocabulary (usually a Noun with a value less than 50 so as to be a Proper Noun). Some flags, a series of messages and some entries in one or more Process tables. One flag shows where they are, the messages provide information about their actions and the process table entries tie it all together.

So imagine a character called Sanec who can walk around independently. He is described in the vocabulary as SANEC (word value 25, Noun). Flag number 20 is used to give his location. Process table 3 will deal with speech to him. While Process table 4 will deal with his movements and actions. The following entries allow him to move around when you ask him too. After a short time he will get 'bored' and vanish in a puff of smoke!

Message 1
Sanec did not seem to understand what you said.

Message 2
No one of that name here!

Message 3
Sanec replies "hello" in a gruff voice.

Message 4
Sanec wanders that way as he has nothing better to do.

Message 5
Sanec the wizard is here.

Message 6
Sanec 'politely' ignores what you say.

Message 7
Sanec turns to face you and in his gruff voice announces;
"I'm bored with all this, I'm off to a bigger game"
and promptly vanishes in a puff of green smoke!

First; Sanecs' presence at a location must be announced. So in Process 1 (which is called after every describe of a location) we check if he is here i.e. flag 20 (his location) is the same as flag 38 (our location). Note that we ensure we are not at location 0 as this is always an introduction screen.

```
SANEC _     SAME      20  38  ;Ensure Sanec is here
            NOTAT      0      ;Player is not in location 0.
            MESSAGE    5      ;Say Sanec is here.
```

To deal with speech too Sanec, we need two entries in Response as follows:-

```
SAY   SANEC PREP      TO        ;this could be omitted to allow
                                 short Verb Noun sentences to be
                                 understood
            SAME      20   38   ;Make sure Sanec is here
            PROCESS    3       ;Deal with any speech
            DONE               ;Prevent drop through with new LS.

SAY   _     PREP      TO        ;again optional
            MESSAGE    2       ;no one of that name here!
            DONE
```

The following entries in Process 3:-

```
*     *     PARSE              ;This entry always carried out to
                                 convert the input string to a LS.

            MESSAGE    1       ;PARSE comes here if it fails to
                                 find a valid phrase
            DONE               ;Note that the LS is corrupt and no
                                 further table entries must be
                                 executed

HELLO       MESSAGE    3       ;Assuming HELLO is a verb in vocab
            DONE               ;So that SAY TO SANEC "HELLO" works

_     _     LT        34   14  ;A movement word said to Sanec?
            MOVE      20       ;See if a connection for that way
            MESSAGE    4       ;Come here and tell player if so
            DONE

_     _     MESSAGE    6       ;He ignores you (i.e. nothing else)
```

Obviously many more entries would be required to give Sanec an appearance of understanding speech, but with a few clever entries he can give a wide variety of responses.

Finally; to give Sanec a chance of disappearing when bored, we need an entry in Process 4 of:

```
SANEC _     EQ        20    2  ;At location two?
            CHANCE    10       ;10% chance
            SET       20       ;Location 255 does not exist
            AT         2       ;are we where he was?
            MESSAGE    7       ;POOFF! - tell player he disappeared
```

And an entry in Process 2 to call table 4 regularly:

```
SANEC _     PROCESS 4
```

In this way a very convincing character can be built up. They add a great deal to the sense of realism in games. Especially if interaction with them is required as part of the solution.

Moving PSI's

A common request we had in the technical department was reporting the movement direction of PSI's. Now, having chained a programmer to a bench for a few hours, we can deal with the problem.

Continuing with the example of our wizard Sanec...

Currently Message 4 is printed no matter which direction he moves. Change the message to read "Sanec wanders " (note the space). Now add six messages 8-13 (normally if you were allowing NorthWest, SouthEast and so on you would need 10 messages). Which read "South.", "East.", "Up." etc

Next create a Sub-process using the Begin new table option (we will assume it creates Process 5), and place the following entries in it:

```
SOUTH _     MESSAGE   8

EAST  _     MESSAGE   9

WEST  _     MESSAGE  10
```

etc. Note that you do not need any DONE actions as the only entry which will trigger will be the one representing the direction Sanec moved.

Now modify the _ _ entry in Process 3 which deals with Sanec's movement to be:

```
_     _     LT        34   14   ;A movement word said to Sanec?
            MOVE      20        ;See if a connection that way
            MES        4        ;Yep so report "Sanec wanders "
            PROCESS    5        ;and the direction
            DONE
```

Note that the MES action does not print a NEWLINE and so the reports will be of the form:

```
Sanec wanders North.
Programmer goes Up the pub....
```

* * * * * *

Having them follow

In order for a PSI to be able to follow the main character, you need only a few simple entries. Continuing with the example of SANEC, we will use flag 21 to say what he is currently doing. A value of zero indicates he is wandering as normal, whereas a value of 255 (SET) will indicate that he is following the player.

Pseudo-Intelligences

In this case any entries which allow SANEC to move by himself must ensure that flag 21 is zero (there are none given in the manual). The following entries are needed in Process 3 (the one which deals with speech to Sanec - IF HE IS AT THE SAME LOCATION AS THE PLAYER):-

```
FOLLO _     SET       21      ;Make SANEC follow
            MESSAGE    x      ;"Sanec says 'OK bud'." etc
            DONE

STOP  _     CLEAR     21      ;Stop him following
            MESSAGE    x
            DONE
```

You might like to add other entries to give him replies to FOLLOW when he is doing so already and so on, or perhaps use CHANCE to make it less likely he will just follow you like a sheep!

Ok now to make the flag do something. Put the following entry in Process 4 (which is called from Process 2 each time frame):-

```
SANEC _     NOTZERO   21        ;Sanec is supposed to follow player
            NOTSAME   20   38   ;He ain't where player is
            MESSAGE    y        ;"Sanec follows you".
            COPYFF    38   20   ;Move him where player is
```

It might be best to position this after the entry allowing Sanec to disappear when 'bored', as it will allow him when following you to still appear sentient.

* * * * * *

And fighting them

Moving away from SANEC lets look at some rather aggressive PSI's; The tables given below are a guide only and some development will be necessary on your part...

A combat system between the player and a PSI can be set up in the following way: Assign a flag to both the player and the PSI which will hold the Combat Value, which will represent the ability of the PSI and player to inflict damage, avoid hits etc. This is a sort of Dexterity, experience and strength all rolled into one - it would be possible to separate these functions but the interactions would be made more complex to code for.

Flag 22 will hold the PSI's CombatValue and Flag 60 to hold the players. It is assumed that Object 2 is a Sword and Object 3 a shield. You may find it useful to make the combat routine general purpose if you have several PSI's and use a sub-process to print the PSI's name when needed according to a flag value.

The first thing we need is a way of comparing two flags. This is

provided by the compare sub-process created elsewhere in this book. Armed with this information we can create an interaction between the player and a PSI. We create a sub-process (which will be called x) that will be called using a PROCESS action each time the PSI decides to take a swipe at the player. A similar table would need to be created for the PSI. This routine will have the following call parameters:

Flag 100 - Potency of the weapon used.
Flag 101 - CombatValue of PSI.

There will also be a check made to see if the player has any defences that can be used to fend off the attack - In this case we will only check for the shield.

```
_   _     COPYFF  101 251   ;Compare PSI against player
          COPYFF   60 250
          PROCESS   z
          EQ      252   1   ;Is player superior to PSI
          CHANCE   60       ;Less likely that attack happens
          MESSAGE  ..       ;" with a weak and glancing blow!"
          DONE

_   _     CARRIED   3       ;Player has the shield?
          CHANCE   70       ;Good chance it helps
          MINUS   100   2   ;Reduces the potency of the attack
          MES      ..       ;" it glances off the shield "

_   _     ZERO    100       ;If no force left..
          MESSAGE  ..       ;"and misses."
          DONE

_   _     MINUS   100  60   ;Make the attack
          MESSAGE  ..       ;"striking a good blow."
```

Note the use of compound messages to produce a valid phrase. This basic shell could be expanded to use further comparisons to account for the difference between combat values in the strength of attack and so on.

It would be called in the following way from the control process for the PSI thus:

```
ATTAC PLAYE NOTZERO flagno    ;The flag that says we are fighting
            RANDOM   90       ;A random element for the PSI
            LT       90  70   ;70%
            COPYOF    2  90   ;allows him to chose a weapon
            SAME     90  38   ;or if you like
            LET     100   4   ;how vicious he feels!
            COPYFF   22 101   ;PSI strength
            MESSAGE  ..       ;"The PSI swipes with a broadsword"
            PROCESS   x       ;
            DONE
```

```
ATTAC PLAYE NOTZERO flagno
          LT      90  50  ;50% etc...
```

Similar entries in Response which call the PSI's attack sub-process would allow the player an attack capability!

You will of course need an entry at the start of Process 2 along the lines of:-

```
*    *     ZERO      60        ;Combat value zero?
           MESSAGE   ..        ;"You crumple to your knees..."
           TURNS
           END                 ;!!
```

* * * * * *

Using Multi-Parse

The ability to give a PSI a list of commands to do has incredible possibilities for the creation of synchronized problems. Where both the PSI and the player must work together.

These sort of problems can add a whole new dimension to adventures and are well worth considering, here are some suggestions:

Imagine a game with a room that is instant death for the player which contains an object that he requires. You could instruct a PSI to go in, get the object and come back out.

Say that in order to kill a certain monster you needed a simultaneous attack from three characters. You could use the following:
SAY TO PSI1 "WAIT,KILL MONSTER"
SAY TO PSI2 "KILL MONSTER"
KILL MONSTER
All three KILL MONSTERs would be carried out in the same time frame.

They say the best way to demonstrate something is by example. So here goes with a short listing of a game with only one problem:

In order to get out of a cavern you need to be lifted on a platform controlled from another room. This can only be achieved by giving a PSI (who happens to be hanging around) a list of things to do. I.e. Go to the cavern and pull the rope. While you in the meantime step onto the platform and wait...

Flag Usage

20 - Location of PSI
21 - copy of flag 20 during movement processing

60 - when 0 indicates platform is on floor, 1 - held by PSI and 2 - held by Player.

195 - Players Verb/Pronoun-Noun Saved
196 - Players Adverb/Pronoun-Adjective Saved

197 - Number of Logical Sentences waiting for PSI
198 - Next storage flag group to store LS in
199 - Next storage flag group to get a LS from
200-206 - Store 0 for LS
207-213 - Store 1 for LS
214-220 - Store 2 for LS

Notes

The principle of the multi-parse is that the entire string is broken down into a list of LSs that the PSI will be required to do. These LSs are then stored (saved if you like) in some flags to be doled out, one per timeframe (use of process 2).

The LSs waiting for the PSI to do are held in a 'queue' which is a computer term for an ordered list. They are actually held in a 'round robin fifo queue'. fifo stands for 'first in first out'. i.e. the first LS given to the PSI must be the first it carries out. While 'round robin' indicates that the LS storage used goes around the available storage flags in a circular motion. i.e. it goes back to the beginning when it falls off the end!

Thus the groups of flags will be used in the order; Store 0, Store 1, Store 2, Store 0 etc. The use of only three storage areas means that only three commands can be queued for the PSI, there is no reason why this cannot be expanded upon. Indeed if you only needed Verb Noun commands to be given to PSI's you could save only those parts of the LS. Thus requiring only two flags per LS not 7!

The extraction of multiple phrases is done by a single process table which calls itself to get the next phrase. This is known as 'recursion' and is simpler than a sequence of entries doing PARSE and PROCESS calls etc. It does limit you to 9 phrases in a string though - Why? (Clue: you can only nest PROCESS calls to a depth of ten.)

Pseudo-Intelligences

Locations

Location 0
I am in a large cavern. On the East wall, high up, is an entrance from which a shaft of light descends. A lifting platform, obviously intended as a means of getting to the entrance, is linked via a series of pulleys on the roof to a steel cable which disappears into a hole in the North wall just above a tunnel.
Location 1
I am standing on a platform
Location 2
I am standing on a ledge overlooking a lush green valley. To the West is an entrance to a large cavern.
Location 3
I am in a small ante-room. A twisting tunnel leads South. A steel cable hangs from the ceiling.

Connections

Location 0 N TO 3
Location 1
Location 2
Location 3 S TO 0

Messages

Message 0
A PSI is here.
Message 1
The PSI doesn't understand.
Message 2
You have said enough to the PSI.
Message 3
You speak to the PSI.
Message 4
The PSI cannot do that.
Message 5
The PSI pulls on the cable.
Message 6
The PSI releases his grip on the cable.
Message 7
The PSI stands on the platform.
Message 8
The PSI steps off the platform.
Message 9
The PSI leaves.
Message 10
A platform
Message 11
The platform
Message 12
rests on the floor of the cavern.

```
Message  13
 hangs just inside the opening.
Message  14
 now
Message  15
 which
Message  16
 jars into motion.
Message  17
A PSI arrives.
Message  18
The PSI can't go that way.
Message  19
You release your grip on the cable.

Response Table

*     *      EQ        60   2        ;Player holding cable?
             CLEAR     60            ;Release it
             MESSAGE   19
             PROCESS    8            ;Cancel DONE flag

I     _      INVEN

GET   PLATF  PREP     OFF            ;Movements on and off platform
             AT         1
             ZERO      60
             GOTO       0
             DESC

GET   PLATF  PREP     OFF
             AT         1
             GOTO       2
             DESC

GET   PLATF  PREP     ON
             AT         0
             ZERO      60
             GOTO       1
             DESC

GET   PLATF  PREP     ON
             AT         2
             NOTZERO   60
             GOTO       1
             DESC

R     _      DESC

QUIT  _      QUIT
             TURNS
             END
```

Pseudo-Intelligences

```
SAVE  _      SAVE

LOAD  _      LOAD

RAMSA _      RAMSAVE

RAMLO _      RAMLOAD 255

SAY   PSI    NOTSAME  20  38        ;Talk to PSI if in cavern
             ATLT      2            ;or on platform etc
             LT       20   2
             PROCESS   3
             DONE

SAY   PSI    SAME     20  38        ;otherwise have to be same
             PROCESS   3            ;location
             DONE

SAY   PSI    MESSAGE  20
             DONE

WAIT  _      OK

PULL  CABLE  AT        3            ;Allow player to hold cable
             ZERO     60
             LET      60   2
             OK

RELEA CABLE  OK

STAND PLATF  PREP     ON
             AT        0
             ZERO     60
             GOTO      1
             DESC

Process 1
*     *      EQ       31   0        ;Deal with start of game
             EQ       32   0
             MODE      1   1        ;Continuous scrolling text
             TIME      8   3        ;Timeouts
             INPUT     7            ;Input at bottom of screen

*     *      NEWLINE                ;Always start a fresh line
             ATLT      2            ;In cavern or on platform
             MES      11            ;"The Platform"

*     *      AT        2            ;Outside cavern
             NOTZERO  60            ;Platform is at top
             MES      10            ;"A Platform"
```

```
*     *     ATLT       2              ;In cavern or on platform
            ZERO      60              ;which is on floor
            MESSAGE   12              ;" rests on the floor."

*     *     ATLT       3              ;Anywhere except anti-room
            NOTZERO   60              ;Platform at entrance
            MESSAGE   13              ;" by the entrance."

*     _     ZERO       0              ;Standard PAW dark stuff
            ABSENT     0              ;for Object list
            LISTOBJ

*     _     PRESENT    0
            LISTOBJ

_     _     SAME      20  38          ;PSI where player is?
            MESSAGE    0              ;"There is a PSI here."
```

Process 2

```
*     *     NOTZERO  197              ;Any commands for PSI
            LET       58 128          ;Allow word matching
            PROCESS    5              ;extract next action for PSI
            CLEAR     58              ;Prevent word matching
```

Process 3 - Deals with speech to PSI

```
*     *     COPYFF    46 195          ;Save 'IT' for player
            COPYFF    47 196
            SET       46              ;No IT at mo!
            SET       47
            PARSE                     ;Get a phrase
            MESSAGE    1              ;not one there
            COPYFF   195  46          ;Restore IT
            COPYFF   196  47
            DONE                      ;all over

*     *     MESSAGE    3              ;"You speak to PSI"
            PROCESS    4              ;extract and store phrases
            COPYFF   195  46          ;Restore IT
            COPYFF   196  47
```

Process 4 - This will extract and store up to three phrases although this could be expanded with a few simple changes/extra entries. Note that this is Recursive as it calls itself!

```
*     *     EQ       197   3          ;Max of three phrases in queue
            MESSAGE    2              ;"Said enough to PSI."
            DONE

*     _     ZERO     198              ;Use store 0?
            COPYFF    33 200
```

```
            COPYFF   34 201
            COPYFF   35 202
            COPYFF   36 203
            COPYFF   43 204
            COPYFF   44 205
            COPYFF   45 206

*     -     EQ      198   1          ;Use store 1?
            COPYFF   33 207
            COPYFF   34 208
            COPYFF   35 209
            COPYFF   36 210
            COPYFF   43 211
            COPYFF   44 212
            COPYFF   45 213

*     -     EQ      198   2          ;Use store 2?
            COPYFF   33 214
            COPYFF   34 215
            COPYFF   35 216
            COPYFF   36 217
            COPYFF   43 218
            COPYFF   44 219
            COPYFF   45 220

-     -     PLUS    197   1          ;One more phrase stored
            PLUS    198   1          ;Next store
            EQ      198   3          ;reached the last?
            CLEAR   198              ;Go back round

-     -     PARSE                    ;Get another phrase
            DONE                     ;No more there so finished

-     -     PROCESS   4              ;Store it
```

Process 5 - Extracts the next phrase from store for the PSI

```
*     *     COPYFF   33 195          ;Save Verb/Adverb of player
            COPYFF   36 196

*     -     ZERO    199              ;Store 0?
            COPYFF  200  33
            COPYFF  201  34
            COPYFF  202  35
            COPYFF  203  36
            COPYFF  204  43
            COPYFF  205  44
            COPYFF  206  45

*     -     EQ      199   1          ;Store 1?
            COPYFF  207  33
            COPYFF  208  34
            COPYFF  209  35
```

```
            COPYFF  210  36
            COPYFF  211  43
            COPYFF  212  44
            COPYFF  213  45

*     _     EQ      199   2        ;Store 2?
            COPYFF  214  33
            COPYFF  215  34
            COPYFF  216  35
            COPYFF  217  36
            COPYFF  218  43
            COPYFF  219  44
            COPYFF  220  45

_     _     MINUS   197   1        ;One less in store
            PLUS    199   1        ;Extract next from one more
            EQ      199   3        ;Reached end?
            CLEAR   199            ;Back to start

_     _     PROCESS   6            ;Do the job
            COPYFF  195  33        ;Restore player Verb/Adverb
            COPYFF  196  36

Process 6 - Commands that can be given to PSI

*     *     EQ       60   1        ;Holding Cable?
            AT        3            ;Where player can see PSI?
            MESSAGE   6            ;"PSI Releases grip"

*     *     EQ       60   1        ;Holding cable?
            CLEAR    60            ;Release grip.
            ATLT      3            ;Can player see effect?
            MES      11            ;Describe "The platform"
            MESSAGE  16            ;" jars into motion."
            MES      11            ;"The platform"
            MES      14            ;" now"
            MESSAGE  12            ;" rests on the ground."

GET   PLATF PREP    OFF            ;GET OFF PLATFORM
            EQ       20   1        ;PSI on it?
            ZERO     60            ;Platform on ground?
            CLEAR    20            ;Put PSI in cavern (loc 0)
            ATLT      2            ;Can player see it?
            MESSAGE   8            ;"PSI steps off."
            DONE

GET   PLATF PREP    OFF            ;GET OFF PLATFORM
            EQ       20   1        ;PSI on it?
            LET      20   2        ;Platform by entrance?
            ATLT      2            ;Player see it?
            MESSAGE   8            ;"PSI steps off."
            DONE
```

Pseudo-Intelligences

```
GET    PLATF PREP     ON                ;GET ON PLATFORM
             ZERO     20                ;PSI on ground?
             ZERO     60                ;along with platform?
             LET      20   1            ;Move PSI to platform
             ATLT      2                ;Can player see it?
             MESSAGE   7                ;"PSI steps on."
             DONE

PULL   CABLE EQ       20   3            ;PSI in anti-room?
             ZERO     60                ;with no one holding cable?
             AT        3                ;Is player here as well?
             LET      60   1            ;PSI holding cable
             MESSAGE   5                ;"PSI grips cable."
             DONE

PULL   CABLE EQ       20   3            ;PSI in anti-room?
             ZERO     60                ;with no one holding cable?
             ATLT      2                ;Can player see result?
             LET      60   1            ;PSI holding cable
             MES      11                ;Describe "The platform"
             MESSAGE  16                ;" jars into motion."
             MES      11                ;"The platform"
             MES      14                ;" now"
             MESSAGE  13                ;" hangs by the entrance."
             DONE

RELEA CABLE  DONE                       ;Is done by any action!

STAND PLATF  PREP     ON                ;STAND ON PLATFORM
             AT        0                ;See above GET ON PLATFORM
             ZERO     60
             LET      20   1
             MESSAGE   7
             DONE

WAIT   _     DONE                       ;Do nothing for a time frame

_      _     LT       33  14            ;Movement?
             PROCESS   7                ;Deal with it
             DONE

_      _     CLEAR   197                ;Can't do it so cancel any
             CLEAR   198                ;waiting LS for PSI.
             CLEAR   199
             SAME     20  38            ;Is player where PSI is?
             MESSAGE   4                ;"PSI can't do it."

Process 7 - Deal with movement for PSI

*      *     COPYFF   20  21            ;Save current location
             MOVE     20                ;Try and move
             NOTSAME  20  21            ;Did location change?
             SAME     21  38            ;Was player there?
```

```
        MESSAGE   9              ;tell them "PSI leaves."

*   *   NOTSAME  20  21         ;Somewhere new?
        SAME     20  38         ;Where player is?
        MESSAGE  17             ;tell them "PSI arrives."

*   *   SAME     20  21         ;No change?
        CLEAR   197             ;Can't go that way so
        CLEAR   198             ;clear any outstanding LS
        CLEAR   199             ;for PSI
        SAME     20  38         ;Player here?
        MESSAGE  18             ;tell them.

Process 8

*   *   NOTDONE                 ;Cancel the 'done' flag
```

Playing

If you do type this in you may like to try some of the following sequences from the starting position...

GET ON PLATFORM, SAY TO PSI "GO NORTH,PULL CABLE AND RELEASE IT" THEN GET OFF IT.

This shows the independence of IT for Player and PSI.

SAY TO PSI "N,PULL CABLE",STAND ON PLATFORM,GET OFF IT

Is the solution, although if you wished to lower the platform after.

SAY TO PSI "N,PULL CABLE & RELEASE IT",GET ON PLATFORM AND OFF IT

Would leave you outside without a platform, while...

SAY TO PSI "STAND ON PLATFORM,WAIT THEN GET OFF IT". N,PULL CABLE,RELEASE IT,S

Would leave you without a means of exit and the PSI outside!

Hints for those intending to sell their work

The adventure market today is very much a hobby market. The majority of sales are made by small companies - often the author and a room in their house. The larger companies tend to stick only to high profile, hideously expensive to write, moneyspinning arcade games. So it is well worth considering marketing the game yourself or pursuading one of the smaller companies to do this for you. If you want to interest them in your game you will have to ensure:

1/ The game is original. Most software houses will know about the vast majority of games previously launched and will spot a ripoff immediately. This also applies to characters 'borrowed' from books or films, ensure you have permission from the copyright owners before using them.

2/ The spelling is thoroughly checked - looking it over once doesn't count! Check any word you are doubtful of with a good dictionary. Also ensure your grammar and punctuation are correct.

3/ The game has been extensively tested. Ask your friends round for an evening and let them try it, others will often try inputs you never dreamed of.

4/ The documentation is legible and neatly presented (typed if possible). See below for some hints on what is needed.

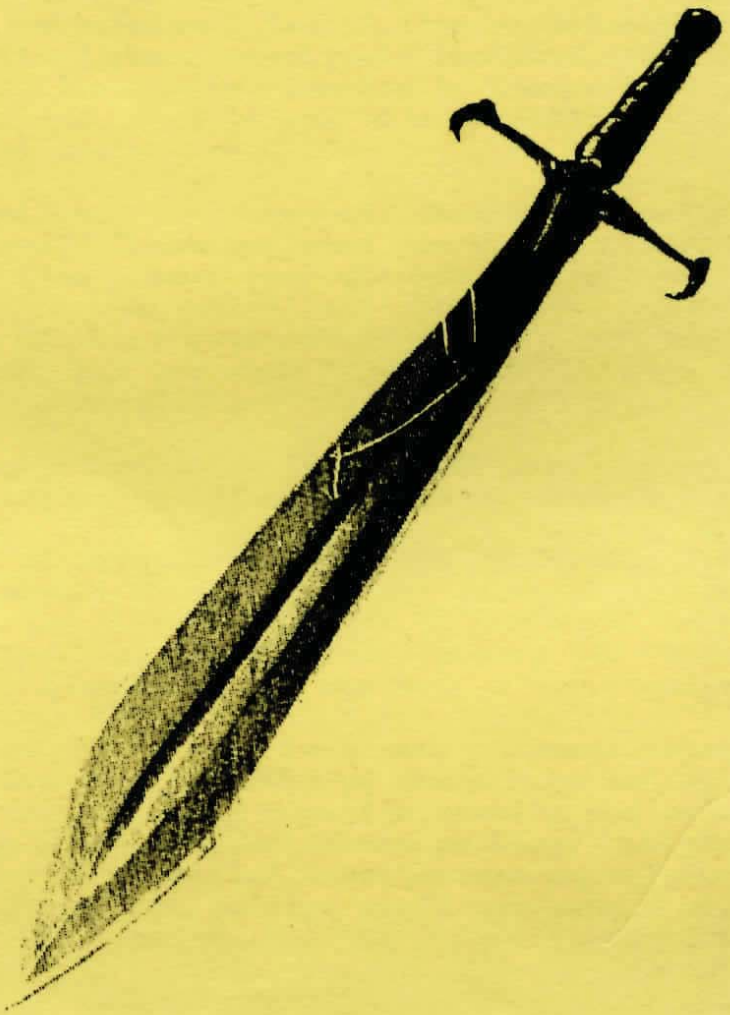
5/ Expect to wait a few weeks for a reply, (sm)all software houses are always overworked.

Most important of all do not expect to become an instant millionaire, adventures are strictly a hobby market.

Documentation for a game is a very personal thing and depends on the game a lot. The majority of adventures can be documented with a tidy map (which shows location numbers and object positions), along with a scene setting sketch perhaps. In addition a step by step solution (or detailed instructions on solving each problem) will be required. It is also worth devising a cryptic help sheet which you can provide for the player should they get stuck. Don't forget that to allow you to make corrections later, a list of the uses of each of the flags will be required.

You obtain copyright by what is known as the common law rights of the creator of a work. I.e. merely writing the game gives you the copyright. Although if you ever needed to prove this you would need some proof. This cen be obtained by posting a sealed, written (and disc/tape recorded) copy to yourself and then NOT opening it. Or you could deposit a copy at a bank and obtain a DATED receipt - although a fee would be charged for this service.

But the most important thing is to enjoy yourself!



© 1986 Gilsoft

Published by Gilsoft
2 Park Crescent, Barry, South Glamogan CF6 8HD
Telephone Barry (0446) 732765

All rights reserved, unauthorised copying, hiring or lending strictly prohibited

PAW Prints

by George March

In this first issue, I'd like to first answer the three most asked questions concerning any future adventure writer, which are..
(A) I know I want to write something, but I don't know what, I've no imagination! (B) Either I've got a few ideas, but that's it! Or I've got so many ideas that I don't know when to stop (this happens very rarely mind) and (C) I want to be able to do such-and-such, but I don't know how to do it?
To which I answer.. (A) Rubbish! Everyone's got an imagination, just some peoples are a bit more abundant. So what can you do to get any ideas? Just like me, you can try asking other people what they would do in a certain situation, given the objects you have in your game to do as they wish with! Your local library will help, or T.V, etc, and are a mine of factual information. The T.V and films (my favourites 'James Bond', 'Our man' and 'In like Flint', 'Mcgyver', etc) are brilliant I find for giving me a prod in the right direction, and newspapers and magazines are full of stories about airplane hi-jacks, kidnappings, etc, for story ideas, and even though I can't condone the nicking of other peoples ideas for stories (though the borrowing of routines is usually okay), a spoof adventure (eg, the 'Boggit') is a workable (if not original) idea. But the way I often go about it, is just to first sketch my few ideas down on paper, and to write a short story with them (the shorter the better!), taking any messages, location descriptions, etc, straight from there! (B) A few ideas (or a short story) are all you'll ever need to start any adventure, not too wordy to start with mind, just to get you into the way of things, and as far as having too many ideas, not everyone will be a good one, so don't try putting too many into a game to start with. Why not keep some routines for a second game (or a larger 128 version), after all a game never needs to be very big to be brilliant, 'Behind closed doors' by Zenobi for example, which only has one location! And (C) if you're ever stuck with something, ask someone else! So why not try writing to me, see the address at the end, or care of the 'Coder'. And even if I can't figure out a problem for you, another reader out there maybe able to! But there's never any need to be stuck with an idea, or have to do without! So why not write for help, or with any ideas, routines, etc, which you think could help someone else, so have a heart and help your fellow programmers and myself write a great column!
Right onto the column properly now, there are quite a few things I wanted to be able to do with the P.A.W for my own creations when I first started out, but never could quite handle, as there was nobody to help until now! Which included the printing of available exits on screen, for example in process 4 the routine could be something like this..

```
_ _ COPYOF 38 110 LET 33 A MOVE 110 SYSMESS A
```

The first action makes a copy of the players present location number into a blank flag (which I've used flag 110 as), the second gives the verb a direction number (I've used the number A as an example), ie, verb number 2 is actually the word 'south'. This next action moves the blank flag with the same room number as the player in the direction given by LET 33 ?, and then, if the flag is able to move in the direction given, the name of the direction is printed. Please note that a copy of the routine above exist for each available direction, just altering the direction number in LET 33 ? to match, and all directions are measured using those available in the connections table. For example..

```
LET 33 14 MOVE 110, etc.
```

Would give a sysmess like 'An open lift door', only if the exits from

the players present room had an exit leading into an elevator, and ENTER was given as verb 14 being said exit, but for example what if there's a locked door which can be opened, then the routine would be..

_ _ AT R ZERO 60 SYSMESS B

So is the player in room R (eg, where the lift doors are?) and have the doors been openned yet? If not then print sys B.

_ _ AT R NOTZERO 60 SYSMESS C

Just as before, just giving a differant answer if the door has been openned. Sysmess B = 'Some closed lift doors. ' and C = 'Open lift doors. '. You must remember the space after each full-stop, for these exits to be printed as a valid sentence. Now the routine in response which prints these exits on screen might be..

EXITS _ SYSMESS D PROCESS 4 NEWLINE DONE

Sysmess D = 'Obvious exit(s) are ' again remember the space! This action then prints an sys without a newline, visits the process (as above) which prints the name of the exit(s) available, and then newlines it to look like a proper message. Now this is okay, if the writer only wishes the directions to be printed on request, but what if we want them to be on screen all the time? Well these routines placed in process 1 will do the trick..

* _ 0 NEWLINE ZERO 0 ABSENT 0 LISTOBJ NEWLINE SYSMESS D PROCESS 4 NEWLINE PROTECT

* _ 1 PRESENT 0 LISTOBJ NEWLINE SYSMESS D PROCESS 4 NEWLINE PROTECT

The first four lines in * _ 0, and the first two in * _ 1 are just as P.A.W contains normally, with just another line space inserted between any objects printed, and/ or directions printed on screen, sysmess D is just the same as in the EXITS _ routine above. I've used another newline to print a line space, after the routine visits the process which generates the exit names, just to make it look a bit cleaner, and the protect action forces any inputs to scroll up underneath the location text (or whatever else is on screen). Here's a simple idea for response to allow the player to change the contents of the open/closed door flag.

OPEN DOOR AT R ZERO 60 SET 60 OK

So if the player opens the door at a particular room (room R), and the door is not openned yet, the routine marks the door as being open and prints the sysmessage 'Okay'. Now the messages used in process 4 must be either MES's on their own, or SYSMESSes, but neither must have a newline, as the routine will look real funny otherwise! And won't be printed as a continues message as they should (eg, 'Obvious exit(s) N. SE. ENTER. '. One of my other problems, was the creation of the ever popular wandering monster bit, which may be of use to a new writer, especially a 128 owner, so these routines first tell whether the wandering whatnot is in the same location as the player, and then generates a random movement pattern (or any other action, such as combat) for it! So first of all in process 1 just after the..

* * 0 AT 0 ANYKEY GOTO 'start room', etc

We need an action which finds and holds the moving P.S.I's present location..

* * ZERO 31 COPYOF 13 65

If this is the very first time the enter key is pressed in a game, the next action makes a copy of the P.S.I's present location into a blank flag (flag 65) and this next action..

* * RANDOM 68

Randomizes another flag, from which the random directions are taken, and is then used as the number of the verb to move the P.S.I in (see: LET 33 ?/MOVE 65 for printing exits on screen). This must 'not' be included in the ZERO 31 routine. This next piece tells if object 13 (which I've taken as our wanderer) is in the players location, and then goes onto a process dealing with its movement, etc.

* _ PRESENT 13 PROCESS 5 DONE

With me so far? Good, then for process 5 we need to handle the finding of the direction the P.S.I is to move in, and then the printing of its direction on screen. So is the random number generated in process 1 less than 15 (ie, a direction?)

* _ 0 LT 68 15 PROCESS 6 MOVE 65 COPYOF 65 13 LET 51 13 SYSMESS F PROCESS 7 DONE

If so, then the routine goes to the process dealing with the generation of the direction chosen by the random number, if the flag is able to move in the direction (sometimes it won't, if the random number isn't the same as a connection) given, and is thus the same as that of a connection, then the copy of the P.S.I's new room is copied back into the P.S.I's present location flag, which then moves the P.S.I in the direction randomly made, it prints sys F and then prints the direction the P.S.I moves off in. Sys F _ 'The _ wanders off '. Remember the space after 'off '. Now the reason I inserted LET 51 13 before the sys is just to allow this routine to be used for other P.S.I's if wished, as this is the flag number which prints the name of an object given in place of an underline in a message. Yes, but what if the random number is not available as a connection? Well that can be handled just like this..

* _ 1 LT 68 15 MESSAGE 1 DONE

The process goes to the routine above, if it cannot move the P.S.I.

* _ 2 GT 68 15 MESSAGE 2 DONE

* _ 3 GT 68 24 MESSAGE 3 DONE, etc

I've left message 1, 2 and 3 upto the writer, and as many exist as the writer wishes, just by measuring whether the random number is greater/ less than a value specified, and now onto process 6 which generates the random number needed, and as before one exists for each available direction, exept for the last one which is taken care of without a chance option, as it is the last one in the process (ie, nothing comes after it)

* _ 0 CHANCE 10 LET 33 2 DONE

* _ 1 CHANCE 10 LET 33 3 DONE, etc upto..

* _ 12 LET 33 14 DONE

These use the conversion nouns from 2 - 14 (or less if the writer wishes to give the player less directions)
This is the last process in our sequence of three (process 7) and is the process dealing with the printing of the direction the P.S.I has moved off in, and again there's one for each available direction.

* _ EQ 68 ? MESSAGE ? DONE

Is the random number used in the movement of the P.S.I the same as an available direction? Again this uses the conversion nouns from 2 - 14, if so then the relevant message is printed, for example 'N. ' for north, 'U. ' for up, etc. Here is also a nice little routine of mine (which Larry Horsfield may (or may not) have used in the 'Probe' (thanks Larry!)), to calculate whether one flag is greater/less than another (something the P.A.W is saddly lacking), and is best used in a process table called from response when needed..

* _ CLEAR A SAME B C NOTDONE

So first of all we need to clear the contents of a chosen flag (A) to use as a counter, and if the two flags you wish to compare are the same then the process moves onto this next routine.

_ _ LET A 1 SUB B C ZERO C NOTDONE

Now if we make counter A equal to 1, and then take the contents of B away from C, if C is zero, then B is obviously greater than C!

_ _ LET A 2 NOTDONE

But if C is not zero, then it comes here and makes A equal to 2, as B is then less than C, ie, if A equals 1 then B is greater than C, but if A is equal to 2, then C is greater then B, which could be used to test if one characters might is stronger than anothers like in a fight, for flags B and C could both be random, but I'll move onto some fight routines next month, so here's where to get in touch for any help you're in need of, or for any hints you can give to other people, also please enclose a S.A.E if you want a personal reply, otherwise it'll get printed here instead..

G. E. MARCH, 93 ROBERTS STREET, NEWCASTLE UPON TYNE, NE15 6BE

FOOD CHART

Devised by C.Hester

1) "Baby I Love Your Whey" - Will To Powder
2) "Don't You Forget About Meals" - Simple Minds
3) "Thank You For The Muesli" - Abba
4) "Manic Souffle" - The Bagels
5) "I Want To Bake It With You" - Bread
6) "Dark Side Of The Mousse" - Keith Floyd
7) "Souperfly Guy" - Eggs-Express
8) "I Just Diet In Your Arms Tonight" - The Cooking Crew
9) "Strawberry Flans Forever" - The Eatles
10) "Food To Fry" - The Rolling Scones

PAW Prints

by George E March, 93 Roberts Street, Newcastle-Upon-Tyne, NE15 6BE.

This month I thought I'd like to try and explain to the new (or even experienced) P.A.W writer, some of the more unusual and interesting abilities of its parser and vocabulary tables.
For instance, any verbs (or conversion nouns, ie, a noun that can be used as a verb if there's no verb given in a sentence) with a number less than 14 are taken as directions (eg, north and enter), any nouns with a number greater than 19, but less than 50 are taken as proper nouns (those names of people and place names, etc), and those with a number greater than 49 are names of objects, etc, but what about those with a number greater than 13, but less than 20? They're still conversion nouns, and so can still be taken as a verb, if no verb is available, but are not directions!
For example if we give a word (that we wish to be both a verb and a noun (eg, 'plant'),the noun number 15, and also make sure that there is no verb with the same number, give a second noun (eg, begonia),the number 52, and then give a verb we need to act as an interpreter (eg, bury) the number 39, it allows us to create a routine such as..

```
PLANT * LET 33 39

BURY PLANT PRESENT 4 SYSMESS 61 PLACE 4 255 SWAP 4 5 ANYKEY DESC
```

Using object 4 = 'A small pot plant', object 5 to be 'A plant buried in some soil' and sysmess 61 to be 'You plant the plant into some compost'. Which now allows us to PLANT the PLANT if it were present, into the ground where we stand, then the routine swaps the plant in a pot for a plant in some soil, and also by changing the noun number in an examine routine, we can both examine the plant and the begonia..

```
X PLANT LET 34 52
```

Which changes the number of the noun put in by the player, into another, to become.

```
X BEGONIA PRESENT 4 SYSMESS 60 MESSAGE 50 DONE

X BEGONIA PRESENT 5 SYSMESS 60 MESSAGE 51 DONE
```

With sys' 60 being 'It's a lively, green begonia, with beautiful pink flowers, ' (remember the space!), message 50 being 'but it's feeling a bit cramped in its too small pot!' and message 51 being 'sitting comfortably in some compost!'. So we can PLANT the PLANT, which, with some slight alteration can become BRIDGE the gap, and cross the BRIDGE, etc, and instead of two, seperate routines giving the same answer, eg..

```
X BRANCH AT 2 MESSAGE 27 NEWTEXT DONE

SEARCH BRANCH AT 2 MESSAGE 27 NEWTEXT DONE
```

Using location number 2 as a tree grove and message 27 to read 'The branches are far too high to reach..', with a newtext to break off any input after something that can't actually be done, we could use instead..

```
X BRANCH AT 2 LET 33 43

SEARCH BRANCH AT 2 MESSAGE 27 NEWTEXT DONE
```

Which thus converts the word examine (or X) into the word, search, to become the routine coming after it! Or we could even do away with the search and examine bits altogether!

* BRANCH AT 2 MESSAGE 27 NEWTEXT DONE

Which works with any verbs at all (for example, GET branch, PULL branch, etc), and now onto that combat sequence I promised you last month. Now there are a few ways of doing this, so I'll detail just two of them for space, though the 'calling' routine for both is nearly the same, with possibly a routine in response to start the ball rolling. For example our first type of combat routines could be started by..

ATTACK ? PRESENT 4 CLS LET 51 4 MESSAGE 63 PROCESS 6 PROCESS 7 DONE

Which, if the player wishes to attack an opponent (who is in the same location), the routine clears the screen of text/graphics, prints a relative message, using LET 51 4, which replaces any underlines in text, with the name of the object number specified, or alternatively in process 1.

* _ PRESENT 4 CHANCE 10 CLS LET 51 4 MESSAGE 63 PROCESS 6 PROCESS 7 DONE

Now you can see that I've used object number 4 as our opponent (the name is upto you), and I've used message 63 to read 'You attack the _', and process 6 as our combat result process. Process 7 prints out the appropriate messages resulting from any combat, and in this first combat example, I've also used a couple of random flags in process 1..

* * RANDOM 80 RANDOM 81

These need not be in the * * ZERO 31 option which I'll mention later, with flag 80 being the players strength, and flag 81 the opponents, we'll also need a third, blank flag as a 'counter' for an on/off effect, which I've taken as flag 82. Now for process 6 (our combat process) in this first example, we have the same flag measuring routine as I detailed in last months 'Coder'. Now we could also provide the player with a weapon to use in the game (I've given examples of two), with object 10 being 'A heavy broadsword' and obj' 11 being 'A boot knife', and then at the beginning of process 6 we could have..

* * 0 CARRIED 10 PLUS 80 15

* * 1 CARRIED 11 PLUS 80 7

Which then adds 15, or 7 to the flag 80 measurments below..

* * 2 CLEAR 82 SAME 80 81 NOTDONE

Which, if the players and his/her opponents strength levels are the same, won't do anything, and goes back to the calling routine in response, which moves the routine onto process 7 (the results messages), and then back onto process 1, and starts the random number sequence off again..

* * 3 LET 82 1 SUB 80 81 ZERO 81 NOTDONE

But, if they're not the same, the routine above first makes flag 82 (our counter) equal to one, then takes the contents of flag 80 away from flag 81 , and if 81 is nothing, then 80 is greater than 81, otherwise..

* * 4 LET 82 2 NOTDONE

82 becomes 2, and thus if flag 82 is nothing, then no one is hurt, if 82 equals 1 then 80 (and therefore the players strength) is greater than 81, and if 82 is 2, then 81 is more than 80! Okay so far? (No! - Ed.)

And so onto process 7, which is visited after the flags are measured, and firstly we have to decide what to do if either the player, or their opponent dies? For this we'll need another object to become the opponents dead body, which I've used as object number 5, again any names are upto you..

```
* * 0 ZERO 81 NEWLINE MESSAGE A SWAP 4 5 ANYKEY DESC

* * 1 ZERO 80 NEWLINE MESSAGE B NEWLINE TURNS NEWLINE SCORE NEWLINE END
```

And so if flag 81 (the opponents strength) is zero, then the opponent is dead, and the routine swaps our live opponent for a dead one, but if flag 80 (the players strength) is nothing, then it prints a message saying so, turns taken, the score, etc, and ends the game for a restart. Now we also need a way of telling the player exactly what's going on, and so in process 7 we'll also need three little routines to measure just that..

```
* _ 0 ZERO 82 MESSAGE C NEWLINE PROCESS 6 PROCESS 7
```

So if 82 is nothing, then no hits are scored, so message C = 'You both miss!', it prints a space between this message and the next (makes it look cleaner) and returns to the beginning of process 6, and does it all over again, until one of them is dead..

```
* _ 1 EQ 82 1 MINUS 81 10 MESSAGE D NEWLINE PROCESS 6 PROCESS 7
```

For the routines given above, if flag 82 is 1, then 80 is more than 81, with message D being 'The opponent is wounded', though I've left out the LET 51 4 this time..

```
* _ 2 EQ 82 2 MINUS 80 10 MESSAGE E NEWLINE PROCESS 6 PROCESS 7
```

Otherwise, if 82 is 2, then the player is hurt, his/her strength is reduced and message E printed, and then back to the beginning again..

Now for this second combat example (which I've designed for a 128 gamer) in response we'll need a 'calling' routine nearlly the same as our first version, but without process 7 being called..

```
ATTACK ? PRESENT 4 CLS LET 51 4 MESSAGE 63 NEWLINE PROCESS 6 DONE
```

Now in process 1 we'll need to set both the players and his/her opponents strength levels to their initial values, using..

```
* * ZERO 31 LET 80 40 LET 81 40
```

Now, I've just set both strength levels to a limit of forty, which could always be added to with food, potions of strength, etc, or reduced by poisons, but they could be set to any value at all below 100, but not above 100, as you can only 'nest' a process upto a limit of ten, ie, you can only recall a process 10 times, before you get an error message. Now for process 6 in our second example, we could use..

```
* * 0 RANDOM 82

* * 1 CARRIED 10 PLUS 82 15
```

* * 2 CARRIED 11 PLUS 82 7

* * 3 ZERO 81 NEWLINE MESSAGE A SWAP 4 5 LET 80 40 ANYKEY DESC

Now you'll notice that the only real alteration I've made to this last routine, is to add a LET 80 40, to put the players strength back upto its initial value again, as the * * ZERO 31 action in process 1 won't be used again, and you've got to set the limits again yourself, or have the player have to find some way of renewing it him/herself, by food for example..

* * 4 ZERO 80 NEWLINE MESSAGE B NEWLINE TURNS NEWLINE SCORE NEWLINE END

You'll notice that most of the routines I've used so far (especially the last one above) are exactly the same as those in the first example..

* _ 0 GT 82 40 PROCESS 7 PROCESS 6

Now this time around, I've used process 7 as that which deals with any combat if the opponent is hurt..

* _ 1 LT 82 41 PROCESS 8 PROCESS 6

Just as the one above, but only if the player is hurt..

* _ 2 SAME 80 81 PROCESS 9 PROCESS 6

Same again, but only if neither the player or opponent is hurt, and now for these next three process tables, all three will need three routines within each, all exactly the same, but only the first two processes will use a minus action, and only the first two routines within each process will need a chance action, and thus for process 7 we'll need..

* _ 1 and 2 CHANCE 35 MINUS 81 A MESSAGE ?

* _ 3 won't need a chance action, with it being the last routine in the process, and all of these three are for the process when the attacker is hurt, so..

* _ 3 MINUS 81 A MESSAGE ?

Now for the routine above, and the following one, I've used flag values A and B to become a number from 1 to 20 (any higher and the odds are probably too much in the players favour, and he/she will probably win every time), the exact value I'll leave upto the writer, just like any names of characters and messages, and now onto process 8 (dealing with the resulting messages if the player is hurt)..

* _ 1 and 2 CHANCE 35 MINUS 80 B MESSAGE ?

* _ 3 MINUS 80 B MESSAGE ?

