

A guide to converting a ZX Spectrum 48K PAWS adventures to Amstrad CP/M using inPAWS

By Gareth Pitchford, 8bitAG.com

Version 1.4 (Updated: November 2018)

This guide will outline how to convert a ZX Spectrum 48K text adventure, produced using Gilsoft's Professional Adventure Writing System, into a CP/M version that can be played on disk-based Amstrad CPC and PCW machines. The CP/M version can also be used on a whole host of other Z80 machines and the inPAWS source can also be compiled into a version for PC DOS.

This document is intended for use by Spectrum text adventure authors to help them convert their own games to the Amstrad machines. It's not intended to help third parties take existing Spectrum games and flood the Amstrad with quickly done ports, without the permission of the original authors... I don't think anybody really wants that!

This guide

Will use the #ifdef markup in inPAWS to show how you can maintain a source file that will compile to both the Spectrum and Amstrad/PC DOS.

Will look at some of the differences between the Spectrum and Amstrad PAWs, addressing issues with system messages, UDGs, and screen & layout problems.

Software Used:

inPAWs by Francisco Javier López, <http://inpaws.speccy.org/indexEng.html>

Microsoft's Visual Code Studio <https://code.visualstudio.com/>

- with Chris Ainsley's .paw syntax highlighter installed, <https://marketplace.visualstudio.com/items?itemName=ainslec.inpaws>

The Professional Adventure Writer (CP/M version) by Gilsoft

- from Stefan Vogt's Gilsoft Repository at <http://8-bit.info/the-gilsoft-adventure-systems/>

WinAPE or other **Amstrad CPC** emulator

- <http://www.winape.net/> (currently being flagged up as a site with issues by Google?)
- Alternative .SCE transfer method - iDSK, see <http://www.cpcwiki.eu/index.php/IDSK> & then use another emulator

Extracting your Spectrum PAWs database and converting it to an inPAWS file

Pop the inpaws.exe file in a folder on your computer, along with the .SNA or .Z80 snapshot of your 48K ZX Spectrum adventure.

From a command prompt navigate to the folder where you've placed your files and execute the inpaws program using the command *inpaws e mygame.Z80 -o mygame.paw*

For example...

```
C:\nether>inpaws e PCW.Z80 -o pcw.paw
PAW Database successfully extracted to file: pcw.paw
-----
45     locations
28     objects
212    messages
76     system messages
13     processes
2      character sets
Database version: 17
-----
C:\nether>
```

This will generate a **.paw** file that you can edit to create a new source that can be used to build the Amstrad version of your adventure.

The Structure of the inPAWS (.paw) file

Load your .paw file into Visual Studio Code.

If you have Chris Ainsley's inPAWS syntax highlighter installed your code will be colour-coded to make it easier to read. Note that each line is numbered. inPAWs will refer to these line numbers should it find an error in your code, making bugs and typos really easy to fix. inPAWs has a great error report system.

The structure of the source code may initially seem confusing but at its core it's really only the Spectrum PAWs database presented in a slightly different format.

Let's look at the main sections of the .paw file...

```
1  DEFAULTS
2  {
3      CHARSET: 1;
4      INK: 6;
5      PAPER: 0;
6      FLASH: 0;
7      BRIGHT: 0;
8      INVERSE: 0;
9      OVER: 0;
10     BORDER: 0;
11 }
```


The **vocabulary** section should be easy to understand.

```
OBJECT 0
{
  "{19}{1}{16}{7}An object that doesn't exist!";
  INITIALLYAT NOTCREATED;
  WORDS _ _;
  WEIGHT 1;
}

OBJECT 1
{
  "{16}{7}{19}{1}A signed photograph";
  INITIALLYAT NOTCREATED;
  WORDS PHOTO _;
  WEIGHT 1;
}

OBJECT 2
{
  "{19}{1}{16}{7}A microfair ticket";
  INITIALLYAT NOTCREATED;
  WORDS TICKE _;
  WEIGHT 1;
}
```

The **objects** section includes both the name of the object and the associated vocabulary word(s), and optionally other information such as the object's weight and its initial status.

Here is an additional example that shows an item that can be worn...

```
OBJECT 15
{
  "{3}{16}{4}A coat of arms";
  INITIALLYAT NOTCREATED;
  WORDS COAT _;
  WEIGHT 1;
  PROPERTY CLOTHING;
}
```

An item that starts in the player's inventory...

```
OBJECT 27
{
  "{16}{4}Some money";
  INITIALLYAT CARRIED;
  WORDS MONEY _;
  WEIGHT 1;
}
```

And here is an item that starts at a specific location...

```
OBJECT 28
{
  "{5}{16}{4}A note.";
  INITIALLYAT 49;
  WORDS NOTE _;
  WEIGHT 1;
}
```

In addition, INITIALLYAT WORN; can be used for objects that should be worn at the start of the game (with the associated PROPERTY CLOTHING).

PROPERTY CONTAINER; sets the object to be a container.

The **messages** section follows after the objects. Once again, each message is enclosed in a pair of speech marks and followed by a semi-colon. If speech marks are required in the actual message they're inserted using \" rather than \".

```
MESSAGES
{
  0: "{20}{0}{20}{1}                                MICROFAIR MADNE
  1: "{19}{1}It's a signed photo of {4}{19}{0}{16}{3}Garth Pitchfork
  2: "{19}{1}It's marked {16}{3}{19}{0}'MICROFAIR 100 TICKET'{16}{6}
  3: "{19}{1}The leaflet advertises the next {16}{3}{19}{0}'Jean-Mic
  4: "{19}{1}A wonderful thing is the pin,^So many things to stick i
  5: "{19}{1}It's just your average five pound note.";
  6: "{19}{1}It's a cuddly toy! Actually, it's meant to look like a
  7: "{19}{1}It's a copy of that wondrous magazine {16}{3}{19}{0}'Ad
  8: "{19}{1}It's just your average laser-gun.";
  9: "{19}{1}It's a small sachet of powder marked {16}{3}{19}{0}'Orc
  10: "{19}{1}It's a copy of {16}{3}{19}{0}'Arnold the Adventurer VI
  11: "{19}{1}Just some seed.";
  12: "{19}{1}The two pound coin is a thing that people think is rar
  13: "{19}{1}It's a combined gardening/adventuring beard.";
  14: "{19}{1}It's a copy of {16}{3}{19}{0}'Wombat'{16}{7}{16}{6}{19
  15: "{19}{1}Just a normal cup!";
  16: "{19}{1}It's black, well that's what the pot says anyway!.^{19
```

Time to address the numbers inside the curly brackets {}.

These numbers are how inPAWS represents the control codes that you use in Spectrum PAWs adventures to select the text colour, the paper colour, brightness, the font to be used and other elements such as the User Designed Graphics (UDGs).

In the Spectrum version of the PAWs, you'd usually insert these through a combination of fairly complicated keypresses. Using the {} is a lot easier and less frustrating, especially these days when many of us have forgotten which buttons activate 'extended mode'.

A control code on its own, with a number between 0 and 5 in, e.g. {1} changes the font.

