Completing Worron Diary

By Richard Paynter

Worron was a Commodore 64 game that I wrote as a teenager in 1986/87 and tried to sell to several companies before signing an advance with Firebird Software in 1987.

At one point, I remember tracking down the Darling brothers (of Codemasters) at their family barn in Banbury on a weekend, much to their annoyance, thanks to my badgering of someone on reception who mistakingly gave me their private phone number. They were not interested, but Firebird were. I'd met Colin Fuidge of Firebird in 1985 when I'd won a 24 hour Soft-Aid Elite-a-thon, and having remembered his name, had taken myself up to their New Oxford St offices and as a result, they sent me a contract and I received a small advance on royalties that I used to buy my first music equipment.

The game was mostly complete, a bit buggy with no sound and was never released. It was very hard to play and I'd be amazed if anyone could ever complete it without cheating.

I thought the game was lost until https://www.gamesthatwerent.com resurrected it out of nowhere when some old disks were found in Darren Melbourne's (of Paranoid Software) possession. I had taken the disks up to their Beckenham offices and showed them the game and must have left a copy there, thankfully.

Now in 2021, I'm going to de-compile the .prg file and try to re-build some source code in java and attempt to complete the game, fixing any outstanding bugs, re-enabling some of the disabled features and adding sound and music.

To do this, I'll use some java code I've developed for another new game I've working on. I'm a java developer in my day job, so I'd written a 6502 development environment that allows me to essentially write 6502 in java, compile it, render the output to a T64 tape file and spin up a VICE emulator all in a few seconds, using Intellij as my IDE. Rather an improvement on Machine Lightning and having to write Worron on the same Commodore 64 machine that I play-tested it on; compile/test cycles would last about 25 mins!

I suspect the memory layout was pretty simple. I'll not be able to reclaim the variable or label names, but I'm sure I can generate temporary ones for those and figure out what they meant by looking at the code and gradually adding context back into the source. The original names were filled with variable names that were mostly just swear words, so probably didn't mean much as I tended to use a lot of Derek & Clive phrases rather than what the variable actually did.

Unfortunately, I don't have my original disks, so whoever ripped this from the Darren Melbourne disks might have made their own changes. Hopefully those will be minimum. I do have several versions, one of which appears to have been 'trained' and another that has my "Firebird Presents" changed to some hacker name. I'm hoping that "worron.prg" is as close to my original source code as possible. (It was)

The following is my diary of this process.

Richard Paynter, Soho, May 2021

27/03/2021

I've started writing a java PrgDecompiler to take "worron.prg" and start to make sense of it. It's a simple format where the first 2x bytes are the lsb/msb of the start address and the rest of the file is the data. I can already see that data runs from:

2049 25
2050 8
2051 194
2052 7
…
40948 150
40949 160
40950 243
40951 95
40952 93
40953 93

Which makes sense as I had no idea how to switch off ROMs back then so won't have crept into the kernal ROM area. I was trying to write the source code and keep the game in the same memory, which was a nightmare.

I had no idea that professionals developed on a machine other than the Commodore 64.

So, given we start at 2049, this is probably a basic bootstrap. I use a similar mechanism in other games. Let's decode that.

```
array("BasicBootstrap",
        0x0f,
        0x08,
        0xcf,
        0x07,
        0x9e,
        PETASCII.asciiToPetscii('2'),
        PETASCII.asciiToPetscii('0'),
        PETASCII.asciiToPetscii('6'),
        PETASCII.asciiToPetscii('5'),
        0x20,
        0x41,
        0x42,
        0x43,
        0x00,
        0x00,
        0x00
);
```

In fact, it is exactly the same address as my new game, which is handy.

I don't recall being able to do this kind of bootstrapping back in the day, so this has most likely been added by the hackers or possibly by Machine Lightening, which I think is the name of the assembler I used.

I think that 2075 is the start address of the code proper:

2075 120 78 opcodeMode:SEI addressMode:Implied
2076 169 a9 opcodeMode:LDA addressMode:Immediate
2077 47 2f opcodeMode:null addressMode:null
2078 133 85 opcodeMode:STA addressMode:ZeroPage
2079 0 00 opcodeMode:BRK addressMode:Implied

This looks like it turns off interrupts and changes something in address 0 which affects which ROMs are loaded in, which probably turns off basic (we'll check that). I must have known how to do that at the very least.

To double check, I can look for 2075 in 4x bytes as petscii:

50
48
55
53

And there it is:

2054 50 32 opcodeMode:null addressMode:null
2055 48 30 opcodeMode:BMI addressMode:Relative
2056 55 37 opcodeMode:null addressMode:null
2057 53 35 opcodeMode:AND addressMode:ZeroPageX

Bingo! So - 2075 is the start… let's skip the stuff before.

So early debugging shows code that I don't recognise… even now I don't use indirect addressing, and I'm certain I never put code in the zero page, but maybe I'm mistaken. If I stop in VICE, then I can see that the title screen runs at 0cb6 which is 3254 - maybe it start at 3072?

I mean what if the .prg file contains a decompression routine + the compressed data? That's not really of much use to me. Really I need to run the game and then dump the memory and perhaps start again after the hackers code has left the building?

Just remembered that I can extract the full memory by doing a m 0000 ffff in the debugger, so I should do that:

Yay. Done that. I now have a text file with the full memory contents, so can now start to pick this apart.

So I've written a java parser for this format and can now see the instructions in memory so time to sort through the mess and make sense of it.

We're going to convert it all into java code, then completely re-compile and run the program from scratch, thus confirming that we've captured everything.

When running the title screen, there appears to be a loop here:


```
.C:0ca0  AD 00 DC    LDA $DC00
.C:0ca3  29 10       AND #$10
.C:0ca5  C9 10       CMP #$10
.C:0ca7  F0 0A       BEQ $0CB3
.C:0ca9  AD 5D 08    LDA $085D
.C:0cac  C9 00       CMP #$00
.C:0cae  D0 03       BNE $0CB3
.C:0cb0  4C 88 13    JMP $1388
.C:0cb3  AD 92 09    LDA $0992
.C:0cb6  C9 03       CMP #$03
.C:0cb8  F0 07       BEQ $0CC1
.C:0cba  C9 06       CMP #$06
.C:0cbc  F0 18       BEQ $0CD6
.C:0cbe  4C A0 0C    JMP $0CA0
```

This is address 3232

This is the de-compiled code:

0ca0 LDA 56320
0ca3 AND #16
0ca5 CMP #16
0ca7 BEQ 10
0ca9 LDA 2141
0cac CMP #0

0cae BNE 3
0cb0 JMP 5000
0cb3 LDA 2450
0cb6 CMP #3
0cb8 BEQ 7
0cba CMP #6
0cbc BEQ 24
0cbe JMP 3232

I think the first thing we can do is to make use of some better labels, so let's start adding the vic/cia/sid labels as at least we know what these are. Then we can iterate and see what hangs off that and label them accordingly. This is like de-coding Enigma, only massively less important!

28/03/2021

I feel like I need to understand the code a bit better in the monitor before I start ripping it. For example, where is the code for the title raster routines?

If I watch where the border colour is changed, I can see we stop here:

```
.C:0d06  BD A8 0A    LDA $0AA8,X
.C:0d09  8D 21 D0    STA $D021
.C:0d0c  BD 84 0A    LDA $0A84,X
.C:0d0f  8D 12 D0    STA $D012
.C:0d12  EE 83 0A    INC $0A83
.C:0d15  AD 83 0A    LDA $0A83
.C:0d18  C9 24       CMP #$24
.C:0d1a  D0 0D       BNE $0D29
.C:0d1c  AD 11 D0    LDA $D011
.C:0d1f  29 F7       AND #$F7
.C:0d21  8D 11 D0    STA $D011
.C:0d24  A9 00       LDA #$00
.C:0d26  8D 83 0A    STA $0A83
.C:0d29  AD 83 0A    LDA $0A83
.C:0d2c  C9 23       CMP #$23
.C:0d2e  F0 13       BEQ $0D43
```

0d06= 3334

We should look for RTI to see the end of the raster routine.

This code in java looks like:

```
3166  $0c5e: LDA 2763,X                 189 203 10    bd cb 0a
3169  $0c61: STA 53280                  141 32 208    8d 20 d0
3172  $0c64: LDA 2728,X                 189 168 10    bd a8 0a
3175  $0c67: STA 53281                  141 33 208    8d 21 d0
3178  $0c6a: LDA 2692,X                 189 132 10    bd 84 0a
3181  $0c6d: STA 53266                  141 18 208    8d 12 d0
3184  $0c70: INC 2691                   238 131 10    ee 83 0a
3187  $0c73: LDA 2691                   173 131 10    ad 83 0a
```

```
3190  $0c76: CMP #36                          201 36       c9 24
3192  $0c78: BNE 13                           208 13       d0 0d
```

I've got something wrong in the de-compile as the addresses don't line up… let's look at that next.

It seems to me that SEI is going to be an early instruction as we set everything up, so let's search for where that might be.

SEI = 0x78

This is the first one that looks like it's something worth investigating further:

```
(C:$0aa5) d 0c0d
.C:0c0d  78          SEI
.C:0c0e  A9 F0       LDA #$F0
.C:0c10  8D 14 03    STA $0314
.C:0c13  A9 0C       LDA #$0C
.C:0c15  8D 15 03    STA $0315
.C:0c18  AD 84 0A    LDA $0A84
.C:0c1b  8D 12 D0    STA $D012
.C:0c1e  AD 11 D0    LDA $D011
.C:0c21  29 7F       AND #$7F
.C:0c23  8D 11 D0    STA $D011
.C:0c26  58          CLI
.C:0c27  A9 81       LDA #$81
.C:0c29  8D 1A D0    STA $D01A
.C:0c2c  AD 0E DC    LDA $DC0E
.C:0c2f  29 FE       AND #$FE
.C:0c31  8D 0E DC    STA $DC0E
.C:0c34  A9 9B       LDA #$9B
```

0c0d = 3085

Let's work backwards from there until we find garbage, see if we can't find the start of my code.

Going back a few bytes, this still looks like it could be valid code:

```
(C:$0c1b) d 0bf1
.C:0bf1  A9 01       LDA #$01
.C:0bf3  8D 7A D8    STA $D87A
.C:0bf6  8D 7B D8    STA $D87B
.C:0bf9  8D 9C D8    STA $D89C
.C:0bfc  8D 9D D8    STA $D89D
.C:0bff  A9 07       LDA #$07
.C:0c01  8D A2 D8    STA $D8A2
.C:0c04  8D A3 D8    STA $D8A3
.C:0c07  8D C4 D8    STA $D8C4
.C:0c0a  8D C5 D8    STA $D8C5
.C:0c0d  78          SEI
.C:0c0e  A9 F0       LDA #$F0
```

```
.C:0c10  8D 14 03    STA $0314
.C:0c13  A9 0C       LDA #$0C
.C:0c15  8D 15 03    STA $0315
.C:0c18  AD 84 0A    LDA $0A84
```

Ok - this looks the most likely as the easiest part of the code:

```
(C:$0b2e) d 0b05
.C:0b05  AD 3C 08    LDA $083C
.C:0b08  C9 00       CMP #$00
.C:0b0a  F0 08       BEQ $0B14
.C:0b0c  A9 00       LDA #$00
.C:0b0e  8D 3C 08    STA $083C
.C:0b11  8D 40 03    STA $0340
.C:0b14  AD 16 D0    LDA $D016
.C:0b17  29 F7       AND #$F7
.C:0b19  8D 16 D0    STA $D016
.C:0b1c  AD 02 DD    LDA $DD02
.C:0b1f  09 03       ORA #$03
.C:0b21  8D 02 DD    STA $DD02
.C:0b24  AD 00 DD    LDA $DD00
.C:0b27  29 FC       AND #$FC
.C:0b29  09 02       ORA #$02
.C:0b2b  8D 00 DD    STA $DD00
```

0b05 = 2821

BINGO!

Goto 2821 in the monitor re-starts the title screen:

I can run this multiple times so clearly I'm clearing all the variables etc. so that's good.

I now know where the game starts from

So - given that knowledge, what's the next step? I wonder if it's creating a new tape image that is able to build from that address. Right now, I'll need to include all of memory until I can start to remove bits that I know are not used.

We need something like:

```
final StaticWord basicAddress = word(2049);
new BaseAddress(assembler, basicAddress, "BasicBootstrap");

final StaticWord sysAddress = word(2065);
final String sysAddressAsString = sysAddress.toString();
assertEquals(sysAddressAsString.length(),4);

array("BasicBootstrap",
        0x0f,
        0x08,
        0xcf,
        0x07,
        0x9e,
        asciiToPetscii(sysAddressAsString.charAt(0)),
```

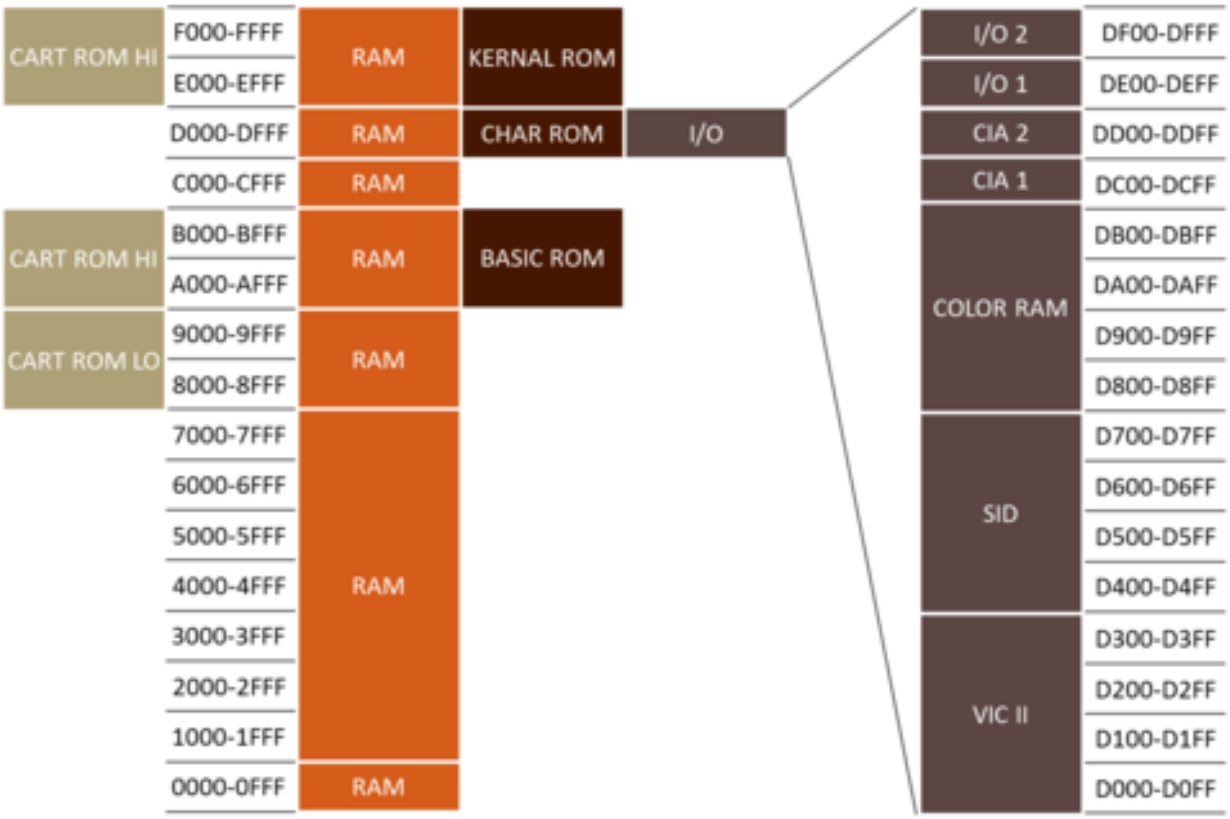
```
        asciiToPetscii(sysAddressAsString.charAt(1)),
        asciiToPetscii(sysAddressAsString.charAt(2)),
        asciiToPetscii(sysAddressAsString.charAt(3)),
        0x20,
        0x41,
        0x42,
        0x43,
        0x00,
        0x00,
        0x00
);

new BaseAddress(assembler, sysAddress, "Sys");
```

Well, this is a start - I've managed to create a new t64 image… the bootstrap is not working, but if I type sys 2061 the game starts

Getting somewhere as I've managed to start up the high score screen from java. But I'm missing data before 2049 so there are probably some variables lower down in memory that I'm not aware of.

Excellent, I've now included data from 512+ and the game starts up correctly. I can now start to try to understand what makes up the memory, perhaps by clearing areas of the 64k that I think are not being used, such as the chip area.

By looking at the C64 memory map:



I know that I didn't know how to switch off ROM back then, so I've cleared all the memory under the 3x ROM area and the game still works.

I know that 0xC000 49152 was a favourite area of mine to start coding from, so I should check what is in there by looking at the monitor.

There is code here, but it doesn't look like my code as it uses addressing modes I didn't even understand:

```
(C:$0ccf) d c000
.C:c000  A0 FF       LDY #$FF
.C:c002  C8          INY
.C:c003  B1 09       LDA ($09),Y
.C:c005  D0 FB       BNE $C002
.C:c007  98          TYA
.C:c008  A6 09       LDX $09
.C:c00a  A4 0A       LDY $0A
.C:c00c  20 55 A6    JSR $A655
.C:c00f  20 74 BF    JSR $BF74
.C:c012  20 EB A5    JSR $A5EB
.C:c015  A0 03       LDY #$03
.C:c017  20 54 AA    JSR $AA54
.C:c01a  85 26       STA $26
.C:c01c  20 54 AA    JSR $AA54
.C:c01f  85 27       STA $27
.C:c021  20 B7 FF    JSR $FFB7
```

```
.C:c024  29 F0        AND #$F0
.C:c026  D0 B4        BNE $BFDC
.C:c028  88           DEY
```

I didn't use indirect Y addressing, so let's try zero'ing this area too.

No - if I do that, I get corruption:

```
// Clear the RAM area at 49152 as this doesn't look like my code, so probably some hacker code
for (int i = 0xc000; i < 0xd000; i++) {
  data.set(i, StaticCByte.value(i % 256));
}
```

This must be the screen definition area. So not code. A good discovery!

29/03/2021

The next thing to investigate is where the sprites are located in memory. They are likely to be contiguous, although I can't remember what tool I used to define them.

Two of the title screen sprites are here:

6200 (25088)
6240 (25152)

Let’s try clearing one of these areas and see if that’s the case.

Doing that produces:

Where do the sprites begin? Back in 87 I would not have paid attention to page boundaries and would have used decimal boundaries by preference.

As it happens:
=25088-(64*17) = 24000 so that seems like a likely place for me to have started putting my sprites.

Actually - looking at the debugger, 6200 is the first sprite and they run contiguously, all the way up to

25088
28800

That's 3712 bytes or 58 sprite definitions.

