Using inPAWS to convert John Wilson's Adventuron game 'Ramsbottom Smith and the Quest for the Yellow Spheroid' to the ZX Spectrum and other retro platforms

Background

John Wilson original wrote his 'Ramsbottom Smith and the Quest for the Yellow Spheroid' game using the Adventuron system by Chris Ainsley. Adventuron is an easy to use, modern authoring system, that runs through a web browser. It can, if you choose, have the look and feel of a classic 8-bit adventure game.

John has been pioneering the use of the software, using it to convert a lot of his old games from their original ZX Spectrum Quill versions, as well as creating a whole batch of new, original text adventures.

'Spheroid' was one such game, and I volunteered to port the new title to the ZX Spectrum, the original 8-bit home of Zenobi Software text adventures. Instead of using The Quill, I chose to utilise its successor, the Professional Adventure Writing System (or PAWS for short).

Because John designs his games and programs them in Adventuron in a very similar way to how he coded his Quilled adventures, it was a relatively straightforward task to port the game to the Spectrum. I've been asked to share some details about process and the tools used, which is what you'll find in this document.

<u>**The Tools: inPAWS**</u>

Although it would've been straightforward to code directly into the ZX Spectrum version of PAWS, running in an emulator, it would've required typing in all of John's (quite lengthy) text from the Adventuron version by hand.

I decided to use an intermediate step, a PC compiler-based system called inPAWs, instead. By utilising this tool, I could simply copy and paste the text from the original Adventuron source file into the inPAWS source file, doing a little bit of tidying up as I went.

Developed by Francisco Javier López, inPAWs was designed as a method of creating PAWS adventures for retro hardware by editing a text document on a modern computer. Once your source code is produced, you run it through a command line compiler which produces several different outputs from which you can make a game. More on those outputs later.

There are some limitations to the system, the main one being that inPAWS can only produce Spectrum 48K-sized adventures. This memory limitation was not an issue for this project.

inPAWS itself is quite a clever piece of software, giving the user the option of not have to think about a lot of the more bookkeeping aspects of adventure writing, such as keeping track of location, flag & object numbers. You can refer to most of these, and even chunks of code, using words instead. inPAWS also lets you organise your code a lot looser; defining vocabulary and writing code right next to objects, rather than separating everything into distinct blocks.

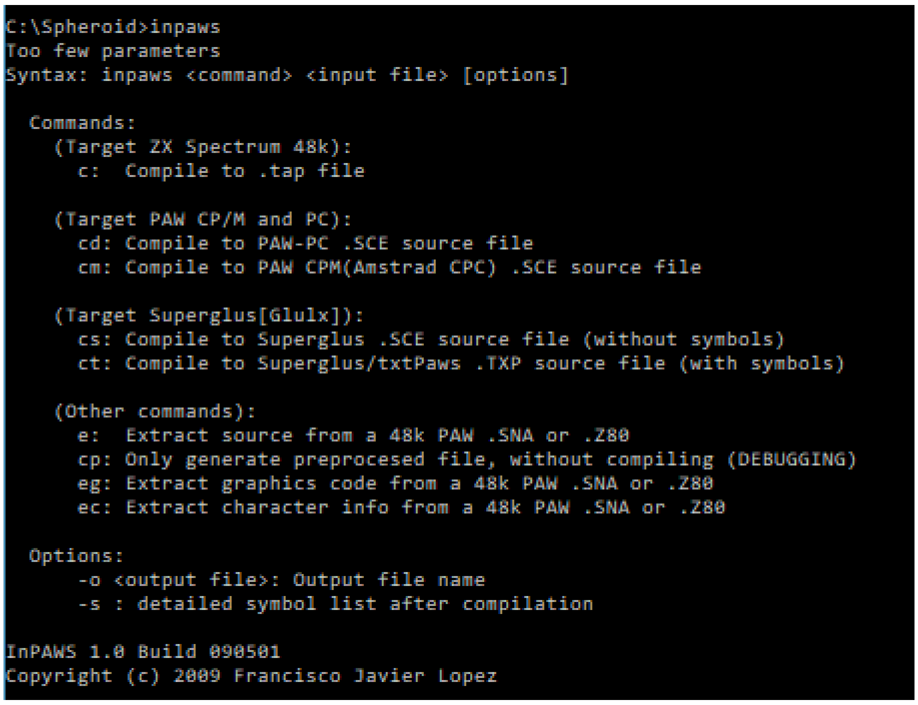
However, because I'm old school, I want full control of my code, so I basically created the adventure in inPAWS, exactly as I would have in the traditional PAWS. For that reason, this document will not be much use as a tutorial of how to program an adventure in inPAWS, but it should explain a few of the basics.

You can download inPAWS from here…. http://inpaws.speccy.org/indexEng.html

(Note: The inPAWS program is not a "Windows" program. It only runs from a command prompt. On Windows 10 you can right click on the Windows button and choose the option 'Command Prompt' to launch such a window. It uses old style DOS commands… if you want to do any serious retro developing then you probably need to learn these. However the command cd (choose directory) will be all you need for now.

```
cd foldername <-- opens up the subfolder with that name
cd .. <-- moves you back up a level.
```

Typing inpaws (when you're in the folder containing the inPAWS program file) will show you the correct syntax the program uses… More on that later…



```
C:\Spheroid>inpaws
Too few parameters
Syntax: inpaws <command> <input file> [options]

  Commands:
    (Target ZX Spectrum 48k):
      c:  Compile to .tap file

    (Target PAW CP/M and PC):
      cd: Compile to PAW-PC .SCE source file
      cm: Compile to PAW CPM(Amstrad CPC) .SCE source file

    (Target Superglus[Glulx]):
      cs: Compile to Superglus .SCE source file (without symbols)
      ct: Compile to Superglus/txtPaws .TXP source file (with symbols)

    (Other commands):
      e:  Extract source from a 48k PAW .SNA or .Z80
      cp: Only generate preprocesed file, without compiling (DEBUGGING)
      eg: Extract graphics code from a 48k PAW .SNA or .Z80
      ec: Extract character info from a 48k PAW .SNA or .Z80

  Options:
      -o <output file>: Output file name
      -s : detailed symbol list after compilation

InPAWS 1.0 Build 090501
Copyright (c) 2009 Francisco Javier Lopez
```

As you will see from the list above, you can also extract the source from an existing 48K Spectrum PAWS game… very useful to see how things work in inPAWS… particularly if you're familiar with intricacies of the adventure in question.

The Tools: Visual Code Studio

If, like me, you're familiar with the editor-based system of the Spectrum PAWS the whole idea of a text editor-based compiler system will probably seem quite alien. One of the advantages of the ZX Spectrum version of PAWS was that you could edit an entry in the database and almost instantly test if your code worked. The editor would also tell you off if you made silly typing errors and refuse to accept your input.

Working with source code in a text editor can be a lot more problematic, with no instant feedback and no error checking meaning you won't spot mistakes until your run it through a compiler. With compilers on old computers this could be quite a lengthy, time-consuming process.

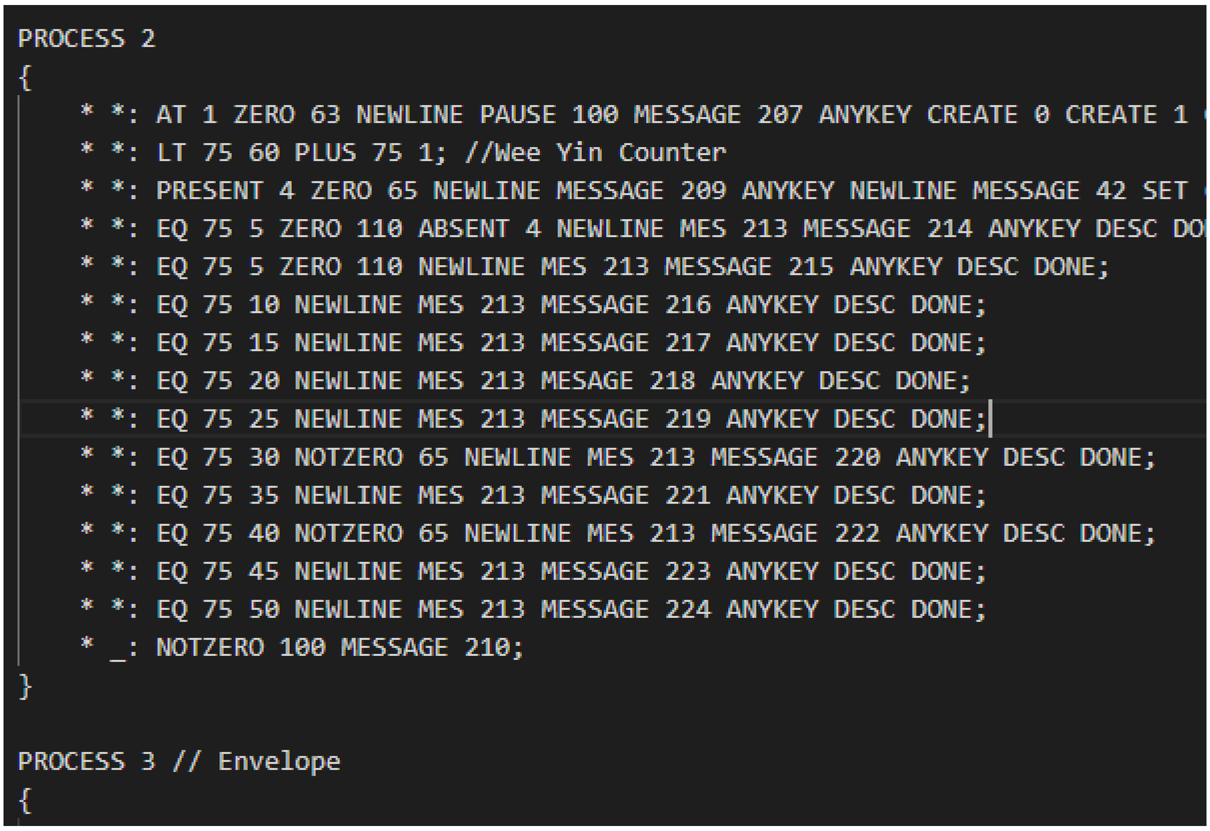
To make it easier to produce a suitable text file to feed into inPAWs, Chris Ainsley has worked with adventure author Stefan Vogt to create a syntax highlighter. What this neat little extension does is watches your code, as you type it into the editor, and prints the different elements in in an specific colour depending on their type. It makes your code a lot more readable and so it is considerably easier to spot where you've made a typing error or other mistake.

The syntax highlighter, together with inPAWS' extremely clear error reports, make the compiler-based system a breeze to use.

