

Copyright © Drag 'N Drop 2016 Produced on RISC OS computers

This issue has been blessed with contributions from the following people: Robin Wilks (Forth) Paul Dunnington (Python Primary School) Christopher Dewhurst (everything else)

The views expressed in this magazine are not necessarily those of the editor. Alternative views are always welcome and can be expressed by either writing an article or a short editorial.

All articles and advertisements are published in good faith. No materials in this publication are meant to be offesnsive or misleading. If you come across something you believe is either of the above please contact the editor using the details below.

> Contact Information Editor: Christopher Dewhurst Email: editor@dragdrop.co.uk www.dragdrop.co.uk

EDITORIAL

Welcome to Volume 8 of *Drag 'N Drop.* The Pi-Top will be featuring on the magazine covers, it's a laptop which could be described as the first native ARM machine since the Acorn A4.

CJE Micros have been hard at work writing RISC OS support software including desktop apps for brightness control and battery indicator. Of course, we've never had so many choices of hardware on which to run RISC OS so if Pi-Top isn't your cup of tea then the ARMX6 or something called the "LaPi AtrixRO" might!

For those attending the London Show and want the *Drag 'N Drop* back issues stick we have a special birthday 'memory cube' (limited stock) Hope to see you there!



Editorial	1
How do I?	2
News	3
Files of the World: Midi	5
Forth on the Pi	11
M/Code stars	14
Floozy	18
Python Primary School	24
Armcode for	00
Beebsters	30
256-colour palette switching	34
Fibonacci Wallpap	er
	38
MemAlloc module	39
Autofocus utility	42

How do I...?

...get the BBC Basic prompt?

To get the BBC Basic prompt press F12 and type *BASIC and press Return. You can change the screen mode with MODE n where n is a number e.g. MODE 7 or MODE 0. Type AUTO for automatic line numbering. Press Escape to stop and type *SAVE "myprog"* followed by Return to store *myprog* on hard disc.

To return to the desktop type *QUIT. Programs listed in *Drag 'N Drop* are assumed to work on all machines with RISC OS 5 e.g. Raspberry Pi, unless otherwise stated.

...open a Task window?

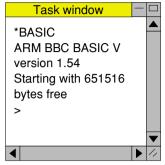
Menu click over the Raspberry icon on the right side of the iconbar and select click on Task window. Or press Ctrl + F12.

Next

xt 1440K

You may need to reserve more memory for the task in which case adjust-click on the Raspberry icon and under Application tasks click and drag the Next slide bar out to the right.

You can also type programs in a task window, hold down Ctrl and press F12. You can't use the cursor editing facility or change MODE, however.



You can also program and run Basic programs from the desktop. Double-clicking on the filer



icon runs it, holding down Shift and double clicking loads it into your text editor.

...select the currently selected directory?

Articles may tell you to set the CSD (currently selected directory). Just click menu over filer window and choose Set directory ^W or you can use the !EasyCSD application presented in *Drag N Drop* 6i1.

...open an Applcation Directory?

Application directories begin with a ! called 'pling'. Hold down shift and double click select to open the directory.

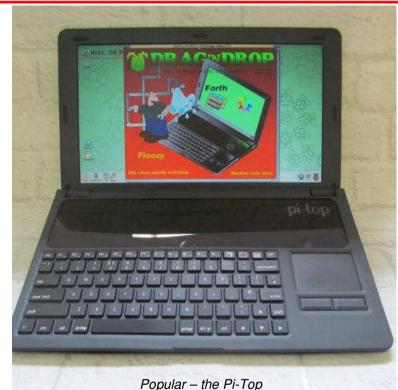
News and App Updates

RISC OS London Show

The RISC OS premier show takes place on 29th October at the St Giles Hotel in Feltham, London. The show runs from 11am to 5pm and costs £5 to get in on the door. Come along for an eclectic mix of Acorn past and RISC OS present including an upgrade to the popular Pi-Top computer. Get the latest *Drag N Drop* and lots of other special offers from our stand including a limited edition memory box (flash USB and chocolates) to celebrate the start of volume 8.

RISCBook Go!

R-Comp Interactive have unveiled the latest option for running RISC OS under emulation on a Windows machine, with the bonus that most legacy RISC OS software works! Prices for the RISCBook Go! start at £349 and more details are to be found at http://www.riscbook.co.uk



Text>Draw 1.32 T

This application aids inputting of text containing different styles (subscript, italic etc.) in Draw. The latest version can be downloaded for free from <u>http://www.chris-johnson.org.uk/software/</u>tdraw.html.

TranJPEG 1.33

A desktop utility for rotating and transforming JPEG photos(using jpegtran) which works on the Raspberry Pi. Download from http://www.chris-johnson.org.uk/ software/tranjpeg.html.

AMPlayer 1.41

AMPlayer is a relocatable module which plays audio MPEG files such as MP3s. Desktop apps like DigitalCD use the module and the latest version of AMPlayer can be downloaded for free from <u>http://</u> <u>www.riscos.info/index.php/</u> <u>AMPlayer</u>.

Impact 3.51

A minor upgrade to fix a fault in the version of the relational database released at the RISC OS Wakefield Show is free to users already on version 3.47 or later. See <u>http://sinenomine.co.uk/</u> <u>software/impact/</u>.

Fireworkz 2.20

A major update to the commercial version of the spreadsheet application is to be released at the London show. A promising new feature is support for a global clipboard so you can copy data from and to other applications and **R** even between icons. Da

Both RISC OS and a Windows version of Fireworkz come on one CD-Rom costing £39 (upgrades from earlier versions at lower cost) with a Getting Started HTML Guide.

CloudFS

Elesar (the people responsible for development of the Titanium



computer) have released CloudFS which is a RISC OS filing system allowing connection to storage on a

remote server. Files can be seen by your smartphone, PC etc. as well. CloudFS costs £28.80 and more details are at <u>http://shop.elesar.co.uk/</u> <u>index.php?route=product/</u> <u>product&product_id=63</u>

ODS

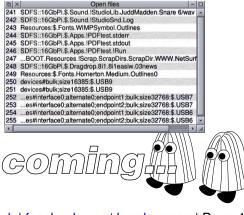
RISC OS FR

David Feugey has been busy supporting french-speaking RISC OS users at <u>https://www.riscos.fr/,</u> English version at <u>https://</u> <u>www.riscos.fr/english.html</u>. The website now hosts a *pret-amanger* version of RPCEmu for newcomers to get started more easily in the RISC OS world. Also RISC OS FR now hosts subdomains, the first one being <u>https://www.henrikbp.riscos.fr/</u>

Closer 1.01

A simple but useful desktop app showing details of currently open files and the opportunity to close them. Version 1.01 can be downloaded from

https://www.henrikbp.riscos.fr/



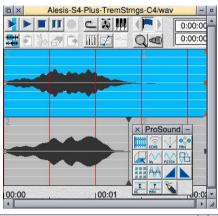


are

StudioSound 2.05

Fully Raspberry Pi-compatible, StudioSound lets you compose abstract electronic music soundscapes by dragging and dropping WAVe samples onto the main window with options to control track volume, pan, digital sound processing (DSP), looping etc. The latest version is FREE and can be downloaded from https://www.henrikbp.riscos.fr/. Note you may also require the THHeap module from

http://www.filebase.org.uk/ software/programming/982





ProSound 2.01

This is a sophisticated sound sample player and editor. A variety of effects can be applied to samples such as WAV files – flange, fade in/out, echo etc. Edited samples can then be used to compose your piece in StudioSound. We have found it ProSound works and feels a lot better than PlayIt and it's also FREE from <u>https://</u> www.henrikbp.riscos.fr/

DisAssem 3.26

A desktop utility to disassemble ARM machine code files and modules and available free from https://www.henrikbp.riscos.fr/

×	SCSI::SKYBOX.\$.32bitting.lzw.LZW
000000	00000000 ANDEQ R0,R0,R0
000008 00000C	LDREQD R0,ER0], 00000000 Andeq R0,R0,R0 00000000 Andeq R0,R0,R0
LZW	0000002C ANDEQ RO,RO,R1 000000030 ANDEQ RO,R0,R0
Miscellaneous	Goto
Goto address 11F4 Cursor	Goto address &00000000 Got
Set mark F6)	Branch by & Bran
Marks	V Update
Data 🕨	Z 10 30282032 EURLL RZ, R8, R2
Find F7	9 0c 634F2039 EQUD &634F203 t 19 39312074 LDMCCDB R1!.(R2.
Options F2	90), 00293039 EOREQ R3, R9, R9
Save file F3 >	000 200C000F R3063 R0,R14,F
Scanner ^F11 >	Kort 74726F4B LDRVCBT R6,[R2], ink. 2C6B6E69 STCCSL CP14.C6.