Now process 9 (our last fight process thankfully!) is the one dealing with any messages printed, if there's no harm caused to either the player or his/her opponent, needing no minus actions at all..

* _ 1 and 2 CHANCE 35 MESSAGE ?

* _ 3 MESSAGE ?

Well, that's it for another month!

Transport in Adventures

by Shaun "Spud" Allaton

A lot of games nowadays seem to implement at least one mode of transport, be it a Cab, Car, Bus or in a few cases, such as in "Sherlock", Trains. Over the coming months I'll be looking at each section and showing how it can be done using PAW.

We'll start though, with the simplest of the four - the car. First of all we'll need to set aside three locations. Locations one and two will be, let's say, an office building and the player's home, and location three will be inside of the car. I've shown the example without making it so the player has to start the car although this could easily be implemented by creating an object called a Car Key and checking that when a DRIVE TO command is entered that the player is carrying the key.

Flags

60 = The location where the car is

Messages

No.	Description
1	I was already in the car.
2	The car isn't here.
3	Climb into what?
4	Climb out of what?
5	I drove
6	home.
7	to the office.
8	I was already here.
9	Drive where?
10	A car stands before me.

Response table/Process table 0

```
CLIMB CAR    PREP IN
             SAME 38 60
             GOTO 3
             DESC

CLIMB CAR    PREP IN
             AT 3
             MESSAGE 1
             DONE

CLIMB CAR    PREP IN
             MESSAGE 2
             DONE

CLIMB _      PREP IN
             MESSAGE 3
             DONE

CLIMB OUT    AT 3
             COPYOFF 60 38
             DESC

CLIMB OUT    NOTAT 3
             MESSAGE 4
             DONE
```

```
DRIVE *        AT 3
               PROCESS 3
               DONE
```

Process table 1

```
*     *        SAME 38 60
               MESSAGE 10
```

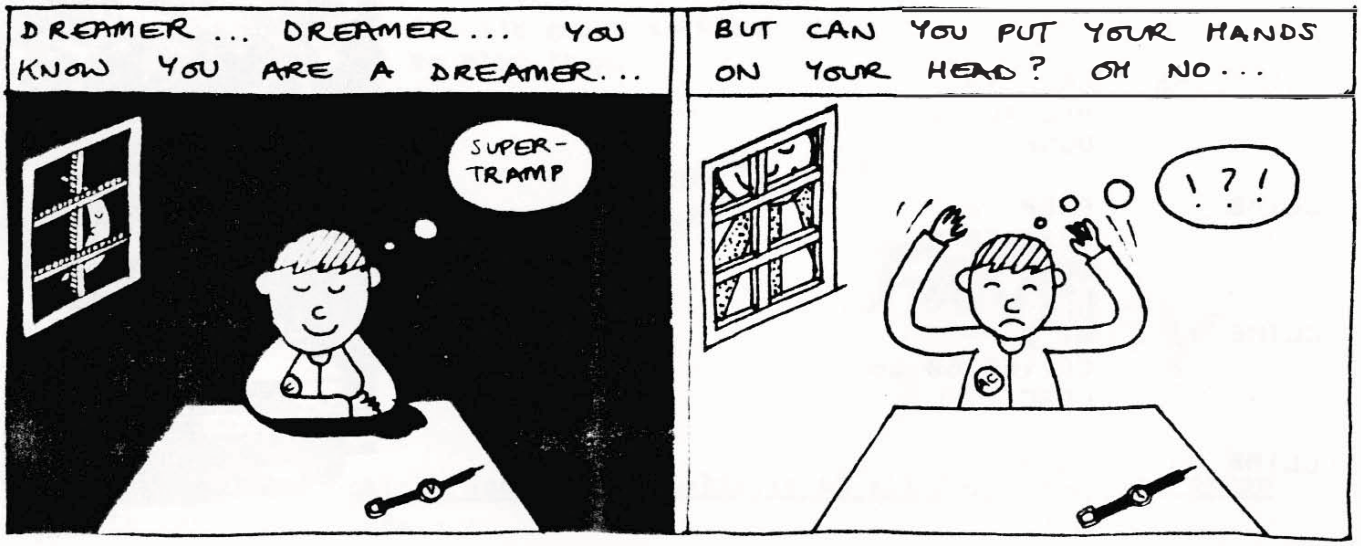
Process table 3

```
DRIVE HOME     NOTEQ 60 1
               LET 60 1
               PAUSE 100
               MES 5
               MES 6
               DONE

DRIVE HOME     MESSAGE 8
               DONE

DRIVE OFFIC    NOTEQ 60 2
               LET 60 2
               PAUSE 100
               MES 5
               MES 7
               DONE

DRIVE OFFIC    MESSAGE 8
               DONE

DRIVE _        MESSAGE 9
               DONE
```

That's all folks! Quite short and also simple. Next time I'll be looking at implementing cabs which, in my view is just as simple as the above (notice I'm doing the simple ones first?!). By the way, if you can't wait until the next time or you have a few problems programming using PAW then drop me a line at 61 Goldcrest Road, Ipswich, Suffolk, IP2 OSF and please enclose a SAE.

The Dreamer

by Chris "letter of the month in TGM021" Hester



PAW Prints

by George March

This issue, I'd like to answer one of my own frequently asked questions, which is 'how can I make my ideas look more professional'? Or to put it another way, 'how can I write routines that people like 'Level 9' have in their games'? So a while ago (about 2 years), I came up with a couple of routines that gave me just what 'Level 9's 'Gnome Ranger' has, an 'Again', an 'Oops', and to go with these two, a 'Find object' routine.. These 'Again' routines are more or less just a universal form of LET 33 ?, the first part of which goes at the very beginning of process 2, which is measured 'after' any commands are entered by the player..

```
* * 0 NOTEQ 33 A COPYFF 33 133 COPYFF 34 134 COPYFF 35 135 COPYFF 36 136
COPYFF 43 143 COPYFF 44 144 COPYFF 45 145 NOTDONE
```

I've used value A, in the NOTEQ action above, to be the verb number of the word 'Again', or 'Repeat', etc, so you'll be able to see that all it does is, if any command that's entered into the game is NOT equal to the word 'Again', then this routine makes a copy of that sentences verb, adverb, first noun, etc, into the seven 'blank' flags given (133-145), and then ends the routine with a NOTDONE, thus telling PAW that the flag numbers have had copies made of them, but that the game isn't affected any, because it thinks that nothing has been done!
Now this section (the business end!) must be placed at the very beginning of response, just in the same way that any LET 33 ? actions have to be placed 'before' the routines that they're transformed into!

```
* * 0 EQ 33 A COPYFF 133 33 COPYFF 134 34 COPYFF 135 35 COPYFF 136 36
COPYFF 143 43 COPYFF 144 44 COPYFF 145 45
```

And so what the routine does is, if the word 'Again' is typed, it copies the copies of the previous sentence's verb, nouns, adjectives, etc, back into the response table, and does what it's already done a second time round! Okay, but what if the word 'Again' is entered more than once? Very easy, as the * * 0 action in process 2 only works if the word 'Again' is NOT typed, so if it is, the * * 0 action in response just copies and re-copies back into itself what's already been done, for as many times as the word 'Again' is entered!

Now for my 'Find object' routine, this next routine first finds the number of the object given in the sentence (ie, the number in flag 51, which is the object number measured by the WHATO action, in place of any underlines in routines, messages, etc) and if the number of the object is less than that of the number after that of the last object in a game (given as value B), and the location of the object is NOT not created, and is therefore actually in the game, isn't worn, is not carried, and isn't at the players present location, then the room number of the player is made to be the same as the room number of the object in question, ie, the player moves to the objects location, with sys' 68 telling the player what happens. You don't need a newline with an ANYKEY/DESC, and it takes up a tiny bit of useful memory anyway, so I haven't bothered!

```
FIND _ 0 WHATO LT 51 B NOTEQ 54 252 NOTEQ 54 253 NOTEQ 54 254 NOTEQ 54
255 COPYFF 54 38 SYSMESS 68 ANYKEY DESC
```

Sysmess 68 = 'You wander round, until you find the _ !' This next routine's nearly the same as that above, just that the object hasn't been found yet, and is thus not created..

```
FIND _ 1 WHATO LT 51 B EQ 54 252 MESSAGE 69 NEWTEXT DONE
```

Message 69 = 'Sorry, I can't find that, it must be hidden somewhere!' and stops any input with a newtext. This next bit, is just as above, but if the object is worn, if so, then sys' 29 = 'You're already wearing the _!' is printed, newlined and newtexted.

```
FIND _ 2 WHATO LT 51 B EQ 54 253 SYSMESS 29 NEWLINE NEWTEXT DONE
```

And again, though if the object is carried this time..

```
FIND _ 3 WHATO LT 51 B EQ 54 254 SYSMESS 25 NEWLINE NEWTEXT DONE
```

Also for this next one, but if the object is in the players location, with sys' 59 = 'But the _ is already here!'

```
FIND _ 4 WHATO LT 51 B EQ 54 255 SYSMESS 59 NEWLINE NEWTEXT DONE
```

But what if the player tries to 'find' any object which doesn't exist in the game, ie, anything which isn't recognised by these first four routines!

```
FIND _ 5 SYSMESS 8 NEWLINE NEWTEXT DONE
```

Well this should take care of that, as this routine should only be encountered if the object in question is NOT a real item, and would thus pass through the other four routines, if so, then sys' 8 is printed and newlined, etc, and to finish off these 'FIND _' routines, how about a little rearrangement of the verb, by making value C below, equal to the number of the verb 'FIND' in vocabulary, you'll also need to make 'FIND' have a number higher than that of the word 'SEARCH' in the vocab' table..

```
SEARCH _ 0 PREP FOR LET 33 C
```

And now onto a few chages to issue one's EXIT _ routines, for a better, cleaner, 128 version (if you've plenty of memory left that is!) Its 'calling' routine from response is more, or less just the same as that found in issue one..

```
EXITS _ SYSMESS 62 PROCESS 4 NEWLINE DONE
```

The first thing we'll need to do is to define four new objects as the names of the main compass directions (though there's absolutely no need for weights of objects and their words to be used!), for example, sys' 62 = 'Obvious exit(s) ' (note tne space after 'exit(s)') to make it look a bit better, with object 4 = 'north', obj' 5 = 'south', obj' 6 = 'east' and obj' 7 = 'west', now if these four (or eight and ten if you include diagonal directions and/or up and down) directions are all taken as not-created (exept for those for the very first room visited, which are placed in the initially at table, as room 0, ie, the introduction screen, which the player cannot visit, and can only 'read'), then with a few calculations of the connections table for each room visitable, and a..

```
* * ZERO 31 LET 53 64
```

In process 1, just to get the look of things right, then in process 4 (called by the EXITS _ routine in response) we could have for example..

```
* _ AT 4 PLACE 4 0 PLACE 6 0 PLACE 7 0
```

One routine of this type would be needed for each room available, and then to end the whole process..

_ _ LISTAT 0 DESTROY 4 DESTROY 5 DESTROY 6 DESTROY 7, etc..

For as many objects as the writer decides to use as direction names! And so, if the EXITS _ were called whilst the player was in room 4, then the above routines in process 4 would give..

'Obvious exit(s) north, east and west.'

As the LET 53 64 action always places commas and an 'and' in any screen listings of objects if used!
And next, how about what I believe to be the ultimate 'Oops' routines going, as the 'Oops' routine given in the PAWS technical manual (and possibly the one given in 'Adventure Contact' number 13, by Mr. Bryan Kitts, though I've never actually read it!) makes the use of ramsave/ ramload impossible, as the one in the manual uses an auto' ramsave in process 2, as an 'Oops', and stops the use of seperate ramsave/ramload routines, whereas the ones given here allow the use of both! So for the very beginning of process 1 (or 2, depending on the writers tastes!)

* * 0 ZERO 66 RAMSAVE

Which, just like that given in the manual, causes an automatic ramsave 'before' anything can be entered by the player, but only if flag 66 is zeroed! These next six routines are all for response.

RAMSA _ 0 ZERO 66 SET 66 RAMSAVE DESC

So if ramsave is typed, and flag 66 is nothing, ie, in 'Oops mode', the routine sets 66, ramsaves and describes the screen, but if 66 has already been set by the routine above, then it can't work a second time can it? And thus, if ramsave is typed whilst in 'ramsave mode'..

RAMSA _ 1 NOTZERO 66 RAMSAVE DESC

This first ramload routine only works if 66 is set, ie, ramsave has been used, it then ramloads the game, and clears 66, putting the game back into 'Oops mode'!

RAMLO _ 0 NOTZERO 66 RAMLOAD 255 CLEAR 66 DESC

Yes, but what if the player tries to ramload whilst in 'Oops mode'? Well this routine should then print message A = 'But you can't ramload if you haven't ramsaved a position first!' and newtexts the game!

RAMLO _ 1 ZERO 66 MESSAGE A NEWTEXT DONE

And for our Oops action whilst in 'Oops mode', which works automatically at the start of the game, and ramloads from the auto' ramsaved position in process 1 (or 2).

OOPS _ 0 ZERO 66 RAMLOAD 255 DESC

And this last one of our six response routines, takes care of the problem, if a player tries to Oops whilst in 'ramsave/ramload mode', with message B = 'You'll have to ramload your already ramsaved position first!'

OOPS _ 1 NOTZERO 66 MESSAGE B NEWTEXT DONE

Now to end with, here's a cute, little location finder for any objects that aren't in the same position as the player, so for response we could have..

* _ NOTEQ 33 A WHATO LT 51 B NOTEQ 54 252 NOTEQ 54 253 NOTEQ 254 NOTEQ 54 255 SYSMESS C PROCESS 5 NEWTEXT DONE

With 'A' being the verb number of the word 'remove', so that this routine won't interfere with the removal (or getting) of any items from a container, 'B' is the number after that of the last objects number, and sys' C = 'The _ is still in the ', please note the space after 'the', and process 5 is concerned with the printing of the name of the objects container on screen, so for process 5..

* _ EQ 54 D LET 51 G

* _ EQ 54 E LET 51 F

With 'D' being the location number where the object is located, ie, a container-room number, and 'G' being the object number of the container, but if you 128'ers with all that memory, wish to make your games a bit more friendly, how about, as in my second example, if we use 'E' as a normal (not container) room number, and 'F' as the name of the room, you would then have to use some objects as the names of these rooms however, just as the EXITS _ examples above (with no need for weights, or words of objects to be used!), such as 'D' = room 5, 'G' = object number 5 'A cardboard box', 'E' = room 6 'the kitchen', though you could also do it like this..

* _ EQ 54 E MESSAGE F

So, again with 'E' being a non-container, normal room number, and message 'F' being the name of the room printed, instead of an object used, and to end this process, we'll need a little routine like the one below, with 'H' = '_!', for the twin * _ EQ 54 D/E routines already given, which prints the name of the object-container/ location where the object in question is contained!

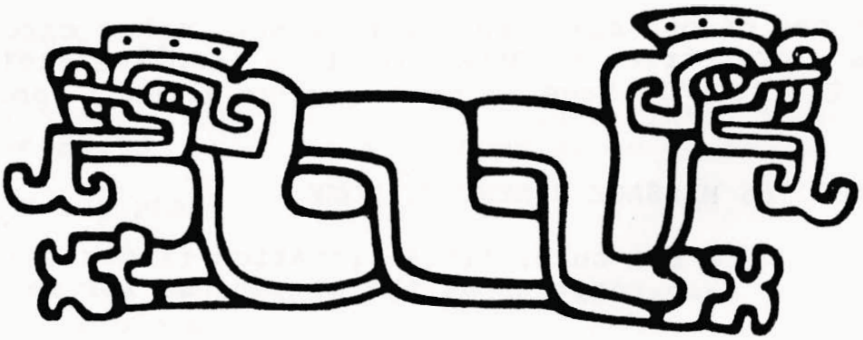
_ _ MESSAGE H DONE

And as always, you can either contact me here via the 'Coder', or at..

93 ROBERTS STREET, NEWCASTLE UPON TYNE, NE15 6BE

For any comments, complaints, help that you need, or hopefully what we're looking for is help you can give to your fellow writers! I mean holding onto your own routines may help you make your own games look good, but helping others with some ideas you've created, not only helps them get their games looking great, but should make you feel very proud in what you've done to help, and it also gets your name and/or if you actually publish your games under your own label, your company's name in print, with full credit! So go on, give a little!

So good bye 'till next month!



Investigating PAW

by Gerald T Kellett B.sc

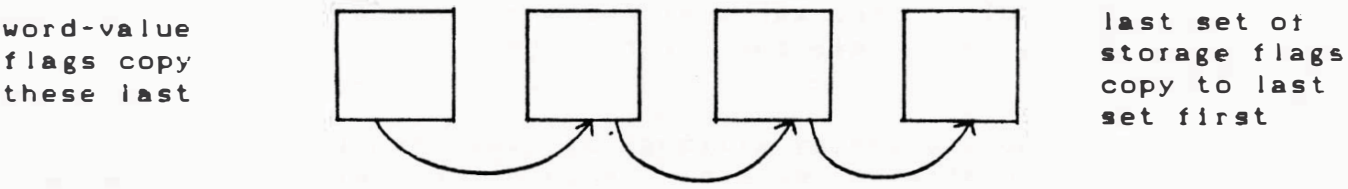
The significant changes to PAW have been the additions of; the user overlays, the multiple PARSE (both these were my ideas incidentally), the user data-Hunk standard and overlay (which though not my idea arose out of the need to have user definable Direction Pointer Tables for TEL's MAP function) and the user transparent storage media handling. The multiple PARSE has been available but undocumented from version A10, it became a documented feature when the other three features were added at version A14.

The user overlays provide for additional functions to be added to the editor, there are four at present, PAW-TEL, PAW-PHOSIS and MEGA in the PTM user overlay add-on, and overlay H the data-Hunk management overlay which is supplied with PAW. Overlay Z is reserved for user's user overlays, the letters K, U and W have also been 'booked'.

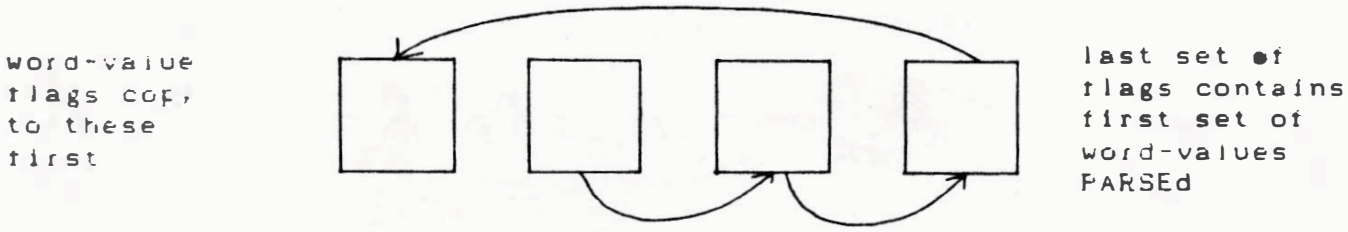
Prior to the modification to allow multiple use of PARSE, only the first logical sentence could be extracted, this meant you could not give more than one command at a time to other characters. Now it is possible to extract many sentences from text within double-quotes. However, unlike player input these must all be extracted in one time-frame, which requires that each sentence in turn is extracted and stored before extracting the next one or returning to deal with direct player commands. There is also a minor difficulty of acting upon multiple commands to other characters synchronously with player actions - it's no good if the commanded character performs all the actions he is instructed to do within one time-frame!

To store multiple PARSEd sentences requires the word-values in flags 33-36 and 43-47 (the word-flags) to be moved to other flags with COPYFF before performing another PARSE. To make this work usefully for more than two sentences (ie with one stored and one still in the word-flags) requires the use of a shift-register setup, with the stored values in flags being moved-on to yet other flags before the word-flags are stored in them.

To act on these synchronously requires the first sentence stored to be moved back into the word-flags, and for each following sentence likewise in each subsequent time-frame until all are dealt with. The shift-register storage is used in what is termed a ring-register mode.



Repeat after each PARSE and then do the following during a time-frame in which multiple PARSE is performed on for subsequent time-frames until all stored sentences are dealt with.



The user data-Hunk facility has many uses, it's primary function is to allow data for user overlays to be stored in a controlled and consistent manner within a database. It can also be used to include such things as a screen-dump routine for the Spectrum +3, or dot-matrix printer screen-dumps for 48K users. These however must be produced in position independent code as a Hunk may move, unless that is, it is installed each time as the first Hunk and no character sets are added.

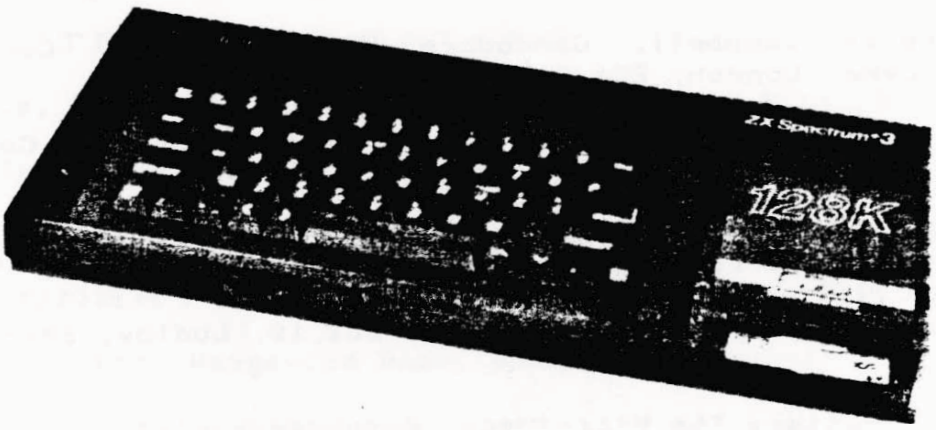
You can also incorporate extra data to be used by EXTERNS, one idea I had was for an AUTOEXAmine function, with a table containing a number for each object which would be the number of the message containing additional info on each object or that of a "nothing special" message.

The user transparent storage media (tape/disk/microdrive cartridge) handling means that anyone producing user overlays does not (except in the case of verify) concern themselves as to what media is being used for loading and saving as they are all dealt with in a consistent manner.

There are handlers for tape and +3 disk, I'm fairly certain there is one for +D/Disciple users, the one for OPUS disk was dropped when this disk-drive was discontinued. There should have been one for Microdrive, but as I was promised a copy of this version when it became available and no such thing has been forthcoming I can only conclude it was never released.

The only other PAW currently in existence is a text-only CP/M one, for CPC, PCW 8256/8512 and Spectrum +3 of course, and any other 3-inch disk-using CP/M machines (the only one that springs to mind is the Tatung Einstein). The originally advertised CPC version was as far as I know never started, and the Commodore 64 version although started has, it seems, been abandoned in favour of 'PAW Espania' (the Spanish language version of PAW) and PCPAW (perhaps.)

As PAW-PHOSIS allows PAW users to build up a library of useful sub-processes, would there be any interest in having a central repository (or swap-shop) for the mutual collection and exchange of these? (Strictly user's own productions and no 'borrowings' from commercial games). Basically these would be compiled onto a tape and all contributers would obtain a copy for the cost of materials, postage and packing, and handling. Any takers?



PAW PRINTS

by George March

I'm beginning this issue with a posthumous dedication to the 'father' of all barbarian adventurers, Robert Erwin Howard (1906 - 1936), who gave life to that golden hero, Conan the barbarian, and to the people who've kept him going these many years hence, L. Sprague De Camp, Lin Carter and Bjorn Nyberg, here's to you all..
Well, after that, how did last month go for you? Mine went pretty awful, as per usual, and I've also been told off slightly by Larry Horsfield (sometime in August) for 'PAW Print's being too straight-laced, too formal and not explaining my routines easily enough (not everyone has the same level of knowledge of PAW, do they?), maybe I'm all facts no fun? (My own words, and I know exactly what he means!) So I'm going to have to start re-thinking my style, but as Larry said in 'Probe' volume 3, number 7, neither 'PAWs for thought', or 'Print's will survive without any help from their readers, so come on you lot! Give a little.. Or a lot if you can spare the time to put pen to paper, lazy lumps! But enough of the sarcastic remarks.. Oh, and by the way, if you're on one of these YTS courses, you have my sincere sympathy! But enough of the moans, and just before I start I'll give a little mention to Mr Darren Rose of Norfolk, for his 'PAW Print's is excellent!' comment in 'Snippets' from 'Coder' number 2, thanks Darren!

So do you remember Larry's column from 'Probe' volume 3, number 4? In which he explained the excellent use of flags to hold the number of objects carried/dropped, worn/removed, or inside of another object, and his new 'inventory' routine, using LET 53 64, for the continuous listing of objects on-screen?
Well here I'd like to offer some slightly different versions of those 'inv' routines, and some totally new and definitely different get all/drop all, etc, routines, but first of all I'd like to re-cap (for those who've never read it, like me, even though I'm writing about it!) some of Mrs Linda Wright (of Marlin Games) routines from 'Adventure Contact' 17.
My many thanks go to both Linda and Larry for this month's 'PAW Print's!

So for Linda's 'Adventure Contact' routines, using the PAW's ability to recognise verbs and nouns above/below a certain value, because any nouns with a number less than 15 are movement words (ie, the names of any directions available), those with a number less than 50, but higher than 19 are the names of people and places, etc, if we use some nouns with a value less than 100, but greater than 49, as object names, and anything else that's described in a message, or location text we give a noun number more than 99, we can come up with some really useful and 'friendly' routines just like these..
So here I've used flag 'A' as a number AFTER that of the last, real, objects number, present in a game, so..

```
GET _ 0 WHATO LT 51 A GT 34 49 LT 34 100 AUTOG NEWLINE DONE
```

Is the noun number (flag 34) given, the name of a real object? (Using WHATO LT 51 A) Then you're allowed to pick it up, I've newlined it just to make it look a bit 'cleaner'!

```
GET _ 1 WHATO LT 51 A GT 34 19 LT 34 50 MESSAGE F NEWTEXT DONE
```

But if the object-noun is the name of a person for example, then message 'F' = 'But _ doesn't wish to follow you unless invited!', which could always be used with something like 'SAY TO BALROG "FOLLOW ME"'.

GET _ 2 GT 34 99 MESSAGE G NEWTEXT DONE

And if the noun is an item relevant to a location description, for example a wardrobe in the corner of a room, and so cannot be moved, then message 'G' = 'Sorry, but it will not move, perhaps it is too heavy to lift?'

GET _ 3 SYSMESS 26 NEWLINE NEWTEXT DONE

With sys' 26 = 'Sorry, but there's nothing like that here!'. Now a variation of some of these GET _ routines could also be used for some DROP _ actions, for example..

DROP _ 0 WHATO LT 51 A GT 34 49 LT 34 100 AUTOD NEWLINE DONE

This first one's just about the same as the GET _ 0 option above..

DROP _ 1 WHATO LT 51 A GT 34 19 LT 34 50 MESSAGE H NEWTEXT DONE DONE

With message 'H' = 'But you don't possess anything like a _ to be able to drop!'.

DROP _ 2 GT 34 99 SYSMESS 28 NEWLINE NEWTEXT DONE

Again, this is nearlly the same as GET _ 2, but with sys' = 'But you don't possess anything like that!'.

DROP _ 3 SYSMESS 28 NEWLINE NEWTEXT DONE

I'll freely admit, my DROP _ routines were entirely based on Linda's GET _ versions, but they're a compliment to them I hope? Now for those variations on the get all/drop all, etc, routines..
Now the first of these get all commands only works if 'none' of the conveyable objects in a game are in the players present location!

GET ALL 0 ISNOTAT A 255 ISNOTAT B 255 ISNOTAT C 255 MESSAGE D NEWTEXT DONE

And thus, if neither objects A, B or C are at the players current location (flag 255), then message 'D' = 'But there isn't anything here to be able to pick up!' There should be one ISNOTAT action for each conveyable object in a game, using examples A, B and C..

GET ALL 1 MESSAGE E PROCESS A ANYKEY DESC

The routine in response will only go onto the GET ALL 1 action, if there is at least 1 conveyable item in the players present location, and it then moves onto process A and redescribes the screen, which thus 'gets' any objects that are present, with message 'E' = 'You pick up anything you can see!', and for the 'business end' of these routines in process A, we need to keep a constant record of the amount of items being carried and worn, with flag 71 being the amount of objects carried, and for some routines I've got following later, flag 11 the number of things that're worn.. So for process A!

* _ 0 ISAT A 255 PLACE A 254 PLUS 71 1

And from the example above, 1 such routine needs to be used for each individual object conveyable in a game, just changing value A into the number of the object concerned.

GET _ 0 AUTOG PLUS 71 1 DONE

Now for any objects being carried (ie, at location 254, with flag 71 being anything but zero) these next few routines are nearly the same as the GET ALLs above.

DROP ALL 0 ZERO 71 MESSAGE F NEWTEXT DONE

Here, DROP ALL 0 'only' works if the player is NOT carrying anything at all (and flag 71 is thus equal to zero), with message 'F' = 'But you're not carrying anything to be able to drop!'

DROP ALL 1 NOTZERO 71 MESSAGE G PROCESS B ANYKEY DESC

So if flag 71 is anything but zero (ie, the player's carrying at least 1 item), then message 'G' = 'You put down anything you're carrying!' is printed, the routine moves onto process B and the screen is cleared with the next keypress. And now onto process B..

* _ 0 ISAT A 254 PLACE A 255 MINUS 71 1

Again there's one such sequence for each individual object! Which takes any items in the carried location and puts them in the players current location.

DROP _ 0 AUTOD MINUS 71 1 DONE

WEAR ALL 0 ISNOTAT P 254 ISNOTAT Q 254 ISNOTAT R 254 MESSAGE H NEWTEXT DONE

Just like the GET ALL 0 action above, this WEAR ALL 0 routine checks to see whether any wearable items are being carried, if not then message 'H' = 'Sorry, but you don't possess anything that's wearable!'.

WEAR ALL 1 NOTZERO 71 MESSAGE J PROCESS C ANYKEY DESC

That last one only works if the player is NOT carrying any wearable items at all (ie, it only comes to this WEAR ALL 1 action, if it bypasses WEAR ALL 0, and the player is carrying at least 1 object, and a wearable item is NOT not-carried, and is therefore at location 254, being carried (logical isn't it? And don't you dare say "No!", Chris!), with message 'J' = 'You put on any wearable objects you're carrying!'.(Eh? - Chris)

WEAR ALL 2 ZERO 71 MESSAGE H NEWTEXT DONE

WEAR ALL 2 only works if the player isn't carrying anything at all, with message H being just the same as for WEAR ALL 0, and now for process C..

* _ 0 ISAT P 254 PLACE P 253 MINUS 71 1 PLUS 11 1

So if the player has a wearable object that's being carried, its position is changed to the object-worn location (room 253), a value of 1 is deducted from the flag holding the amount of items being carried, and added to the amount of objects being worn flag!

WEAR _ 0 AUTOW MINUS 71 1 PLUS 11 1 DONE

And for a REMOVE ? of the same nature..

REMOVE ALL 0 NOTZERO 11 MESSAGE K PROCESS D ANYKEY DESC

Is the player wearing at least 1 object (ie, flag 11 is notzeroed? If so, then message 'K' = 'You take off any items being worn!' is printed, and the sequence then moves onto process D..

REMOVE ALL 1 ZERO 11 MESSAGE L NEWTEXT DONE

With message 'L' = 'But you're not wearing anything to be able to take off!', and now for process D..

* _ 0 ISAT P 253 PLACE P 254 MINUS 11 1 PLUS 71 1

The last of these get, drop, wear, etc, routines is..

REMOVE _ 0 AUTOR MINUS 11 PLUS 71 1 DONE

And for an all purpose 'put object into/remove it from a container' action of the same type, I've used these next few routines to allow a 'wicker basket' to be able to hold upto 7 items and no more (just to make it look a bit more realistic), so if we use flag 15 as the amount of objects being held inside the basket, and object 3 as the basket itself, we can have..

FILL BASKET PRESENT 3 LET 33 L LET 34 M LET 44 N

I've used verb L as the number of the verb PUT, M as the first noun number of the word ALL, and N as the number of the second noun BASKET, which then allows us to have..

PUT ALL 0 NOUN2 BASKET PRESENT 3 NOTZERO 71 LET 51 3 MESSAGE O PROCESS E ANYKEY DESC

And so, if the basket is in the same location as the player, and he/she is holding at least 1 object, then message 'O' = 'You put anything you are carrying into the _!', with flag 51 converting the number of the object given, into the text description of said item, and using that instead of any underlines in text!

PUT ALL 1 NOUN2 BASKET PRESENT 3 NOTZERO 71 GT 15 7 LET 51 3 MESSAGE P NEWTEXT DONE

With message 'P' = 'Sorry, but the _ is far too full to be able to hold anything else!', and for process E..

* _ 0 ISAT P 254 LT 15 8 PLACE P 3 MINUS 71 1 PLUS 15 1

So for process E, is any object being carried, and is the amount of objects inside the basket less than maximum, ie, less than 8? If so, location of object number 'P' is changed to be that of 'inside the basket', and the relevant flags are altered!

PUT _ 0 NOUN2 BASKET PRESENT 3 LT 15 8 WHATO LT 51 A EQ 54 254 AUTOP 3 MINUS 71 1 PLUS 15 1 DONE

Likewise for PUT _ 0 above, if the object mentioned (using WHATO LT 51 A) is less than a number after that of the last real objects number, and is therefore a 'real' object in the game, the relevant objects location is changed to the inside of the container-object, but only if the amount of items inside the container is less than the maximum amount specified (ie, less than 8), and the flags are changed.

PUT _ 1 NOUN2 BASKET PRESENT 3 GT 15 7 MESSAGE P NEWTEXT DONE

This one above, just like PUT ALL 1, checks to see whether the maximum amount of objects is being held within the basket, and if so, prints message P (see PUT ALL 1 for message P), and now, how about this for allowing the player to remove all of the objects held in the basket, in

one go!

EMPTY BASKET PRESENT 3 LET 33 O LET 34 M LET 44 N

Here I've used value O as the number of the verb REMOVE, see FILL BASKET for the values of M and N, which thus gives us..

REMOVE ALL 0 NOUN2 BASKET PRESENT 3 NOTZERO 15 LET 51 3 MESSAGE Q PROCESS F ANYKEY DESC

If flag 15 is notzero, then there is at least 1 object held inside the basket, if so, then message 'Q' = 'You remove any objects from inside the _ !', and so onto process F..

REMOVE ALL 1 NOUN2 BASKET PRESENT 3 ZERO 15 LET 51 3 MESSAGE R NEWTEXT DONE

If flag 15 is zero, then there are no objects at all within the basket, so message 'R' = 'But there's nothing in the _ for you to remove!', and the following for process F..

* _ 0 ISAT P 3 PLACE P 254 PLUS 71 1 MINUS 15 1

Is object P at location 3, and thus 'inside the basket'? If so, then it's placed in the player's 'carried' location, the amount of things carried is added to, and the amount inside the basket reduced by 1..

REMOVE _ 0 NOUN2 BASKET PRESENT 3 NOTZERO 15 WHATO LT 51 A EQ 54 3 AUTOT 3 MINUS 15 1 PLUS 71 1 DONE

REMOVE _ 1 NOUN2 BASKET PRESENT 3 ZERO 15 LET 51 3 MESSAGE X NEWTEXT DONE

Message 'X' = 'But there's nothing inside the _ for you to remove!'. Those last two routines don't really need any explanation I think, other than to repeat that flag 15 is the amount of objects inside the basket, and I should also point out that (like the example for object P) 1 such routine should exist for each conveyable object (though items that would be far too heavy for the player to lift in any case, such as a wardrobe for instance, need not be included in this conveyable whatnot process), or in the cases of processes C and D, each wearable/removable object in a game! Also, if the player starts the game off by wearing, or carrying anything, apart from obviously needing to include any such objects in the 'initially-at' table, you'd also need to include a routine like this next one, in process 1..

* * ZERO 31 LET 71 2 LET 11 1

So, for the above, if this is the player's first keypress (ZERO 31), and he/she is carrying 2 objects and wearing a third, then make the 'carried' flag equal to 2 and the 'worn' flag equal to 1, or if there are 2 items held in the basket at the start of the game, then the * * action above might also include..

* * ZERO 31 LET 15 2

And now onto Larry Horsfield's slightly changed 'inventory' commands, for which we'll need to re-write a few of the system messages already held in PAW, so for response, let's change the I _ routine already there into..

I _ 0 NOTZERO 71 NOTZERO 11 SYSMESS 11 SYSMESS 9 LISTAT 254 SYSMESS 47

SYSMESS 10 LISTAT 253 SYSMESS 51 DONE

I _ above only works if the player is both carrying, AND wearing at least 1 object, if so, then the new sysmessage '11' = 'You're ', sysmess '9' = 'carrying ' (don't forget the spaces!), sys '47' = ' and ', and sysmess '10' = 'wearing ', with locations 254 and 253 holding the objects being carried and worn respectively, with sys' 51 being a full stop with a newline after it!

I _ 1 ZERO 11 NOTZERO 71 SYSMESS 11 SYSMESS 9 LISTAT 254 SYSMESS 51 DONE

The only difference for I _ 1 is if the player is holding something, but wearing nothing!

I _ 2 ZERO 71 NOTZERO 11 SYSMESS 11 SYSMESS 10 LISTAT 253 SYSMESS 51 DONE

So that I _ 1 and 2 only bother to list any objects that ARE being worn, or carried, and will not bother to say something such as 'You're wearing nothing', or 'You're carrying nothing!' for example. But what if the player isn't carrying, or wearing anything at all? (Oooer — Ed)

I _ 3 ZERO 71 ZERO 11 MESSAGE S NEWTEXT DONE

The message 'S' = 'Sorry, but you don't have anything in your possesion at all!'

Well, that's it for another month, ta-ta for now, and as usual for any help you want and/or help you can give to your fellow adventu-writers, please contact me here via the 'Coder', along with any other comments, or my address is.. 93 ROBERTS STREET, NEWCASTLE UPON TYNE, NE15 6BE

PAW MARKS

George March answers your PAW problems - this month. how to prevent objects being dropped in a maze.

My god! I've actually had a lovely letter off someone wanting some help - I must be doing something right? At last..

So to begin with, my many thanks go out to Miss Alison York, of somewhere in Bradford (you didn't say where though, Alison?) for my first, and only so far, letter to 'Print's, and at least I think you're a Miss, as you didn't say that either (pity!), you could be a Mrs, or a Ms even if you're the liberated type!
I was delighted to read a friendly. interesting (though sadly a bit short!) letter, the message on the back of your envelope was particularly heartwarming, Alison, I'll treasure it, my very first letter!
It was also very helpful that you detailed which type of machine you were using (memory wise), as I may have given you some routines too large for a P.A.W database, which was nearly full already (it's alright for those 128ers with lots of memory to waste!), so as I don't know how much of your memory I'm allowed to fiddle on with, I'm going to keep it down to a minimum, okay, Alison?

It's also a great pity you couldn't have been a bit more specific about what you wanted, but here goes..
So, if you only wanted your non-drop routines to work in specific rooms, and to work normally in others, you could try..

DROP _ 0 WHATO LT 34 255 EQ 54 254 ATLT 12 MESSAGE A NEWTEXT DONE

Now these routines must come at the very beginning of any DROP _ routines you may already have in response, and the rest is easy enough, as the WHATO gives the number of the object mentioned, and if it's a real object (ie, flag 34 is less than 255) and is being carried (is in room 254), and you're also in a room with a number less than, for example, 12, message A is printed and newtexted, which could always work the other way, using ATGT ?, so for ATLT 12, Alison, you'd have to make sure any room that you DO want your DROP _ to work normally in, have a room number greater than 11, it being 1 less than that used by the ATLT ?

DROP _ 0 WHATO LT 34 255 EQ 54 254 MESSAGE B NEWTEXT DONE

Which just gives a 'But you'll have to use the _ somehow, you can't just put it down!' type of message, so that the player would have to find a use for whatever object it was, to use it up, before they could pick up anything else, but it's a horrible trick to play I must admit! So once you've got that you don't need the normal..

DROP _ AUTOD DONE

Action anymore. Or..

DROP _ 0 WHATO LT 34 255 EQ 54 254 CHANCE 50 MESSAGE A NEWTEXT DONE

To give the player the disability of only being able to drop whatever they want SOME of the time, and there's always..

DROP _ 0 WHATO LT 34 255 EQ 54 254 ATLT 12 PUTO A MESSAGE C DONE

Which could give you a routine something like that used in 'Very Big Cave Adventure', so that another character could come along, steal anything that's dropped, using PUTO A to place anything dropped into room A, the other characters inventory!

I don't know if any of the above are any good to you, but you didn't really tell me what you wanted off me, you did mention your map, but I didn't know if you wanted any hints with it? But if you ever have any problems no matter how small (and that goes for any other readers out there!), for anything at all, or if I haven't given you just what you want here, Alison, please write again and maybe you can be a bit more specific, and tell me just what you want, I'll try and do my best!

So there you go! A whole column just for you, well worth a 20p stamp (yes, it's gone up again!) to get your name in print, just like my one, my only reader so far, Miss Alison York!

TRANSPORT IN ADVENTURES

by Shaun Allaton

PART II: TAXI!

In my last section I showed how to implement cars into your PAWed adventure. Basically this section is the same but with a few minor changes and additions. Of course, on this occasion, you don't actually drive the car, that's left upto the cabbie so we'll also have to add some character interaction as well. Also, to make things more realistic, I've made this version so that the player has to pay for the ride (oo-er). If he/she doesn't then they get a black eye and meets our friend in heaven.

Flags

60 - The location of the cab
61 - Amount of money the player is carrying
62 - Cab fare

Objects

0 - some money

Messages

01 A cab pulls up beside me.
02 I was inside a cab, so couldn't call another one.
03 One was already here.
04 I couldn't call one from here.
05 I was already in the cab.
06 A cab wasn't here.
07 The cabbie wasn't happy with you trying to get away without paying so sends you to that place in the sky.
08 Climb out of what?
09 Climb into what?
10 The cabbie wasn't here.
11 "That's '
12 " pronounced the cabbie as we reached our destination.
13 "We're already here guv."
14 "Sorry mate. I don't know where that is!"
15 I didn't have any money.
16 The cabbie wasn't here.
17 You owe him nothing, nothing at all!
18 The money amounted to '
19 A cab stands before me.

Process Table 0/Responce Table

```
HAIL  CAB     NOTSAME 38 60
              ATGT 0
              ATLT 3
              MESSAGE 1
              COPYFF 38 60
              DONE
```

This first command checks that a) the cab is not already in the same location and b) that the player can call a cab from where he/she is standing i.e they are in location 1 or 2. Then PAW tells the player that a cab pulls up and then the location of the player (flag 38) is copied so that the location of the cab (flag 60) is the same.

```
HAIL  CAB     AT 3
              MESSAGE 2
              DONE
```

Tells the player they can't call a taxi if already inside one.

```
HAIL  CAB     SAME 38 60
              MESSAGE 3
              DONE
```

Basically the same as above but checks to see if the cab is standing next to the player i.e ready for the player to climb into.

```
HAIL  CAB     ATGT 3
              MESSAGE 4
              DONE
```

In this demo, if the player is in a location with a higher value than 3 then they cannot call the cab, and are told so.

```
CLIMB CAB     PREP IN
              NOTAT 3
              SAME 38 60
              GOTO 3
              DESC
```

First PAW checks that as well as CLIMB CAB, IN has also been typed in, then PAW checks if the player is in the same location as the cab (SAME 38 60), if so, then the player is moved to location 3 (inside the cab).

```
CLIMB CAB     PREP IN
              AT 3
              MESSAGE 5
              DONE
```

This tells the player that he/she is already inside the cab.

```
CLIMB CAB     PREP IN
              MESSAGE 6
              DONE
```

This tells the player that a cab isn't around.

```
CLIMB OUT     AT 3
              ZERO 62
              COPYFF 60 38
              DESC
```

First PAW checks that the player is inside the cab and that the player doesn't owe any money to the cabbie then it copies the cab's outside location (flag 60) to that of the player (flag 38) and describes the new location.

```
CLIMB OUT     AT 3
              NOTZERO 62
              PAUSE 100
              MESSAGE 7
              PAUSE 100
              END
```

Death routine.

```
CLIMB OUT     NOTAT 3
              MESSAGE 8
              DONE
```

If the player isn't inside the cab then PAW asks the player: "Climb out of what?"

```
CLIMB _       PREP IN
              MESSAGE 9
              DONE
```

If the player just types CLIMB IN then PAW asks: "Climb into what?"

```
PAY   CABBI   AT 3
              NOTZERO 62
              CARRIED 0
              SUB 62 61
              CLEAR 62
              OK
```

First off PAW checks that some money is being carried (object 0) then subtracts the amount of the fare from the amount of the money which the player is carrying and then clears the amount of fare due to 0.

```
PAY   CABBI   AT 3
              NOTZERO 62
              NOTCARR 0
              MESSAGE 15
              DONE
```

The player is told that they possess no money.

```
PAY   CABBI   AT 3
              ZERO 62
              MESSAGE 17
              DONE
```

This tells the player that no fare is due.

```
PAY   CABBI   NOTAT 3
              MESSAGE 16
              DONE
```

Tells the player that the cabbie isn't around.

```
COUNT MONEY   MES 18
              PRINT 61
              DONE
```

Informs the player how much dosh is being carried.

```
SAY   CABBI   AT 3
              PROCESS 3
              DONE
```

If the player is inside the cab then PAW goes to Process Table 3 to check for speech to the cabbie.

```
SAY   CABBI   MESSAGE 10
              DONE
```

Tells the player that the cabbie isn't around.

Process Table 1

```
*     *       AT 0
              SET 61
              GOTO 1
              DESC
```

If the player has just started a new game (is at location 0) then set the amount of money being carried to 255 and remember when entering the money in as an object to set it so that it is carried at the start of the game.

```
*     *       SAME 38 60
              MESSAGE 19
```

If the cab is beside the player then he/she is told so at the end of the location description.

Process Table 3

```
*     *       PARSE
```

Gets the speech that the player wishes to give to the cabbie.

```
*     OFFIC   NOTEQ 60 2
              LET 60 2
              PLUS 62 3
              MES 11
              PRINT 62
              MES 12
              DONE
```

Something I forgot to mention at the start was that the two locations the player can visit are his/her home (location 1) and the office (location 2). This routine first checks that the player isn't already at the office and then drives to the office and tells the player the amount owed and adds it to flag 62.

```
*     OFFIC   MESSAGE 13
              DONE
```

Tells the player that they are already at the destination.

```
*     HOME    NOTEQ 60 1
              LET 60 1
              PLUS 62 3
              MES 11
              PRINT 62
              MES 12
              DONE

*     HOME    MESSAGE 13
              DONE
```

The above two commands work in the same way as the previous two.

And believe it or not that's it. Quite a long routine but it's effective and I've found it works well.

Next time I'll be looking at how to implement Trains into your PAWed games and like BR. how to make them run late!!