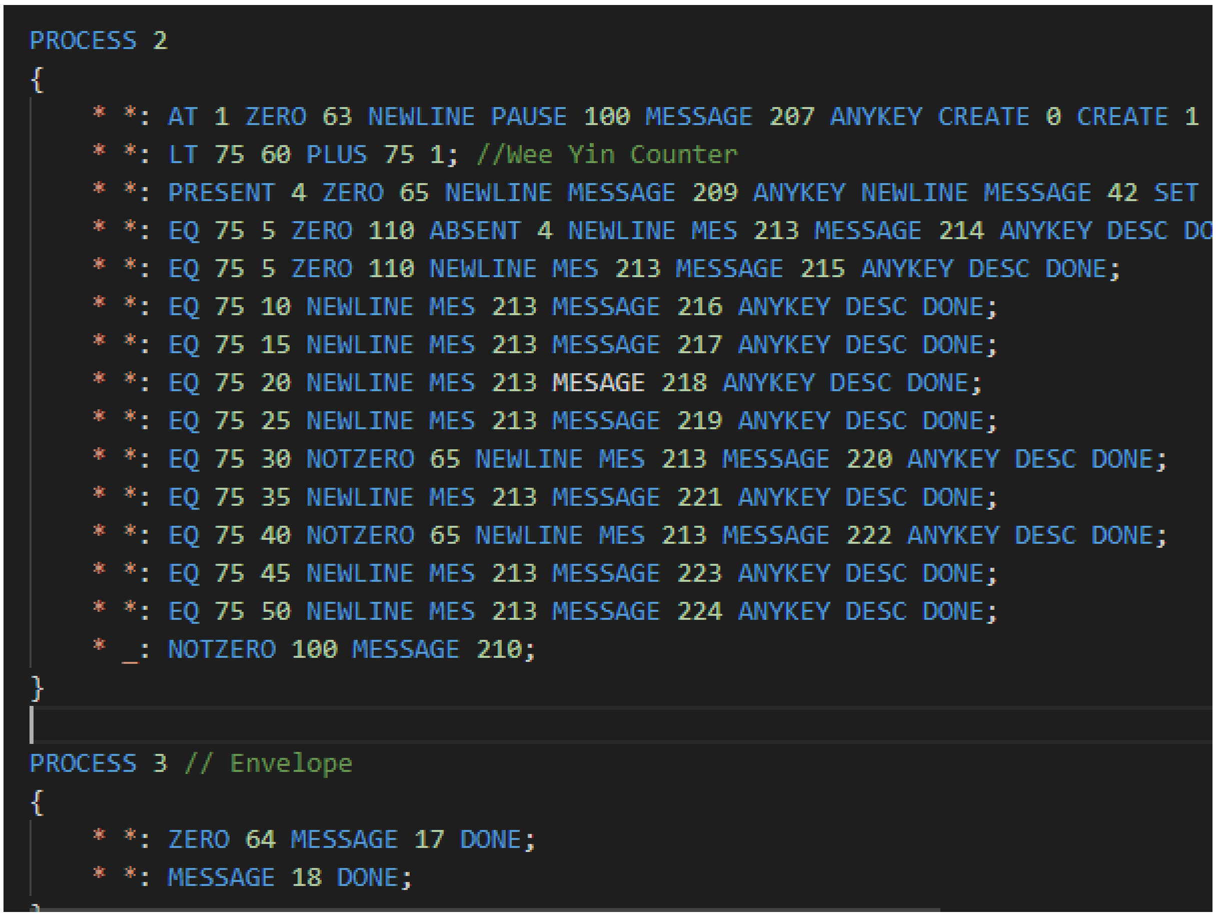
To use Chris' syntax highlighter you need to download and install the free program Visual Code Studio from Microsoft. https://code.visualstudio.com/ This is a great text editor and organisational tool in itself, but with Chris' extension (available from the marketplace at https://marketplace.visualstudio.com/items?itemName=ainslec.inpaws) it becomes indispensable for this type of work.

When the extension is installed it will recognise any file with the extension .paw as being an inPAWS file and should highlight the code accordingly.

Here's a comparison between a section of the Spheroid inPAWs code in VCS without the extension active and in VCS with the syntax highlighter on…



```
PROCESS 2
{
    * *: AT 1 ZERO 63 NEWLINE PAUSE 100 MESSAGE 207 ANYKEY CREATE 0 CREATE 1
    * *: LT 75 60 PLUS 75 1; //Wee Yin Counter
    * *: PRESENT 4 ZERO 65 NEWLINE MESSAGE 209 ANYKEY NEWLINE MESSAGE 42 SET
    * *: EQ 75 5 ZERO 110 ABSENT 4 NEWLINE MES 213 MESSAGE 214 ANYKEY DESC DO
    * *: EQ 75 5 ZERO 110 NEWLINE MES 213 MESSAGE 215 ANYKEY DESC DONE;
    * *: EQ 75 10 NEWLINE MES 213 MESSAGE 216 ANYKEY DESC DONE;
    * *: EQ 75 15 NEWLINE MES 213 MESSAGE 217 ANYKEY DESC DONE;
    * *: EQ 75 20 NEWLINE MES 213 MESAGE 218 ANYKEY DESC DONE;
    * *: EQ 75 25 NEWLINE MES 213 MESSAGE 219 ANYKEY DESC DONE;
    * *: EQ 75 30 NOTZERO 65 NEWLINE MES 213 MESSAGE 220 ANYKEY DESC DONE;
    * *: EQ 75 35 NEWLINE MES 213 MESSAGE 221 ANYKEY DESC DONE;
    * *: EQ 75 40 NOTZERO 65 NEWLINE MES 213 MESSAGE 222 ANYKEY DESC DONE;
    * *: EQ 75 45 NEWLINE MES 213 MESSAGE 223 ANYKEY DESC DONE;
    * *: EQ 75 50 NEWLINE MES 213 MESSAGE 224 ANYKEY DESC DONE;
    * _: NOTZERO 100 MESSAGE 210;
}

PROCESS 3 // Envelope
{
```



```
PROCESS 2
{
    * *: AT 1 ZERO 63 NEWLINE PAUSE 100 MESSAGE 207 ANYKEY CREATE 0 CREATE 1
    * *: LT 75 60 PLUS 75 1; //Wee Yin Counter
    * *: PRESENT 4 ZERO 65 NEWLINE MESSAGE 209 ANYKEY NEWLINE MESSAGE 42 SET
    * *: EQ 75 5 ZERO 110 ABSENT 4 NEWLINE MES 213 MESSAGE 214 ANYKEY DESC DO
    * *: EQ 75 5 ZERO 110 NEWLINE MES 213 MESSAGE 215 ANYKEY DESC DONE;
    * *: EQ 75 10 NEWLINE MES 213 MESSAGE 216 ANYKEY DESC DONE;
    * *: EQ 75 15 NEWLINE MES 213 MESSAGE 217 ANYKEY DESC DONE;
    * *: EQ 75 20 NEWLINE MES 213 MESAGE 218 ANYKEY DESC DONE;
    * *: EQ 75 25 NEWLINE MES 213 MESSAGE 219 ANYKEY DESC DONE;
    * *: EQ 75 30 NOTZERO 65 NEWLINE MES 213 MESSAGE 220 ANYKEY DESC DONE;
    * *: EQ 75 35 NEWLINE MES 213 MESSAGE 221 ANYKEY DESC DONE;
    * *: EQ 75 40 NOTZERO 65 NEWLINE MES 213 MESSAGE 222 ANYKEY DESC DONE;
    * *: EQ 75 45 NEWLINE MES 213 MESSAGE 223 ANYKEY DESC DONE;
    * *: EQ 75 50 NEWLINE MES 213 MESSAGE 224 ANYKEY DESC DONE;
    * _: NOTZERO 100 MESSAGE 210;
}

PROCESS 3 // Envelope
{
    * *: ZERO 64 MESSAGE 17 DONE;
    * *: MESSAGE 18 DONE;
```

Can you spot where I've made a typing error?

The inPAWS source code

inPAWs provides an example adventure source (that matches the tutorial adventure in the PAWS manual) and also gives you an example blank file that you can edit. The blank file is a good scaffold that I thoroughly recommend using. You can go through it and add to the existing structure to build up your adventure.

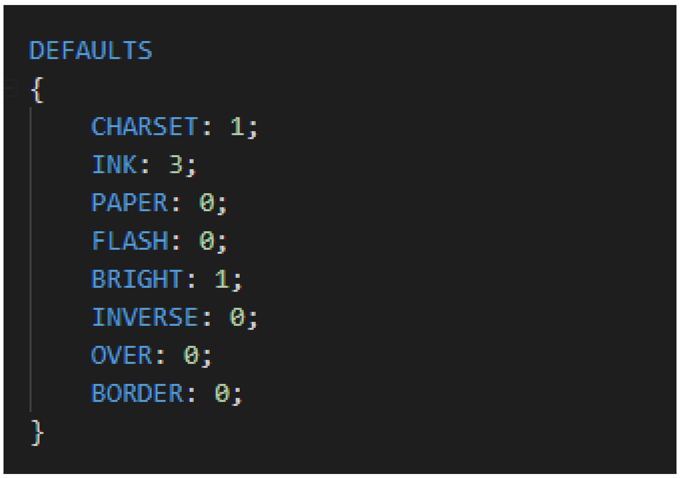
I won't go into detail about the actual process of coding the adventure. You will need to read other reference texts if you want to learn the PAWS and inPAWS language.

I will, however, share some examples from the Spheroid final code, alongside the Adventuron original source code where appropriate. Because I was doing things old-school, it should be very familiar to anyone who has used PAWs. You can view the complete inPAWS source code on my website.

Some key sections of the inPAWS source…

Defaults

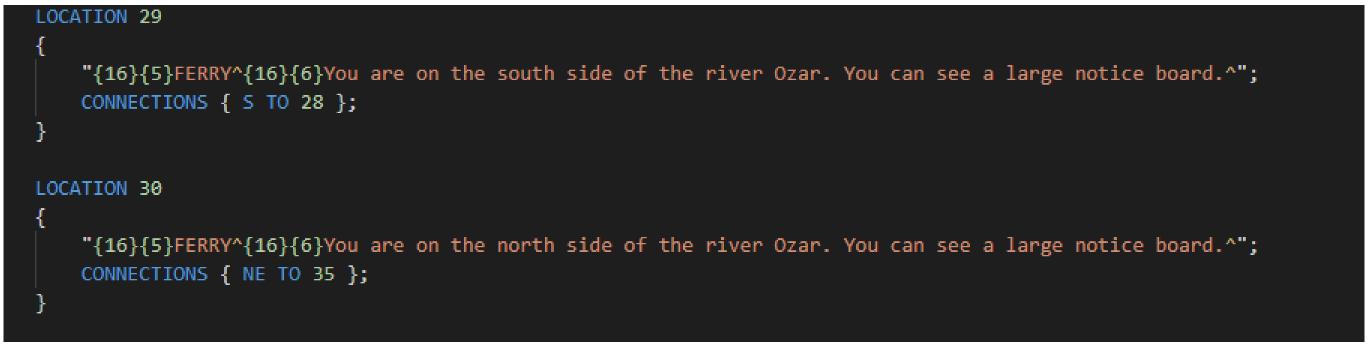
Sets the default colours for Spectrum adventures. Using Speccy colour numbers…



```
DEFAULTS
{
    CHARSET: 1;
    INK: 3;
    PAPER: 0;
    FLASH: 0;
    BRIGHT: 1;
    INVERSE: 0;
    OVER: 0;
    BORDER: 0;
}
```

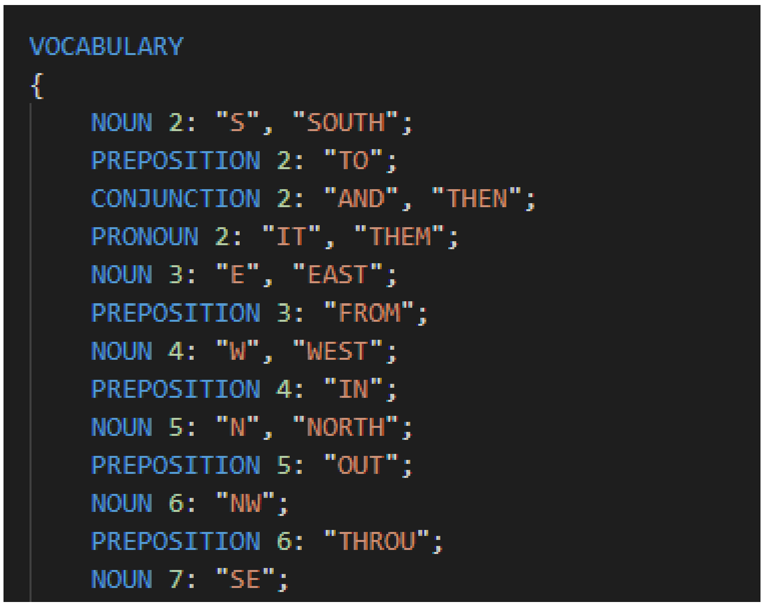
Locations

This is where you define the locations and the connections between them… as Spheroid is only a one location game the example below is from the Jack Lockerby game, Dragon Quest…