So I must have set bank 1 rather than the default bank 0 for graphics, which explains why when I did a soft reset, there was still text at 1024 as this was never overwritten by the game. This means I need to look for screen memory in 16384-32768 range.

I've also found the scrolling message, which starts at:

4990 (18832)
5090 (20624)

= 1792 bytes or 7 pages exactly

So looking at the code from 2821+, I think we have code here:

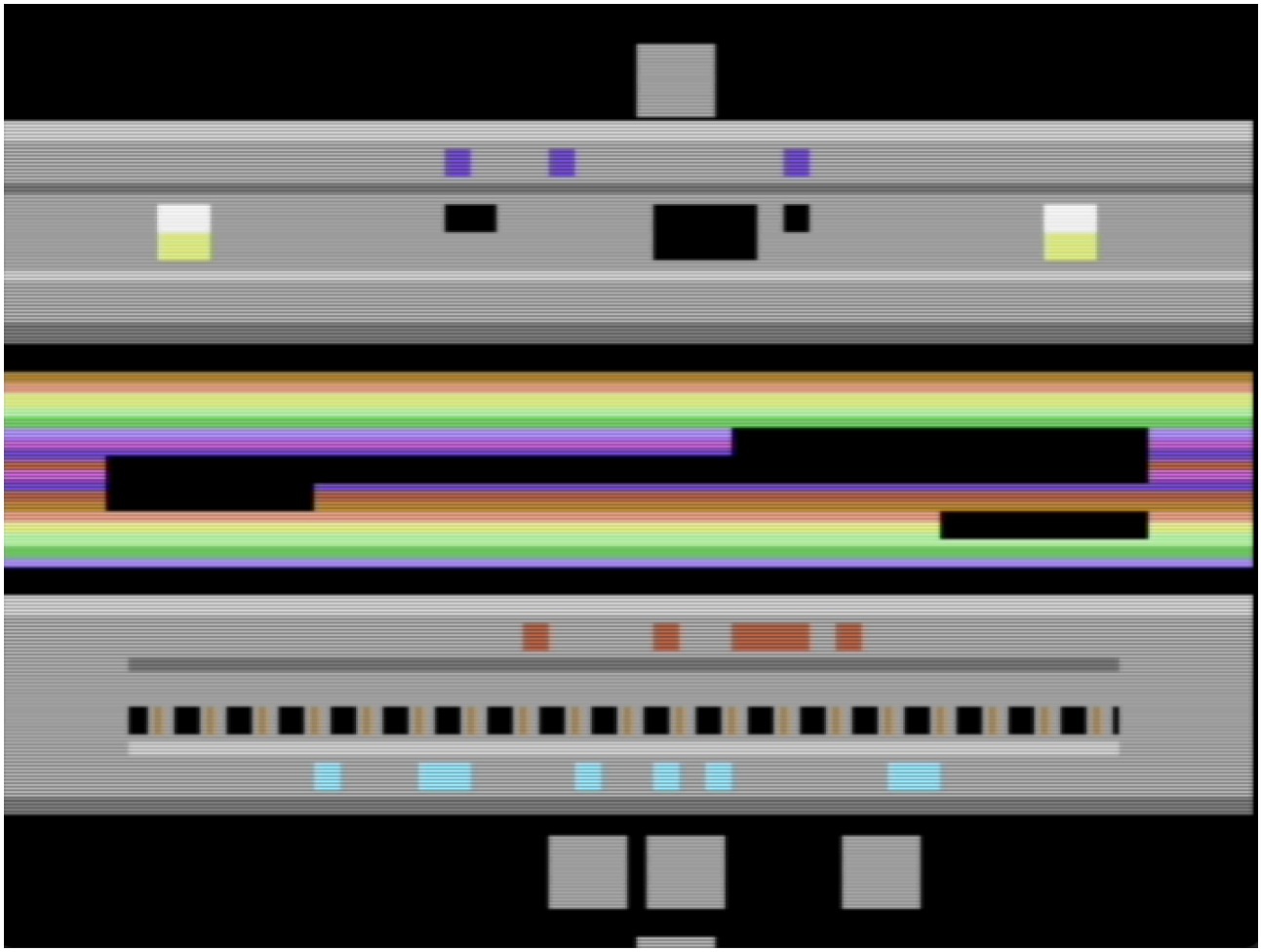
2821-4826 (code - possibly title screen)
4827-5000 (empty)
5000-5018 (some very short routine)
5021-6338 (data or blank?) - this is some kind of data since if I blank it, things change
6338-12964 (code)
12766-18234 (empty)
18234- (data?)

Let's clear the possible data areas to confirm.

30/03/2021

Where is the character set stored? We know it should be in bank 1, from 16384 to 32768, but where? We'll look for that today.

Meanwhile, I've experimented by only writing out the data from 2821-4826, which I think is the code for the title screen. When I do that, this happens:



Everything works, but I have no graphics or scrolling message data, which tells me that this is indeed the code, but the data - as we know - is elsewhere.

Doesn't crash though, so that's a good start.

When I hit fire, it does - unsurprisingly.

I think the next thing to do is generate instructions for this area of code and compile it. We can then diff it against what I loaded and make sure we're producing the same code/data.

I've managed to do that. The instructions I create can be compiled into exactly the same data. In fact, I've overwritten that data with the compiled code and all works, so in theory this is the first time I've re-compiled the game since 1987, which is fun.

The next thing to do is to try to generate the source code as java and see if I can get that to compile.

So I've now generated some basic java code for the title screen:

```
final class TitleScreenCode extends Code {
  public TitleScreenCode(final Assembler assembler) {
    super(assembler);

    LDA_Absolute(2108);
    CMP_Immediate(0);
    BEQ_Relative(8);
    LDA_Immediate(0);
    STA_Absolute(2108);
    STA_Absolute(832);
    LDA_Absolute(53270);
    AND_Immediate(247);
    STA_Absolute(53270);
    LDA_Absolute(56578);
    ORA_Immediate(3);
    STA_Absolute(56578);
    LDA_Absolute(56576);
    AND_Immediate(252);
    ORA_Immediate(2);
    STA_Absolute(56576);
    LDA_Absolute(53272);
    AND_Immediate(15);
    ORA_Immediate(208);
    AND_Immediate(240);
    ORA_Immediate(14);
    STA_Absolute(53272);
    LDA_Absolute(53270);
    ORA_Immediate(8);
    STA_Absolute(53270);
```

Let's see if I can compile this and whether it runs.

Great - I've compiled this code and it produces the same data.

I've now captured some of the Worron source code in java. Funny.

The next thing I need to do is generate a label map, so that I can refer to addresses by name. I'll generate labels such as "LABEL16384" based off the address and we can rename them later as it becomes clear what they do.

I think we'll just generate the code like above and we'll manually go through and figure out what the address mean and assign labels etc. We can just do a replace in java.

That said, I think that the JMP/JSR/Branch need to be automatically calculated, so maybe we shouldn't do that. Let's assign a label to each address.

31/03/2021

There are some Kernal ROM routines that I seem to be calling, so let's annotate those better:

So I've started generating the labels, but some of the labels are for half-way through an instruction, which implies self-modifying code, which I'm rather surprised I knew how do do back then…

We need to generate a label for each line of code and choose the label closest to the instruction address and add an offset.

STA_Absolute(LABEL3574);

```
3557  $0de5: STA 3574                    141 246 13     8d f6 0d
3560  $0de8: LDA 2240,X                   189 192 8      bd c0 08
3563  $0deb: STA 3573                    141 245 13     8d f5 0d
3566  $0dee: LDX 2688                    174 128 10     ae 80 0a
3569  $0df1: STX 2563                    142 3 10       8e 03 0a
3572  $0df4: LDA 18832,X                   189 144 73     bd 90 49
3575  $0df7: TAX                         170            aa
```

Ok - got that working:

```
STA_Absolute(reference(LABEL3572).add(2));
LDA_AbsoluteX(LABEL2240);
STA_Absolute(reference(LABEL3572).add(1));
```

The next thing is relative instructions.

BNE_Relative(108);

These need to point at labels. So I need to add the offset to the address and come up with a label.

Let's try to generate the routine at 5000 so that we have 2x routines.

Yay - I've now managed to combine 2x separate regions of code:

```
BNE_Relative(LABEL4781);
RTS_Implied();
new BaseAddress(assembler, word(5000), "SomeRoutine");
```

```
label(LABEL5000);
LDA_Immediate(0);
STA_Absolute(VICSpriteEnableRegister);
LDA_Absolute(VICScreenControlRegister1);
AND_Immediate(239);
STA_Absolute(VICScreenControlRegister1);
LDA_Immediate(1);
STA_Absolute(LABEL5125);
JMP_Absolute(LABEL6536);
```

And all works. Now to find the main region of code, where the main game is.

In order to find out where the code is, let's capture all the jump addresses, such as JMP/ JSR/Branch

This should give us a good sense of where we have code and whether we've found it all.

Here they are:

jumpAddresses:
- 2100
- 2836
- 2978
- 2996
- 3023
- 3041
- 3176
- 3232
- 3251
- 3265
- 3267
- 3286
- 3288
- 3322
- 3369
- 3392
- 3395
- 3406
- 3417
- 3420
- 3423
- 3440
- 3453
- 3473
- 3493
- 3531
- 3532
- 3542
- 3572
- 3613
- 3637

- 3654
- 3657
- 3680
- 3721
- 3758
- 3779
- 3789
- 3790
- 3832
- 3835
- 3856
- 3882
- 3897
- 3913
- 3948
- 3985
- 4018
- 4063
- 4137
- 4163
- 4202
- 4217
- 4220
- 4243
- 4256
- 4272
- 4349
- 4369
- 4389
- 4409
- 4439
- 4451
- 4479
- 4569
- 4648
- 4689
- 4697
- 4710
- 4781
- 4806
- 5000
- 6536
- 6582
- 6592
- 6711
- 6731
- 6815
- 6858
- 6870
- 6878
- 6934
- 7024

- 7026
- 7044
- 7100
- 7110
- 7126
- 7165
- 7204
- 7219
- 7228
- 7231
- 7248
- 7251
- 7319
- 7322
- 7325
- 7328
- 7331
- 7334
- 7379
- 7394
- 7398
- 7413
- 7417
- 7432
- 7436
- 7471
- 7474
- 7478
- 7513
- 7516
- 7520
- 7535
- 7539
- 7574
- 7577
- 7581
- 7616
- 7619
- 7622
- 7654
- 7655
- 7671
- 7703
- 7704
- 7738
- 7748
- 7749
- 7759
- 7786
- 7796
- 7797
- 7818

- 7861
- 7885
- 7918
- 7947
- 7950
- 7963
- 8155
- 8166
- 8175
- 8184
- 8193
- 8202
- 8211
- 8220
- 8229
- 8233
- 8268
- 8286
- 8302
- 8326
- 8338
- 8357
- 8429
- 8483
- 8486
- 8499
- 8508
- 8529
- 8550
- 8577
- 8604
- 8631
- 8664
- 8678
- 8711
- 8732
- 8753
- 8774
- 8795
- 8816
- 8837
- 8869
- 8893
- 8932
- 8966
- 8969
- 9009
- 9034
- 9058
- 9083
- 9096
- 9118

- 9148
- 9175
- 9202
- 9226
- 9247
- 9268
- 9295
- 9310
- 9334
- 9367
- 9391
- 9418
- 9442
- 9469
- 9486
- 9488
- 9505
- 9507
- 9525
- 9526
- 9551
- 9552
- 9578
- 9592
- 9608
- 9634
- 9641
- 9648
- 9655
- 9662
- 9669
- 9676
- 9683
- 9690
- 9693
- 9719
- 9722
- 9765
- 9791
- 9802
- 9836
- 9866
- 9880
- 9891
- 9909
- 9916
- 9923
- 9930
- 9944
- 9958
- 9972
- 9986

- 10000
- 10014
- 10028
- 10042
- 10043
- 10052
- 10076
- 10099
- 10118
- 10122
- 10138
- 10158
- 10168
- 10170
- 10216
- 10246
- 10271
- 10286
- 10288
- 10306
- 10321
- 10336
- 10351
- 10366
- 10381
- 10396
- 10411
- 10426
- 10441
- 10456
- 10471
- 10480
- 10488
- 10531
- 10539
- 10549
- 10564
- 10587
- 10606
- 10607
- 10612
- 10635
- 10654
- 10655
- 10674
- 10683
- 10699
- 10700
- 10716
- 10717
- 10721
- 10746

- 10756
- 10768
- 10786
- 10796
- 10801
- 10817
- 10826
- 10835
- 10844
- 10849
- 10873
- 10882
- 10892
- 10902
- 10910
- 10940
- 10943
- 10993
- 11015
- 11020
- 11058
- 11071
- 11115
- 11164
- 11274
- 11368
- 11376
- 11377
- 11484
- 11488
- 11504
- 11656
- 11667
- 11675
- 11742
- 11801
- 11976
- 12019
- 12114
- 12133
- 12143
- 12199
- 12293
- 12299
- 12316
- 12328
- 12339
- 12346
- 12387
- 12397
- 12404
- 12426

- 12445
- 12458
- 12466
- 12476
- 12478
- 12515
- 12525
- 12543
- 12555
- 12556
- 12573
- 12771
- 12780
- 59953
- 65212
- 65490

Ok - this is good… there appears to be a:

```
JMP_Absolute(LABEL2100);
```

```
.C:0834  A9 3C       LDA #$3C
.C:0836  8D 5D 08    STA $085D
.C:0839  4C 05 0B    JMP $0B05
```

And another routine at `0B05`

01/04/2021

April Fool's Day!

Like a fool, I'm up at 5am as my wife is vaccinating, so may as well do some Worron.

I grabbed the jump addresses yesterday. I'm going to make my life easier by annotating them with code regions so I can see where we are missing regions that I've yet to identify.

And here is the annotated jump address list:

- 2100 codeRegion:RoutineA
- 2821 codeRegion:TitleScreen
- 2836 codeRegion:TitleScreen
- 2978 codeRegion:TitleScreen
- 2996 codeRegion:TitleScreen
- 3023 codeRegion:TitleScreen
- 3041 codeRegion:TitleScreen
- 3176 codeRegion:TitleScreen
- 3232 codeRegion:TitleScreen
- 3251 codeRegion:TitleScreen
- 3265 codeRegion:TitleScreen
- 3267 codeRegion:TitleScreen
- 3286 codeRegion:TitleScreen

- 3288 codeRegion:TitleScreen
- 3322 codeRegion:TitleScreen
- 3369 codeRegion:TitleScreen
- 3392 codeRegion:TitleScreen
- 3395 codeRegion:TitleScreen
- 3406 codeRegion:TitleScreen
- 3417 codeRegion:TitleScreen
- 3420 codeRegion:TitleScreen
- 3423 codeRegion:TitleScreen
- 3440 codeRegion:TitleScreen
- 3453 codeRegion:TitleScreen
- 3473 codeRegion:TitleScreen
- 3493 codeRegion:TitleScreen
- 3531 codeRegion:TitleScreen
- 3532 codeRegion:TitleScreen
- 3542 codeRegion:TitleScreen
- 3572 codeRegion:TitleScreen
- 3613 codeRegion:TitleScreen
- 3637 codeRegion:TitleScreen
- 3654 codeRegion:TitleScreen
- 3657 codeRegion:TitleScreen
- 3680 codeRegion:TitleScreen
- 3721 codeRegion:TitleScreen
- 3758 codeRegion:TitleScreen
- 3779 codeRegion:TitleScreen
- 3789 codeRegion:TitleScreen
- 3790 codeRegion:TitleScreen
- 3832 codeRegion:TitleScreen
- 3835 codeRegion:TitleScreen
- 3856 codeRegion:TitleScreen
- 3882 codeRegion:TitleScreen
- 3897 codeRegion:TitleScreen
- 3913 codeRegion:TitleScreen
- 3948 codeRegion:TitleScreen
- 3985 codeRegion:TitleScreen
- 4018 codeRegion:TitleScreen
- 4063 codeRegion:TitleScreen
- 4137 codeRegion:TitleScreen
- 4163 codeRegion:TitleScreen
- 4202 codeRegion:TitleScreen
- 4217 codeRegion:TitleScreen
- 4220 codeRegion:TitleScreen
- 4243 codeRegion:TitleScreen
- 4256 codeRegion:TitleScreen
- 4272 codeRegion:TitleScreen
- 4349 codeRegion:TitleScreen
- 4369 codeRegion:TitleScreen
- 4389 codeRegion:TitleScreen
- 4409 codeRegion:TitleScreen
- 4439 codeRegion:TitleScreen
- 4451 codeRegion:TitleScreen
- 4479 codeRegion:TitleScreen

- 4569 codeRegion:TitleScreen
- 4648 codeRegion:TitleScreen
- 4689 codeRegion:TitleScreen
- 4697 codeRegion:TitleScreen
- 4710 codeRegion:TitleScreen
- 4781 codeRegion:TitleScreen
- 4806 codeRegion:TitleScreen
- 5000 codeRegion:RoutineC
- 6536 codeRegion:MainGame
- 6582 codeRegion:MainGame
- 6592 codeRegion:MainGame
- 6711 codeRegion:MainGame
- 6731 codeRegion:MainGame
- 6815 codeRegion:MainGame
- 6858 codeRegion:MainGame
- 6870 codeRegion:MainGame
- 6878 codeRegion:MainGame
- 6934 codeRegion:MainGame
- 7024 codeRegion:MainGame
- 7026 codeRegion:MainGame
- 7044 codeRegion:MainGame
- 7100 codeRegion:MainGame
- 7110 codeRegion:MainGame
- 7126 codeRegion:MainGame
- 7165 codeRegion:MainGame
- 7204 codeRegion:MainGame
- 7219 codeRegion:MainGame
- 7228 codeRegion:MainGame
- 7231 codeRegion:MainGame
- 7248 codeRegion:MainGame
- 7251 codeRegion:MainGame
- 7319 codeRegion:MainGame
- 7322 codeRegion:MainGame
- 7325 codeRegion:MainGame
- 7328 codeRegion:MainGame
- 7331 codeRegion:MainGame
- 7334 codeRegion:MainGame
- 7379 codeRegion:MainGame
- 7394 codeRegion:MainGame
- 7398 codeRegion:MainGame
- 7413 codeRegion:MainGame
- 7417 codeRegion:MainGame
- 7432 codeRegion:MainGame
- 7436 codeRegion:MainGame
- 7471 codeRegion:MainGame
- 7474 codeRegion:MainGame
- 7478 codeRegion:MainGame
- 7513 codeRegion:MainGame
- 7516 codeRegion:MainGame
- 7520 codeRegion:MainGame
- 7535 codeRegion:MainGame
- 7539 codeRegion:MainGame