The control code {7} forces a line break but the symbol ^ is preferred (and used by default) by inPAWS.

Some control codes work in pairs. For example {16}{5} or {19}{1}.

{16} denotes the ink colour. So {16}{3} would print the text that follows it in Magenta. {16}{7} in white.

{17} selects the background colour. So {17}{4} would change the background, of text following the codes, to green.

{19} selects the brightness. {19}{0} is the default and {19}{1} is bright. {18} and {20} work in a similar fashion for flashing text and inverse video.

When you import an existing Spectrum game, as in the example above, you will sometimes see redundant control codes... It was such a faff to enter these on the Spectrum (particularly on a non-rubber-keyed model) that it was easy to include unseen extra codes!

Should you wish to use the {}, ^ or " symbols in your text as normal characters then place a \ before them. E.g. \} or \^.

The other numbers you may see inside {} brackets, such as {145} or {150}, refer either to special characters (such as the copyright symbol) or to **user defined graphics**. These are the extra characters a user of the Spectrum version of PAWs can create themselves, using the inbuilt character editor. Often UDGs are used as part of the input prompt or as a graphical "press any key". I'll address UDGs more fully in the conversion section later on, as we'll need to remove them in the Amstrad version.


```
CHARACTERS
{
    240, 150, 150, 240, 15, 105, 105, 15,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    8, 24, 56, 127, 127, 56, 24, 8,
    16, 24, 28, 254, 254, 28, 24, 16,
    0, 0, 0, 0, 0, 0, 0, 250,
    255, 254, 252, 248, 240, 224, 192, 128,
    0, 0, 36, 219, 219, 36, 0, 0,
    1, 3, 7, 15, 31, 63, 127, 255,
    128, 192, 224, 240, 248, 252, 254, 255,
    0, 0, 0, 15, 8, 8, 8, 8,
    0, 0, 0, 248, 8, 8, 8, 8,
    8, 8, 8, 15, 0, 0, 0, 0,
```

As is information on the **character sets** used.

The next section needs a little more explanation...

```
1251 PROCESS 3;
1252 PROCESS 4;
1253 PROCESS 5;
1254 PROCESS 6;
1255 PROCESS 7;
1256 PROCESS 8;
1257 PROCESS 9;
1258 PROCESS 10;
1259 PROCESS 11;
1260 PROCESS 12;
1261 PROCESS 13;
1262
```

At this point in the source code you need to tell inPAWS which additional process tables you have used. All the processes used in the original imported database will be automatically “called out” here but if you add any more, during your editing, then you must mention them in this section.

In the example above, a 14th process table would require the addition of **PROCESS 14;** to line 1262 in the code.

RESPONSE

```
{
 * *: AT 0 PROCESS 13 DONE;
 * *: ATGT 39 PROCESS 7 DONE;
 S *: AT 8 NOTZERO 64 MESSAGE 43 DONE;
 W *: AT 11 ZERO 87 MESSAGE 115 DONE;
 NO *: AT 2 NOTCARR 2 MESSAGE 22 DONE;
 NO *: AT 2 CARRIED 2 MESSAGE 23 LET 38 4 ANYKEY DESC DONE;
 NO *: AT 37 ZERO 70 MESSAGE 55 DONE;
 NO *: AT 20 ZERO 92 MESSAGE 121 DONE;
 ASCEN *: AT 32 NOTZERO 72 MESSAGE 66 LET 38 33 ANYKEY DESC DONE;
 ASCEN *: AT 33 MESSAGE 66 LET 38 32 ANYKEY DESC DONE;
 ASCEN *: AT 37 MESSAGE 199 DONE;
 I _ : INVEN;
 SHOW PHOTO: AT 37 ZERO 70 CARRIED 1 MESSAGE 57 ANYKEY MESSAGE 58 ANYKEY
 SHOW LEAFL: AT 3 CARRIED 3 ZERO 61 MESSAGE 19 ANYKEY MESSAGE 20 CREATE
```

Now we get on to the **response table** or Process 0 (as Amstrad PAWs refers to it). This works exactly as it does in PAWs, the only difference is that each entry is presented horizontally and terminated with a semi-colon. The syntax highlighter's colour coding really helps you parse the code here and is very useful for spotting typing errors should you end up editing or adding to the code.

PROCESS 1

```
{
 * *: NOTZERO 76 COPYFF 38 75;
 * *: ZERO 250 AT 0 ANYKEY MODE 4 1 ABILITY 255 255 CLS PROMPT 75 SET 250 DONE;
 * *: AT 3 ZERO 61 SYSMESS 58;
 * *: AT 7 SYSMESS 60;
 * *: AT 8 NOTZERO 64 SYSMESS 61;
 * *: AT 29 ZERO 68 NOTZERO 67 SYSMESS 62;
 * *: AT 32 ZERO 72 SYSMESS 63;
 * *: AT 32 NOTZERO 72 SYSMESS 64;
 * *: SAME 38 75 SYSMESS 65;
 * *: ZERO 77 AT 26 ANYKEY CLS MESSAGE 79 ANYKEY MESSAGE 80 ANYKEY MESSAGE 81 ANYKEY
 * *: ZERO 77 AT 26 ZERO 76 MESSAGE 84 ANYKEY PROCESS 3;
 * *: ZERO 77 AT 26 NOTZERO 76 MESSAGE 85 ANYKEY MESSAGE 86 ANYKEY MESSAGE 87 ANYKEY
```

Process 1, the process that is mainly used to add extra information to the location text, follows the response table.

```

PROCESS 2
{
  * *: NOTZERO 102 CLS PRINTAT 8 0 MESSAGE 182 ANYKEY CLEAR 102 LET
  * *: AT 17 NOTZERO 90 MINUS 89 1;
  * *: AT 17 ZERO 90 LET 89 3 SET 90;
  * *: AT 0 MODE 4 1 ABILITY 255 255;
  * *: NOTZERO 84 NOTZERO 85 SET 86 CLEAR 84 CLEAR 85 PLUS 30 20;
  * *: ZERO 89 AT 17 ZERO 88 PROCESS 4 CLS PROCESS 3 DONE;
  * *: NOTZERO 93 ZERO 94 MESSAGE 144;
  * *: ATGT 39 NOTZERO 93 ZERO 94 MINUS 96 1;
  * *:
}

```

Then it's **process 2**, which is considered the computer's turn of the game.

```

PROCESS 3
{
  * *: MESSAGE 88;
  * *: NEWLINE TURNS END DONE;
  * *: CLS;
}

PROCESS 4
{
  * *: PROCESS 6 INK 7 PROCESS 5 INK 7 PROCESS 6 INK 4 PROCESS 5 INK 4 P
}

PROCESS 5
{
  * *: BEEP 10 150 PICTURE 0 BEEP 10 200 CLS BEEP 10 100 MESSAGE 120 PIC
}

```

After that the source code continues detailing any additional process tables you may have used in your Spectrum adventure. The last process table forms the end of the source code.

I hope that you're now a bit more familiar with what the inPAWS code looks like and can identify the sections in your own adventure. Now it's time to edit the code so it will work in the Amstrad CP/M version of the PAW.