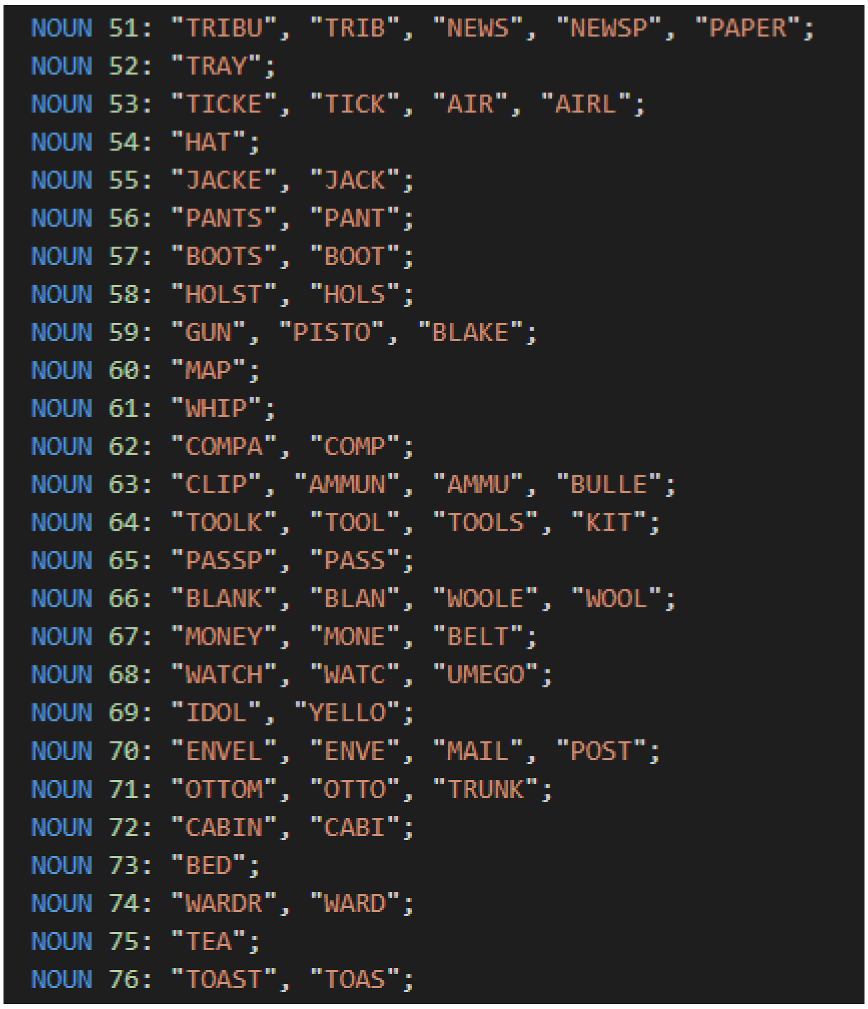
```
LOCATION 29
{
    "{16}{5}FERRY^{16}{6}You are on the south side of the river Ozar. You can see a large notice board.^";
    CONNECTIONS { S TO 28 };
}

LOCATION 30
{
    "{16}{5}FERRY^{16}{6}You are on the north side of the river Ozar. You can see a large notice board.^";
    CONNECTIONS { NE TO 35 };
}
```

Vocabulary



```
VOCABULARY
{
    NOUN 2: "S", "SOUTH";
    PREPOSITION 2: "TO";
    CONJUNCTION 2: "AND", "THEN";
    PRONOUN 2: "IT", "THEM";
    NOUN 3: "E", "EAST";
    PREPOSITION 3: "FROM";
    NOUN 4: "W", "WEST";
    PREPOSITION 4: "IN";
    NOUN 5: "N", "NORTH";
    PREPOSITION 5: "OUT";
    NOUN 6: "NW";
    PREPOSITION 6: "THROU";
    NOUN 7: "SE";
```

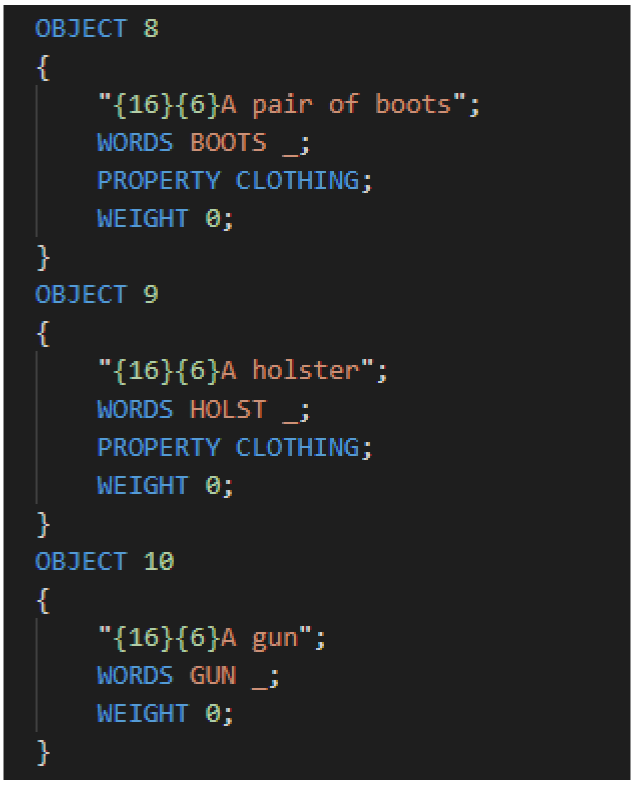
Vocabulary can be defined, just as in PAWS with different categories of words. In the example below you can see how you can add different synonyms for each word, and also shortened abbreviations if you wish. You just need to think carefully about potential conflicts, such as the preposition UNDER and the object UNDERWEAR, which is also shortened by the parser to UNDER.



```
NOUN 51: "TRIBU", "TRIB", "NEWS", "NEWSP", "PAPER";
NOUN 52: "TRAY";
NOUN 53: "TICKE", "TICK", "AIR", "AIRL";
NOUN 54: "HAT";
NOUN 55: "JACKE", "JACK";
NOUN 56: "PANTS", "PANT";
NOUN 57: "BOOTS", "BOOT";
NOUN 58: "HOLST", "HOLS";
NOUN 59: "GUN", "PISTO", "BLAKE";
NOUN 60: "MAP";
NOUN 61: "WHIP";
NOUN 62: "COMPA", "COMP";
NOUN 63: "CLIP", "AMMUN", "AMMU", "BULLE";
NOUN 64: "TOOLK", "TOOL", "TOOLS", "KIT";
NOUN 65: "PASSP", "PASS";
NOUN 66: "BLANK", "BLAN", "WOOLE", "WOOL";
NOUN 67: "MONEY", "MONE", "BELT";
NOUN 68: "WATCH", "WATC", "UMEGO";
NOUN 69: "IDOL", "YELLO";
NOUN 70: "ENVEL", "ENVE", "MAIL", "POST";
NOUN 71: "OTTOM", "OTTO", "TRUNK";
NOUN 72: "CABIN", "CABI";
NOUN 73: "BED";
NOUN 74: "WARDR", "WARD";
NOUN 75: "TEA";
NOUN 76: "TOAST", "TOAS";
```

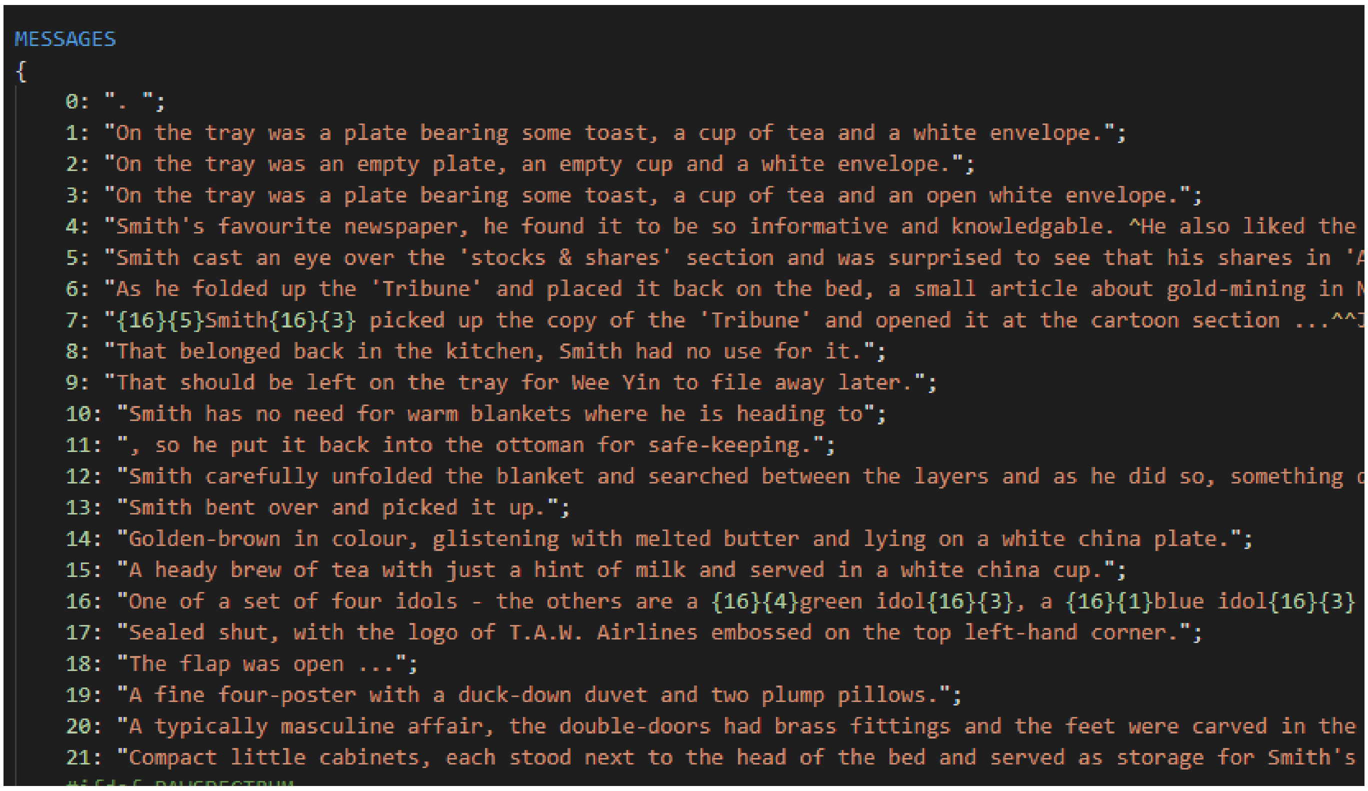
Objects

Objects can be wearable, if they're set as clothing. They can be containers. They can all have individual weights. You attach a word to each object so the parser can recognise them.



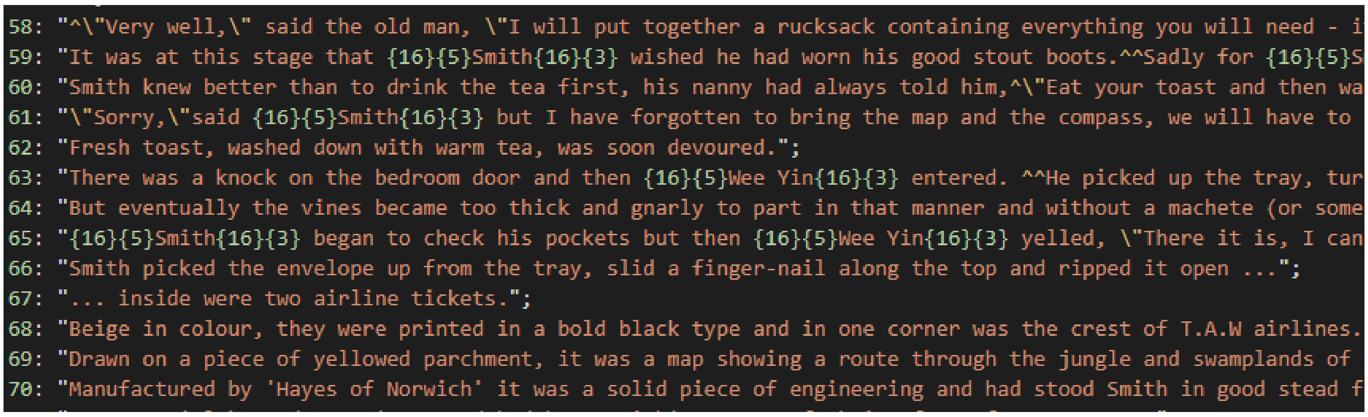
```
OBJECT 8
{
    "{16}{6}A pair of boots";
    WORDS BOOTS _;
    PROPERTY CLOTHING;
    WEIGHT 0;
}
OBJECT 9
{
    "{16}{6}A holster";
    WORDS HOLST _;
    PROPERTY CLOTHING;
    WEIGHT 0;
}
OBJECT 10
{
    "{16}{6}A gun";
    WORDS GUN _;
    WEIGHT 0;
}
```