Last but not least I'm looking for adventure games to publish —

USEFUL PAW ROUTINES

by Darren Rose

AUTOMATIC EXIT PRINTING

To print all available exits as part of the location description:-

Process 1

```
_     *     MES 100

_     S     COPYFF 38 100   LET 33 2   MOVE 100   ADD 100 101
            NOTZERO 100   MES 101

_     E     COPYFF 38 100   LET 33 3   MOVE 100   ADD 100 101
            NOTZERO 100   MES 102

_     W     COPYFF 38 100   LET 33 4   MOVE 100   ADD 100 101
            NOTZERO 100   MES 103

_     N     COPYFF 38 100   LET 33 5   MOVE 100   ADD 100 101
            NOTZERO 100   MES 104

_     U     COPYFF 38 100   LET 33 11   MOVE 100   ADD 100 101
            NOTZERO 100   MES 105

_     D     COPYFF 38 100   LET 33 10   MOVE 100   ADD 100 101
            NOTZERO 100   MES 106
```

Messages

```
100  Visible exits -
101  S
102  E
103  W
104  N
105  U
106  D
```

SWEAR PROTECTION

A very useful feature in an adventure is swear protection, this routine ends the game when a swear word is entered.

Vocabulary

Put all the swear words you can think of in here as Noun 150.

Response Table

```
*     (swear word)     BEEP 10 10   MESSAGE 50   ANYKEY   END
```

Message 50 = "Don't swear, just for that you can start again."

AGAIN

This is another useful routine, used in all professional adventures.

Verb 56 = AGAIN

Response Table

```
AGAIN *      CLEAR 240   EQ 33 56   COPYFF 233 33   COPYFF 234 34
             COPYFF 243 43   COPYFF 244 44   COPYFF 245 45   SET 240

*     *      COPYFF 33 233   COPYFF 34 234   COPYFF 43 243
             COPYFF 44 244   COPYFF 45 245

_     _      MOVE 38   DESC

_     _      LT 33 14   SYSMESS 7   DONE
```

OOPS!

Another useful command is OOPS so if you make a mistake the game will go back a command when you type it.

```
*     *      0  EQ 33 29   RAMLOAD 255   DESC

*     *      1  RAMSAVE
```

CLOCKS

This is a very useful feature for a real-time clock running in memory.

Flags used

150 minutes
151 hours
152 days

Process 2

```
*     *      PLUS 150 1   EQ 150 60   CLEAR 150   PLUS 151 1
             EQ 151 24   CLEAR 151   PLUS 152 1   EQ 152 7   CLEAR 152
```

Process 6

```
_     _      COPYFF 151 153   GT 153 11   MINUS 153 12

_     _      PRINT 153   MES 2   PRINT 150   GT 151 11   MESSAGE 3
             DONE

_     _      MESSAGE 4
```

Response Table

```
LOOK  CLOCK  MES 1   PROCESS 6   DONE
```

Messages

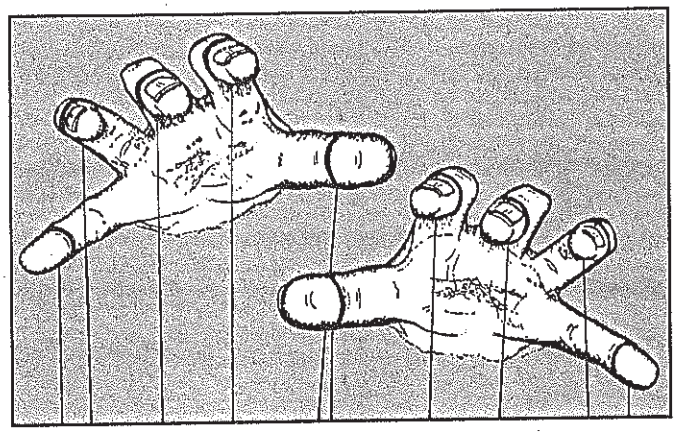
1 The time is
2 :
3 PM
4 AM

Using this feature I have written routines for shops which open and close, and also for buses and trains that arrive and leave at certain times. At the moment I am writing a routine for characters who arrive and leave and do certain things at specific times.

JOHN AND HELEN GET INTER-ACTIVE

Adventure-writing utilities are terrific programs, but the wonderful things they can do aren't always that easy to discover – not without a lot of head-scratching, late nights, muttering and mistake-making along the way. It's a lot more simple if you've got an expert to help you, someone who's good enough to share their own secrets with you. Such a wonderfully warm and kind-hearted human bean is Mark Cantrell, author of that marvellous game, *Diablo!*

Mark's sent me the following article explaining how you can have more than one main character in your *PAW*'d adventure, and switch control of the game between them. So over to Mark...



The following routines will allow an adventure to use a number of characters which the player can switch between at any point in the game, as seen in commercial releases like *Runestone* and *Terrors Of Trantoss*. When not controlled by the player, such a character could become a normal PSI (Pseudo-Intelligent Character).

This first routine will simply show you how to switch between two characters, John and Helen. In this example, the PSIs are entered in the object table as objects – John is the first entry, Helen the second. Their nouns should be entered below 50 in the noun table, and also entered in the object words table.

RESPONSE:

```
CONTROL JOHN
EQ 13 10; check controlling John already?
SYSMESS 87; then say so
DONE; finish the command
CONTROL JOHN
NOTZERO 14; is he dead?
SYSMESS 88; then say so
DONE
CONTROL JOHN
PROCESS 6; call primary exchange
routine
DONE
CONTROL HELEN
EQ 13 20; controlling her already?
SYSMESS 87; then say so
DONE
CONTROL HELEN
NOTZERO 15; is she dead?
SYSMESS 89; then say so
DONE
CONTROL HELEN
PROCESS 6; switch to primary exchange
routine
DONE; finish response section
CONTROL
SYSMESS 90 DONE; prints "Control
who?" if noun is not HELEN or JOHN
```

The above section of the routine establishes the player's ability to change characters. Each character will need the three lines. The first checks that the player is not already controlling that character, and if he is then a message is printed to say so (message 87). The second line checks that the particular character is alive, and if they're not then the player will be told he cannot take control because that character is dead (messages 88/89). The third line calls the first section of the exchange routine... of which more in a mo'.

The last line merely asks the player which character he wishes to assume if he has failed to specify one, mis-typed the name or typed the name of a character they cannot control. Since characters can die in most adventures and since with this routine the game need not come to an end until all the player characters are dead, then the player must not be able to enter commands when in control of a dead character. Of course in a small example such as this, with just two characters, the game can simply put the player in control of the surviving PSI. When using more than two characters the following lines will be needed...

RESPONSE:

```
*_*0
EQ 13 10; in control of John?
NOTZERO 14; is he dead?
NOTEQ 33 x; verb is not CONTROL
SYSMESS 88; "He is dead."
DONE; exit the table
*_*1
EQ 13 20; controlling Helen?
NOTZERO 15; is she dead?
NOTEQ 33 x; verb is not CONTROL
SYSMESS 89; "She is dead."
DONE; exit the table
```

These entries intercept any command made which is not CONTROL. *PAWS* will still be taking its turn at the game, which means that while you are dead game time will still be passing for other characters. Once the character is dead you may want to move it into an empty room, otherwise you'll find the game printing reports of a character's movements to you when it's supposed to be six feet under!

FLAGS:
11: John's room
12: Helen's room
13: character indicator
14: John dead?
15: Helen dead?
61: indicate changing characters
62: dump/collect worn or carried objects

SYSTEM MESSAGES:
87: "You are already controlling that person!"
88: "John is dead."
89: "Helen is dead."
90: "Control who?"

ROOMS:
4: John's carried objects store
5: John's worn object store
6: Helen's carried object store
7: Helen's worn object store

The next section of the routine, the primary exchange section, begins the process of changing characters. Each character must have one line associated with it for each of the other characters. So, for example, if the game has four characters then there must be three of these lines per character...

PROCESS SIX:

```
CONTROL JOHN EQ 13 20; controlling
Helen?
GOTO 6
PROCESS 7; dump Helen's carried objects
GOTO 7
SET 62; set to worn objects
PROCESS 7; dump Helen's worn objects
SET 61; indicate changing character
LET 16 10; set control to John
CLEAR 62; reset to carried objects
CONTROL HELEN
EQ 13 20; controlling John?
GOTO 4
PROCESS 7; dump John's carried objects
GOTO 5
SET 62
PROCESS 7; dump John's worn objects
SET 61; indicate changing character
LET 13 20; change to Helen
CLEAR 62; reset to carried objects
```

The above lines form the primary exchange routine. The routine checks which character the player is controlling by using the EQ 13 'condact'. The routine must determine which character is being controlled so that the objects can be properly dealt with, hence there being one entry for each of the other characters, as mentioned.

Having determined which character is currently controlled, the line then places the player in that character's carried object store. It then calls a sub-routine to dump all the objects in that location. The

routine then sends the player to that character's worn object store. Flag 62 is set to indicate that worn objects must be dumped or collected, depending on which part of the exchange routine called it, then the dumping routine is called once more.

Now Flag 61 is set, which will ensure that the secondary exchange routine is active in Process Two and also tell the dumping routine to collect objects instead. Flag 13 is then set to 20 to indicate that the new character has been selected, and Flag 62 is cleared to indicate that the second part of the routine must collect objects that should be carried.

PROCESS SEVEN:

```
*-*
ZERO 62; move carried objects?
ZERO 61; dumping objects?
DOALL 254; scan for carried objects
*-*
ZERO 62; as above
ZERO 61; as above
PUTO 255; place objects in current
location
DONE; repeat or end loop
*-*
NOTZERO 62; move worn objects?
ZERO 61; as above
DOALL 253; scan for worn objects
*-*
NOTZERO 62; as above
ZERO 61; as above
PUTO 255; put all objects in current
location
DONE; repeat or end loop
```

The above lines deal with dumping objects and this is the section which is active when called from the primary exchange routine. The lines look at which objects are being dumped, either carried or worn (ie Flag 62 set or clear), and establishes a DOALL loop in the relevant location – room 254 for the player's carried objects and 253 for the player's worn objects.

The next line then processes the objects discovered by the DOALL loop, checks for the correct type (ie carried or worn) and then places the objects that have been found into the current location (255) which would be the relevant object store. The DONE 'condact' continues the loop or exits it if no other objects are found.

PROCESS SEVEN:

```
--
ZERO 62; move carried objects?
NOTZERO 61; collecting objects?
DOALL 255; scan for objects in location
--
ZERO 62; as above
NOTZERO 61; as above
PUTO 254; place objects in carried room
DONE; finish loop
--
NOTZERO 62; move worn objects?
NOTZERO 61; as above
DOALL 255; scan for objects in location
--
NOTZERO 62; as above
NOTZERO 61; as above
PUTO 253; place objects in 'worn' room
DONE; finish loop
```

The next part of the routine deals with collecting objects from the newly assumed character's store rooms and placing them in the player's inventory rooms (253 and 254). It works in the same way as the first four lines, but the DOALL loop looks for objects in the player's position, the object store rooms, and places them in the inventory. These lines could be separated into two sub-routines, but bear in mind that each process table has an overhead of 12 bytes. The secondary exchange routine is entered in Process Two...

PROCESS TWO:

```
*-*
EQ 13 10; now controlling John?
NOTZERO 61; collecting objects?
GOTO 4
PROCESS 7; collect carried objects
GOTO 5
SET 62; indicate worn objects
PROCESS 7; move worn objects
CLEAR 62
CLEAR 61; reset exchange flags
COPYFF 11 38; put player in new location
DESTROY 1; remove character object
DESC; describes the new location
*-*
EQ 13 20 NOTZERO 61 GOTO 6
PROCESS 7 GOTO 7 SET 62 PROCESS 7
CLEAR 62 CLEAR 61COPYFF 12 38
DESTROY 2 DESC; as above but for Helen
JOHN
*-*
EQ 13 10; controlling John?
ZERO 14; is he alive?
COPYFF 38 11; update PSI's position flag
*-*
EQ 13 20 ZERO 15 COPYFF 38 12; as
above but for Helen
```

The first two lines of this routine take care of collecting the character's objects and placing them into the player's inventory. It is essentially the opposite process to dumping the old character's objects. It then places the player in the location previously occupied by the new character as a PSI and removes the character as an object. It then describes the location.

The next two lines maintain the position of the character by updating the character's location flag so that it may be placed at its location correctly when the player assumes another character.

Finally, Process One must have a couple of lines to move the PSIs and keep the object in the same place that its location flag indicates...

PROCESS ONE:

```
*-*
NOTEQ 13 10; not controlling John?
ZERO 14; is he alive?
COPYFO 11 1; place object at specified
location
*-*
NOTEQ 13 20 ZERO 15 COPYFO 12 2; as
above but for Helen
```

This routine is merely the bare bones. You may want to provide your characters with some attributes, such as morale, strength, stamina etc, in which case each PSI could have its own store flags to hold these values and be exchanged between such flags and some work flags whenever the player assumed control of another character. In that way, one set of processes could be used upon the attribute flags for all characters when under player control rather than a separate process for each.

If any of the above is not clear, contact Mark at 112 Upperwoodlands Road, Bradford BD8 9JE. He won't be able to answer each and every *PAWS* question, so don't inundate him with all your programming queries.

KET KORNERED

A lot of you have been asking me questions lately about *Mountains Of Ket*. Which is all very well, but even I've had trouble with a handful of them. Hard to believe, dear hearts, I know, but true. So, lo and behold, who should come charging to my rescue but that solver supremo Tom Frost. And here's his solution...

N, GET CHAIN, S, S and hear "TALPS" (if not, go S-N-S until you do). E, E, N, N, BUY AXE, S, S, S, CHOP LOG, GET LOGS, N, N, E, N, GIVE LOGS, READ MAP, S, DROP MAP, W, N, SELL AXE, BUY POTION AND LAMP, DRINK POTION, S, S, W, BUY AND RIDE HORSE, E, S, E, N, N, E, E, GET DICE, W, S, E, E, GET BEARD, DISMOUNT HORSE, LIGHT LAMP, S, W, GET FOOD, E, S, W, FEED DOG, GET DOG, E, N, TIE DOG, DROP BEARD AND DICE, S, E, GET WAND, W, N, E, E, SAY POLO, E, GET AND READ GAZETTE, W, S, SAY TALPS, S, GET BOTTLE, D, FILL BOTTLE, S, GIVE GAZETTE, N, U, N, N, W, DROP GOLD, DROP BOTTLE AND SWORD, GET DICE, E, S, S, S, S, S, SAVE, PLAY DICE, Y (press any key and if dice game lost then LOAD and play again 'til won).

N, N, N, N, W, DROP DICE, GET BOTTLE, E, E, SAY POLO, E, S, S, U, POUR OIL, OPEN DOOR, W, W, SAVE, N and fight. Don't take the escape option. If you lose a lot of energy points then LOAD and go through fight sequence again.

GET RING, S, D, GET CLOAK, U, E, OPEN DOOR, E, D, W, GET NECKLACE, E, N, N, W, W, DROP RING AND NECKLACE, DROP CLOAK, E, E, SAY POLO, E, E, S, FILL BOTTLE, GET DIAMOND, SAVE, N and fight. As before, LOAD and fight again if you lose energy points.

W, S, S, U, OPEN DOOR, W, W, D, S, SAVE. In any fight with the dragon, take the first possible escape option then take the appropriate direction. E, S, S, S, DRINK ELIXIR, S, GET HAT, N, N, N, N, N, GET BRACELET, S, W and if energy level is now unacceptable just LOAD and try again from last SAVE position.

N, U, E, OPEN DOOR, E, D, N, N, W, W, DROP BRACELET AND DIAMOND, DROP BOTTLE, WEAR HAT, GET AND WEAR CLOAK, GET AND WEAR BEARD, S, S, S, GET ARMOUR, N, N, N, DROP BEARD, DROP HAT AND CLOAK, GET AND WEAR RING, WEAR ARMOUR, S, S, DROP LAMP, S, D and Y if you wish to load the next part.

GENERAL NOTES

- For foreign adventures, a Polo is an English mint sweet, so you must SAY POLO at the mint wall.
- Do not initiate a fight with anybody or anything.
- It's possible to accumulate a high energy level to carry forward to the next adventure in the trilogy, *Temple Of Vran*.
- You will drown at the lake if you carry too many items.
- The effect of the elixir is only just sufficient for the moves indicated.
- Any items that are dropped and then taken by the rats are found in the maze.
- There's no apparent method of ensuring a victory in the dice game with the ogre.
- TALPS is merely "SPLAT" spelt backwards, *Splat* being an earlier arcade game from the same publishers, Incentive.

LOOK AT THESE TWIN GLOBALS!

Yet again, the mega-stupendous *YS* has come up with an exclusive adventure-type offer for its lucky old readers. Nowhere else in the entire known universe, not even in the blackest black hole way off in the farthest-flung corner of the galaxy (the corner where the dust all gathers), will you be able to get your hands on these two-for-one tapes from Global Games.

Global's the company that's run by William Young, he of Spectrum Adventure Exchange Club fame, and wee Wullie from Bonnie Scotland is offering *YS* readers the chance to buy two of his adventures for just £1.99, including postage. Or four for, erm, £3.98. Or if you're feeling really rich you can get all six for... gosh... oh, about... three twos is six, less... about £5.97. Clip the pretty little coupon toot sweet, bargain-hunters.

If you want to know a smidgeon more about the games, then *Number 6* is based on the cult TV series *The Prisoner*; *Virus* is a Dr Goo adventure well reviewed in the past; and *Once Upon A Lily Pad* is a chucklesome little tale of life as a frog.

TO: Dept *YS*, Global Games, 4 Kilmartin Lane, Carluke, Lanarkshire, Scotland ML8 5RT.

FROM:

Name .....

Address .....

.....

..... Post Code .....

I enclose my cheque/PO/money order for £..... for the following Speccytrum tapes which I have ticked at £1.99 each.

❑ Tape One: *Number 6: In The Village/Once Upon a Lily*

❑ Tape Two: *Dark Storm/Virus*

❑ Tape Three: *The Limehouse Mystery/Survivor*

DROP YOUR KNICKERS, TAKE TWO

Last month Mark Cantrell put objects into containers, this month he takes them out...

Removing Objects from the Container

Removing the objects is just as easy as putting them in. The example here ignores any dealings with the wearing or removal of objects, though in most adventures REMOVE is likely to be used for taking off clothing etc. This shouldn't make it any more complicated. It also depends how fussy you want to be. If the player is wearing a raincoat and types in PUT RAINCOAT IN BAG, it's up to you whether you insist the raincoat has to be removed first, or whether it can go straight into the bag in one action – a bit like whether you make the player type OPEN DOOR before he or she can go through it, or whether you respond with "You open the door and go through."

```
REMOVE   NOUN2 BAG; Referring to
         the bag?
         PRESENT 1; Is the bag
         here?
         AUTOT 1; Take the object if
         it's inside
         DONE

REMOVE   NOUN2 BAG; As above
         ABSENT 1; The bag isn't
         here
         SYSMESS 26; Prints "There
         isn't one here!"
         DONE

REMOVE   SYSMESS 60; "Remove
         from what?"
         DONE

REMOVE   ALL      NOUN2 BAG;
                  As above
         PRESENT 1; As above
         DOALL 1; Look for objects
         inside the bag
```

There is no need to specify the container in the REMOVE commands, as I'll demonstrate in a moment when covering GET. Of course, this does not apply to the ALL option. You can't nest DOALL loops (i.e. perform a DOALL in one location, switching each time to a DOALL in another location) so you must be specific when GETting ALL objects from a container.

New users of *PAWS* should remember that it is important which order the commands are placed in, and again to finish off with a DONE. In this example, the first input checks whether the bag is present or not, and if it is then it acts upon that, with the DONE command telling it to check no further. If the bag isn't present, the program goes on to check your next entry, and will process that command instead – logic tells you that either the bag is present or it is absent. It has to be one or the other.

All these lines work in a similar fashion to the PUT lines. However, the AUTOT condact does the opposite to AUTOP. It looks for the specified object in the specified location (room one) and then processes it in a way similar to GET, reporting the successful manipulation of that object, if the player has it, or reporting if no such object is to be found inside the container.

The following lines do the same job as the lines for PUT. However, the DOALL loop for the ALL option looks for objects in the container rather than in the player's inventory.

Get and Drop

Let's consider how the GET command works in conjunction with containers:

```
GET      PRESENT 1; Is the bag
         around?
         WHATO; Which object is
         being referred to?
         EQ 54 1; Is the object in the
         bag?
         AUTOT 1; Get the object
         DONE

GET      ALL      NOUN2 BAG
         PRESENT 1
         DOALL 1; See REMOVE
         ALL
```

These lines deal with taking objects from the container. The first line checks that the object is present. It then looks at the noun1/adjective1 used in the input to look up the object in the object word table, updating the relevant flags for the last 'GET entries which, if the object was not found in the container, will be processed.

```
GET      LT 34 x
         GT 34 20; Nouns referring to
         objects?
         WHATO; Which object is
         referred to?
         EQ 55 0; Does it have zero
         weight?
         SYSMESS 61; Then report
         that it is immobile
         DONE

GET      LT 34 x
         GT 34 20; As above
         AUTOG; Get specified
         object
         DONE

GET      ALL      DOALL 255;
         Process all objects
```

The first line checks that the noun entered corresponds to an object. In other words those nouns above 'x' may refer to places etc, not objects, and so would lead to strange results if they were fed into the GET process. Those nouns below 20 are not object nouns, being movement nouns, for instance. Having established that the nouns are valid, the object is looked up in the object word table. Should the object have a weight of zero then it is an immobile object and this fact is reported to the player.

The next line again checks that the nouns are valid and then performs an AUTOG. This AUTOmatically Gets any specified object and reports this to the player, also reporting if the player already has it, or is wearing it, or if it is not in the present location. The last line merely sets up an ALL option which scans for objects in the player's location and processes them.

The opposite action – dropping objects into the container – has already been explained under PUT. All you have to do is substitute the verb PUT for DROP. However, the PUT lines don't explain the general DROP lines.

```
DROP     AUTOD; Drop the object
         done
         DONE

DROP     ALL      DOALL 254;
                  Scan for carried
                  objects
```

These two lines are so simple that they are self-explanatory. In fact most of these entries are fairly easy to understand. They were included here because any explanation of containers cannot really be considered complete unless the placing or removing of objects is mentioned. And when mentioning GET and DROP you cannot really omit the general lines.

However, the use of NOUN2 to specify the container was used in the first *PAWS* adventures and may be something which newcomers to *PAWS* are still inclined to do. It is quite unnecessary, and in fact it saves you some memory if you refrain from specifying the container in the GET condacts. It's also more user-friendly to allow the player to just type, for example, GET KEY rather than GET KEY FROM POCKET.

Thanks for those words of wisdom, Mark, and for making it all so jolly simple and straightforward! (Ahem.) Mark's happy to answer any questions you've got on the routines, but please don't inundate him with general *PAWS* queries, and be sure to enclose an sae when you're writing to him, at 112 Upperwoodlands Road, Bradford, West Yorkshire BD8 9JE.

❄ ***Phoenix*** is the newie from Zenobi. It's the chilling tale of an eternal being, The Traveller In Black (that's you), who's summoned to rid a once peaceful village of an entity known as The Abomination. This depraved demon from the primal darkness is one seriously *bad* dude!

The author's come up with some juicy, highly descriptive text, and also researched and implemented some pretty fine myth/legend-based problems for the player to bump into on his travels. A summer smash-hit of a game if ever there was one! *Phoenix* costs £1.99 and you can get it from Zenobi Software, 26 Spotland Tops, Cutgate, Rochdale, Lancashire OL12 7NX (overseas customers add £1 to cover the extra postal cost).

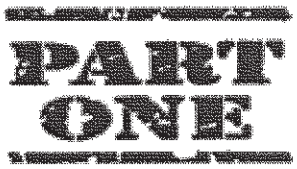
❄ **Delbert The Hamster Software** will probably be a new name to most of you. Scott Denyer is the brains behind the outfit, and his first release ***Desmond And Gertrude*** is out now. You play both Desmond and his sweetheart Gertrude, the aim being to get the two young lovebirds together. But Gertrude's dad, King Norbert, isn't keen on his daughter getting it on with a bloke who sleeps in the gutter and is slightly less brainy than the village idiot. So he builds a huge dividing wall down the middle of the village to keep the two apart!

And talk about value for money – there's even a small FREE game on the B-side of the tape. What a barg! *Desmond And Gertrude* could be yours if you send a cheque (or PO) for £2.99 (made payable to Scott Denyer) to, erm, Scott Denyer (of Delbert The Hamster Software), 9 Orchard Way, Flitwick, Beds MK45 1LF.

As space is a bit short for my first column (due to Mike's final bits and pieces) I'd better remind you that you can get in touch with me via the usual *YS* address, and I'm quite happy to help you out with any adventure-related problem, query, hint, tip or whatever (though if you can supply an sae with your letter then so much the better!).

Please address your letters to Tim Kemp, Adventures, *YS* (it makes life so much easier!) and, all being well, I'll see you next month. Ta-ra!

PLAYING AND WRITING ADVENTURE GAMES



First of all, what are adventure games? If you've never played one (you sad person), adventures are games that require the player to use a lot of thought, logic and reasoning to solve the puzzles programmed into the game by the author. The manual dexterity and fast reflexes required of most arcade games are completely redundant in adventure games.

This is one reason why these sort of games are so popular with adults, who make up the majority of adventure players. However, if you've got an inquiring mind and some time to spend anyone, even you, can play adventures and join the growing number of enthusiasts in Britain.

Although some adventures have graphics of some sort most are text-only and these are the ones we'll deal with for this feature. On screen you will see, in the top half, a description of the 'room' or 'location' you are currently in. This will include the 'exits' through which you can move to another location, given usually as compass directions, objects which form part of the location and other objects lying around that you can pick up and manipulate.

As you move in one of the directions described, eg. North, the location description will change, either by scrolling up the screen or, as in most adventures these days, by the new location description appearing in the text 'window'.

The idea of an adventure game is to solve the problems, puzzles and situations posed by the game as you progress through the adventure to the eventual goal. Hopefully these puzzles will range in difficulty from easy to hard, providing the player with plenty of mental stimulation, which is what adventure playing is all about.

Adventure games can be set in all sorts of different scenarios, but the two most popular are fantasy (eg. Dungeons and Dragons) and Science-Fiction (eg. our own Starship Quest). The majority of Spectrum adventures are 48K and only have one part, but there are also lots of multi-part adventures available in 48K and 128K-only versions.

MAP PROCEDURES

Before you sit down in front of your Speccy to play an adventure, equip yourself with a pencil, eraser and plenty of paper. This so that you may draw a 'map' of the adventure as you move around. A typical adventure map is illustrated below. This is actually a map of the mini-adventure I will be instructing you how to write in this very article.

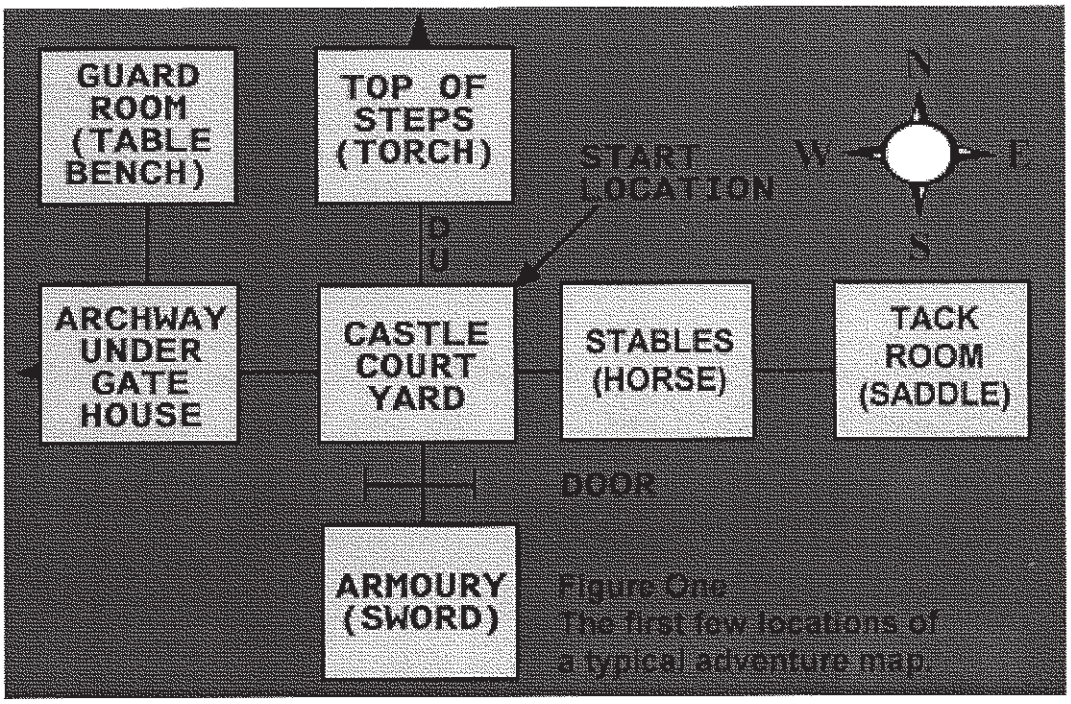


Figure One
The first few locations of a typical adventure map.

When drawing a map you should make squares as small as you can, according to how small you can actually write. A lot of adventure players even draw themselves a sheet of squares which they photocopy over and over to provide mapping paper. I myself do this using 10mm squares separated by 5mm gaps. As you move around the adventure, write in the boxes a short location name and any objects or characters you see there. Where there is an exit via a door that is locked, I draw a 'bar' across the exit line in a different colour to indicate a locked door, but you will certainly develop your own ways of indicating such features on your own maps.

MANIPULATE YOUR SURROUNDINGS

In all adventure games, you play by entering commands to manipulate objects and interact with characters, etc. You move around the locations by typing in compass directions, eg. N, E, S, W, NW, SE, etc. although in a very few games - eg. my own adventures "Magnetic Moon" and "Starship Quest" - nautical terms are used in parts of the game. (Starboard, Port etc.)

Most adventures accept VERB-NOUN inputs, eg. GET SWORD or WEAR HELMET, but the modern, highly sophisticated adventure writing utilities allow full English sentences to be input and understood by the computer, eg. GET HELMET AND WEAR IT THEN DROP SWORD AND GO NORTH. They also give the programmer the option to require the player to input multi-word commands that include adverbs, adjectives and prepositions! An example of such an input would be PUT THE BLUE BALL CAREFULLY INTO THE YELLOW BOX.

Three important verbs that are used in nearly all adventure games are EXAMINE, SEARCH and LOOK (the latter often used with a direction eg. NORTH, or a preposition eg. IN, AROUND or UNDER). When you have found an object and have picked it up, the first thing any adventurer will do is EXAMINE it to find out if there is anything unusual about it.

Some objects described in the location texts will be static, eg. a wardrobe or a bed, which you would not normally be able to pick up. This means that they have been put there for a purpose. You would normally SEARCH WARDROBE (often after first OPENIng it if closed) to see if there is anything useful inside. You would LOOK UNDER BED to see if there is anything hidden beneath it, and so on. It just takes a little thought to work these things out as they are basically common sense

actions.

CHARACTERISTIC CHARACTERS

Most adventures will have characters in them, either static ones who stay in the same location all the time, or PSI's - PSeudo Intelligent characters. These move from location to location, often independently of the player's inputs and actions. You can usually converse with these characters using the verbs TALK, SAY, ASK or TELL, and you will have to GIVE them objects or perform tasks for them. When conversing with characters, it is often necessary to put the speech in quotes, eg. SAY TO KING "HELLO", if you want the program to understand what you are inputting.

Some adventures are written with beginners in mind, others for the more experienced adventurer. The most basic of all puzzles is the locked door, where you have to find the key to unlock the door and thus open it to allow you to proceed into new locations. This key will be hidden somewhere and you may have to perform all sorts of tasks before you obtain it. Many puzzles will involve situations where you have to perform a task within a certain number of moves.

Death comes in all shapes and forms in adventure games, but, unlike most arcade games where you would have to restart from scratch, adventure games allow the player to SAVE their current game position to tape or disk at any stage and reload it after the player is 'killed' as well as at the start of the next playing session. Most adventures these days also have a RAMSAVE facility, whereby the player's position is saved to memory. This is particularly useful when you think that going in a certain direction or performing a particular action will result in the 'death' of the character.

STIMULATING INDEED

A good adventure game should stimulate your mind and make you want to go back to it time after time to try to solve it. As mentioned above, you can save your game position to tape or disk at any point during the game, and this is very handy when you have become 'stuck' at a certain point in the adventure. I have found that turning the computer off and doing something else while thinking about the problem you are trying to solve is more productive than sitting and staring at the monitor screen.

You do, however, tend to get very odd looks when, having had a brainwave and realized the solution to a perplexing problem, you exclaim "Ah! THAT'S how you stop the dragon eating the fairy princess!!" out loud on a crowded bus whilst on your way to work! Believe me, I've actually done it!

There can't be many adventure players who have not, at some stage, expressed the wish to write their own adventure game. Up to a few years ago, the only adventure games you could buy were produced, like arcade games, by big commercial software houses. Then came the advent of the 16-bit computer and, at around the same time, Role-Playing Games, or RPG's, started to really take off.

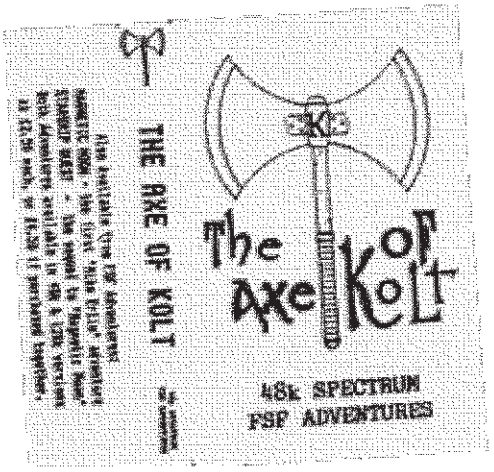
The traditional text-only or text-graphic adventure games became fewer and fewer as the commercial adventure software houses all abandoned 8-bit computing and switched to 16-bit hardware and RPG software. Luckily, at around the same time as this was happening, two software companies released adventure-writing utilities that would enable anyone with the desire to actually produce their very own adventure game. These utilities were "THE QUILL" by Gilsoft and the "GRAPHIC ADVENTURE CREATOR" (or "GAC") by Incentive.

HOMEGROWN ADVENTURES

These utilities opened the way for ordinary adventure players to write their own adventures - and

write them they did! The 'home-grown' adventure market has kept 8-bit adventuring alive, especially on the Spectrum, and there are literally hundreds of titles available for an average price of around £2.50 each.

Home-grown adventure games are available only through mail order, and the 'companies' that sell them are usually only one-person (or family) operations working from their homes, eg. Compass Software (run by Jon Lemmon), River Software (Jack Lockerby), Delbert the Hamster Software (Scott Denyer), my own Fantasy and Sci-Fi Adventures (FSF), The Guild (Tony Collins) and the daddy-of-'em-all, Zenobi Software, run by John Wilson.



These days, Zenobi specializes in publishing adventures written by amateur authors who don't want the hassle of selling their adventures. They currently have over 100 adventures on their lists.

GAC is no longer available commercially, and The Quill has been superseded by the "PROFESSIONAL ADVENTURE WRITING SYSTEM", which is know as PAW. This excellent utility is still available from Gilsoft and details are given below. I myself use PAW to write my adventures and I have found over the past three or four years that many potential adventure authors have been put off writing an adventure using this utility by the apparent complexity of the instruction manual.

While I myself, having started adventure writing on the BBC using The Quill, had no trouble understanding the PAW manual, I do believe that it tries to teach you too much, too soon. The mini-adventure it contains instructions for contains two PSI's and also gives details for producing a graphic picture to illustrate one of the locations.

I will state at this point that I think that graphics, specifically location graphics, are a waste of time for this type of adventure and I will not be referring to them at all in this series. Over the next few months I will be giving instructions on how to write a small adventure of just eight locations, but including some techniques not mentioned in the PAW tutorial game which I think are more appropriate and useful to the beginner.

First of all, however, you must get yourself a copy of PAW. You can obtain an original copy of PAW from Gilsoft, South Glamorgan. The tape version of the utility is available in a starter pack, costing £9.95, which consists of the PAW tape, the Tutorial Manual and the Quick Reference Guide (listing all the "conducts" the program uses). This contains all you need to produce a basic adventure. Disk versions of PAW are available on 3" disk for the +3 (and on 3.5" disk for +D Interface owners) at £20.95, but this includes the PAW "PTM" user overlays which are available on tape in the PAW Technical Pack at £8.95. If you already have a tape version of PAW, and wish to upgrade to a disk version, you should return the CASSETTE ONLY to Gilsoft with a cheque/PO for £5.50 for a +3 version or £4.49 for a 3.5" +D version. All prices quoted include P&P.

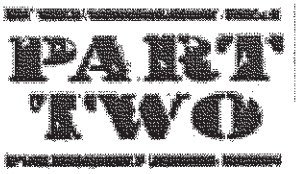
You can, of course, buy a second hand copy of PAW, but if you do ensure that all the manuals are present, especially if it includes the PTM User Overlays and that the version you are purchasing is one of the later versions, ie. A14 onwards. The earlier versions are lacking in certain features.

Next month we'll be starting on the adventure writing side of things, so until then, b-bye...!

By Larry Horsfield

PLAYING AND WRITING ADVENTURE GAMES

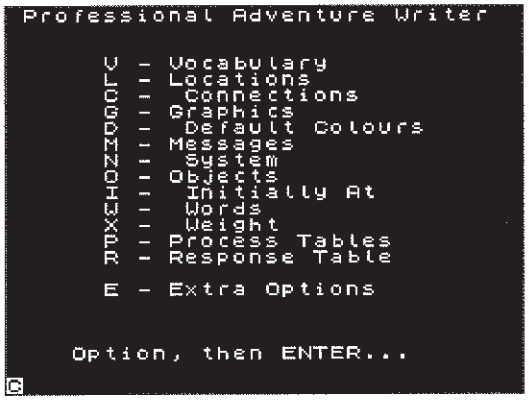
The second part of Larry Horsfield's adventure writing tutorial gives us a basic outline of the first location in "Dawn Patrol", your very own adventure ...



Having obtained a copy of the Professional Adventure Writing System (universally known as PAW) from Gilsoft you can power up your Speccy and load it with the instructions relating to your version of the Spectrum. Before I start though it's important to note the differences in the system when using different types of Spectrum. If you have a 128K Speccy then load PAW in 128K mode, it will work perfectly well on a normal +2. However if you have a 128K +2A (the one with the black casing) or a +3 beware, you should not, at any time, select and use the Text Compressor, option K on the main menu.

Due to the different circuit board installed in the +2A and the +3, the text compressor of the tape version will not work. If you do try to use it while your database is loaded it may become corrupted. Gilsoft are aware of this anomaly but have not been able to remove the bug.

If you have a 3in +3 disk or 3.5in +D disk version of PAW make sure to read the instructions relating to making a backup of PAW. With disk versions you can save your database onto your 'working disk', and if you save it using the filename "START", your database will load straight away from disk whenever you load PAW.



The first page of the main menu of PAW. Enter whichever function you need.

GETTING STARTED

O,K. PAW has loaded and the title page is on screen. Press any key to go to the Main Menu. You will find all the functions you need to create your adventure in this menu.

The adventure I'm going to teach you how to program is entitled 'Dawn Patrol' and consists of 8 locations, including location 0 which will be the "title page". The map of the adventure is shown in figure 1. The plot is as follows.

You are a soldier who is due out on mounted patrol in a few minutes. Some pranksters among your comrades have hidden your helmet and chainmail and you must find these before you can join the patrol. I will

also be including flag manipulation and screen display techniques that are not given in the PAW tutorial adventure. So make sure you follow closely.

IMPORTANT: Throughout these articles I will be instructing you to input entries into PAW, be they locations, messages or responses/process table entries.

Everything to be entered will be placed within brackets: {}. The "I" in the first set of brackets instructs PAW to insert the entry into the database. In later articles I will dispense with the brackets and the "I", as by then you should've got the hang of how entries are input into your PAW database.

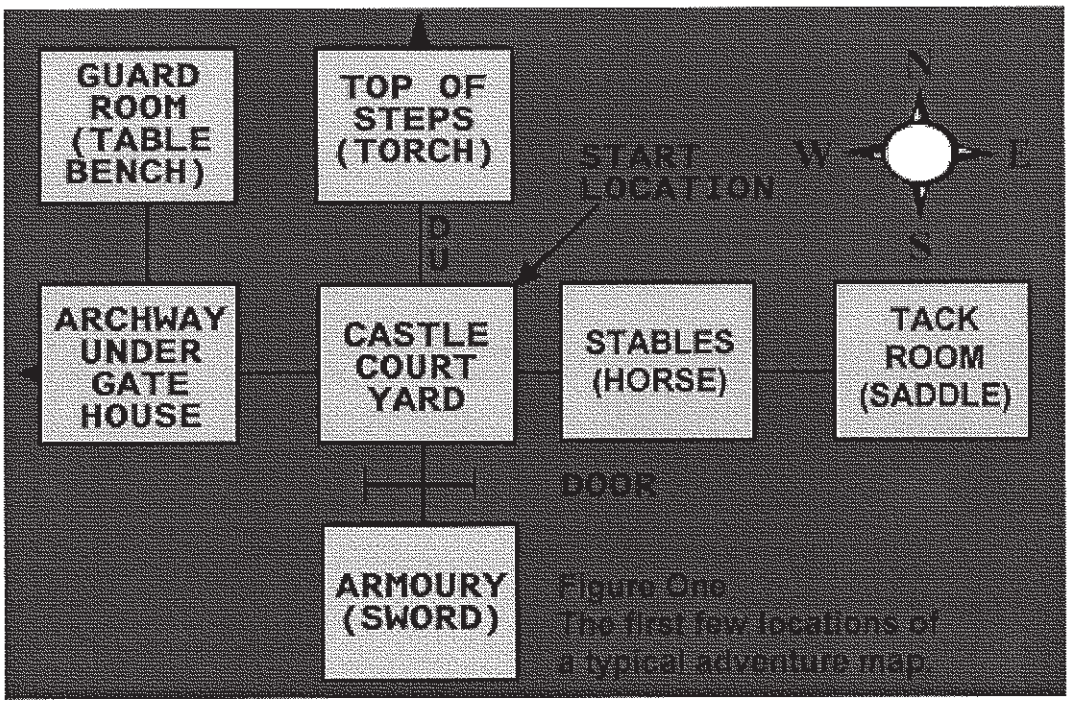


Figure One
The first few locations of a typical adventure map

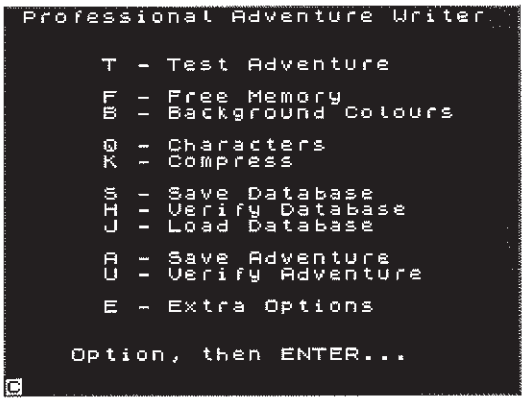
LOCATIONS

Let's start with the locations (which we will shorten to Locn). From the main menu, select L to enter the sub menu for location text. Locn 0 is already present and if you type {A 0} (Amend locn 0) you will see that a location text is already present. To clear this press EDIT twice. Any PAW adventure automatically starts at locn 0 and this is used as the title/introduction page. As we don't want the title to be on the very top line, we must insert a couple of blank lines. To do this, press EXTEND MODE then the 7 key followed by delete. You will see the cursor skip down one line. This is known as an ESCC7. Do this again to leave two blank lines at the top. Now press the space bar 10 times and then type the title {DAWN PATROL}. Now we want three more blank lines so repeat the ESCC7 inputs three times. Now for the introductory storyline, i.e. the plot.

Type the following in. Don't worry about words going onto the next line or there being a space at the start of any of the lines.

{You are a soldier in the army of King Zog and are due out on mounted patrol in a few minutes. Some pranksters have hidden your helmet and chainmail, so you must find these before you ride out to join your comrades - before your commander notices your absence! }

We must now insert the other 7 locns of the adventure so, from the locn Sub Menu enter I. You will see Location 1 at the top left of the screen and the cursor at the bottom left. Insert the text for the following locations, then if you type P you will see all 8 locations scroll up the screen.



The further adventures of the PAW main menu. So much choice!

Locn 1: {I am in the castle courtyard. The stables are east and an archway leads west under the gatehouse. On the north side a flight of steps goes up to the castle entrance and to the south is the door to the armoury, which is }

Locn 2: {The armoury is a cool, dark chamber with all manner of weapons in racks on the walls. The door, north, is }

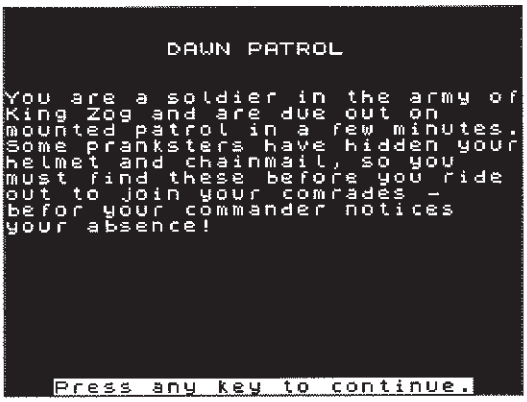
Locn 3: {I stand in the castle stables, with freshly laid straw underfoot. An opening leads east into the tack room and the courtyard is to the west. }

Locn 4: {This is the tack room, where racks and pegs hold all kinds of horse riding equipment. The stables are west. }

Locn 5: {At the top of the steps is the entrance to the castle itself. Two soldiers stand here on sentry duty, holding long halberds crossed in front of the open doorway. I can see the throne room north and behind me the steps lead down to the courtyard. }

Locn 6: {Under the gatehouse archway I can see, further to the west, the horse patrol I am supposed to be part of. A low doorway leads north into the gatehouse, and the courtyard lies east. }

Locn 7: {This spartan chamber is the guard room. The only comforts are a bench and table, and a fire that burns merrily in the hearth. On one of the walls is a large notice. }



The first ever adventure message you will produce. Well done. Have some tea.

You will notice that locn's 1 and 2 are missing a word at the end of the text. This will be printed on-screen when the adventure is running and I will detail how this is done later. You can also see that I have put a blank space at the end of each locn of text. This is because of messages which will be tagged on to the end of each locn text. Note that I also ALWAYS put a blank space at the end of every message I insert into a database and for the purpose of this tutorial you should do so as well.

PROCESS AND RESPONSE

Having inserted the locations, we must now insert the connections between them so that we can move around. This is done in the connections table for the locations where we can freely move around, but from the title page to location 1 we need an entry in PROCESS table 1. Movement into and out of the armoury, locn 2, is dealt with in the RESPONSE table, and this will be dealt with later. Process table 1 is scanned by PAW every time you move into a new location, or a DESC action is acted upon. From the locations Sub Menu, type Z to get back to the MM and then select option P, the process tables.