Files of the World

5 MIDI Files

In this instalment we look at Midi Files a development of the Musical Instrument Digital Interface (MIDI) standard devised in the 1980s.



We'll first take a quick look at an application which plays Midi files, then the Midi file structure itself, and finally armed with the information learnt we'll write our own BASIC program to display and play Midi files – very simple, it has to be said, but with much scope for enhancement.

The Midi player I use on my Raspberry Pi is ReMIDI and its home page can be found here: <u>http://www.amplitude.demon.nl/</u> remidi.html.

Download rel061p.zip (2477k)

complete with samples. Copy this out into a suitable



location on your hard disc. Then download lyonix compatible

version without samples and

copy it over the top. Double click to put ReMIDI on the icon bar.

Next you need some Midi files, there are thousands of free ones on the internet.

http://www.mfiles.co.uk/ is a good place to start and is compatible with Netsurf.

Midi files will usually be recognised and filetyped MIDI by Netsurf. If they aren't then use the filer to set the filetype to &FD4.

Then just drag the file onto ReMIDI's iconbar icon and it will play.

Midi files are sequences of musical events and can be thought of as like instructions on which note(s) to play on the piano keyboard and when.

The actual piano sound sample MTho is stored on your computer (within hex). ReMIDI in fact). Note this is Th different from WAV files which we looked at last time. Waveform the M data isn't included in Midi files, 6 but just directions on when and how the 6 to play them. first?

We're going to look at a simple tune, *Twinkle Twinkle Little Star* as an example.

Figure 1 is an annotated dump of the first 256 bytes of *Twinkle*. The left hand column is the offset in bytes from the start of the file, the middle column the bytes in hex, the right hand column the Ascii representation – which is what you would see in a memory editor. Lines across the top of bytes in the middle column group particular bytes in order.

By now we're used to files having a *header* of some sort. This is the first few bytes of a file identifying what the file is. Midi files are no exception.

tich note(s) to play on the piano yboard and when. The actual piano sound sample MThd (the bytes 4D 54 68 64 in stored on your computer (within hex).

> The next word is the length of the header in bytes (not including the MThd) which has always been 6 but subject to change. Why is the 6 the fourth byte and not the first? I'll come to that later.

Forth on the Pi

Forth is a computer language created by Charles Moore in the 1960s. "It behooves new programmers to sample all languages available. Forth is the only one that's fun," he said.

In the beginning Forth wasn't so much a computer language as a complete programming system with operating system, editor, assembler plus a compiler for finished code and an interpreter for program development.

In Forth there are just a few primitive instructions called words (forming the nucleus of the system) with all other words being defined in terms of these primitives, or each other.

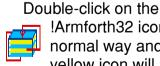
So Forth is 'extensible' - new words are treated the same as those in the nucleus. To run a new word its name only has to be typed (or loaded) and parameters are passed between words via the stack.

Nowadays Forth is usually simulated and is available for many different OSs including **BISC OS on the Pi**.

!ARMForth is available from mv website http://www.rforth.uk and was originally written in 1992 by Rob Turner. Since the source code is written in BBC Basic and assembly code (using Basic's built-in assembler) it was relatively easy to update it to run on modern RISC OS machines.

I have called the updated version ARMForth32 to distinguish it from the original software which would run on the Acorn Archimedes only.

Download ForthPi/zip from the Downloads section. Then extract **!ARMforth32** to a suitable location on your hard disc e.g. in 16GbPi.\$.Programming.



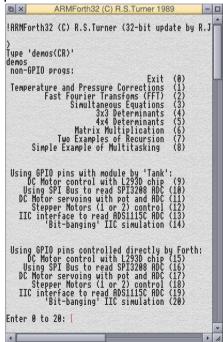
!Armforth32 icon in the normal way and a blue and yellow icon will appear on the icon bar.

Clicking select on this icon will launch the Forth application specified in the file

!ARMforth32.Examples.

!Startup. In this case !Startup is a Forth program giving a menu of example programs. Type **demos** followed by Return for a list of example programs, then a number between 1 and 20 to run example programs.

Incidentally I have not used a Pi3 so don't know if there are any problems with ARMForth32 it.



M/Code Scrolling Stars

Here we feature several machine code routines which scroll a field of stars on the screen, ideal for backgrounds in arcade games.

After typing one or more listings of your choice save it in case you've made a typing slip in the assembly code which could hang the machine.

The top left hand corner of the screen has an address in Ram which varies from screen mode to screen mode and machine to machine.

Some calls to the operating system are made to ascertain this value, the extent of the screen area, and also the number of pixels across the screen. A line by line commentary on how the program works is given below.

S 7	ARS						
	10REM	***	M/C	Scr	olling	g Sta	ars *
**							
	20REM	(c)	Drag	1 'N	Drop	Oct	2016
	30MODI	E 13					
	400N	ERROR	REF	PORT	:PRIN1	[" al	: ";E
RL :	END						
	500FF						