Messages



```
MESSAGES
{
    0: ". ";
    1: "On the tray was a plate bearing some toast, a cup of tea and a white envelope.";
    2: "On the tray was an empty plate, an empty cup and a white envelope.";
    3: "On the tray was a plate bearing some toast, a cup of tea and an open white envelope.";
    4: "Smith's favourite newspaper, he found it to be so informative and knowledgable. ^He also liked the
    5: "Smith cast an eye over the 'stocks & shares' section and was surprised to see that his shares in '
    6: "As he folded up the 'Tribune' and placed it back on the bed, a small article about gold-mining in
    7: "{16}{5}Smith{16}{3} picked up the copy of the 'Tribune' and opened it at the cartoon section ...^^
    8: "That belonged back in the kitchen, Smith had no use for it.";
    9: "That should be left on the tray for Wee Yin to file away later.";
    10: "Smith has no need for warm blankets where he is heading to";
    11: ", so he put it back into the ottoman for safe-keeping.";
    12: "Smith carefully unfolded the blanket and searched between the layers and as he did so, something
    13: "Smith bent over and picked it up.";
    14: "Golden-brown in colour, glistening with melted butter and lying on a white china plate.";
    15: "A heady brew of tea with just a hint of milk and served in a white china cup.";
    16: "One of a set of four idols - the others are a {16}{4}green idol{16}{3}, a {16}{1}blue idol{16}{3}
    17: "Sealed shut, with the logo of T.A.W. Airlines embossed on the top left-hand corner.";
    18: "The flap was open ...";
    19: "A fine four-poster with a duck-down duvet and two plump pillows.";
    20: "A typically masculine affair, the double-doors had brass fittings and the feet were carved in the
    21: "Compact little cabinets, each stood next to the head of the bed and served as storage for Smith's
```

In the messages above the codes {16}{5} and {16}{3} are used to change the colour of the text. The second of the pair of numbers selects the ink colour. 5 is cyan and 3 is magenta… Look on a real Spectrum 48K keyboard to see why!

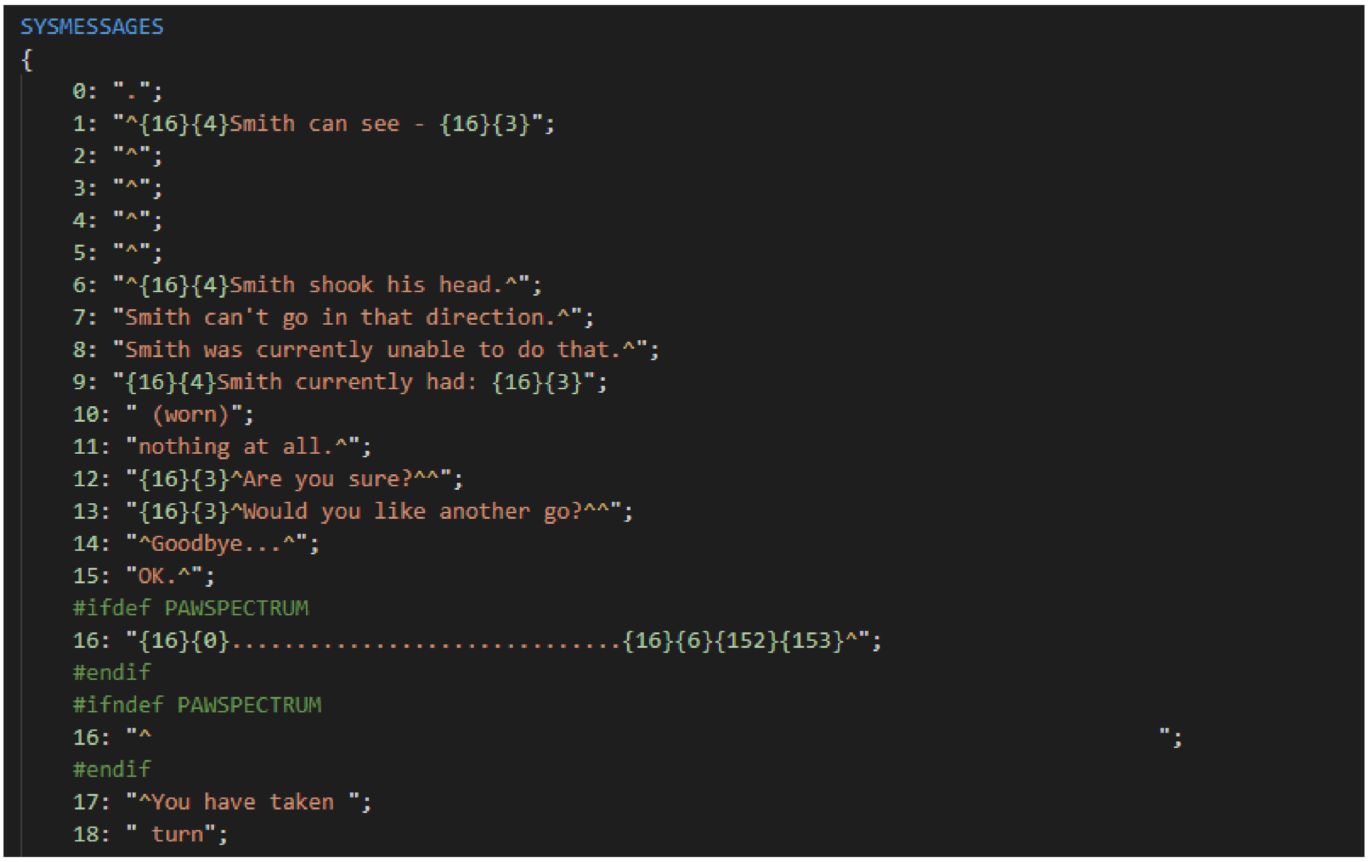


```
58: "^\"Very well,\" said the old man, \"I will put together a rucksack containing everything you will need - i
59: "It was at this stage that {16}{5}Smith{16}{3} wished he had worn his good stout boots.^^Sadly for {16}{5}S
60: "Smith knew better than to drink the tea first, his nanny had always told him,^\"Eat your toast and then wa
61: "\"Sorry,\"said {16}{5}Smith{16}{3} but I have forgotten to bring the map and the compass, we will have to
62: "Fresh toast, washed down with warm tea, was soon devoured.";
63: "There was a knock on the bedroom door and then {16}{5}Wee Yin{16}{3} entered. ^^He picked up the tray, tur
64: "But eventually the vines became too thick and gnarly to part in that manner and without a machete (or some
65: "{16}{5}Smith{16}{3} began to check his pockets but then {16}{5}Wee Yin{16}{3} yelled, \"There it is, I can
66: "Smith picked the envelope up from the tray, slid a finger-nail along the top and ripped it open ...";
67: "... inside were two airline tickets.";
68: "Beige in colour, they were printed in a bold black type and in one corner was the crest of T.A.W airlines.
69: "Drawn on a piece of yellowed parchment, it was a map showing a route through the jungle and swamplands of
70: "Manufactured by 'Hayes of Norwich' it was a solid piece of engineering and had stood Smith in good stead f
```

Speech marks must be proceeded by a backslash \. The ^ is used to force a line break.

System Messages

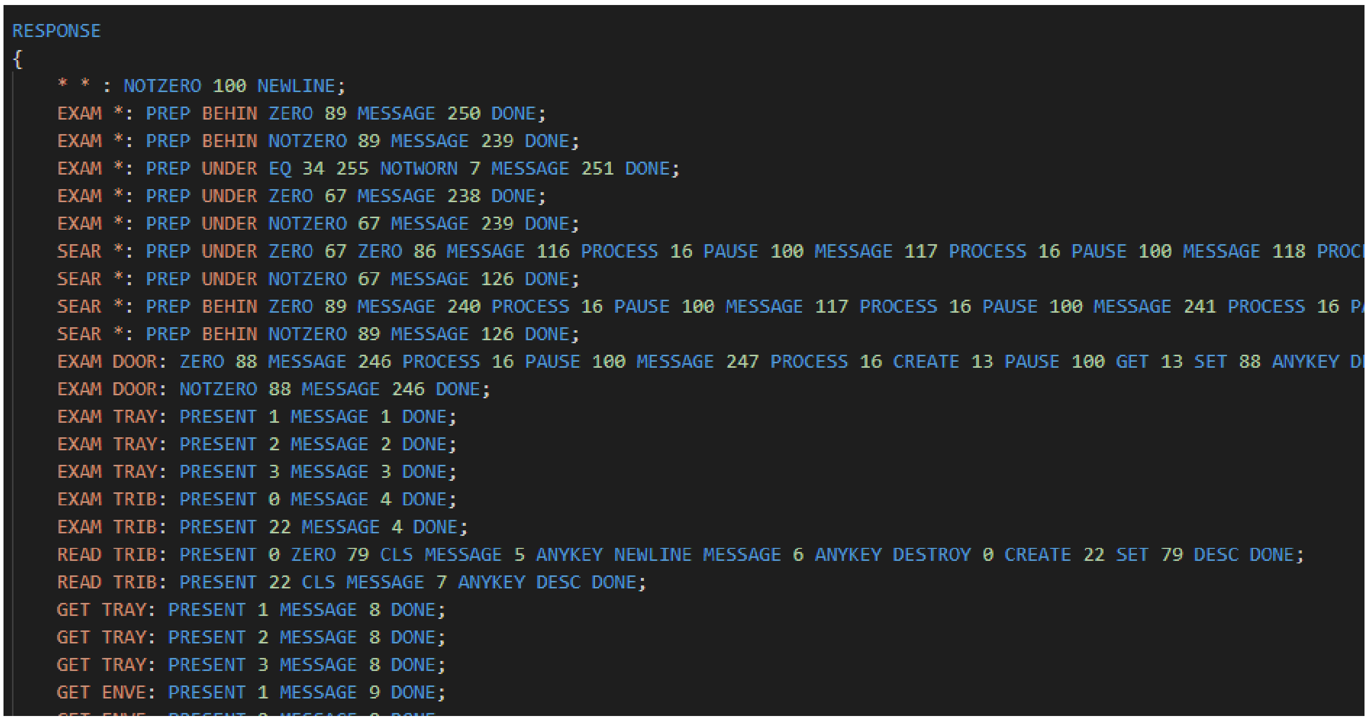
Many of the system messages are reserved for the PAWS program to use, although you can change them from the defaults. I will explain the sections highlighted in green a little later on.



```
SYSMESSAGES
{
    0: ".";
    1: "^{16}{4}Smith can see - {16}{3}";
    2: "^";
    3: "^";
    4: "^";
    5: "^";
    6: "^{16}{4}Smith shook his head.^";
    7: "Smith can't go in that direction.^";
    8: "Smith was currently unable to do that.^";
    9: "{16}{4}Smith currently had: {16}{3}";
    10: " (worn)";
    11: "nothing at all.^";
    12: "{16}{3}^Are you sure?^^";
    13: "{16}{3}^Would you like another go?^^";
    14: "^Goodbye...^";
    15: "OK.^";
    #ifdef PAWSPECTRUM
    16: "{16}{0}..............................{16}{6}{152}{153}^";
    #endif
    #ifndef PAWSPECTRUM
    16: "^                                                                 ";
    #endif
    17: "^You have taken ";
    18: " turn";
```