Converting Your Spectrum PAWs code to Amstrad CP/M PAWs

Now technically, if the Spectrum adventure that you're converting is very simple and doesn't do anything that the Amstrad PAWs doesn't like, you could attempt to compile your .paw file into an Amstrad .SCE source file at this stage.

```
C:\nether>inpaws cm pcw.paw -o pcw.SCE
Compilation successfully finished
Resulting file: pcw.SCE
-----
45      locations
28      objects
212     messages
76      system messages
13      processes
-----
```

However, when you go to compile it into a database on the Amstrad, it's highly likely you'll get a whole host of errors. Like this...

```
1224 .....
*** ERROR 53 - Character value > 127 present in text - value(144)
1224 .....
*** ERROR 53 - Character value > 127 present in text - value(144)
1224 .....
*** ERROR 53 - Character value > 127 present in text - value(144)
1224 .....
*** ERROR 53 - Character value > 127 present in text - value(144)
1224 .....
*** ERROR 53 - Character value > 127 present in text - value(144)
1224 .....
*** ERROR 53 - Character value > 127 present in text - value(144)
1224 .....
*** ERROR 53 - Character value > 127 present in text - value(144)
45 texts processed
*** Compilation ends with 230 ERRORS
A>|
```

230 errors! Yikes!

So, let's look at the steps you'll need to take to adapt it...

Using the #ifdef tag to amend the code

If you want to tailor your .paw file to exclusively create an Amstrad version then you can just make direct edits to it. However if, like me, you want to keep the option of using your new .paw file as a source for both an Amstrad adventure and a revised Spectrum version of your game, then you should use the #ifdef and #ifndef tags.

These tags allow us to provide two versions of code in the source file. Code surrounded by **#ifdef PAWSPECTRUM** and **#endif** will be included in the Spectrum version. Code highlighted by **#ifndef PAWSPECTRUM** and **#endif** (note the n in #ifndef) will be used to generate the Amstrad (and PC DOS) editions of the game.

There will be plenty of examples of this code as we work through adapting our game file.

Step 1: Address any issues with the System Messages

The Amstrad version of the PAWs uses system message numbers 54 to 60 for file operations. Despite a warning in the Spectrum PAWs manual about this, most Spectrum authors won't have kept those messages free.

So, our first step is to move the existing conflicting system messages to higher numbers and insert the versions that the Amstrad needs for the disk operations.

```
54: "{19}{1}Type in name of file:^";
55: "";
56: "";
57: "{19}{1}You've already done that!^";
58: "{19}{1} {4}{16}{5}Oh my goodness! - {16}{3}{19}{0}Garth Pitchfork {4}{16}{5}{19}{1}is here!";
59: "{19}{1}How?^";
60: "{16}{3} {19}{0}{19}{1}{19}{0}Mangy Rodrigues {4}{5}{4}{16}{5}{19}{1}is here - She seems to want s
```

Several of the Amstrad's system messages have been used in this example, so my first step is to copy this block of code and paste it to the end of the existing system messages.

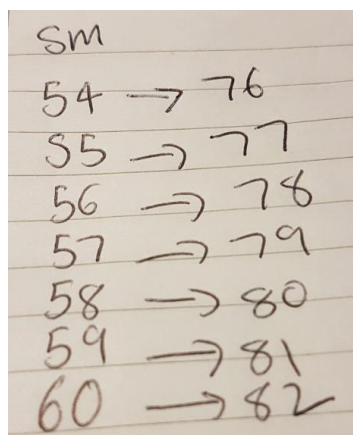
```
70: "You drop the _^";
71: "You pick up the _^";
72: " The window is open.";
73: "{16}{4}{19}{1}POSITION SAVED TO RAM...";
74: "{16}{4}{19}{1}POSITION BEING LOADED FROM RAM..";
75: "^{16}{5}Introduction?^";
54: "{19}{1}Type in name of file:^";
55: "";
56: "";
57: "{19}{1}You've already done that!^";
58: "{19}{1} {4}{16}{5}Oh my goodness! - {16}{3}{19}{0}Garth Pitchfork {4}{16}{5}{19}{1}is here!";
59: "{19}{1}How?^";
60: "{16}{3} {19}{0}{19}{1}{19}{0}Mangy Rodrigues {4}{5}{4}{16}{5}{19}{1}is here - She se
```

Now, let's change the system message numbers so they follow on from the last of the original system messages.

```
73: "{16}{4}{19}{1}POSITION SAVED TO RAM...";
74: "{16}{4}{19}{1}POSITION BEING LOADED FROM RAM..";
75: "^{16}{5}Introduction?^";
76: "{19}{1}Type in name of file:^";
77: "";
78: "";
79: "{19}{1}You've already done that!^";
80: "{19}{1} {4}{16}{5}Oh my goodness! - {16}{3}{19}{0}Garth Pitchfork {4}{16}{5}{19}{1}is here!";
81: "{19}{1}How?^";
82: "{16}{3} {19}{0}{19}{1}{19}{0}Mangy Rodrigues {4}{5}{4}{16}{5}{19}{1}is here - Sh
```

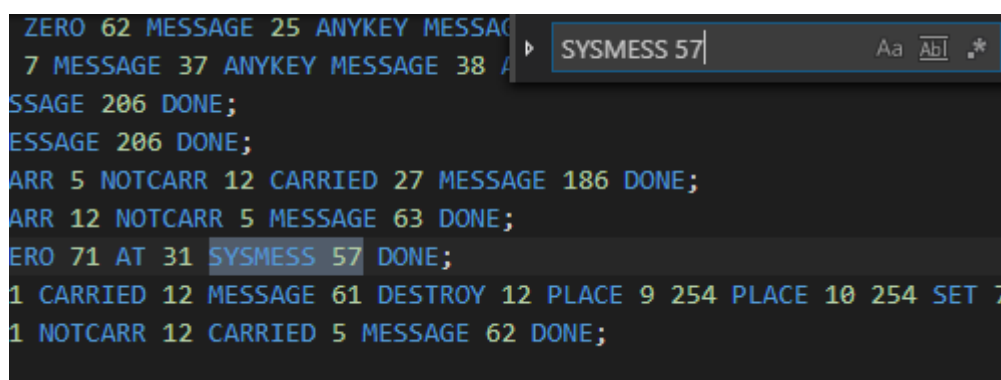
Technically I could've omitted message 76, 77 and 78 in this example (especially as messages 77 and 78 weren't used) but for clarity I'm just going to change those too.

It's worth keeping a note of the changes in message number for this next step.