60PRINT TAB(9,15)"SCROLLING STA RS DEMO"r pointer70ns%=403600PL loop\repeat until80PROCmc370LDMFD sp!,{pc} \return to Ba90CALLinit380100FOR i%=0 TO ns%+4-4 STEP 4380110i%!stars%=RMD(!scrsiz%)+!scrt380.scroll00%400STMFD sp!,{lr} \store copy of120NEXTBasic return addr130CALL plot410BL plot \repset stars140REPEAT420LDR r5,scrend% \screen ram e160a=TMKEY(5)430LDR r6,scrsiz% \screen ram size in r6180END440MDR r0,stars% \star table a190440MDR r0,stars% \star table a19010 r 1, star pointer200DEF PROCmc450MOV r1,#ns%-1 \no of stars mi200DEF PROCmc460DR r4,srars% \star table a19010 r 1, star pointer200DEF PROCmc460MDV r2,r1,lsl#2 \r2 = r1*4200DR r0,stars% \star table adcreen addr10 n r 8screen in r4200DR r0,stars% \star table adcreen addr10 n r 8screen?200LDR r4,r4,Ir31 \load byte fscreen?300LDR r4,r4,#RND(255) \EOR it500BL plot \replot stars3040BR r4,r4,#		
RS DEMO"360BPL looprepeat until70ns%=40360BPL looprepeat until80PR0Cmc370LDMFD sp!,{pc}return to Ba90CHLLinit370LDMFD sp!,{pc}return to Ba100FOR i%=0 To ns%*4-4 STEP 43803801100i%istars%=RND(!scrsiz%)+!scrt390, scroll3800p%480STMFD sp!,{lr} \store copy ofBasic return addr130CALL plot410BL plot \erase stars420LDR r5, scrend% \screen ram e140RCPEAT440EDR r6, scrsiz% \screen ram s160a=TNKEY(5)430LDR r6, scrsiz% \screen ram s170UNTIL 0440EDR r6, scrsiz% \star table a190440EDR r8, stars% \star table a190200DEF PROCmc210DIM code% 1000440EDR r4, xpix% \no of pixels a220FOR pass=0 TO 2 STEP 2460LDR r4, xpix% \no of pixels a220FOR pass=0 TO 2 STEP 2460UDR r4, xpix% \no of pixels a220FOR pass=0 TO 2 STEP 2480MOU r2, r1, lsl#2 \r2 = r1*4200DEF PROCmc500HOU r3, r3, r4 \add no pixel280MOU r1, #ns%-1 \no of stars m510CMP r3, r5 \off bottom o100 r3reen addr500HOD r3, r3, r4 \add no pixel380EDR r4, r73] \load byte f530STR r3, r6 \subtract scr380EDR r4, r4, #RND(255) \sub r50BPL loop2 \until done all 5340STRB r4, Lr3] \store back560BL plot \replot stars	60PRINT TAB(9,15)"SCROLLING STA	r pointer
70ns%=4080PR0Cmc370LDMFD sp!,{pc} \return to Ba90CALLinit370LDMFD sp!,{pc} \return to Ba100FDR i%=0 TO ns%*4-4 STEP 4390.scroll110i%!stars%=RND(!scrsiz%)+!scrt390.scroll07%120NEXT120NEXT390.scroll130CHL plot400SIMFD sp!,{lr} \store copy of140REPEAT400LR r5,scrend% \screen ram e160a=TNKEY(5)420LDR r5,scrend% \screen ram s160a=TNKEY(5)420LDR r6,scrsiz% \screen ram e160a=TNKEY(5)4400DR r0,stars% \star table a1904400DR r0,stars% \star table a200DEF PR0Cmc450M0U r1,#ns%-1 \no of stars mi220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a220FOR pass=0 TO 2 STEP 2700RDR r0,stars% \star table ad230P%=code%470.loop2250.plot470Alloop2280M0U r1,#ns%-1 \no of stars m500RD r3,r3,r4 \add no pixel300LDR r6,stars% \star table ad500RD r3,r3,r4 \add no pixel300LDR r4,r1sl#2 \r2 = ri*4500RD r3,r3,r4 \add no pixel300LDR r4,r4,#RND(255) \store530STR r3,Lr0,r21 \store update300EDR r4,r4,#RND(255) \store500BL loop2 \until done all 5340STRB r4,Lr31 \store back560BL plot \replot stars	RS DEMO"	360BPL loop \repeat until
90CALLinitsic100FOR i%=0 TO ns%*4-4 STEP 4380110i%!stars%=RND(!scrsiz%)+!scrt380, scroll0p%390, scroll120NEXT390, scroll130CALL plot4408TMFD sp!,{lr} \store copy of130CALL scrolln r5160a=TMKEY(5)420LDR r6, scrsiz% \screen ram e160a=TMKEY(5)430LDR r6, scrsiz% \screen ram s160a=TMKEY(5)430LDR r6, scrsiz% \screen ram s160a=TMKEY(5)430LDR r6, scrsiz% \screen ram s170UNTIL 01001200DIM code% 1000100200DEF PROCmc100220FOR pass=0 TO 2 STEP 2460LDR r4, xpix% \no of pixels a230P%=code%cross screen in r4240E OPT pass700, nop250, plot470, loop2250, plot400MOV r2, r1, lsl#2 \r2 = r1*4200MOV r1, #ns%-1 \no of stars m10us l in r1500ADD r3, r3, r4 \sdd no pixel290, loopscreen addr300LDR r3, Lr0, r2] \get star'sscreen address520SUBGE r3, r3, r6 \subtract scr320LDR r4, r4, #RND(255) \EOR itwith 255Soreen ram330EOR r4, r4, ffx12 \store backin screen ram330EOR r4, r4, ffx12 \store backin screen ram340STRB r4, [r3] \store backin screen ram340STRB	70ns%=40	done all 50
90CALLinitsic100FOR i%=0 TO ns%*4-4 STEP 4380110i%!stars%=RND(!scrsiz%)+!scrt380op%390.scroll120NEXT390.scroll130CALL plot410BL plot \erase stars140REPEAT420LDR r5,scrend% \screen ram e160a=TMKEY(5)430LDR r6,scrsiz% \screen ram s160a=TMKEY(5)430LDR r6,scrsiz% \screen ram s160a=TMKEY(5)430LDR r6,scrsiz% \screen ram s160a=TMKEY(5)430LDR r6,scrsiz% \screen ram s180END440ADR r0,stars% \star table a1901 in r1200DEF PROCmc1 in r1220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a230P%=code%1 in r1250.plot470.loop2250.plot490MOV r2,r1,lsl#2 \r2 = r1*4200MOV r1,#ns%-1 \no of stars m500ADD r3,r3,r4 \add no pixel280MOV r2,r1,lsl#2 \r2 = r1*4500STR r3,Cr0,r21 \get star's300LDR r3,Cr0,r21 \get star's530STR r3,Cr0,r21 \store update330EOR r4,r4,#RND(255) \EOR it500BL loop2 \until done all 5340STRB r4,[r31 \store back560BL plot \replot stars		370LDMFD sp!.{pc} \return to Ba
100FOR i2=0 TO ns%+4-4 STEP 4 110i%!stars%=RHD(!scrsiz%)+!scrt 390.scroll 400STMFD sp!,{lr} \store copy of Basic return addr 410BL plot \erase stars 420LDR r5,scrend% \screen ram e nd in r5 430LDR r6,scrsiz% \screen ram s ize in r6 4400DR r0,stars% \star table a ddr in r0 200PEF PROCMC 200PEF PROCMC 200PEF PROCMC 200PF r0,screen ram s 220FOR pass=0 TO 2 STEP 2 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% 230P%=code% <th></th> <th></th>		
110i%!stars%=RND(!scrsiz%)+!scrt390.scroll0P%120NEXT400STMFD sp!,{lr} \store copy of130CALL plot410BL plot \erase stars140REPEAT410BL plot \erase stars160a=INKEY(5)430LDR r5,scrend% \screen ram e160a=INKEY(5)430LDR r6,scrsiz% \screen ram s180END440ADR r0,stars% \star table a190900EF PR0Cmc200DEF PR0Cmc450MOV r1,#ns%-1 \no of stars mi210DIM code% 100010 r f220FOR pass=0 TO 2 STEP 2450MOV r1,#ns%-1 \no of stars mi230P/*code%450MOV r2,r1,1sl#2 \r2 = r1*4260STMFD sp!,{lr}480MOV r2,r1,1sl#2 \r2 = r1*4260STMFD sp!,{lr}500MOV r2,r1,1sl#2 \r2 = r1*4280MOV r1,#ns%-1 \no of stars mi500MDD r3,r3,r4 \add no pixelsaut in r0500MDD r3,r3,r6 \subtract scr290,loop530STR r3,F0,r21 \store update300LDR r4,r4,#RND(255) \EOR it530STR r3,F0,r21 \store update330EDR r4,r4,#RND(255) \EOR it500BPL loop2 \until done all 5340STRB r4,r631 \store back500BL plot \replot stars	100FOR i%=0 TO ns%+4-4 STEP 4	
op%400STMFD sp!,{lr} \store copy of Basic return addr 410BL plot \verase stars 420LDR r5,scrend% \screen ram e 		
120HEXTBasic return addr130CALL plot4108L plot Verase stars140REPEAT420LDR r5, scrend% Vscreen ram e150CALL scrollnd in r5160a=INKEY(5)430LDR r6, scrsiz% Vscreen ram s170UNTIL 0ize in r6180END4400DR r0, stars% Vstar table a190460LDR r4, stars% Vstar table a200DEF PROCmc460LDR r4, stars% Vstar table a200DEF PROCmc460LDR r4, stars% Vstar table a2200DEF PROCmc460LDR r4, stars% Vstar table a2200PX=code%cross screen in r4240C OPT pass470, loop2250, plot480MOV r2, r1, ls1#2 Vr2 = r1*4260STMFD sp!, {lr}500ADD r3, r3, r4 Vadd no pixel280MOV r1, #ns%-1 No of stars msacross screeninus 1 in r1500ADD r3, r3, r4 Vadd no pixel280MOV r2, r1, ls1#2 Vr2 = r1*4500BDD r3, r3, r6 Vsubtract scr280MOV r2, r1, ls1#2 Vr2 = r1*4500SUBGE r3, r3, r6 Vsubtract scr280MOV r2, r1, ls1#2 Vr2 = r1*4500SUBGE r3, r3, r6 Vsubtract scr280LDRB r4, [r3] Vload byte from screen ram330EOR r4, r4, #RND(255) VEOR itsocreen ram330EOR r4, r4, #RND(255) VEOR itsocreen ram340STRB r4, [r3] Vstore back560BL plot Vreplot stars		