Response table

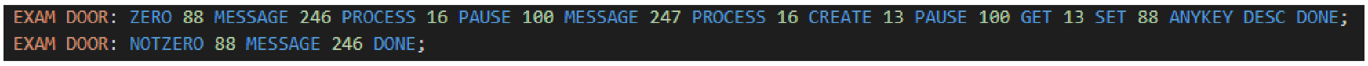
The response table will be very familiar to anyone who has used the PAWS.



```
RESPONSE
{
    * * : NOTZERO 100 NEWLINE;
    EXAM *: PREP BEHIN ZERO 89 MESSAGE 250 DONE;
    EXAM *: PREP BEHIN NOTZERO 89 MESSAGE 239 DONE;
    EXAM *: PREP UNDER EQ 34 255 NOTWORN 7 MESSAGE 251 DONE;
    EXAM *: PREP UNDER ZERO 67 MESSAGE 238 DONE;
    EXAM *: PREP UNDER NOTZERO 67 MESSAGE 239 DONE;
    SEAR *: PREP UNDER ZERO 67 ZERO 86 MESSAGE 116 PROCESS 16 PAUSE 100 MESSAGE 117 PROCESS 16 PAUSE 100 MESSAGE 118 PROC
    SEAR *: PREP UNDER NOTZERO 67 MESSAGE 126 DONE;
    SEAR *: PREP BEHIN ZERO 89 MESSAGE 240 PROCESS 16 PAUSE 100 MESSAGE 117 PROCESS 16 PAUSE 100 MESSAGE 241 PROCESS 16 P
    SEAR *: PREP BEHIN NOTZERO 89 MESSAGE 126 DONE;
    EXAM DOOR: ZERO 88 MESSAGE 246 PROCESS 16 PAUSE 100 MESSAGE 247 PROCESS 16 CREATE 13 PAUSE 100 GET 13 SET 88 ANYKEY D
    EXAM DOOR: NOTZERO 88 MESSAGE 246 DONE;
    EXAM TRAY: PRESENT 1 MESSAGE 1 DONE;
    EXAM TRAY: PRESENT 2 MESSAGE 2 DONE;
    EXAM TRAY: PRESENT 3 MESSAGE 3 DONE;
    EXAM TRIB: PRESENT 0 MESSAGE 4 DONE;
    EXAM TRIB: PRESENT 22 MESSAGE 4 DONE;
    READ TRIB: PRESENT 0 ZERO 79 CLS MESSAGE 5 ANYKEY NEWLINE MESSAGE 6 ANYKEY DESTROY 0 CREATE 22 SET 79 DESC DONE;
    READ TRIB: PRESENT 22 CLS MESSAGE 7 ANYKEY DESC DONE;
    GET TRAY: PRESENT 1 MESSAGE 8 DONE;
    GET TRAY: PRESENT 2 MESSAGE 8 DONE;
    GET TRAY: PRESENT 3 MESSAGE 8 DONE;
    GET ENVE: PRESENT 1 MESSAGE 9 DONE;
```

The system scans through the response table, looking to match the pink works with the input given by the player. The CONDACTS in blue are very easy to understand.

For example…

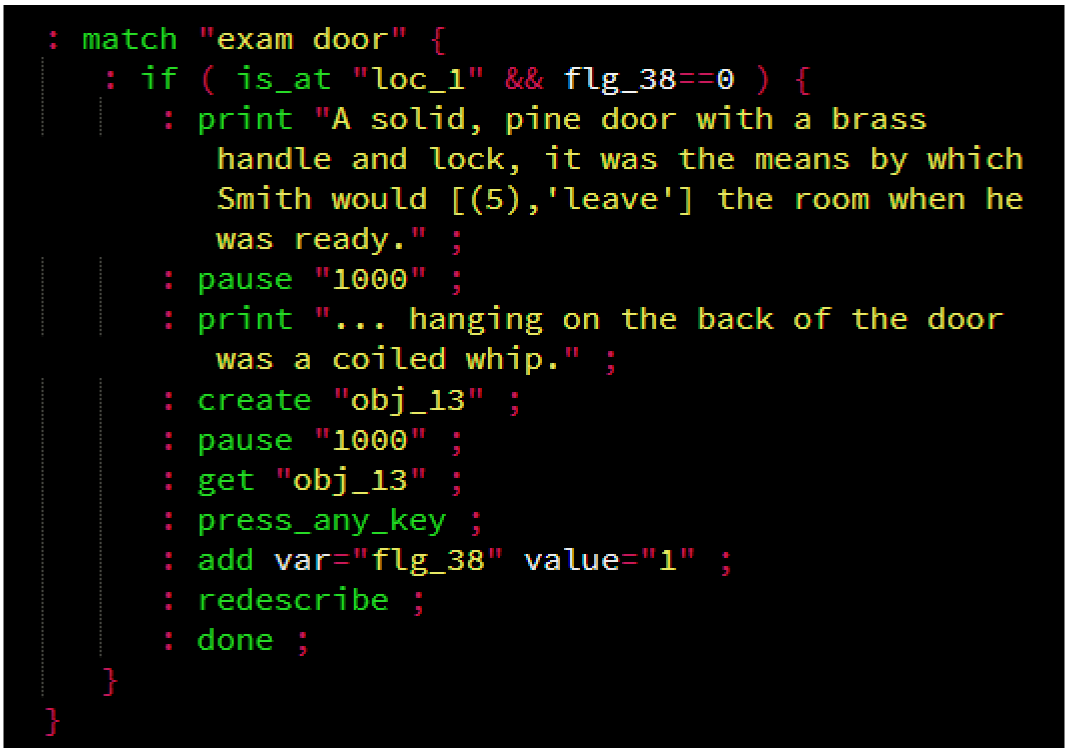


```
EXAM DOOR: ZERO 88 MESSAGE 246 PROCESS 16 PAUSE 100 MESSAGE 247 PROCESS 16 CREATE 13 PAUSE 100 GET 13 SET 88 ANYKEY DESC DONE;
EXAM DOOR: NOTZERO 88 MESSAGE 246 DONE;
```

These two lines deal with the response to the player typing in EXAMINE DOOR. Ignore the PROCESS 16 bits (that's something connected with adding extra line spacing). Let's look at the important parts of the first of the two lines…

If flag 88 is zero, as it is at the start of the game, it prints a message (246) about the door.
It waits (PAUSE 100) prints a second message (247) about finding a whip.
It then creates object 13, the whip (CREATE 13), waits (PAUSE 100), makes the player take the object (GET 13).
It ends up by setting flag 88 to a non-zero value (SET 88) so that particular line isn't called a second time should the player choose to examine the door again.

Here's the Adventuron version of the same code. It's virtually identical. John does an additional check to see if the player is at location 1… as the player is always at location 1 I've omitted that in the inPAWS version…



```
: match "exam door" {
   : if ( is_at "loc_1" && flg_38==0 ) {
      : print "A solid, pine door with a brass
         handle and lock, it was the means by which
         Smith would [(5),'leave'] the room when he
         was ready." ;
      : pause "1000" ;
      : print "... hanging on the back of the door
         was a coiled whip." ;
      : create "obj_13" ;
      : pause "1000" ;
      : get "obj_13" ;
      : press_any_key ;
      : add var="flg_38" value="1" ;
      : redescribe ;
      : done ;
   }
}
```

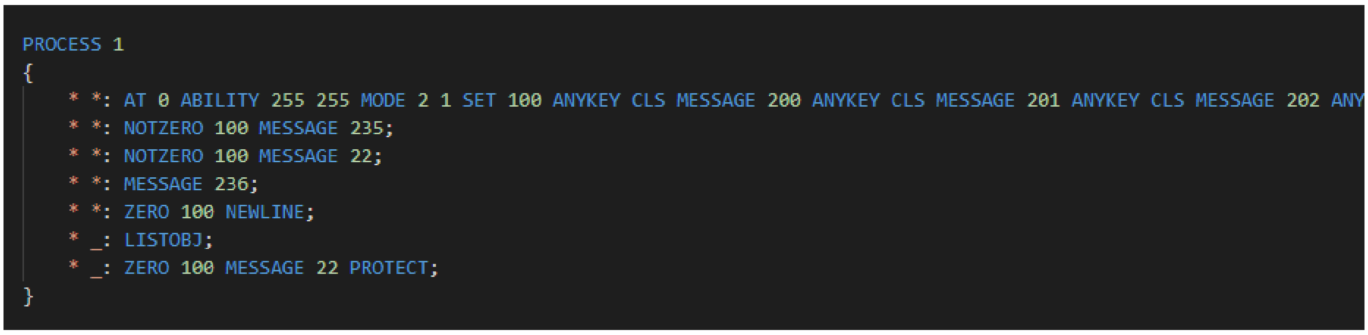
In the Adventuron version flag 38 is used where flag 88 was used in the inPAWS version. PAWS reserves certain flag numbers for use by the system (flag 38 actually holds the player's location in PAWS) so refer to the PAWS manuals to find values that are safe to use in inPAWS (or let the inPAWS system be in charge of assigning numbers… as a tiny bit of a control freak, I can't do that personally!)

Process tables

The process tables in PAWS work a lot like the response table but they don't (usually) match things against the player's input. Process 1 and 2 are used for special system purposes but you can define additional processes yourself to use for subroutines or to provide additional clarity and structure to your code.

Process 1….

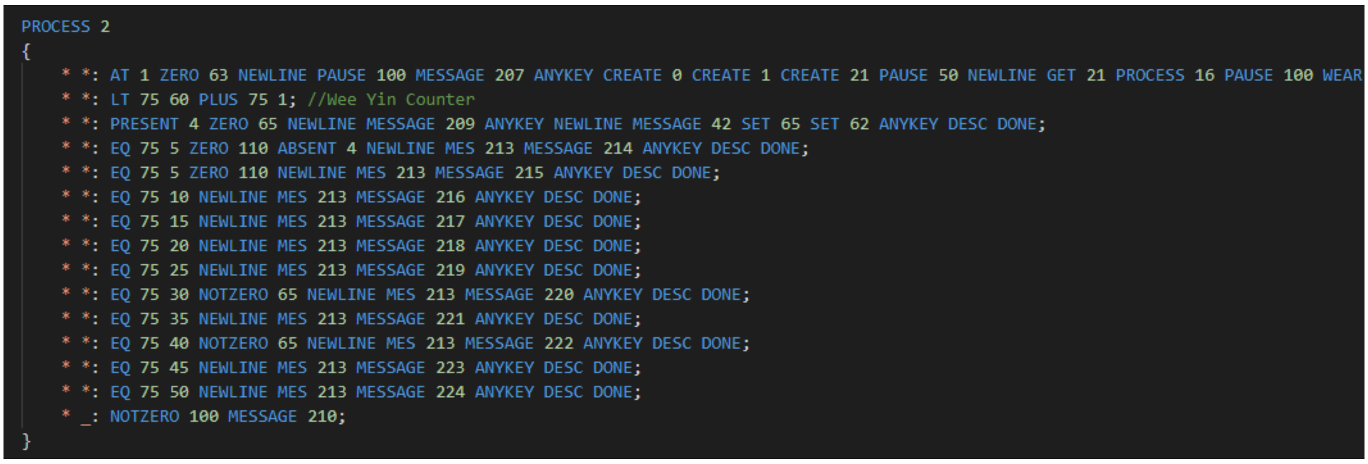
Process 1 is mainly used to add extra information to a location description. The commands in process 1 run before process 2 and before the player is allowed to type their command. In Spheroid, the process 1 table sorts out some of the location text formatting. The first entry (* *: AT 0 ABILITY 255 255…) is triggered on the title page (location 0) and sets the values of some key variables before printing up the intro text.



```
PROCESS 1
{
    * *: AT 0 ABILITY 255 255 MODE 2 1 SET 100 ANYKEY CLS MESSAGE 200 ANYKEY CLS MESSAGE 201 ANYKEY CLS MESSAGE 202 ANY
    * *: NOTZERO 100 MESSAGE 235;
    * *: NOTZERO 100 MESSAGE 22;
    * *: MESSAGE 236;
    * *: ZERO 100 NEWLINE;
    * _: LISTOBJ;
    * _: ZERO 100 MESSAGE 22 PROTECT;
}
```

Process 2…

Process 2 can be considered the PAWS' turn at the game. It's triggered after the location is described but also after every single turn by the player. In Spheroid this table is mostly used to run a counter that triggers the times when Wee Yin comes into the room and says something. Flag 75 is increased each turn (until it gets to 60) and if the flag equals certain amounts the commands next to it are carried out. The command DONE will always exit the table and no further parts of the table will be processed.



```
PROCESS 2
{
    * *: AT 1 ZERO 63 NEWLINE PAUSE 100 MESSAGE 207 ANYKEY CREATE 0 CREATE 1 CREATE 21 PAUSE 50 NEWLINE GET 21 PROCESS 16 PAUSE 100 WEAR
    * *: LT 75 60 PLUS 75 1; //Wee Yin Counter
    * *: PRESENT 4 ZERO 65 NEWLINE MESSAGE 209 ANYKEY NEWLINE MESSAGE 42 SET 65 SET 62 ANYKEY DESC DONE;
    * *: EQ 75 5 ZERO 110 ABSENT 4 NEWLINE MES 213 MESSAGE 214 ANYKEY DESC DONE;
    * *: EQ 75 5 ZERO 110 NEWLINE MES 213 MESSAGE 215 ANYKEY DESC DONE;
    * *: EQ 75 10 NEWLINE MES 213 MESSAGE 216 ANYKEY DESC DONE;
    * *: EQ 75 15 NEWLINE MES 213 MESSAGE 217 ANYKEY DESC DONE;
    * *: EQ 75 20 NEWLINE MES 213 MESSAGE 218 ANYKEY DESC DONE;
    * *: EQ 75 25 NEWLINE MES 213 MESSAGE 219 ANYKEY DESC DONE;
    * *: EQ 75 30 NOTZERO 65 NEWLINE MES 213 MESSAGE 220 ANYKEY DESC DONE;
    * *: EQ 75 35 NEWLINE MES 213 MESSAGE 221 ANYKEY DESC DONE;
    * *: EQ 75 40 NOTZERO 65 NEWLINE MES 213 MESSAGE 222 ANYKEY DESC DONE;
    * *: EQ 75 45 NEWLINE MES 213 MESSAGE 223 ANYKEY DESC DONE;
    * *: EQ 75 50 NEWLINE MES 213 MESSAGE 224 ANYKEY DESC DONE;
    * _: NOTZERO 100 MESSAGE 210;
}
```

Process 3 onwards can be created and defined by the user themselves. In Spheroid processes 3-15 are used to build up the various ending text.