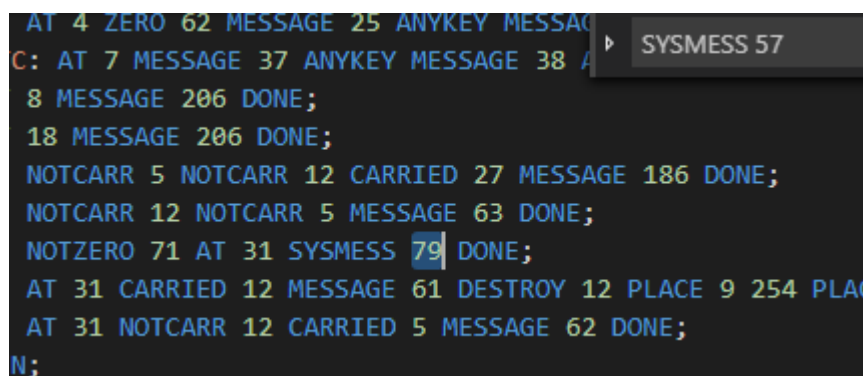


SM
54 → 76
55 → 77
56 → 78
57 → 79
58 → 80
59 → 81
60 → 82

Right, now use the find feature of the editor to identify places where these system messages were used and amend the numbers accordingly.



```
ZERO 62 MESSAGE 25 ANYKEY MESSAGE 37  
7 MESSAGE 37 ANYKEY MESSAGE 38 /  
SSAGE 206 DONE;  
ESSAGE 206 DONE;  
ARR 5 NOTCARR 12 CARRIED 27 MESSAGE 186 DONE;  
ARR 12 NOTCARR 5 MESSAGE 63 DONE;  
ERO 71 AT 31 SYSMESS 57 DONE;  
1 CARRIED 12 MESSAGE 61 DESTROY 12 PLACE 9 254 PLACE 10 254 SET 7  
1 NOTCARR 12 CARRIED 5 MESSAGE 62 DONE;
```



```
AT 4 ZERO 62 MESSAGE 25 ANYKEY MESSAGE 37  
C: AT 7 MESSAGE 37 ANYKEY MESSAGE 38 /  
8 MESSAGE 206 DONE;  
18 MESSAGE 206 DONE;  
NOTCARR 5 NOTCARR 12 CARRIED 27 MESSAGE 186 DONE;  
NOTCARR 12 NOTCARR 5 MESSAGE 63 DONE;  
NOTZERO 71 AT 31 SYSMESS 79 DONE;  
AT 31 CARRIED 12 MESSAGE 61 DESTROY 12 PLACE 9 254 PLAC  
AT 31 NOTCARR 12 CARRIED 5 MESSAGE 62 DONE;  
N;
```

Until all the relevant system message references (where they exist) have been changed.

Now, we need to go back to system messages 54 to 60 and replace this section with the following code...

(You can grab the Amstrad part of this code from the *new-eng.paw* file that's in the demo folder of the inPAWs download)

```
#ifdef PAWSPECTRUM
54: "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
```

The first block of code defines the new system messages 54 to 60 for the Spectrum version. The second block adds the relevant messages for the Amstrad version.

Our system messages should now be sorted although it's highly likely that we'll return to these blocks of code to look at them again when we move on to address the UDGs.

Step 2: Removing the UDG codes

The Amstrad version of the PAWs doesn't use colours, fonts or UDGs. We can safely forget about the colour & font codes, as they'll be ignored when we compile our adventure, but we need to address the UDGs.

Any embedded codes referring to UDGs must be removed as they'll throw up errors when our adventure is compiled on the Amstrad (as we saw when we tried to compile the game at the beginning of this section).

This is one of the most time-consuming parts of the conversion, but the find tool again comes to our rescue.

In the example game I'm working on converting, UDGs are used in various places. Such as on the title page of the adventure...



tested the game and tidy it up so it looks nice on the Amstrad screen, but for now let's move on...

Nothing else in my location text, but some of my messages use UDGs, such as this GAME OVER message so I'll need to either edit that or provide an Amstrad alternative...

```
86: "{19}{1}{16}{5}Smok collapses and you nick his laser-gun!^";
87: "{19}{1}{16}{5}The tramp tells you that he's going to take the 'point-eared prat down the nic
88: "^{6}^{16}{7}{19}{1}{16}{5}{19}{1} {144}{144}{144}{144}{144}{144}{144}{144}{144}^{1
89: "{19}{1}{4}{16}{5}It was Friday the 13th of November, and the day of yet another Microfair. T
90: "{19}{1}{19}{1}{16}{5}You were looking{19}{1} foward to this one. There would be no journeys
91: "{19}{1}{16}{5}Of course, you were going to deliver your new game to DTHS. But there would be
92: "{19}{1}{16}{5}And so you found yourself standing in front of Sir Clive's exhibition complex
```

Once again, I'll just do a quick game over message for now and tidy it up to look better later...

```
87: "{19}{1}{16}{5}The tramp tells you that he's going to take the point-eared prat down the
#ifdef PAWSPECTRUM
88: "^{6}^{16}{7}{19}{1}{16}{5}{19}{1} {144}{144}{144}{144}{144}{144}{144}{144}^{1
#endif
#ifndef PAWSPECTRUM
88: "^^GAME OVER^^You are dead and thus, by default, have failed to complete your quest.^";
#endif
89: "{19}{1}{16}{5}It was Friday the 13th of November, and the day of yet another Microfair
```

Another message (162) needed editing, but I've decided to change it in both versions...

```
160: "{19}{1}{16}{5}Get ready! ", says the man. ";
161: "This adventure accepts all the standard abbreviations and commands as well as...^{4}{16}{5}
162: "Really stuck? Check out SolutionArchive.com"; // changed original SAE reference
163: "A normal gun.";
164: "The box is quite small with a keypad, with numbers, on its lid.";
```

You can use a double slash // to add a comment to your code. Well worth doing if you're changing things!

In my adventure there were quite a few more messages to edit before I moved on to look at the system messages.

```
14: "^^{4}{4}{4}{16}{3}{1}{19}{1}{127}1992 Electric Storm Productions^^";
15: "OK.^";
16: "^^ {18}{1}{5}{3}{18}{0}{18}{1}{5}{3}{16}{4}{16}{6}{19}{1}{144}{144}{18}
17: "{19}{1}You have moved ";
18: "{19}{1} time";
19: "{19}{1}s";
20: "^^";
```

System message 16 is the "press any key" prompt. I'm going to add an Amstrad-specific version of that, not using UDGs, remembering that the Amstrad screen is 80 columns wide...

```
16: "^^ {18}{1}{5}{3}{18}{0}{18}{1}{5}{3}{16}{4}{16}{6}{19}{1}{144}{144}{18}
#ifdef PAWSPECTRUM
16: "^^ (+)^^";
#endif
```

I can play about with the line spacing and how that looks later if I want to.

*** DOG: AT 23 ZERO 64 MESSAGE 56;**

...to print an additional message at location 23 when the dog (indicated by flag 64) was there. The DOG part of the entry is ignored by Spectrum PAWs. It's just there so the programmer can remember that that is the line that prints the message about the dog.

Although this was a clever little trick it has unintended consequences in CP/M PAWs. The CP/M version of the PAWs interprets Process 1 and 2 differently to the Spectrum version. In CP/M PAWs the NOUN & VERB entries are not ignored. They are not just skipped over. CP/M will attempt to match NOUNS & VERBS in all process tables, not just those related to the response table (process 0).