140REPERT420LDR r5,scrend% \screen ram e150CALL scrollnd in r5160a=INKEY(5)430LDR r6,scrsiz% \screen ram s170UHTIL 0ise in r6180END4400DR r0,stars% \star table a1904400DR r0,stars% \star table a200DEF PROCmc45000V r1,#ns%-1 \no of stars mi210DIM code% 1000r0, stars% \star table a220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot48000U r2,r1,lsl#2 \r2 = r1*4260STMFD sp!,{lr}4800U r2,r1,lsl#2 \r2 = r1*4280MOU r1,#ns%-1 \no of stars m500ADD r3,r3,r4 \add no pixel280MOU r2,r1,lsl#2 \r2 = r1*4500ADD r3,r3,r6 \subtract scr280MOU r2,r1,lsl#2 \r2 = r1*4520ENE r3,r3,r6 \subtract scr280MOU r2,r1,lsl#2 \r2 = r1*4520ENE r3,r3,r6 \subtract scr300EN r4,r4,#RND(255) \EOR itstore backwith 255store back340STRB r4,[r3] \store back6500BL plot \replot stars	120NEXT	Basic return addr
140REPERT420LDR r5,scrend% \screen ram e150CALL scrollnd in r5160a=INKEY(5)430LDR r6,scrsiz% \screen ram s170UHTIL 0ise in r6180END4400DR r0,stars% \star table a1904400DR r0,stars% \star table a200DEF PROCmc45000V r1,#ns%-1 \no of stars mi210DIM code% 1000r0, stars% \star table a220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot48000U r2,r1,lsl#2 \r2 = r1*4260STMFD sp!,{lr}4800U r2,r1,lsl#2 \r2 = r1*4280MOU r1,#ns%-1 \no of stars m500ADD r3,r3,r4 \add no pixel280MOU r2,r1,lsl#2 \r2 = r1*4500ADD r3,r3,r6 \subtract scr280MOU r2,r1,lsl#2 \r2 = r1*4520ENE r3,r3,r6 \subtract scr280MOU r2,r1,lsl#2 \r2 = r1*4520ENE r3,r3,r6 \subtract scr300EN r4,r4,#RND(255) \EOR itstore backwith 255store back340STRB r4,[r3] \store back6500BL plot \replot stars	130CALL plot	410BL plot \erase stars
150CALL scrollnd in r5160a=INKEY(5)430LDR r6,scrsiz% \screen ram size in r6180END440ADR r0,stars% \star table a190440ADR r0,stars% \star table a200DEF PROCmc450MOV r1,#ms%-1 \no of stars mi210DIM code% 1000nus 1 in r1, star pointer220FOR pass=0 TO 2 STEP 2460LDR r4,xpi% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot480MOU r2,r1,lsl#2 \r2 = r1*4260SIMFD sp!,{Ir}480MOU r2,r1,lsl#2 \r2 = r1*4280MOU r1,#ms%-1 \no of stars misacross screen300MOU r2,r1,lsl#2 \r2 = r1*4500ADD r3,r3,r4 \add no pixel300MOU r2,r1,lsl#2 \r2 = r1*4510CMP r3,r5 \off bottom o320LDRB r4,Lr33 \load byte fsacross screen330EOR r4,r4,#RND(255) \EOR it530STR r3,[r0,r2] \store update330EOR r4,r4,#RND(255) \EOR it500PL loop2 \until done all 5340STRB r4,[r3] \store back500PL loop2 \until done all 50560BL plot \replot stars	140REPEAT	420LDR r5,scrend% \screen ram e
160a=INKEY(5)430LDR r6,scrsiz% \screen ram s170UNTIL 0ize in r6180END4400DR r0,stars% \star table a1904400DR r0,stars% \star table a200DEF PROCmc4500MOV r1,#ms%-1 \no of stars mi210DIM code% 1000nus 1 in r1, star pointer2200P% code%460DR r4,xpi% \no of pixels a230P% code%cross screen in r4240E OPT pass470.loop2250.plot480MOU r2,r1,ls1#2 \r2 = r1*4260STMFD sp!,{Ir}480MOU r2,r1,ls1#2 \r2 = r1*4280MOU r1,#ms%-1 \no of stars misacross screeninus 1 in r1500ADD r3,r3,r4 \add no pixel280MOU r2,r1,ls1#2 \r2 = r1*4500ADD r3,r3,r4 \add no pixel380MOU r2,r1,ls1#2 \r2 = r1*4500ADD r3,r3,r4 \add no pixel320LDRB r4,Fr3] \load byte f530STR r3,F0,r2] \store update330EOR r4,r4,#RND(255) \EOR it530BTR r3,F0,r2] \store updatewith 255340STRB r4,Fr3] \store back6560BL plot \replot stars560BL plot \replot stars	150CALL scroll	
1/0000111 0122 in r6180END4400DR r0,stars% \star table a190200DEF PROCmc210DIM code% 1000450MOV r1,#ns%-1 \no of stars mi220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot480MOV r2,r1,ls1#2 \r2 = r1*4260STMFD sp!,{Ir}490LDR r3,Er0,r2] \get star's s270ADR r0,stars% \star table ad500ADD r3,r3,r4 \add no pixeldr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars m500ADD r3,r3,r6 \subtract scr280MOV r2,r1,ls1#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300MOV r2,r1,ls1#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr310LDR r3,Er0,r2] \get star's530STR r3,Er0,r2] \store update330EDR r4,r4,#RND(255) \EOR itwith 255340STRB r4,Er3] \store back500BL plot \replot stars	160a=INKEY(5)	
190ddr in r0200DEF PROCmc450MOV r1,#ns%-1 \no of stars mi210DIM code% 1000nus 1 in r1, star pointer220FOR pass=0 TO 2 STEP 2460LDR r4,xpi% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot480MOV r2,r1,ls1#2 \r2 = r1*4260STMFD sp!,{1r}490LDR r3,Er0,r21 \get star's s270ADR r0,stars% \star table adcreen addrdr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars m500ADD r3,r3,r4 \add no pixel280MOV r2,r1,ls1#2 \r2 = r1*4500ADD r3,r3,r4 \add no pixel280MOV r2,r1,ls1#2 \r2 = r1*4500ADD r3,r3,r6 \subtract scr280MOV r2,r1,ls1#2 \r2 = r1*4510CMP r3,r5 \off bottom of300MOV r2,r1,ls1#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr310LDR r3,Er0,r21 \get star's530STR r3,Er0,r21 \store update320LDRB r4,Fr31 \load byte fsacress screenrom screen ram540SUBS r1,r1,#1 \decrease star330EDR r4,r4,#RND(255) \EOR itpointerwith 255340STRB r4,Er31 \store backin screen ram560BL plot \replot stars	170UNIIL 0	ize in r6
190ddr in r0200DEF PROCmc450MOV r1,#ns%-1 \no of stars mi210DIM code% 1000nus 1 in r1, star pointer220FOR pass=0 TO 2 STEP 2460LDR r4,xpi% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot480MOV r2,r1,ls1#2 \r2 = r1*4260STMFD sp!,{1r}490LDR r3,Er0,r21 \get star's s270ADR r0,stars% \star table adcreen addrdr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars ms across screen1nus 1 in r1500ADD r3,r3,r4 \add no pixel280MOV r2,r1,ls1#2 \r2 = r1*4500ADD r3,r3,r4 \add no pixel280MOV r2,r1,ls1#2 \r2 = r1*4500SIDE r3,r3,r6 \subtract scr300MOV r2,r1,ls1#2 \r2 = r1*4500SIDE r3,r3,r6 \subtract scr300MOV r2,r1,ls1#2 \r2 = r1*4500SIDE r3,r3,r6 \subtract scr300DR r4,r4,#RND(255) \EOR ityointerwith 255340SIRB r4,Er31 \store back340SIRB r4,Er31 \store back6560BL plot \replot stars		440ADR r0,stars% \star table a
210DIM code% 1000nus 1 in r1, star pointer220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot480MOV r2,r1,lsl#2 \r2 = r1*4260STMFD sp!,(1r)490LDR r3,Er0,r21 \get star's s270ADR r0,stars% \star table adcreen addrdr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars ms across screen1nus 1 in r1500ADD r3,r3,r4 \add no pixel280MOV r2,r1,lsl#2 \r2 = r1*4500ADD r3,r3,r6 \subtract scr290.loopf screen?300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300LDR r3,Er0,r21 \get star'ssoreen ram size if so330EOR r4,r4,#RND(255) \EOR it550BPL loop2 \until done all 5340STRB r4,Er31 \store back6in screen ram560BL plot \replot stars		
210DIM code% 1000nus 1 in r1, star pointer220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot480MOV r2,r1,lsl#2 \r2 = r1*4260STMFD sp!,(1r)490LDR r3,Er0,r21 \get star's s270ADR r0,stars% \star table adcreen addrdr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars ms across screen1nus 1 in r1500ADD r3,r3,r4 \add no pixel280MOV r2,r1,lsl#2 \r2 = r1*4500ADD r3,r3,r6 \subtract scr290.loopf screen?300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300LDR r3,Er0,r21 \get star'ssoreen ram size if so330EOR r4,r4,#RND(255) \EOR it550BPL loop2 \until done all 5340STRB r4,Er31 \store back6in screen ram560BL plot \replot stars		450MOV r1,#ns%-1 \no of stars mi