I hope that gives you an overview of the inPAWS file. Please feel free to have a nose through the complete source file to see how it all works.

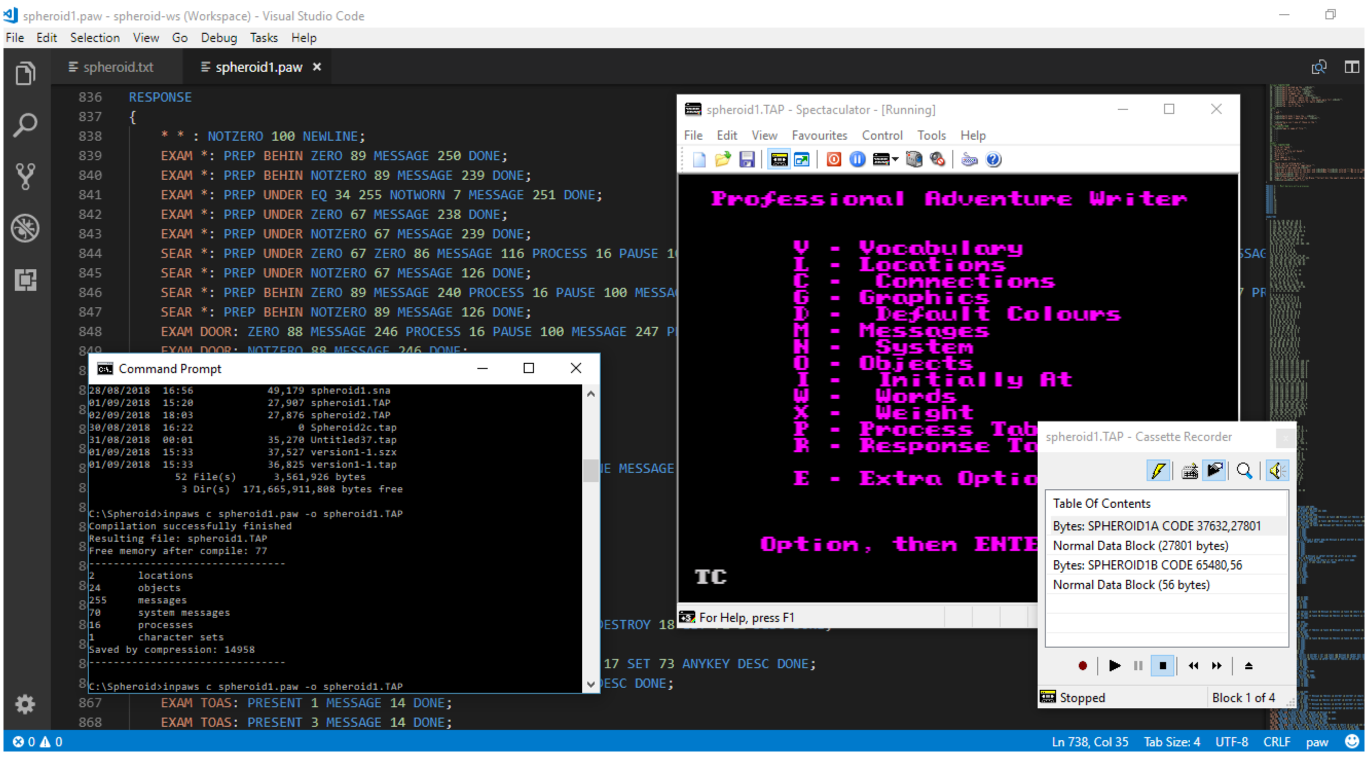
<u>Recommended Workflow</u>

I would recommend that you regularly test your code as you go along. With the right setup this can be extremely quick and easy to do.

Thanks to modern PCs, you can create a very efficient workflow which is almost as fast as using the original editor-based PAWS. In fact, the original PAWS editor is a key part of my setup.

To make things easier, all my code was kept in the same folder on my hard drive, together with a copy of the inPAWS program. Obviously, you need to make sure you create regular copies and backups of your key files.

I generally programmed with three windows open at once…


spheroid1.paw - spheroid-ws (Workspace) - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
spheroid.txt
spheroid1.paw
spheroid1.TAP - Spectaculator - [Running]
File Edit View Favourites Control Tools Help
For Help, press F1
spheroid1.TAP - Cassette Recorder
Stopped
Block 1 of 4
Ln 738, Col 35 Tab Size: 4 UTF-8 CRLF paw

```
RESPONSE
{
    * * : NOTZERO 100 NEWLINE;
    EXAM *: PREP BEHIN ZERO 89 MESSAGE 250 DONE;
    EXAM *: PREP BEHIN NOTZERO 89 MESSAGE 239 DONE;
    EXAM *: PREP UNDER EQ 34 255 NOTWORN 7 MESSAGE 251 DONE;
    EXAM *: PREP UNDER ZERO 67 MESSAGE 238 DONE;
    EXAM *: PREP UNDER NOTZERO 67 MESSAGE 239 DONE;
    SEAR *: PREP UNDER ZERO 67 ZERO 86 MESSAGE 116 PROCESS 16 PAUSE 1
    SEAR *: PREP UNDER NOTZERO 67 MESSAGE 126 DONE;
    SEAR *: PREP BEHIN ZERO 89 MESSAGE 240 PROCESS 16 PAUSE 100 MESSA
    SEAR *: PREP BEHIN NOTZERO 89 MESSAGE 126 DONE;
    EXAM DOOR: ZERO 88 MESSAGE 246 PROCESS 16 PAUSE 100 MESSAGE 247 P
    EXAM DOOR: NOTZERO 88 MESSAGE 246 DONE;
    EXAM TOAS: PRESENT 1 MESSAGE 14 DONE;
    EXAM TOAS: PRESENT 3 MESSAGE 14 DONE;
```

```
28/08/2018  16:56            49,179 spheroid1.sna
01/09/2018  15:20            27,907 spheroid1.TAP
02/09/2018  18:03            27,876 spheroid2.TAP
30/08/2018  16:22                 0 Spheroid2c.tap
31/08/2018  00:01            35,270 Untitled37.tap
01/09/2018  15:33            37,527 version1-1.szx
01/09/2018  15:33            36,825 version1-1.tap
              52 File(s)      3,561,926 bytes
               3 Dir(s)  171,665,911,808 bytes free

C:\Spheroid>inpaws c spheroid1.paw -o spheroid1.TAP
Compilation successfully finished
Resulting file: spheroid1.TAP
Free memory after compile: 77
--------------------------------
2       locations
24      objects
255     messages
70      system messages
16      processes
1       character sets
Saved by compression: 14958
--------------------------------

C:\Spheroid>inpaws c spheroid1.paw -o spheroid1.TAP
```

```
Professional Adventure Writer

V - Vocabulary
L - Locations
C -  Connections
G - Graphics
D -  Default Colours
M - Messages
N -  System
O - Objects
I -  Initially At
W -  Words
X -  Weight
P - Process Tab
R - Response To

E - Extra Optio

Option, then ENTE

TC
```

Table Of Contents
Bytes: SPHEROID1A CODE 37632,27801
Normal Data Block (27801 bytes)
Bytes: SPHEROID1B CODE 65480,56
Normal Data Block (56 bytes)

1. The VCS window, where I was writing the code.

2. The command prompt window, which I used whenever I needed to compile the code.

3. The emulator window (in my case Spectaculator) with a Spectrum version of PAWS loaded. I would recommend you use either the 128K or (my personal preference) the +3 version of PAWs. Simply because this means you don't need to worry about having to load in the overlays from tape (or disk) as the 48K memory gets full.

Note: If you're using the +3 version, you'll need to pop into the Y menu and change the input/output device to tape by typing T.

You can download an emulator image of PAWS from the authorised Gilsoft repository on Stefan Vogt's website… http://8-bit.info/the-gilsoft-adventure-systems/ If you don't own a physical copy of the system, then you should probably consider sending a donation to Tim Gilberts or consider purchasing his new version of PAWS once it has been released.

These were the steps in my workflow…

1. Type each bit of source code in Visual Code Studio and save the file.

2. Go to the command prompt window and type the command…
inpaws c spheroid1.paw -o spheroid1.TAP

(A useful tip is to use the UP cursor to bring up the last command you typed in)

This (**c**)ompiles my **spheroid1.paw** source code and (**o**)utputs it as **spheroid1.TAP** which is a ZX Spectrum emulator tape file.

3. Load the tape file into the emulator. Then type J (load) and load the database from that tape file into the PAWs.

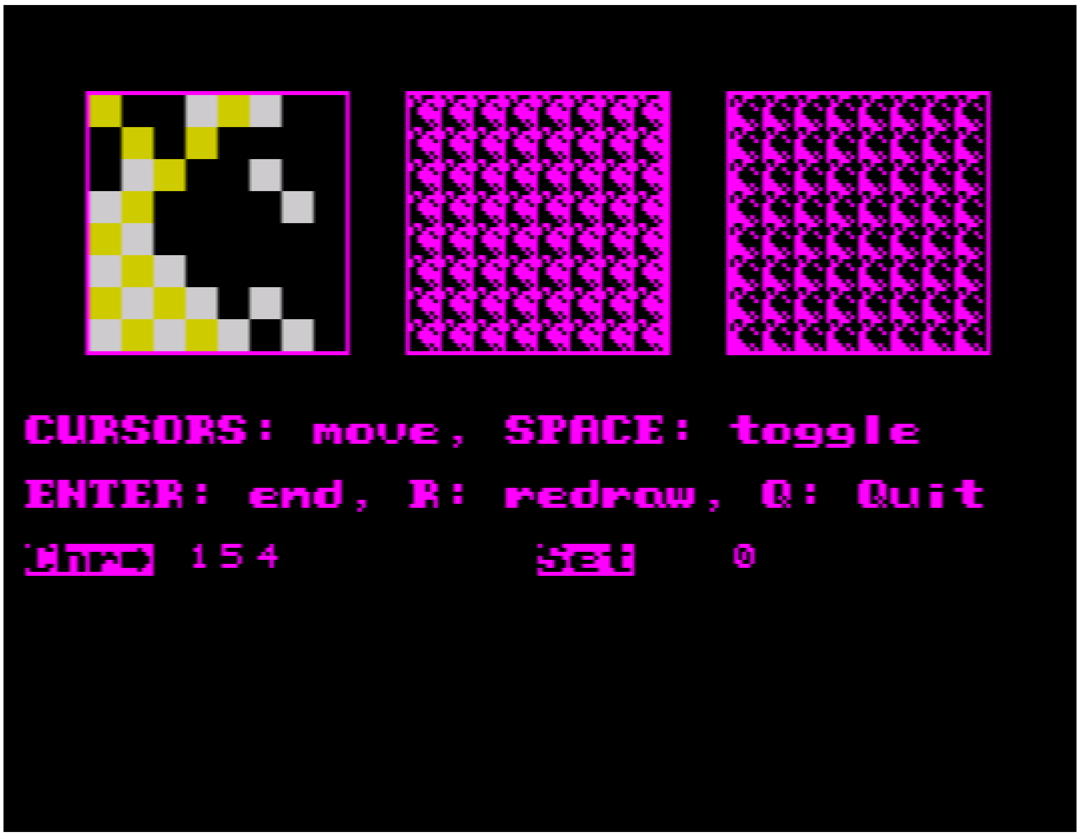
4. Type T for testing mode and you can try out your adventure.