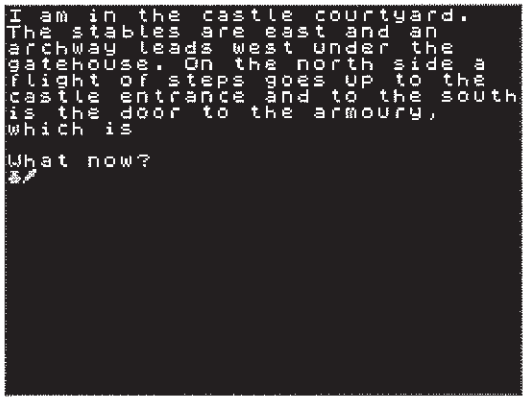
The first process table you go to is Process (PRO) 2, which PAW scans after each input from the player - in effect it is PAW's "go" at the adventure!

We need, as I said, to put an entry into PRO 1, so select that process table by typing S 1 and pressing ENTER. If you press P you'll see just two entries in the table, which deal with displaying objects on the screen according to whether it is dark and the light source that is available - more on that later.

Before I go on, I must tell you about the "null word" symbols used in PAW, these being the asterisk symbol "*" and the underline "_" symbol - this latter has another use which I'll mention later. The "*" will always be placed BEFORE any vocabulary word used in PAW and the "_" symbol. This symbol will appear AFTER the "*" or a word. The positioning of entries in the response/process tables of PAW is often very important. You can use the null words to place entries in a certain position so that they will be acted upon in preference to other entries if necessary. This will be illustrated later.

PROCESS 1

The two existing PRO 1 entries are "*_". The entry we need to insert into PRO 1 should go AFTER these two, so we use the same symbol. So, type I (for insert) followed by an asterisk then a space then an underline symbol, i.e. {I * _} then press ENTER. What you typed will now be at the top left with the cursor at bottom left. The entry we need to insert is as follows: {AT 0 ANYKEY GOTO 1 DESC}. This means that at AT location 0 the program should wait for ANY KEY to be pressed, then to GO TO location 1 and DESCribe that location. If you have typed that correctly, press ENTER. The word "Inserted" will appear top left under the "I *_" and you should press any key to return to the SM.



The second part of the first ever adventure message you will produce.

Now press Z for the MM and select option C for the connections table. If you screenprint the table by pressing P, you'll see that blank entries are present for all locations. Ignoring location 0, we'll insert the connections from loc 1, so type {A 1} and press ENTER. Consulting the map or reading the text for locn 1, you can go east to the stables, west to the archway and up the steps.

On the map I have indicated there is a door that leads south to the armoury and initially this door will be closed. Also, I intend that the armoury will be a dark location, where you will need a light source to be able to see, and therefore going south from location 1 and north from location 2 will be dependant on two of the FLAGS which PAW uses.

So, the connections entry for loc 1 will be: {E 3 W 6 U 5}. Below are listed the entries for the other locations, except for locn 2 which will be dealt with later:

Location 3: W 1 E 4
Location 4: W 3
Location 5: D 1
Location 6: E 1 N 7
Location 7: S 6

You will also notice there is no entry for going west from location 6, nor for going north from locn 5. This is because the player can only go west to join the patrol on the horse, and the sentries will

stop you going north into the castle. Having entered all those connections, we can at last test the adventure. Go to the MM and select option T to test the adventure. You will be asked if you require diagnostics, but just press ENTER. On the title page, you are instructed to "PRESS ANY KEY TO CONTINUE" and doing this will take you to location 1, the castle courtyard. From here you should be able to move into any location except the armoury. To break out of the Test Adventure mode, input QUIT then follow the prompts to return to the MM.

Before we deal with the movement into and out of the armoury, we must insert the messages that we wish to appear on screen in response to the player's inputs. This is where we'll pick up the story next month. In the meantime get yourself a copy of PAW and try out what we've done so far. I hate to sound like a school teacher but if you don't understand what we've done up 'till now you'll start getting lost next month.

Remember PAW is available from Gilsoft, South Glamorgan. It costs £9.95 as a tape starter pack with manual and is also available on +3 and +D disks and £20.95.

By Larry Horsfield

Previous article in series (issue 128)
Next article in series (issue 130)

Sinclair User
November 1992

PLAYING AND WRITING ADVENTURE GAMES

This month our mini adventure heats up and you get your first real chance to explore your newly created world. Adventure General, *Larry Horsfield's* Dawn Patrol marches on...

From the MM, select option M, the message table. There are 255 (0-254) messages available for use, as well as a further 200 system messages, (55-254) making a total of 455 altogether. We don't need that many, just the following to start with, as well as amending message 0, which is already present, like we did for the locations. Call message 0 by typing {A 0}, delete the text present and amend as below. As mentioned earlier in this tutorial, you should put a blank space at the end of each message. Especially message 0.

Message 0 The door is
Message 1 closed.
Message 2 open.
Message 3 The door swings open ...
Message 4 You close the door which shuts with a bang ...
Message 5 You can't.
Message 6 You can't leave the castle on foot"
Message 7 The sentries bar your progress. "Common soldiers are not allowed in the throne room" one of them barks at you.

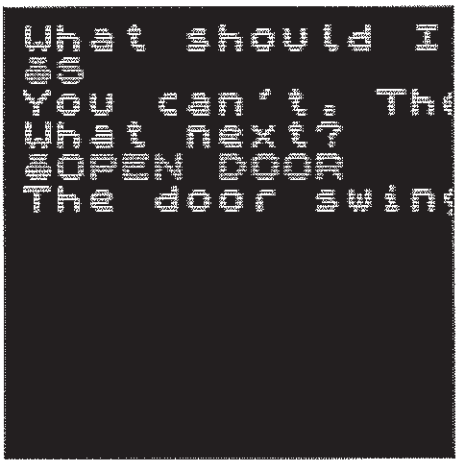


Fig 1. **You can't go south silly, the door's closed!**

These will do for now, but we must also insert a few words into PAW'S vocabulary so that the player's input can be understood, so select option V from the MM. The various types of words, e.g. verbs and nouns, are in different numbered sections. For now we only need to insert one noun.

Nouns are Section 2 and if you press {P 2} you will see that that a lot of nouns, mostly movement words, (i.e. West, East) are present. The number which a word is given is very important, as detailed in the manual, and we will insert DOOR as a noun with the number 50, so from the SM type {I DOOR 50 2} and press ENTER.

VERBAL AMENDMENT

Now we will amend the verbs. If you press {P 0}, you'll see that there are quite a few verbs, but there are some we don't want and a couple we will "shift". To delete any word, just type D then the word and press

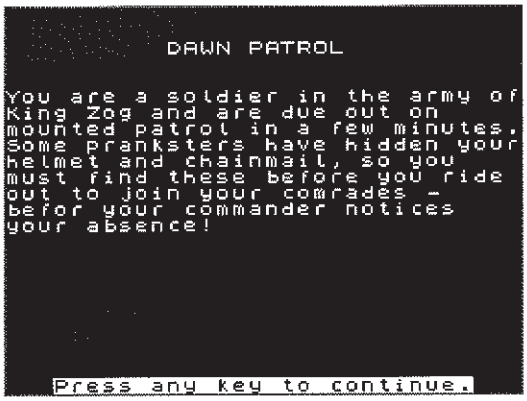


Fig 2. **Dawn Patrol begins again. Just think a little ...**

enter. As there will be no graphics in our adventure, we don't want the verb PICS, so type {D PICS} and press ENTER. Do the same for DESCE and ASCEN. Gilsoft have inserted the word PUT with the same number, and therefore meaning, as DROP. Likewise LOOK is the same as EXAMINE (PAW only uses the first five letters of any word with six or more letters). We will want these two to have their own meaning, so first of all delete PUT and LOOK, then reinsert them as verbs numbers 32 and 33: {I PUT 32 0} and {I LOOK 33 0}. Two new verbs are needed, OPEN and SHUT - we will also insert CLOSE as a synonym for SHUT - verbs 34 and 35: {I OPEN 34 0} {I SHUT 35 0} and {I CLOSE 35 0}. To make life easy, insert the letter 9 as an abbreviation of QUIT, G for GET, and the letters RS for RAMSAVE and RL for RAMLOAD. These would be {I Q 25 0}, {I G 20 0}, {I RS 28 0}, and {I RL 29 0} respectively.

Now to insert first of all, the entries allowing movement into locn 2. To indicate that the door is open, we will use Flag 11. All the "user flags" (11 to 28 and 60 to 255) have the value 0 unless you alter them, so if flag 11 equals 0 the door is closed. When the player types in OPEN DOOR we will SET flag 11, and this will give it the value 255.

From the vocabulary SM, go to MM and select option R, the response table. There are quite a lot of entries already present, mostly dealing with manipulating objects and game saving routines. The first entry we will put in is when the player tries to go south when the door is shut. Type {I S *} and press ENTER then Input the entry {AT 1 ZERO 11 MES 5 MES 0 MES 1 DONE}. Note there is a difference between MES and MESSAGE. MES prints a message after will appear on the same line, whereas a MESSAGE always has a "carriage return" after it and anything else will appear on the next line. So, if at locn 1 the player tries to go south and flag 11 is 0, PAW will respond by printing "You can't. The door is closed".

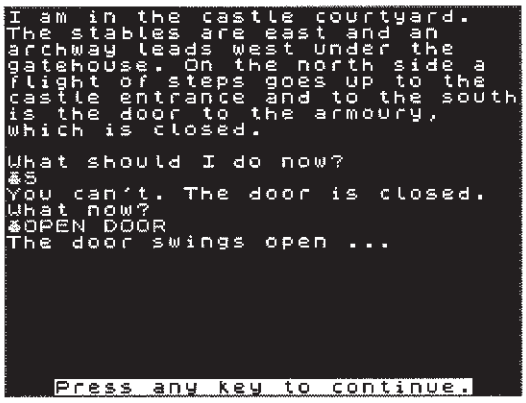


Fig 3. **... And you'll go far. Now try opening the door.**

DARKNESS AND LIGHT

As I said earlier, the armoury will be dark when you enter it, unless you are carrying a source of light. "Darkness" is controlled with flag 0 and if flag is not 0 (i.e. has a value anything other than 0) the location the player is in will be in the dark. We will need two entries in response to allow for this when we move into and out of location 2, and they are:

{I S *} {AT 1 NOTZERO 11 GOTO 2 SET 0 DESC} {I N *} {AT 2 NOTZERO 11 CLEAR 0 GOTO 1 DESC}

Note that SET gives any flag the value of 255, which is the maximum number you can give to any of PAW'S flags. Now for the actual entries required for the player to open the door. You may have realised that this action can be done both in location 1 and 2, and we can allow for this in our entries. Here is the entry you will require

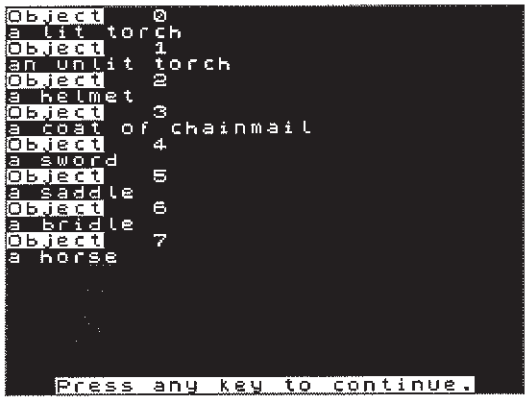


Fig 4. **The Objects Table. All objects must be here.**

plus that for closing the door again:

{I OPEN DOOR} {ATGT 0 ATLT 3 ZERO 11 MES 3 ANYKEY SET 11 DESC}
{I SHUT DOOR} {ATGT 0 ATLT 3 NOTZERO 11 MES 4 ANYKEY CLEAR 11 DESC}

Note that the action CLEAR resets any flag to 0 again. In the above entries we have used ATGT and ATLT, so if the player is AT greater than loon 0 or AT less than loon 3, we can only be at locations 1 or 2. Now, you may have thought that if the door is closed and you are in the armoury, you will not be able to go north. You may have already worked out the entry we will therefore need, which is {I N *} {AT 2 ZERO 11 MES 5 MES 0 MES 1 DONE}.

You will see that I have included a keypress and that the location redescribes when you OPEN or SHUT the door. This is because we want to alter the word that will appear at the end of locns 1 and 2 to indicate the door is open or shut. This is dealt with in process table 1, so switch to this table by typing {Z} {P} {S 1} from the Response SM. As we want to attach messages 1 or 2 to the end of the texts for locns 1 and 2, we will need to insert the appropriate entries so they are scanned by PAW BEFORE the two entries which print the object lists on the screen. Therefore we will insert them using the asterisk symbols, i.e. * * . The four entries are:

{I * *} {AT 1 ZERO 11 MES 1}
{I * *} {AT 1 NOTZERO 11 MES 2}
{I * *} {AT 2 PRESENT 0 ZERO 11 MES 1}
{I * *} {AT 2 PRESENT 0 NOTZERO 11 MES 2}

If, after entering these, you press P, you will see that they have been placed by PAW at the start of Process 1, before the two * entries. You will notice that the entries appertaining to the locn 2 include the condition PRESENT 0, which means that the light source must be present if the message does not appear on screen, and that we have not put a condition DONE or DESC in these entries. This is because we want PAW to continue scanning process 1 after acting upon any of these entries.

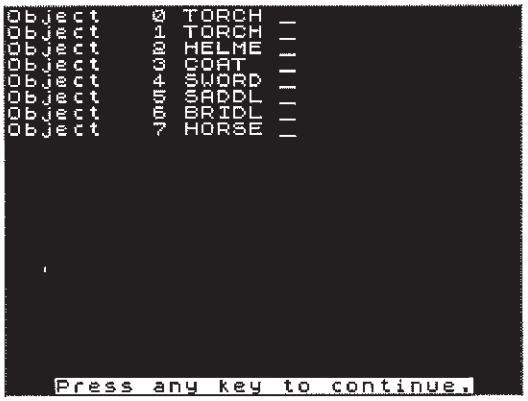


Fig 5. **The Object Word Table.**

We need two more entries in response at this point,the first of which will be print message 6, if the player tries to go west from the archway, and the second will print mesage 7 if the player tries to go north from locn 5:

{I W *} {AT 6 MESSAGE b DONE} {I N *} {AT 5 MESSAGE 7 DONE}

Now test the adventure again and in locn 1 you will see that the text has been completed by the word "CLOSED.", and if you try to go south the appropriate message is printed on screen, as in figure 1. Input OPEN DOOR and after the keypress the word "OPEN." is substituted and you can move south into the armoury. Hang on though! Didn't we set it so that the armoury would be dark when we enter it? How come it's light then? Well, the answer is to type the letter i (for inventory) and press ENTER. Aha! You are carrying a lit pocket torch. This is object 0 and thus is a source of light. PAW has set it to be carried at the start of the adventure. This brings us nicely to the next

section of the adventure - objects. After playing around with the adventure for a while, inputting OPEN, CLOSE or SHUT DOOR, etc, type in Q to quit and follow the prompts back to the MM. Select option 0 for the Object Table and after amending object 0 as below, insert the other seven objects. Note that when amending object 0, you don't need to delete everything then start again. Delete the word (lit), including the space after torch, and the full stop, then use the arrow left key to place the cursor after the "t" of pocket. Delete everything to the left and type {a lit} so that it reads as below, although note that when inserting object texts you should NOT put a blank space at the end of each:

Object 0: a lit torch
Object 1: an unlit torch
Object 2: a helmet
Object 3: a coat of chainmail
Object 4: a sword
Object 5: a saddle
Object 6: a bridle
Object 7: a horse

After inserting these objects we will need to insert nouns in the vocabulary and afterwards amend the Object word, Weight and Initially At tables so that we can manipulate the objects. To save space I have listed the nouns to be inserted and the Object Weight and Initially At tables as they should appear in your database in figure 6.

Nouns	**Object weight tables**	**Object initially at tables**
TORCH 51	Object 0 weighs 1	Object 0 not created
TORCH 51	Object 1 weighs 1	Object 1 at location 5
HELME 52	Object 2 weighs 1	Object 2 not created
COAT 53	Object 3 weighs 1 WR	Object 3 not created
SWORD 54	Object 4 weighs 1	Object 4 at location 2
SADDL 55	Object 5 weighs 1	Object 5 at location 4
BRIDL 56	Object 6 weighs 1	Object 6 at location 4
HORSE 57	Object 7 weighs 20	Object 7 at location 3

Figure 6

Only three of the object weights (table x) need amending. We can wear the helmet and chainmail coat so the attribute for each - the last number amended - needs to be altered to 2 (Wear/ Remove). The two amendments would be: {A 2 1 2} and {A 3 1 2}. As we would not normally be able to pick up a horse, we will be able to amend its weight, the third number, thus: {A 7 20 0}. The Initially At table, option 1, is amended as above but only two numbers are required, e.g. {A 1 5} or {A 3 252}.

SPECIAL LOCATIONS

PAW has four "special locations" where objects are not created (252), worn (253), carried (254) and in the present location (255). The Object Word table (W) needs to be amended using the noun for each object, but of course there are TWO torches, so objects 0 and 1 are given the same nouns. Each object word is amended thus: {A 1 TORCH _} the underline symbol being where you would insert an adjective, for example RED or GREEN, LONG or SHORT, where there are two similar objects of different sizes or colours, etc.

```
Message    0
The door is
Message    1
closed.
Message    2
open.
Message    3
The door swings open ...
Message    4
You close the door, which shuts
with a bang ...
Message    5
You can't.
Message    6
You can't leave the castle on fo
ot!
Message    7
The sentries bar your progress.
"Common soldiers are not allowed
 in the throne room" one of them
 barks at you.

Press any key to continue.
```

Fig 7. **The Message Table.**

If you're wondering why we are not differentiating between the lit and the unlit torch i.e. by inserting the adjectives LIT and UNLIT in the Object word table, this is because only one torch is present in the adventure game at any one time, so there's no need to be specific.

That brings us to the end of this instalment of my PAW mini-adventure programming article. You should now insert a blank tape into your datacorder - or blank, formatted disc into your disc drive - and then select option S from the main menu to save your database. You will be prompted to type in your filename, e.g. DAWN. But if you are using a disc version of PAW use the file name START for one save, then save the database again using a different name, i.e. DAWN. You should always make at least two copies of any database as a precaution. See you next month ... Larry.

Previous article in series (issue 129)
Next article in series (issue 131)

Sinclair User
December 1992

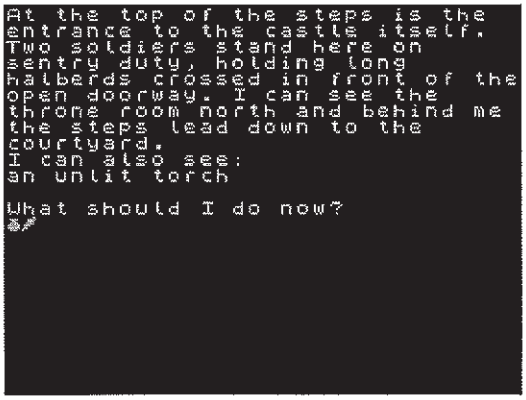
PLAYING AND WRITING ADVENTURE GAMES

***Larry Horsfield* returns with his Adventure writing helmet on again ...**

As mentioned last month, you cannot pick up the horse. To ensure this we must place an entry in process table 1 to limit the amount and weight of objects carried. I will also show you how to place an entry In a certain position in a process table.

Select process table 1 and type {I} The number at the end indicates to PAW that we wish this entry to be the FIRST * * entry in the table. Now enter {ABILITY 10 10}. This tells PAW the player can carry 10 objects with a total weight of ten units. Note that objects worn are NOT included when PAW is checking objects carried but their weight IS.

Now we can test the adventure once again, and this time there are objects in some of the locations which we can manipulate! However, the screen display looks a bit untidy; as In locations where you can see an object, the message "I can also see:" Is printed on a new line with the object underneath as in figure 2. Perhaps it would look neater if the message was printed after the full stop of the location text with the object listed after. Easily done!



You should get hold of the torch, that's what you should do next matey. But what next?

First of all quit the test adventure by selecting option N, the system messages texts. Amend system message (sysmess) 1 by deleting the colon (:) at the end but leaving the blank space. Also amend sysmess 48 by pressing DELETE to get rid of the "carriage return" then press SPACE to put a blank space after the full stop. Next go to process table 1 and amend the first * * entry - which we inserted as above - as follows: {ABILITY 10 10 LET 53 64}. When flag 53 is given the value 64, all object lists are printed as sentences.

Next amend the first of the two existing "* _" entries, which determine how the object lists are displayed, to delete the NEWLINE condition, then insert a new * entry BETWEEN the second and third entries that just contain the condition NEWLINE i.e. {I * _ 2}

{NEWLINE}. If you now test the adventure you will see that the screen display looks much neater.

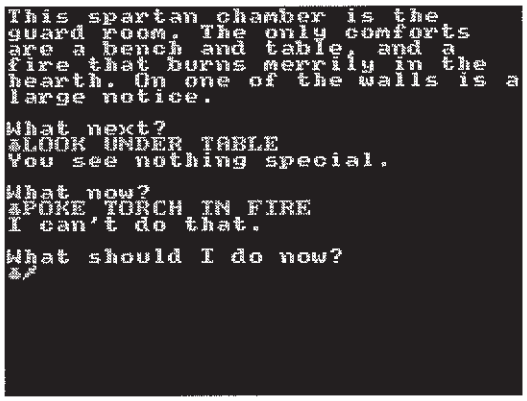
However if you were to GET the TORCH and then type I to display your inventory, you will see that the object is still on the next line underneath the message "I have with me". If you switch to the response table and press P to screenprint the table, you will see the two entries "I" and "GET I" which both use the INVEN action. This will always print the object lists in a column, so we must use another method to print our inventory. First of all, return to the System Messages table, option N, and amend SYSMESS 10 to read {I am wearing }. Now switch to the response table and amend both the I and GET I entries so that instead of INVEN, each has the following:

{SYSMESS 9 LISTAT 254 NEWLINE SYSMESS 10 LISTAT 253 NEWLINE DONE}

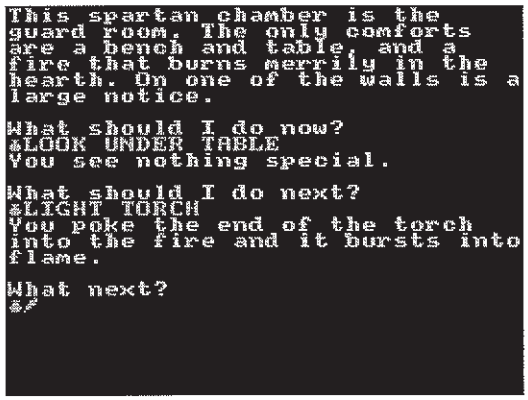
Actually, the GET I entry could be deleted from the database, as very few adventurers use that input these days.

If you now test the adventure again and, after picking up a few objects, press I, the display looks even better, as in figure 3. You may however have noticed that if you continue to press, for example, I to display your inventory, the location text will scroll off the top of the screen. Some adventure players prefer this, but most like the location text to remain displayed while the inputs scroll underneath. This can be easily set with the MODE condition.

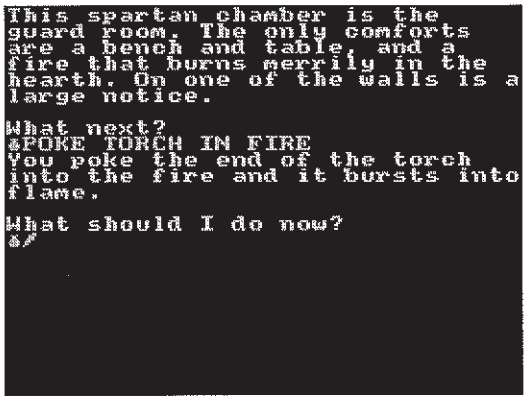
Go back to process table 1 and amend the first * * entry, adding the line {MODE 4 1 }, so that the whole entry reads: ABILITY 10 10 LET 53 64 MODE 4 1. You must also amend the third * entry so it reads NEWLINE. If you now test the adventure and make several "I" inputs you will see them scroll up under the location text.



Always cover all possibilities. Are 'Poke torch' and 'Light' the same?



Now you can see under the table.



And now 'Poke' works too. Hurrah!

Another thing we can do to improve the look of the adventure is to change the standard Spectrum character set that is resident in PAW for one that is more pleasing to the eye. On the B side of the PAW cassette (or the A side of the disk) there are 22 different character sets, any 5 of which can be loaded into a database. We will load character set 8 into our mini-adventure.

Select option Q, characters, from the MM and, if you press P from the SM, you will see that only the UDGs (User Definable Graphics) and the shades are present. To be able to load a character set in we must first insert a "blank" set of standard Spectrum characters. To do this press I then ENTER. If you press P you'll see the set is displayed. Now, from the SM, type L 1 and then, when asked for the name of the file, type SET8 (note the absence of a space) and press ENTER. If you have a disk version of PAW, the set will load from the disk, but tape users will have to wind the tape on to the

correct position. Loading the set only takes a short while and, once loaded you can view the set by pressing P.

Having done that, go to the MM and select option B, the background colours. We do not intend to alter the border, paper and Ink colours, (black, black and white) so type 0, 0 and 7 when prompted.

When prompted for the character set number, type 1. The whole database will now use the newly loaded character set which I'm sure you will agree, looks a whole lot better.

Right then, having improved the look of our adventure, let's put in some gameplay. As the plot says, your helmet and chain mail are hidden somewhere, which is why in the Initially At table they are not created. To find these we must SEARCH or LOOK for them. We will imagine that your helmet has been hidden under the bench in the guard room and your chainmail in the straw in the stable.

We have to SEARCH the STRAW and LOOK UNDER the BENCH to find the objects. Also, in order to see in the armoury we will need to be able to light the torch, which you will do in the guard room (from the fire!). Go to the vocabulary table and insert the following verbs and nouns with the numbers as shown:

Verbs: SEARC 36 LIGHT 37 EXTIN 38
Nouns: BENCH 58 STRAW 59 FIRE 60

Now go to the message table and insert the following messages:

Message 8: {What's this you've found under the bench? It's your helmet, which you pick up.}
Message 9: {You search the straw revealing an object.....}
Message 10: {You poke the end of the torch into the fire and it bursts into flame.}
Message 11: {You find nothing of any use.}
Message 12: {You see nothing special.}

Go to the response table and insert the following entries:

```
LOOK BENCH     PREP UNDER
               AT 7
               ZERO 12
               MESSAGE 8
               PLACE 2 254
               SET 12
               DONE

SEARCH STRAW   AT 3
               ZERO 13
               MES 9
               ANYKEY
               CREATE 3
               SET 13
               DESC
```

These two entries show two ways of revealing hidden objects. Note the use of the flags to prevent the entries being implemented again if the inputs are repeated by the player. If the player did repeat the input SEARCH STRAW, the response would be 'I can't do that.' which isn't very good. So, insert the following two entries in response to give more positive responses, and also amend the EXAMI _ entry as below:

```
SEARC _ MESSAGE 11
        DONE
LOOK _  MESSAGE 12
        DONE
EXAMI _ MESSAGE 12
        DONE
```

The use of the underline symbol will make these the last entries using the verbs SEARC, LOOK and EXAMI. You could also, as many adventure authors do, abbreviate the word EXAMI to EXAM and even, as I do, just 'X'. Now to deal with the torch. We will need two entries in response to cover lighting and extinguishing the torch, and we will need two process tables as well. First of all, though we need to insert a few more messages, thus:

Message 13: {The torch is }
Message 14: {unlit. }
Message 15: {lit. }

Now go to the Process table 2 and "begin" two new tables, 3 and 4. Switch back to response and insert the following two entries:

```
EXTIN TORCH PROCESS 4
            DONE
LIGHT TORCH AT 7
            PROCESS 3
            DONE
```

Now select process table 3 and insert the following entries:

```
* * CARRIED 0     * * CARRIED 1
    MES 13           MESSAGE 10
    MESSAGE 15       SWAP 1 0
    DONE             DONE

_ _ SYSMESS 8
    DONE
```

The entries needed for process 4, in which we will extinguish the torch:

```
* * CARRIED 0     * * CARRIED 0
    MES 13           ZERO 0
    MESSAGE 14       SWAP 1 0
    DONE             OK
```

```
* * CARRIED 0      _ _ SYSMESS 8
    NOTZERO 0         DONE
    SYSMESS 15
    PAUSE 75
    SWAP 0 1
    DESC
```

There are two entries which deal with extinguishing the torch because if you put it out in a bright situation, there is no need to redescribe the location, whereas in a light location we need to show it is now dark. The SYSMESS 8 entries in each table print "You can't do that." if you are not carrying either torch. After inserting all the above entries, test the adventure again and, after picking up the torch, input LIGHT TORCH and EXTIN TORCH in various locations and both carrying and not carrying both torches. Now when you enter the armoury, you will see and be able to pick up the sword.

Now, the player of this adventure may try a different input to light the fire, for example POKE TORCH IN FIRE, as suggested by the message that appears on the screen when you LIGHT THE TORCH, and we should try to anticipate such inputs from the player.

Insert the verb POKE as verb number 39 in the vocabulary, giving it the synonyms INSER(t)and HOLD if you wish (and any other words you think might have the same meaning). Switch to the response table and insert the following entry:

```
POKE TORCH PREP IN
NOUN2 FIRE
AT 7
PROCESS 3
DONE
```

While we are in the guard room, you'll see there is a notice on the wall. We will use this to illustrate another PAW technique that, in a normal full size adventure, can save you memory. Regarding the notice there are two things you can do with it, i.e. EXAMINE IT or READ IT. Now in some adventures you might find that the author has used a different message for the response to each of these inputs. Insert the following message:

```
Message 16 The notice on the wall reads:
NO SPITTING IN THE FIRE!
```

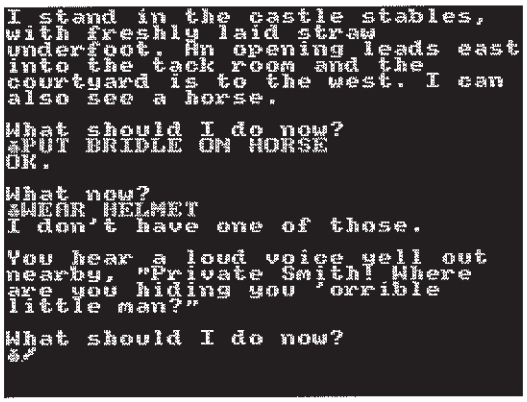
Now insert the verb READ in the vocabulary, verb number 40 and the noun NOTIC, noun number 61. Switch to the response table and insert the following two entries:

```
EXAMI NOTICE LET 33 40

READ NOTICE AT 7
             MESSAGE 16
             DONE
```

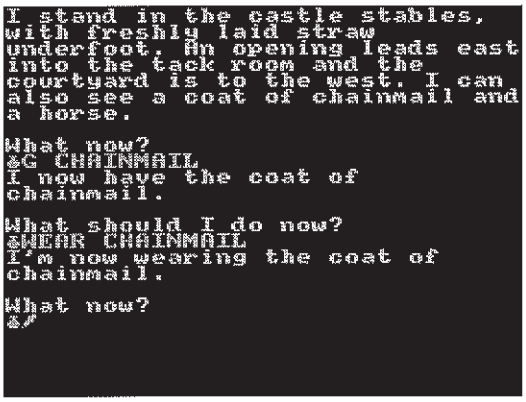
The first entry tells PAW to make flag 33, which holds the verb number of the players last input, change to value 40, which of course is the number for the verb READ. As long as there is an entry READ NOTICE, it will be acted upon. You can also do the same for any of the words in an entry

using flags 34 to 45. See the PAW manual for which words they refer to. You should note, however, that when using such entries in your database you can only alter words with a larger value than the first one. For example, you can alter words EXAMI to READ, 30 to 40, but you cannot alter SEARC to LOOK, 36 to 33.

If you now test the adventure you can do everything in the game except put the saddle and bridle on the horse and ride it.



Sounds like trouble. Is time running out for our hero? Get the rest of the equipment.



Now you're almost sorted. The final part of the adventure is next month. Join me then.

That brings me to the end of this instalment of my PAW mini-adventure programming article. You should now insert a blank tape into your datacorder or a blank, formatted disc into your disc drive and then select option M from the main menu to save your database. If you are using a disc version of PAW use the filename START for one save, then save the database again using a different name i.e. DAWN. you should always make at least two copies of any database as a precaution. See you next month.

Previous article in series (issue 130)
Next article in series (issue 132)

Sinclair User
January 1993

PLAYING AND WRITING ADVENTURE GAMES

This is the final part of *Larry Horsfield's* PAW adventure writing tutorial. Remember, you'll need a full version of PAW to follow the instructions Larry has given in the last four months, but if you're serious about writing adventures then it's money very well spent. Meanwhile why not let Larry tell you how to saddle and mount your horse ...

Last month we ended with just two problems left. How to saddle and bridle the horse. We will start with the saddle and there are two inputs the player can use to implement this action. Hold on though, surely the word saddle is a noun? Not to worry, there is an easy way around this, and that is to insert a "null word" as the verb. The two response table entries for the saddle, using flag 14, plus the entry for putting the bridle on the horse (flag 15), which is similar to the saddle, are thus:

```
PUT SADDLE PREP ON
           NOUN 2 HORSE
           CARRIED 5
           PRESENT 7 DESTROY 5
           SET 14
           OK
_ SADDL    NOUN2 HORSE
           CARRIED 5
           PRESENT 7
           DESTROY 5
           SET 14
           OK
PUT BRIDL  PREP ON
           NOUN 2 HORSE
           CARRIED 6
           PRESENT 7
           DESTROY 6
           SET 15
           OK
```

Now we come to the entry that will, provided all the gameplay has been completed, mark the final

input in our mini-adventure, i.e. RIDE HORSE. First of all, insert the verb RIDE into the vocabulary as verb 41. YOU could also insert the verb mount as a synonym. Now go to the process tables and begin TWO new tables, 5 and 6. Before we insert any entries in Response or Process, switch to the message texts and insert the following messages:

Message 17: {The horse has no }
Message 18: saddle.
Message 19: bridle.
Message 20: You don't have a sword.
Message 21: You are not wearing the _.
Message 22: You mount the horse and ride out of the castle to join your comrades on the dawn patrol ...

WELL DONE!
You have finished the adventure!

You should position 'WELL DONE' in message 22 so it is in the centre of the screen, and you should put a blank line at the bottom using the ESCC7 routine. I have put brackets around message 17 as there is a space at the end and you'll notice that I've used the underline symbol in message 21. This will print as the object name in place of the "_" when combined with certain contacts in the respective entries. O.K., now go to the response table and insert the following entry:

```
RIDE HORSE AT 3
PROCESS 5
DONE
```

Now switch to the response tables and select Process 5. We will handle the conditions that determine whether the player can RIDE the HORSE by inserting entries in Process 5, checking to see if the player is wearing the helmet and chainmail, not carrying the sword and if the saddle and bridle are not on the horse. The checking entries in Process 5 are thus:

```
* * NOTWORN 2      * * NOTWORN 3
    WHATO             WHATO
    LET 51 2          LET 51 3
    MESSAGE 21        MESSAGE 21
    DONE              DONE

* * NOTCARR 4      * * ZERO 14
    MESSAGE 20        MES 17
    DONE              MESSAGE 18
                      DONE

* * ZERO 15
    MES 17
    MESSAGE 19
    DONE
```

The WHATO action, "WHAT Object", is explained in the PAW manual, but in this case the LET 51 2, referring to object 2 the helmet (or 3 the coat of chainmail), indicates to PAW that that number is being referred to and the description of that object number is printed in the message in place of the "_" character. If you Print the system messages, you will see that quite a lot of them use this facility.

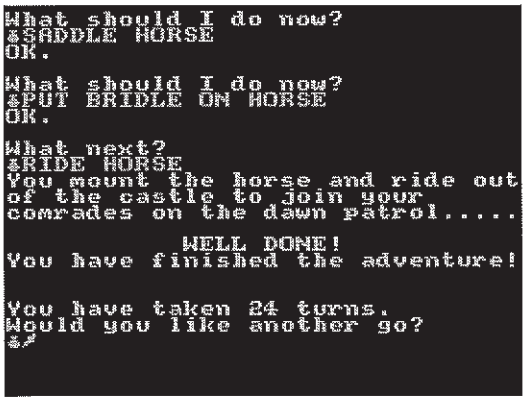
The final entry to be inserted in Process 5 is the one which is acted upon when all the player is wearing is the helmet and chainmail, carrying the sword and the saddle and bridle are on the horse. This entry is thus:

```
* * MESSAGE 22
    SET 16
    TURNS
    END
```

We have not put any conditions in this entry because the previous entries have checked for the objects being either worn or carried and the saddle and bridle being on the horse. I have included the SET 16 line because of some of the entries we will be putting in later. If the player is wearing the helmet and chainmail, carrying the sword and the saddle and the bridle are on the horse, when RIDE HORSE or MOUNT HORSE is typed in, message 22 is printed on the screen, the number of turns you have taken is shown and you are asked if you want to play again. The 'full solution' for the adventure, using the minimum number of inputs is, by the way, as follows:

U, GET TORCH, D, W, N, LOOK UNDER BENCH, WEAR HELMET, LIGHT TORCH, S, E, OPEN DOOR, S, GET SWORD, N, E, E, GET ALL, W, SEARCH STRAW, GET CHAINMAIL AND WEAR IT, SADDLE HORSE, PUT BRIDLE ON HORSE, RIDE HORSE ... Adventure complete.

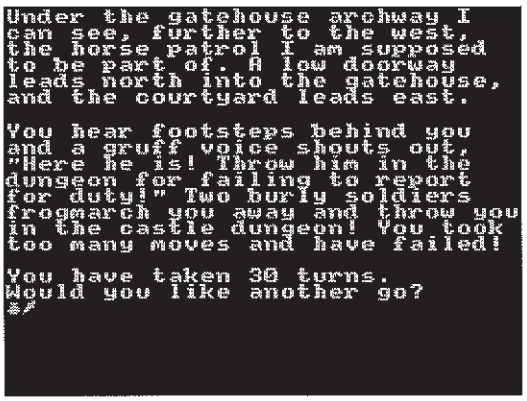
That, however, isn't the end of the programming. You may have realised that firstly we inserted another process table which we haven't used yet and that we haven't made any use at all of process table 2, which I explained was PAW's 'turn' at the adventure. There also isn't any rush to complete the adventure, although the storyline mentions that if you haven't joined your comrades, your commander will notice your absence. We will therefore use this part of the plot to provide a bit of excitement by having your commander come looking for you if you haven't completed the adventure within 30 moves. First of all, insert the two following messages:



Well done, you've completed the adventure and all is well. Now invent your own one or expand Dawn Patrol.

MESSAGE 23: You hear a loud voice yell out nearby, "Private Smith! where are you hiding you 'orrible little man?"

MESSAGE 24: You hear footsteps behind you and a gruff voice shouts out, "Here he is! Throw him in the dungeon for failing to report for duty!" Two burly soldiers frogmarch you away and throw you in the castle dungeon! You took too many moves and have failed!



The fail message in Dawn Patrol. This adds an element of urgency to the game, players

You should start each message by inserting a blank line at the top using the ESCC7 routine (i.e. Extend mode - 7 - Delete), and you should also put a blank line at the end of message 24. This is to separate the messages when on screen. After entering those messages, go to the process table and, in process2, insert this short entry:{* *} {PROCESS 6} Now select process

6 and insert the following entries:

won't get away with being lazy.

```
* * ZERO 16   * * ZERO 16
   ZERO 5        EQ 5 10
   LET 5 31      MESSAGE 23

              * * ZERO 16
                  EQ 51
                  MESSAGE 24
                  TURNS
                  END
```

In addition to the ordinary User Flags (11 to 28 & 60 to 255) there are 9 "auto-decrement" flags (2 to 10) which are decremented if non-zero by PAW. Some of these as explained in the manual, operate under certain conditions, but flags 5 to 8 just decrement every input by the player or PAW. In the last entry you have to make a "death" message appear when the flag you use equals 1, and hence add 1 to the number of moves when inserting the first entry so that the sequence will work properly. Once all the above has been inserted into your database, test the adventure and if you make some meaningless moves you will see the first message appear after 20 moves, then the final "death" message 10 moves later.

WHERE DO WE GO FROM HERE?

So far I have given you the instructions, the rest is up to you! You can expand 'DAWN PATROL' by inserting 'EXAMINE' messages and response table entries for all objects in the adventure. How about putting in more locations around the castle and maybe a puzzle or two of your own devising, it's all up to you.

If you are serious about adventure writing, I advise you to start by writing a small adventure, about twice the size of 'DAWN PATROL', and then work your way upwards to a full size adventure.

The average 48K adventure should be about 40 - 50 locations. There should be an evenly balanced number of puzzles ranging in difficulty from dead easy to quite hard.

If you are working in 128K, the game can of course be much bigger, but don't fall into the trap of putting in loads of 'walk through' locations. (If you are a 'humorous' adventure writer, watch out! Most adventure players are adults and do not appreciate 'modern' humour, nor do they like smuttiness or bad language, a lot of players are women).

OK, so you've written your adventure, now what? Well, if you're interested in a serious attempt at an adventure, you should have the game 'playtested'. This involves giving the game to someone who plays through the adventure, trying to solve the puzzles in various ways and checking for spelling and grammatical errors. You should give the game to a couple of playtesters first, then, once they have found bugs you can correct them. After this give the game to another two friends to playtest again, they are bound to find bugs the others missed.

MARKETING YOUR ADVENTURE:

At last you have a finished, playtested and debugged, adventure. The big question now is how do you sell your game? The hard way to do it is on your own, and you should really sit down and consider what this option involves. It means buying blank cassettes, jiffy bags, stamps, paying for

photocopies etc. not to mention hours spent loading and test playing copies of your adventure.

The alternative is to submit your finished adventure to one of the established adventure software houses, such as Zenobi software, FSF Adventure or The Guild. If you do this, don't just send a badly labelled cassette and a note saying 'this is my adventure, will you sell it?' as has happened to me a couple of times. Give whoever you send it to full details of the plot, also provide a map and a full solution.

The software house will evaluate your offering and tell you whether or not they are interested in publishing it under their label. Don't feel too dejected if your adventure isn't accepted, ask why it was turned down and what could be done to improve it. If your adventure is good enough the software house will accept it and you will probably be asked to sign a contract regarding the royalties the software company will pay you.

Whatever you do, you are not going to become an overnight millionaire. The adventure side of computer games is an ever decreasing section of the market and income from even the best selling adventures is quite small compared to the vast amounts of money that can be earned from arcade games. Most adventure houses will pay you 30-40p per adventure sold, usually paid quarterly.

So, why do we do it? Well most of us are doing it because we love adventure games, and the money side of things is an added bonus. The only exception to this among the current Spectrum adventure software companies is Zenobi Software, better known as John Wilson, who has made a business out of adventure publishing which is his sole source of income. This is a success story which is unparalleled in the world of adventure writing,with over 100 titles currently available.

I hope my articles over the lost few months have been of interest to you, and my tutorials have been easy to follow. I am willing to evaluate your games and help you out if you have any problems. Please give full details of your game and send a full copy of your database on tape, 3" +3 disk or 3.5" +D disk. Please note, however, that I cannot help you with the plot of your adventure!

LARRY HORSFIELD,
FSF ADVENTURES,
LONDON.

Other useful addresses:

Zenobi Software,
Rochdale.

The Guild,
Birmingham.

Compass Software,
Great Yarmouth.

That brings me to the end of the last instalment of my PAW mini-adventure programming tutorial. You should now insert a blank tape into your datacorder, blank, formatted disc into your disc drive or whatever disk you've been using up until now and then select option M from the main menu to save your database. If you are using a disc version of PAW use the filename START for one save, then save the database again using a different name i.e. DAWN. You should always make at least two copies of any database as a precaution. Good luck with your new adventure writing hobby.

By Larry Horsfield

The Tipshop

A central archive for all Spectrum and SAM games hints, tips, cheats, maps, hacks and pokes.

HOME | 0123 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Search: For: Anything at all

Exact: Check Aliases: GO

Professional Adventure Writer

Gilsoft International

Download from World Of Spectrum

Number of entries found: 6

Tips

CONTAINERS

First put your container's name in nouns with a value of 50 or higher so it will be recognised

as 'it'. Put in the object text the name of your container and note down its number. Under Object Weights remember to put it as a container. Under Words put its noun and adjective (if it
has one).

Insert the Location with the same number as the object and leave it blank, ie object 7 - Location 7. Put the following into response:

I PUT -0
PREP IN NOUN2 (container)
PRESENT (container number)
AUTOP (location of inside of
container) DONE

For example,
PREP IN NOUN2 BAG PRESENT7 AUTOP7 DONE
I GET -0
PREP OUT NOUN2 BAG PRESENT7 AUTOP7 DONE
I LOOK BAG
PREP IN MESSAGE2 LISTAT7
DONE

where message 2 is 'In the bag is:'

INTRODUCTION SCREEN
To make a tune play as soon as the game has loaded go to Process1 and:

```
I**
AT 0 BEEP 10 150 BEEP 10
100 BEEP 1050 BEEP 10
100 BEEP 10 150 PAUSE
15 BEEP 10100 BEEP 10
100 (or your own little
tune).
```

Also, to make your game look more professional key this into Process1:

```
I **
AT 0 ANYKEY GOTO 1
DESC
```

GRAPHICS

Remember always to select default colours for the pictures - otherwise they are treated as subroutines. When drawing subroutines always remember to remove the p lots that you use. Otherwise they will be only drawn at the same spot, ie if you plot a subroutine at 197 56 you

cannot put it anywhere else on a personal picture - only at 197 56.

MONEY
If you want a currency in your game choose a flag to hold the number of coins carried and insert two messages: 'I am carrying' and 'coins', so that if flag 100 held the number of coins and the messages were 1 and 2 a condact could read

```
MESSAGE1 PRINT 100
MESSAGE2 DONE.
```

I hope these notes help some budding adventure-writers.

(Stephen Flint in Crash issue 46 , Page 107) 6974

Adventure-writing utilities are terrific programs, but the wonderful things they can do aren't always that easy to discover - not without a lot of head-scratching, late nights, muttering and mistake-making along the way. It's a lot more simple if you've got an expert to help you, someone who's good enough to share their own secrets with you. Such a wonderfully warm and kind-hearted human bean is Mark Cantrell, author of that marvellous game, Diablo!

Mark's sent me the following article explaining how you can have more than one main character in your PAW'd adventure, and switch control of the game between them. So over to Mark...

he following routines will allow an adventure to use a number of characters

which the player can switch between at any point in the game, as seen in commercial releases like Runestone and Terrors Of Trantoss. When not controlled by the player, such a character could become a normal PSI (Pseudo-Intelligent Character).