220FOR pass=0 TO 2 STEP 2460LDR r4,xpix% \no of pixels a230P%=code%cross screen in r4240E OPT pass470.loop2250.plot480MOV r2,r1,lsl#2 \r2 = r1*4260STMFD sp!,{lr}490LDR r3,Er0,r2] \get star's s270ADR r0,stars% \star table adcreen addrdr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars ms across screeninus 1 in r1500ADD r3,r3,r4 \add no pixel280MOV r2,r1,lsl#2 \r2 = r1*4500ADD r3,r3,r6 \subtract scr300MOV r2,r1,lsl#2 \r2 = r1*4510CMP r3,r5 \off bottom o300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300LDR r3,Er0,r2] \get star'sscreen?300EDR r4,r4,#RND(255) \EOR it530STR r3,Er0,r2] \store update300STR r4,r4,#RND(255) \EOR it550BPL loop2 \until done all 5340STRB r4,Er3] \store back0in screen ram560BL plot \replot stars	210DIM code% 1000	nus 1 in r1, star pointer
240C OPT pass470.loop2250.plot480MOV r2,r1,lsl#2 \r2 = r1*4260STMFD sp!,{lr}490LDR r3,[r0,r2] \get star's s270ADR r0,stars% \star table adcreen addrdr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars ms across screeninus 1 in r1500ADD r3,r3,r4 \add no pixel290,loop510CMP r3,r5 \off bottom o300MOV r2,r1,lsl#2 \r2 = r1*4510CMP r3,r6 \subtract scr300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300MOV r2,r1,lsl#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr300LDR r3,Er0,r2] \get star'sscreen?300LDR r4,r4,#RND(255) \EOR it530STR r3,Er0,r2] \store update300EOR r4,r4,#RND(255) \EOR it550BPL loop2 \until done all 5340STRB r4,Er3] \store back0in screen ram560BL plot \replot stars	220FOR pass=0 TO 2 STEP 2	460LDR r4, xpix% \no of pixels a
250.plot480MOV r2,r1,lsl#2 \r2 = r1*4260SIMFD sp!,{lr}490LDR r3,Lr0,r2] \get star's s270ADR r0,stars% \star table adcreen addrdr in r0500ADD r3,r3,r4 \add no pixel280MOV r1,#ns%-1 \no of stars m500ADD r3,r3,r4 \add no pixel290.loop510CMP r3,r5 \off bottom o300MOV r2,r1,ls1#2 \r2 = r1*4510CMP r3,r5 \off bottom o300MOV r2,r1,ls1#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr310LDR r3,Lr0,r2] \get star's520SUBGE r3,r3,r6 \subtract scr320LDRB r4,Lr3] \load byte f530STR r3,Lr0,r2] \store update330EOR r4,r4,#RND(255) \EOR it530BPL loop2 \until done all 5340STRB r4,Lr3] \store back0in screen ram560BL plot \replot stars	230P%=code%	
270HDR r0,stars%Star table ad creen addrdr in r0500ADD r3,r3,r4Nadd no pixel280MOV r1,#ns%-1s across screeninus 1 in r1510CMP r3,r5Noff bottom o290,loopf screen?520SUBGE r3,r3,r6Subtract scr300MOV r2,r1,ls1#2Nr2 = r1*4520SUBGE r3,r3,r6Subtract scr310LDR r3,Lr0,r2]Nget star'sseen ram size if so530STR r3,Lr0,r2]Nstore update320LDRB r4,Lr3]Nload byte fd address540SUBS r1,r1,#1Ndecrease star330EOR r4,r4,#RND(255)NEOR itStore back0340STRB r4,Lr3]Nstore back0560BLNeplot	240E OPT pass	470.loop2
270HDR r0,stars%Star table ad creen addrdr in r0500ADD r3,r3,r4Nadd no pixel280MOV r1,#ns%-1s across screeninus 1 in r1510CMP r3,r5Noff bottom o290,loopf screen?520SUBGE r3,r3,r6Subtract scr300MOV r2,r1,ls1#2Nr2 = r1*4520SUBGE r3,r3,r6Subtract scr310LDR r3,Lr0,r2]Nget star'sScreen address530STR r3,Lr0,r2]Nstore update320LDRB r4,Lr3]Nload byte fd address540SUBS r1,r1,#1Ndecrease star330EOR r4,r4,#RND(255)NEOR itStore back0340STRB r4,Lr3]Nstore back0560BLNeplot	250.plot	480MOV r2,r1,ls1#2 \r2 = r1*4
270HDR r0,stars%Star table ad creen addrdr in r0500ADD r3,r3,r4Nadd no pixel280MOV r1,#ns%-1s across screeninus 1 in r1510CMP r3,r5Noff bottom o290,loopf screen?520SUBGE r3,r3,r6Subtract scr300MOV r2,r1,ls1#2Nr2 = r1*4520SUBGE r3,r3,r6Subtract scr310LDR r3,Lr0,r2]Nget star'sScreen address530STR r3,Lr0,r2]Nstore update320LDRB r4,Lr3]Nload byte fd address540SUBS r1,r1,#1Ndecrease star330EOR r4,r4,#RND(255)NEOR itStore back0340STRB r4,Lr3]Nstore back0560BLNeplot	260STMFD_sp!,{1r}	490LDR r3,[r0,r2] \get star's s
280MOV r1,#ns%-1 \no of stars m inus 1 in r1s across screen290.loop510CMP r3,r5 \off bottom o f screen?300MOV r2,r1,ls1#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr screen address310LDR r3,Lr0,r2] \get star's screen addressscreen? 520SUBGE r3,r3,r6 \subtract scr een ram size if so 530STR r3,Lr0,r2] \store update d address320LDRB r4,Lr3] \load byte f rom screen ram 330EOR r4,r4,#RND(255) \EOR it with 255Store back 0 560BL plot \replot stars	2/UHDK r0,stars% \star table ad	
inus 1 in r1510CMP r3,r5Noff bottom o290,loopf screen?300MOV r2,r1,ls1#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr310LDR r3,Lr0,r2] \get star'sscreen address320LDRB r4,Lr3] \load byte f530STR r3,Lr0,r2] \store update320LDRB r4,Lr3] \load byte fd address330EOR r4,r4,#RND(255) \EOR it540SUBS r1,r1,#1 \decrease star340STRB r4,Lr3] \store back560BL plot \replot stars		500ADD r3,r3,r4 \add no pixel
290,loopf screen?300MOV r2,r1,ls1#2 \r2 = r1*4520SUBGE r3,r3,r6 \subtract scr310LDR r3,Lr0,r2] \get star'sseen ram size if soscreen address530STR r3,Lr0,r2] \store update320LDRB r4,Lr3] \load byte fd address330EOR r4,r4,#RND(255) \EOR it540SUBS r1,r1,#1 \decrease starwith 255340STRB r4,Lr3] \store back550BPL loop2 \until done all 5340STRB r4,Lr3] \store back0	280MOV r1,#ns%-1 \no of stars m	s across screen
300MOV r2,r1,ls1#2 \r2 = r1*4 310LDR r3,[r0,r2] \get star's screen address520SUBGE r3,r3,r6 \subtract scr een ram size if so 530STR r3,[r0,r2] \store update d address320LDRB r4,[r3] \load byte f rom screen ram 330EOR r4,r4,#RND(255) \EOR it with 255520SUBGE r3,r3,r6 \subtract scr een ram size if so 530STR r3,[r0,r2] \store update 540SUBS r1,r1,#1 \decrease star pointer 550BPL loop2 \until done all 5 0340STRB r4,[r3] \store back in screen ram\store back 560BL plot \replot stars	inus 1 in r1	
310LDR r3,[r0,r2]\get star's screen addresseen ram size if so 530STR r3,[r0,r2]\store update update320LDRB r4,[r3]\load byte f rom screen ramd address 540SUBS r1,r1,#1\decrease star pointer330EOR r4,r4,#RND(255)\EOR it yointerpointer550BPL loop2\until done all 5 S60BL plot340STRB r4,[r3]\store back in screen ram0	290,100p	t screen?
screen address 320LDRB r4,[r3] \load byte f rom screen ram 330EOR r4,r4,#RND(255) \EOR it with 255 340STRB r4,[r3] \store back in screen ram 530STR r3,[r0,r2] \store update 540SUBS r1,r1,#1 \decrease star 550BPL loop2 \until done all 5 6 560BL plot \replot stars	300MUV r2,r1,151#2 \r2 = r1*4	5205086E r3,r3,r6 \subtract scr
320LDRB r4,[r3]\load byte fd addressrom screen ram540SUBS r1,r1,#1\decrease star330EOR r4,r4,#RND(255)\EOR itpointerwith 255550BPL loop2\until done all 5340STRB r4,[r3]\store back0in screen ram560BL plot\replot stars	310LVK r3,Lr0,r21 \get star's	een ram size it so
rom screen ram 330EOR r4,r4,#RND(255) \EOR it with 255 340STRB r4,[r3] \store back in screen ram 560BL plot \replot stars	screen address	53051K r3,Lr0,r21 \store update
330EOR r4,r4,#RND(255) \EOR it with 255 340STRB r4,[r3] \store back 0 in screen ram 560BL plot \replot stars		d address
with 255	TOM SCREEN TAM	0405085 r1,r1,#1 \decrease star
340STRB r4,[r3] \store back 0 in screen ram 560BL plot \replot stars		FEADDI lass? Sustil dass all F
in screen ram 560BL plot \replot stars		
350SUBS r1,r1,#1 \decrease sta 570LDMFD sp!,{pc} \restore Basic		
JUST STOLEN STATE AND A STOLEN STRATES AND A STOLEN STOLEN STRATES AND A STOLEN STOLEN STRATES AND A STOLEN STATES AND A STOLEN STATES AND A STOLEN S		570LDMED on (not)meeting Pacing
Drag'N Drop Autumn 2016 dragdrop.co.uk facebook.com/dragdropmag Page 14		