- 7574 codeRegion:MainGame
- 7577 codeRegion:MainGame
- 7581 codeRegion:MainGame
- 7616 codeRegion:MainGame
- 7619 codeRegion:MainGame
- 7622 codeRegion:MainGame
- 7654 codeRegion:MainGame
- 7655 codeRegion:MainGame
- 7671 codeRegion:MainGame
- 7703 codeRegion:MainGame
- 7704 codeRegion:MainGame
- 7738 codeRegion:MainGame
- 7748 codeRegion:MainGame
- 7749 codeRegion:MainGame
- 7759 codeRegion:MainGame
- 7786 codeRegion:MainGame
- 7796 codeRegion:MainGame
- 7797 codeRegion:MainGame
- 7818 codeRegion:MainGame
- 7861 codeRegion:MainGame
- 7885 codeRegion:MainGame
- 7918 codeRegion:MainGame
- 7947 codeRegion:MainGame
- 7950 codeRegion:MainGame
- 7963 codeRegion:MainGame
- 8155 codeRegion:MainGame
- 8166 codeRegion:MainGame
- 8175 codeRegion:MainGame
- 8184 codeRegion:MainGame
- 8193 codeRegion:MainGame
- 8202 codeRegion:MainGame
- 8211 codeRegion:MainGame
- 8220 codeRegion:MainGame
- 8229 codeRegion:MainGame
- 8233 codeRegion:MainGame
- 8268 codeRegion:MainGame
- 8286 codeRegion:MainGame
- 8302 codeRegion:MainGame
- 8326 codeRegion:MainGame
- 8338 codeRegion:MainGame
- 8357 codeRegion:MainGame
- 8429 codeRegion:MainGame
- 8483 codeRegion:MainGame
- 8486 codeRegion:MainGame
- 8499 codeRegion:MainGame
- 8508 codeRegion:MainGame
- 8529 codeRegion:MainGame
- 8550 codeRegion:MainGame
- 8577 codeRegion:MainGame
- 8604 codeRegion:MainGame
- 8631 codeRegion:MainGame
- 8664 codeRegion:MainGame

- 8678 codeRegion:MainGame
- 8711 codeRegion:MainGame
- 8732 codeRegion:MainGame
- 8753 codeRegion:MainGame
- 8774 codeRegion:MainGame
- 8795 codeRegion:MainGame
- 8816 codeRegion:MainGame
- 8837 codeRegion:MainGame
- 8869 codeRegion:MainGame
- 8893 codeRegion:MainGame
- 8932 codeRegion:MainGame
- 8966 codeRegion:MainGame
- 8969 codeRegion:MainGame
- 9009 codeRegion:MainGame
- 9034 codeRegion:MainGame
- 9058 codeRegion:MainGame
- 9083 codeRegion:MainGame
- 9096 codeRegion:MainGame
- 9118 codeRegion:MainGame
- 9148 codeRegion:MainGame
- 9175 codeRegion:MainGame
- 9202 codeRegion:MainGame
- 9226 codeRegion:MainGame
- 9247 codeRegion:MainGame
- 9268 codeRegion:MainGame
- 9295 codeRegion:MainGame
- 9310 codeRegion:MainGame
- 9334 codeRegion:MainGame
- 9367 codeRegion:MainGame
- 9391 codeRegion:MainGame
- 9418 codeRegion:MainGame
- 9442 codeRegion:MainGame
- 9469 codeRegion:MainGame
- 9486 codeRegion:MainGame
- 9488 codeRegion:MainGame
- 9505 codeRegion:MainGame
- 9507 codeRegion:MainGame
- 9525 codeRegion:MainGame
- 9526 codeRegion:MainGame
- 9551 codeRegion:MainGame
- 9552 codeRegion:MainGame
- 9578 codeRegion:MainGame
- 9592 codeRegion:MainGame
- 9608 codeRegion:MainGame
- 9634 codeRegion:MainGame
- 9641 codeRegion:MainGame
- 9648 codeRegion:MainGame
- 9655 codeRegion:MainGame
- 9662 codeRegion:MainGame
- 9669 codeRegion:MainGame
- 9676 codeRegion:MainGame
- 9683 codeRegion:MainGame

- 9690 codeRegion:MainGame
- 9693 codeRegion:MainGame
- 9719 codeRegion:MainGame
- 9722 codeRegion:MainGame
- 9765 codeRegion:MainGame
- 9791 codeRegion:MainGame
- 9802 codeRegion:MainGame
- 9836 codeRegion:MainGame
- 9866 codeRegion:MainGame
- 9880 codeRegion:MainGame
- 9891 codeRegion:MainGame
- 9909 codeRegion:MainGame
- 9916 codeRegion:MainGame
- 9923 codeRegion:MainGame
- 9930 codeRegion:MainGame
- 9944 codeRegion:MainGame
- 9958 codeRegion:MainGame
- 9972 codeRegion:MainGame
- 9986 codeRegion:MainGame
- 10000 codeRegion:MainGame
- 10014 codeRegion:MainGame
- 10028 codeRegion:MainGame
- 10042 codeRegion:MainGame
- 10043 codeRegion:MainGame
- 10052 codeRegion:MainGame
- 10076 codeRegion:MainGame
- 10099 codeRegion:MainGame
- 10118 codeRegion:MainGame
- 10122 codeRegion:MainGame
- 10138 codeRegion:MainGame
- 10158 codeRegion:MainGame
- 10168 codeRegion:MainGame
- 10170 codeRegion:MainGame
- 10216 codeRegion:MainGame
- 10246 codeRegion:MainGame
- 10271 codeRegion:MainGame
- 10286 codeRegion:MainGame
- 10288 codeRegion:MainGame
- 10306 codeRegion:MainGame
- 10321 codeRegion:MainGame
- 10336 codeRegion:MainGame
- 10351 codeRegion:MainGame
- 10366 codeRegion:MainGame
- 10381 codeRegion:MainGame
- 10396 codeRegion:MainGame
- 10411 codeRegion:MainGame
- 10426 codeRegion:MainGame
- 10441 codeRegion:MainGame
- 10456 codeRegion:MainGame
- 10471 codeRegion:MainGame
- 10480 codeRegion:MainGame
- 10488 codeRegion:MainGame

- 10531 codeRegion:MainGame
- 10539 codeRegion:MainGame
- 10549 codeRegion:MainGame
- 10564 codeRegion:MainGame
- 10587 codeRegion:MainGame
- 10606 codeRegion:MainGame
- 10607 codeRegion:MainGame
- 10612 codeRegion:MainGame
- 10635 codeRegion:MainGame
- 10654 codeRegion:MainGame
- 10655 codeRegion:MainGame
- 10674 codeRegion:MainGame
- 10683 codeRegion:MainGame
- 10699 codeRegion:MainGame
- 10700 codeRegion:MainGame
- 10716 codeRegion:MainGame
- 10717 codeRegion:MainGame
- 10721 codeRegion:MainGame
- 10746 codeRegion:MainGame
- 10756 codeRegion:MainGame
- 10768 codeRegion:MainGame
- 10786 codeRegion:MainGame
- 10796 codeRegion:MainGame
- 10801 codeRegion:MainGame
- 10817 codeRegion:MainGame
- 10826 codeRegion:MainGame
- 10835 codeRegion:MainGame
- 10844 codeRegion:MainGame
- 10849 codeRegion:MainGame
- 10873 codeRegion:MainGame
- 10882 codeRegion:MainGame
- 10892 codeRegion:MainGame
- 10902 codeRegion:MainGame
- 10910 codeRegion:MainGame
- 10940 codeRegion:MainGame
- 10943 codeRegion:MainGame
- 10993 codeRegion:MainGame
- 11015 codeRegion:MainGame
- 11020 codeRegion:MainGame
- 11058 codeRegion:MainGame
- 11071 codeRegion:MainGame
- 11115 codeRegion:MainGame
- 11164 codeRegion:MainGame
- 11274 codeRegion:MainGame
- 11368 codeRegion:MainGame
- 11376 codeRegion:MainGame
- 11377 codeRegion:MainGame
- 11484 codeRegion:MainGame
- 11488 codeRegion:MainGame
- 11504 codeRegion:MainGame
- 11656 codeRegion:MainGame
- 11667 codeRegion:MainGame

- 11675 codeRegion:MainGame
- 11742 codeRegion:MainGame
- 11801 codeRegion:MainGame
- 11976 codeRegion:MainGame
- 12019 codeRegion:MainGame
- 12114 codeRegion:MainGame
- 12133 codeRegion:MainGame
- 12143 codeRegion:MainGame
- 12199 codeRegion:MainGame
- 12293 codeRegion:MainGame
- 12299 codeRegion:MainGame
- 12316 codeRegion:MainGame
- 12328 codeRegion:MainGame
- 12339 codeRegion:MainGame
- 12346 codeRegion:MainGame
- 12387 codeRegion:MainGame
- 12397 codeRegion:MainGame
- 12404 codeRegion:MainGame
- 12426 codeRegion:MainGame
- 12445 codeRegion:MainGame
- 12458 codeRegion:MainGame
- 12466 codeRegion:MainGame
- 12476 codeRegion:MainGame
- 12478 codeRegion:MainGame
- 12515 codeRegion:MainGame
- 12525 codeRegion:MainGame
- 12543 codeRegion:MainGame
- 12555 codeRegion:MainGame
- 12556 codeRegion:MainGame
- 12573 codeRegion:MainGame
- 12771 codeRegion:MainGame
- 12780 codeRegion:MainGame
- 59953 codeRegion:null
- 65212 codeRegion:null
- 65490 codeRegion:null

The last 3x addresses are kernal rom routines.

I think this might be it for the code, there might be others that I jump to in more of an indirect way, but I suspect my coding was not that sophisticated back then.

So we need to get to a point where the entire image is built from java without reference to the original data file, so we need to build data arrays.

Let's start off with the earliest data:

```
final int LABEL646 = 646;
final int LABEL647 = 647;
final int LABEL648 = 648;
final int LABEL788 = 788;
final int LABEL789 = 789;
final int LABEL832 = 832;
```

For starters, we can probably change our base address to 646, so let's try that for starters.

That seems to work.

Let's have a look through the code and see if we can figure out roughly what these are?

646 - appears unused as we just write to it
647 - the same
648 - the same
788 - the same
789 - the same
832 - the only value we seem to read and write, we use it as part of some subtraction

This means we should be able to start from 832

If we do that, things don't work, so there is definitely something we need there.

What do these addresses mean on the memory map?

Aha - these are special areas for basic:

646 - Current color, cursor color. Values: $00-$0F, 0-15.
647 - Color of character under cursor. Values: $00-$0F, 0-15
648 - High byte of pointer to screen memory for screen input/output
788+789 - Execution address of interrupt service routine
832 - part of the datasette buffer - which is probably why it's the only variable we appear to use

I've now detected whether a label is an ARRAY or not but checking whether it is accessed via an indexed instruction or not.

We can see looking through the variable names now, that this is true as there are multiple bytes between these:

```
final int LABEL2108 = 2108;
final int ARRAY2109 = 2109;
final int ARRAY2117 = 2117;
final int ARRAY2125 = 2125;
final int ARRAY2133 = 2133;
final int LABEL2141 = 2141;
final int ARRAY2142 = 2142;
final int ARRAY2174 = 2174;
```

Let's try manually adding data for 2108 which has a value of zero in the monitor

mmm… if I just set these 6 operating system/basic values, then the game doesn't start up, which makes me think there is other state in there that I need to capture - at least for now until I can get rid of it. I'm thinking that I don't rock the boat for now and just create an array of everything from 646 - 1024 and see if that works.

Great, I've managed to add an array of data for the 646-2049 area. We'll further refine that.

02/04/2021

So I started generating some of the longer arrays and I came across a limitation in java where method names cannot be longer than 64k which is ironic. So I'm having to move the longer arrays into their own file and keep the shorter ones that I'm likely to edit more often, in memory, plus I'll want to see their values more than - say - the character data or screen tile data.

Done it… can now run the game completely self contained from the WorronCode file. I've generated data files for each array of a certain size to work around the array limits, but I think this is It. I can now start to refactor.

Let the fun begin.

Now I can start getting rid of some of the vic labels and replacing them with ones in the VIC class etc.

I've changed all the labels for vic/sid/cia1/cia2… next up I'm going to change bit wise operations such as AND/ORA to use binary representation rather than decimal.

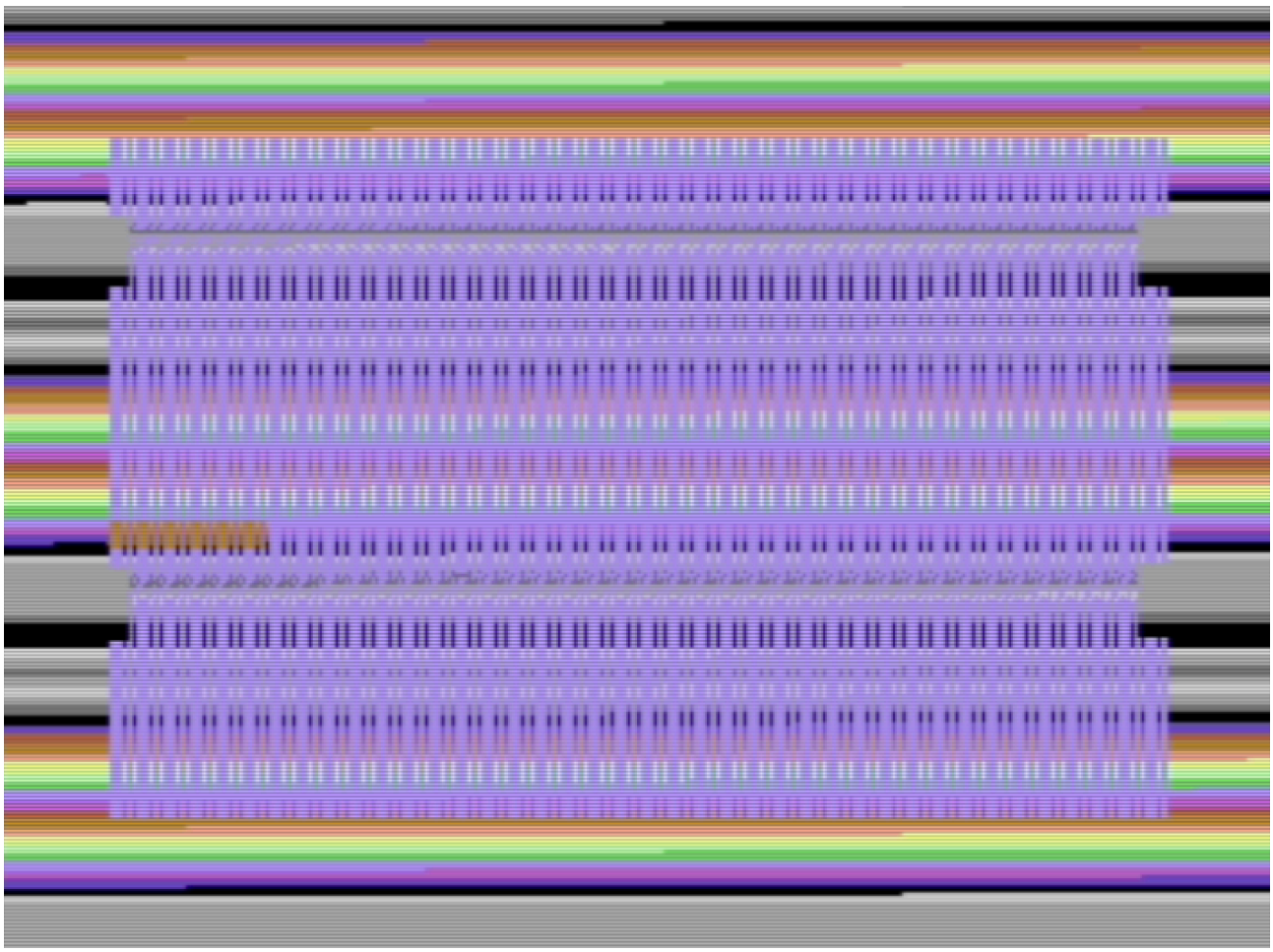
I've changed all the AND_Immediate/ORA_Immediate… there are no EOR_Immediate as I had no idea what that did back in 1987. haha. Funny.

Next up, let's change anything that sets the colour to point at the colour java code I've written for my other project.

So I've gone through and annotated the current code better, but now I'm going to try to remove an instruction which will be a test of the compilation as it'll mean all the code is shunted up; if there is any code that is dependent on the current order, I'll soon find out!

```
// Check for whether fire is pressed on joystick
LDA_Absolute(ciaPortA);
AND_Immediate(0b00010000);
CMP_Immediate(0b00010000);
```

I guess that didn't work:



This tells me that I've perhaps got some code somewhere in some of those arrays that I don't yet know about.

Clearly I need to understand the code a LOT better before I can start adding/removing any new/existing instructions.

Let's start plugging in labels here:

```
array(SomeAbsoluteAddressMsbArray3, 73, 74, 75, 76, 77, 78,
79).isPageAligned(false);
// TODO: NOT ALL ARRAY18832
array(SomeAbsoluteAddressLsbArray3, reference(ARRAY18832).lsb(),
reference(ARRAY18832).lsb(), reference(ARRAY18832).lsb(),
reference(ARRAY18832).lsb(), reference(ARRAY18832).lsb(),
reference(ARRAY18832).lsb(),
reference(ARRAY18832).lsb()).isPageAligned(false);
```

(73*256)+144=18832
(74*256)+144=19088
(75*256)+144=19344
(76*256)+144=19600
(77*256)+144=19856
(78*256)+144=20112
(79*256)+144=20368

This was the scrolling message, as it was 7 pages long

One thing I haven't figured out yet is where the character memory is stored.

The bank is defined by:

```
static public StaticWord cia2PortASerialBusAccess = add(56576, 
"PortASerialBusAccess"); // $DD00
```

We seem to set it 2x which is a bit odd… either way, it is the same value:

```
LDA_Absolute(cia2PortASerialBusAccess);
AND_Immediate(0b11111100);
ORA_Immediate(0b00000010);
STA_Absolute(cia2PortASerialBusAccess);
```

We know from seeing where the sprites were above, that we're here:

```
//%10, 2: Bank #1, $4000-$7FFF, 16384-32767.
```

Bank 1

So our character memory is somewhere here. Where, is determined by:

```
// Memory setup register. Bits:
// Bits #1-#3: In text mode, pointer to character memory (bits 
#11-#13), relative to VIC bank, memory address $DD00. Values:
// %000, 0: $0000-$07FF, 0-2047.
// %001, 1: $0800-$0FFF, 2048-4095.
// %010, 2: $1000-$17FF, 4096-6143.
// %011, 3: $1800-$1FFF, 6144-8191.
// %100, 4: $2000-$27FF, 8192-10239.
// %101, 5: $2800-$2FFF, 10240-12287.
// %110, 6: $3000-$37FF, 12288-14335.
// %111, 7: $3800-$3FFF, 14336-16383.
// Values %010 and %011 in VIC bank #0 and #2 select Character ROM 
instead.
// In bitmap mode, pointer to bitmap memory (bit #13), relative to 
VIC bank, memory address $DD00. Values:
// %0xx, 0: $0000-$1FFF, 0-8191.
// %1xx, 4: $2000-$3FFF, 8192-16383.
// Bits #4-#7: Pointer to screen memory (bits #10-#13), relative 
to VIC bank, memory address $DD00. Values:
static public StaticWord vicMemorySetupRegister = add(53272, 
"MemorySetupRegister"); // $D018
```

There seem to be two values:

```
LDA_Absolute(vicMemorySetupRegister);
AND_Immediate(0b11110000);
ORA_Immediate(0b00001100);
STA_Absolute(vicMemorySetupRegister);
```

```
LDA_Absolute(vicMemorySetupRegister);
AND_Immediate(0b11110000);
ORA_Immediate(0b00001110);
STA_Absolute(vicMemorySetupRegister);
```

Which would indicate values of 12 and 14, so
12 = 12288+16384 = 28672 = $7000
and
14 = 14336+16384 = 30720 = $7800

Unless I've misread that, it would appear as though we had 2x character sets? Maybe we do.