This first routine will simply show you how to switch between two characters, John and Helen. In this example, the PSIs are entered in the object table as objects - John is the first entry, Helen the second. Their nouns should be entered below 50 in the noun table, and also entered in the object words table.

```
RESPONSE:
CONTROL JOHN
EQ 13 10; check controlling John already?
SYSMESS 87; then say so
DONE; finish the command
CONTROL JOHN
NOTZERO 14; is he dead?
SYSMESS 88; then say so
DONE
CONTROL JOHN
PROCESS 6; call primary exchange routine
DONE
CONTROL HELEN
EQ 1320; controlling her already?
SYSMESS 87; then say so
DONE
CONTROL HELEN
NOTZERO 15; is she dead?
SYSMESS 89; then say so
DONE
CONTROL HELEN
PROCESS 6; switch to primary exchange routine
DONE; finish response section
CONTROL -
SYSMESS 90 DONE; prints "Control who?" if noun is not HELEN or JOHN
```

The above section of the routine establishes the player's ability to change characters. Each character will need the three lines. The first checks that the player is not already controlling that character, and if he is then a message is printed to say so (message 87). The second line checks that the particular character is alive, and if they're not then the player will be told he cannot take control because that character is dead (messages 88/89). The third line calls the first section of the exchange routine... of which more in a mo.

The last line merely asks the player which character he wishes to assume if he has failed to specify one, mis-typed the name or typed the name of a character they cannot control. Since characters can die in most adventures and since with this routine the game need not come to an end until all the player characters are dead, then the player must not be able to enter commands when in control of a dead character. Of course in a small example such as this, with just two characters, the game can simply put the player in control of the surviving PSI. When using more than two characters the following lines will be needed...

RESPONSE:

* *0
EQ 13 10; in control of John?
NOTZERO 14; is he dead?
NOTEQ 33 x; verb is not CONTROL
SYSMESS 88; "He is dead."
DONE; exit the table

* *1
EQ 1320; controlling Helen?
NOTZERO 15; is she dead?
NOTEQ 33 x; verb is not CONTROL
SYSMESS 89; "She is dead."
DONE; exit the table

These entries intercept any command made which is not CONTROL. PAWS will still be taking its turn at the game, which means that while you are dead game time will still be passing for other characters. Once the character is dead you may want to move it into an empty room, otherwise you'll find the game printing reports of a character's movements to you when it's supposed to be six feet under!

FLAGS:
11: John's room
12: Helen's room
13: character indicator
14: John dead?
15: Helen dead?
61: indicate changing characters
62: dump/collect worn or carried objects

SYSTEM MESSAGES:
87: "You are already controlling that person!"
88: "John is dead."
89: "Helen is dead."
90: "Control who?"

ROOMS:
4: John's carried objects store
5: John's worn object store
6: Helen's carried object store
7: Helen's worn object store

The next section of the routine, the primary exchange section, begins the process of changing characters. Each character must have one line associated with it for each of the other characters. So, for example, if the game has four characters then there must be three of these lines per character...

PROCESS SIX:
CONTROL JOHN EQ 1320; controlling Helen?
GOTO 6

PROCESS 7; dump Helen's carried objects
GOTO 7
SET 62; set to worn objects
PROCESS 7; dump Helen's worn objects
SET 61; indicate changing character
LET 16 10; set control to John
CLEAR 62; reset to carried objects
CONTROL HELEN
EQ 1320; controlling John?
GOTO 4
PROCESS 7; dump John's carried objects
GOTO 5
SET 62
PROCESS 7; dump John's worn objects
SET 61; indicate changing character
LET 1320; change to Helen
CLEAR 62; reset to carried objects

The above lines form the primary exchange routine. The routine checks which character the player is controlling by using the EQ 13 'condact'. The routine must determine which character is being controlled so that the objects can be properly dealt with, hence there being one entry for each of the other characters, as mentioned.

Having determined which character is currently controlled, the line then places the player in that character's carried object store. It then calls a sub-routine to dump all the objects in that location. The routine then sends the player to that character's worn object store. Flag 62 is set to indicate that worn objects must be dumped or collected, depending on which part of the exchange routine called it, then the dumping routine is called once more.

Now Flag 61 is set, which will ensure that the secondary exchange routine is active in Process Two and also tell the dumping routine to collect objects instead. Flag 13 is then set to 20 to indicate that the new character has been selected, and Flag 62 is cleared to indicate that the second part of the routine must collect objects that should be carried.

PROCESS SEVEN:
* *

ZERO 62; move carried objects?
ZERO 61; dumping objects?
DOALL 254; scan for carried objects

ZERO 62; as above
ZERO 61; as above
PUTO 255; place objects in current location
DONE; repeat or end loop
* *

NOTZERO 62; move worn objects?
ZERO 61; as above
DOALL 253; scan for worn objects

* *

NOTZERO 62; as above
ZERO 61; as above
PUTO 255; put all objects in current location
DONE; repeat or end loop

The above lines deal with dumping objects and this is the section which is active when called from the primary exchange routine. The lines lcok at which objects are being dumped, either carried or worn (ie Flag 62 set or clear), and establishes a DOALL loop in the relevant location - room 254 for the player's carried objects and 253 for the player's worn objects.

The next line then processes the objects discovered by the DOALL loop, checks for the correct type (ie carried or worn) and then places the objects that have been found into the current location (255) which would be the relevant object store. The DONE 'condact' continues the loop or exits it if no other objects are found.

PROCESS SEVEN:

ZERO 62; move carried objects?
NOTZERO 61; collecting objects?
DOALL 255; scan for objects in location

ZERO 62; as above
NOTZERO 61; as above
PUTO 254; place objects in carried room
DONE; finish loop

NOTZERO 62; move worn objects?
NOTZERO 61; as above
DOALL 255; scan for objects in location

NOTZERO 62; as above
NOTZERO 61; as above
PUTO 253; place objects in 'worn' room
DONE; finish loop

The next part of the routine deals with collecting objects from the newly assumed character's store rooms and placing them in the player's inventory rooms (253 and 254). It works in the same way as the first four lines, but the DOALL loop looks for objects in the player's position, the object store rooms, and places them in the inventory These lines could be separated into two sub-routines, but bear in mind that each process table has an overhead of 12 bytes. The secondary exchange routine is entered in Process Two...

PROCESS TWO:
* *

EQ 13 10; now controlling John?
NOTZERO 61; collecting objects?

GOTO 4
PROCESS 7; collect carried objects
GOTO 5
SET 62; indicate worn objects
PROCESS 7; move worn objects
CLEAR 62
CLEAR 61; reset exchange flags
COPYFF 1138; put player in new location
DESTROY 1; remove character object
DESC; describes the new location
* *

EQ 1320 NOTZERO 61 GOTO 6
PROCESS 7 GOTO 7 SET 62 PROCESS 7
CLEAR 62 CLEAR 61COPYFF 1238
DESTROY 2 DESC; as above but for Helen
JOHN
* *

EQ 13 10; controlling John?
ZERO 14; is he alive?
COPYFF 38 11; update PSI's position flag
* *

EQ 1320 ZERO 15 COPYFF 38 12; as above but for Helen

The first two lines of this routine take care of collecting the character's objects and placing them into the player's inventory. It is essentially the opposite process to dumping the old character's objects. It then places the player in the location previously occupied by the new character as a PSI and removes the character as an object. It then describes the location.

The next two lines maintain the position of the character by updating the character's location flag so that it may be placed at its location correctly when the player assumes another character.

Finally, Process One must have a couple of lines to move the PSIs and keep the object in the same place that its location flag indicates...

PROCESS ONE:
* *

NOTEQ 13 10; not controlling John?
ZERO 14; is he alive?
COPYFO 11 1; place object at specified location
* *

NOTEQ 1320 ZERO 15 COPYFO 122; as above but for Helen

This routine is merely the bare bones. You may want to provide your characters with some attributes, such as morale, strength, stamina etc, in which case each PSI could have its own store flags to hold these values and be exchanged between such flags and some work flags whenever the player assumed control of another

character. In that way, one set of processes could be used upon the attribute flags for all characters when under player control rather than a separate process for each.

(Mark Cantrell in Your Sinclair issue 54 , Page 63) 8751

Mark says that in the PAWS manual you're advised to save the various overlays onto a short tape, but it doesn't tell you the start address of the overlays. This doesn't matter to the average user, but those with a bit of extra technical know-how might welcome theinfo.

"Welcome, info," they will say. So here are the results of Mark's investigations.

All overlays start at 29632, but the differing lengths and end addresses are...

PAWOVR 1
Length 4585 End 34217
PAWOVR 2
Length 1186 End 30818
PAWOVR 3
Length 3679 End 33311
PAWOVR 4
Length 2875 End 32507
PAWOVR 5
Length 2147
End 31779
PAWOVR H
Length 955
End 30587

Mark says the files can then be loaded into the old Speccy using the LOAD "" CODE command, and saved out again as, for example SAVE "PAWOVR 1" CODE 29632, 4585.

(Mark Walker in Your Sinclair issue 64 , Page 60) 8815

(This was the last tip printed in Mike Gerrard's era of the YS Adventures section. Sniff. GS)

PAWS makes life easier in dealing with objects by the way it uses condacts (conditional actions) which automatically put objects into containers, or remove them, However, you still have to arrange the containers yourself and see if they are present or not.

Each container must have an empty room of the same reference number as its object text, to use as its inside'. So, in effect, when you're placing an object inside another object which you've specified as a container, what you're actually doing is placing it in an empty room somewhere in the adventure map - but a room that has no paths into it or exits out.

One suggestion is to arrange all your containers in the object table so that they number from one onwards. The actual rooms that the player can enter - the

ordinary locations - can then be numbered starting one higher than the last container number. All the previous rooms therefore form the insides of all your containers - bags, sacks, pockets or whatever you want. Object zero is, of course, the source of light, and room zero is your title page.

An object is specified as a container in the object attributes table (X) - the table where you also specify whether an object can be worn, and its weight. Each object must have attributes entered for it. These are its weight (I use a weight of zero to indicate that an object cannot be carried, e.g. an elephant!) and its status. This can be either no further attribute (zero), a container (one), a wearable item (two) or a container that can also be worn, like a rucksack (three).

According to the PAWS technical guide, if you give a container a weight of zero then the sum weight of everything in that container will not be taken into account when the program determines how much weight the player is carrying in total. This way, you can carry an unlimited amount of weight. You don't have to do this, it's just one possibility for you. Of course if you do do it, and want to use a particular weight to specify an object that cannot be carried, then you'll have to use a weight other than zero.

Putting Objects into the Container

Because PAWS uses special condacts to take the work out of moving objects in and out of containers, these commands are quite easy. For example, if we have a bag as object number one, then in the attributes table we enter its weight and an attribute of one (to indicate a container, as shown above). We then enter condacts as follows in the response table...

PUT NOUN2 BAG; Specify the bag?
PRESENT 1; Is the bag present?
AUTOP 1; Put object in bag
DONE; Finish command

PUT NOUN2 BAG; As above
ABSENT1; No bag around?
SYSMESS 26; Tell player 'There isn't one!"
DONE; Finish command

PUT SYSMESS 60; "Put it where?"
DONE; Finish command

PUT ALL NOUN2 BAG As above
PRESENT 1; Is the bag here?
DOALL 254; Process all carried objects
DONE; Finish command

These example lines show how easy it is to make use of containers. The first line checks the LS (Logical Sentence) beyond the simple verb/noun to see which container the player means. It your game only has one container, then you wouldn't need this condact. If you have more than one bag then you can

distinguish them by using adjectives (green bag, large bag, leather bag etc). The line then checks that the appropriate bag is present, and the AUTOP condact AUTOmatically Processes the specified object. If the object is being carried then it's put inside the bag. Otherwise, the condact reports the status of the object (you're not carrying it, it isn't here, you're wearing it etc). DONE then finishes off that little command, so that the program doesn't automatically go on to the next in line. Don't forget your DONEs!

The next line processes the command if the bag is absent, letting the player know. The SYSMESS (SYStem MESSage) can of course be any number (you don't have to use 26 for "There isn't one!"), just as long as you specify the right number for that message (a bit obvious for experienced users I know, but something novices need to be reminded of).

The next line would follow all the extended LS commands, and is used to ask the player where they wish to put an object if they haven't specified a container. Alternatively, you can just arrange for the object to be dropped, to let the player know he forgot to say IN THE BAG.

The last line obviously performs the PUT ALL option. Again it checks which container the player is referring to, and that it is actually there. It then initiates a DOALL loop which looks for carried objects and processes the following lines in turn for each object it finds. Although it's listed last here, in PAWS it's among the first lines to be scanned for this command. You must check that the bag is present because if you don't you'll get 'There isn't one!" printed onscreen for each object, instead of just the once.

(Mark Cantrell in Your Sinclair issue 69 , Page 49) 8835

Last month Mark Cantrell put objects into containers, this month he takes them out...

Removing Objects from the Container

Removing the objects is just as easy as putting them in. The example here ignores any dealings with the wearing or removal of objects, though in most adventures REMOVE is likely to be used for . taking off clothing etc. This shouldn't make it any more complicated. It also depends how fussy you want to be. If the player is wearing a raincoat and types in PUT RAINCOAT IN BAG, it's up to you whether you insist the raincoat has to be removed first, or whether it can go straight into the bag in one action - a bit like whether you make the player type OPEN DOOR before he or she can go through it, or whether you respond with You open the door and go through.'

REMOVE NOUN2 BAG; Referring to the bag?
PRESENT 1; Is the bag here?
AUTOT 1; Take the object if it's inside
DONE

REMOVE NOUN2 BAG; As above
ABSENT 1; The bag isnt here

SYSMESS 26; Prints 'There isn't one herel"
DONE

REMOVE SYSMESS 60; "Remove from what?"
DONE

REMOVE ALL NOUN2 BAG; As above
PRESENT 1; As above
DOALL 1; Look for objects inside the bag

There is no need to specify the container in the REMOVE commands, as I'll demonstrate in a moment when covering GET. Of course. this does not apply to the ALL option. You can't nest DOALL loops (i.e. perform a DOALL in one location, switching each time to a DOALL in another location) so you must be specific when GETting ALL objects from a container.

New users of PAWS should remember that it is important which order the commands are placed in, and again to finish off with a DONE. In this example, the first input checks whether the bag is present or not, and if it is then it acts upon that, with the DONE command telling it to check no further. If the bag isn't present, the program goes on to check your next entry, and will process that command instead - logic tells you that either the bag is present or it is absent. It has to be one or the other.

All these lines work in a similar fashion to the PUT lines. However, the AUTOT condact does the opposite to AUTOP. It looks for the specified object in the specified location (room one) and then processes it in a way similar to GET, reporting the successful manipulation of that object, if the player has it, or reporting if no such object is to be found inside the container.

The following lines do the same job as the lines for PUT. However, the DOALL loop for the ALL option looks for objects in the container rather than in the player's inventory.

Get and Drop

Let's consider how the GET command works in conjunction with containers:

GET PRESENT 1; Is the bag around?
WHATO; Which object is being referred to?
EQ 54 1; Is the object in the bag?
AUTOT 1; Get the object
DONE

GET ALL NOUN2 BAG
PRESENT 1
DOALL 1; See REMOVE ALL

These lines deal with taking objects from the container. The first line checks that the object is present. It then looks at the noun1/adjectivel used in the input to look up the object in the object word table; updating the relevant flags for the last 'GET entries which, if the object was not found in the container, will be processed.

```
GET LT34x
GT 3420; Nouns referring to objects?
WHATO; Which object is referred to?
EQ 55 0; Does it have zero weight?
  SYSMESS 61; Then report that it is immobile
DONE

GET LT34x
GT 3420; As above.
AUTOG; Get specified object
DONE

GET ALL DOALL 255; Process all objects
```

The first line checks that the noun entered corresponds to an object. In other words those nouns above 'x' may refer to places etc, not objects, and so would lead to strange results if they were fed into the GET process.

Those nouns below 20 are not object nouns, being movement nouns, for instance. Having established that the nouns are valid, the object is looked up in the object word table. Should the object have a weight of zero then it is an immobile object and this fact is reported to the player.

The next line again checks that the nouns. are valid and then performs an AUTOG. This AUTOmatically Gets any specified object and reports this to the player, also reporting if the player already has it, or is wearing it, or if it is not in the present location. The last line merely sets up an ALL option which scans for objects 'in the player's location and processes them.

The opposite action - dropping objects into the container - has already been explained under PUT. All you have to do is substitute the verb PUT for DROP. However, the PUT lines don't explain the general DROP lines.

```
DROP AUTOD; Drop the object done
DONE

DROP ALL DOALL 254; Scan for carried objects
```

These two lines are so simple that they are self-explanatory. In fact most of these entries are fairly easy to understand. They were included here because any explanation of containers cannot really be considered complete unless the placing or removing of objects is mentioned. And when mentioning GET and DROP you cannot really omit the general lines.

However, the use of NOUN2 to specify the container was used in the first PAWS adventures and may be something which newcomers to PAWS are still inclined to do. It is quite unnecessary, and in fact it.saves you some memory if you refrain from specifying the container in the GET condacts. It's also more user-friendly to allow the player to just type, for example, GET KEY rather than GET KEY FROM POCKET.

(Mark Cantrell in Your Sinclair issue 70 , Page 47) 8836

Having read your favourable review of Gilsoft's PAW, I must say that I agree wholeheartedly with your comments. I have discovered several tips which I feel will be of interest to your readers. The first two involve the implementation of two sophisticated commands used in the best adventures - AGAIN (repeat last command), and OOPS (back one move).

OOPS

This is hinted at in the manual. Here is how to actually achieve it. Insert OOPS as a synonym of I RAMLOAD (verb 29) and delete RAMSAVE. Then insert the following entries into the process table:

**0 E2 33 29 Is verb
RAM LOAD/OOPS
RAMLOAD 255 (see 1 below)
Restore posn
DESC Redescribe (new) Iocn

**1 RAMSAVE Save posn

Notes

1 To limit the power of OOPS with regard to certain commands, you may wish to keep certain flags. If, for example flag 30, the score, is to be preserved replace RAMLOAD 255 with COPY FF 30 255 RAMLOAD 254 COPY FF 255 30. You may also wish to reduce the score using MINUS 30 x.

2 It may alternatively be desirable to completely disable OOPS at certain times by setting a flag, say 100, and inserting in entry **0, as: E2 100 0 conduct at the start

AGAIN

Insert AGAIN (and desired synonyms] as a new verb number x. Use some storage flags (for convenience I have used 100 + number of flag whose value is to be stored). Then insert the following entries in the response table (which will come after the OOPS entries):

2 E233 x
COPYFF 133 33 33= VERB
COPYFF 134 34 34= NOUN1
COPYFF 135 35 35 = ADJECTIVE1
COPYFF 136 36
Restore 36 = ADVERB
COPYFF 143 43
Command 43 = PREPOSITION
COPYFF 144 44
(ie LS) 44 = NOUN2
COPYFF 145 45
45 = ADJECTIVE2

3 COPYFF33 133
COPYFF 34 134
COPYFF 35 135
COPYFF 36 136

Save Command (le L5)
COPYFF 43 143
COPYFF 44 144
COPYFF 45 145

NB The OOPS and AGAIN commands MUST be at the start of the table before the normal entries. This is why I have used the ** E2 33 ? technique rather than inserting straightforward OOPS - and AGAIN - entries.

Finally, a couple of brief tips I found helpful.

Sometimes it's desirable to see if an object can be picked up without actually trying to do it using AUTOG. The following routine does this and is best used as a new process table. It assumes the object in question is the current LS noun (adjective), so if this is not true flags 34 and 35 will need to be set to the appropriate values. It SETS flag 200 if picking up is possible and CLEARS it, otherwise flags 198 and 199 are also used.

* * 0 CLEAR 200 set 200 to zero

WHATO set the object flags
SAME 54 38 Is the object here?
COPYFF 37 199 Can the
SUB 1 199 player hold
GT 1990 one more object?
WEIGHT 199 Can the player cope
ADD 55 199 with the extra weight?
COPY FF 52 198
PLUS 198 1
suB 199 198
GT 198 0
SET 200 successful, so set flag 200 to 255

**1 DONE return to calling table

I also found it useful to create a special 'rubbish bin' location in say, loc'n d. This is due to the inability of the program to differentiate between objects which have not been created, and objects which have been created but then a room 1 we find object 1 , the pie, and that the pie, ifcarried, may be eaten. Then we use the following.

EAT PIE CARRIED 1 carrying pie
MESSAGE 'It's tasty' message
PLACE 1 d put it away
DONE finished

Now the obvious thing to do would have been to DESTROY 1. But then the followin entry for SEARCH ROOM wou have allowed us to discover yet another pie, after eating the first. However, since the pie is now in loc'n d, the routine will NOT allow another pie to be found.

SEARCH ROOM 0 AT 1 in room 1
ISAT 1 252 pie not yet found
MESSAGE 'Found' message
CREATE 1 create pie

DONE
SEARCH ROOM 1 MESSAGE 'Nothing special' Else ...
DONE

of course this could all be done with flags, but you only need one 'rubbish bin' to work for ALL such objects, and also you could make visiting the 'rubbish bin' part of the game (cf pirate's treasure chests in games such as Classic Adventure and Lords Of Time).

(Paul Fulcher in Crash issue 48 , Page 56) 8946

22 OCT 2009 17:09 522209

Brought to you courtesy of Gerard Sweeney and Nick Humphries
plus various other good eggs.

TT: 0 NQ: 3

CRACK CITY

It's taken a lo-o-o-o-ong time for this game to make it to a commercial release, as I first saw it umpteen moons ago when author Garry Cappucinni sent it in for review. Gary was planning to publish it himself under the awful title of '*Snap*', the first in a proposed trilogy. The other two were to be called – you guessed it – '*Crackle*' and '*Pop*'. I sent it back for the unusual reason that I thought it was simply too good! Too good to be wasted by trying to sell it himself and maybe only flogging a few copies, when he ought to be getting on with writing the sequels. I suggested he get a software house to publish it – and that he change the title. So here it is, published by Zenobi Software as *Crack City*, the first part of the *Snow Dogs Trilogy*. A vast improvement on *Snap*!

The game takes place in the near future, with you playing the part of a secret agent in the Secret Intelligence Foundation. The setting is New Washington, a city built to support Washington's increasing administration side and the north-east USA's ever-growing population. As a result of a Mafia drugs operation which is being uncovered, you've been called back from your holiday in the Maldives. And I wish I had a fiver for every secret agent who's been called back from holiday in order to star in an adventure game! But that's the only cliché, the rest is fresh and well thought-out.

In fact, you get to choose which of three secret agents you want to be. There's Mick Hammelford, who scores well on the shooting range but will never be heavyweight champeen of the world; Louis Chorbenski, a Polish giant whose brain rattles about in his head like a pea in a whistle; and finally Ed Macpherson, who's English and a general all-round good egg.

Crack City was written using *PAWS*, but you'll never believe it when you see the screen layout, with its changing graphics, icons, complex status line and loads of other features. It'll give *PAWS* users heart attacks as they try to figure out how it's been done. In the graphics section at the top of the screen there's more information than you'll find in the average MI5 Filofax. If there had only been the 3D perspective graphic of each location that would have been impressive enough, but that takes up a small portion of the screen and is surrounded by other information.

On the left a series of icons shows the available exits in white, or grey if not currently available. This changes if you find a secret exit by searching around. The faces shown also change, as characters appear and disappear in the same location as yourself – nurses, policemen and even Dwight, your faithful assistant, who's given to wandering in and out looking lost 'til you tell him to stick with you. Oh yes, you can talk to all these characters as well, although that's subject to the usual problems that speech in an adventure throws up. A lot of people spend a lot of time saying "Hi" to each other.

On the right a section is given over to the objects you're carrying, showing how close you are to the maximum permitted in your hands, your pocket, your wallet and so on. This isn't just a gimmick, as it does help if you look at them to see if you can pick up one more object, or need to put something down first.

Below this little lot is the status line, showing the time of day, your cash, skill level, SIF rating, mortality level, score and number of turns taken. Again, they're not just put there for show, most are an integral part of the game – if it's night you'll need to sleep, if your mortality level's low and your turns are increasing, watch out.

It helps to have these things permanently visible on the screen, rather than have to keep typing separate commands for each one. You'd spend so much time checking on your status you'd never find time to do anything! And if you're worried that all this stuff is bound to slow the game down, then think again. Everything appears almost instantly, and that includes the graphics. Half a second and they're there, a testament to the programmer, to *PAWS* and to the good old Spectrum. If, however, you don't want all this stuff, you can just as easily switch it all off and play *Crack City* as a straight text adventure, with all the sophisticated commands that *PAWS* allows, like multiple inputs, OOPS, RAMSAVE and so on.

But what of the gameplay itself? Well, inevitably with all these memory-hungry features, it doesn't have the depth of a Dungeon Adventure, let's be honest. But the author's done his best to make the problems more complex, and that works better in a game with fewer locations. Your first task is to find something to eat, which means not being distracted by each and every problem you'll encounter near the start – get out there and find something to munch! I did get a bit annoyed when I kept dying of hunger, but it's a sign of how engrossing the game is that although I thought it was happening every few moves, in fact I'd taken 78 turns on one occasion and not realised it. My score slowly improved too. First a shameful 2%... then 7%... then 12%... a reason to keep playing.

Not that you'll need much reason to keep playing *Crack City* – you'll play it because you want to. The game is quite simply brilliant, and puts most home-grown adventures to shame. A few years ago it would have been snapped up by someone like Mastertronic or Firebird and sold thousands, no doubt about it. Never mind, it's available, that's all that matters, and discerning adventurers will be adding it to their collections without delay.

Title.....*Crack City*
Price.....£2.49
Publisher.....Zenobi Software 26 Spotland Tops, Cutgate, Rochdale Lancs OL12 7NX

Graphics
Text
Value for Money
Personal Rating
9

EL RINCÓN DEL PARSERO

Para todos aquellos que intentan dar a sus aventuras un «look» profesional van dos útiles rutinillas.

1. Salidas automáticas. Para imprimir en pantalla todas las salidas posibles:

Proceso 1

```
*      MES 1000
S      COPYFF 38 100 LET 33 2 MOVE 100 ADD 100 101
       NOTZERO 100 MES 101
E      Lo mismo excepto LET 33 3 y MES 102
O      Lo mismo excepto LET 33 4 y MES 103
N      Lo mismo excepto LET 33 5 y MES 104
Arriba Lo mismo excepto LET 33 11 y MES 105
Abajo  Lo mismo excepto LET 33 10 y MES 106
```

Mensajes

100 Salidas visibles
101 Sur
102 Este
103 Oeste
104 Norte
105 Arriba
106 Abajo

Por supuesto debes definir antes en el vocabulario tus direcciones de acuerdo con los valores en la sentencia LET 33 x. Es un método bastante rudimentario, pero lo que queremos es darte la idea general. Esperamos tus mejoras.

2. Un reloj. Este reloj en tiempo real lo puedes poner en tiendas que se abren y se cierran, para autobuses o trenes que llegan y se van a un tiempo fijado, etc.

Banderas
150 minutos
151 horas
152 días

Proceso 2

```
* *  PLUS 150 1  EQ 150 60   CLEAR 150   PLUS 151 1
      EQ 151 24   CLEAR 151   PLUS 152 1  EQ 152 7
      CLEAR 152
```

Proceso 6 (por ejemplo)

```
- -  COPYFF 151 153  GT 153 11  MINUS 153 12
- -  PRINT 153  MES 2  PRINT 150  GT 151 11
     MESSAGE 3  DONE
- -  MESSAGE 4
```

Tabla de Respuestas

VER RELOJ MES 1 PROCESS 6 DONE

Mensajes

1 La hora es
2 :
3 PM
4 AM

El Rincón del Parsero

RUDIMENTOS DE P.S.I.

Los Personajes Semi Inteligentes están de moda en las aventuras, pero hay que tener cuidado con ellos, planearlos minuciosamente y, sobre todo, ver si en realidad son necesarios.

Pensemos que consumen gran cantidad de esa preciosa memoria de que tan escasos andamos los creadores, dándose el caso de que, a mayor complejidad de su rutina interna, menos nos queda para el diálogo con que ellos han de expresarse.

Empezaremos, como siempre, por lo más simple.

1. Un tipo que te impide el paso. Hay que llevar un objeto. Si no lo llevas, no hay tu tía, te mata.

```
RESPUESTAS   - - AT (número de habitación
                 donde está)
             CARRIED (El no. del objeto)
             MESSAGE 100
             DONE
             - - AT (No. de la habitación)
             NOTCARRIED (el No. del objeto)
             MESSAGE 101
             TURNS
             SCORE
             END
MENSAJES     100: Pasas porque llevas el (lo que sea).
             101: Lo siento, furioso, el tipo te mata.
```

2. Al llegar a una localidad, si te demoras un tiempo, aparece fulano y te mata.

```
PROCESO 1    - - AT (No. de loc)
             TIME (tiempo) 1
RESPUESTAS   - - AT (No. de loc)
             TIMEOUT
             MES 100
             SCORE
             TURNS
             END
```

MENSAJE 100: Aparece el fiero fulano y te suicida.
Nota: Puedes poner el tiempo que quieras.

El Rincón del Parsero

Dos grandes problemas con los que se enfrentan los hacedores de aventuras son tratados hoy en este rincón: los insultos y las repeticiones.

PROTECCIÓN CONTRA INSULTOS

Viene muy útil para desterrar los malos hábitos de ciertos agresivos aventureros.

Vocabulario

Pon todos los insultos que se te ocurran como Verbos con el mismo número.

Tabla de Respuestas

(insulto)

BEEP 10 10
MESSAGE 50 ANYKEY END

Mensajes

Mensaje 50 = «No insultes, comienza de nuevo).

REPETICIÓN

Usado en muchas aventuras profesionales, se trata de evitar el tener que teclear toda una frase de nuevo, basta teclear «Repetir».

Vocabulario

Verbo 56 = REPETIR (o similar)

Tabla de respuestas

```
REPETIR *   CLEAR 240 EQ 33 56 COPYFF 233 33
            COPY FF 234 34 COPYFF 243 43 COPYFF
            244 44 COPYFF 245 45 SET 240
*     *     COPYFF 33 233 COPYFF 34 234 COPYFF
            43 243 COPY FF 44 244 XOPYFF 45 245
-     -     MOVE 38 DESC
-     -     LT 33 14 SYSMESS 7 DONE
```

Para mayor pena, interviene el cáncer y se carga al pobre Iñaki. Estás en sus funerales contemplando la gran herencia que te tocó: una cinta de ordenador donde te cuenta sus andanzas.

Ya en la segunda parte debes enfrentarte a las varias pruebas a que te somete el guardián, unas de cultura general, otras de honradez y una puñetera adivinanza.

GALEN

Autor: Gerardo Oporto Jorrin, de Santander.

Fantasía Tolkieniana.

Has vuelto a tierras donde tu familia tuvo que huir hace 17 años, dejando a tu padre del cual no se supo nada más.

Hay bosques, cascadas, gemas, un familiar ermitaño, un vagabundo, temibles Orcos, la clásica caverna donde hay que llevar la antorcha, un brujo, el espejo mágico, un típico acertijo para encontrar en cual de las puertas está el Rey Orco y cosas por el estilo.

La ayuda viene de las páginas de un extraño, pero indispensable Manual para Jóvenes Castores.

EL GATO CON BOTAS

Autor: Ignacio González-Cutre Coll

Gráficos: El Sr. González-Cutre (Padre) De Murcia.

Como novedad, tiene música y efectos sonoros en muchas partes y una muy buena presentación.

Narra la historia del Gato con botas y del hijo del molinero (Marqués de Carabás) a quien debemos hacer rico y que se case con la princesa.

Y con esto acabamos la primera muestra de los guiones. Adios.

ANDRES R. SAMUDIO - 1990

El Rincón del Parsero

Continuando con el tema iniciado el pasado mes, seguimos en nuestro Rincón del Pasero analizando algunos rudimentos de los personajes semi-inteligentes.

MENSAJE 100: Aparece el fiero fulano y te suicida.
Nota: Puedes poner el tiempo que quieras.

3. Un personaje bueno que aparece y te dá un valioso objeto si llevas algo a cambio.

```
RESPUESTAS    — — AT Locno.
                  CARRIED objno. 2 (el que él
                  desea)
                  MESSAGE 100
                  SWAP 2 3 (Te lo cambia)
                  DONE
```

MENSAJE 100: El hada buena, al ver que llevas el (objeto que sea), te lo cambia por un (lo que sea) y desaparece con un suave «puff».

4. Rutina que permite varios personajes aparecer y desaparecer de una forma aparentemente aleatoria.

```
MENSAJES 1: El — está aquí.
         2: El — ha llegado.
         3: El — se dirige hacia
         4: el norte.
         5: el sur.
         6: el este.
         7: el oeste.
         8: arriba.
         9: abajo.
PROCESO 5     * * COPYFF 33 254
                  CLEAR 252
                  CLEAR 253
                  SAME 38 251
                  SET 252
PROCESO 5     S — CHANCE 35
                  LET 33 2
                  MOVE 251
                  SET 253
                  NOTZERO 252
                  MES 3
                  MESSAGE 5
              E — ZERO 253
                  CHANCE 55
                  LET 33 3
                  MOVE 251
                  SET 253
                  NOTZERO 252
                  MES 3
                  MESSAGE 6
              O — ZERO 253
                  CHANCE 45
                  LET 33 4
                  MOVE 251
                  SET 253
                  NOTZERO 252
                  MES 3
                  MESSAGE 7
              N — ZERO 253
                  CHANCE 45
                  LET 33 5
                  MOVE 251
                  SET 253
                  NOTZERO 252
                  MES 3
                  MESSAGE 4
ARRIBA        — ZERO 253
                  CHANCE 45
                  LET 33 11
                  MOVE 251
                  SET 253
                  NOTZERO 252
                  MES 3
                  MESSAGE 8
ABAJO         — ZERO 252
                  CHANCE 20
                  LET 33 10
                  MOVE 251
                  SET 253
                  NOTZERO 252
                  MES 3
                  MESSAGE 9
              — — COPYFF 254 33
                  SAME 38 251
                  NOTZERO 253
                  MESSAGE 2
PROCESO 1:    — — SAME 38 (ver nota 2)
                  LET 51 (ver nota 1)
                  MESSAGE 1
PROCESO 2:    — — LET 51 (ver nota 1)
                  COPYFF 251 (ver nota 2)
                  PROCESS 5
                  COPYFF 251 (ver nota 2)
```

(Estos procesos 1 y 2 se repiten para todos los personajes que quieras poner en tu juego.)

Nota 1: En los LET 51 hay que poner el número de objeto dado a cada personaje.

Nota 2: En los SAME y COPYFF hay que poner el número de bandera asignada a cada personaje.

Os ruego analiceís la anterior rutina pues los principios que utiliza son fundamentales para lograr un mejor entendimiento del P.A.W.

Por cierto: ¡Enviad vuestras rutinas! Si son de interés general, serán publicadas bajo vuestro nombre.

EL ESPACIO MALIGNO

Autor: Daniel Pérez Espinosa, Getafe, Madrid.

Tema Magia Fantasía. Muy parecido al de la Historia Interminable. Resulta que el malvado brujo de turno ha escapado y trama venganza contra el Reino Fantástico, robando a su Rey y enviando las fuerzas del Mal contra la Tierra.

Precisamente esas fuerzas son el Espacio Maligno que tú y tu amigo elfo debeís vencer. Para ello debéis conseguir que te den la Pócima Antimaligna, la Corona Psicotantánica, la Esfera de la Vida, etc.

LAS MINAS PERDIDAS

Autor: Alvaro Mangado Cruz, de Pamplona

Eres un gran cazador que vuelve de lo que considera su última expedición en Africa.

Pero resulta que dos aristócratas te contratan para seguir el rastro de otro explorador que se perdió buscando las minas del Rey Solomón.

En tu travesía por la selva encontrarás todo tipo de fieras y tribus salvajes.

Luego te espera otra región mucho más seca: el desierto.

MÁS RESULTADOS PARA EL CONCURSO

El siguiente grupo de aventuras pasa a unirse con las seleccionadas anteriormente:

La Destrucción de América, de C. Bádenas.

Mundo Fantástico, de D.T. Soft (Hnos. Diaz Tellez).

Olimpo en Guerra, de Edo. José Villalobos.

El libro Negro, de Year Zero Soft.

Don Juan, de J. Rodríguez Parra.

Diatmar, de S. Marqués Solís.

Seguiremos informando.

ANDRÉS REPELENTE SAMUDIO - 1990

El Rincón del Parsero

1-UN SUSTO PARA LOS GROSEROS

Nada mejor para hacer escarmentar a los aventureros maleducados que ésta rutina que simula un reset como respuesta a un insulto.

Tabla de Respuestas

```
Insulto    Insulto   CLS
                     PRINTAT 10 0
                     MESSAGE 1
                     ANYKEY
                     BORDER 7
                     PAPER 7
                     CLS
                     PAUSE 20
                     PAPER 0
                     CLS
                     PAUSE 50
                     PAPER 7
                     CLS
                     PRINTAT 20 0
                     MESSAGE 2
                     PAUSE 200
                     PAPER 0
                     BORDER 0
                     CLS
                     MESSAGE 3
                     ANYKEY
                     DESC
                     DONE
```

Mensajes:
1- ¡Te lo has buscado!
2- (C) 1982 Siclair Research Ltd.
3- Jié, Jié, Te doy otra oportunidad.

2-SALIDAS SECRETAS

No debes poner la salida en la tabla de conecciones, puesto que sólo debe permitirse el paso si el jugador hace la acción adecuada.

En nuestro caso sería apretar un oculto botón que se supone estaba detrás de un armario que ya has separado de la pared Este.

Tabla de Respuestas:

```
Apretar Botón  AT (No. de Localidad)
               SET 200
               MESSAGE X
               DONE
Este     —     AT (No. de Localidad)
               NOTZERO 200
               GOTO (donde quieras)
               DESC
               DONE
```

Mensajes:
X: Un panel se desliza dejando ver una oscura entrada.

3-CLAVE DE PROTECCIÓN

La manera más simple de poner un sistema de clave para una segunda parte o para iniciar un juego es usando * * para que la entrada quede al inicio de la Tabla de Respuestas.

Define la Palabra Clave como un verbo. (En éste caso 100).

```
*  *   EQ 33 100
       SET 200
       OK
*  *   ZERO 200
       MESSAGE X
       DONE
```

Mensajes:
X : Clave Inválida.

4-UNA ORDEN SALVADORA

La siguiente es una utilísima rutina, además es corta y fácil de instalar. Debe ponerse si en tu aventura has puesto sitios en los que el jugador se pueda quedar encerrado y le espere una triste suerte. P. E., una cárcel o un baúl donde te has metido.

Entonces se le dá al aventurero una oportunidad de poder volver a la localidad anterior si teclea la palabra mágica. Por supuesto que para encontrar dicha palabra habrá tenido que hacer alguna otra cosa, así que unos la sabrán y otros no. En nuestro caso la palabra es: Libertad.

Tabla de Respuestas:

```
Libertad  —  COPYFF 244 38
             COPYFF 245 244
             DESC
—         —  MOVE 38
             COPYFF 245 244
             COPYFF 38 245
             DESC
```

Como explicación: Hacer la última localidad la localidad actual.
Hacer la previa localidad actual la nueva última localidad.
Describir la localidad.
Movida si existe salida.
Hacer la última localidad la localidad actual.
Guardar la nueva localidad actual.
Describir la nueva localidad.

Así la bandera 245 es una mera copia de la 38 y se usa como bandera de trabajo. Cuando el jugador se mueve la 245 se copia en la 244 y la nueva localidad se copia a la 245.

Al teclear «Libertad», o lo que sea, se copia la 244 a la 38 y el contenido de la localidad actual se copia en la 244 que se describe.

Es muy alentador observar como este pequeño rincón va tomando vida propia y expandiéndose hasta llegar a ocupar toda una habitación.

Hoy presentamos dos aportaciones de los lectores esperando que el ejemplo cunda y cada vez enviéis rutinas o mejoréis las ya publicadas.

NOTA: Las rutinas se han dejado tal cual las han enviado sus creadores, si hay errores o maneras más eficaces de ejecutar la misma acción, os toca a vosotros decirlo.

SALIDAS MEJORADAS

Enviada por Luis F. Sánchez Palacios, de Ciudad Real.

Ventajas: si el jugador teclea una dirección errónea, además de decirselo, les recuerda las que hay. Si no hay salidas, también se le informa.

Tabla de Resupuestas:

Salidas – PROCESS 3,
– * LT 33 15, MOVE 38, DESC
– – LT 33 15, SSMESS 7, PROCESS 3, DONE

Procesos 3
* * COPYFF 38 100, LET 33 13, MOVE 100, ADD 100 101, NOTZERO 120, SET 16, ST 22
* * Igual excepto: LET 33 2, SET 17
* * Igual excepto: LET 33 3, SET 18
* * Igual excepto: LET 33 4, SET 19
* * Igual excepto: LET 33 9, SET 20
* * Igual excepto: LET 33 10, SET 21
– – ZERO 22, MES 7, DONE
– – MES 0, CLAR 22
– – NOTZERO 16, MES 4, CLEAR 16
– – NOTZERO 17, MES 1, CLEAR 17
– – NOTZERO 18, MES 2, CLEAR 18
– – NOTZERO 19, MES 3, CLEAR 19
– – NOTZERO 20, MES 5, CLEAR 20
– – NOTZERO 21, MES 6, CLEAR 21
– – SYSMESS 20

Procesos 1
SALID – NOTAT 0 (y cualquier otra excepción donde no queramos que aparezcan las alidas) PROCESS 3

Mensajes
0: Salidas visibles:
1 a 6, todas las salidas
7: No hay ninguna salida visible.

Vocabulario: Por supuesto poner todas las salidas y sinónimos. Subir y bajar son nombres convertibles a verbos para que el intérprete entienda «IR ARRIBA».

PASOS SONOROS

Un ANÓNIMO que insiste en que «no te voy a decir quién soy y además la carta no te la mando desde mi ciudad natal» (¡Oiggs! qué terrible suspense, no sé si podré dormir esta semana), que se hace llamar GUMP y se describe como «pequeño ser decrépito, pero más ágil y salido de lo que tu mente pueda imaginar», envía una rutinilla que dice «crea mucho ambiente».

Procesos 1
NOTSAME 38 200 (comprueba que no es una redescripción), COPYFF 38 200 (copia el número de la nueva localidad) BEEP 3 35, BEEP 3 30, BEEP 3 32, PAUSE 25, BEEP 3 35, BEEP 3 30, BEEP 3 32, PAUSE 25, BEEP 3 35, BEEP 3 30, BEEP 3 32, PAUSE 25, BEEP 3 35, BEEP 3 30, BEEP 3 32.

ANDRES RAMBO SAMUDIO - 1990

INVENTARIO:

MENSAJES SISTEMAS 9-TIENES
10-TIENES PUESTO
11-NO TIENES NADA

PROCESO X - DISTINTO DE 1 Y 2.

I * CLEAR 11 DOALL 253
I – PLUS 11 1 DONE

RESPUESTAS

I * CLEAR 44 PROCESS X NOTZERO 1 SYSMESS 9 LISTAT 254
I * NOTZERO 11 SYSMESS 10 LISTAT 253
I – ZERO 1 ZERO 11 SYSMESS 11
I – DONE

EXAMINA: Desde TR a PRX

— — MESSAGE x Ultima entrada por si fallan todas
DONE las demás, "nada especial"

Para cada objeto/cosa

Ex — ① AT x (¿hace falta?)
② PRESENT/CARRIED x
MESS x
DONE

HELP: Desde TR a PRX.

— — MESS x Por si fallan
DONE

HELP — AT/CARRIED/PRESS/ABSENT
MESS x
DONE

OOPS (PR 2) RAMSAVE
(RSP) OOPS — COPYFF x y
RAMLOAD 250 - Si se quiere conservar
COPYFF y x alguna bandera

El Rincón del Parsero

Hoy daremos una sencilla rutina para el diálogo de un PSI. Se trata de una forma básica y ampliable en diversos grados, pero muy interesante para comprender todo el mecanismo fundamental.

*** * MESS 1 PARSE MESSAGE 2 NEWTEXT DONE**

Pon entremedio todas las respuestas a cualquier SL (Sentencia Lógica) que dé al PSI, hasta terminar el Proceso con:

— — MESSAGE 2 NEWTEXT DONE

Nota: Recuerda de poner las " " en las frases que el PSI tenga que entender.

Mensajes: 1- "El 1.° te mira y dice... "
2- «Si no vas a decir nada interesante, no pienso seguir escuchándote»

ANDRES REAGAN SAMUDIO - 1991

El Rincón del Parsero

Esta vez el audaz ARPAK, conocido por los mortales como Francisco Martín Vida, nos envía varios «granitos» que juntos llegan a formar una eficaz piedra para conseguir un mejor lucimiento en nuestra obra.

INVENTARIO MOLÓN

La ventaja es que separa lo que llevas puesto de lo que llevas en las manos, evitando el antiestético «llevas un transatlántico, la sección de caballeros de un gran almacen (puesto), y una brillante partícula subatómica».

Algo en las manos y nada puesto: enumera lo primero y suprime el nauseabundo «tienes puesto nada». Nada en las manos y si algo puesto: enumera lo puesto y elimina «llevas nada». Algo en las manos y algo puesto: enumera los dos independientemente. Tan desgraciadito que ni en las manos ni puesto: El mensaje del sistema No. 11.

Mensajes del sistema:
9: Llevas. 10: Tienes puesto. 11: No tienes nada.

Proceso x

INVEN	*	DOALL 254
INVEN	—	PUTO 0, DONE
LOAD	*	DOALL 253
LOAD	—	PUTO 254, DONE
SAVE	*	DOALL 254
SAVE	—	PUTO 253, DONE
QUIT	*	DOALL 0
QUIT	—	PUTO 254, DONE

Proceso 1

*	*	AT 0, ANYKEY, GOTO 1, DESC

Respuestas (en ese mismo orden)

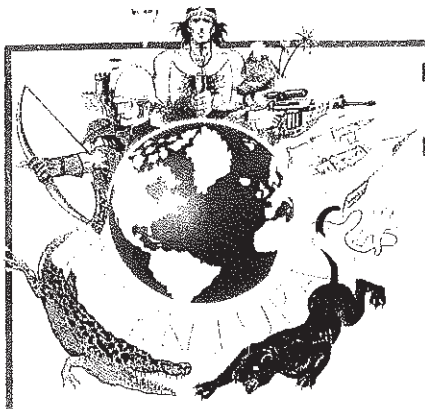
INVEN	*	SET 44, SET 34, COPYFF 1 11, PROCESS x, NOTZERO 11, SYSMESS 9 LISTAT 0
INVEN	*	LET 33 (+load), PROCESS x, LET 33 (+inven), NOTZERO 1, SYSMESS 10, LISTAT 254
INVEN	*	ZERO 1, ZERO 11, SYSMESS 11, DONE
INVEN	*	LET 33 (+save), PROCESS x, LET 33 (+quit), PROCESS x, DONE

Nota 1: Deja la Loc. 0 como presentación, sin que se pueda teclear en ella.