Drag'N Drop Autumn 2016 | dragdrop.co.uk | facebook.com/dragdropmag | Page 14



The local water board has claimed to invent a revolutionary new pipe to replace existing ones but unfortunately the Tetraflurine Carbonadium making up the pipes has disintegrated and you have been employed to put all the pipes back before the houses are flooded!

Move the cursor around the screen with the Z, X, / and @ keys and press Return to lay a pipe section from the display in the top left corner.

If you fail to connect the pipe before the water reaches you then you lose the game!

The game features animated water, six types of pipe section and and a high score table.

Type in the Basic listing and save it before running, there is a short machine code section and if there are bugs it could crash your machine.

Save it on your hard disc and double click from the desktop to run.



Procedures

- *cycle* Cycle the colours, calls machine code routine to scan screen to change pixels of value D% to E%.
- *dead%* Did not manage to complete pipeline so display failure message
- delay Delay routine
- *completed* Game complete routine

game Main game proc

- hiscore Player has achieved hall
- of fame, invite to enter name *init* Sets up the sprites.
- *mc* Assemble machine code *scores* Display high score and instructions
- space Print message and await
- press of space bar
- sprite() Plot sprite number A% at
 (X%,Y%)
- *text()* Centre and Print multicoloured text

Variables

- *dead%* is TRUE if flooze seeps out of unconnected pipe section
- *dir* Current direction water is flowing
- fin% only TRUE when pipeline is

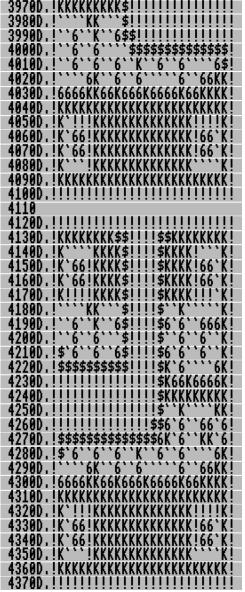
fx,fy Flooze co-ordinates fzi.fzc Flooze counters fza Flooze colours (Basic GCOL values) fzb Flooze colours (screen pixel colour values) Map of playing area arid() hi\$() High score names hi%() High scores *nextt* Next pipe section to be laid score% current score sheet% current screen number *tile* Current pipe section being

laid

complete

xpos,ypos player position





Python Primary School

pyth n

In classtime in the Summer 2016 edition of *Drag 'N Drop* we looked at a Python program **Temp/py** with routines to controll a temperature sensor connected to the Pi's GPIO (General Purpose Input/Output port) and examples of how to access RISC OS's SWI calls.

To complement **Temp/py** we'll look at second program in class today named **PyPressure/py** which uses a Freescale I2C



Precision Altimeter to read the air pressure. This chip runs on 3.3 volt and so

can be directly connected to the GPIO pins on the Pi.

I bought my MPL3115A2 Altitude/Pressure/Temp Sensor Breakout Board from Hobbytronics. You may need the ROOL wiki for the information on OS_IICOp and also the iic_transfer structure, they are not in the PRMs.

The application note AN4519 for the MPL3115A2 can be found at <u>http://www.freescale.com/</u> as well as the data sheet at <u>http://</u> <u>cache.nxp.com/files/sensors/doc/</u> <u>data_sheet/MPL3115A2.pdf</u> <u>?fsch=1&sr=10&pageNum=1</u>).

I originally used **swi.integers()** to read the block of data returned by OS_IICOp and another five lines to separate each byte, 22 lines in all.

I then remembered **swi.tuples()** and when I worked out how to use the list of tuples there were only 10 lines of code, a useful saving.

There are four lines of comments at the program start, then line 6 **imports time** as well as **swi**. A polling loop was used to start with but changed to **time.sleep()** to wait for the conversion to finish.

Line 7 creates an eight-byte block for the register reads in the global namespace so the **read_iic()** function as well as the rest of the program can access it. We only read six bytes here but blocks are defined in words, so two words equals eight bytes.

Lines 9 to 12 are the **read_iic()** function definition which takes three parameters:

• iic is the IIC address of the sensor

• register the address to start reading from

• **number** is how many reads to make.

Line 10 defines another block of one word and is initialised with the **register** variable, so we now have **dat.start** as a pointer to register.

Line 11 defines yet another block of six words, initialised as two **iic_transfer** structures, (address, start register, data length).

The first writes the register to start reading from passing one

Armcode for Beebsters

Having learnt machine code on the BBC Micro there were some aspects of Arm machine code with which I initially struggled.

So I thought I would put together a couple of articles (this issue of *Drag 'N Drop* and the next) to help people reared on 8bit 6502 assembly transfer their skills to writing 32-bit Armcode.

All versions of BBC Basic have a built-in assembler, something which was unique in Beeb days (contemporary micros just left programmers to "hand assemble" code and poke it into memory.) A typical assembly looks like:

DIM code 100	
FOR pass=0 TO 2 STEP	2
P%=code	
EOPT pass	
Need Street Stre	
<pre>\assembly code</pre>	
Ni i i	
JNEXT	

Memory is reserved with the DIM statement and P% is set to the address of *code* where the machine code begins.

The above framework is the same whether you are on 6502 or Armcode.

On the BBC Micro you could assemble code in odd corners of memory by setting P% directly:

FOR (pass=0 900	TO	2	STEP	2	
COPT	pass					

This is dodgy on RISC OS because there is no guarantee which part of the memory your program resides in.

PAGE is normally get &8F00 so you can sometimes get away with resetting PAGE to a higher value and assembling at &8F00 but isn't advisable.

You should always DIMension a block of memory on RISC OS to assemble code in.

6502 programs generally always end with an RTS to get back to Basic:

		FFEE
DIM	code	20

ĽDA #ASC"A":JSR oswrch ∖code for A RTS ∖return to BBC Basic

Before Jumping to the SubRoutine the 6502 processor pushes the return address (the address in the Program Counter) onto the stack. When the RTS is encountered the address pulled from the stack and put onto the Program Counter so execution continues where it left off.

At first sight there seems to be no equivalent of RTS in Armcode. How do you return to Basic after the following?

MOV R0,#ASC"A" \Ascii code for A SWI "OS_WriteC" \write it

In Armcode the Program Counter is held in Register 15. It's more commonly called "the PC" instead of "Register 15". Before a subroutine, a copy of the PC is put into Register 14 (R14). R14 is also called the link register or LR for short.

256 colour palette switching

One of the few drawbacks of RISC OS 5 compared to earlier versions of the operating system is that the hardware can only display Modes with 256 (or more) colours.

The lower colour depths (2-, 4or 16-colour Modes) available on RISC OS 2-4 were useful for animation. This was achieved by palette switching using VDU 19 but these Modes aren't available on the Pi.

Whilst VDU19 can still be used in 256 colour modes of the Pi the shade of the colours can only be subtly altered and not switched to another colour to achieve animated effects.

The trick I'll reveal in this article is to get a short machine code routine to scan the screen and change pixels of a specified colour to another.

It is necessary to understand the relationship between colours as Basic sees them and how the display hardware sees them.

There are 64 colours available

with the COLOUR and GCOL statements numbered 0 to 63. 64 to 127 are the same as 0 to 63 and 128 to 255 the same as 0-127 but set the text background in the COLOUR statement.

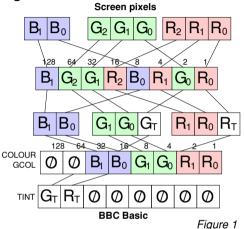
Bits 0-1 specify the amount of red, bits 2-3 the green and 4-5 yellow.

Number 15 (%001111, green plus red) is a bright yellow colour and the following program plots a yellow pixel at the top left hand corner of the screen.

CONTICI		5 3010				
10	REM Pr	ogram 3	1			
20	MODE 1	.3				
- 30 I	PRINT					
40 (GCOL 0	,3				
50 I	PLOT 6	9,0,10	923			
60 I	DIM T	8				
70	!T=149	!īT!4=-	-1			
80 \$	SYS"OS	_Read	∕duVar	ʻiab	les'	I,T,I
90	R=!T					
100	PRINT	POINT	(0.102	23)		
110	PRINT	?R				

The pixel is then read back using Basic's POINT statement which gives 15 as you would expect. However the byte stored in the screen ram (peeked in line 110) is different, it's 119.

What happens is the bits in the Basic colour are all repositioned in the screen colour as shown in Figure 1.



The 'tint' is a two bit value used by Basic to brighten or darken its 64 colours thereby achieving the 64*4=256 colours (although only 64 are available at one time)

As you can see it's a dog's breakfast but a study of our yellow pixel will make things clear. It starts off in Basic as:



There are four possible screen

Fibonacci Wallpaper

This short program shows off the power and speed of BBC Basic on the Raspberry Pi by generating patterned wallpaper on the screen.

It uses the Fibonacci sequence first documented by the 12th century Italian mathematician. Each number in the sequence is the sum of the previous two. If you reduce the result modulo 10 then the pattern repeats after a while.

