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.

Program: T.J.Gilberts, G.Yeandle and P.Wade
Graphics: D. Peeke, K.Maddocks and A.Williams
Manuals: T.J.Gilberts

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.

Contents

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.

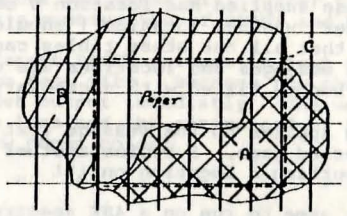
This effect can be overcome in one of four main ways:

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.

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.)

```
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
```

```
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
```

```
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!

Published by Gilsoft
2 Park Crescent, Barry, South Glamogan CF6 8HD
Telephone Barry (0446) 732765