Nota 2: (+load), etc. son los números asignados a estos verbos en el vocabulario.

Nota 3: Si utilizas la base de datos START, borra la entrada I —, pues no hace falta.

ANDRES RADIOACTIVO SAMUDIO - 1990



ESPECIAL

Como véis, el tema de hoy está dedicado principalmente al Rincón del Parsero. La razón es la cantidad de cartas que hemos recibido pidiendo que aclaráramos lo de COGER y SACAR DE. Ahí va la respuesta.

PARSERO

PROGRAMAS DEL CONCURSO ELIMINADOS Y SELECCIONADOS

Siguiendo con el informe iniciado en el número anterior, paso ahora a detallar el resultado de haber probado con detenimiento 8 nuevas aventuras:

AVENTURAS ELIMINADAS

LA DESTRUCCIÓN DE AMÉRICA, de Conrado Bádenas: Los problemas son demasiado retorcidos en ocasiones.

EL GATO CON BOTAS, de Ignacio González-Cutre: El programa es demasiado fácil al ceñirse con exactitud al desarrollo del cuento. Ello elimina a un programa en el que los gráficos son excelentes.

DON JUAN, de Javier Rodríguez Parra: Los gráficos, aunque efectistas, son algo pobres. La dificultad es elevada en ocasiones.

HAMPA 1930, de J. Izquierdo y L. Calvo: Realizada con el GAC. Tiene varios puntos poco lógicos.

AVENTURAS SELECCIONADAS

MEMORIAS DE UN HOBBIT, de JSJ Soft: Buen tratamiento del tiempo con variación día-noche, variando los graficos con ello.

ALERTA OVNI, de Ricardo Villalba Martínez: Tiene una buena trama argumental. Los gráficos no son muy brillantes, pero meten en ambiente.

JEKYLL VS HIDE, de Morbosoft: Es muy graciosa, con problemas lógicos y buenos gráficos. Es de destacar el cambio de actitud del protagonista al cambiar a Mr. Hide.

EL ANILLO, de Juan Antonio Paz Salgado: Destaca sobre todo por lo original de su primera parte, en la que eres... un anillo. Todas las acciones se ejecutan a través de órdenes a PSIs, que está muy bien tratados.

Juanjo Muñoz

El Rincón del Parsero

MANIPULACION DE OBJETOS Y CONTENEDORES

El Professional Adventure Writing System (PAW) tiene órdenes que ponen o sacan, de una manera automática, los objetos dentro de un contenedor. Sin embargo, para que todo funcione, es preciso que los contenedores estén ordenados adecuadamente y comprobar si están presentes o no.

Para cada contenedor hay que dejar una habitación vacía, que no se usará en el juego. Se usa como el interior del contenedor y debe llevar, además, el mismo número que el texto para objetos del contenedor. Por lo tanto, cuando metemos un objeto dentro de otro, lo que estamos haciendo es poniéndolo en una habitación vacía y desconectada de toda la aventura.

Por ello es necesario haber planeado con anterioridad cuantos contenedores vamos a usar, para poder ponerlos en orden en la tabla de objetos y dejar esas habitaciones libres. Las habitaciones en las cuales se va a mover el jugador, deben empezar a numerarse a partir de la última que se usó como contenedor.

Claro, que el objeto cero (0) será la fuente de luz y la habitación cero (0) donde pondrás la pantalla de créditos.

En la tabla de atributos de los objetos, hay que especificar los contenedores, si son llevables o no y su peso. Según la guía técnica del PAW, (pag. 66 nota 2) si se le da un peso de cero (0) a un contenedor, el peso de los objetos contenidos dentro de él no serán suma-

dos cuando el programa comprueba cuánto peso estás llevando. Si lo haces así, hay que tener en cuenta que se puede llevar, dentro del contenedor, un peso infinito de objetos.

METIENDO

El objeto número 1 es una bolsa, con un peso de x y llevable.

Tabla de Respuestas

```
METER NOUN2 BOLSA   hay que especificar «bolsa»
      PRESENT 1     ¿está la bolsa?
      AUTOP 1       pon el objeto dentro
      DONE          termina orden

METER NOUN2 BOLSA   como arriba
      ABSENT 1      ¿no está?
      SYSMESS 26    «No hay una» o «Eso no está», etc.
      DONE

METER SYSMESS 60    ¿Dónde?
      DONE

METER NOUN2 BOLSA
TODO  PRESENT 1     ¿está la bolsa?
      DOALL 254     busca en el inventario
```

NOUN2 BOLSA, es para asegurarnos de que la Sentencia Lógica compruebe mas allá que el simple verbo/nombre para ver a qué contenedor se refiere. Si fuera el único, no sería necesario. También puede ser ampliado con el adjetivo verde, blanco o el que se quiera mediante la orden ADJECT2.

AUTOP se encarga de procesar el objeto especificado. Si está llevado, lo pone dentro de la bolsa. Si no, da una respuesta indicando si está presente, ausente o puesto.

DOALL 245 procesa todos los objetos llevados. La ponemos aquí como la última opción, pero en PAW es de las primeras líneas que debe comprobarse. Hay que estar seguros de que se ejecute cuando la bolsa esté presente, porque si no lo está, nos dará una respuesta para cada objeto procesado.

SACANDO

Tabla de Respuestas

```
SACAR NOUN2 BOLSA
      PRESENT 1
      AUTOT 1
      DONE
SACAR NOUN2 BOLSA
      ABSENT 1
      SYSMESS 26
      DONE
SACAR SYSMESS 60    ¿De dónde?
      DONE
SACAR NOUN2 BOLSA
TODO  PRESENT 1     ¿está la bolsa?
      DOALL 1
```

Los comentarios son los mismos, excepto que:

AUTOT busca por el objeto especificado en la localidad especificada, (en este caso la 1), y lo coge GET. Si no lo encuentra, lo procesa de la misma forma que AUTOP.

DOALL 1 lo que hace es buscar el objeto dentro de la bolsa.

SIN DECIR DE DONDE

Pero es posible, para una mayor sencillez, el prescindir de tener que SACAR y usar COGER, así como también de tener que especificar de dónde lo sacamos.

```
COGER PRESENT 1     ¿está el contenedor?
      WHATO         ¿a qué objeto se refiere?
      EQ 54 1       ¿Está en la bolsa?
      AUTOT 1       coge el objeto
      DONE          basta
```

WHATO mira el nombre1/adjetivo1 usado en la entrada (input), para buscar el objeto en la tabla de relación objetos-palabra, actualizando las banderas adecuadas para el último objeto mencionado.

Esto no funciona bien para COGER TODO, que debe ser hecho igual a SACAR TODO. El problema es que no se pueden anidar circuitos (loops) DOALL (hacer un DOALL en una localidad y luego hacer otro DOALL para otra localidad), por ello hay que ser específicos al coger todos los objetos de un contenedor.

Por lo tanto COGER TODO = NOUN2 BOLSA, PRESENT 1, DOALL 1

Pero las siguientes líneas deben preceder a la entrada general de COGER, para cuando el objeto no esté dentro del contenedor.

```
COGER LT 34 X       marcamos límite superior para
                    objetos
      GT 34 20      marcamos límite inferior para
                    objetos
      WHATO         ¿A qué objeto se refiere?
      EQ 55 0       ¿Tiene un peso de cero?
      SYSMESS 61    informa que no es objeto movi-
                    ble
      DONE
COGER LT 34 X       como arriba
      GT 34 20      como arriba
      AUTOG         coge el objeto especificado
      DONE
COGER TODO DOALL 255  procesa todos los objetos
```

La primera entrada para asegurarnos que el nombre corresponde a un objeto. Es decir, los nombres mayores que X pueden ser de lugares, etc., no de objetos y ello daría problemas si «alimentasen» el proceso GET. Luego separamos también los nombres más bajos que 20, que suelen ser direcciones de movimiento.

Ya seguros de que se trata de un objeto válido, se busca en la tabla de objetos-palabras. Si su valor es cero (0), se responderá al jugador que no es movible.

La segunda entrada comprueba que el objeto sea válido y lo coge automáticamente.

La última entrada busca los objetos que estén en la misma localidad del jugador, no dentro de la bolsa, y los procesa.

Si se quiere hacer para DEJAR, sólo hay que cambiar el verbo METER por DEJAR, aunque hay que añadir dos entradas más para cuando se trate del uno común de DEJAR, es decir no dentro de la bolsa.

```
DEJAR AUTOP
      DONE
DEJAR TODO DOALL 254  busca entre los objetos llevados
```

Estas dos entradas son tan simples que se explican por sí solas y han sido puestas aquí para no omitir los DEJAR normales.

SACAR DE O SIMPLEMENTE COGER

Aquí las opiniones están divididas. Desde las primeras aventuras hechas con PAW se usó el SACAR DE (contenedor) y es la forma preferida de Tim, su creador, y usada en COZUMEL; pero hay otros que creemos en la forma más simple, COGER y de hecho estamos trabajando en ello y solucionando los diversos problemas que se presentan cuando se trata de contenedores múltiples para que se puedan usar las dos formas simultáneamente.

Pero el problema no es tan sencillo y, si no se usa con cuidado pueden montarse verdaderos follones entre lo que el jugador quiere coger de dentro de un contenedor o coger de la habitación; por ello la mayoría de los programadores prefieren evitar estos líos.

En cuanto a dejar, creo que debe seguir siendo especificado si lo quieres dejar en el suelo o dentro de la bolsa o lo que sea.

Pero en gustos no hay nada escrito.

ANDRES ROLLERO SAMUDIO - 1991

TECNICAS AVANZADAS DE PROGRAMACION DE AVENTURAS

Por... Javier San José

Se inicia una nueva sección encaminada a presentar ciertas técnicas, trucos, rutinas, ideas, etc... para la programación de vuestras aventuras. Esta sección tratará de llevar los parsers existentes más allá de sus posibilidades. En principio los listados que se incluirán serán para el PAW, el parser, quizá, más conocido por todos vosotros, aunque además de los listados se incluirá la correspondiente descripción del funcionamiento de las rutinas, con lo que su adaptación a otros sistemas o lenguajes de programación no debería ser muy compleja. Espero que esta sección sea de vuestro interés; aún así desearía que me comuniquéis vuestras dudas, sugerencias, etc... y de esta forma adecuar la sección a vuestros gustos. Para ello podéis escribir a:

Javier San José
Ronda del Ferrocarril 75, 7º A
Miranda de Ebro
09200 BURGOS

Vamos a empezar esta sección con un pequeño truco de programación para el PAW. Yo siempre he considerado que una aventura conversacional debería facilitar un poco el considerable trabajo de tecleo por parte del jugador. En este aspecto es útil el uso de abreviaturas para los más comunes verbos de movimiento y de acción.

Así es muy común usar N para NORTE, I para INVENTARIO, M para MIRAR, etc... Pero además en las aventuras de AD es común usar una forma abreviada para hablar con los PSIs que encontremos. Se trata de sustituir el típico: DECIR NOMBRE "texto", por una forma más cómoda: NOMBRE "texto", con lo que nos ahorramos el consiguiente tecleo. En PAW es fácil conseguir esto con una simple entrada en la Tabla de Respuestas:

```
*        *        GT  34 30
                  LT  34 50
                  EQ  33 255
                  LET 33 24
```

Explicaré lo más brevemente posible el funcionamiento de esta entrada que, por cierto, debe ser la que aparezca en último lugar si hay más entradas * *. La bandera 34 es la que contiene el nombre de la frase tecleada. Lo primero es ver si es un nombre propio de personaje (podemos tener otros tipos de nombres propios como ciudades, ríos,...). En el caso que nos ocupa estarían situados entre los números 30 (GT 34 30) y 50 (LT 34 50) no inclusives. Por supuesto que estos números pueden variar de acuerdo con vuestra aventura.

Lo siguiente es comprobar si en la frase no aparece ningún verbo (bandera 33 igual a 255, EQ 53 255) con lo que podemos asegurar que se tecleó sólo el nombre de algún PSI, posiblemente seguido de un diálogo, pero eso ya lo veremos en la rutina de diálogo de es PSI. Si no aparece ningún verbo en la frase, colocaremos el verbo DECIR (LET 33 24) que en este caso tiene número 24 en el vocabulario. Como antes, este número puede variar de aventura en aventura así que debéis colocar el que se adecúe a la vuestra.

Y ya está. En algún lugar de la Tabla de Respuestas tendríamos las entrada para diálogos con los PSIs (DECIR NOMBRE ...) de la forma típica. No explicaré como crear estas entradas ya que en el manual del PAW, o de cualquier otro parser que uséis, está bastante bien explicado.

EL LIMITE DE LA IMAGINACION. TRUCOS CON EL PAW

Me decido a escribir este artículo a la vista del poco dominio que tienen los actuales programadores del PAW, aunque yo lo achaco más a su falta de imaginación. Para aclararnos de una vez, me estoy refiriendo a la forma de hacer las Salidas con el PAW, que siempre se dice que no se puede, que si el PAW no tiene un condacto exclusivo para eso, que si tumba, que si taramba... ...pues no señor, lo que hay que tener es un poco de coco y usarlo para pensar. Bien es cierto que el PAW no tiene un condacto propio para poder hacer las salidas, pero sí que lo tiene para hacer el INVENTARIO, que para el caso es lo mismo (Varios nombres perfectamente puntuados con , e y). Todo lo que tenemos que hacer es definir las salidas como objetos, ponerlas en una localidad que no usemos y hacer un inventario de esa localidad. Esta es la explicación en bruto, ahora vayamos paso por paso:

MENSAJES:

```
1: "Las salidas visibles son "
2: "La única salida visible es "
3: "No hay ninguna salida visible. "
4: ". "
```

MENSAJES DEL SISTEMA:

```
46: ", "
47: "y "
48: ""
```

OBJETOS:

```
30: Sur
31: Este
32: Oeste
33: Norte
    ...Así hasta poner una salida en cada objeto.
```

BANDERA 100 LIBRE

RESPUESTAS

```
SALID _     NOTZERO 0      Está oscuro
            ABSENT ZERO    No hay fuente de luz
            SYSMESS 0      "No se ve nada. "

SALID _     PROCESS 3      Llama al Proceso 3
            DONE
```

PROCESOS 3:

```
_     _     COPYFF 38 100  Copiamos nuestra posición en
                           la bandera 100
            LET 33 2       Ponemos como verbo de la frase
                           actual Sur
            MOVE 100       Intentamos mover la bandera 100
            NOTZERO 100    Si se ha conseguido mover
            CREATE 30      Crea el objeto Sur
            PUTO 6         Lo pone en la localidad 6
            PLUS 100 1     Añade 1 a la bandera 100

_     _     COPYFF 38 100  Copiamos nuestra posición en
                           la bandera 100
            LET 33 3       Ponemos como verbo de la frase
                           actual Este
            MOVE 100       Intentamos mover la bandera 100
            NOTZERO 100    Si se ha conseguido mover
            CREATE 31      Crea el objeto Este
            PUTO 6         Lo pone en la localidad 6
            PLUS 100 1     Añade 1 a la bandera 100

            ...Así hasta acabar con todas las salidas
```

```
_    _    EQ 100 0
          MES 3          "No hay ninguna salida. "
          DONE

_    _    EQ 100 1
          MES 2          "La única salida visible es "

_    _    NOTEQ 100 1
          MES 1          "Las salidas visibles son "

_    _    LET 51 255     Otro día os explico porqué
          LISTAT 6       Hace inventario de la localidad 6
          MESSAGE 4      ". "
          DESTROY 30     Destruye el objeto Sur
          DESTROY 31     Destruye el objeto Este
          DESTROY 32     Destruye el objeto Oeste
               ...Igual con las demás salidas-objetos
          CLEAR 100      Borra el valor de la bandera 100
          DONE
```

Y ahora para hacer que cuando tecleamos una dirección errónea nos lo diga y nos escriba las salidas posibles:

MENSAJES:

```
5: "No puedes ir en esa dirección. "
```

AL FINAL DE LA TABLA DE RESPUESTAS:

```
_    _    GT 33 13       El verbo actual no es de
                        movimiento
          SYSMESS 8      "No puedes hacer eso. "
          DONE

_    _    MOVE 38        Como el verbo actual sí es de
                        movimiento, intenta mover al
                        jugador
          DESC           Describe la localidad

_    _    MES 5          "No puedes ir en esa dirección. "
          PROCESS 3      Llama al Proceso 3 para que
                        escriba las salidas posibles
          DONE
```

Y esto es todo, seguro que más de uno se habrá subido por las paredes , pero la cosa no era tan difícil, y mucho menos imposible. (Ahora por lo menos os ahorraréis un montón de mensajes, como por ejemplo PEDRO AMADOR LOPEZ en JOHNNY VAYNA, que lo hizo poniendo un mensaje de salidas para cada una de las localidades). Si nuestro querido director (eso va por tí, Juanjo) me deja seguir con esta sección, en el próximo artículo podríamos ver cómo diferenciar los objetos con artículo definido (EL, LA) e indefinido (UN, UNA) para cuando los llevamos o los vemos en una localidad ("Llevas UN rifle. ") , o cuando los manipulamos ("Dejas EL rifle. ").

Programación con SINTAC | **Trucos con el PAW** | Línea de ayuda telefónica

DUDAS SOBRE EL PAW

Por... Javier San José

JULIO BAYONA BERISO

1.- Cuando introduzco palabras en el vocabulario me surge un problema; introduzco el verbo ENCENDER, hasta ahí todo bien, pero necesito introducir también el adjetivo ENCENDIDO y cuando lo intento me pone que esa palabra ya está en el vocabulario. ¿Cómo puedo solucionar esto?.

Me temo que no se puede solucionar de ninguna forma ya que el PAW no admite dos palabras iguales, cuyas 5 primeras letras coincidan, en el vocabulario.

2.- ¿Se puede poner un mensaje antes de que aparezca la descripción de la pantalla y que al apretar una tecla aparezca ésta (la descripción) debajo del mensaje?.

La forma de hacer esto que tú quieres es la siguiente:

- la descripción de la localidad 0 se usará como pantalla introductora (copyright, créditos, etc...).
- en Procesos 1 colocaremos la siguiente entrada:

```
*    *    AT        0
          ANYKEY
          CLS
          MODE      1    3
          MESSAGE   0
          ANYKEY
          GOTO      1
          DESC
```

Siendo el mensaje 0 el que quieres imprimir antes de la descripción de la primera localidad; en este caso la 1 (GOTO 1). El MODE 1 3 es necesario para que el DESC no borre el mensaje que hemos puesto (MESSAGE 0), después de pulsar una tecla. Si vas a utilizar otro modo de pantalla añade otra entrada en Procesos 1 de la forma:

```
*    *    ZERO      100
          SET       100
          MODE      0    3
```

Aquí se usa la bandera 100 para indicar que el modo debe cambiarse sólo una vez; al inicio del juego estará a 0 por lo que comprobamos esta condición (ZERO 100) tras lo cual la ponemos a un valor distinto de cero (SET 100) y ajustamos el modo de pantalla (MODE 0 3). Puedes colocar el modo de pantalla que más se ajuste a tus necesidades, además puedes usar otra bandera en vez de la 100.

PREGUNTAS Y RESPUESTAS SOBRE EL PAW

Por... Javier San José

JUAN ANTONIO PEREIRA VARELA

1.- En el juego ABRACADABRA aparece una opción cuya función es la de repetir la última acción realizada al pulsar la tecla V. ¿Cómo podría conseguir este efecto?.

En primer lugar para poder repetir la última frase tecleada habrá que guardarla en algún sitio. Debemos guardar el contenido del las banderas 33, 34, 35, 36, 43, 44, 45 que corresponden al verbo, nombre 1 y 2, adjetivo 1 y 2, adverbio y preposición. Para ello usaremos las banderas 16 a 22. Al inicio de la tabla de respuestas deberemos añadir las siguientes entradas:

Una entrada para guardar la última sentencia lógica. XX representa el número dentro del Vocabulario del verbo V que será el que provoque que se repita la última sentencia introducida:

```
*   *   NOTEQ     33   XX
        COPYFF    33   16
        COPYFF    34   17
        COPYFF    35   18
        COPYFF    36   19
        COPYFF    43   20
        COPYFF    44   21
        COPYFF    45   22
```

Y otra entrada para recuperar la última sentencia guardada:

```
V     _     COPYFF    16   33
            COPYFF    17   34
            COPYFF    18   35
            COPYFF    19   36
            COPYFF    20   43
            COPYFF    21   44
            COPYFF    22   45
            DESC.
```

Con esta sencilla técnica conseguimos los efectos deseados.

2.- El control del cursor que permite el PAW es, a mi entender, bastante flojo, ya que si tecleas una respuesta larga y antes de darle a INTRO te das cuenta de que has cometido un error, tienes que borrar la frase hasta donde esté el fallo. ¿Podrías proporcionarme alguna rutina de control del cursor?.

Siento decirte que no, y la razón es bien sencilla: habría que modificar en gran medida el propio PAW y ello sin listados de código fuente comentados es una labor bastante difícil. Por tanto deberás conformarte con el cursor que proporciona el PAW.

LUIS VICENTE ARELLANO

1.- Sé que hay una bandera que controla el input, pero ¿hay alguna que controle el ANYKEY?

El único control que disponemos del condacto ANYKEY es a través de la bandera 49 (que es la misma que controla el input). Además es un control bastante limitado ya que sólo podemos decir si queremos que el tiempo muerto (TIMEOUT) pueda ocurrir en medio de un ANYKEY o no. Esto se consigue mediante el bit 2 de esta bandera: seteado permite que el tiempo muerto ocurra en medio de un **ANYKEY.**

LUIS GUILLERMO DEL CORRAL MAZORRA

1.- ¿Cómo puedo conseguir que cuando pido un inventario, examino un contenedor o entro en una localidad el inventario en vez de aparecerme en columna me aparezca en fila?

Mediante la sentencia LET 53 64 conseguirás que los listados de objetos aparezcan en fila.

2.- Quiero saber como hacer que en la aventura se escriba en minúsculas cuando escribes al querer hacer algo y como pasar a minúsculas los sets de caracteres del PAW, a parte de como hacer que cuando cambio de set la O no cambie a %.

Respecto a escribir en minúsculas en mitad de la aventura es imposible (a menos que modifiques el

PAW entero) ya que el PAW sólo admite mayúsculas. Lo de pasar a minúsculas los sets de caracteres del PAW no entiendo porque lo preguntas ya que TODOS LOS SETS DEL PAW TIENEN TANTO LAS MAYUSCULAS COMO LAS **MINUSCULAS DEFINIDAS.**

Para que al cambiar de set la O (supongo que te refieres a la O acentuada, es decir 'ó') no se transforme en el '%' debes evitar usar el set de caracteres 0 si usas vocales acentuadas ya que este no se puede redefinir y la posición de la O acentuada la ocupa el '%'. Si te ocurre con cualquier otro set deberás modificar en ese set, mediante el editor de caracteres, el carácter '%' y colocar en su lugar una O acentuada.

ROBERTO BERNARDO

1.- Uno de los problemas es que al grabar en cinta una aventura hecha con el PAW, la grabación se hace en 3 partes: la primera el intérprete, la segunda la "Base-A" y la tercera la "Base-B". De esta manera, cualquier persona que adquiera la aventura puede cargar los dos últimos bloques en el PAWInt érprete, o investigar su interior. Para solucionar el problema he conseguido unir las 3 cargas en una sola gracias a un monitor de código máquina pero ahora no sé en qué dirección hacer el RANDOMIZE para ejecutar la aventura.

Para juntar los 3 bloques en uno sólo deberás hacer lo siguiente:

- CLEAR 28671
- Cargar el primer bloque (intérprete) y realizar los siguientes pokes: POKE 34902,0: POKE 34903,0: POKE 34904, 0
- LOAD "" CODE: LOAD "" CODE para cargar bases de datos A y B.
- Grabarlo todo junto con: SAVE "nombre" CODE 28672,36863

Para cargarlo usa este pequeño programa en BASIC:

```
10 CLEAR 28671
20 LOAD "" CODE
30 RANDOMIZE USR 34793
```

2.- Otro problema es que no se en que dirección van los overlays del PAW, y tengo varios programas en CM que pueden servir de overlays. Me gustaría que me proporcionara también esta dirección.

Cierto es que los overlays del PAW son pequeños programas en CM que se cargan a medida que se necesitan. También es cierto que los programadores del PAW han empleado un formato especial a la hora de escribir estos programas y dudo mucho que los programas en CM que tienes se ajusten a este formato. De todas formas es raro que necesites usar esos programas como overlays a menos que de alguna forma modifiquen la base de datos en sí.

Técnicas de Programación de Aventuras | **Preguntas sobre el PAWS** | Feedback

DUDAS SOBRE EL PAW

Por... Javier San José

ALEJANDRO FERNANDEZ MONTE

1.- ¿Qué he de hacer para animar imágenes?

El PAW no es un sistema que esté especialmente preparado para la animación de imágenes. No obstante utilizando unas sencillas técnicas podemos conseguir animación. Todo se basa en el uso de los UDGs y los juegos de caracteres. En ellos definiremos las secuencias de gráficos que componen la animación. Posteriormente compondremos, en mensajes, los bloques gráficos adecuados. Cada bloque que forme parte de la animación en un mensaje. Después lo único que hay que hacer es repetir cíclicamente, en la misma posición de pantalla estos mensajes, de la siguiente forma:

```
_    _  PRINTAT   0   0
        MES       mensaje1
        PAUSE     10  (esta pausa depende de la velocidad de animación)
        PRINTAT   0   0
        MES       mensaje2
        PAUSE     10
        ...
```

2.- Si en un gráfico se ve un baúl, ¿qué puedo hacer para que se vea abierto o cerrado?.

Dibuja un gráfico del baúl sin tapa y designalo como subrutina gráfica. Luego crea otros dos gráficos (en localidades que no uses) en los que meterás: en uno el gráfico del baúl con la tapa cerrada y en otro el baúl con la tapa abierta. En algún lugar del programa (proceso 1, por ejemplo) coloca lo siguiente; suponiendo que si bandera 100 igual a 0, el baúl está cerrado, y en otro caso está abierto

```
_    _  AT        localidad del baúl
        ZERO      100
        PICTURE   num. dibujo baúl cerrado

_    _  AT        localidad del baúl
        NOTZERO   100
        PICTURE   num. dibujo baúl abierto
```

CARLOS ALBERTO PARAMIO DANTA

1.- Quisiera que al hacer un inventario o simplemente al entrar en una localidad en la que hay objetos que éstos se listaran seguidos uno de otro y no en columna. Y además que el inventario separe lo que llevas de lo que llevas puesto.

Para que el PAW liste los objetos uno detrás de otro debemos colocar en la bandera 53 el valor 64, con ello los condactos LISTOBJ y LISTAT listarán un objeto detrás de otro separados por comas. Ahora bien, misteriosamente el condacto INVEN hace caso omiso de esta bandera. Para solucionar esto, y además que nos aparezcan por separado los objetos puestos de los llevados, ahí va una entrada que debes colocar en Respuestas:

```
_    I  LET       53  64
        SYSMESS   9
        LISTAT    254
        SYSMESS   10
        LISTAT    253
```

Debes modificar el mensaje del sistema 10 para que ponga algo similar a: "Llevas puesto:"

2.-¿Cómo puedo hacer para que al pedir las salidas el PAW te diga las posibles, y que al escribir una dirección a la que no puedes ir te diga que no puedes ir allí y te recuerde las salidas posibles?

Un proceso que te permite imprimir las salidas posibles es el siguiente:

```
*    *  NOTZERO   0        ; si está oscuro no imprime salidas
        ABSENT    0
        DONE
```

```
*     _   MES        11        ; "Salidas:"

_     _   COPYFF     38  11
          LET        33  2     ; NORTE tiene número 2 en vocabulario
          MOVE       11
          NOTZERO    11
          MES        3         ; Mensaje 3="Norte "

_     _   COPYFF     38  11
          LET        33  3     ; SUR tiene número 3 en vocabulario
          MOVE       11
          NOTZERO    11
          MES        4         ; Mensaje 4="Sur "
          .                    ; colocar una entrada para cada dirección
          .                    ; este, oeste, noreste,...
          .
```

En la tabla de Respuesta colocaremos una entrada como la siguiente:

```
_     SALIDAS     PROCESS  x
                  DONE
```

Donde x es el número de Proceso dónde hayamos metido lo anterior. Ten en cuenta que las conexiones que hayas colocado en la tabla de Respuestas no saldrán reflejadas en pantalla al llamar al anterior procedimiento. En cuento a la segunda parte de tu pregunta, siento decepcionarte pero no ha encontrado ningún método de llevarlo a cabo con el PAW.

Ofertas | **Dudas sobre el PAW** | Comentario de Espejos

DUDAS SOBRE EL PAW

Por... Javier San José

RAUL MORENO MERINO

1.- ¿Se pueden hacer inventarios en los que salgan los objetos separados por comas y los últimos por 'y'?

En el manual, como ya habrás observado, viene un modo de hacer que los objetos que hay en una localidad salgan en una lista continua, separados por comas, en vez de uno debajo de otro. Esto se consigue poniendo un LET 53 64, pero tiene la desventaja de que sólo funciona con los condactos LISTOBJ y LISTAT, no con INVEN. Por tanto, para sacar un inventario en el que salgan los objetos seguidos habrá que usar LISTAT en vez de INVEN. Sabiendo que LISTAT saca una lista de los objetos que haya en cierta localidad, y que la localidad 254 es la correspondiente al los objetos que lleva el jugador, y la 253 la correspondiente a los objetos que lleva puestos, habrá que colocar en la tabla de Respuestas una entrada de la forma:

```
I     _     LET       53   64
            SYSMESS   9
            LISTAT    254
            SYSMESS   10
            LISTAT    252
```

Y modificar los mensajes del sistema 9 y 10 para que contengan:

```
 9: Llevas:
10: Tienes puesto:
```

Con esto conseguirás un inventario de la forma que deseas.

2.- Quiero poner un marcador en la esquina superior derecha con los turnos y el porcentaje. ¿Cómo podría hacerlo?.

El mayor problema que se presenta es el mantener la posición de impresión que había antes de imprimir el marcador; ya que si imprimimos sin más el marcador, debajo nos saldrían los mensajes de nuestra aventura, 'machacando' lo que hubiese en pantalla. Para evitar este tipo de situaciones en el que queremos imprimir algo en una posicón específica de pantalla y luego volver al punto de impresión dónde estábamos, el PAW nos provee de dos potentes condactos: SAVEAT y BACKAT. SAVEAT guarda en memoria la posición actual de impresión y BACKAT la recupera. Estos condactos sólo nos permiten guardar una posición de impresión, así si hacemos dos SAVEAT, será la segunda posición de impresión la que se guarde. Para imprimir un marcador en la zona superior de la pantalla, por tanto, nos valdremos de estos dos condactos. Como lo que queremos imprimir son los turnos y el porcentaje de aventura resuelto, nos valdremos de los condactos TURNS y SCORE. Pero primero modificaremos un poco los mensajes del sistema para que se ajusten a nuestras necesidades. Los mensajes del sistema a modificar son:

```
17: (vacio)
18: []turno
19: s
21: (vacio)
22: %
```

Donde (vacio) indica que el correspondiente mensaje no debe contener nada, y [] indica que debemos dejar un espacio. Luego en Procesos 2 añadiremos una entrada de la forma:

```
*     *     SAVEAT
            PRINTAT   0    17
            TURNS
            PRINTAT   0    28
            SCORE
            BACKAT
```

IMPORTANTE: Al principio de cada descripción de localidad habrá que dejar una línea en blanco, que será donde se impriman los marcadores.

J. DANIEL CARBONELL COB

1.- Quiero hacer que aparezca un gráfico en la pantalla y para ello voy a Procesos 1 y pongo la

entrada: * * AT x PICTURE xy, siendo x el número de localidad y xy el número del gráfico, (que en realidad son el mismo número). De esta forma aparece el gráfico, pero a la hora de poner el texto en la parte que quiero no lo consigo.

Cuando se crea una localidad, como sabes, en la tabla de gráficos se añade una entrada por si queremos colocar un gráfico en esa localidad. Pero estas entradas, de inicio, el PAW las marca como subrutinas (gráficos que pueden llamarse desde otros gráficos y que no se dibujan cuando estamos en la localidad a la que corresponden). Si quieres ver esto elige la opción D del menú principal y luego P. Para que estos gráficos se dibujen al entrar en la localidad debemos decirle al PAW que no son subrutinas. Esto se hace también con la opción D del menú principal y luego tecleando: A graf papel tinta, siendo 'graf' el número del gráfico, y 'papel' y 'tinta' los atributos por defecto para ese gráfico. Después en Procesos 1 añadiremos una entrada de la forma:

```
*       *       GRAPHIC   2
                MODE      3     3
                LINE      6
```

Esto si quieres un gráfico que llegue hasta la linea 6 de pantalla, a partir de la cual se imprimirá la descripción de la localidad, y además el gráfico no hará scroll. Si lo que quieres es un gráfico que no permanezca en pantalla, es decir, que haga scroll cunado la pantalla se llene de texto, deberás poner:

```
*       *       GRAPHIC   2

                MODE      2     3
                LINE      6
```

JOSE COLETES CAUBET

Este amable socio nos envía un truco que será util a todos los usuarios del PAW. Consiste en lo siguiente:

'Cuando entres en una localidad pulsa SYMBOL SHIFT y la tecla Y, y aparecerá el mensaje 35 del Sistema (el del tiempo pasa). Hay que tener en cuenta que las banderas de turnos no se mueven, y además se ejecutará cualquier TIMEOUT que esté en Procesos 2 (siempre que esté preparado para la situación en que estás).'

Añadiré algo más al truco que nos envía José Coletes, y es que éste sólo funciona si no se ha tecleado nada todavía en la línea de 'input' del jugador. El truco, como observaréis, es muy útil para aquellas aventuras en las que hay que esperar un número de turnos para que ocurra algo.

El Mundo del Rol | **Dudas sobre el PAWS** | Soluciones Serializadas

TECNICAS AVANZADAS DE PROGRAMACION DE AVENTURAS

Por... Javier San José

A partir de este capítulo, y en adelante, iré exponiendo, paso a paso un sistema que aprovecha al máximo las posibilidades del PAW para la creación de una aventura multijugador.

Pero, ¿qué es una aventura multijugador?; pues se trata simplemente de una aventura en la que podemos controlar a dos o más personajes simultáneamente. Las posibilidades son infinitas pero exige un esfuerzo mayor por parte del programador ya que, si una aventura con un sólo personaje jugador es complicada de hacer, una aventura multijugador multiplica proporcionalmente la dificultad.

El sistema propuesto consta de una serie de Procesos. Recomiendo a los que estéis interesados en ir incorporando este sistema a vuestras aventuras que respetéis los números de los procesos que iré dando y que los coloquéis en el mismo orden; esto evitará problemas.

Una serie de conceptos antes de empezar. Utilizaré a cuatro personajes jugadores, número fácilmente modificable, que denominaré genéricamente Guerrero, Enano, Mago y Sacerdotisa. Cada uno vendrá definido por una serie de parámetros:

- energía vital, indica la resistencia del personaje a las heridas. Sirve para los combates en los cuales un golpe certero disminuirá esta puntuación; así en como otras situaciones (fuego, caídas, venenos,...)
- fuerza de ataque, indica el daño que causa ese personaje en combate cuerpo a cuerpo.
- posición, es la localidad actual del personaje
- raciones, cantidad de comida que lleva encima
- dinero, cantidad de ídem que lleva encima
- localidad de objetos llevados y localidad de objetos puestos, son dos localidades auxiliares (dos para cada personaje) usadas cuando se intercambia entre un personaje y otro

Además, el Mago, viene definido por otros dos parámetros:

- energía mágica, se gasta cuando se realizan hechizos
- nivel mágico, índice de la sabiduría arcana del personaje

La mayoría de estos parámetros ayudarán a implementar un sistema de combates a base de tiradas de dados. Pero eso más adelante. La tabla siguiente resume los valores para todos los personajes:

	Guerrero		Mago		Enano		Sacerdotisa	
	Band	val	Band	val	Band	val	Band	val
Energía vital	66	50	68	25	70	60	72	35
Fuerza ataque	--	2	--	0	--	1	--	0
Posición	67	9	69	9	71	9	73	9
Raciones	74	2	76	0	78	1	80	0
Dinero	75	?	77	?	79	?	81	?
Loc.obj.llevados	1		3		5		7	
Loc.obj.puestos	2		4		6		8	
Energía mágica	--	--	88	200	--	--	--	--
Nivel mágico	--	--	89	1	--	--	--	--

Como se ve las banderas usadas, de momento, son de la 66 a la 81 y las banderas 88 y 89. Las localidades reservadas van de la 1 a la 8. Otra bandera que usaremos será la número 23: número de personaje actual (1=Guerrero, 2=Mago, 3=Enano, 4=Sacerdotisa).

Podemos empezar insertando unas entradas en el Proceso 1 para imprimir mensajes de que el Enano está aquí, el Mago está aquí, etc... para cuando controlemos a un personaje y alguno de los otros esté en nuestra misma localidad. Las entradas a añadir serían:

```
_    GUERRERO        NOTEQ      23    1
```

```
                    SAME      67   38
                    MES       'El '
                    MES       'Guerrero'
                    MESSAGE   ' está aquí.'
_     MAGO          NOTEQ     23   2
                    SAME      69   38
                    MES       'El '
                    MES       'Mago'
                    MESSAGE   ' está aquí.'
...
```

Sólo se incluyen, por motivos de espacio, las entradas correspondientes al Guerrero y al Mago; las del Enano y la Sacerdotisa su pueden deducir de estas fácilmente.

Explicaré el funcionamiento de estas entradas. Primero comprobamos que el jugador no está controlando al personajes (NOTEQ 23 1 para el Guerrero, NOTEQ 23 2 para el Mago, etc...). Esto es así para que no se imprima el mensaje 'El Guerrero está aquí.' cuando el jugador esté controlando al Guerrero, por ejemplo.

Luego miramos si el personaje está en la misma localidad que el que controla el jugador, comparando la bandera de posición del personaje con la localidad actual (bandera 38). Así: SAME 67 38 para el Guerrero, SAME 69 38 para el Mago,...

Si se dan estas dos condiciones se imprime un mensaje del tipo 'El ... está aquí.'. (NOTA: en el caso de la Sacerdotisa el mensaje a imprimir sería del tipo 'La ... está aquí.', evidentemente).

Ahora vamos a insertar un proceso que inicialice las banderas de los personajes:

-Proceso 9:

```
_     GUERRERO      LET       66   50
                    LET       67   9
                    LET       74   2
_     MAGO          LET       68   25
                    LET       69   9
                    LET       88   200
                    LET       89   1
_     ENANO         LET       70   60
                    LET       71   9
                    LET       78   1
_     SACERDOTISA   LET       72   35
                    LET       73   9
```

Este proceso inicializa las banderas correspondientes a la energía vital, fuerza de ataque, posición, raciones, dinero, energía mágica y nivel mágico de los personajes. Obsérvese que la localidad de inicio elegida para todos el la número 9.

En Proceso 1 colocaremos una llamada a este proceso en la secuencia de inicialización, de la forma:

```
_     *             AT        0
                    (aquí establecer modo pantalla, ABILITY,...)
                  ANYKEY
                  PROCESS   9
                  LET       23   1
                  GOTO      9
                  DESC
```

El LET 23 1 hace que le personaje que controlamos de inicio se el Guerrero. Como todos los personajes comienzan en la localidad 9, hacemos un GOTO 9 seguido de un DESC.

De momento esto es todo por ahora. En la próxima entrega veremos cómo cambiar de un personaje a otro dentro del juego.

Opinión | **Técnicas de Programación** | Línea de Ayuda Telefónica

Técnicas de Programación de Aventuras

TECNICAS AVANZADAS DE PROGRAMACION DE AVENTURAS

Por... Javier San José

En esta nueva entrega veremos los procesos que nos permitirán cambiar de un personaje a otro:

- Proceso 15:

```
_    *    DOALL   1
          WHATO
          PUTO    254
```

- Proceso 16:

```
_    *    DOALL   2
          WHATO
          PUTO    253
```

Estos dos procesos sirven para que cuando cambiemos al Guerrero, los objetos que llevaba figuren como cogidos o puestos. Así, pasamos los de la localidad auxiliar 1 a llevados y los de la localidad auxiliar 2 a puestos. Habrá que hacer otros dos procesos para cuando pasemos del Guerrero a otro personaje:

- Proceso 17:

```
_    *    DOALL   254
          WHATO
          PUTO    1
```

- Proceso 18:

```
_    *    DOALL   253
          WHATO
          PUTO    2
```

De esta forma cuando cambiemos del Guerrero a otro personaje, los objetos del primero se guardarán en las localidades auxiliares, mientras que los del segundo pasarán de sus localidades auxiliares a ser llevados o puestos. Los procesos 19 al 30 contendrán las correspondientes entradas para el resto de los personajes. Así:

- Proceso 19:

```
_    *    DOALL   3
          WHATO
          PUTO    254
```

- Proceso 20:

```
_    *    DOALL   4
          WHATO
          PUTO    253
```

- Proceso 21:

```
_    *    DOALL   254
          WHATO
          PUTO    3
```

- Proceso 22:

```
_    *    DOALL   253
          WHATO
```

```
                PUTO     4
```

Esto serían para el Mago. Igualmente haríamos otros tantos procesos para el Enano y la Sacerdotisa. Los procesos que nos permitirán cambiar de personaje son los siguientes:

- Proceso 31:

```
_       GUERRERO    EQ      24      1
                    LET     23      1
                    COPYFF  67      38
                    PROCESS 15
                    PROCESS 16
                    PROCESS 35
                    DESC
_       MAGO        EQ      24      2
                    LET     23      2
                    COPYFF  69      38
                    PROCESS 19
                    PROCESS 20
                    PROCESS 35
                    DESC
...
```

- Proceso 32:

```
_       GUERRERO    EQ      67      255
                    MESSAGE 'Ese personaje está muerto.'
                    DONE
_       MAGO        EQ      69      255
                    MESSAGE 'Ese personaje está muerto.'
                    DONE
...
_       _           EQ      23      1
                    COPYFF  38      67
                    PROCESS 17
                    PROCESS 18
_       _           EQ      23      2
                    COPYFF  38      69
                    PROCESS 21
                    PROCESS 22
...
```

La bandera 24 es usada como bandera de trabajo, y contendrá el número del personaje al que queremos cambiar. En el Proceso 31 comprobamos a qué personaje se quiere cambiar (EQ 24 1 si queremos cambiar al Guerrero). Una vez que sepamos a cuál de los personajes queremos cambiar, colocamos su número en la bandera 23 (LET 23 1 si cambiamos al Guerrero), nos colocamos en la localidad de ese personaje (COPYFF 67 38 en el caso del Guerrero, lo cual pone la localidad del Guerrero como localidad actual) y por último ponemos sus objetos como llevados o puestos (PROCESS 15, PROCESS 16 para el Guerrero).

El Proceso 35 es un proceso implicado en el manejo de localidades oscuras (se explicará más adelante) y sólo se usará en caso de que haya alguna de estas. El Proceso 32 es el que deberá ser llamado desde la Tabla de Respuestas. EN él miramos si el personaje al que queremos cambiar está muerto (EQ 67 255 para el Guerrero) y si es así imprimimos un mensaje y salimos del proceso.

Un personaje muerto tiene como número de localidad la 255. Luego comprobamos el personaje actual guardando su localidad y sus objetos en las localidades auxiliares, y finalmente llamamos al Proceso 31 para seleccionar al nuevo personaje con lo que completamos el proceso. En la Tabla de Respuestas aparecerá una entrada de la forma:

```
CAMBIAR _           GT      34      25
                    LT      34      30
                    COPYFF  34      24
                    MINUS   24      25
                    NOTSAME 23      24
                    PROCESS 32
                    DONE
```

Esta entrada lo primero que hace es comprobar si el nombre del personaje al que queremos cambiar es el nombre de uno de los personajes, es decir, si está entre el 26 y el 29 (inclusives), que son los números de los nombres de los personajes en el vocabulario:

```
26 ->   Guerrero
27 ->   Mago
28 ->   Enano
29 ->   Sacerdotisa
```

Se recomienda que se definan como nombres propios y con números consecutivos siendo el Guerrero el primero, el Mago el segundo, y así sucesivamente. Una vez seguros de que es el nombre de uno de los personajes lo copiamos en la bandera 24 y le restamos 25 para que nos quede un número entre el 1 y el 4. Miramos después que el personaje al que queremos cambiar no sea el que manejamos (NOTSAME 23 24) y si no lo es lo cambiamos (PROCESS 32) completando así el proceso.

Y hasta aquí llega esta entrega. En la próxima veremos más cosas sobre control de múltiples personajes.

Petra	**Técnicas de Programación de Aventuras**	Preguntas sobre el PAWS

TECNICAS DE PROGRAMACION

LOS PERSONAJES PSEUDO-INTELIGENTES (IV)

Si en la anterior entrega enumerábamos las características más envidiables de los PSI mejor formados en toda la historia de las conversacionales asistidas por ordenador, comenzamos rápidamente en este nuevo ejemplar escribiendo poderosas rutinas para llevar a la práctica todos y cada uno de los apartados descritos.

I. LA MOVILIDAD

¡Qué maravilla observar cómo tus creaciones deambulan de una lado a otro, yendo y viniendo en un fastuoso alarde de habilidad programatoria!, ¡y qué regocijo al imaginar, siquiera, la horrenda frustración del jugador que necesita con urgencia la colaboración de uno de tus PSI, y que se recuesta sobre su asiento al pensar que el maldito puede estar en CUALQUIERA de las múltiples localidades y en constante movimiento! En efecto muchachos, es éste uno de los grandes atractivos de la aventura...

Para una rutina de movimiento aleatorio os remito al número 7, donde ya escribimos una que cumplía perfectamente con esta función. Sin embargo, nos resta por comentar algunos que otros tipos de movimiento igualmente interesantes. Por ejemplo, en innumerables ocasiones os habréis visto en la obligación de teclear algo así como DILE AL FANTASMA "SIGUEME". ¿Qué ocurre cuando tecleamos algo así? Una entrada en Respuestas se encarga de llamar a una tabla de Procesos ocupada en albergar a un gran pedazo de nuestro PSI -visto en números anteriores- pues bien, es en esa tabla donde incluiréis una entrada más o menos así:

```
SIGUEME   _     LT          11    1
                SET         11
                LET          5   10
                MESSAGE "El fantasma asiente"
                DONE
```

SIGUEME _ MESSAGE "¡Estoy en ello!"