I found this method to be extremely quick and efficient; allowing changes to be made to the code and quickly tested. It gives you the best of both worlds… the advantages of quick text entry through the compiler and the bonus of almost instant testing through the Spectrum editor.

Fonts and UDGs

The inPAWS documentation itself clearly explains how to implement a custom font, coloured text and UDGs in your Spectrum adventures.

To summarise that documentation, you use the inPAWS command line program to extract the fonts and any graphics from an emulator snapshot file. This produces a chunk of text that you can paste into your inPAWS source code file, meaning that whenever you compile your source code in the future your font and graphics are included.



```
CURSORS: move, SPACE: toggle
ENTER: end, R: redraw, Q: Quit
Chr. 154          Set    0
```

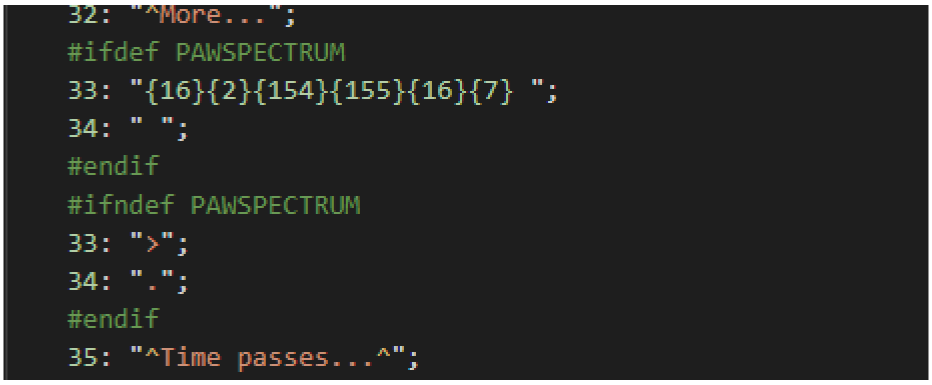
(The handy UDG & font editor built into the Spectrum version of the PAWS)

The easy way to create the characters section of your inPAWS file is to load your chosen font into the PAWS editor, making any changes required to the UDGs, and then save out the full adventure (option A) to a blank emulator tape file. Reset your emulator into 48K mode and load that adventure in to make a snapshot of it.

Use that snapshot as the file that you process using the inPAWS command line program.

Planning for an Amstrad CP/M version

One thing I haven't mentioned about the inPAWS source code yet is these green ***ifdef*** tags…



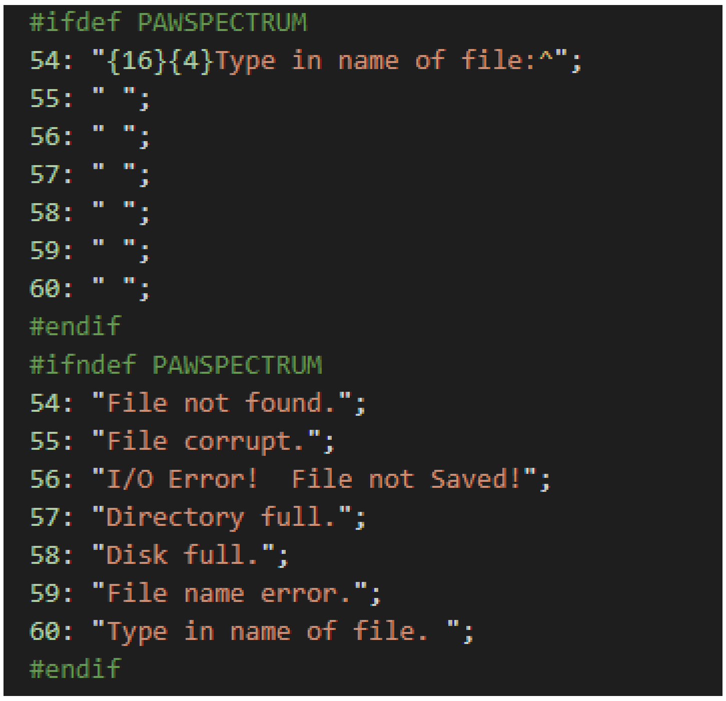
```
32: "^More...";
#ifdef PAWSPECTRUM
33: "{16}{2}{154}{155}{16}{7} ";
34: " ";
#endif
#ifndef PAWSPECTRUM
33: ">";
34: ".";
#endif
35: "^Time passes...^";
```

As well as producing adventures for the ZX Spectrum, you can also use inPAWS to make games for Amstrad CP/M (the disk-based CPCs and PCWs) and old school PC DOS.

PAWS on the Amstrad CP/M is different to the Spectrum version. It is compiler-based rather than editor-based. There are also subtle differences in the use of some of the flags and system messages.

If you want to use a common source to create a game for more than one platform then you use those green ifdef/endif tags to define parts that are specific to each system.

For example, the Amstrad version reserves some of the system messages for use as messages about saving and loading to disk. Here are those sections…



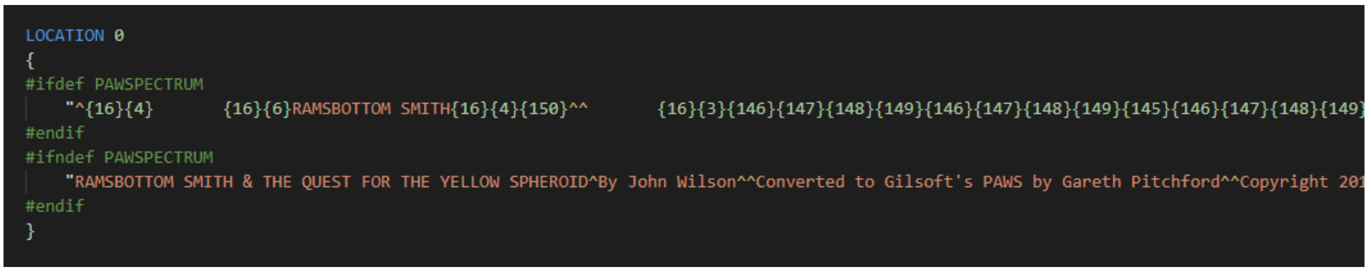
```
#ifdef PAWSPECTRUM
54: "{16}{4}Type in name of file:^";
55: " ";
56: " ";
57: " ";
58: " ";
59: " ";
60: " ";
#endif
#ifndef PAWSPECTRUM
54: "File not found.";
55: "File corrupt.";
56: "I/O Error!  File not Saved!";
57: "Directory full.";
58: "Disk full.";
59: "File name error.";
60: "Type in name of file. ";
#endif
```

In my code, I've not used the corresponding Spectrum versions of the messages *(#ifdef PAWSPECTRUM*) but I have to include them or else there will be an error on compiling (because you must use message numbers in order).

There are also layout differences between the Spectrum and Amstrad versions of PAWS. Such as the input line prompt. CP/M doesn't allow you to use your own graphics so you need to provide two versions of those messages. One for your Spectrum game and one for your Amstrad one.

Similarly, you can't use UDGs in the Amstrad version (the CP/M version uses those codes for text compression). This may mean that you may want to create an alternate version of your title screen for the Amstrad. Or a different 'anykey' prompt that doesn't use any graphical elements. Or you may wish to have two versions of certain messages that are formatted for different screen sizes. The Spectrum screen is 32 columns wide but the CP/M version has a massive 80 columns of text.

For example, here is location 0 (the intro screen) in Spheroid. The Spectrum version uses UDGs (user defined graphics) in the title screen… these are the bits with the codes {146}{147}{148} etc.



```
LOCATION 0
{
#ifdef PAWSPECTRUM
    "^{16}{4}       {16}{6}RAMSBOTTOM SMITH{16}{4}{150}^^       {16}{3}{146}{147}{148}{149}{146}{147}{148}{149}{145}{146}{147}{148}{149}
#endif
#ifndef PAWSPECTRUM
    "RAMSBOTTOM SMITH & THE QUEST FOR THE YELLOW SPHEROID^By John Wilson^^Converted to Gilsoft's PAWS by Gareth Pitchford^^Copyright 201
#endif
}
```

The non-Spectrum version (#ifndef PAWSPECTRUM) doesn't use any UDGs as Amstrad PAW won't accept them. You need to either eliminate or provide an alternate version of any parts of your inPAWS source that will cause issues like this if you want it to work on Amstrad too.

Producing an Amstrad CP/M version

My source file will compile both for Spectrum and CP/M. It's a little trickier to make the Amstrad version but here are the basic steps you'll need to take.

When you have a finished writing your inPAWS source file, used your inPAWS program to compile an Amstrad-compatible .SCE file using the command…

inpaws cm spheroid1.paw -o spheroid1.SCE

Note, I'm using the **cm** option this time.

You've now got a .SCE file, which is the raw file that gets fed into PAWS CP/M. It's very easy to corrupt this file. I know, because I managed to do it several times before Stefan Vogt came to my rescue and took me through a process that works.

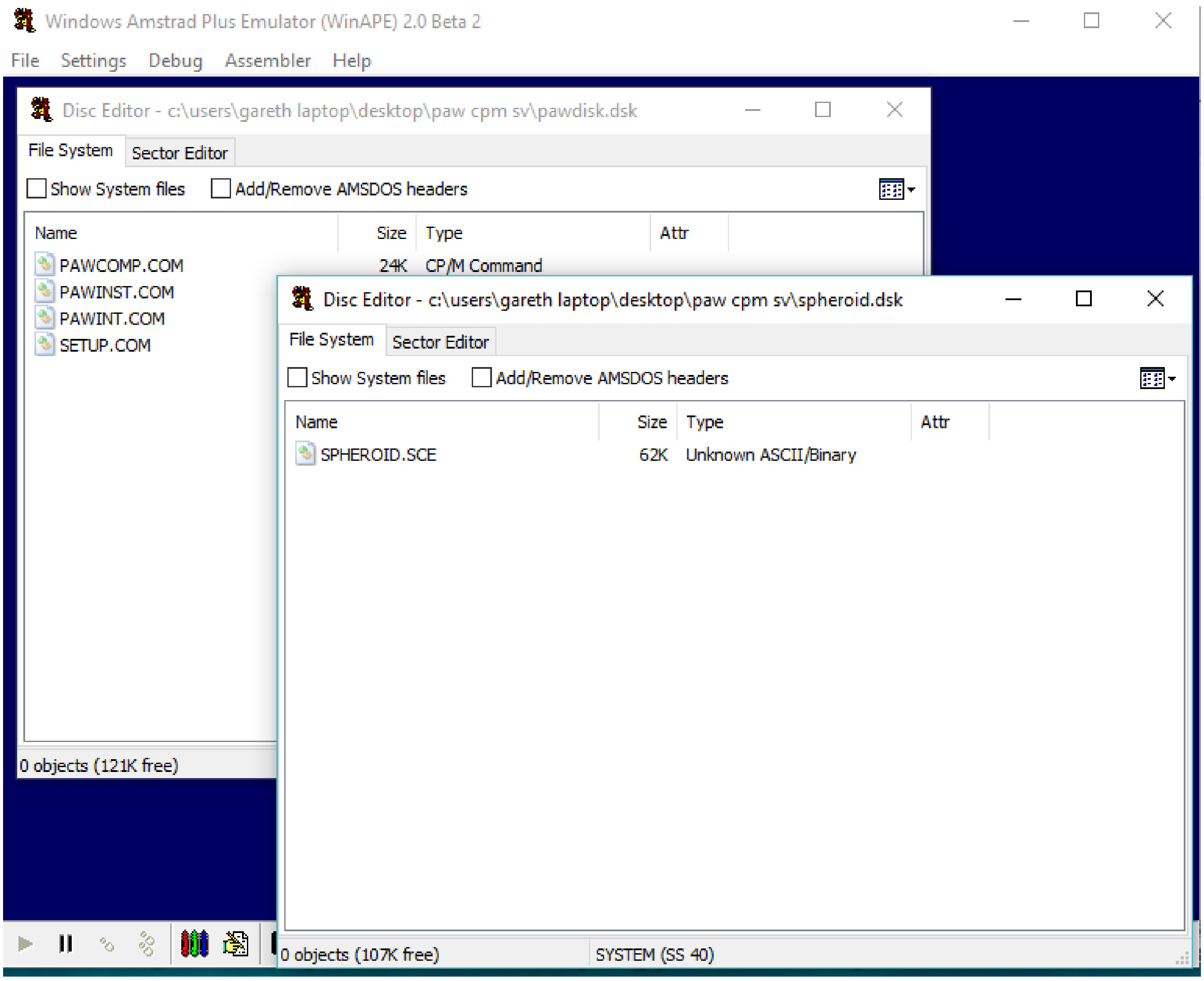
Here's a process that works…

I use the excellent WinApe Amstrad CPC emulator which can be downloaded from… http://www.winape.net/

You will need a copy of the PAWS CP/M disk image and you will probably need a second blank CP/M disk image (you can always ERAse files off an existing one) to have enough room to compile and save your adventure.