I think it is definitely at 30720 as there are 8 byte arrays that look like character animations:

e.g.

```
// TODO: These feel like characters that we animate as they are 8
characters long
array(ARRAY30936, 85, 85, 125, 121, 121, 105, 85,
85).isPageAligned(false);
array(ARRAY30944, 255, 255, 213, 213, 218, 216, 216,
216).isPageAligned(false);
array(ARRAY30952, 255, 254, 86, 86, 166, 54, 54,
54).isPageAligned(false);
array(ARRAY30960, 54, 54, 54, 246, 86, 86, 170,
170).isPageAligned(false);
```

I didn't appear to know about LSR/ROL/ASL/ROR back in the day, so must have explicitly written out all the scrolling combinations… cor dear… makes life simpler for me though I guess.

Let's pick one of these characters and see if we can figure out what is going on:

```
array(ARRAY30936, 85, 85, 125, 121, 121, 105, 85,
85).isPageAligned(false);
```

We seem to write to this via:

```
LDA_AbsoluteX(ARRAY33032);
STA_AbsoluteY(ARRAY30936);
```

```
array(ARRAY33032, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 117, 85,
85, 89, 85, 85, 85, 85, 125, 121, 121, 105, 85, 85, 85, 85, 125,
121, 122, 106, 90, 85).isPageAligned(false);
```

I think there are 4x frame of animation here… let's slow down the game and see if that's the case.

4x frames of animation makes sense of course as the x resolution of our characters is halved with multi-colour mode so we can only move 2x pixels at a time.

The next thing… how big are our tiles?

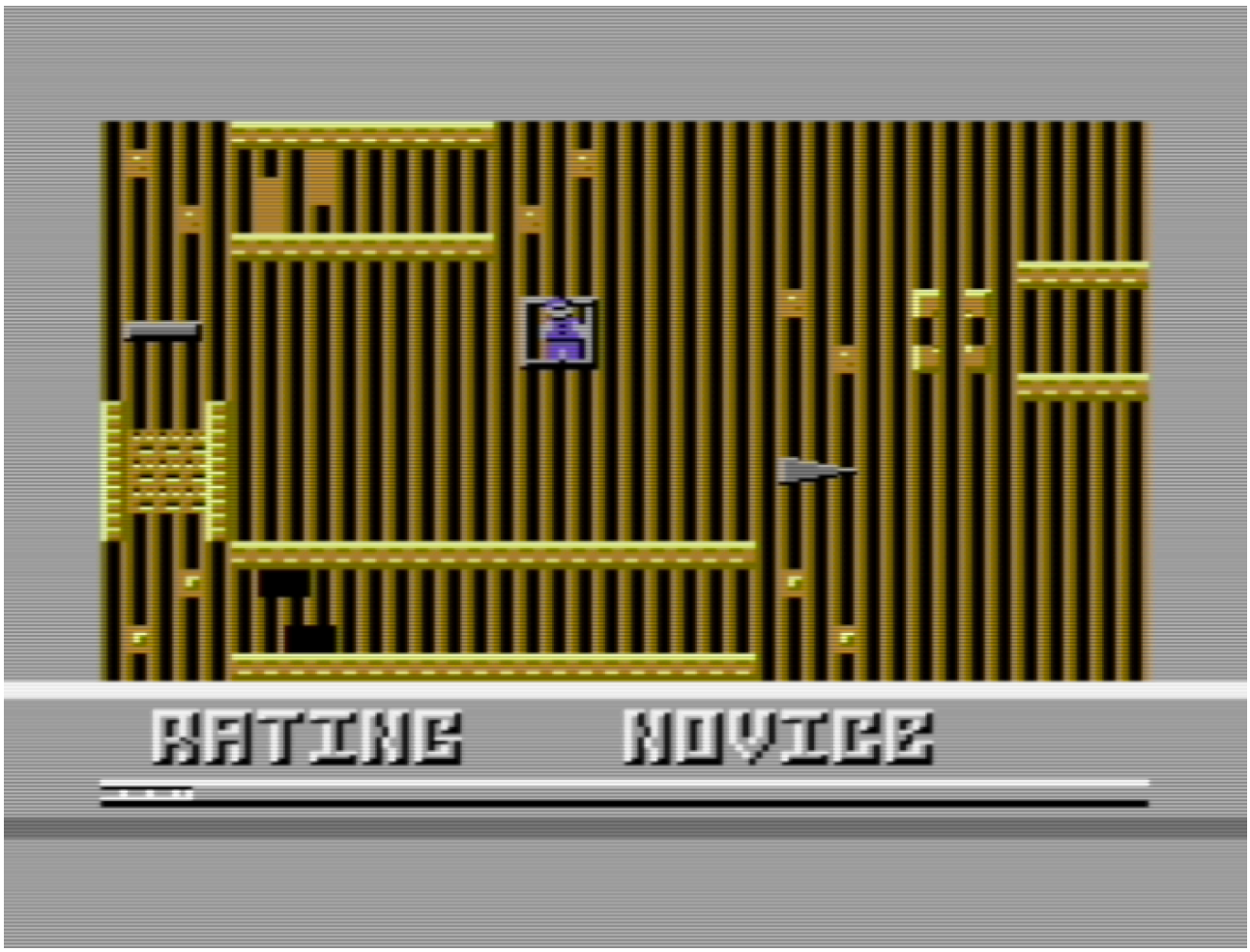
I think they are 5x5 characters, which is 25 characters per tile. That means our screen data should be 8x5=40 tiles in size.

Before we get to that, let's experiment by hacking the memory around the character set to confirm where we think it is:

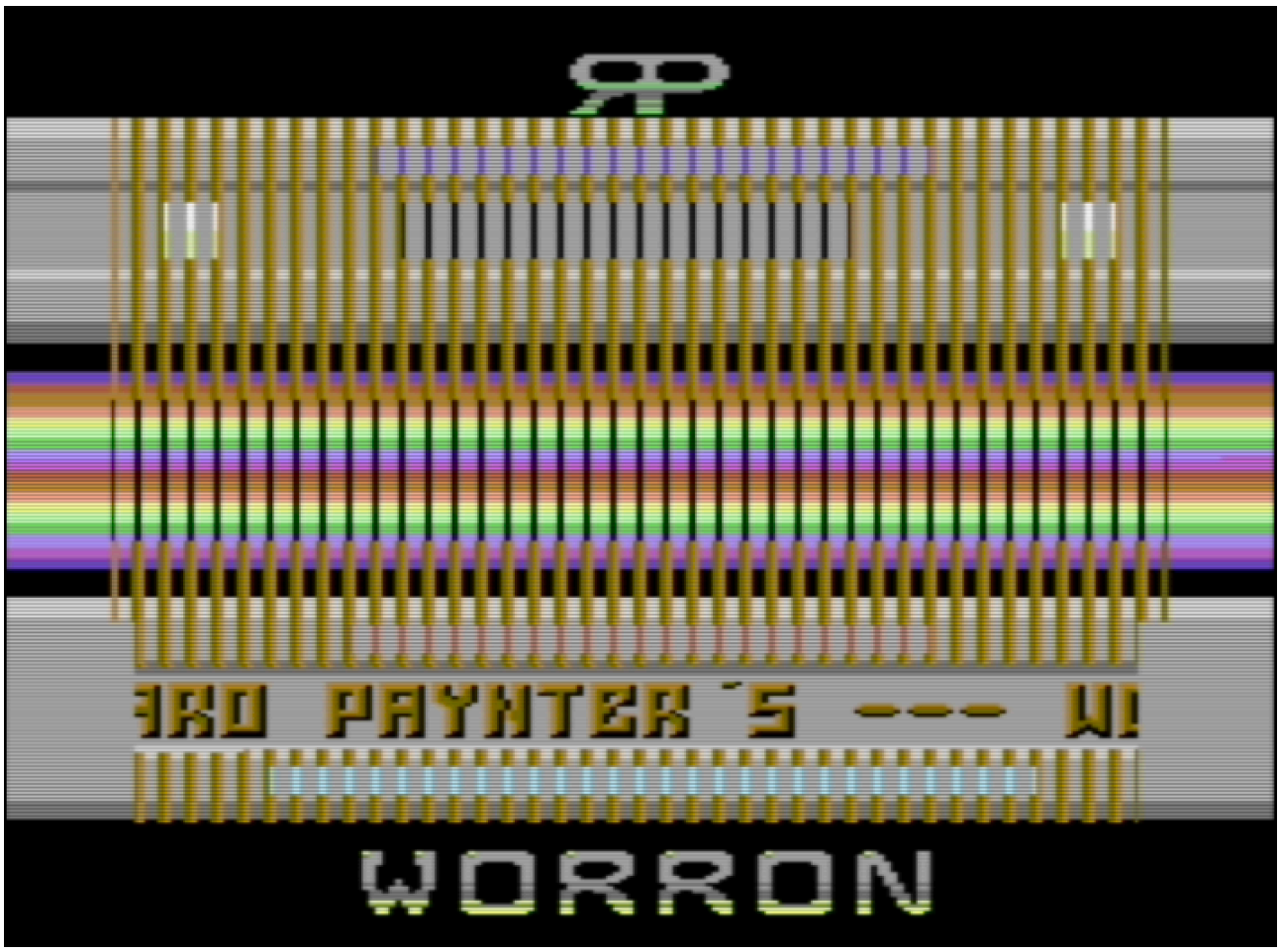
7000 is definitely character memory as I've changed that byte to be FF and I can see an artefact on the large characters in the scrolling message. So that's one character set found. Is there another?

Let's set the whole of that 2k in the debugger to be FF and see if all graphics disappear.

Filling from 7800 - 8000 produces:

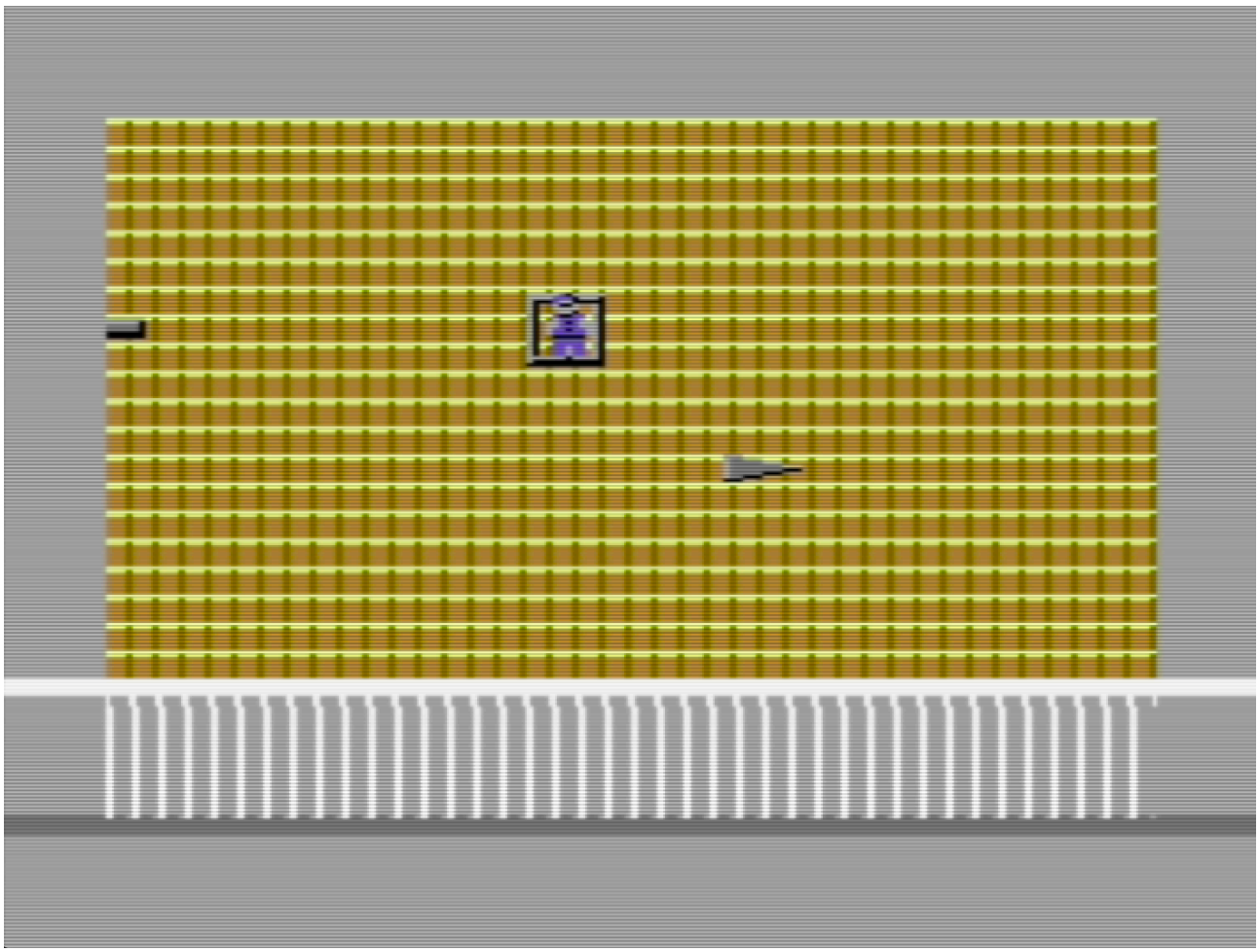


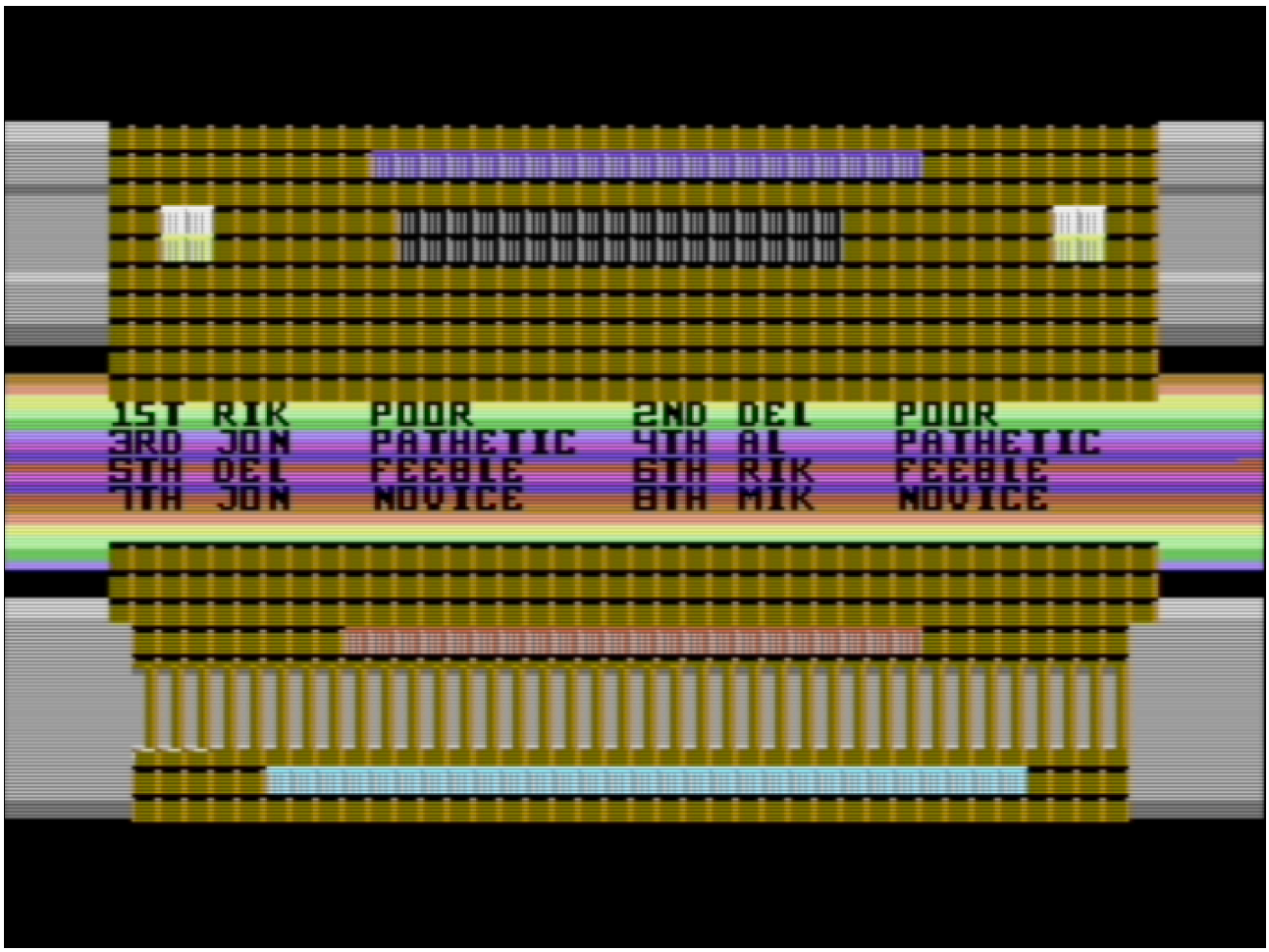
We still have large character text. What about on the title screen?



This definitely tells me that we have 2x character sets of 2k each.

Let’s try setting 7000 - 7800:





Let's see if I can find where the characters for the hi-score are?

We know that they are in the 7800-8000 character set.