This is a section of Process 1 from the Jack Lockerby game, the Islands of Sinbad.

```
* *: AT 7 ISAT 1 10 MESSAGE 14 PROTECT;|
* *: AT 7 ISAT 2 10 MESSAGE 34 PROTECT;
* *: AT 12 MESSAGE 22;
* *: AT 0 MODE 4 0 LET 53 64 INPUT 0 ANYKEY CLS MESSAGE 35 ANYKEY GOTO 23 ABILITY 7 30 DESC;
* *: COPYFF 39 100 LISTOBJ PROCESS 11 NEWLINE;
* SEA: ATGT 44 ATLT 49 SYSMESS 106 NEWLINE;
* SEA: ATGT 20 ATLT 27 NOTAT 22 SYSMESS 106 NEWLINE;
* MONKE: AT 30 ABSENT 19 EQ 128 255 CREATE 19 PLACE 19 255 MESSAGE 118 ANYKEY MINUS 128 200 DESC;
_ _ : PROCESS 10;
```

Jack has used SEA and MONKE in the process table as a reminder that those lines print a message about the SEA and a message about a MONKEY being present. In the Spectrum version of the game this is fine the entries are processed as normal but in the Amstrad conversion, the lines with * SEA and * MONKE on are skipped over! This means that vital parts of the game never trigger.

You will need change any such references, while being mindful about the order that the table will ultimately be sorted in PAWs.

In this case, we can simply replace the SEA/MONKEY with * or indeed _.

```
* *: AT 7 ISAT 2 10 MESSAGE 34 PROTECT;
* *: AT 12 MESSAGE 22;
* *: AT 0 MODE 4 0 LET 53 64 INPUT 0 ANYKEY CLS MESSAGE 35 ANYKEY GOTO 23 ABILITY 7 30 DESC;
* *: COPYFF 39 100 LISTOBJ PROCESS 11 NEWLINE;
* *: ATGT 44 ATLT 49 SYSMESS 106 NEWLINE;
* *: ATGT 20 ATLT 27 NOTAT 22 SYSMESS 106 NEWLINE;
* *: AT 30 ABSENT 19 EQ 128 255 CREATE 19 PLACE 19 255 MESSAGE 118 ANYKEY MINUS 128 200 DESC;
_ _ : PROCESS 10;
```

You will need to check your process 2 too. You'll need to replace any VERBs & NOUNS used as aide-mémoire's there too.

If Process 1 or 2 make calls to other process tables, using them as subroutines, then you will need to alter those processes too.

Don't just change every process table in your adventure source code, though. If a process table is called from the response table then the NOUN & VERB entries there will be often be intentional and important, and should not be changed, because they will be used as a match for the parser.

Step 4: Make a SCE file, try to compile it on Amstrad and fix any layout issues

We're technically at the stage where we should now be able to compile the adventure on the Amstrad and see how it looks.

First you need to make the .SCE file using inpaws.

So, from the command line, type...

inpaws cm game.paw -o game.SCE

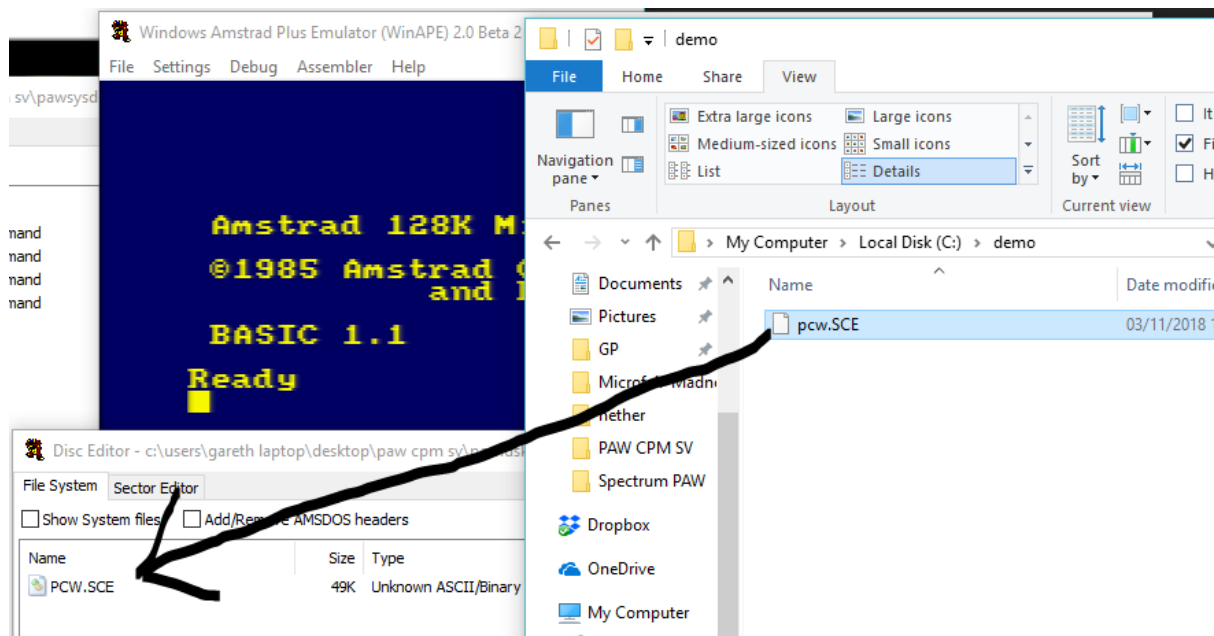
(Where game.paw is the name of your .paw source file)

This should create the file *game.SCE* in your working folder. This is the file we need to put onto a virtual Amstrad floppy disk to load into the emulator.

At this stage I tend to use two Floppy disk images in my Amstrad emulator. I load the PAWs system disk into drive A and I place a second blank CP/M disk in drive B. If you don't have a suitable blank CP/M disk image, you can use an existing disk image and delete the files off it.

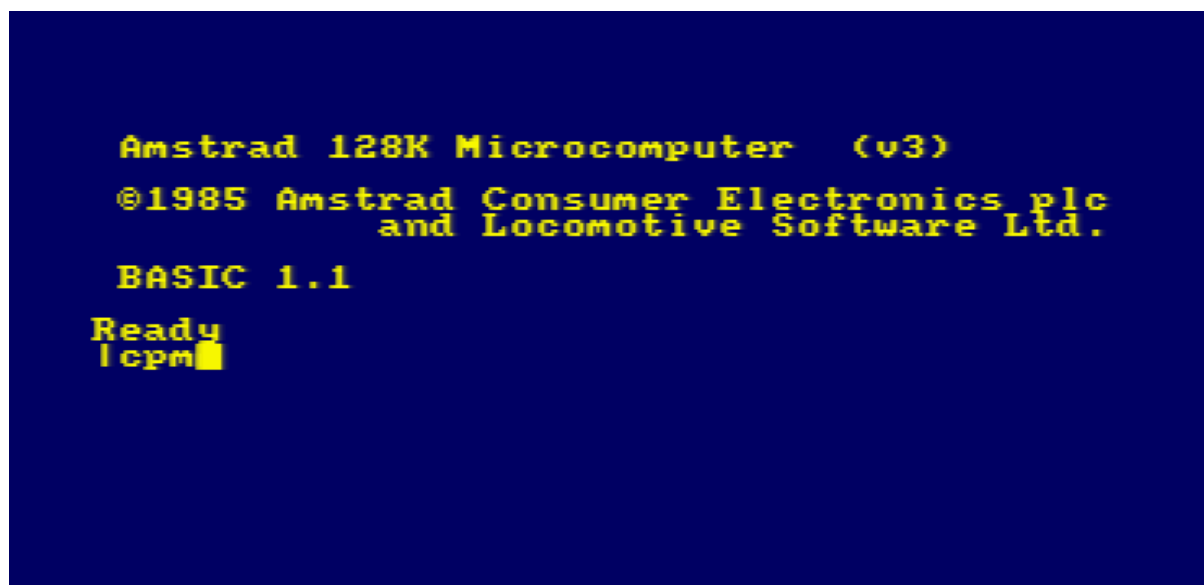
(You can probably get away with just using a single disk image, if you make sure there is plenty of space on it.)