Select a suitable machine in your emulator (such as CPC128). Load your PAW disk in drive A and your blank disk in drive B using the file menu.

In WinAPE you can open up the disk drives in a window and drop and drag files onto them to edit on the fly. Drag your .SCE file from your desktop/or folder into the edit disk menu and a copy will appear on the disk image.


Windows Amstrad Plus Emulator (WinAPE) 2.0 Beta 2
File
Settings
Debug
Assembler
Help
File System
Sector Editor
Show System files
Add/Remove AMSDOS headers

Disc Editor - c:\users\gareth laptop\desktop\paw cpm sv\pawdisk.dsk

Name	Size	Type	Attr
PAWCOMP.COM	24K	CP/M Command	
PAWINST.COM			
PAWINT.COM			
SETUP.COM			

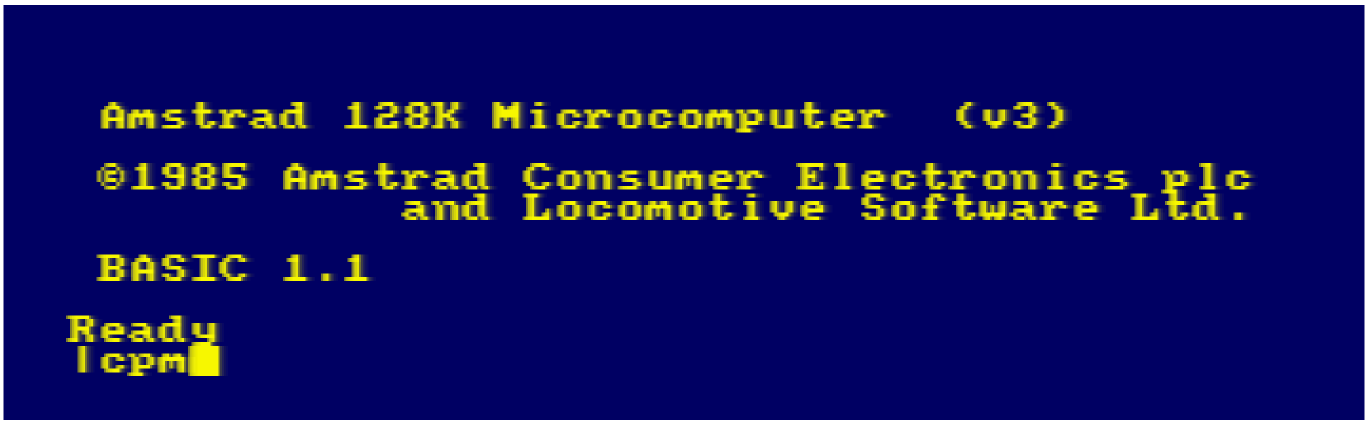
0 objects (121K free)

Disc Editor - c:\users\gareth laptop\desktop\paw cpm sv\spheroid.dsk

Name	Size	Type	Attr
SPHEROID.SCE	62K	Unknown ASCII/Binary	

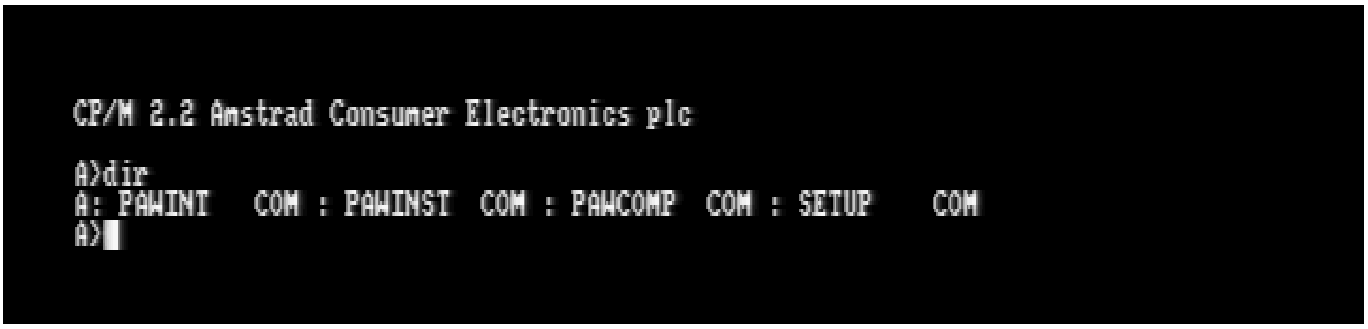
0 objects (107K free) SYSTEM (SS 40)

Launch CP/M off the disk by typing |CPM. (To get the | on my emulator I need to hold down SHIFT and the [key that's next to P)



```
 Amstrad 128K Microcomputer  (v3)

©1985 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.

BASIC 1.1

Ready
|cpm
```

The two key programs on the PAWS CP/M disk are PAWCOMP and PAWINT.

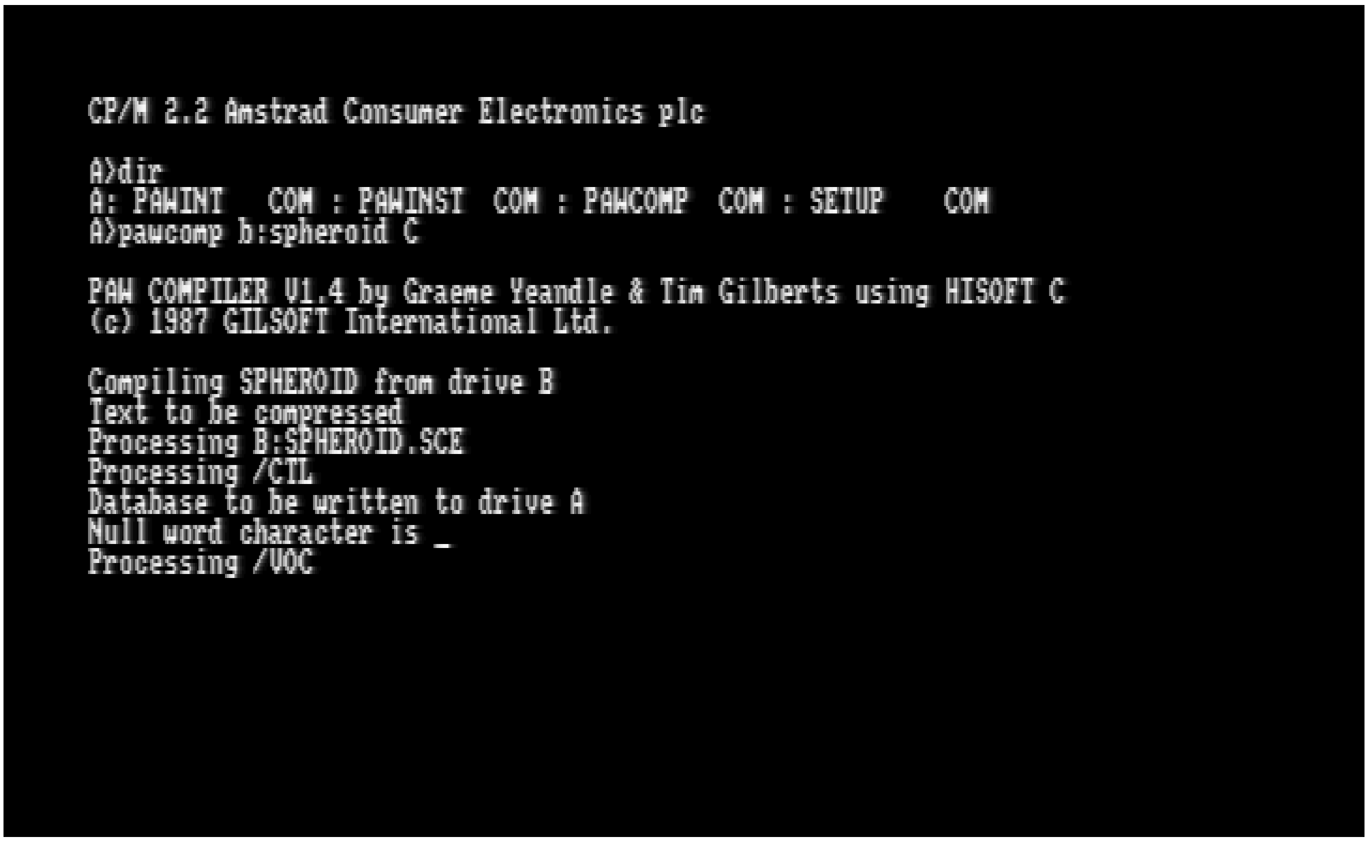


```
CP/M 2.2 Amstrad Consumer Electronics plc

A>dir
A: PAWINT   COM : PAWINST  COM : PAWCOMP  COM : SETUP    COM
A>
```

You feed your .SCE into PAWCOMP to create a PAW database file. (.PDB)

e.g. **PAWCOMP b:spheroid C**

(The b: because my file is on drive B and the C as I need to compress the text otherwise that particular adventure is too large to run)



```
CP/M 2.2 Amstrad Consumer Electronics plc

A>dir
A: PAWINT   COM : PAWINST  COM : PAWCOMP  COM : SETUP    COM
A>pawcomp b:spheroid C

PAW COMPILER V1.4 by Graeme Yeandle & Tim Gilberts using HISOFT C
(c) 1987 GILSOFT International Ltd.

Compiling SPHEROID from drive B
Text to be compressed
Processing B:SPHEROID.SCE
Processing /CTL
Database to be written to drive A
Null word character is _
Processing /VOC
```

Then you run the database you've created using PAWINT

e.g. **PAWINT b:spheroid C**

You'll get an option to play the game or save it out as a standalone file. You may need to do some disk juggling or insert a new disk in order to have room to save.

Once you've made your Amstrad CP/M standalone file, that file can be transferred to other CP/M systems. One of the advantages of CP/M is it's very portable across Z80 machines.

Thanks…

I've been inspired to use inPAWS by the work of Stefan Vogt, who must surely now have some sort of record for the amount of times he's ported his own game to different (or the same!) platforms, namely his sci-fi adventure Hibernated 1. Starting with a Quilled version on the C64, he went on to use inPAWS to create PAWed versions for the ZX Spectrum, PC DOS and CP/M (releasing that for both Amstrad and Commodore 128) before switching to the DAAD system and publishing for C64, Amstrad CPC, Spectrum, PC DOS, Atari ST and Amiga. My thanks go to him for sharing his experiences and extensive technical knowledge. You can find out more about his Hibernated 1 game here… https://8bitgames.itch.io/hibernated1

Thanks also goes to Chris Ainsley for the inPAWS syntax highlighter and his Adventuron system. Tim Gilberts & Gilsoft for the PAWS, Francisco Javier López for inPAWS, and Mark Hardisty for his guide that helped me import the familiar Zenobi 'rustic' font.

And, of course, many thanks to John Wilson for letting me play about with his game and help bring it home to the ZX Spectrum.