I think it's here:

```
(C:$7810) memchar 7800 8000
>C:7800 ********
>C:7801 ********
>C:7802 ********
>C:7803 ********
>C:7804 ********
>C:7805 ********
>C:7806 ********
>C:7807 ********

>C:7808 *....***
>C:7809 ..**..**
>C:780a ..**..**
>C:780b ......**
>C:780c ..**..**
>C:780d ..**..**
>C:780e ..**..**
>C:780f ********
```

```
>C:7810 ......**
>C:7811 ..**..**
>C:7812 ..**..**
>C:7813 ......**
>C:7814 ..**..**
>C:7815 ..**..**
>C:7816 ......**
>C:7817 ********

>C:7818 ......**
>C:7819 ..**..**
>C:781a ..******
>C:781b ..******
>C:781c ..******
>C:781d ..**..**
>C:781e ......**
>C:781f ********

>C:7820 .....***
>C:7821 ..**..**
>C:7822 ..**..**
>C:7823 ..**..**
>C:7824 ..**..**
>C:7825 ..**..**
>C:7826 .....***
>C:7827 ********

>C:7828 ......**
>C:7829 ..**..**
>C:782a ..******
>C:782b ....****
>C:782c ..******
>C:782d ..**..**
>C:782e ......**
>C:782f ********

>C:7800  00 00 00 00  00 00 00 00  78 cc cc fc  cc cc cc
00   ........x.......
>C:7810  fc cc cc fc  cc cc fc 00  fc cc c0 c0  c0 cc fc
00   ................
>C:7820  f8 cc cc cc  cc cc f8 00  fc cc c0 f0  c0 cc fc
00   ................

>C:79b0 .......*
>C:79b1 ..***..*
>C:79b2 ..***..*
>C:79b3 ..***..*
>C:79b4 ..***..*
>C:79b5 ..***..*
>C:79b6 .......*
>C:79b7 ********
```

```
>C:79b8 **...***
>C:79b9 **...***
>C:79ba ***..***
>C:79bb ***..***
>C:79bc ***..***
>C:79bd ***..***
>C:79be *......*
>C:79bf ********

>C:79c0 .......*
>C:79c1 ..***..*
>C:79c2 *****..*
>C:79c3 .......*
>C:79c4 ..******
>C:79c5 ..***..*
>C:79c6 .......*
>C:79c7 ********

>C:79c8 .......*
>C:79c9 ..***..*
>C:79ca *****..*
>C:79cb **.....*
>C:79cc *****..*
>C:79cd ..***..*
>C:79ce .......*
>C:79cf ********

>C:79d0 ..***..*
>C:79d1 ..***..*
>C:79d2 ..***..*
>C:79d3 .......*
>C:79d4 *****..*
>C:79d5 *****..*
>C:79d6 *****..*
>C:79d7 ********

>C:79d8 .......*
>C:79d9 ..***..*
>C:79da ..******
>C:79db .......*
>C:79dc *****..*
>C:79dd ..***..*
>C:79de .......*
>C:79df ********

>C:79e0 .......*
>C:79e1 ..***..*
>C:79e2 ..******
>C:79e3 .......*
>C:79e4 ..***..*
>C:79e5 ..***..*
>C:79e6 .......*
```

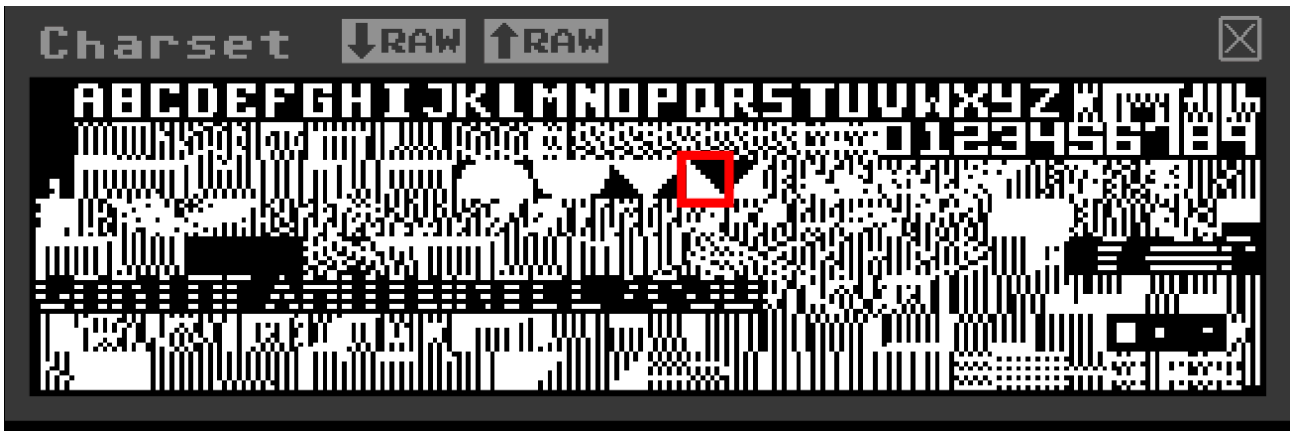
```
>C:79e7 ********

>C:79e8 .......*
>C:79e9 ..***..*
>C:79ea *****..*
>C:79eb *****..*
>C:79ec *****..*
>C:79ed *****..*
>C:79ee *****..*
>C:79ef ********

>C:79f0 .......*
>C:79f1 ..***..*
>C:79f2 ..***..*
>C:79f3 .......*
>C:79f4 ..***..*
>C:79f5 ..***..*
>C:79f6 .......*
>C:79f7 ********

>C:79f8 .......*
>C:79f9 ..***..*
>C:79fa ..***..*
>C:79fb .......*
>C:79fc *****..*
>C:79fd *****..*
>C:79fe *****..*
>C:79ff ********
```

We should be able to set 7808-780f to be FF and we'll lose A… let's try that:

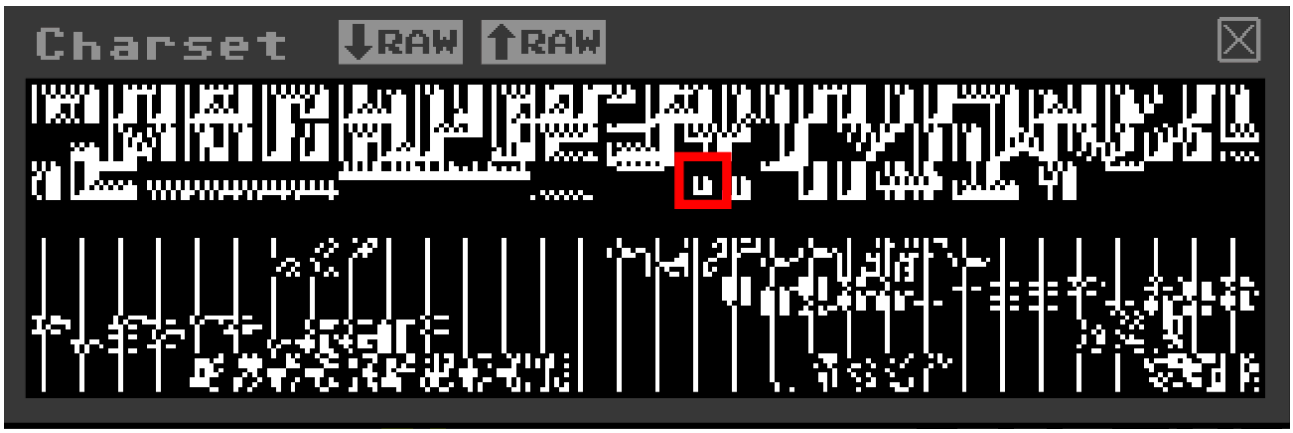
YES!

The character definitions are:
00: Space
01-26: A-Z
54-64: 0-9

30720 (7800-8000):



28672 (7000-7800):



In fact, it looks like I don't really use more than about the 1st 1K of the 28672 character set as the lower position is animated character data that I think might be copied.

03/04/2021

I wanted to confirm the addresses I was seeing yesterday for character set, so I'll spin up the c64 debugger and see where memory is rotating (i.e. animating characters)

Memory seems to be 'animated' from:
7720 - 7770
30496 - 30576

No - I was mistaken - those are not animations.

Instead, in the game, I can see plenty of 'animations' occurring around the 7974 area (31092) which is our main character set.

We will make an effort today to make the code relocatable. Our test to see if we've managed to achieve this is to do the following, by adding a single new NOP:

```
new BaseAddress(assembler, word(2821), "TitleScreen");
label(TitleScreen2821);
NOP_Implied();
```

This still fails so back to the drawing board.

Aha - This needs fixing as I've found some data that explicitly refers to the current addresses (as lsb/msb pairs):

```
array(SomeAbsoluteAddressMsb4, 80, 80, 80, 80, 80, 80, 80, 80, 80,
80, 80, 80, 80, 80, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81,
81, 81, 81, 81, 81, 81, 81).isPageAligned(false);
array(SomeAbsoluteAddressLsb4, 168, 177, 186, 195, 204, 213, 213,
222, 222, 231, 231, 240, 240, 249, 249, 2, 2, 11, 11, 20, 20, 29,
29, 29, 38, 38, 47, 47, 47).isPageAligned(false);
```

Having added some asserts, I see that we have 32 msb, but only 29 lsb. If this is indeed level data, this might explain why the game gets corrupted on later levels… we'll return to this as one of the many bugs that needs fixing, but not yet!

So I need to determine what these addresses are and make sure we have a label for each, if not already.

Let's pick the first one = 20648

We don't have a label for this.

It seems to sit in the middle of this:

```
array(TitleScreenScrollingMessage,
loadArray(TitleScreenScrollingMessage,
10864)).isPageAligned(false);
```

Clearly our scrolling message is not this big, so we need to split out this 10k lump into something smaller. Maybe the first thing to do is re-define the scrolling message as a java string, which we'll convert on the fly? Let's do that. We know it's 7k long. Intellij helpfully points out that 17 year old Richard cannot spell 'INNOVATION' so we'll edit his immature scrolling message text. Funnily enough, I still listen to John Carpenter soundtracks, so somethings have not changed haha.

```
final String scrollingMessage =
        "                                WELCOME TO RICHARD PAYNTER'S
--- WORRON ---    I KNOW THAT SCROLLING MESSAGES ARE NOT THE MOST
RECE" +
        "NT INOVATION BUT YOU MIGHT AS WELL READ THIS ONE
BECAUSE WHAT I'M TRYING TO DO HERE IS BASICALLY TWO THINGS ---
ONE IS T" +
        "O TRY AND RELAX YOU BY GIVING YOU SOMETHING NICE TO LOOK
AT AND READ WHILE TRYING TO FIND OUT HOW TO START THE GAME ---
AND T" +
        "HE OTHER IS THAT I HAD SOME SPARE MEMORY LYING AROUND
WHICH I THOUGHT I COULD CRAM LOTS OF MEANINGLESS BITS OF INFO INTO
FO" +
        "R EXAMPLE DID YOU KNOW THAT THE TASMANIAN OSPREY FLIES
SIDEWAYS DURING THE HEIGHT OF THE OSPREAN MATING SEASON      DID
YOU KNOW" +
        " THAT      WELL I DIDN'T UNTIL I WROTE IT DOWN HERE IN
THIS ENCYCLOPEDIA OF MINDLESS TRIVIA      AND NOW FOR SOME LONG
AWAITED H" +
        "ELLOS      HELLO     HELLO     HELLO     I SUPPOSE YOU
THINK THAT WAS FUNNY     AND WHY NOT    NOW LET'S BE SERIOUS --- I
WOULD " +
        "LIKE TO SAY HELLO TO DERYCK BANKS FOR HELPING ME WRITE
THIS MASTERPIECE OF ENGLISH LITERATURE AND FOR MAKING AWFUL CUPS
OF TEA  " +
        "    HELLO TO JONATHON BERTRAM HEDLEY FOR INANE 'ON THE
SPUR OF THE MOMENT' COMMENTS LIKE --- 'FASCINATING' OR 'ARE YOU
REALLY GO" +
```

```
        "ING TO TRY AND SELL THIS'    THANK YOU JON      ALEC FOR
THE DISC DRIVE      ALEC 'CONCORDE' MURRELL FOR LENDING ME HIS
DAD'S LP" +
        "'S EVEN THOUGH HIS DAD DIDN'T KNOW I HAD THEM AND
PROMPTLY SENT THE POLICE 'ROUND WITH A SEARCH WARRANT      THANK
YOU TO YOU F" +
        "OR BUYING THIS      MUSIC DURING THE LONG AND TIRING
PROGRAMMING SESSIONS PROVIDED UNKNOWINGLY BY JOHN CARPENTER IN
ASSOCIATION " +
        "WITH ALAN HOWARTH      --- GOOD STUFF ---      FINALLY A
FAREWELL TO THE 'HONOURABLE' DIM TANG DELL FROM PARANOID --- BO
DIDDLY " +
        "--- MIKE THE MAN --- DILL BALONEY        PS IF YOU WANT
A FREE DRINK GO TO BRIAN WEBBER OF GILLINGHAM --- REMEMBER RP
";
```

Childish, nonsense, some of which I don't actually understand 35 years later haha… plus some spelling mistakes, but we're not about improving the game as we should respect how it was, we're just about adding the missing bits.

I have set myself some rules here… don't photoshop the past… keep as much of the game as it was, warts and all. Only fix the bugs and add the stuff that I simply could not do back then, such as music/sound.

Let's change the source of the data for this string, then we can delete the data file we generated yesterday.

Before we do, we need to split this line to 2x:

```
array(TitleScreenScrollingMessage,
loadArray(TitleScreenScrollingMessage,
10864)).isPageAligned(false);
```

Ok - managed to do that… My wife is off to run a marathon in Northampton today, so I've sent her this:

These are the address at:

- 0 lsb:168 msb:80 address:20648 50a8 length:9
- 1 lsb:177 msb:80 address:20657 50b1 length:9
- 2 lsb:186 msb:80 address:20666 50ba length:9
- 3 lsb:195 msb:80 address:20675 50c3 length:9
- 4 lsb:204 msb:80 address:20684 50cc length:9
- 5 lsb:213 msb:80 address:20693 50d5 length:0
- 6 lsb:213 msb:80 address:20693 50d5 length:9
- 7 lsb:222 msb:80 address:20702 50de length:0
- 8 lsb:222 msb:80 address:20702 50de length:9
- 9 lsb:231 msb:80 address:20711 50e7 length:0
- 10 lsb:231 msb:80 address:20711 50e7 length:9
- 11 lsb:240 msb:80 address:20720 50f0 length:0
- 12 lsb:240 msb:80 address:20720 50f0 length:9
- 13 lsb:249 msb:80 address:20729 50f9 length:0
- 14 lsb:249 msb:80 address:20729 50f9 length:9
- 15 lsb:2 msb:81 address:20738 5102 length:0
- 16 lsb:2 msb:81 address:20738 5102 length:9
- 17 lsb:11 msb:81 address:20747 510b length:0
- 18 lsb:11 msb:81 address:20747 510b length:9
- 19 lsb:20 msb:81 address:20756 5114 length:0
- 20 lsb:20 msb:81 address:20756 5114 length:9
- 21 lsb:29 msb:81 address:20765 511d length:0
- 22 lsb:29 msb:81 address:20765 511d length:0

- 23 lsb:29 msb:81 address:20765 511d length:9
- 24 lsb:38 msb:81 address:20774 5126 length:0
- 25 lsb:38 msb:81 address:20774 5126 length:9
- 26 lsb:47 msb:81 address:20783 512f length:0
- 27 lsb:47 msb:81 address:20783 512f length:0
- 28 lsb:47 msb:81 address:20783 512f length:0
- 29 lsb:47 msb:81 address:20783 512f length:0
- 30 lsb:47 msb:81 address:20783 512f length:0
- 31 lsb:47 msb:81 address:20783 512f length:-1

Ok - making some progress. I want to start by finding obvious things. Let's search for the sequence RIK from the hi-score. This will be expressed as:

dataArray = {int[3]@798}
 0 = 18
 1 = 9
 2 = 11

So we need to try to find this sequence.

RIK found at index:18636
PATHETIC found at index:18682
NOVICE found at index:18762
DEL found at index:18656
POOR found at index:18642
FEEBLE found at index:18722
MIK found at index:18776

Other strings:
RICHARD found at index:2548
PAYNTER'S found at index:18880
FIREBIRD found at index:2455
PRESENT found at index:2464
REMAINING found at index:5439
MEN found at index:2529
GRID found at index:5423
REMAINING found at index:5439
WORRON found at index:5416
DERYCK found at index:19749
OSPREY found at index:19386
PRESS found at index:2512
FIRE found at index:2455
COMMENCE found at index:2526
PROGRAMMED found at index:2534
RATING found at index:5454

Tracked down the "PRESS FIRE TO COMMENCE" string

```
label(PrintPressFireToCommenceLoop);
LDA_AbsoluteX(PressFireToCommenceArray);
STA_AbsoluteX(ARRAY30425);
LDA_Immediate(2);
```

```
STA_AbsoluteX(colourRAM.add(729));
INX_Implied();
// TODO: Can't we compare this with the string?
assertEquals(pressFireToCommenceString.length(),22);
CPX_Immediate(22);
BNE_Relative(PrintPressFireToCommenceLoop);
```

One thing I've not identified yet is where the screen memory is stored.. the vice monitor should be able to tell us that.

```
(C:$0cc6) screen
Displaying 40x25 screen at $7400:

            firebird present

  [\       !#_(&!" !&(*,-!/0        [\
  ]^       "# $'"2 %')+)." 1        ]^




LLLLLL    KLLLLLLLLL KLLL KLLL K  KLLL
     L    L      L  L L  L L  L LQ L
     L RQ L KLLL LLLN LLLN MLLN LOQL
     LRPOQL L  L L OQ L OQ      L OL
     MP  ON MLLN N  OLN  OLLLLLLN  N



         press@fire@to@commence

XXXXXXXXXXXXXXXX88lg_Xfg(j"#lgXX12(jXXf
XXXXXXXXXXXXXXXX9:mih!hi)kUUmiXX34)kXX*

      programmed by richard paynter
```

So 7400 = 29696

Ok - so this explains what the 2nd kilobyte of character set 2 is used for and why I thought it was being animated… it is actually the screen memory.

We should now be able to remove all arrays that point to this area of memory as they'll presumably be calculated.

So there are a whole load of variables that are in the screen memory:

```
// TODO: START OF SCREEN MEMORY 29696
// TODO: START OF SCREEN MEMORY 29696
// TODO: START OF SCREEN MEMORY 29696
```

```
// TODO: All of the below should be replaced by calls directly to
the addres, rather than indirectly via a label
array(ARRAY29696, 32).isPageAligned(false);
array(ARRAY29697, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32).isPageAligned(false);
array(ARRAY29736, 32).isPageAligned(false);
array(ARRAY29737, 32, 32, 32, 32, 32, 32, 32, 32,
32).isPageAligned(false);
array(ARRAY29746, loadArray(ARRAY29746, 70)).isPageAligned(false);

// Start of top large character data
// I think this is a variable that represents the top left
character to render
array(SomethingRelatedToLargeCharacterTopLeftArray,
32).isPageAligned(false);
// I think this is a variable that represents the top right
character to render
array(SomethingRelatedToLargeCharacterTopRightArray,
32).isPageAligned(false);
array(LABEL29818, 155).isPageAligned(false);
array(LABEL29819, 156, 32, 32, 32, 32, 32, 32,
32).isPageAligned(false);
array(ARRAY29827, 161, 163, 159, 168, 166, 161, 162, 32, 161, 166,
168, 170, 172, 173, 161, 175, 176, 32, 32, 32, 32, 32, 32, 32,
32).isPageAligned(false);
array(LABEL29852, 155).isPageAligned(false);
array(LABEL29853, 156, 32, 32).isPageAligned(false);
// End of top large character data ???
```

Let's start by changing 32 (space) to zero for now as I think we should be clearing the screen anyway.

