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.

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

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).

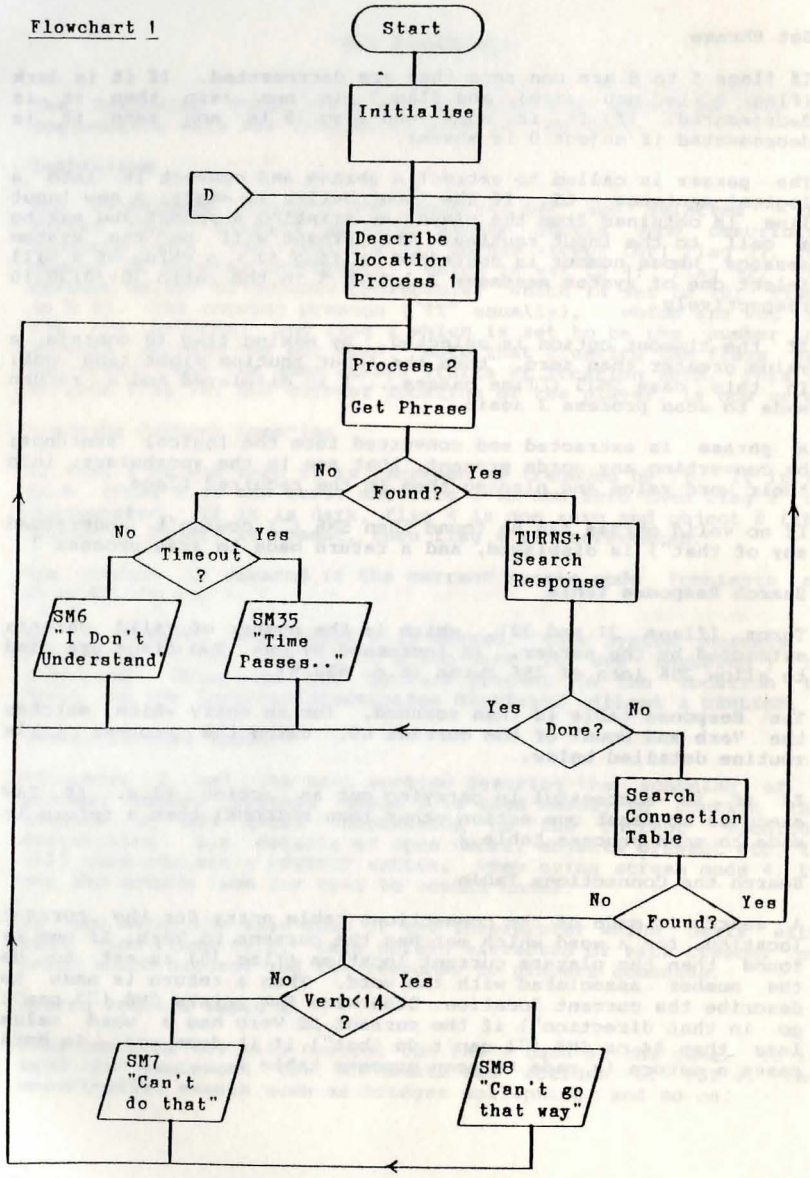
The Response table is then scanned, for an entry which matches the Verb and Noun1 of the current LS, using the process table routine detailed below.

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.

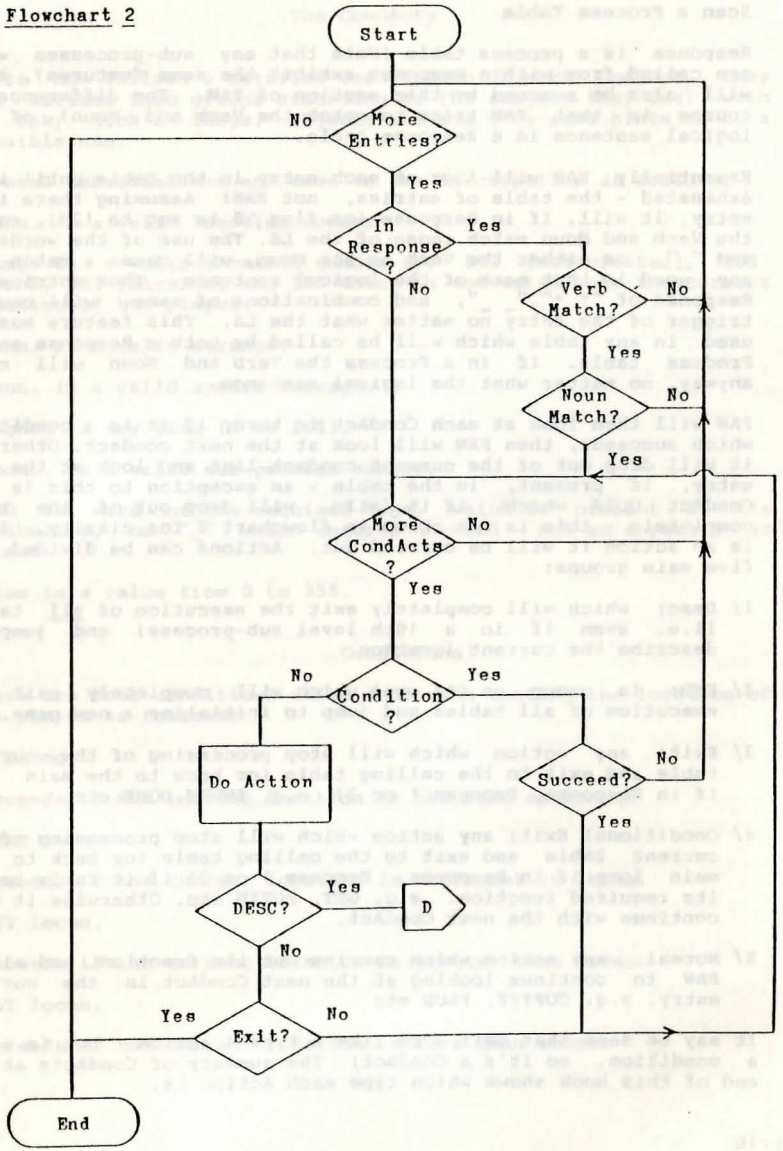
Search the Connections Table

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



Flowchart 2



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 all 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.

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 flagno1 flagno2

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

SUB flagno1 flagno2

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

COPYFF flagno1 flagno2

The contents of Flag flagno1 is copied to Flag flagno2.

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.

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

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 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 before 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:

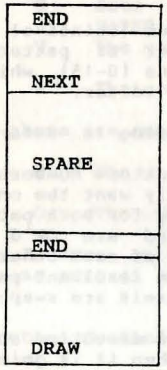
1/ Some user overlays may provide no way to Save and Load the data from their hunks to use in other databases. PAW-TELs' DPT is a trivial example. You could use the Hunk Management to do this using its Save and Load commands.

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;



The end of string marker

Any commands still undrawn

Available memory

Temporary end marker

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.

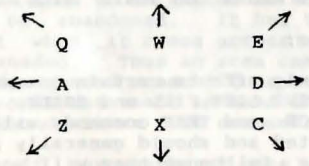
d) Calling the same routine you are drawing will cause a "Limit Reached" error as the limit of 10 subroutine levels will have been reached.

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.

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