DONE

Y en Procesos 2:

```
SIGUEME   _     GT          11    1
                NOTZERO      5
                NOTSAME (flagpsi) 38
                COPYFF      38 (flagpsi)
                MESSAGE "El fantasma aparece"
```

Y más bonito aún, también en Procesos 2:

```
SIGUEME   _     EQ           5    1
                CLEAR       11
                MESSAGE "El fantasma dice: No veo porqué he de seguirte más"
```

Ya lo tenemos. Veamos ahora qué ocurre exactamente. En la tabla de Procesos, la orden PARSE interpreta el mensaje entre comillas que dice "SIGUEME" (sinónimos aconsejados: SIGUE, SEGUI), y entonces encuentra nuestra primera entrada. Si el PSI no está siguiendo al jugador (11=0), se pone a ello (11=255) y lo hace saber. Si lo sigue, ignora la primera entrada (porque 11 es diferente de 0) y también se lo comunica al jugador. Habiendo cambiado el valor de 11, o habiendo dicho que el PSI ya está siguiendo al aventurero, lee luego Procesos 2 y se encuentra con que si a) 11 es distinto de 0 (está siguiendo al jugador), b) 5 no es 0 (otra comprobación), c) el PSI no está en la misma localidad que el jugador, entonces copia el contenido de la bandera 38 en la bandera que lleve la localidad del PSI, lo que hace que PSI=Jugador, y lo hace saber.

Si alguna de estas comprobaciones falla, no sale de la tabla, si no que sigue buscando otras entradas -un programa aún debe hacer muchas comprobaciones, ajustes, cambios, etc-, y si además, la bandera 5 tiene el valor 1, se da por terminado el acto de seguimiento, por así decirlo, restaurando el valor de 11 a 0. El jugador está en postura, por supuesto, de volver a teclear SIGUEME al PSI para que todo vuelva a empezar.

Acordaos de desactivar la rutina de movimiento aleatorio cuando el PSI esté siguiendo al jugador, mediante un simple condacto al principio de la entrada principal (estaba en P2) que chequee que 11

sea 0.

Otro tipo de movimiento, cuando simplemente sea interesante que el PSI se mueva en una dirección determinada si así se lo indica el jugador. DILE AL FANTASMA "VETE AL NORTE" (como muy bien habéis pensado, el ordenador entiende una simple N).

En la tabla de Procesos asignada:

```
_        _   (insertar en   LT        34   14
              posición 0)   MOVE (flagpsi)
                            MESSAGE "El fantasma se marcha"
                            DONE

_        _   (insertar en   LT        34   14
              posición 1)   MESSAGE "El fantasma no puede ir en esa dirección"
                            DONE
```

Muy bien. Esta es una entrada sencilla que, además, la tenéis comentada en el libro Guía de Referencia que trae consigo el PAW, por lo que no le daremos mayor importancia.

Algo que aporta un extraordinario encanto a un PSI es que sea capaz de, diligentemente, dirigirse por el camino más corto y rápidamente a un punto de destino. Ya hablamos de esto anteriormente, por lo que únicamente nos ocuparemos ahora de hacerlo realidad.

Se trata de hacer que un PSI vaya desde un sitio a otro pasando por un número variable de localidades. Es por esto que no escribiremos una rutina lista para ser tecleada, como hacemos siempre, si no que que explicaremos cómo funciona todo.

En primer lugar, debemos establecer cuando y dónde va a iniciarse la ruta. Sería algo así:

```
_        _   EQ      (flagpsi) 17
             LT          11     1
             SET         11
```

Aquí, en Procesos 2, esta entrada comprueba que el PSI esté en la localidad 17, lo cual es un modo tan bueno como cualquier otro de establecer una condición a modo de ejemplo. Vosotros, en vuestros propios programas, probablemente necesitéis alguna que otra comprobación -chequear el valor de una bandera, verificar que un objeto ha sido dejado, cogido, puesto, etc- pero esto no os supondrá -¡eso espero!- un problema. Ahora que el ordenador sabe que debe activar la ruta, haremos una llamada a una nueva tabla de procesos que contendrá exclusivamente nuestra ruta. En realidad, como sucedía en el movimiento aleatorio, también en P2, una si PSI=Jugador y otra para cuando éste no sea el caso.

```
_        _   GT          11     1
             SAME   (flagpsi) 38
             PROCESS     X
             SET         12

_        _   LT          12     1
             GT          11     1
             NOTSAME (flagpsi) 38
             PROCESS     Y

_        _   CLEAR       12
```

Examinemos esto. Aquí, la bandera 12 actúa como un control para que la segunda entrada de las tres que forman el llamamiento a la rutina de movimiento no se ejecute si la primera de ellas ya lo ha hecho. Más lógico sería poner un sencillo DONE en esta última, para que PAW ordenase una salida de la tabla, ¿no es cierto? Bien, la verdad es que no. Establezcamos ahora una especie de regla de oro: Por lo general, y salvo muy pocas excepciones, las tablas 1 y 2 jamás contendrán condactos que ordenen salidas de tablas, puesto que deben ser leídas enteras.

Luego, PAW envía al intérprete a leer la tabla de Procesos X si el PSI está en la misma localidad que el jugador, y la Y si no es así. Vosotros deberéis asignar el número de tablas que queráis. Veamos pues cómo sería:

```
_        _   EQ     (flagpsi) 17
             LET        33    13
             MOVE   (flagpsi)
             SAME   (flagpsi) 38
             MESSAGE "El fantasma aparece"
             DONE
```

La primera de estas dos entradas, la establecida en X comprueba que el PSI está en 17, intenta moverlo al norte (definida en el vocabulario como nombre de valor 13) y si lo consigue

(simplemente porque existe una salida al norte registrada en la tabla de conexiones en la localidad 17) se lo hace saber al jugador, y ¡voilá! nuestro fantasma ya está en otra localidad.

La que se encuentra en Procesos Y hace lo mismo, pero suponiendo que estuviésemos ya en la SIGUIENTE localidad, a donde va el PSI luego de haberse desplazado al norte, entonces nos diría que el fantasma aparece porque se movió al norte. Como veis, las dos entradas se complementan perfectamente. Para que el PSI se moviera desde la nueva localidad (digamos, 18) a una nueva que se encuentra, digamos también al este, habría que modificar los EQ (bandera PSI) 17 por un EQ (bandera PSI) 18, el LET por un LET 33 3 (por que ESTE es 3 en el vocabulario) y en el MESSAGE de X, decir que "el fantasma va al este". ¡Y ya está! El esquema de las localidades sería este:

```
--+ +-----+ +--
? +-|  18 +-|  ?
  | |     ++-+->
--+ +----++ +--
    +----++
    |  17||
    |    |
    +-----+
```

Pensad que los cambios que debéis hacer para que el PSI haga un recorrido mucho más completo son evidentes, pero acordaos de desconectar la rutina alterando el valor de la bandera que cumplía esta función (en nuestro ejemplo, la 11). En la próxima entrega trataremos el interesante tema del apartado 2, ¡la manipulación de objetos!

Carlos Sisí

Editorial | **Técnicas de Programación** | Preguntas y respuestas

TECNICAS DE PROGRAMACION

Los Personajes Seudo Inteligentes (V)

Tal y como quedó definido en su día, en esta ocasión nos atreveremos a perfilar una de las facultades más útiles y efectistas de los PSIs. Estamos hablando de la manipulación de objetos.

De hecho, uno de los detalles más comentados del clásico por excelencia entre las aventuras asistidas por ordenador (El Hobbit, por supuesto) es esa frustrante faceta de Gandalf que le mueve a quitarte un objeto en primera instancia, seguidamente a preguntarte qué es, y ya en el tercer turno, es cuando accede a devolvértelo. Si en medio de esta interesante secuencia (¡que yo mismo siempre he emulado en alguno de mis juegos!) se da el caso de que Gandalf se mueve en alguna dirección válida, se llevará el objeto consigo, con lo cual deberemos recurrir al diálogo para volver a recuperarlo (¡e imagináos, sólo por un momento, lo que ocurre cuando el objeto es la espada luminosa y el lugar el interior de las oscuras cavernas goblins!).

Nuestros lectores más avejentados recordarán que en el número 8 dibujamos una serie de útiles rutinas que permitían a un personaje coger, dejar, ofrecer y aceptar objetos siempre bajo las indicaciones del jugador. Recordarlas aquí de nuevo sería para muchos de vosotros algo reiterativo, y aquellos que no hayan podido disfrutar de aquella suscripción siempre pueden referirse al servicio de números atrasados.

Como casi todo el trabajo está hecho, podemos esforzarnos por escribir la secuencia arriba comentada. Se trata de un aprendizaje ciertamente interesante, pues en el método que emplearemos para representar semejante acción se esconden los rudimentos que en LA AVENTURA ORIGINAL daban vida al Terrible Pirata...

Siempre en Procesos 2:

```
_          FANTASMA  SAME      38         Bandera PSI
                     LT        5          1
                     WHATO
                     EQ        54         254
                     PUTO      (Número de localidad)
                     MESSAGE "El fantasma te quita _"
                     COPYFF    54         12
                     SET       11
                     LET       5          3

_          FANTASMA  EQ        5          2
                     LT        11         1
                     SAME      38         Bandera PSI
                     EQ        12         (Número de localidad)
                     MESSAGE "El fantasma dice '¿Qué es esto?'"

_          FANTASMA  SAME      38         Bandera PSI
                     EQ        5          1
                     LT        11         1
                     EQ        12         (Número de localidad)
                     MESSAGE "El fantasma te devuelve _"
                     PUTO 254

_          FANTASMA  CLEAR     11
```

Hmmm... con estas cuatro entradas todo debería funcionar correctamente. Incluso podéis eliminar cualquier condacto del tipo SET 11, LT 11 1 y CLEAR 11 (con lo cual, la última entrada desaparece) si introducís las entradas con un orden inverso, es decir, la tercera como 0, la segunda como primera y ésta como segunda... pero así, para comprender su funcionamiento, lo veréis más claro. ¡Recordad evitar los DONE y otros condactos similares que provoquen salidas de tabla cuando estéis en la 2 ó en la 1!.

Estudiemos ahora sus interioridades. En primer lugar, se comprueba que el fantasma y el jugador están ambos en el mismo lugar (SAME 38 Bandera PSI). Luego se verifica que la secuencia no ha sido ya inicializada (que 5 sea 0; aunque también es válido ZERO 5, que además consume menos bytes) y se procede a activarla, verificando que el objeto a "sustraer" está siendo actualmente llevado por el jugador. Se pone entonces en la localidad que hace las veces de contenedor del PSI y se informa de ello al jugador. Además, se setea la bandera de control 11 y se prevee que la secuencia entera ocupará tres fases (de ahí el valor 3 de la bandera del sistema auto-decrementable 5). Adicionalmente, se habitúa una bandera de trabajo que se activa en previsión de que el contenido de la bandera 54 pueda cambiar en medio de la secuencia.

Luego, se hacen una serie de comprobaciones para verificar que el PSI sigue teniendo en su poder el

objeto arrebatado al jugador (pudiera ser que, con el uso de las rutinas explicadas en el CAAD 8, el jugador hubiese decidido recuperar su objeto pidiéndolo oralmente -más o menos, se entiende- al PSI), y el segundo mensaje es impreso.

En tercer lugar, la nueva serie de comprobaciones rutinarias es hecha y el objeto es devuelto al jugador atendiendo a las mismas normas que las usadas para quitárselo. La bandera de control número 11, ya en la última entrada, sigue haciendo sus chequeos para que todo funcione como es esperable.

Ahora bien, si en lugar de preguntar -bastante estúpidamente, por cierto- "¿qué es esto?", nuestro PSI hubiese activado una ruta de movimiento como las descritas en nuestro número 11, ¡tendríamos un PSI incluso más potente y realista que el afamado Big Bag Pirate! (quien simplemente hacía su localidad 0, por así decirlo, y el objeto era colocado en su guarida, PERO NO **EXPERIMENTABA NINGUN VIAJE REAL HACIA SU GUARIDA).**

Estos son detalles meramente atmosféricos, pero tratándose de una aventura conversacional, ¡estaréis de acuerdo conmigo en que esto es lo más importante!

Si además de todo esto, y luego de que el pérfido y horrorosísimo ladrón hubiese depositado su botín en la guarida, activáis la rutina de movimiento aleatoria también ampliamente documentada en anteriores entregas, ¡el PSI se pondría a recorrer el área de juego hasta que encontrase al jugador de nuevo, momento en el cual le arrebataría otro objeto y vuelta a empezar!

Por cierto, ¿cómo hacer para que el PSI deje sus botines en la guarida...?

En Procesos 2:

```
_          LADRON     EQ    (Bandera PSI)   (Número localidad guarida)
                      PROCESS    X
```

Y en Procesos X (X es cualquier número de procesos libre):

```
DEJAR      _          WHATO
                      EQ         54         (Número localidad)
                      PUTO       255
                      MESSAGE "El ladrón deja _"
                      DONE

DEJAR      TODO       DOALL      (Número de localidad)
                      DONE
```

El orden de estas dos entradas no debería ser motivo de problemas, pero para un correctísimo funcionamiento haríais bien en emplazar la entrada DEJAR TODO inmediatamente antes de su precedente (por otro lado, no veo cómo podría hacerse de otro modo).

Si no os gusta que el mensaje "El ladrón deja _" salga con cada objeto que lleva, suprimid el MESSAGE de DEJAR _ e incluid, seguidamente, y después del DOALL en la otra entrada, un MESSAGE "El ladrón deposita su botín en el suelo.".

Pues esto es todo por esta vez, aventureros. En el próximo ejemplar seguiremos atendiendo fielmente el plan de trabajo que creamos juntos en el número 9. Hasta entonces, ¡feliz "adventuring"!

Carlos Sisí

Editorial | **Técnicas de Programación** | Soluciones Serializadas

TECNICAS DE PROGRAMACION

LOS PERSONAJES SEUDO INTELIGENTES (y VI)

Concluímos al fin la que ha venido siendo una larguísima serie dedicada al manejo y estudio de los PSIs. Hemos visto cómo se mueven a su antojo y libre albedrío, cómo son capaces de dialogar -o al menos, que muchos estáis con la idea de matizar en mente, responder a nuestros requerimientos-, recoger y manipular con total libertad de acción toda clase de objetos y cacharros, e incluso hemos planteado la forma de representarlos en el ordenador para conseguir una personalidad u otra, dependiendo del tipo de personaje que pensemos crear.

De todas estas cualidades, la más abstracta es sin duda la referida en último lugar. Se trata de una faceta de las personas que las hacen odiosas, divertidas, serias, controvertidas y etc, describiendo así un amplísimo y vasto espectro del que podemos sustraer, mediante el uso combinado de algunos elementos, PSIs tan prodigiosos como irrepetibles.

Es sin embargo bastante difícil determinar de qué modo podemos llevar esto a cabo. No hay ninguna rutina que defina el que un personaje cualquiera sea avaro, generoso o simplemente reticente a ceder sus pertenencias. Esto se consigue de un modo bien distinto, y es perfilando sutilmente los distintos módulos de creación de un PSI hasta conseguir que dé el efecto de ser de un modo u otro.

Por ejemplo, tenemos la rutina -ya presentada en anteriores entregas del CAADde tratamiento de objetos. Tal y como la expusimos, nuestro PSI siempre dejaría un objeto que se encontrase en su haber cuando así se lo pidiese el jugador. Un personaje desconfiado podría no querer abandonar algo que es legítimamente suyo sólo porque un extraño -presumiblemente- se lo indica de ese modo. Sería necesario alterar la rutina en cuestión y hacer que, en el caso de que determinados parámetros no fuesen válidos, el PSI se negara en redondo a dejar el útil con un mensaje parecido a: "¿Porqué debería yo dejar _?"

Esto puede parecer bastante lógico y, cuanto menos, obvio, pero desgraciadamente no sucede a menudo en las aventuras. Consideremos otro ejemplo. Esta vez, tomado de un juego popular aunque bastante novedoso -por lo que, quizá, algunos de vosotros pudiera no conocerlo-. Se trata de SCAPEGHOST, una sensacional producción británica donde la fuerza de los problemas está basada en la interacción con los muchos, muy sotisficados y muy creíbles personajes (¡a pesar del hecho de que sean fantasmas!)

Para empezar, podemos traer a escena a la figura del enterrador. Este personaje es en gran medida secundario, aunque su habilidad para aportar atmósfera al compendio de la aventura lo sitúan en los más altos papeles jamás interpretados por un PSI. Nos lo encontramos avanzando diligentemente hacia algún lugar específico del cementerio, hablando consigo mismo y portando una pala. Cuando llega allí donde se encuentra nuestra propia fosa, el individuo comienza a trabajar con toda naturalidad, musitando cierta clase de improperios hacia el tipo de obligaciones que posee, pero con un encanto tal que parece que estemos viendo una de esas películas americanas donde los borrachines mascan tabaco y hablan con la boca torcida.

Es tan grande el grado de realismo y detalle impreso en este personaje que, en algún momento, el sepulturero incluso decide descansar sobre su pala unos instantes, volviendo al trabajo algunos turnos después. Una vez que la fosa está debidamente cubierta, podemos asistir al prodigioso peregrinaje de este hombrecillo hacia su caseta de herramientas, lugar donde tiene lugar un desmesurado derroche de habilidades por parte de estos PSIs a los que no puedo, por menos, otorgarles todo tipo de calificativos favorables. El enterrador abre la puerta y se mete en la caseta. Allí, deja su pala dando a entender que su jornada ha acabado, y se reune con el supervisor del cementerio, con quien intercambia unas palabras del todo triviales. Luego, apagan la luz y salen juntos de la caseta, se montan en un vehículo y abandonan la acogedora tranquilidad del camposanto para dirigirse, tal y como pudimos captar de su conversación, a algún bar a tomarse unas copas: ¡y todo sucede en un puñado de turnos y no tiene la maldita relevancia!

No sé si resultaré excesivamente insistente, o por decirlo de otro modo, pesado, pero creo que sería muy bonito que un día decidiese comprar una de las aventuras que por correo se distribuyen en creciente oferta, y encontrase unos PSIs si no parecidos, mejores, destinados por ahora únicamente a las grandes producciones del software. Si vuestra excusa era el poco dominio del PAW, ahora tenéis finalmente los medios. El buen gusto es vuestro. Ojalá os haya ayudado un poco a hacer esto realidad.

¡ESE GRAFICO ESTA VIVO!

Tan importante, quizá, como los PSIs, son esos bonitos dibujitos que aparecen en el mejor de los casos sobre lo que a nosotros nos interesa de verdad: la información del texto descriptivo y los mensajes con los que el ordenador se comunica con nosotros. Es patente el enorme revuelo que las

discusiones sobre este aspecto han levantado en las páginas de los mejores fanzines del mundo, ¿nos sirven, nos gustan, o son obsoletos?

No es tarea mía, por supuesto, arrojar alguna opinión (que las tengo) sobre este tema, sino el pensar en todas aquellas personas que consideran agradable para el jugador el tener en un recuadro más o menos vistoso las mejores representaciones gráficas alusivas a las grandes proezas que éste se ve obligado a hacer, y tratar de ayudarles en la medida de lo posible con nuevas y frescas ideas sobre el uso del editor gráfico que el PAW incorpora.

Así, podemos empezar considerando un concepto que se ve innegablemente ligado al aspecto gráfico: el movimiento. Aparentemente, es virtualmente imposible conseguir que nuestras creaciones artísticas se vean elevadas a otros estadios de belleza mediante el vehículo movimiento, como no sea metiendo las narices en el código máquina (o el BASIC, si somos lo suficientemente hábiles como para paliar la lentitud de éste) y luego incorporar este trabajo a la aventura via comandos EXTERN. En realidad, no es necesaria tamaña complicación porque, como todos vosotros sabéis, el PAW es tan maravillosamente versátil que siempre podemos manipularlo para que obedezca nuestros caprichos (¡sé de uno que lo usa como base de datos!)

El desarrollo del proceso que nos llevará a hacer realidad nuestros deseos es bastante más ingenioso que técnico, por así decirlo. Básicamente, todo se reduce a superponer una imágen sobre otra regulando constantemente el tiempo que cada secuencia va a estar en pantalla, y cuidando siempre, antes de dibujar el nuevo "fotograma", borrar el anterior con exquisita delicadeza para que "nadie note nada".

Esto de borrar la secuencia anterior se consigue no insertando espacios en blanco con el condacto PRINTAT y MESSAGE, como he visto hacer por ahí, porque el resultado es brusco, la zona a limpiar sufre un "blanqueo" descarado que el jugador aprecia con cierto deje de desagrado y el resultado es negativo. Es preferible "encriptar" la zona a re-dibujar como parte integrante del dibujo completo, e incluir, ya en la segunda secuencia, el grupo de órdenes que van a borrar esta zona, utilizando el caracter ASCII número 32 (el espacio) para las zonas con una estructura de pixels muy compleja y las opciones OVER e INVERSE para perfilar esas otras lineas y puntos que quedan siempre sobrantes.

Ofreceremos, por supuesto, un ejemplo completo de esta técnica, pero habrá de ser necesariamente en el próximo número porque en éste se nos acaba el espacio. Hasta entonces, os sugiero que practiquéis lo expuesto porque ningún aprendizaje es tan fructífero como el que os proporcionará la propia experiencia. ¡Hasta la vista!

Carlos Sisí

Comentario de juegos de Rol | **TÚcnicas de Programación** | Soluciones Serializadas

Programming with the Balrog

Aron Watson from New Zealand has a great tip for PAW users trying to get that little bit extra out of their adventure creator...

When creating a large adventure on the PAW, sooner or later you will run out of message space. That's after using 255 messages and 195 free system messages, giving you a total of 450 messages. But, when implementing half a dozen good characters, they could take up 255 messages rather fast, and that's excluding any other game messages.
The routine below allows you to use location text as messages.

Note: TEST should be a verb in the vocabulary.

```
/PRO 0

TEST _ COPYFF 38 10 ; Copies players loc. into flag 10
GOTO 1 ; Goto message
DESC ; Displays message

/PRO 1

_ _ AT 1
COPYFF 10 38 ; Copies flag 10 to players flag 38
DONE
```

Location 1 = "Change to your own message!"

So, if you had an adventure with 100 locations and used the spare locations as messages, this would give you a total of 602 messages!

Simon "Aragorn" Netherwood would love to know how Fergus McNeil and other adventure authors manage to get the Quill to accept four word sentences.

Simon also has some tips on character interaction:-

'With the Quill/PAW it is perfectly possible to have interactive, wandering non-player characters almost like a Level 9 game...

A character who moves away or does something when you are present. For example:
Starts
+----------+ Objects: 1. The Balrog Location 2
! Location 3 ! 2. Rod Lawton Location 1
! Office !
+-/-------\--+
/ \ Messages: 1. The Balrog goes NW.
+----/-------+ +-\----------+ 2. The Balrog goes SE.
! Location 1 !_! Location 2 ! 3. The Balrog goes east.
! Rod's Room ! ! Balg's Lair ! 4. The Balrog sees Rod,
+----------+ +----------+ "Mmmmm! Food!" shouts the cruel crusader before eating poor Rod.

Status Table

Entry 1 at 2 { Checks you're at location 2 }
present 1 { Checks Balrog is present }
message 1 { 'The Balrog goes NW.' }
place 1 3 { Puts Balrog in location 3 }

Entry 2 at 3
present 1
message 2
place 1 1

Entry 3 at 1
present 1
message 3
place 1 2

Entry 4 present 1 { Checks Balrog is present }
present 2 { Checks Rod is present }
message 4 { Print message }
destroy 2 { Removes Rod! (Hehehe!) }

For a long time the Balg has been praising the power of Gilsoft's PAW and the unlimited possibilities of the EXTERN command (this allows you to import your own machine code routines - for example, graphics or sound). Unfortunately the EXTERN command isn't easy to use - you need to know quite a lot about machine code to make the most of it. Aron Watson from New Zealand has been hacking away and brings to us the following routine. Now that Aron has shown how to use EXTERN, perhaps more budding machine coders could add to the PAW? Take it away Aron:-

This EXTERN routine is a simple, unambitious program but it should show other authors how they too can use the EXTERN command for their own PAW games. It allows you to choose the screen colours. You will need an assembler/monitor (eg Maxam II or Devpac). Enter Fig.1 into your game, for example, the start of your intro sequence. Then compile it and then create your finished game.

Fig. 1

```
/PRO 1
_ _ EXTERN 0
```

You will also have to find out where your game finishes in memory so you can set the ORG in Fig.2. Now, using the assembler, enter Fig.2. Assemble it, save it, then using your monitor, load your finished game at &100. Load your newly assembled code exactly after the game in memory. Then change the EXTERN vector at &106 from JP &000 to JP <address of new m/c routine>. Then save the whole thing out to disc. Now run your game and all should (hopefully!) work...

Fig. 2

```
ORG <address after game in memory>
LD DE,SETPAPER.
LD C,9
CALL &0005
LD DE,SETPEN.
LD C,9
CALL &0005
RET
SETPEN. DEFB &1B,&62,&?? ; Change ??'s to desired
SETPAPER. DEFB &1B,&63,&?? ; number (&00-&FF)
```

PROGRAMACION EN EL P.A.W. (I)

Quiero comenzar esta sección diciendo que está dedicada a vosotros, si teneis alguna duda de como realizar una rutina o del funcionamiento general del P.A.W. sólo teneis que escribir a la dirección del fanzine.

Para empezar, quisiera dar unas lineas generales de lo que es el P.A.W. y de cono realizar una buena aventura:

1- Primero teneis que tener una idea, imaginar de que va ha tratar vuestra aventura, -se supone que si estais delante del PAW eso ya lo tendrìais que tener pensado-, la realización de un mapa es fundamental antes de empezar a programar, en el debereis dar un número a cada localidad, os aconsejo que empeceis por la localidad 2 y dejeis la 0 para presentación y la 1 para uso auxiliar.

2- Bueno, ya tenemos nuestro mapa, ahora es el momento de cargar el PAW, también es hora de discurrir un poco e inventar las descripciones de las localidades. ¡Por favor!, ser algo originales, no os limiteis a: "estàs en tal sitio, puedes ver tal y tal", las descripciones tienen que ser ricas en contenido y con un doble significado, el cual el jugador deberá localizar si quiere avanzar en vuestra maravillosa aventura.

3- Se supone que habeis realizado unas de las mejores descripciones del mundo aventurero, así que vayamos a Conexiones e introdujamos las direcciones adecuadas (un fallo muy común en los escritores novatos es realizar las conexiones en un sentido pero en el otro no, es decir que si de la localidad 4 yendo al Oeste vamos a la 5 ¡Desde la localidad 5 al Este vamos a la 4!). También os

recomiendo introducir un pequeño laberinto pues es de agradecer por los mejores aventureros (y la perdición de los novatos).

4- Bien, hasta aquí todos los conversacionales son iguales, a partir de aquí la cosa empieza a ramificarse, puedes empezar a poner objetos, o a hacer gráficos, o hay un amigo mío que empieza a poner palabras y palabras en el vocabulario (lo cual no recomiendo).

Yo aconsejo que si vas ha meter gráficos en tu aventura vayas a Procesos, y en la Tabla 1 introduzcas la siguiente rutina (muy elemental, por cierto, perdonarme los grandes escritores y los que dominen el PAW, pues estoy dando sólo líneas generales).

```
I * *
      AT 0 ANYKEY GOTO 2 MODE 3 2 GRAPHICS 2
      LINE 16 DESC
```

Esta simple rutina es una rutina obligada en una aventura con gráficos. Pasamos a explicarla: AT 0(Si estamos en la localidad 0), ANYKEY(Espera a que se pulse una tecla), GOTO 2(Vete a la localidad 2), MODE 3 2(Gráfico siempre que se entre en la localidad y no hace scroll), GRAPHICS 2(Para que los gráficos aparezcan siempre que el jugador entre en una localidad), LINE 16(A partir de donde se escribirá el texto), DESC(Describe la localidad y termina).

En el próximo número seguiremos analizando rutinas (Más complicadas, lo prometo), y contestando las preguntas que me formuleis).

JON

PROGRAMACIÓN EN EL P.A.W.

Parece ser que todos los lectores tienen asumido eso de que "la mejor aventura es la más larga y con un número mayor de localidades", pues no.

La mejor aventura es aquella que tiene problemas inteligentes, donde pasan cosas en un determinado espacio de tiempo sin que tú influyas para nada.

No se debe comenzar un mapa de una aventura colocando las cosas al tuntún pues primero tenemos que ajustarlo al guión (veis por que decía en el número anterior que primero hay que hacer un guión) y según sea ese guión, ajustar el mapa. Pues si el juego va de una base futurista en forma de torre que llega hasta el cielo (lo apunto para una nueva aventura) no vas a hacer un mapa enorme que necesites cuatro hojas para dibujarlo, harás unas pocas localidades en el bajo, después un ascensor (o un teletransportador) y el primer piso, otro ascensor y el siguiente piso, y hasta podéis continuar la torre por los subsuelos. Pero la idea fundamental (la cual quiero que recordéis) es: No creáis que por

EL ARTE DEL MAPEADO

hacer un gran mapa tendréis un gran juego.

Muy bien, aún no hemos terminado con el mapa, así que no empecéis a programar aún. Hasta ahora sólo hemos hablado de la relación guión-mapa, pero todavía nos queda la relación problemas-mapa y tipo-mapa.

Problemas-mapa. Esta es la relación en la que caen gran cantidad de programadores con buenas intenciones pero sin las ideas claras. Tú puedes tener una gran cantidad de buenos problemas ideados, pero fallar en la colocación de los mismos, o sea, que no puedes hacer un mapa a la buena de Dios y luego decir: "Bueno, en esta esquina pongo el problema este taaaan interesante, luego en la otra esquina aquel que tan poco está mal...", pues no señores. Tienes que saber colocar tus problemas para que el jugador no se aburra o se sienta tan presionado que desista. "Zipi y Zape" es, a mi entender, el programa español que más cae en este error. Si algunos habéis tenido la desdicha de ver este juego y sobre todo el mapa, habréis comprobado que es exactamente lo anteriormente

dicho "un mapa casi rectangular con los problemas colocados ¡exactamente! en las esquinas del mismo, lo que te hace dar 40000 vueltas a todo el mapa sólo para conseguir un objeto".

Por lo tanto os recomiendo que pongáis los problemas en vuestro mapa de manera ordenada y equitativa, y que no haga que el jugador se aburra de dar vueltas.

Tipo-mapa. Hay varios tipos de mapas:

De caracter abierto. Donde el jugador tiene acceso a una amplia zona de juego con varios problemas iniciales en los que el orden de resolución de los mismos no afecta en absoluto para acabar la aventura, generálmente estas aventuras suelen dejar un amplio mapeado inicial el cual se va expandiendo (en pocas pantallas) a medida que vamos solucionando los problemas.

De caracter lineal. Este tipo de juegos tiene un mapa muy definido, por el que casi seguro que vas a tener que pasar por todas las localidades y que seguir los problemas a rajatabla, según el programador lo haya ideado.

Normálmente estos problemas suelen ir encadenados, y al solucionar uno, poco después te encontrarás con otro, si lo solucionas, podrás continuar, pero si no, te quedarás atascado hasta que "El viejo Archivero" se apiade de ti. Este tipo de aventuras no suelen quedar tan bien como los de caracter abierto, pues suele ser el tipo de juegos en el que más pronto te desanimas.

Mezcla: Son mis preferidos, pues suelen ser en los que mayor entretenimiento hay y mejores problemas tiene (salvo excepciones). Suelen ser

los juegos que tienen un gran mapeado y que tienen (como agudamente habrás intuido) partes de abierto y partes de lineal, lo cual le hace el tipo de programa con más diversión, más duración, más variación (lo cual es fundamental, pues si no, se haría aburrido) y donde es más facil colocar buenos personajes y problemas. O sea, lo más parecido a la realidad que existe. Yo siempre pongo como ejemplo el (para mi) mejor programa español de este estilo: El Ojo del Dragón, pero claro, un programa de esta índole suele nesitar mucha memoria.

Me despido por este mes. Sólo avisaros que en el próximo número empezaremos a ver tipos de problemas y como hacerlos. Abur.

J O N

COLABORACIONES

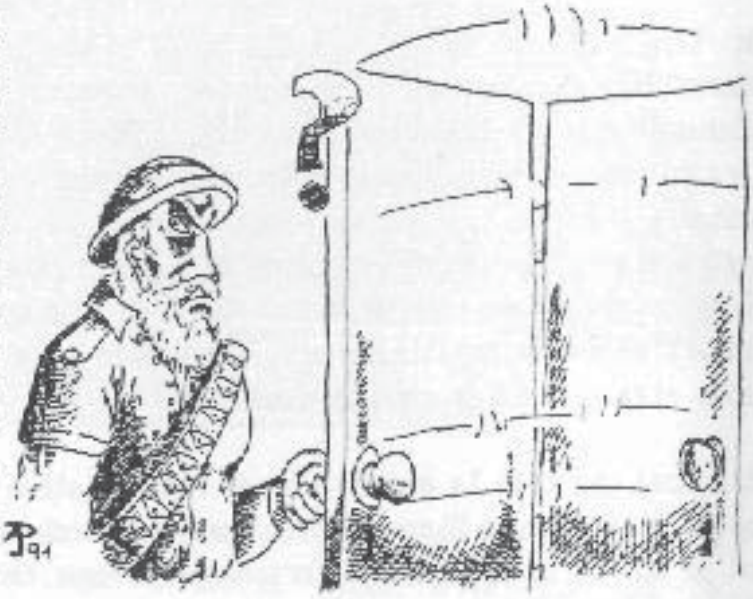
¿Quieres expresar tus ideas en este, tu fanzine? ¿Eres un maestro en temas relacionados con la aventura? ¿Te gustaría enseñarnos? Has jugado a algún programa del que te gustaría hacer una critica?

Envianos todo lo que quieras. Si tiene un mínimo de interés, a todo lo referente a estos temas, lo publicaremos con mucho gusto. También aceptamos dibujos, ilustraciones, etc...

Si te gustaría crear una sección, esta es tu oportunidad. "A Través del Espejo" no es nada sin vosotros.

PROGRAMACIÓN EN EL P.A.W.

Bueno, ya estamos en como realizar los problemas, y si nos paramos a pensar, ¿cual es el más típico problema en un conversacional?. Pues claro, una puerta que nos impide el paso. Si sois lectores de Microhobby os habréis dado cuenta que la única rutina que no era para el PAW en la sección "El rincón del Parsero" era precisamente la puerta (que salió para el G.A.C.), pero como no me pareció muy bien, aquí tenéis una manera de realizar una puerta en el PAW.



En primer lugar deberéis tener libre la bandera 210 (por ejemplo), y la puerta nos permitirá el acceso desde la localidad 10 a la 11 por el Norte. Entonces habrá que poner estos mensajes:

1- Abres la puerta.
2- Cierras la puerta.
3- La puerta ya estaba abierta.
4- La puerta ya estaba cerrada.
5- ¡Thump! te golpéas contra la puerta cerrada!

A continuación haremos la siguiente relación:

Bandera 210 = 0 : Puerta cerrada.
Bandera 210 = 10 : Puerta abierta.

Y ya tenemos todo dicho, ahora la rutina:

- TABLA DE RESPUESTAS -

```
I NORTE _  AT 10 EQ 210 0 MESSAGE 5 DONE
I NORTE _  AT 10 EQ 210 10 GOTO 11 DESC
I SUR _    AT 11 EQ 210 0 MESSAGE 5 DONE
I SUR _    AT 11 EQ 210 10 GOTO 10 DESC
```

```
I ABRIR PUERTA ATLT 12 ATGT 9 EQ 210 0 MESSAGE 1 LET 210 10 DONE
I ABRIR PUERTA ATLT 12 ATGT 9  EQ 210 10 MESSAGE 3 DONE
I CERRAR PUERTA ATLT 12 ATGT 9 EQ 210 0 MESSAGE 4 DONE
I CERRAR PUERTA ATLT 12 ATGT 9 EQ 210 10 MESSAGE 2 LET 210 0 DONE
```

Y ahí tenéis la rutina. Esta rutina, simplemente hace las funciones de abrir y cerrar la puerta, te avisa si la intentas abrir cuando ya lo está y hace lo mismo si esta cerrada e intentas cerrarla. También te castiga con un golpetazo si intentas atravesarla estando cerrada. Pero la rutina se puede arreglar (o mejor dicho: ampliar), y así puedo aprovechar para pediros vuestra colaboración, mandando las contestaciones de las preguntas que yo os formularé sobre una de las rutinas que hayamos hecho, pero por esto no vamos a regalar nada (que más quisiera yo que andar repartiendo cosas entre los lectores) pero vuestro nombre saldrá publicado en esta sección como los que habéis contestado bien, y analizaremos vuestras respuestas. Espero que secundéis esta idea, pues puede ser muy beneficiosa para todos, pero si no respondéis se mete en el baul de los recuerdos y se olvida. De todas maneras la decisión está en vuestras manos.

La pregunta de este mes es:
¿Cómo haríais en la rutina de la puerta que en la descripción de la localidad apareciera el mensaje: "La puerta está abierta/cerrada" según corresponda? Bueno, no os quejéis porque esta vez es muy fácil, pero la dificultad aumentará según el tipo de rutina.

Ahora la respuesta a un amigo mío:
Marcelino Villarino Aguilar me ha preguntado como hacer que en una localidad, si insultas a un personaje este se vaya enfadando poco a poco hasta llegar a un grado que te mate, pudiendo apaciguarlo pidiéndole perdón. Bueno "Marce" (de FOZ, Lugo), creo que la pregunta de poco le servirá a otros lectores, pero ahí va:

El personaje está en la localidad 10:
Habrá que poner los siguientes mensajes:

5- El tío parece ofendidio en extremo.
6- El tío, ya a punto de estallar, te pega un mamporro que te manda a Jupiter.
7- Parece que el tío se ha calmado.

- TABLA DE RESPUESTAS -

```
I DECIR FULANO AT 10 PROCESS 3
```

- PROCESOS 3 -

```
I * * PARSE
I INSULTO (CUALQUIERA) _ EQ 5 0 MESSAGE 5 LET 5 5
I PERDON NOTEQ 5 0 LET  5 0 MESSAGE 7
```

- PROCESOS 2 -

```
I TIO CABREADO AT 10 EQ 5 1 MESSAGE 6 TURNS END
```

Bueno, por este mes se ha acabado, en el próximo número más.

J O N

COLABORACIONES

¿Quieres expresar tus ideas en este, tu fanzine? ¿Eres un maestro en temas relacionados con la aventura? ¿Te gustaría enseñarnos? Has jugado a algún programa del que te gustaría hacer una crítica?

Envianos todo lo que quieras. Si tiene un mínimo de interés, a todo lo referente a estos temas, lo publicaremos con mucho gusto. También aceptamos dibujos, ilustraciones, etc...

Si te gustaría crear una sección, esta es tu oportunidad. "A Través del Espejo" no es nada sin vosotros.

PROGRAMACIÓN EN EL P.A.W.

Para este mes se me ha ocurrido daros la rutina de un alegre fantasmón que se pasea como Perico por su casa y nos incordia lo más que puede (debido a su condición de fantasma, pues en realidad es de lo más majete), ¿y por qué una rutina de este estilo?, pues por que creo que es interesante y a su vez se presta a muchas modificaciones para que vosotros la uséis como mejor os convenga.

Este fantasmón no es de esos PSIs que dicen: "Manolaco se va al Norte" o:"Manolaco aparece", pues entonces ya sabemos por donde anda, nuestro fantasma (a quién llamaremos Don Fantasma) se materializará y desmaterializará por ahí a su gusto, sin que nosotros sepamos donde aparecerá (¿o es que normalmente véis a un fantasma por ahí de paseo?).

Pero primero habrá que asignarle una bandera a Don Fantasma, y como soy algo morboso le daremos la 13.

Para que nuestro ensabanado amigo se mueva tendremos que hacer una rutina como esta:

Mensaje 10: "Don Fantasmón se materializa"
Mensaje 11: "Don Fantasmón desaparece entre humo rosa"
Mensaje 12: "Aquí está Don fantasmón"

```
Procesos 1:
1 _ FANTASMON SAME 38 13 LT 5 3 GT 5 0 MESSAGE 12

Procesos 2:
1 _ FANTASMON EQ 5 0 RANDOM 34 PROCESS X (Diferente de 1 o 2)

Procesos X:
1 _ _ LT 34 25 SET 15
1 _ _ LT 34 50 GT 34 24 SET 16
1 _ _ LT 34 75 GT 34 49 SET 17
1 _ _ LT 34 101 GT 34 74 SET 18
1 _ _ SAME 38 13 MESSAGE 11
1 _ _ NOTZERO 15 LET 34 13 MOVE 13 SAME 38 13 MESSAGE 10 LET 5 3
1 _ _ (Igual excepto: NOTZERO 16 LET 34 2)
1 _ _ (Igual excepto: NOTZERO 17 LET 34 3)
1 _ _ (Igual excepto: NOTZERO 18 LET 34 4)
```

Ahora deberéis ir a Procesos 1 y en la rutina que di en el número 1, poner antes del DESC; un LET 5 3 y un LET 13 (locno) según donde queréis que Don Fantasma empiece a jugar.

Don Fantasma ya se mueve por nuestro juego, pero por ahora no hace nada, así que vamos a hacer que necesite de una gema para sobrevivir, y que la busca por todo el juego, por supuesto, esa gema es necesaria para nosotros y que con fantasma la coja no nos conviene en absoluto.

Mensaje 13: "Don Fantasma coge la gema y contento te dice: "Ahora ya me he ganado la vida eterna" y desaparece para siempre de este mundo".

Procesos 2:

```
I _ GEMA COPYOF 4 100 (SUPONIENDO QUE ESTE SEA EL NUMERO DE LA GEMA) SAME 100 13
    DESTROY 4 SAME 38 13 MESSAGE 13
I _ GEMA ISAT 4 252 SET 13
```

Con esto, el fantasmón ya nos habrá hecho bastante la puñeta. Ahora vamos a hacer una entrada para examinar al fantasma: Mensaje 14: "Es una pobre alma en pena que vaga por este mundo en busca de la gema dorada".

Respuestas:

```
I EXAMINAR FANTASMON SAME 38 13 MESSAGE 14 DONE
```

También podemos hacer que el fantasmón nos de la lata cada vez que nos vea:

Mensaje 15: "Don Fantasma dice: "UUUUUuuuuu...."
Mensaje 16: "Buuusco la gema dorada, ¿podrIIIas ayudarme?"

Procesos 2:

```
I _ _ SAME 38 13 CHANCE 25 MESSAGE 15
I _ _ SAME 38 13 CHANCE 45 MESSAGE 16
```

Y así Don Fantasma nos dará la paliza cada vez que nos vea. Ahora vamos a conversar con Don Fantasma:

Mensaje 17: "Don Fantasma dice:"¿Eh?"
Mensaje 18: "Don Fantasma dice:"Yo sí que necesito ayuda"
Mensaje 19: "Don Fantasma dice:"Es una gema meravillosa, y ella me dará la vida eterna".

Respuestas:
1 DECIR FANTASMA SAME 38 13 PROCESS 6 (Por ejemplo)

Procesos 6:
1 * * PARSE MESSAGE 17 DONE
1 AYUDA _ MESSAGE 18 DONE
1 - GEMA MESSAGE 19 DONE

Así Don Fantasma parecerá algo real (un fantasma no lo es) y si le decimos algo de la gema, este nos obsequiará con una descripción de lo que poseerla le otorgará.

Bueno, este es un PSI bastante creible pero también es interesante que Don Fantasma padezca cleptomanía.

Mensaje 19: "Don Fantasma te coge el _"
Mensaje 20: "Don Fantasma te lo devuelve"

Procesos 2:
1 _ LADRON SAME 38 13 CHANCE25 WHATO LT 51 255 EQ 54 254 MESSAGE 19 PUTO 1

Procesos 6:
1 DAME _ WHATO EQ 54 1 PUTO 254 MESSAGE 20

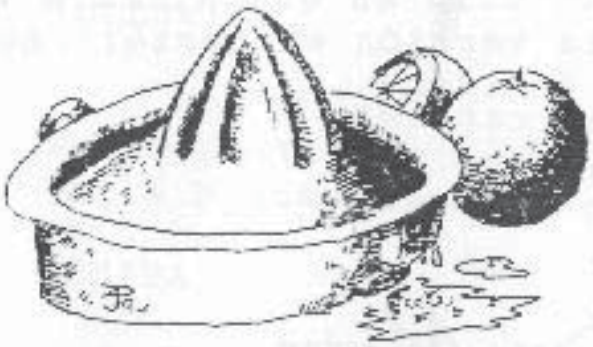
Bueno, yo creo que nuestro fantasma es bastante creible, por lo que lo dejamos aquí.

Respecto a la pregunta del mes pasado, resulta que cuando la leáis en el número anterior, este número ya estará en avanzado proceso de maquetación, por lo que hasta el mes que viene no sabremos los acertantes, por de pronto, la pregunta de este mes es: En la rutina de movimiento de Don Fantasma como haríais para que os dijera: Don Fantasma va al Norte, Don Fantasma va al Oeste...

Por este mes nada más. Chao.

Jon.

COMO EXPRIMIR EL COMANDO EXTERN.



En el número cinco ya se hablaba de la combinación de Código Máquina y Paws, mediante el comando Extern, pero no se daba ningún consejo sobre como aprovecharlo. Primero tenemos que saber que queremos hacer, y más concretamente como y en que aspecto del juego.

- Borrados de pantalla: Hay muchos y variados, aunque sólo son efectivos los hechos en C/M. Se puede escoger uno que borre la pantalla en espiral, gradualmente, como una persiana de un lado a otro etc...

- Presentación y desaparición de textos: Aquí se presentan las mismas opciones que en el apartado anterior, pero con algunos añadidos: Scroll en cualquier dirección, aparición gradual por modificación de tintas (de 0 a 7 si el papel es negro, de 7 a 0 si el papel es blanco).

- Sonido: Bien sea un FX (sonido de trueno, una bomba, una ametralladora...) o una melodia, en el menú o al final del juego, claro que una buena melodia ocupa más memoria que un FX.

- Subjuegos: Famosos en juegos como "Worm in the Paradise" o "The Micro Mutant". Son juegos sencillos, como un puzle o un comecocos, que suelen estar de adorno aunque a veces son indispensables para acabar el juego.

- Bloques gráficos: [[Ahaha!!. He aquí el más profundo e interesante de los usos a los que se puede someter el comando extern. Bien puedes meter bloques grandes para la pantalla inicial, que bloques de 1K para una pantalla normal del juego. Ayudado por una sencilla rutina, también se puede

del juego. Ayudado por una sencilla rutina, también se puede dejar boquiabierto al jugador con un sprite o gràfico mòvil, ensamblado para imprimir sobre fondo de un color, esto es, sin necesidad de màscara.

- Para que os hagàis una idea, un gràfico de 6 x 4 con cinco fases de animación ocupa menos de 1K de memoria, pero el efecto del movimiento queda màs bonito que un gràfico que ocupe lo mismo.

En fin, como habèis visto, se puede hacer de todo con el comando EXTERN.

Si se os acaban los 2K que deja libres el PAW, habrèis de buscar memoria libre en otras zonas de la memoria... pero de eso ya hablarè en otro momento. Deseandoos una aventura "pui potita", me auto-externeo.