I'm now going to go through all the variables for screen memory and replace them with screenCell(x,y)

Worron character sprite definitions are next.

We should start to assign variable names to these.

04/04/2021

Which of the 8x sprite pointer values are used for various characters?

spritePointer3 seems to always be used for the man, but the other sprite pointers appear to be used for anything, including the portal.

I was looking for PATHETIC yesterday, but only found it in the hi-score table, but I was looking for the source string:

PATHETIC found at index:20666

05/04/2021

Today, I want to try to make the code relocatable, which means finding all lsb/msb array references and replacing them with labels.

There are 16 ratings, starting at NOVICE.

Ok - I've decompiled the various rating levels:

- 0 20648 NOVICE
- 1 20657 FEEBLE
- 2 20666 PATHETIC
- 3 20675 POOR
- 4 20684 FAIR
- 5 20693 NOT BAD
- 6 20702 AVERAGE
- 7 20711 GOOD
- 8 20720 VERY GOOD
- 9 20729 AMAZING
- 10 20738 FANTASTIC
- 11 20747 SPLENDID
- 12 20756 SUPERB
- 13 20765 EXCELLENT

- 14 20774 SUPREME
- 15 20783 HUMAN

I seem to have 2x arrays of lsb/msb pointing at the ratings text, each of different sizes… no wonder my code used to crash!

I think I've found a culprit for non-relocatable code:

```
LDA_Immediate(240);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineLsb);
LDA_Immediate(12);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineMsb);
```

This is address 3312

Which points to this code:

```
3312  $0cf0: LDA VICInterruptStatusRegister                173 25 208     ad 19 d0
3315  $0cf3: AND #1                                  41 1          29 01
3317  $0cf5: BNE TitleScreen3322                          208 3         d0 03
3319  $0cf7: JMP KernalROMNormalIrqInterrupt                76 49 234      4c 31 ea
```

There are others too, the next being:

```
LDA_Immediate(57);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineLsb);
LDA_Immediate(16);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineMsb);
```

Address = 4153

```
4153  $1039: LDA VICInterruptStatusRegister                173 25 208     ad 19 d0
4156  $103c: AND #1                                  41 1          29 01
4158  $103e: BNE TitleScreen4163                          208 3         d0 03
4160  $1040: JMP KernalROMNormalIrqInterrupt                76 49 234      4c 31 ea
```

```
LDA_Immediate(49);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineLsb);
LDA_Immediate(234);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineMsb);
```

Address = 59953 - this seems very high up in memory $ea31 - in the kernal rom?

Yes:

```
static public StaticWord kernalROMNormalIrqInterrupt = add(59953,
"NormalIrqInterrupt"); // $EA31
```

Last one:

```
LDA_Immediate(243);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineLsb);
LDA_Immediate(27);
STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineMsb);
```

Address = 7155

```
7155  $1bf3: LDA VICInterruptStatusRegister                173 25 208     ad 19 d0
7158  $1bf6: AND #1                                  41 1          29 01
7160  $1bf8: BNE MainGame7165                              208 3         d0 03
7162  $1bfa: JMP KernalROMNormalIrqInterrupt                76 49 234      4c 31 ea
```

Ok - that still hasn't made the code relocatable… need to hunt some more.

These:

```
final int
OperatingSystemAndBasicPointersRAMCurrentColorCursorColor = 646;
final int
OperatingSystemAndBasicPointersRAMColourOfCharacterUnderCursor =
647;
final int
OperatingSystemAndBasicPointersRAMHighByteOfPointerToScreenMemoryF
orScreenInput = 648;
final int
OperatingSystemAndBasicPointersRAMExecutionAddressOfInterruptServi
ceRoutineLsb = 788;
final int
OperatingSystemAndBasicPointersRAMExecutionAddressOfInterruptServi
ceRoutineMsb = 789;
final int OperatingSystemAndBasicPointersRAMPartOfDatasetteBuffer
= 832;
```

Appear to mostly be read, so I'm thinking that they are only read by the kernal rom code. The lsb/msb looks fishy and a likely source of error.

I think I've managed to make the code relocatable, yay. I can now add some NOP which will shunt all the code down and things start up ok, apart from the fact that we start off on

the hi-score screen. I'll need to fix that. Basically, I dropped the 600+ byte area and just load from 2049.

I've also fixed what I think is an issue with not clearing bit 8 of the raster line when we select a new raster line. I've seen a crash occur several times that looks like the one in Bruce Lee where we try to interrupt at a line that can never happen as we select the lsb raster line, but leave the msb as 1, so try to interrupt at - say - 500 which never happens.

We'll see if that was actually the issue over time. (It was)

Unfortunately, some of my work has messed up some of the graphics on the hi-score page, so I'll need to revisit that.

Should be easy to fix at some point.

I've also noticed PLEASE INSERT YOUR NAME - this is at:
PLEASE found at index:2426

Let's change the loop to display this string so it counts down, start of an improvement of the code. Ok - so I could resist replacing a count up and CMP with a count down and BPL!

And I've fixed the hi-score issue whilst I'm there, some bug I introduced yesterday when trying to make it easier to set characters at a row/column

06/04/2021

Think I've definitely fixed the msb raster line bug now.

I'd like to figure out the format of the tile data.

I guess I can stop when we write to the first character of screen memory and see where we are.

So - it looks like it's here:

```
6931  $1b13: MainGame6934
6931  $1b13: LDX LABEL6532                    174 132 25     ae 84 19
6934  $1b16: SelfModifyAddress3
6934  $1b16: LDA 65280,Y                     185 0 255      b9 00 ff
6937  $1b19: SelfModifyAddress4
6937  $1b19: STA 65280,X                     157 0 255      9d 00 ff
6940  $1b1c: INY                        200          c8
```

By debugging, it looks like the screen is drawn rows at a time:

```
ABBBB&&
```

Here we have 1.5 tile rows drawn. Let's carry on debugging until the whole row is drawn

We seem to draw one row of tiles, then we fill in the countdown graphics:

```
ABBBB&&&&&&&&&&BBBBC
F[SSSSSSYSSSSSSSS[F
FSSSSSYSYSSSSSSSSSF
FSS[SSYSSSSSSSS[SSF
FSSSA&&&&&&&&&CSSSF
```

...

Then 2 rows later:

```
ABBBB&&&&&&&&&&BBBBC
F[SSSSSSYSSSSSSSS[F
```

```
FSSSSSYSYSSSSSSSSSF
FSS[SSYSSSSSSSSS[SSF
FSSSA&&&&&&&&&&CSSSF
FSSSF          LLXLM     ABBBBJ   I&&&&&
HSSSF          LSSSN     F[SSSS\I]SSSSSS
SSSSF          LSSSN     FSSSSSGSKSSSSSS
GSSSF          LSSSN     FSS[SS_E^SSSSSS
FSSSF          MNNNN     FSSSAJ   I&&&&&
-EEE-                    FSSSF
-###-                    HSSSF
-###-                    SSSSF
-###-                    GSSSF
-III-                    FSSSF
```

.....

Odd way of doing that Richard…

07/04/2021

Let's see if I can figure out how to set infinite lives by removing the DEC command when we die.

We start off with 7 men.

There are two places where we set a value of 7:

```
LDA_Immediate(7);
STA_Absolute(LABEL2564);
```

```
STA_Absolute(LABEL5721);
LDA_Immediate(7);
```

We can stop in the debugger to figure this out by stopping where we write the number to the screen:

```
Displaying 40x25 screen at $7400:
ABBBB&&&&&&&&&BBBBC   .
F[SSSSSSYSSSSSSSS[F
FSSSSSSYSYSSSSSSSSSF  .
FSS[SSYSSSSSSSSS[SSF
FSSSA&&&&&&&&&&CSSSF        .
FSSSF          LLXLM     ABBBBJ   I&&&&&
HSSSF          LSSSN     F[SSSS\I]SSSSSS
SSSSF          LSSSN     FSSSSSGSKSSSSSS
GSSSF          LSSSN     FSS[SS_E^SSSSSS
```

```
FSSSF          MNNNN     FSSSAJ   I&&&&&
-EEE-    .               FSSSF
-###-                    HSSSF
-###-          .         SSSSF
-###-                    GSSSF
-III-                    FSSSF
FSSSD&&&&&&&&&&&&&&&&&&&&ESSSF
FSS[SKdefGSSSSSSSSSSSSSSS[SSF
FSSSSK   GSSSSSSSSSSSSSSSSSF
F[SSSKfedGSSSSSSSSSSSSSSSSS[F
DBBBB&&&&&&&&&&&&&&&&&&BBBBE
........................................
XXXX"#lg$%XXfjlg"#@a12$%12$%fgXX(jxyXXXX
XXXXUUmi&'XX*>miUUbcXY&'XY&'pqXX)kXUXXXX
GGGGGGGGGGGGGCCCCCCCCCCCCCCCCCCCCCCCCCC
LLLLLLLLLLLLHHHHHHHHHHHHHHHHHHHHHHHHHH
```

Rows 21/22 columns 34,35 offset = 874 address=30570 776A

I think it might be this:

```
DEC_Absolute(LABEL5292);
```

Seems to start off at 22

Yes - that was it… if I comment that out, the men remaining stays at 7. Not sure why the odd values, but presumably some offset into large text

Anyway, I can now do this:

```
if (!infiniteLives){
  DEC_Absolute(LABEL5292);
}
```

This will allow me to debug later levels more easily.

I've just turned on infinite lives and done a complete run through of the game and got to the end. I was pleasantly surprised that there were no bugs and I managed to complete the game. So the fix to the 8th bit of the raster line seems to have greatly helped.

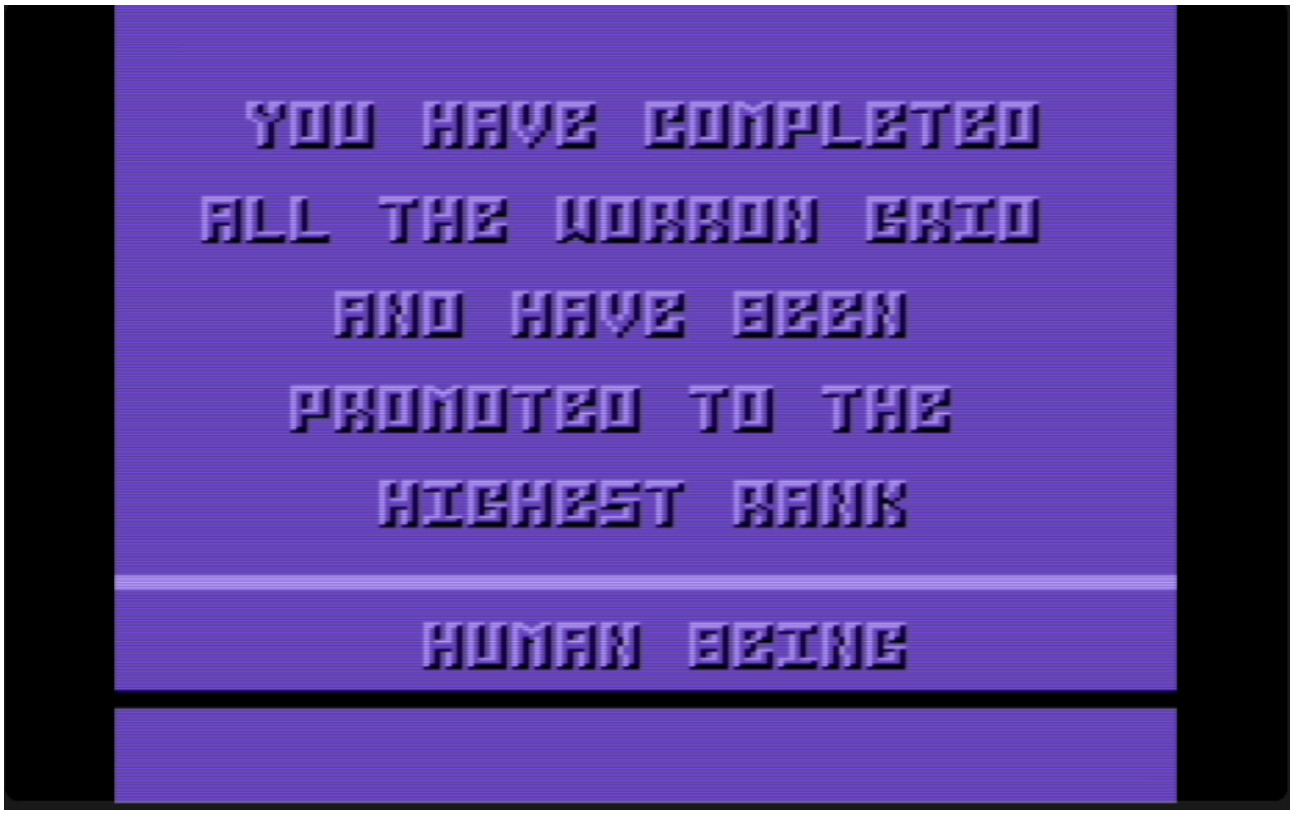
When the game is completed, there is a message:

YOU HAVE COMPLETED
ALL THE WORRON GRID
AND HAVE BEEN
PROMOTED TO THE
HIGHEST RANK

HUMAN BEING

Where is this text?

YOU HAVE found at index:5554
PROMOTED found at index:5615



09/04/2021

Today I've started to replace some of the calls to the kernal rom routines. I only seem to use a couple and these are to do with setting up the raster irq and something to do with colour ram that I don't fully understand.

Once I've bypassed these, I can completely switch out the kernal rom. Not sure I need the memory, but I'm doing it more for purity reasons and because the code is made more complex. I now know better how to more easily set up a raster interrupt and it isn't like I did it in 1986!

11/04/2021

I'm going to add my music routine from my other game in progress to the Worron title screen. I'll use the temp music on that initially. To get this working, I need to do the following:
- Bring over the random page of data that I use to drive parts of the routine
- Add frameLsb which increments once per frame and is used for timing
- Add the music data under the 4k section under the chips at D000

I think before I do this, though, I perhaps need to completely get rid of the use of the Kernal ROM routines, then I'll have more scope. So let's do that first of all. Hopefully I won't have any raster timing issues, but I'm expecting I will as I'm not entirely sure how the

raster bars on the title screen are so smooth already. I didn't know much about timing in 1987… or maybe I've forgotten that I did?

So I've got one ROM routine left that I appear to be calling several times:

```
static public StaticWord kernalROMOutputCharOnCurrentDevice =
add(65490, "OutputCharOnCurrentDevice"); // $FFD2
```

What does this do?

I've found something here:

https://www.atariarchives.org/mlb/chapter7.php

This example is part of the Commodore kernal.
There is a trick to the way this sort of table works. Notice that each member of the table begins with 4C. That's the JMP instruction and, if you land on it, the computer bounces right off to the address which follows. $FFD2 is a famous one in Commodore computers. If you load the accumulator with a number (LDA #65) and then JSR FFD2, a character will be printed on the screen. The screen location is incremented each time you use it, so it works semi-automatically. In other words, it also keeps track of the current "cursor position" for you.

So I appear to be using it to write to the screen and increment some pointer. So I must be using it to write characters to the screen memory I'm guessing.

These are my use-cases:

Before we get to that, I should be able to get rid of this now that I've changed the way we start/stop the raster routine:

```
final int
OperatingSystemAndBasicPointersRAMExecutionAddressOfInterruptServi
ceRoutineLsb = 788;
final int
OperatingSystemAndBasicPointersRAMExecutionAddressOfInterruptServi
ceRoutineMsb = 789;
```

Actually, I can't get rid of this as I think we're relying on the basic raster routine to direct us to our raster line. We need to remove our dependency on the other ROM routine first of all, then we can clean this up.

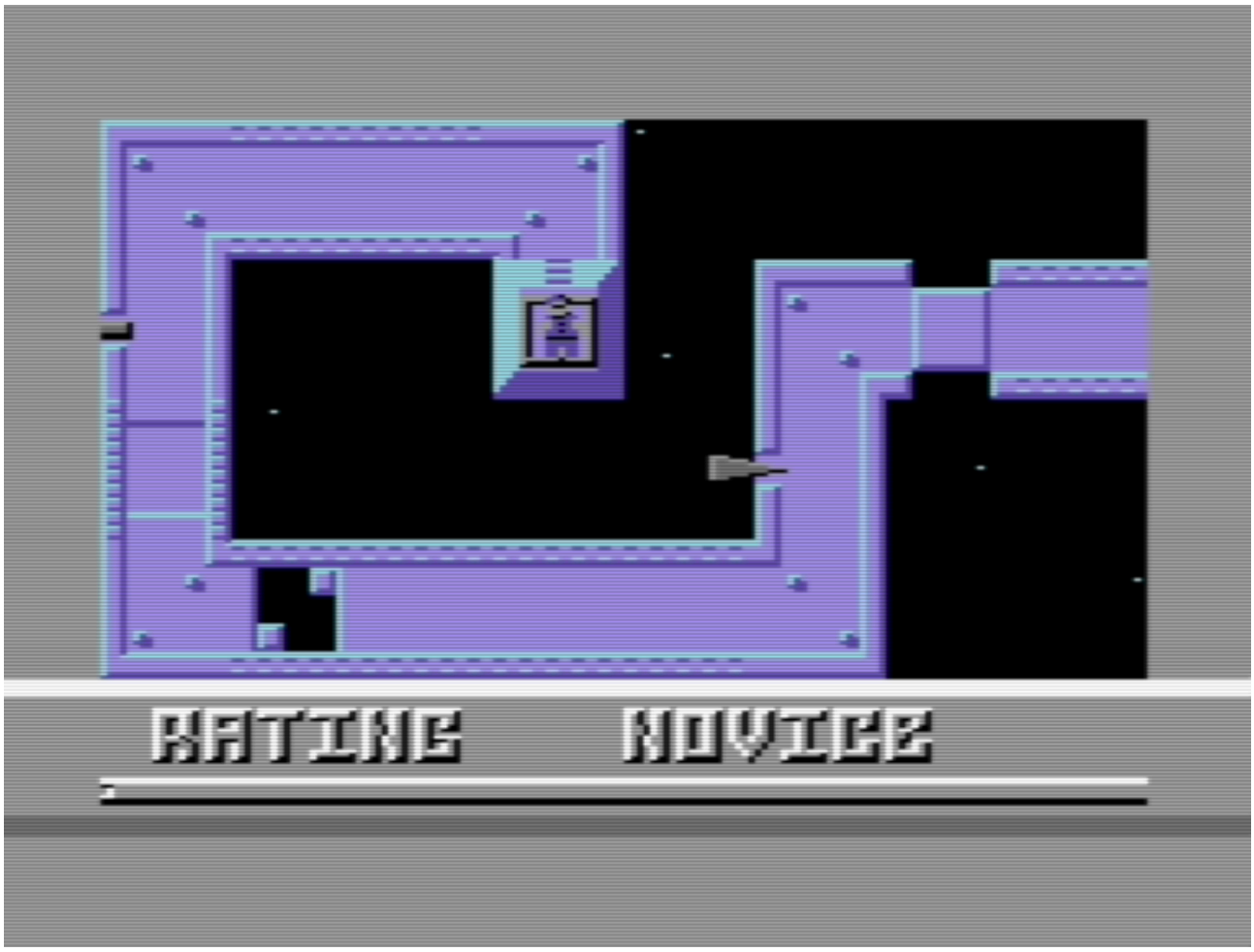
Got distracted as found the data that defined the level bas-relief colours, so have defined a Level class in java along with a BasRelief class. There were only 4x combinations of colour, so I've added a further 4x.