This repetition is exploited by the program. The starting numbers are chosen at random between 1 and 9. These are indexes into a *table* of colours. *xw* and *yw* are the dimensions of the rectangle chosen at random but within the limits of the Mode 13 screen, 320 pixels across and 256 down.

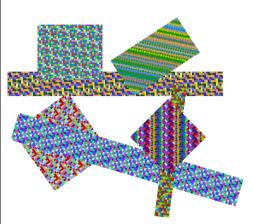
The address of the top left hand corner of screen ram is found by a SYStem call and stored in *R*. *S* is the offset from this address.

There are 320 pixels across

the Mode 13 screen, the rectangle may be smaller than this however so once a line of pixels of length *xw* is filled 320 is added to *S*.

If you see a design you like you can tap the S key to save it as a sprite to the current directory. It can be loaded into Paint and perhaps incorporated into scenery sprites for games.

FibWall listing
10REM Fibonacci Wallpaper
20REM By C.R.Dewhurst
30REM (c) Drag 'N Drop October
2016
40MODE 13:0FF
50DIM table 9
60!table=149:table!4=-1
70SYS "OS_ReadVduVariables",tab
le,table
80R=!table
90N=0
100REPEAT
110 CLS
120 xw=RND(320)
130 yw=RND(255)
140 S=0
150 FOR x=0 TO 9
160 x?table=RND(255)
170 NEXT
180 n1=RND(9)
190 n2=RND(9)



200 FOR Y=1 TO yw
210 FOR X=1 TO xw
220 S?R=n2?table
230 temp=n1
240 n1=n2
250 n2=(n2+temp)MOD10
260 S=S+1
270 NEXT
280 S=S+(320-xw)
290 NEXT
300 A=INKEY(200)
310 IF A=ASC"S" PROCsave
320UNTIL 0
330END
340:
350DEF PROCsave
360VDU24,0;1023-yw*4;xw*4;1023;
3/Va\$="screensave fib"+LEFI\$("VV
0",3-LENSTR\$N)+STR\$N
380N=N+1
3900SCLI a\$
400ENDPROC

Drag'N Drop Autumn 2016 | dragdrop.co.uk | facebook.com/dragdropmag | Page 39

MemAlloc module

This is a relocatable module originally dating from the early days of RISC OS which has been improved and updated to run on the Raspberry Pi.

It allows you to control various aspects of RISC OS – size of the Ram disc, system sprite area, etc. – from within programs by use of nine extra 'star' commands.

The listing assembles and saves the module. To install double click **MemAlloc**, press Ctrl+F12 and type ***Help MemAlloc** to list the commands and ***Help** *command* (where *command* is one of the commands provided) for further information.

MemAlloc listing

10REM MemAlloc 20REM Updated for 32 bit machin es 30REM by Drag 'N Drop, October 2016 40 50i1\$="If it is not possible to obtain this amount of memory, " 60i2\$="If it is not possible to do this, "

70o\$="the optional * command wi	390EQUS "MemAlloc "+CHR\$9+"0.2 "
11 be executed."+CHR\$13	+MID\$(TIME\$,5,11)+CHR\$0
80	400ALIGN
90DIM Q% 3000	410
100FOR 1%=0 TO 3 STEP 3	420\Command table
110P%=0	430.commands
1200%=0%	
	440FNassemblecommands
130COPT 1%+4	450EQUD 0 ;end of command table
140.header	460
150EQUD 0 ; Start offset	470\Code_for #SystemSize
160EQUD 0 ; Initialisation offse	480.code1
t	490STMFD sp!,{1r}
170EQUD 0 ; Finalisation offset	500MOV R2,#&0800
170EQUD 0 ; Finalisation offset 180EQUD 0 ; Service call handler	510BL getnum
offset	520MOV_R0.#0
190EQUD title ; Title string off	530BL doit
set.	540LDMFD sp!,{pc}
200EQUD help ; Help string offse	550
t.	560.help1
210EQUD commands ; Help and comm	570EQUS"*SystemSize allows you t
and keyword table	o set the system heap size in Kbyt
220EQUD 0	es, "
230EQUD 0	
240EQUD 0	580EQUS i1\$ 590EQUS o\$
250EQUD 0	600.syntax1
260	610EQOS"Syntax: *SystemSize <siz< th=""></siz<>
270\for 32 bit compatibility	e>_E<*command>]"+CHR\$0
280EQUD 0	620ALIGN
290EQUD_module_flags	630
300.module_flags EQUD 1	640\Code_for *RMASize
310	650.code2
320\Title string	660STMFD_sp!,{1r}
330.title	67AMOU R2.#&1AAA
330.title 340EQUS "MemAlloc"+CHR\$0	680BL getnum 690MOV R0,#1 700BL doit
350HL16N	690MOV ⁻ R0,#1
360	700BL doit
370\Help string	710LDMFD sp!,{pc}
380.help	720.help2
	k facebook.com/dragdropmag Page 39

Drag'N Drop Winter 2016 | <u>dragdrop.co.uk</u> | <u>facebook.com/dragdropmag</u> | Page 39

730FDUS"*RMASize allows you to s	1888 cuntavé	1430
730EQUS"*RMASize allows you to s et_the_RMA_size in Kbytes."	1090EQUS"Syntax: *SpriteSize <siz< td=""><td>1449\Cada for RMAEroo</td></siz<>	1449\Cada for RMAEroo
740EOUS il¢	e> [{*command}]"+CHR\$0	1450.code7
740EQUS i1\$ 750EQUS o\$_	1100ALIGN	1469STMED col {10}
760.syntax2	1110	1470MOH D2 #01000
770EOUCUCATANA +DMOCita /cita)	1120\Code for FontSize	1460STMFD sp!,{lr} 1470MOV R2,#&1000 1480BL getnum
770EQUS"Syntax: *RMASize <size></size>		14000L 90110M 14000TMED
[{*command}]"+CHR\$0	1130.code5 1140STMFD sp!,{lr} 1150MOV R2,#&0400 1160BL getnum 1170MOV R0.#4	147031111V SP:,\N1,N27
780ALIGN	114051MFV SP!, (1r)	15100MUV K0,#3
790	1130MUV KZ,#₩0400	13105W1 "US_MODUIE"
800\Code_for *ScreenSize	TIPART GELUNW	1320LVMFV SP!,(K0,K1)
810.code3	1170MUV KU,#4	1530SUBS R1,R1,R2
820STMFD_sp!,{lr} 830MOV_R2,#&01E0	1170MOV R0,#4 1180BL doit	1540BM1 exit
830MUV K2,#&01E0	1190LDMFD sp!.{pc}	1490STMFD sp!,{R1,R2} 1500MOV R0,#5 1510SWI "OS_Module" 1520LDMFD sp!,{R0,R1} 1530SUBS R1,R1,R2 1540BMI exit 1550BEQ exit 1550BEQ exit
840BL getnum 850MOV R0,#2	1200.help5	TOTODILLE A DEST (UDINT)
850MOV R0,#2	1210EQUS "*FontSize allows you to	15/0MUV KU.#1
860BL doit	set the font cache size in Rbytes	1580SWI "OS_ReadDynamicArea"
870LDMFD sp!,{pc}	. "	1590MOV R0,R1
880.help3	1220EQUS i1\$	1600LDMFD sp!,{R1,R2}
890EQUS"*ScreenSize allows you t	1230EQUS o\$	1610ADD R2,R2,R0
o set the screen memory size in Kb	1240.syntax5	1620MOV R0,#1
ytes. "	1250EQOS"Syntax: * FontSize <size></size>	15005W1 05_ReadDynamichrea 1590MOV R0,R1 1600LDMFD sp!,{R1,R2} 1610ADD R2,R2,R0 1620MOV R0,#1 1630BL doit
900EQUS i1\$	[{*command5]"+CHR\$0	1640.eXIT LVMFV SP!,\PC}
910EQUS o\$	1260ALIGN	1650.help7
920.syntax3	1260ALIGN 1270	1660EQUS "*RMAFree allows you to
930EQUS"Syntax: *ScreenSize <siz< td=""><td>1280\Code for RAMFSSize</td><td>set the free space in the RMA in K</td></siz<>	1280\Code for RAMFSSize	set the free space in the RMA in K
e> E<*command>]"+CHR\$0	1290.code6	bytes. "
940ALIGN	1300STMFD_sp!,{lr}	1670EQUS i2\$ 1680EQUS_o\$_
950	1310MOV R2,#&1000	1680EQUS o\$
960\Code_for_SpriteSize	1280\Code for RAMFSSize 1290.code6 1300STMFD sp!,{lr} 1310MOV R2,#&1000 1320BL getnum 1330MOV R0,#5 1340BL doit	1690.syntax7
970.code4	1330MOV ⁻ r0.#5	1700EQUS "Syntax: *RMAFree <free< td=""></free<>
980STMFD sp!.{lr}	1340BL doit	space> [{*command>]"+CHR\$0
980STMFD_sp!,{lr} 990MOV_R2,#&1000	1350LDMFD sp!,{pc}	1/10HL16N
1000BL getnum 1010MOV R0,#3	1360.help6	1720
1010MOV ⁻ R0.#3	1370EQUS "*RAMFSSize allows you t	1730\Code for SpriteFree
1020BL doit	o set the RAM disc size in Kbytes.	1740.code8
1030LDMFD sp!,{pc}		1750STMFD sp!.{lr}
1040.help4	1380EQUS i2\$	1750STMFD sp!,{lr} 1760MOV R2,#&1000
1050EQUS"*SpriteSize allows you t	1390EQUS o\$ 1400.syntax6 1410EQUS "Syntax: *RAMFSSize <siz< td=""><td>1770BL getnum</td></siz<>	1770BL getnum
1050EQUS"*SpriteSize allows you t o_set the system sprite size in Kb	1400.syntax6	1780STMFD sp!,{R1,R2}
ytes."	1410EQOS "Syntax: *RAMFSSize <siz< td=""><td>1790MOV R0,#8</td></siz<>	1790MOV R0,#8
1060EQUS i1\$	e> L{*command>J"+UHK\$0	1800SWI "US SpriteUp"
1070EQUS o\$	1420ALIGN	1810SUB R2, R2, R5
		k facebook.com/dragdronmag Page 40