How you get your .SCE file onto the disk image will vary depending on your emulator. I use **WinAPE** for Windows which allows you to insert the disks into the various drives and then edit the disks (file > drive > edit). This brings up a window for the disk where you can drag and drop files from your computer directly onto the disk image.

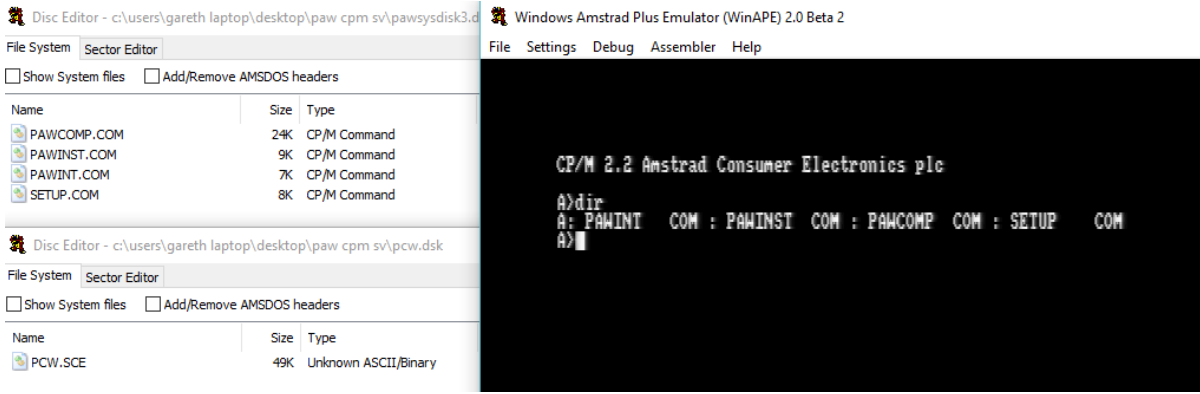


Once you've got your .SCE file onto your disk image you can load up CP/M and get to work on compiling your adventure.

Type **|cpm** by holding shift and the [key (next to the P on a standard UK keyboard) to get the | symbol. This will boot the CP/M operating system off the PAWs disk.



The CP/M system will now load up.

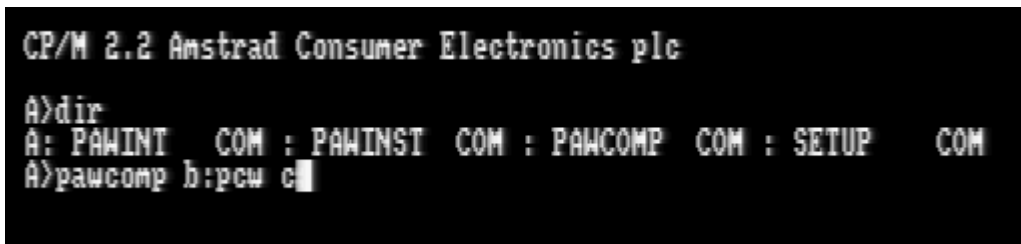


Run the PAWs compiler to turn your .SCE file into a PAWs database .PDB file by typing...

PAWCOMP B:gamename C

...at the command prompt. Where gamename is the name of your gamename.SCE file and B: is the disk it's on. The C is optional, but it tells the compiler to compress the text... You'll probably need to do this.

So, in my example, I input ***PAWCOMP B:PCW C***



This process will take several minutes to complete. Hopefully, at the end of it you will get a compiled game without any errors, like so...

```
1 entries processed successfully
Processing /PRO 5
1 entries processed successfully
Processing /PRO 6
1 entries processed successfully
Processing /PRO 7
47 entries processed successfully
Processing /PRO 8
2 entries processed successfully
Processing /PRO 9
1 entries processed successfully
Processing /PRO 10
2 entries processed successfully
Processing /PRO 11
2 entries processed successfully
Processing /PRO 12
1 entries processed successfully
Processing /PRO 13
2 entries processed successfully
Processing end routines
Compressing the text saved 9845 bytes
Database ends @ address 27783
12920 bytes free in current memory size
*** Compilation ends OK
A>|
```

If you don't, then first check that the errors you're seeing aren't caused by a missed control code. If you're still seeing errors then check the "additional problems" section at the end of this document for other potential issues.

Hopefully, you should now have a compiled PDB file on disk A. What I tend to do at this point is use the disk editor to delete the .SCE file on disk B and then move the PDB file from A to B.

Now it's time to play and test your game. There is a second program on the PAWs disk to do this. It's the interpreter, **pawint**.

Type...

pawint b:gamename c

Where, once again, gamename is the first part of the gamename.PDB filename. B: presumes that, like me, you've moved it over to the second disk. Adding the C to the command presents us with the option to play or make a self-executing standalone version of the game.

```
A>pawint b:pcw c
PAW Interpreter Version 1.6 by Graeme Yeandle & Tim Gilberts
(c) 1987 GILSOFT International Ltd.

Press C to Create a copy of the adventure or P to Play the adventure
|
```

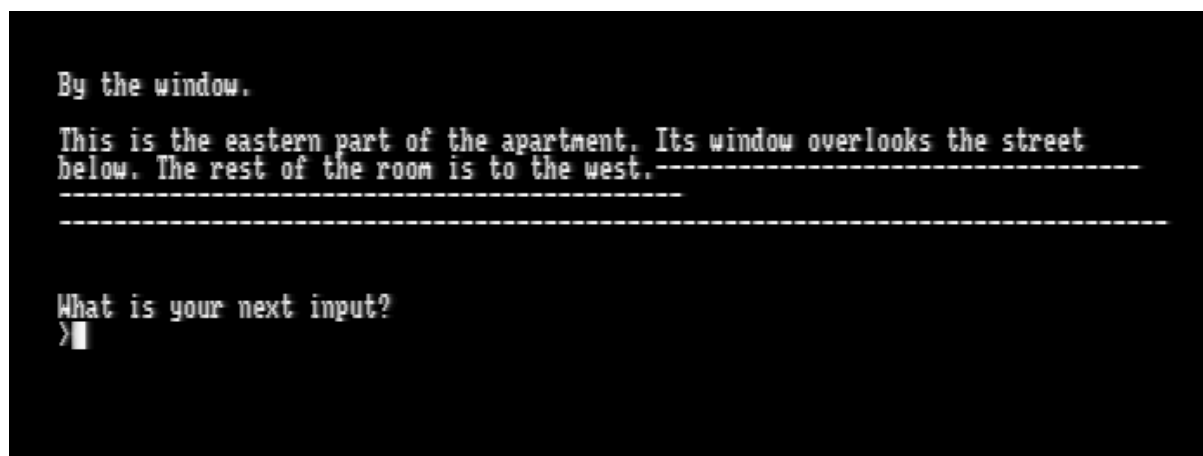
For now, just hit P to play your adventure. It's far from being done. You'll be looking for any bugs but what you're really concentrating on now is any layout issues.