I've made sure these are spread out interestingly across all the levels:

```
final BasRelief basRelief0 = new BasRelief(yellow, orange, brown);
final BasRelief basRelief1 = new BasRelief(cyan, lightBlue, blue);
```

```
final BasRelief basRelief2 = new BasRelief(yellow, lightRed,
brown);
final BasRelief basRelief3 = new BasRelief(white, lightRed,
brown);
final BasRelief basRelief4 = new BasRelief(white, lightGreen,
green);
final BasRelief basRelief5 = new BasRelief(white, lightGrey,
grey);
final BasRelief basRelief6 = new BasRelief(white, grey, darkGrey);
final BasRelief basRelief7 = new BasRelief(white, yellow, orange);
```

We also no longer start with such a muddy colour, but a nice blue:



It's still entirely unclear to me what this kernal function does.

When we step into it, I that it does an indirect JMP to the address held at:
806

Which is f1ca = 61898

I bet this is some clear screen routine. No - it outputs a character to the screen.

It loads some value from 9a = 154 which appears to be a value of 3… this is some default value that I've not set.

According to the memory map guide, this is a value of 3 = screen.

It then JMP to e716 with the value we passed in (147) in A:

To be honest, I've no bloody idea what this is doing. I'm tempted to just comment it out and fix it up as I see fit. Maybe then I'll learn what it does.

I think I've sussed it now… these routines were just about placing characters on the screen and setting colour memory. I'm sure they were related to clearing the screen, Though I'm not entirely sure how. No matter, I've re-written the code myself and am ready to remove the kernal rom completely now.

There is one thing left to sort before we can turn off kernal rom, which is:

```
void setUpRasterInterrupt(final String RasterRoutine) {
  final int
OperatingSystemAndBasicPointersRAMExecutionAddressOfInterruptServi
ceRoutineLsb = 788;
  final int
OperatingSystemAndBasicPointersRAMExecutionAddressOfInterruptServi
ceRoutineMsb = 789;

  LDA_Immediate(reference(RasterRoutine).lsb());

STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineLsb);
  LDA_Immediate(reference(RasterRoutine).msb());

STA_Absolute(OperatingSystemAndBasicPointersRAMExecutionAddressOfI
nterruptServiceRoutineMsb);
}
```

12/04/2021

So - I still need to rip out the old kernel ROM raster routine. I feel like I need to understand a little better what is currently going on. For starters, Worron makes use of sprite collision detection, so I need to make sure I don't accidentally turn that off as it is part of the interrupt mechanism. So if I explore how that works initially, before I go much further.

In passing, it looks like I've introduced a bug on the high-score screen:

This will be because of my changing the clear screen routines.

It looks like sprite collisions are used to enter the portal, so I need to turn off sprite collisions other than the portal. The portal can be multiple sprites especially on later levels where there are more than one portal and or where the portal might be on the same screen.

I've made the man flash different colours when he explodes, which I think is a nice improvement.

It would be nice to add a sound fx here for teleporting.

Something I've been wondering about recently is whether the code to detect background collisions is still in place. I should look for a read of the screen memory in the debugger around about where the man is walking. This is something that I must have originally switched off when I took the disks up to Paranoid, most likely because the game was so fiendish without doing that. But I know the code to collide with all the background elements is in there somewhere.

I think row 2, cell 2, should do it as this is the middle character on the top-left tile.

= 29696 + (40*2) + 2 = 29737 = 0x7429

The character that is in this corner is: 1b = 27

If we tread on this character, when it is in a certain state, we should die. So there must be something in the code where we store 27. So let's look for that.

There is this:

```
array(WorronGrid2Array, 1, 23, 15, 18, 18, 15, 14, 27, 7, 18, 9,
4, 28, 0, 0).isPageAligned(false);
```

I think that's it… so I should figure out what this does.

This 27 ends up being here in the compiled code:

5692 $163c: 27 27 1b

So let's see what reads that address?

Mmm.. nothing doing there.

So looking at the character set, the animated characters seem all over the place, so I'm sure I'd have to have defined an array of characters that we want to check against or treat as 'death characters'

I need to find another character.

I think 27.28.29,30 Let's do a search for 28

I think I've found the code… I was being too logical… of course, the Richard Paynter of 1986 would have just written the code out long hand rather than putting it in tables:

```
// This looks like the routine that would check for character
collision, yet nothing jumps here
// Maybe this is because I commented out the call to this code
label("CharacterCollision");
LDY_Immediate(1);
CPX_Immediate(27);
BEQ_Relative(MainGame9608);
CPX_Immediate(224);
BNE_Relative(MainGame9634);
label(MainGame9608);
LDA_Absolute(LABEL6285);
CMP_Immediate(1);
BEQ_Relative(MainGame9634);
CMP_Immediate(0);
BEQ_Relative(MainGame9634);
CMP_Immediate(2);
BEQ_Relative(MainGame9634);
CMP_Immediate(3);
BEQ_Relative(MainGame9634);
CMP_Immediate(4);
BEQ_Relative(MainGame9634);
```

```
STY_Absolute(SpriteCollisionDetected);
label(MainGame9634);
CPX_Immediate(28);
BNE_Relative(MainGame9641);
JMP_Absolute(MainGame9693);

label(MainGame9641);
CPX_Immediate(29);
BNE_Relative(MainGame9648);
JMP_Absolute(MainGame9693);

label(MainGame9648);
CPX_Immediate(30);
BNE_Relative(MainGame9655);
JMP_Absolute(MainGame9693);

label(MainGame9655);
CPX_Immediate(31);
BNE_Relative(MainGame9662);
JMP_Absolute(MainGame9693);
```

etc.

These look like our characters

Looks like X would contain the character this is underneath the man.

So I guess I need to find the routine that loads that into X then I'll most likely find where I might have called this code.

To confirm this, let's check this code:

```
CPX_Immediate(185);
BEQ_Relative(MainGame9765);
CPX_Immediate(186);
BEQ_Relative(MainGame9765);
CPX_Immediate(187);
BEQ_Relative(MainGame9765);
CPX_Immediate(188);
```

And see if these characters look like animated characters?

There are only 2 pair of lsb/msb arrays where we read from the screen memory, so it has to be one of these two.

Our options are:

```
final String TileScreenMemoryMsbArray =
"TileScreenMemoryMsbArray";
final String TileScreenMemoryLsbArray =
"TileScreenMemoryLsbArray";
```

Which I believe is used to draw the screen, or:

We know this code works as we're not allowed to steer around certain characters. It is just the collision part of this that doesn't work.

Maybe we can look for a reference to a wall character?

So 38 is a wall character, we we must have code that stops us running into that.

Actually - space is not the background character, so what is the normal background character that we can run over?

= d3 = 211

So this definitely the character as if I paint the screen with this, then I can walk anywhere I want.

I can't find any reference to this.

I can't see any code that explicitly looks for 211. Let's confirm that by painting the screen with other characters and confirming that we can run over them.

It seems that we're prevented from walking through walls which are at characters 65-74 - surely I should be able to search for this?

Ok - I think I've found it:

```
5958  $1746: ARRAY5966
5958  $1746: 65                                   65          41
5959  $1747: 66                                   66          42
5960  $1748: 67                                   67          43
5961  $1749: 68                                   68          44
5962  $174a: 69                                   69          45
5963  $174b: 70                                   70          46
5964  $174c: 71                                   71          47
5965  $174d: 72                                   72          48
5966  $174e: 73                                   73          49
5967  $174f: 74                                   74          4a
5968  $1750: 38                                   38          26
5969  $1751: 45                                   45          2d
5970  $1752: 46                                   46          2e
5971  $1753: 47                                   47          2f
5972  $1754: 52                                   52          34
5973  $1755: 53                                   53          35
5974  $1756: 84                                   84          54
5975  $1757: 85                                   85          55
5976  $1758: 86                                   86          56
5977  $1759: 88                                   88          58
5978  $175a: 89                                   89          59
5979  $175b: 90                                   90          5a
```

5980	$175c: 91	91	5b
5981	$175d: 92	92	5c
5982	$175e: 93	93	5d
5983	$175f: 94	94	5e
5984	$1760: 95	95	5f
5985	$1761: 96	96	60
5986	$1762: 97	97	61
5987	$1763: 98	98	62
5988	$1764: 99	99	63
5989	$1765: 100	100	64
5990	$1766: 101	101	65
5991	$1767: 102	102	66
5992	$1768: 106	106	6a
5993	$1769: 203	203	cb
5994	$176a: 204	204	cc
5995	$176b: 205	205	cd
5996	$176c: 206	206	ce
5997	$176d: 207	207	cf
5998	$176e: 208	208	d0
5999	$176f: 209	209	d1
6000	$1770: 210	210	d2
6001	$1771: 114	114	72
6002	$1772: 112	112	70
6003	$1773: 119	119	77
6004	$1774: 120	120	78
6005	$1775: 117	117	75
6006	$1776: 118	118	76
6007	$1777: 124	124	7c
6008	$1778: 125	125	7d
6009	$1779: 126	126	7e
6010	$177a: 135	135	87
6011	$177b: 136	136	88
6012	$177c: 137	137	89
6013	$177d: 154	154	9a
6014	$177e: 199	199	c7
6015	$177f: 236	236	ec
6016	$1780: 237	237	ed
6017	$1781: 238	238	ee
6018	$1782: 239	239	ef
6019	$1783: 121	121	79
6020	$1784: 122	122	7a
6021	$1785: 198	198	c6
6022	$1786: 182	182	b6
6023	$1787: 183	183	b7
6024	$1788: 197	197	c5
6025	$1789: 198	198	c6
6026	$178a: 199	199	c7
6027	$178b: 200	200	c8
6028	$178c: 252	252	fc
6029	$178d: 252	252	fc
6030	$178e: 253	253	fd
6031	$178f: 254	254	fe

6032 $1790: 255 255 ff

Let's check whether these are all wall characters? Most of them seem to be… so let's go with this and find our where this array is used.

Found it! This variable is used to store the character under the man:

6033 $1791: CharacterUnderMan

Now we're getting somewhere.

13/04/2021

Up early, so I'm going to pull in the 75 character wall index, sort it and define it in code rather than a data file. Sorting it will make it easier to see what the characters are rather than just in a jumbled array. I've also doubled the performance of the code that checks against this list with a couple of simple improvements to the code that were beyond my teenage self, clearly.

I've removed duplicates so it is now of length 72 rather than 75.

```
final int[] wallCharacterArray = new int[]{38, 45, 46, 47, 52, 53,
65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 84, 85, 86, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 106, 112, 114,
117, 118, 119, 120, 121, 122, 124, 125, 126, 135, 136, 137, 154,
182, 183, 197, 198, 199, 200, 203, 204, 205, 206, 207, 208, 209,
210, 236, 237, 238, 239, 252, 253, 254, 255};
```

So now that I've discovered this:

```
LDA_Absolute(CharacterUnderMan);
```

I should be able to call the routine that checks the background collisions against holes in the floor etc.

I need to look for a collision with character 38 which is an electrified wall, so not part of the above.

So 38 is part of the wall character array:

5958 $1746: WallCharacterArray
5958 $1746: 38 38 26

Of course it would be as I'm prevented from walking through it. Is is supposed to be something that kills us though? Not sure… let's search for something we're definitely sure should kill us instead.

So the crumbling ground on the first screen is character 0x23 = 35

Yay - getting there… I've managed to trigger dying if we run over some characters, although only if we're moving.

We're getting some good clashes with background characters now. Something that is not right is that the animated wall I can walk through. I think we should not make this fatal and should add it to the list of
Wall characters.

Character 38

That was my fault when I changed the code:

```
// Iterates over all wall characters to see if the current
character is one of these
LDX_Immediate(wallCharacterArray.length - 1);
LDA_Absolute(CharacterUnderMan);
label(WallCharacterArrayLoop);
CMP_AbsoluteX(WallCharacterArray);
BEQ_Relative(IsAWallCharacter);
DEX_Implied();
BNE_Relative(WallCharacterArrayLoop);
```

Should have been:

```
// Iterates over all wall characters to see if the current
character is one of these
LDX_Immediate(wallCharacterArray.length - 1);
LDA_Absolute(CharacterUnderMan);
label(WallCharacterArrayLoop);
CMP_AbsoluteX(WallCharacterArray);
BEQ_Relative(IsAWallCharacter);
DEX_Implied();
BPL_Relative(WallCharacterArrayLoop);
```

Ok - fixed that.

Let's add in the random array now which we'll use for the music routine and probably the man dying. We also need to add frameLsb which we increment during the game.

We also need to find the code in the vertical blank in the game.

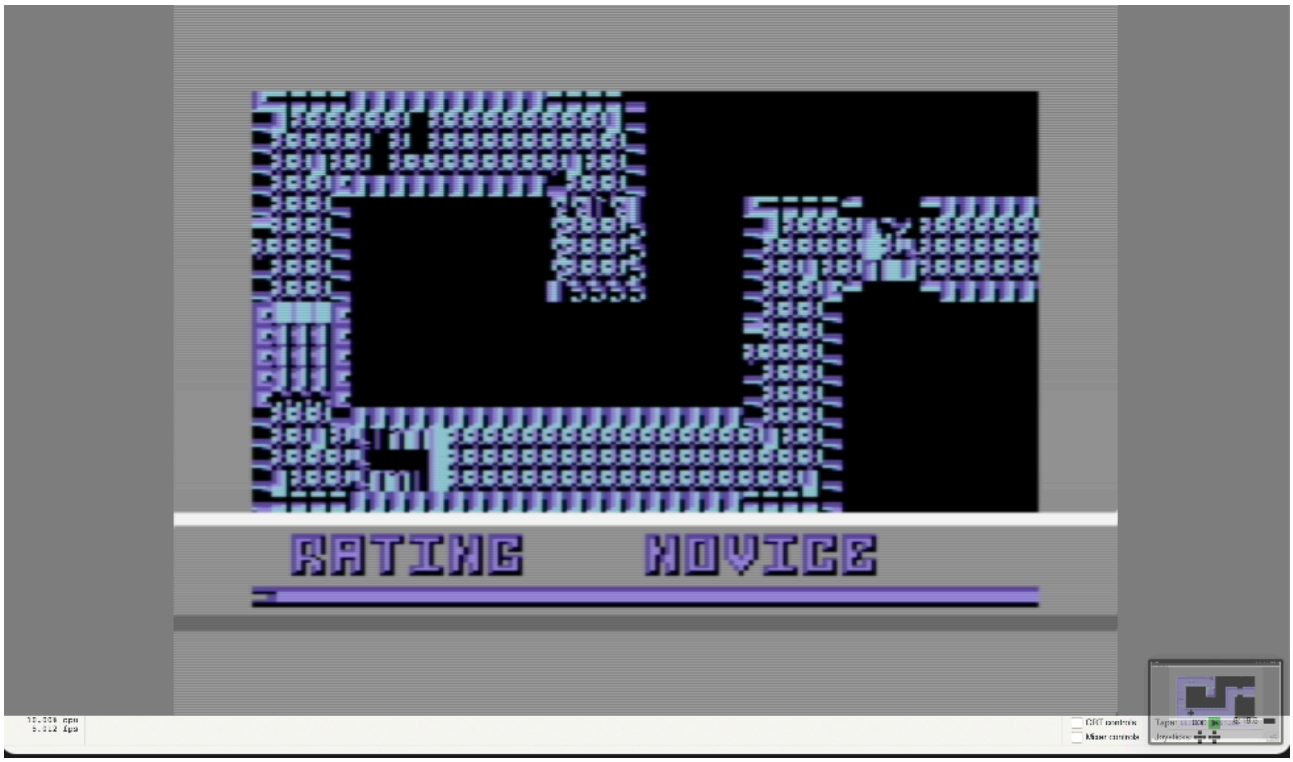
We have these raster lines:

```
array(MainGameRasterLineArray, 210, 215, 249,
255).isPageAligned(false);
```

255 is what we're after.

14/04/2021

I want to fix a glitch when the man dies. We end up with an ugly frame or two where it looks like we're not showing the correct character set:



There are 2x characters sets being displayed here as the large text is separate to the game area, so I need to track down where we set these character sets up.

This is done via:

```
static public StaticWord vicMemorySetupRegister = add(53272, "MemorySetupRegister"); // $D018
```

Whoops - the high score screen appears to have disappeared!

So this bug above is caused - I think - by not calling the code to set the above register. The raster lines are triggered, but I suspect that we don't branch into the right code. So I need to find where we change the character set.

15/04/2021

I did some work to simplify the way we set up the vic chip for things like character set etc. I'm going to take that further and create an array of size 4 for these values which should hopefully remove the bug from above.

Things are improving and looking smoother. However, I need to find the code where we first draw the screen, so I can set up the level colour scheme at that point.

Found where I should do that… the transition between each level is now much smoother.

16/04/2021

Time to return to setting up the raster interrupts in a simpler fashion. There is too much going on at the moment with CIA timers and the like.

Let's see if I can get rid of some of those first of all:

```
LDA_Absolute(cia2PortADataDirectionRegister);
ORA_Immediate(0b00000011);
STA_Absolute(cia2PortADataDirectionRegister);
```

17/04/2021

This morning, I'm trying to migrate to using a non kernal raster routine. This far, I'm seeing odd artefacts:

I think this may be related to timing issues. I was using the kernal routine which did a lot of other things, so perhaps it was just the case that it happened to take an entire raster line and everything magically lined up. I can't imagine I knew enough to do that intentionally when I was 16?