Drag'N Drop Winter 2016 | dragdrop.co.uk | facebook.com/dragdropmag | Page 40

1820LDMFD sp!.{R0.R1} 1830SUBS R1,R1,R2 1840BMI exit2 1850BEQ exit2 1860STMFD_sp!,{R0,R1} 1870MOV R0.#3 1880SWI "OS_ReadDynamicArea" 1890MOV R0,R1 1900LDMFD sp!,{R1,R2} 1910ADD R2,R2,R0 1920MOV R0,#3 1930BL doit 1940.exit2 LDMFD sp!,{pc} 1950.help8 1960EQUS^{*} *SpriteFree allows you to set the free space in the syste m sprite area in Kbytes. " 1970EQUS i2\$ 1980EQUS o\$ 1990.suntax8 2000EQUS "Syntax: *SpriteFree <fr ee space> [{‡command>]"+CHR\$0 2010ALIGN 2020 2030\Code for FontFree 2040.code9 2050STMFD sp!,{lr} 2060MOV R2,#&400 2070BL getnum 2080STMFD_se!,{R1,R2} 2090MOV R0,#0 2100SWI "Font_CacheAddr" 2110SUB R2,R2,R3 2120LDMFD sp!,{R0,R1} 2130SUBS R1,R1,R2 2140BMI exit3 2150BEQ exit3 2160STMFD_sp!,{R0,R1} 2170MOV R0,#4 2180SWI "OS_ReadDynamicArea" 2190MOV R0,R1 2200LDMFD sp!,{R1,R2}

2210ADD R2.R2.R0 2220MOV R0.#4 2230BL doit 2240.exit3 LDMFD sp!.{pc} 2250.help9 2260EQUS'"*FontFree allows you to set the free space in the font ca che in Kbutes. 2270EQUS 12\$ 2280EQUS o\$ 2290.syntax9 2300EQUS "Syntax: *FontFree <free space> [{*command>]"+CHR\$0 2310ALIGN 2320 2330.getnum 2340STMFD_sp!,{lr} 2350MOV R1.R0 2360MOV R0.#\$20000000 2370SWI "OS ReadUnsigned" 2380MOV R2, R2, LSL #10 2390.loop (DRB R0, CR1, #0] 2400CMP R0, #0 2410CMPNE R0,#10 2420CMPNE R0,#13 2430MOVEQ R1,#0 2440LDMEQFD sp!,{pc} 2450CMP R0,#32 2460ADDNE R1, R1, #1 2470BNE loop 2480.loop2 LDRB R0,ER1,#0] 2490CMP R0,#0 2500CMPNE R0,#10 2510CMPNE R0,#13 2520MOVEQ R1,#0 2530LDMEQFD sp!,{pc} 2540CMP R0,#32 2550ADDEQ R1.R1.#1 2560BEQ 100P2 2570LDMFD sp!,{pc} 2580 2590.doit

2600STMFD sp!,{R0-R2,lr} 2610SWI_"OS_ReadDynamicArea" 26105W1 "US_KeadDynamicHrea" 2620LDMFD sp!,{R0,R3} 2630LDMFD sp!,{R2,1r} 2640RSB R1,R1,R2 2650STMFD sp!,{R3,1r} 2660SWI "XOS_ChangeDynamicArea" 2670LDMFD sp!,{R0,1r} 2680MOVVC pc,1r 2690STMFD sp!,{Ir} 2700TED po #0 2700TEQ R0,#0 2710SWINE "OS_CLI" 2720LDMFD sp!,{pc} 2730 2740] 2750NEXT 2760 2770a\$="#SAVE MemAlloc "+STR\$"Q%+ " +"+STR\$"P% 2780PRINT a\$ 27900SCLI a\$ 2800*settype memalloc module 2810END 2820 2830DEF FNassemblecommands 2840RESTORE 2850FOR com%=1 TO 9 2860READ command\$ 2870COPT 1%+4 2880EQUS command\$+CHR\$0 2890ALIGN 2900EQUD_EVAL("code"+STR\$com%) 2910EQUD &FF0001 2920EQUD_EVAL("syntax"+STR\$com%) 2930EQUD EVAL("help"+STR\$com%) 2940] 2950NEXT 2960=0 2970 2980DATA SystemSize,RMASize,Scree nSize,SpriteSize,FontSize,RAMFSSiz e,RMAFree,SpriteFree,FontFree

Drag'N Drop Winter 2016 | dragdrop.co.uk | facebook.com/dragdropmag | Page 41



Usually on RISC OS you have to click somewhere in a desktop window for it to gain the input focus. This short machine code utility automatically focuses the input on the window under the pointer.

The listing assembles and saves a file called AF. Double click AF to install. To quit, right click on the Raspberry icon, click menu over Autofocus in the Application tasks section and choose Quit.

One easy improvement you can make is to package up the AF file inside an application with an application sprite, !Boot and !Run file.

Aı	utofoc			g		
	10REM	Autof	OCUS			
	20REM	Autor	natica	ally	focuse	s inj
ut						_
	30REM	on wi	ndow	unde	er poin October	ter.
	<u>40REM</u>	Drag	'N Dr	°op ()ctober	2010
	50					
	60DIM	code	200			
	70FOR	pass=	0 TO	3 \$1	EP 3	
	80P%=0					
	900%=0	ode 👘				

100EOPT pass+4	
110	
ĪŽŌMOV R0,#200	
130LDR R1,task	
140HDK KZ,name	
1505WL "Wimp_initialise"	
160	
170.loop	
180MOV R0,#0	
190ADR R1,block	
200SWI "Wimp_Poll"	
210TEQ_R0,#17	
220TEQNE R0,#18	
230BNE skip	
240LDR R0,[R1,#16]	
250TEQ R0,#0	
260BEQ exit	
270.skip	
270.skip 2805WI "Wimp_GetPointerInfo"	
270LVN N0.LN1.#12J	
300LDR R2,temp	
310TEQ K0.K2	
320BEQ 100P	
330STR R0,temp 340MOV R8,R1	
340MOV R8,R1	
350MVN R1,#0	
360MOV R2,#0	
370MOV R3,#0	
380MVN R4;#0	
390MVN R5,#0 400SWI "Wimp_SetCaretPosition" 410M0U R1 R8	
400SWI "Wimp_SetCaretPosition"	
TIONOV NIJNO	
420B 100P	
430	
440.exit	
450SWI "Wimp_CloseDown" 460MOVS PC,RI4_	
460MUVS PC, K14	
470.temp EQUD 0	

480
490.task EQUS "TASK"
500.name EQUS "Autofocus"+CHR\$0:
ALIGN
510.block EQUS STRING\$(32,CHR\$0)
520
530]
540NEXT
550a\$="*SAVE AF "+STR\$"code+" +"
+STR\$~P%
560PRINT a\$
5700SCLI a\$
580*settype af absolute

Line 60 reserve 200 bytes of memory to assemble code into.

Lines 70-100 assemble code at O% but as if to run at P% by using OPT with bit 3 set.

Lines 120-150 is the equivalent of SYS "Wimp_Initialise",200, "TASK","Autofocus".

Lines 180-200 call Wimp_Poll and test for reason codes 17,18 (message).

Lines 280-410 obtain position of pointer and mouse button state, offset 12 is window handle under pointer. If same as last time round loop skip. Otherwise set up parameter block and call SWI to focus the input.

Drag'N Drop Autumn 2016 | dragdrop.co.uk | facebook.com/dragdropmag | Page 42