PAWs CP/M games are (by default) 80 columns wide, rather than the standard 32 columns of Spectrum PAWs, so there will be differences to how everything looks on screen.

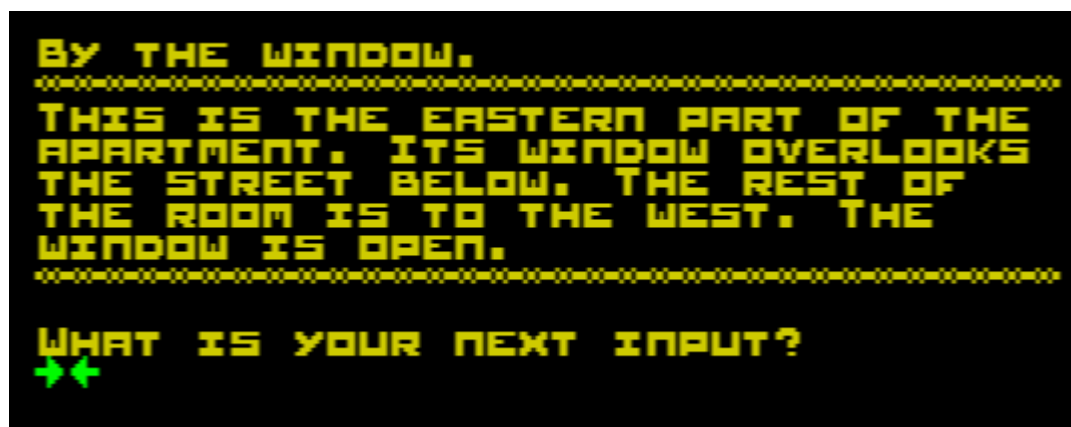
If you've manually centred text in the Spectrum version, using spaces, then you may want to adjust the number of spaces to hit the centre of the Amstrad screen.

The Spectrum version also gives you more control of how you can print things on the screen. If you've used some of those advanced features you'll have to make some changes for your Amstrad version.

Here is an issue that I noticed at one point in the example database I'm currently working through...



I've got two dividing lines on screen, and one is in the wrong place. In the Spectrum version, the first of those dividing lines would've been printed underneath the "By the window." text.



This was accomplished on the Spectrum by using some of the more advanced screen manipulation commands...

```

* _ : AT 41 NOTZERO 93 SYSMESS 72,
* _ : ZERO 250 ATLT 40 CHARSET 1 PROMPT 0 NEWLINE LISTOBJ SYSMESS 0 NEWLINE PROTECT;
* _ : AT 38 WORN 13 ZERO 69 ANYKEY CLS MESSAGE 52 ANYKEY MESSAGE 53 ANYKEY MESSAGE 54 ANYKEY PLACE 12 25
* _ : ATGT 39 CHARSET 2 PROMPT 69 SAVEAT PRINTAT 1 0 SYSMESS 67 BACKAT NEWLINE LISTOBJ SYSMESS 67 NEWLI
* _ : AT 40 ZERO 93 ANYKEY CLS MESSAGE 141 ANYKEY MESSAGE 142 ANYKEY MESSAGE 143 ANYKEY PLACE 19 254 PL
}

```

...namely the PRINTAT / BACKAT conducts which allow you to print text at a certain position on the screen then return back to where you were previously printing.

The Amstrad doesn't support advanced screen management commands so this needs recoding for the CP/M version.

```

* _ : AT 38 WORN 13 ZERO 69 ANYKEY CLS MESSAGE 52 ANYKEY MESSAGE 53 ANYKEY MESSAGE 54 ANYKEY PLACE 12 25
#ifdef PAWSPECTRUM
* _ : ATGT 39 CHARSET 2 PROMPT 69 SAVEAT PRINTAT 1 0 SYSMESS 67 BACKAT NEWLINE LISTOBJ SYSMESS 67 NE
#endif
#ifndef PAWSPECTRUM
* _ : ATGT 39 CHARSET 2 PROMPT 69 NEWLINE LISTOBJ SYSMESS 67 NEWLINE PROTECT;
#endif
* _ : AT 40 ZERO 93 ANYKEY CLS MESSAGE 141 ANYKEY MESSAGE 142 ANYKEY MESSAGE 143 ANYKEY PLACE 19 254

```

My solution here is to just get rid of the additional dividing line. If I really want it to appear there then I can just add it manually to the location text for that section of the Amstrad game. In this instance, it only applies to a small number of locations and I have enough free memory to add the extra 80 characters to each specific location text.

Another thing I spotted...

```

Would you like another go?
>n
©1992 Electric Storm Productions

```

The copyright symbol in the Spectrum version is produced with the code {127}. On the Amstrad it's best to use (c) as CP/M PAWs won't accept the Amstrad's code for ©, which is {164}.

```

Your next action, please?
>exam sign
The sign reads, "CAN YOU COMPLETE THE VIRTUAL REALITY GAME?
Entrance fee only `5 with a chance of winning `5,000!!".

```

Again, the Spectrum version uses a code {96} to get the £ symbol but the Amstrad PAWs doesn't like this or (apparently) using £ itself. So, until I can find another solution, I'll settle for this...


```
TY GAME?^{16}{4}Entrance fee only {96}5 with a chance of winning {96}5,000!!{5}{16}{3}\".";  
TY GAME?^{16}{4}Entrance fee only 5 POUNDS with a chance of winning 5,000 GBP!!{5}{16}{3}\"."
```

And, barring a few other similar corrections and a few over '25-years later' bug fixes, that's the Amstrad version of my game done.

One final thing to bear in mind is your use of colour in your Spectrum adventure and the lack of any in the Amstrad version. If you've relied on coloured text to give subtle clues then you may wish to add extra hints in the text of the Amstrad version.

When you're happy with how everything looks and you've tested everything thoroughly you can use **pawint** again to create a standalone **.com** file which can be placed on a CP/M disk or disk image and played by others (choose the option 'create a copy').