Carlos Garcia de Paredes

ANUNCIOS. TARIFAS

Las tarifas para un anuncio, durante un fanzine, son las siguientes:

1/4 de pàgina.....500 Ptas.
1/2 de pàgina.....1000 Ptas.
1 pàgina.....2000 Ptas.
Contraportada.....3000 Ptas.

Para màs de un anuncio o màs de una pàgina, tenemos tarifas especiales, llàmanos, te atenderemos con mucho gusto.

NUEVAS VERSIONES DE PAW

Quién no se a preguntado alguna vez que para que servía la opción "Z" de PAW (Overlays del usuario) si no salía al mercado ningún overlay y en el manual no se daba la más mínima información sobre como crear estos overlays. Pues ahora tengo por fin en mis manos algunos de esos overlays que sí circulan por las redes de piratillas.

Los overlays que henos podido ver son los siguientes:

H- Hunk
M- Mega A01.1
P- Paw Phosis A01
T- Paw Tel A01.1
Y- Bytes Localización V,2.0

El primero de ellos (H) ya venía con la versión en castellano que distribuye aventuras AD, el M, P y T, llevan copyright de "1988 G T Kellet/Kelsoft" y no sabemos a ciencia cierta de donde han podido salir. El último (Y) ha sido escrito por el amigo Daniel Querol, nuestro conocido solucionador.

Vamos a tratar de explicar que es lo que hace cada uno de estos overlays:

H- El Hunk Management A01 es un extraño overlay sobre el que se cierne una tupida capa de misterio, ya que el manual no dice nada, en AD no saben nada, Tim Gilberts no sabe explicarlo en castellano, en fin, a mi entender, reserva en la memoria un espacio y te permite cargar algo ahí, pero realmente tiene escasa utilidad, ya que ni siquiera te dice en que dirección está ese

espacio.

M- Mega A01.1 es un overlay que permite tener simultàneamente todas las opciones en memoria, es decir, que no hay que acceder al disco cada vez que quieres usar una de ellas. Esto està muy bien, pero como es lògico, en cuanto empieza a quedar poca memoria su uso ya no es efectivo, y aunque estè toda la memoria libre no mete las opciones de Gràficos, Compresiòn, Selecciòn de periférico, etc...

P- El Paw Phosis A01 es un overlay de complicado manejo pero de elevadas prestaciones, ya que nos permite: copiar una tabla de procesos, transferir entradas, imprimir una sola entrada (impresora), grabar una tabla, cargar una tabla, verificarla, nos da informaciòn sobre Localidades, Mensajes, etc... Ademàs de permitirnos acceder directamente (sin tener que volver al menù principal) a processos, respuestas, objetos, mensajes, conexiones, etc.

Esto nos permite, entre otras muchas cosas, tener nuestra propia biblioteca de rutinas y tablas de procesos, para ir ponièndolas en nuestras aventuras. Tambièn nos permite algo menos licito pero màs divertido, como es robar esa rutina de salidas de aquella aventura o copiar el movimiento de un personaje de tal otra, etc.

T- Paw Tel A01.1: Extraño, muy extraño, le verdad es que no hemos conseguido manejarlo en su totalidad, pero parece ser que sirve para hacer el mapeado màs còmodamente. Permite buscar conexiones no sòlo en su tabla, si no las ocultas en las tablas de procesos o respuestas, etc. Tambièn da acceso directo a algunas de las opciones del menù

principal.

Y- Bytes Localización V,2.0 es un overlay muy útil creado por Daniel Querol. Lo que hace es muy simple, lo seleccionamos y nos da una lista que comprende: Número de localidad, Bytes que ocupa el gráfico y Bytes que ocupa el texto de dicha localidad. Cabe sañalar que no es del todo perfecto, ya que se equivoca en la última localidad y nos da una cantidad de texto desmesurada. A ver si alguien toma ejemplo y programa más overlays.

Sólo resta decir que estos overlays solamente funcionan en una determinada versión inglesa de PAW que es exactamente la A17 +3, cuyo EXTVEC es:34768 y el PRTADD es:29585. Poca utilidad pues para los epañoles que, como no, vamos a hacer la pregunta de rigor, ¿A que espera AD (agente en España) para traducir y adaptar estos y otros overlays a nuestro Paw? ¿Acaso se han desentendido de nosotros?. Por que no es eso lo que se dice en el manual. Esperamos ansiosamente, AD.

The Dark Master.

Programación en el P.A.W.

MEJORAS EN LA RUTINA DE MOVIMIENTO DE PSIs.

Seguimos con la rutina de movimiento (la 3ª ya ¿no?) con unas interesantes mejoras con respecto a las anteriores.

1. Esta rutina no es específica para un personaje como fue la anterior, sino que podréis adaptarla a vuestro gusto.

2. Pero lo más importante y revolucionario, y por lo que me siento tamañamente orgulloso, es que esta rutina tiene la capacidad de servir ella sola para todos los personajes que queráis, sólo teniendo que hacer por cada personaje unas llamadas en Procesos 2 y poco más, el increible ahorro de memoria es descomunal.

La rutina principal, y que se ubica en las Tablas de Procesos 3 y

4 es esta:

Procesos 3

```
*  *  GT   158  0
      LT   158  25
      SET  152

*  *  GT   158  24
      LT   158  50
      SET  153

*  *  GT   158  49
```

```
      LT        158  75
      SET       154

*  *  GT        158  74
      LT        158  101
      SET       157

*  *  NOTZERO   152
      COPYFF    250  100
      LET        33    2
      MOVE      100
      ADD       100  101
      NOTZERO   100
      SAME      250   38
      MES         0
      MESSAGE     4

*  *  NOTZERO   153
      COPYFF    250  100
      LET        33    3
      MOVE      100
      ADD       100  101
      NOTZERO   100
      SAME      250   38
      MES         0
      MESSAGE     5

*  *  NOTZERO   154
      COPYFF    250  100
      LET        33    4
      MOVE      100
      ADD       100  101
      NOTZERO   100
      SAME      250   38
      MES         0
      MESSAGE     6

*  *  NOTZERO   157
      COPYFF    250  100
      LET        33   13
      MOVE      100
```

```
      ADD       100  101
      NOTZERO   100
      SAME      250   38
      MES         0
      MESSAGE     3

_  _  PROCESS     4
_  _  DONE
```

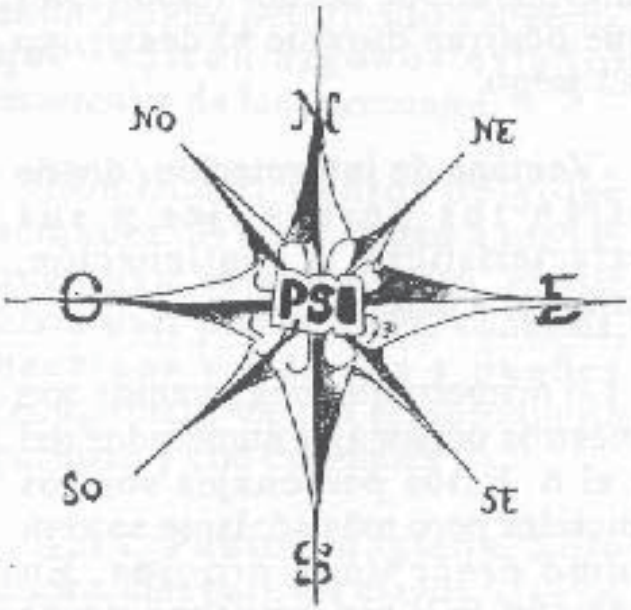
Procesos 4

```
S  _  NOTZERO   152
      LET        34    2
      MOVE      250
      CLEAR     152
      SAME      250   38
      MESSAGE     1

E  _  NOTZERO   153
      LET        34    3
      MOVE      250
      CLEAR     153
      SAME      250   38
      MESSAGE     1

O  _  NOTZERO   154
      LET        34    4
      MOVE      250
```

```
      CLEAR      154
      SAME       250  38
      MESSAGE      1

N  _  NOTZERO    157
      LET         34  13
      MOVE       250
      CLEAR      157
      SAME       250  38
      MESSAGE      1

_  _  CLEAR      152
      CLEAR      153
      CLEAR      154
      CLEAR      157
```

Esta es la rutina principal, pero ahora os voy a explicar el proceso para crear un personaje con esta rutina.

1. Ve a objetos y (si quieres que tu aventura sea realmente profesional) crea dos objetos de la siguiente manera:



-Primer objeto: "a (nombre del personaje)"

-Segundo objeto: "Nombre del presonaje"

Ahora deberás guardar dos banderas, una para la locno del PSI, y otra especial.

Llamaremos X1 al primer objeto, X2 al segundo, PSI1 a la primera bandera y PSI2 a la otra.

Ahora en Procesos 2 hacer lo siguiente:

Procesos 2

```
* (nom. PSI) ZERO      PSI2
             RANDOM    158
             LET        51  X2
             COPYFF    PSI1  250
             PROCESS     3

* (nom. PSI) COPYFF    250 PSI1

_  _        COPYFO    PSI1  X1
```

¡Ah!. Acordaros también de poner al principio del programa: LET PSI1 (Locno de comienzo).

Bueno, espero que esta rutina os sirva de algo. Abur.

JON.

Como mapear una aventura y no perderse en el intento (I).

En esta serie explicaremos como mapear una aventura y que vuestro mapa no sea un caos total.

Empezemos dando unas líneas generales de como debéis organizar el mapa según el tipo de aventura a la que os enfrentáis y de como debéis marcar en él los objetos y los problemas. En números posteriores daremos trucos para mapear laberintos de todo tipo (ilógicos, que desaparecen objetos...).

Sin más preámbulos comenzemos:

1. La primera partida.

Cuando te enfrentes por primera vez a una aventura deberás estar acompañado de lápiz y papel y bastante tiempo.

Empieza por pasearte por el juego mapeando todo y apuntando los objetos que encuentres. El mejor mapa es el de los cuadrados unidos por líneas, en cada cuadrado escribe el objeto/s que hay y todo lo que veas de interés. Una de las cosas más importantes es la brújula. Dibujala en una esquina de la hoja, y así no tendrás confusiones de este y oeste.

Ahora habrá que examinar el juego y ver si tiene gran cantidad de subidas y bajadas, por que si es así deberemos variar la brújula de esta forma:

Pero si en cada altura hay gran cantidad de localidades que las superiores te impiden ver las inferiores deberás recurrir al mapa por alturas, osea, hacer un mapa de cada altura por separado.

Los personajes, si se mueven, deberemos indicar que existen y poner en una hoja su inventario inicial, pues no podremos poner en el mapa donde están pues siempre varían de localidad (a no ser que siempre los encuentres en el mismo sitio al comenzar una partida). Ejemplo:

Hazard en Johnnie Verso.
Optimus en El Ojo del Dragón.

Si encontramos un problema deberemos apuntarlo también en el mapa (si el problema es estático, porque sino no deberemos apuntarlo, pues varía de localidad).

Ejemplo de problema no estático:

Un personaje que nos requiere un objeto para ayudarnos, ese personaje se moverá y el programa que lleva consigo también.

Gregor en Johnnie Verso.
Joe en Johnnie Verso.

2. Diferencias en las conexiones.

En una aventura no sólo existen las direcciones del compás, sino que tenemos también; subir/bajar, entrar/salir... También hay puertas que nos impiden acceder a diferentes localidades, por lo tanto deberemos diferenciar todo esto en nuestro mapa.

Para entrar/salir yo uso: <===>

Para entrar/salir con una puerta que nos bloquea el paso: <=|=>

Para subir/bajar: <===>

Para subir/bajar con una puerta que nos bloquea el paso: <=|=>

Así mataréis dos pajaros de un tiro, pues además de marcar las conexiones apuntaréis un problema (que es esa puerta bloqueándonos el paso).

Bueno, por este mes ya es bastante, en el próximo empezaremos en el proceloso y difícil campo de mapear un laberinto. Abur.

JON.

Como mapear una aventura y no perderse en el intento.(y II)

Una de la cosa que más preocupa a los jugadores nóbeles es cuando en sus andanzas por la aventura descubren una serie de localidades con la misma descripción y un montón de conexiones que parecen estar algo liadas: Los Laberintos.

Por JON.

Los laberintos han existido en los conversacionales desde siempre. En la biblia del aventurero (lease "The Hobbit" ya había uno de los más complicados laberintos que existen, las famosas: "Goblin's Dungeons". En los juegos más recientes, "Johnnie Verso", "El Ojo del Dragón", todos tienen su (o sus) laberintos.

Pero los laberintos no han sido iguales siempre, se puede decir que han evolucionado, (dejando aparte a Bilbo Bolson cerrado, pues él y su aventura se adelantaron al tiempo) y los primeros laberintos eran simplemente una serie de localidades iguales con direcciones liadas, eso quiere decir que por ejemplo, si en un laberinto vas al Este y después al Oeste no tienes porque estar en la misma localidad, o que si vas al Norte en una localidad puedes no haberte movido... y gran cantidad de "salvajadas" para que el aventurero "disfrute" de la aventura.

Ahora los laberintos se han modernizado, y nos los han complicado más, pues ahora nos hacen las "cerdadas" de que si dejas algún objeto en el laberinto, este desaparece, o se cambia de localidad para si nos movemos y nos lo volvemos a encontrar, creer que no nos habemos movido.

Si queréis saber como se mapea un laberinto seguir leyendo...

Un laberinto sin desaparición de objetos se mapea de la siguiente manera:

1- Dejamos un objeto en la localidad de comienzo del laberinto.
2- Nos movemos en una dirección.
3- Si el objeto vuelve a estar aquí es porque no nos hemos movido. Entonces podremos dibujar en el recuadro de la localidad, una flecha que vuelva a la misma localidad por esa dirección.
4- Nos volvemos a mover en otra dirección, si no nos encontramos el objeto, es porque estamos en otra localidad, la apuntamos, dejamos otro objeto y repetimos el proceso, así hasta que encontremos algo distinto o salgamos del laberinto.

Con los laberintos de desaparición de objetos es bastante más difícil, pues hay que recurrir a la lógica.

Si hay un personaje cerca podemos usarlo como referencia para ir mapeando. (Diciéndole que vaya a una dirección, y si vuelve a desaparecer es porque no se ha movido) ¿No?.

Si no hay ningún personaje cerca, la cosa se complica, así que deberemos pensar: "Si voy al Este entro en el laberinto, así que voy a ir al Este y después al Oeste". Si resulta que vuelves al punto de partida. ¡Eureka!, ya tenemos una parte del laberinto mapeado, ahora volvemos al Este.

Nos movemos en otra dirección, (por ejemplo: Norte) si ahora hacemos: Sur y Oeste y volvemos a estar en la localidad inicial hay probabilidades de que al Norte de esa localidad haya otra. Ahora hacer: "Este, Norte y Oeste" si no estamos en el principio... ya tenemos dos localidades del laberinto.

Y así tenéis que ir haciendo con todas, pues, si sabéis como volver al punto de partida, podréis volver y saber si os habéis movido, ¿comprendéis?, os diré que con este sistema mapeé el laberinto del pantano en El Ojo del Dragón (que se las trae el dichoso laberinto).

Para los laberintos donde los objetos se cambian de localidad recurrir a la anterior explicación. ¡Y no os dejéis ningún objeto!.

Espero que estas explicaciones os hayan servido, y que no cojáis manía a los laberintos, pues mapearlos son de lo más divertido. Claro, si no hay personajes que te arreen leches o que te capturen y te lleven a una celda, que si no...

Programación en el P.A.W.

INVENTARIO DE LOS PSIs.

Por JON.

Otra cosa muy importante en un PSI y que da mayor profesionalidad a vuestras aventuras es el que los PSIs lleven su propio inventario, de como hacer eso vamos a hablar ahora.

Como ya dijimos en anteriores ocasiones para cada PSI completo, (esto requiere que se mueva independientemente, que podamos hablar con él, que nos pueda seguir y que tenga su propio inventario) hay que guardar lo siguiente:

1- Una bandera auxiliar para comprobar si nos está siguiendo y así anular la rutina de movimiento.
2- Una bandera que lleva la localidad actual del PSI.
3- Un objeto que sea del tipo: a nombre de. Ejemplo "a Jones"
4- Otro objeto del tipo: nombre: a nombre. Ejemplo: "Jones"
5- Una tabla de procesos para el diálogo con él.

Y además, si queremos que ese personaje tenga su propio inventario tendremos que darle una localidad que esté aislada de las del juego.

Muy bien, ahora que ya hemos definido todo lo relacionado con el PSI os mostraré la rutina que realiza el inventario y el acto de DAR/RECIBIR objetos del PSI:

Locno= Localidad del inventario del PSI.
Message 1: "Jones acepta _."
Message 2: "Jones está llevando "
Message 3: "Jones te da _."

Tabla de Respuestas:

I DAR _ Noun2 (nombre personaje) Same (bandera locno PSI) 38 Whato Lt 51 255 eq 54 254 Puto (locno inven PSI) Message 1 Done

Procesos X (La tabla que hayas asignado para diálogo con el PSI)
I DAME _ Same (bandera locno PSI) 38 Whato Lt 51 255 Eq 54 (locno inven PSI) Message 3 Puto 254 Done
I I _ Mes 2 Listat (locno inven PSI) Done

O más profesional todavía:

Message 4: "No puedes darle eso, pues no lo tienes."
Message 5: "Eso no lo tiene."

Tabla de respuestas:

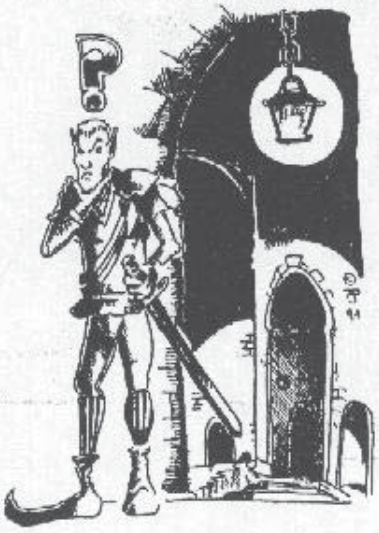
I DAR _ Noun2 (Nombre PSI) Same (bandera locno PSI) Whato LT 51 255 Noteq 54 254 Message 4 Done

Procesos X:

I DAME _ Whato Lt 51 255 Noteq 54 (Locno inven PSI) Message 5 Done

La rutina funciona de la siguiente manera:

En la rutina donde le das un objeto al PSI, primero comprueba el "A QUIEN" se lo vas a dar con un simple "NOUN2: NOMBRE PSI" para evitar confusiones si hubiera más de un PSI en la localidad. Después comprueba si ese personaje está aquí con un "SAME (flagno locno PSI) 38", después pone en la bandera 54 el nombre de la SL actual con "WHATO", comprueba si es un objeto valido con "LT 51 255" y comprueba si está siendo llevado con un "EQ 54 254", después si todo es favorable lo pone en la locno del inventario del PSI con "PUTO" (locno inven PSI), se lo dice al jugador ("MESSAGE 1") y termina "DONE".



En la rutina que pide un objeto al PSI, todo es similar, exceptuando que la comprobación de si el objeto está siendo llevado, se hace a la locno del PSI: "EQ 54 (locno PSI)" y pone el objeto en la locno 254 (cogido) con "PUTO 254", por todo lo demás es igual.

Para pedirle el inventario al PSI, primero ponemos el mansaje: "Jones está llevando " y después listamos los objetos de la localidad del PSI con un simple "LISTAT (locno inven PSI)".

Las otras dos series de condactos, son iguales a las anteriores sólo que en vez de comprobar si los objetos están siendo llevados por el personaje o por el jugador para cambiarlos de lugar, lo que hace es totalmente lo contrario, osea: "NOTEQ 54 254" y "NOTEQ 54 (locno inven PSI)".

¿Ok?, espero que esto os haya servido de algo, ya sabéis, si tenéis dudas solamente escribir al fanzine. Chao.

Programación en el P.A.W.

Por JON.

Lo siento, de verdad que lo siento, en el nº 8 salió una rutina que seguro que más de uno se sintió atraido por ella, y cuando se dispuso a teclearla; ¿Qué pasó?, ¡Que faltaban los mensajes! Lo siento, entono el "Mea Culpa" y ahí van los mensajes:

Mensaje 1: "_ aparece."
Mensaje 0: "_ se va al"
Mensaje 3: " norte."
Mensaje 4: " sur."
Mensaje 5: " este."
Mensaje 6: " oeste."

Me ha llegado una carta de un amigo nuestro (Juan Antonio Paz Salgado) que nos envía una rutina mejor para la respuesta a Marcelino aparecida en esta sección anteriormente, aquella donde nos pedían una rutina para un personaje que se cabreaba cuando lo insultabas, llegando a matarte; hasta ahora me parece muy bien, pero lo que ya no me parece tan bien es que diga que la que yo dí estaba mal. Amigo Juan Antonio, tu y yo enfocamos la pregunta de distinto modo, que me digas que es más acertada la tuya, vale, pero la nuestra funciona correctamente. ¿Vale?, de todas maneras gracias, y a ver si la gente se anima a escribir a la sección.

Aquí está la rutina:

Banderas:
100: Localidad tipo.
5: Cabreo del tipo (cabreo superior a umbral=muerte)

Mensajes:
a- El tipo parece ofendido.
b- El tipo, ya a punto de estallar, te pega un manporro y te destroza la cabeza.
c- Parece que eso lo calma un poco.

Tabla de respuestas:
| Decir Fulano SAME 38 100 PROCESS 3

Process 3:
| ** PARSEDONE
| Insulto _ PLUS5 4 MESSAGE b
| Perdon _ MINUS 5 4 MESSAGE c

Process 2:
| Tio Jodido SAME38 100 GT 5 umbral(valor) MESSAGE c TURNS END

Nos dice Juan Antonio:"Con este sistema el tio se va enfadando poco a poco a medida que lo insultas y también se va calmando con el tiempo (es decir, no te mata en cuanto te ve). De todas formas es una rutina excesivamente sencilla, se podrían incluir otros parámetros como la supceptibilidad del PSI (usando el umbral), el tiempo del olvido de una ofenda, etc."

Muy bien, después de esta extensa aclaración seguimos con lo nuestro. Algunosamigos que han probado la aventura de pronta aparición:Freeze Aliens, me han preguntado la manera de realizar lo siguiente: ¿Cómo haces para que cuando uno de los personajes se encuentre con el Amien, su cadaver aparezca en la localidad donde lo mataron?. Como veis, la pregunta es interesante, y aquí va la respuesta:

Mensaje 0: "Escuchas a (personaje) gritar, después un ruido, entonces, nada..."
Mensaje 1: "El Alien se abalanza sobre (personaje) y lo despedaza."
Bandera Alien: 254
Bandera personaje: 200
Objeto "cadaver personaje": 3

Process 2:
| ** COPYFO 230 3
| ** SAME200 245 PROCESS 3

Process 3:

```
I SI * CLEAR 200 COPYFF
245 230 SAME 38 245
MESSAGE 1 DONE
I NO * MESSAGE 0 DONE
```

Bien, espero que la rutina os sirva de algo, por lo menos teneis la seguridad de que funciona, pues es la misma que está en el Freeze Aliens, pero claro, vosotros, tendreis que hacerle las modificaciones necesarias para vuestras aventuras.

Limitaciones del PAW:

Dentro del PAW hay algunas características que creo de importante mención, como es el caso del problema de la página 0. En esta página es donde se introduce todo lo relacionado con la programación, es decir, que aunque vosotros hayais abierto otra página, la programación propiamente dicha (vocabulario, Tablas de procesos y respuestas) se introducirán en la página 0, por lo que tendréis que ser muy cuidadosos y tener muy claro si queréis hacer vuestra aventura para 48K o para 128K; si la hacéis para 48K no hay problema, pero si la hacéis para 128K es mejor que sigáis esta norma: Solamente cargar el PAW abrir otra página, en esta sólo introduciréis los mensajes, y cuando hayáis terminado (que seguramente no os quedará prácticamente memoria en esa página) abrir otra y empezar con localidades, gráficos y demás, así no tendréis problemas con la página 0. ¿OK?. Pues vale.

Otro gran problema es la cabecera Basic protegida de los programas realizados con ella, que a los que sólo conocen el PAW, tendrán que prescindir de pantalla de presentación, pero como aquí estamos para eso, aquí va la cabecera de las aventuras realizadas con el PAW, eso sí, en 48K:

```
10 CLEAR 28671: BORDER 0:
PAPER 0: INK 7: CLS
20 LOAD "" SCREEN$: POKE
23570,16
30 LOAD "" CODE: LOAD ""
CODE: LOAD "" CODE: POKE
23570,6
40 POKE 34902,0: POKE
34903,0: POKE 34904,0
50 RANDOMIZE USR 34793
```

Bueno PAWadictos, os dejo hasta el més que viene. Abur.

AVENTURAS ESPAÑOLAS

presenta

El Mundo Mágico

"A pesar del frío reinante esta noche, el sudor me empieza a resbalar por la frente. Desde aquí ya puedo divisar el pueblo, donde me espera mi maestro.
¡Maldita sea! ¿Dónde demonios estará la joya que me queda por entregar al Gran Brujo?. Quizás en las montañas, en el castillo de la malvada bruja, en la espesura del bosque...
pero ¿donde buscarla?
¡Oh, no!
La noche se desvanece, y con ella mi última esperanza de ser el sucesor de mi maestro.
Mi tiempo terminó..."

Pedidos contra-reembolso* a:
Alberto R. Cuesta
Aventuras Españolas
Po. de la Estación 15, 5-B
28807-Alcala de Henares
MADRID
Tfno. (91) 888-96-79

* PRECIO DE LA AVENTURA (48K O 128K)+INSTRUCCIONES: 800 PTS. (GASTOS DE ENVIO INCLUIDOS)

SMASH HITS

1º... Tolum y Cobal
2º... La estación
3º... Nick Tracy
4º... Abracadabra
5º... O-37
6º... Gwain the Halfling
7º... Robin Hood
8º... El ojo del dragón
9º... Aventura espacial
10º... Cozumel

Estas son las 10 aventuras que os recomendamos este mes. Como vereis, hay de todo, ingleses, españoles; antiguos, nuevos; de la bolsa, comerciales, etc...; fantásticos, espaciales, arqueológicos; de todas las clases, aunque todos son estupendos, al menos eso nos parece a nosotros.

RUTINAS

Seguimos esperando vuestras colaboraciones, pero mientras tanto...

Turnos y puntuación en pantalla

```
-SYSMESS:
1/
-PROCESO 2:
SAVEAT PROCESS X BACKAT
PRINTAT 0 25 TURUS SCORE DONE
-PROCESOS X
X= 3-255
```

PS: Todos los mensajes del sistema del "1" al "22" excepto el "4", Van en Blanco.

PPS: Mandad vuestras rutinas.

rutinas

He aquí una rutina que permite introducir claves de acceso para segundas partes, etc. Algunos dirán que ya apareció en cierta revista una rutina para eso, pero esta es mucho mejor, (!!!incluso es mejor que la del Jabato y la Aventura Original!!!)

La palabra "clave" puede estar compuesta de un verbo, un nombre y un adjetivo. En el ejemplo pondremos "caballo" como nombre (110 en el vocabulario), "fuerte" como adjetivo (111 en el vocabulario), y "galopa" como verbo (112 en el vocabulario)

La localidad 0 será esta:

Introduce la clave:

Y para que esté en color blanco, justo antes de escribir, pulsaremos EXTENDED MODE, CAPS SHIFT y 7, y justo después otra vez EXTENDED MODE, CAPS SHIFT y 8, para que deje el color como está.

EN PROCESOS 1:

```
- -  AT 0                    - -  AT 0
     INK 0                        ZERO 5
     LET 5 1                      DESC
```

RESPUESTAS:

```
* *  NOTEQ 33 112             * *  AT 0
     NOTEQ 34 110                  GOTO 1
     NOTEQ 35 111                  INK 7
     AT    0                       DESC
     DESC
```

También hay que tener en cuenta que si cada vez que nos matan hacemos END, nos pedirá la clave otra vez. Para evitarlo, debemos tener apuntadas las banderas que usamos en la aventura y donde pongo END cambiarlo por un CLEAR a todas las banderas, y un GOTO 1 y un DESC.

PROGRAMACION

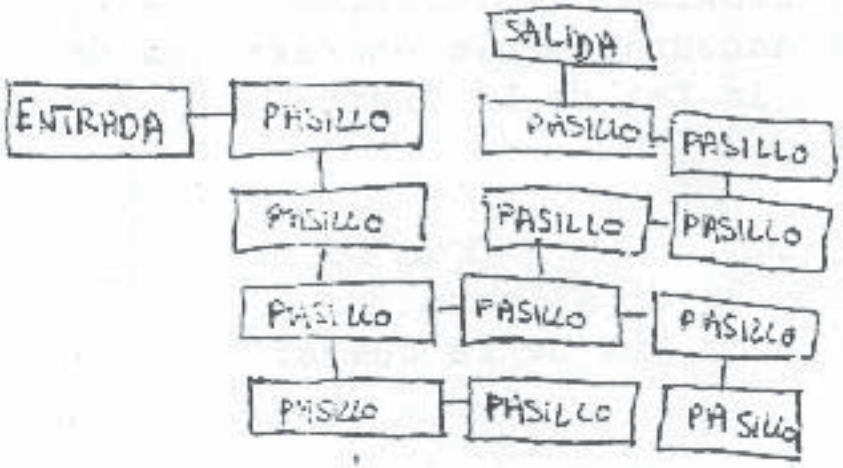
A pesar de que os dijimos que íbamos a comenzar con los PSI's en este ESPAC, vamos a hablar sobre los laberintos, el terror de los inexpertos, pero lo que da vida a las aventuras según los "veteranos".

El 90% de las aventuras llevan incluido algún laberinto. Hay desde dos hasta más de cien localidades.

Podemos encontrar varias clases de laberintos:

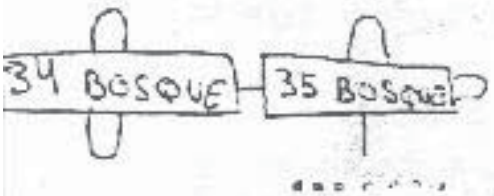
a) Clásicos.
b) Clásicos con repetición de localidades.
c) Ilógicos.

Los primeros son los típicos laberintos de toda la vida que tantas veces hemos resuelto en revistas de pasatiempos en las que unas localidades se entrelazan logicamente. Por ejemplo:



Son totalmente fáciles de mapear, y no es necesario siquiera ir dejando objetos, por lo que si quieres que tu aventura sea difícil (excepto en ciertas ocasiones en que son necesarios), no te recomendamos que utilices este tipo.

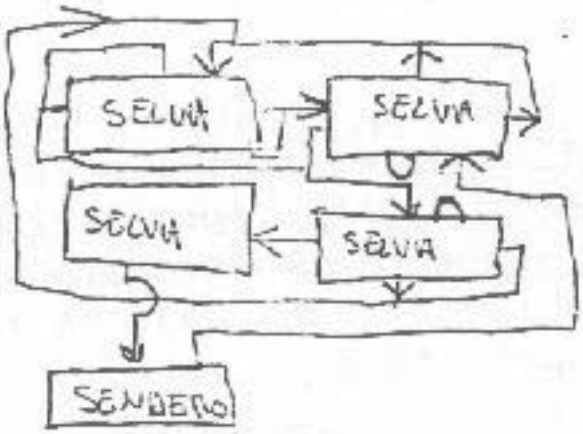
Los del segundo tipo son iguales que los anteriores, sólo que en los bordes del laberinto, si se intenta salir, se redescribe de nuevo la localidad actual, dando la impresión de que se ha llegado a otra distinta. Esto se hace conexionando la localidad con ella misma. Por ejemplo:



CONEXIONES
Localidad 34
N. 34, S 34, E 35, O 34
Localidad 35
N 35, E 35, O 34, S ...

Este es el tipo que usan los programadores de AD generalmente, aunque muchos programadores no lo usan porque les parece demasiado fáciles de resolver.

El tercero es el más difícil, pues como su propio nombre indica es ilógico, es decir, si por ejemplo de la sala de máquinas vamos al norte, llegamos al almacén, pero si del almacén vamos al sur, llegamos al corredor central. Son enormementes difíciles de resolver. Además se pueden usar también conexiones a la misma localidad. Un ejemplo:



En este tipo de laberintos se pueden dejar localidades aisladas y sin conexiones de entrada, como colmo de la perversidad, pero si se intenta mapear dejando objetos, se pierden en esa localidad. Este tipo de laberintos suelen hacer que el jugador se acuerde, no muy bien por cierto, de todos los familiares del programador.

Una cosa que es imperdonable, es que haya objetos con su localidad inicial en un laberinto, pues ayuda enormemente a mapearlo, especialmente si son laberintos de pocas localidades. Tampoco está bien hacer un laberinto de ciento y pico localidades, complica demasiado la cosa, pero sobre gustos no hay nada escrito.

Si teneis alguna duda o quereis que tratemos algún asunto en especial escribid al club poniendo en el sobre:
"SECCION DE PROGRAMACION. Elrond". Y será tratado en el siguiente CPAC.
Y hasta el próximo CPAC.

Elrond

rutinas

Os ofrecemos aquí la más maravillosa, estupenda y novedosa rutina: una rutina que os permitirá poder corregir errores en las frases tecleadas o poder repetirlas sin tener que reescribirlas.

Debemos insertar en el vocabulario lo siguiente:

I V 253 Ø
I P 254 2

En Respuestas tecleemos lo siguiente:

* * EQ 33 253
COPYFF 253 33

* * EQ 34 254
COPYFF 254 34

* * COPYFF 33 253
COPYFF 34 254

Aseguraros de que estas entradas queden al principio de la tabla, por el orden en que aquí están escritas.

Para usar esta rutina debemos tener en cuenta:

-Siempre que tecleemos V, se repetirá el último verbo, y siempre que tecleemos P el último nombre.

Si por ejemplo tecleamos COERG CUERDA, con que tecleemos COGER P valdrá.

Si tecleamos V se repetirá la última frase.

PROGRAMACION

!Por fín! Hoy vamos a comenzar a hablar sobre el largo y complicado tema de los PSI's, ese mundo de personajillos que pueblan nuestras aventuras. Desde esa rata que tanto nos hizo sufrir en "Don Quijote" hasta la bella y desgraciada Zyanya de "Cozumel". He puesto estos dos ejemplos como el del PSI que en realidad no existe, a uno que es bastante complicado. Los hay mejores y peores. Personalmente, creo que el mejor PSI es Gangalf, de "The hobbit".

Para empezar, hagamos una división de los tipos de PSI's. Comencemos sacando los dos tipos de los que hablamos antes. De los que son más complicados saquemos dos tipos más: fijos o móviles. Los fijos son muy simples, pues sólo necesitan una, a lo sumo dos tablas de procesos. El típico ejemplo lo tenemos en ese anciano que nunca nos deja pasar por un puente si no le damos determinado objeto, o si no acertamos una puñetera adivinanza. Algunos programadores, bastante vagos, por cierto, usan una sola tabla para hablar a todos los PSI's fijos. ¿Qué cómo lo hacen? Muy fácil. Si a un PSI que está en la localidad 27, y no se va a mover de ella, quieres decirle NELUCNE ET, en la tabla usada para hablarles ponen:

```
NELUCNE ET      AT  27
                MESSAGE ...
                ...
```

Y así se ahorran los bytes de tener que usar una tabla para cada uno. Si hay algo que se le vaya a poder decir a todos, no se pone ningún AT al principio.

Con este tipo de PSI's, nos ahorramos una rutina que nos indique cuales están presentes, pudiéndolo indicar en el texto de la localidad donde se encuentra.

Creo que de los PSI's fijos siempre se ha hablado mal injustamente, pues si en nuestra aventura necesitamos un dependiente, no creo que sea lógico que si le decimos que se dirija hacia el Sur, cumpla nuestra orden. O si no, no tenéis mas que fijaros en la aventura mejor programada: la vida real. Si en la realidad vamos a una tienda y le decimos al dependiente:"Vete al sur", lo más seguro es que, o llame a los loqueros, o que nos conteste: "Y tu a la mierda", en vez de cumplir nuestra orden.

Los móviles se subdividen en dos: los kolegas que te siguen, y los que van a su aire. Como ejemplo de los primeros tenemos a Taurus del Jabato, y de los segundos al Orco de "Gwain the Haffling". Los que te siguen, cuando lo hacen se setea una bandera, y cuando no, se limpia:

PROCESOS (Donde se habla al PSI)

```
SIGUEME _  ZERO (Nº de bandera)
           SET        "
           ...
```

ESPERAME _ NOTZERO (Nº de bandera)
CLEAR "
...

PROCESOS 1

* * NOTZERO (Nº de bandera)
COPYFF 38 (Bandera localidad PSI)

Los segundos tienen una tabla que los dirige. La manera más fácil de programarlos es numerar consecutivamente las localidades por donde se pasee, y en su tabla se aumenta(o disminuye) su bandera de localidad.

Como ejemplo vamos a programar entre todos un PSI, que podría ser un hobbit, que va a ser móvil, y que, aunque algo independiente, nos seguirá si se lo pedimos.

Para nuestro querido hobbit, de nombre Ordo, vamos a necesitar una localidad libre, ya veremos para qué, que podría ser la 99, y, de momento, 3 banderas: la 11, la 12 y la 13. La bandera 11 va a llevar el nº de localidad de Ordo, y la 12 y 13 ya veremos para qué usarlas.

Lo primero que debemos hacer es que podamos examinarlo, algo que todos deberíamos saber hacer. Bueno, por si acaso andamos mal de inspiración:

RESPUESTAS:

```
EX ORDO  SAME 38 11
         MESSAGE (Un joven hobbit muy
                  simpático.)
         DONE
```

Nuestro hobbit va a usar también dos tablas de procesos, para adquirir personalidad, pero de eso trataremos ya en el siguiente CPAC, pues creo que ya hemos hablado bastante por hoy, así que, se despide de vosotros desde LORIEN:

Elrond

Bueno, queridos y queridas, lectores viciosos que leeis desesperadamente esta sección. Se acabó la espera. Ha llegado la hora de hacer gráficos de verdad. Ha llegado la hora de pasar...

DEL BOCETO AL PIXEL

En el pasado CPAC cerrabamos (por el momento), el tema de los bocetos. De todos modos, si quereis que hablemos más del tema, si quereis que hablemos de construcciones, ropajes, edificios, de árboles o animales, escribir a mi dirección pidiéndolo, y, si hay mucha demanda, atenderé vuestras peticiones.

Bueno, el caso es que ahora el grafista, teniendo en la mano unos bocetos que le satisfacen a él y al programador, decide empezar a dejarse la vista haciendo gráficos y más gráficos basados en los bocetos que tantos sudores le han costado.

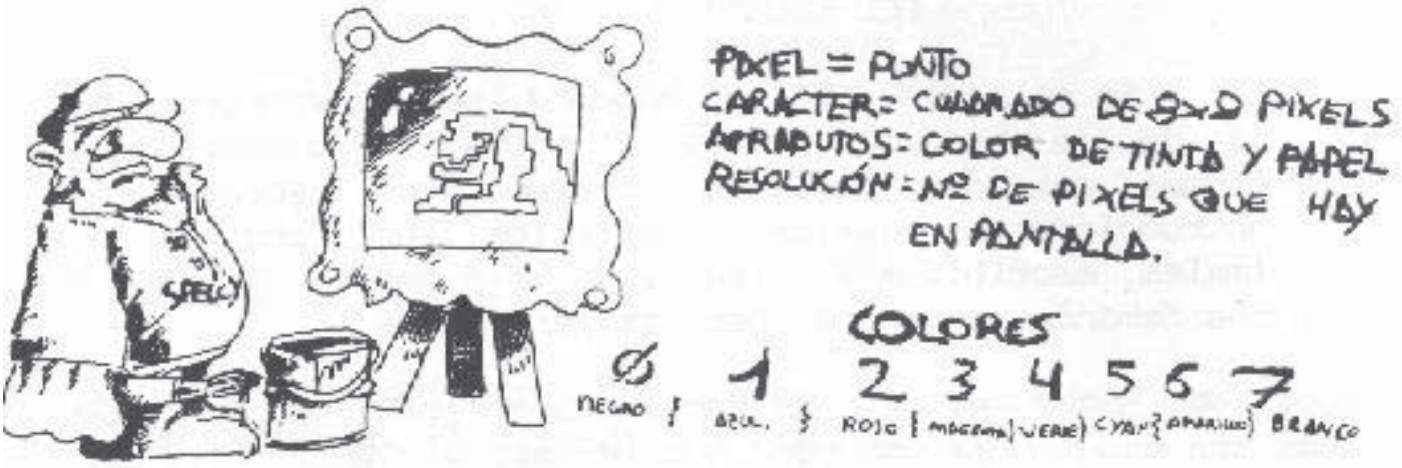
Por lo general es mas fácil hacer los gráficos que dibujar los bocetos. En muchas ocasiones, los programadores de una aventura piden a un dibujante consagrado que les haga los bocetos, y ellos mismos los pasan a la pantalla. Es más fácil hacer los gráficos, pero también es más aburrido. A la hora de dibujar no tienes ninguna restricción, pero a la hora de diseñar tienes muchas:

- la resolución
- los atributos o colores
- la cantidad de memoria que te ha concedido el programador
- un largo etc.(EEEEEEEETTTTTTTTTOOOOOOOO)

Como ya he dicho, es bastante fácil, y es por eso que no voy a tratar el tema en profundidad. Sólo explicar brevemente el PROBLEMON que representan los atributos.

En cada cuadrado de 8 x 8 pixels (puntos) sólo puedes utilizar dos colores a elegir entre una paleta de 8:

(FIG. 1)



Esto puede dar problemas si queremos hacer gráficos muy detallados a la vez. Hay tres métodos para solucionar el problema de los atributos. A saber:

1º-Hacer gráficos en blanco y negro, con lo que se consigue más detalle.

2º-Hacer gráficos en colores, pero muy separados unos de otros para que no se mezclen.

3º-Hacer gráficos a todo color. La más difícil, cara y bonita de las tres (la que suelo usar yo)

NOTA: Cuando digo en blanco y negro, no me refiero sólo a esos colores. Me refiero a cualquier gráfico que utilize un color de papel fijo y otro de tinta fijo.

Dado el éxito del miniconcurso de hace dos números, ahora os presento un nuevo enigma:

Debeis decir que tipo de gráfico predomina en cada uno de estos juegos:

- Jabato
- El ojo del dragón
- La estación
- Jack the ripper
- La isla del tesoro
- Los templos sagrados
- Fafnir

Enviáis la respuesta con una tarjeta que indique vuestros datos personales al apartado 450000000 de Andorra, y conseguireis un viaje para tres personas a ninguna parte. No, en serio. En el próximo número doy la solución...

Y ahora, como veo que no habeis tenido suficiente, vamos a hablar de otro tema que considero apasionante: un fenómeno que yo he dado a llamar *"CONTORSIONISMO PAWEIANO"*. No tiene nada que ver con el contorsionismo, aclararé al gracioso de turno.

Este fenómeno, que se debe intentar evitar, aunque sea caro, consiste en que los gráficos hechos con el PAW... Parecen hechos con el PAW. A veces no importa, pero, por favor, comparad la figura 2 con la 3. En la dos se nota que es un dibujo del PAW, porque los sombreados están rodeados por una linea.

(FIG. 2)

(FIG.3)

Ahora decidme, con la mano en el corazón y los pies sobre la tierra: ¿No es más bonito el tres, aunque salga mas caro por tener que borrar lineas que hayas hecho antes? EL QUE DIGA "NO" se la carga. La bestia ha matado a muchas personas, y una más no es nada.

Bueno, reflexionad sobre el tema, ahora que las vacaciones "sàn acabau". Si mortales, las bestias también tenemos vacaciones...

Repito una vez más mi llamamiento de siempre: si te preguntas cualquier cosa; crees que me explico mal; te crees más listo que yo, y quieres llevar la sección; me quieres mostrar gráficos o dibujos de los que te sientas orgullosos; o solamente quieres conocer a una simpática bestia de gelatina peluda y mirada profunda...¡ESCRIBE, condenado!

Pablo Jordi Atienza
C/. Colombia nº 52, 3º B
28016 MADRID

... Qué hasta ahora tan solo me ha escrito D.Eduardo Castilla Vico, y no me lo he comido... bueno, no entero.....

Pablo Jordi Atienza

(La bestia de la cueva)

PROGRAMACION

Desgraciadamente ya acabaron las vacaciones, y debemos continuar asistiendo a clase. Para intentar paliarlo, aunque sólo sea un poco, continuemos hablando de los PSI's. Antes de continuar, he de pediros disculpas por despedirme, en el CPAC número tres, desde Lorien, en lugar de hacerlo desde Rivendell.

Continuemos programando a nuestro Ordo. En este fanzine haremos su tabla de respuestas particular, y su movimiento. Una de las dos tablas de procesos del personaje, la usaremos para definirla como su vocabulario (como la tabla de respuestas, pero de un personaje). Introduzcamos entonces primero:

PROCESOS X

```
* *  MES (Hablas con Ordo)
      PARSE
      MESSAGE (. No entiende ni jota.)
      DONE

* *  MES (. Te contesta:)

_ _  MESSAGE ("Explicate mejor.")
      DONE
```

RESPUESTAS

```
DECIR ORDO  SAME 38 11 (Ver CPAC Número 3)
            PROCESS X
            DONE
```

Luego sólo nos queda ir introduciendo en procesos X las cosas que queramos que se le puedan decir a nuestro amigo, entre las cuales se encuentran "Hola", "Adiós", "¿Cómo te llamas?", etc..., y ya Ordo nos entenderá.

Entre las cosas que hay que añadir en procesos X, se encuentran:

```
SIGUEME _   ZERO 12
            MESSAGE ("Vale.".)
            SET 12
            DONE

ESPERAME _  NOTZERO 12
            MESSAGE ("Vale.".)
            CLEAR 12
            DONE
```

... Y en procesos 1:

```
* *  NOTZERO 12
      COPYFF 38 11

* *  NOTZERO 12
      PROCESS Y

* * Inicial AT 0
            ...
            ...
            LET 11 34
```

Para que el hobbit nos siga si se lo pedimos, y si no, que vaya a la tabla Y, donde deberemos insertar estas entradas:

```
* *  GT 11 28
      ZERO 13
      MINUS 11 1

* *  EQ 11 28
      SET 13

* *  LT 11 38
      NOTZERO 13
      PLUS 11 1

* *  EQ 11 38
      CLEAR 13
```

Y con ésto se movera de la localidad 28 a la 38. Este es un ejemplo muy sencillo, no es necesario complicarlo más, aunque si posible, y bastante fácil. En él véis como crear movimiento en un personaje. Luego pueden influir cantidad de factores, si es de dia o de noche, si le hemos dado determinado objeto, si hemos hablado a su amigo XXXXXX, y todas las que querais imaginaros.

Bueno, creo que, de momento, ésto es suficiente. Os saluda desde la Ultima Morada...

Elrond