Your inPAWs source can also be used (if correctly marked-up) to produce a revised Spectrum version of your game and also a PC DOS version. The PC DOS version is constructed similarly to the Amstrad version. First you use inPAWs to create a specific PC DOS .SCE file before compiling that .SCE file on a PC. The only real issue you might face is that the PC you use for the compiling stage needs to be running a 32-bit version of Windows. Once you've made the database, though, it will run on any device that can use DOSbox.

I hope that this document has helped you convert your Spectrum adventure to the Amstrad. What follows are some additional, rarer issues that some adventures may throw up.

Additional problems you may encounter...

You can't convert a 128K game

Sadly inPAWS only works with 48K adventures. You can't convert anything larger, such as a 128K adventure. If your Spectrum game is particularly close to the 48K limit then you may have to think carefully about any edits or additions you make to adapt it to Amstrad... although, the space freed by the font should usually help.

Errors when compiling an inPAWs file when nouns have "numerical" names

inPAWS doesn't currently seem to like nouns that begin with numbers e.g. 10P (for a 10p coin) when they're used in the source code, such as for the response table entry **EXAM 10P**. I've also experienced a few odd glitches with "numerical" nouns, even when inPAWS has seemingly successfully compiled the file without errors. To get around these problems, use a defined non-numerical synonym in the code instead, such as **EXAM COIN**. EXAM 10P will still work in your game.

Blank entries causing compilation errors on the Amstrad

When creating the Amstrad version, inPAWS will ignore or strip-out any of the Spectrum commands that are not used on the Amstrad including a lot of the formatting commands such as INK, PAPER etc.

If you've used these in an entry that doesn't have any other CondActs in, for example

```
* * INK 7 PAPER 0;
```

then that will create a blank entry in the table of your database that will cause odd errors when you try to compile them on the Amstrad.

```
* * ;
```

Either combine these commands with entries on another line, remove them, or use an #ifdef PAWSPECTRUM / #endif to only apply them to the Spectrum version.

Here is an example, from the Jack Lockerby game Kidnapped, of an empty line of code causing issues...

```
PROCESS 4
{
  * *: ZERO 2 DONE;
  * *: NOTEQ 103 1 DONE;
  * *: SAVEAT;
  * *: EQ 100 19 PRINTAT 5 0;
  * *: EQ 100 20 PRINTAT 4 0;
  * *: EQ 100 21 PRINTAT 3 0;
```

In the Spectrum code above you can see that there is a line which only has SAVEAT on it. CP/M PAWs does not understand that CondAct so inPAWs will strip it out when you compile your .SCE file leaving you with an empty line, which the Amstrad compiler does not like...

```
/PRO 4
ZERO 2
DONE
NOTEQ 103 1
DONE
EQ 100 19
EQ 100 20
EQ 100 21
```

Split lines appear in your text

When you play your game on the Amstrad you may see odd line breaks in the location or message text. When you go back into your code you'll notice that your Spectrum source had seemingly random ^ line breaks manually forcing new lines rather than letting PAWs format the text automatically.

There is a known issue with the Spectrum PAWs where a word coloured by character codes will become unglued to the adjoining punctuation.

For example, when colouring the text like this:

"{16}{7}The car was {16}{5}blue{16}{7}."

PAWs will see "blue" and "." separately. It's not normally an issue, but when the "blue." appears near the end of the line, you can sometimes end up with "blue" on the first line and the "." Punctuation on the second.

The car was blue

A workaround in Speccy PAWs was to put a line break before the “blue”, to force both “blue” and “.” onto the next line and stop them being split. i.e.

```
“{16}{7}The car was^{16}{5}blue{16}{7}.”
```

The car was blue.

Be aware of this and adjust the code for your Amstrad version to remove these workarounds as it’s a bit more noticeable when the line suddenly breaks in an odd place when your lines are 80 characters long.

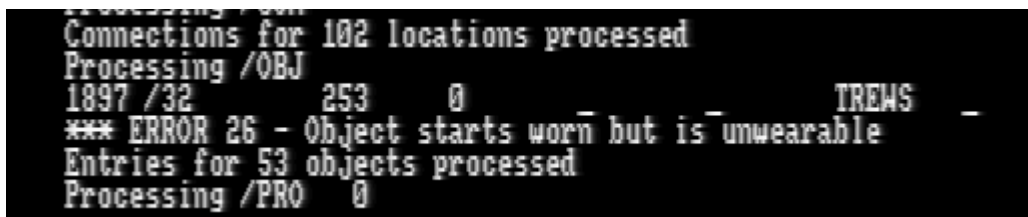
Some code works in Spectrum PAWs but not in Amstrad PAWs

Amstrad PAWs doesn’t like you using the LET command on the player location flag, for some reason. E.g. LET 38 5 (set the player location to location 5). In most cases, simply replacing it with GOTO will be fine, e.g. LET 38 5 becomes GOTO 5. If the author is doing complex operations on the location flag then things may be more problematic.

Error 26 – Object starts worn but is unwearable

This issue was spotted by John Wilson, when converting some of Jack Lockerby’s old Spectrum games to Amstrad CP/M.

A working Spectrum adventure database throws up an error when converted to run on the CP/M version of PAWS, namely **error 26 – Object starts worn but is unwearable**.



```
Connections for 102 locations processed
Processing /OBJ
1897 /32 253 0 TREWS
*** ERROR 26 - Object starts worn but is unwearable
Entries for 53 objects processed
Processing /PRO 0
```

The CP/M PAWs is once again being a lot stricter about how an adventure is defined or, in this case, how an object is defined.

Let’s look at an example database, namely Jack Lockerby’s Kidnapped game, which is what produced the error shown above.


```
OBJECT 32
{
  "your usual clothes";
  INITIALLYAT WORN;
  WORDS TREWS _;
  WEIGHT 0;
}
```

To correct the issue, we simply need to add an additional line to define object 32 as being wearable/removable....

```
OBJECT 32
{
  "your usual clothes";
  INITIALLYAT WORN;
  WORDS TREWS _;
  WEIGHT 0;
  PROPERTY CLOTHING;
}
```

Depending on the game, you may need to add a line in the response table that stops the player taking off this object (that the Spectrum author never originally defined as removable).

In the case of Kidnapped, there is already a response (for REMOV TREWS) to handle this...

```
REMOV ALL: DOALL 253;
REMOV SHIRT: LET 34 55;
REMOV SOCKS: LET 34 55;
REMOV SHOES: LET 34 55;
REMOV CLOTH: LET 34 55;
REMOV TREWS: MESSAGE 64 DONE;
REMOV _: AUTOR DONE;
WEAR ALL: DOALL 254;
WEAR _: AUTOW DONE;
```

(Thanks again to John Wilson for bringing this additional issue to my attention)

Still stuck? Need more help?

You can find lots of Spectrum to Amstrad conversions on my website at:

<http://8bitAG.com/games>

with all the associated PAWs and inPAWs source files.

There are additional documents on inPAWs and Spectrum/Amstrad PAWs, including a comparison between the two versions of PAWs on the website at:

<http://8bitAG.com/info>

Please don't hesitate to get in touch if you wish to discuss your conversions.