Let's try adding some NOP

```
void startRasterInterrupt() {
  if (useNewRasterApproach) {
    STA_ZeroPage(rasterInterruptOldA);
    STX_ZeroPage(rasterInterruptOldX);
    STY_ZeroPage(rasterInterruptOldY);
    INC_Absolute(vicInterruptStatusRegister);
    // Add some NOP so that we line ourselves up with the next
raster line
    for (int i = 0; i < 5; i++) {
      NOP_Implied();
    }
  }
}
```

That's definitely moved the issue along… just need to experiment with the right number of NOP

This is 12:

And finally 13 brings us back to normal… excellent… I've now completely switched off the ROM. Perhaps the ROM routine handle bad raster lines for us? Maybe.

There are still some artefacts on the rainbow rasters, which might be caused by looking at arrays that cross page boundaries, perhaps?

I should use my new Block technique which will put arrays into their own page and ensure that code never crosses page boundaries when indexing. On the other hand, remember that this is not the aim of this exercise!

So there are a couple of things to still fix up before I allow myself to start thinking about sound.

- I want to move arrays into a block so that we have a consistent access to them and we don't find this moves around.
- I want to fix the glitch on the rainbow effect on one line
- I want to fix the high score screen which I appear to have broken somehow.

I've also noticed that I've lost the border sprites… this is probably timing related

This will be annoying to fix as I can't remember the trick… it was something to do with blanking the screen… ah - that'll be it then… I removed some of that code.

I think this is related to the high-score issue as if I blank the whole screen, I get the same effect.

Fixed it… phew… now to add all arrays to a block

It's a bit of a mess this code, but my aim isn't to re-write it, so let's just work with its limitations. I've spent 35 years programming since then, so I'm bound to be critical haha. I will patch over things with overrides rather than trying to re-factor too much. The code is too cut/paste so any re-factoring will be doomed to failure. We're not talking a professional here :)

The high score screen is working again, mostly, though I need to remove the GAME OVER text.

18/04/2021

I've fixed the raster flicker on the title screen, but my efforts to pack arrays more efficiently has made the game crash after a few levels. I really think I need to back-pedal from My block array. Remember, Richard, this is not about improving the code of Worron, merely about patching up the bugs. So let's roll back to where we were. Stop being a purist!

The raster flicker was correct by doing:

```
label(TitleScreenHighScoreRasterRoutine);
startRasterInterrupt();
LDX_Absolute(TitleScreenRasterColourIndex);
LDA_AbsoluteX(TitleScreenRasterBorderColourArray);
LDY_AbsoluteX(TitleScreenRasterBackgroundColourArray);
cycleTiming.delay(23);
STA_Absolute(vicBorderColour);
STY_Absolute(vicBackgroundColour);
```

I want to work out how to jump straight to the game complete message, so I can short-cut to that to test things out.

So I've seen 100 being mentioned a few times around the code base and I think it is because there are 100 screens. I should start to model the screen number for the start of a level so that I can more easily jump around the game to test things out. Let's play through the game and capture this in java.

I've figured out how to make any level act as the final level and jump to the end of game message, which helps.

I'm now going to clear the screen before the high-score.

Next I'm going to rip out the 1980s attempts at sound on the high-score screen.

Done that.

I need a break from coding, so I'm going to start trying to get some music ideas together in Logic Pro for the Worron music. I quite fancy having something in 7/8. A lot of the 80s SID musicians were into their 70s prog as were about 10 years older than myself, so in homage to their love of that music, I'll select a typically proggy time-signature. Plus it will help convey a sense of anxiety as will be hard to tap your foot to!

It won't sound like some of my other music, that's for certain:

19/04/2021

So I wrote a sketch in Logic Pro for my Worron main game music and I'm pretty pleased at the way it turned out.

I think the next thing I should do is get the music routine I wrote last year wired into the game along with the temp music I wrote whilst developing that music routine.

Then I can start importing the new music.

We'll put the music data under the chips as there is a spare 4k there if you switch out the chips via address 1.

YES! I've managed to get the music routine working… at last there is some music in Worron… next I need to program in the Worron music.

There is something amusing about storing SID data under the SID.

20/04/2021

I've managed to find a place on all 3x raster routines to call my new frame routine, including the music. I've played through the various combinations and generally the transition is smooth and I don't hear any breaks in the music, apart from the start of the game which is not only a bit visually clumsy, but also creates audio jitter. I think we can improve what happens here. I wonder where we are checking for the fire button being pressed? I should examine that next as I think this would be a good improvement.

So the first time I press fire, I'm on raster: 189

The next time: 293

So we are starting the game at some random raster position, which won't help a smooth transition.

And I've fixed that as well… my dummy raster routine that I added between the title screen and main game, I've made that call the new routine. It all seems rather seamless now, which is great.

I can now start replacing the temp music with the new Worron score. For the moment, I'll key in the score I came up with in Logic Pro the other day, though I envisage having a couple more tunes for the high score + finishing the game. Maybe even a different tune for the title screen. We'll see.

I may even dabble in fx… I'm not sure… I feel like an imposter, producing music that is starting to sound like some of the - albeit - poorer SID tunes of the day. But it is leaps and bounds above what my 17 year old self could have dreamed of producing.

21/04/2021

More work on the Worron soundtrack… just playing around with bass sounds. I think I may need to revisit the hard-reset SID issue as I'm sure I can hear some artefacts in there. Not sure yet.

It's fun to hear some original music finally start to play over the game after all this time, even if it is only 4 bars or so on repeat.

22/04/2021

A Track is formed of multiple Patterns. However, each Pattern could be a predefined unit that we want to be transposed, so we can re-use parts. So we should have the idea of a TrackPattern which is a Pattern + it's Notes, with a transpose value.

Ok - done that.

23/04/2021

I'm making good progress on the Worron main theme… I've started to key in the first 8 bars or so and have improved the music routine so that each pattern can be transposed when part of a different track.

24/04/2021

I'm going to change my instrument java code so that pitch/pulse/wave events are local to an instrument. We can do some processing when encoding it if there is some commonality.

In this way, it means that I can alter instrument data without having to worry about how it is affecting other instruments. If there is shared commonality, we'll determine that once we've compiled the data. We can then point at the same shared data arrays.

I'd like to revisit the names I use for the music structures.

From the top:

TrackArray (There are multiple tracks, such as title music, game music, high score music, end of level music)
- TrackVoiceArray (Each track has 3x voices)
- - TrackVoicePatternArray (Each track and voice has a number of patterns, each pattern being transposed. This allows us to reuse patterns cheaply at a different pitch)
- PatternArray
- - NoteArray
- InstrumentArray
- WaveEventArray
- PitchEventArray

Let's re-factor the code to reflect this.

I think to prove that we've done this, we should use a different track of some temporary 1 bar of music for the title screen.

Excellent… I've managed to fix the music routine so that I can now change a single track value and switch between 2x tracks.

So I've set up dummy tracks for each of the following:

```
final public Track trackTitleScreen;
final public Track trackMainGame;
final public Track trackCompletedGrid;
final public Track trackGameOver;
final public Track trackGameCompleted;
final public Track trackHighScore;
```

The next thing I want to do is configure whether a track repeats or just plays through once and stops? Right now, it loops around, so let's do that.

So I've now added 7 tracks as I've added an additional track for when the man dies as a sound effect can just be expressed as a track.

Good day's work I think.

25/04/2021

Going to add a new temp track for the start and end of the level when the portal open and closes.

Ok - I've managed to wire in the temp track for when we leave the level, but now need to find the entry.

Done that too now. It's funny to see sync'd up sound after all these years.

So what sort of actual sound do I want when the portal opens? Some kind of pitch up and pitch down noise would work.

Whilst doing this I've also noticed a bug that it the joystick moves the moment we start the level, the portal animation does not occur, so I need to fix that.

This means the game starts with the title track still playing.

Fixed it.

The next thing I want to fix is that the transition from title screen to the first level is messy and the screen is left with artefacts on it whilst we construct it.

I think if we change the colour scheme before drawing the game area, that would be better. Possibly blanking the screen too.

Let's add a variable that we'll use to blank the screen.

26/04/2021

First thing I'm going to do this evening is make the man flash in colour when on the portal to give the impression of appearing/disappearing.

There is a small pause at the start of the game before the portal opens, which I'd like to remove.

Yay found it and fixed it. Looks much better now.

So the next thing is to get the portal open and close sound working.

So I've got that working. I'm thinking that I'd like to do the end of level music next.

Actually, I quite like the temp music I used, with a bit of refinement.

I think I've got an issue with my music routine about the way it uses the gate off. I need to re-think how this works.

The crux of the matter is that if we are moving from one ADSR to another, then we need to set the gate bit to zero, 2x frames before we apply the new ADSR to avoid the classic ADSR bug in SID.

It strikes me that we could automate this if we knew the ADSR that was coming up. Or… if we double buffer the music data, which might be an alternative?

Actually, it's not that simple as we might want to restart our current ADSR. We have to make this an explicit, controllable thing I think.

So each note should either:
- Continue into the next note without affecting the gate bit
- Trigger the gate bit for the last 2x frames of the note.

Basically, we only want to do this when we're playing a note that has the gate bit on. Maybe we need to annotation an instrument as being legato. If an instrument is Staccato, we'll trigger the gate bit, else we'll leave it as it is.

I think I've implemented this. I might generalise this to be how many frames we want to leave a gap between notes as that might be more useful. Rather than a boolean.

We can use that to create really staccato fx. A value of zero would total legato. I think I like that.

Time to enter the notes for the main tune.

Ok - adding more notes has shown up some kind of data corruption… it seems to be because I'm storing data under the SID chip… but when I read it, I'm not seeing the values.

Maybe I can't store data under SID? No - I can. It's just a bug!

28/04/2021

I'm making good progress on keying in the music routine now. The further along I go, the more notes I can re-use. I'll come back later to refine the instruments I'm using.

Ha - my tune is now so long that I run out of time and die… will need to think about that. The tune on Logic Pro takes about 4 mins before it loops around.

29/04/2021

I was thinking about things I need to work on on the music routine and I thought of the following:
- I should put empty events on soundtrack so that they can be shared between multiple instruments
- I should add an event for filter on/off events for an instrument.
- I should add events to a Track for global events such as filter and perhaps even volume.

Before I get to that, my main game music is too long and the timer runs out, so I need to find out how to switch that off. We will only be able to hear the full track on the longer Levels where we have a checkpoint.

So I've entered all the notes for the main game music, albeit I still need to work on the instruments themselves + filters etc.

How much space has that taken up from D000+ ?

It's only 1707 bytes so far, which is pretty bloody good, given the main tune is over 4x mins long. My compression must be quite good. It helps that I've got java doing some crunching for me that would have been nearly impossible to do manually back in the day. It figures out all the unique notes etc. and assigns ids to those.

I've also worked out how to turn off the countdown, so that helps me listen to the whole tune.

So each Track has 3 TrackVoice, so it makes sense to have a TrackGlobal. Well - do I need that? That's the Track, surely? I need events on the track that cycle around. These Events need to be the same length as the voices so that they align correctly.

So what things do I want? Let's start off simple and add volume. We need these events to be able to vary.

So I have added TrackGlobal as this makes more sense given we have 3x TrackVoice.

Oh - I'd already added GlobalEvent in the past with:
- FilterResonanceEvent
- FitlerFrequencyEvent

I'm not sure I'm going to use these, so I'll leave these to one side.

We need fine control so that different parts of a track may have different filter events than others. Equally, we want to reuse part of an event, much like we have:

Track
- TrackVoice
- - TrackVoicePattern
- - - Pattern
- - - - Note

What we need is to be able to change the global events after a given duration.

Track
- TrackGlobal
- - TrackGlobalEvent
- - - duration: Int
- - - VolumeEvent
- - - FilterEvent

etc.

This works nicely. Then we need to have a certain number of events that match the track length.

We'll start by having one TrackGlobalEvent that is the same frameCount as all the notes in the Track and we'll just set the volume once to get ourselves going.
Then we can enhance it further by turning it into an event itself.

30/04/2021

How do we want to lay out these track global events in memory?

Track
- TrackGlobal
- - List<TrackGlobalEvent>
- - - VolumeEvent
- - - FilterCurve
- - - FilterSettings
- - - etc.

Each TrackGlobalEvent has a frameCount which tells us how long it lasts for before we move to the next one in the array for that track.

The frameCount must match the frameCount of the 3x TrackVoices so that they align.

We are not looking to share TrackGlobalEvent between tracks as they are unique per frameCount within a Track.

We can assign a TrackGlobalEvent a unique id within the context of the whole soundtrack, so that we can store the data in one single array for each soundtrack.

We then just need an array of TrackGlobalEvent.soundtrackIndex for each Track. So that just means we need to store, for each Track:
- An array indicating the length of the TrackGlobalEvents for that track + 2x more array for the lsb/msb of this array
- An array of that length containing the soundtrackIndex of that TrackGlobalEvent. + 2x more array for the lsb/msb of this array

01/05/2021

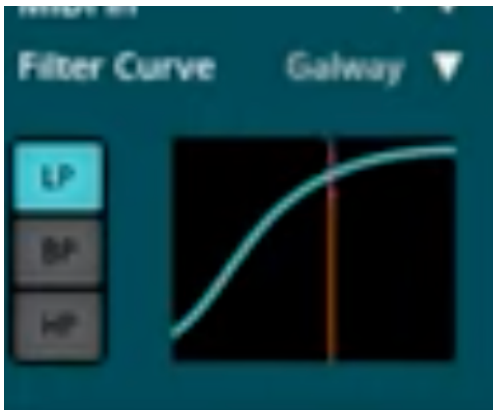
Today I'm wiring in my music routine global event track.

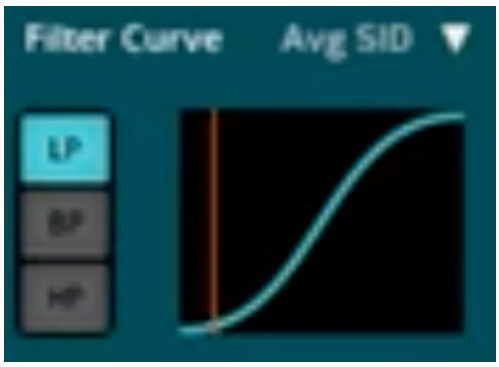
I think I'm going to make global events have a duration that is a 16 bit number.

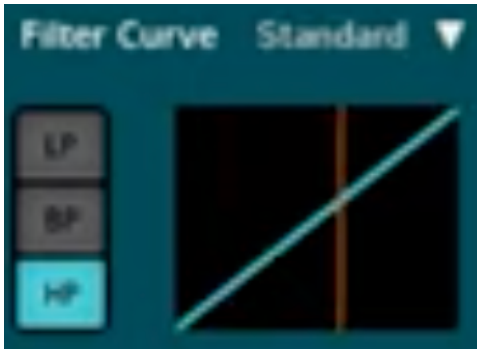
On the other hand, we could store all the data as 8 bit but apply the tempo and make it 16 bit? That might work.

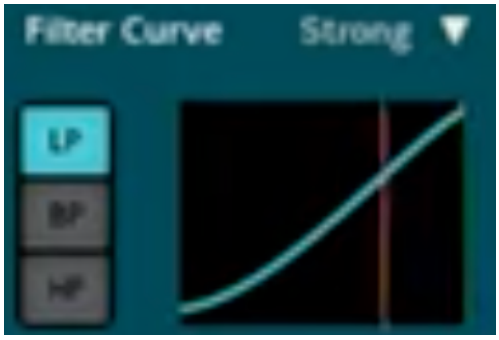
I'm going to store all note lengths as small values and create multiplication look-up tables to calculate the multiply the 4x note lengths by the tempo.

I was also thinking about filter curves. So the filter frequency is an 11 bit 0-2047 value. I was looking at the Insidious plug-in which applies various filter curves, so
This would allow us to have a filter curve index that maps to a non-linear value in a table. I'm going to add this. I captured some of the options from a YouTube video:









Galway is interesting. I think there was a Tim Follin one. Either way, I'll come up with my own curve; this is more to trigger the idea to do this.

So various effects such as a filter sweep will sweep backwards and forwards along this curve, or possibly a step filter which will move sequentially or jump to positions in this curve.

So I've made a change to that we can apply a tempo multiplier to note duration. We will need to do the same to the global event duration. So we'll need to build a 16-bit multiplier table for each unique value of duration. There are 4x of these for each note duration and there will be others for each unique value of global event duration. Because we store duration values as the smallest value we can, tempo has to be at least 5 before it starts sound unhurried. A tempo of 1x would be 1x frame per note, which is ridiculous as are tempos slightly above this.

So if we did the obvious thing and multiplied in a loop, we'd have to go around at least 5 or 6 times before we get to a range we're happy with. I'm happy that we store duration as very small values as this allows us to have long notes expressed in small numbers. But we need to get to the actual duration quickly.

We've already done this for note duration and I'm happy with that implementation. Maybe the solution for global events is to do the same things as with notes. We'll create a unique value of duration and given that an index. That way, each duration is expressed as a duration.

So we'll implement NoteLength to have a Duration object internally and the same with a global event. That way, we can point our notes + global events at a Duration index. From there, they can easily find multiplication tables based on tempo.

Done that…

```
54851 $d643: SoundtrackDurationArray
54851 $d643: 1                          1          01
54852 $d644: 2                          2          02
54853 $d645: 4                          4          04
54854 $d646: 7                          7          07
54855 $d647: 56                         56         38
54856 $d648: 112                        112        70
54857 $d649: 14                         14         0e
54858 $d64a: 84                         84         54
54859 $d64b: 28                         28         1c
```

We can now generalise the NoteDuration code to work against Duration.

First, we need to change the encoding of a NoteLength to go via a Duration.

Actually, we can get rid of NoteDuration altogether as no longer serves a purpose and can just be replaced with Duration.

So I think I've got the basic global event stuff working and looping around correctly in time with the 3x voices. I can now change volume using the global events, which is a good start.

From experimenting, it seems that an audible pop is created when we change the volume, but not when we change the value of low/band/high pass filter enable flags.

So what are the things I want to control via a global event?

The most important thing to control is the frequency cutoff. I want to be able to have things like step filters where we jump around various cutoffs, so jumping to absolute values.
I also want the frequency cutoff to be able to move from that first absolute value in a direction over a number of frames.

Our filter curve will allow us to express the frequency as a single byte value, though we'll have a lot less than 256 steps I think.
How about the following for each event:
- filterCurveIndex
- duration

This will allow us to program in sweeps

I think I also want to control whether an instrument has the filter enabled, via the global event.

04/05/2021

Did some more work on the music routine

05/05/2021

I want to get rid of the flicker when the game is started. This is because we check for the joystick fire press at any random raster line, so we need to move that check to a raster interrupt, probably at the start of the vblank.

06/05/2021

I think I need to add duration to wave events

I've also added a great sound for the end of the level where we have random bleeps whilst we're transporting. Happy with that.

I need to add tempo to the Track rather than hard-code it.

For maximum flexibility, I think tempo needs to be on the global event.

07/05/2021

Going to remove the delay event from an instrument as doesn't make any sense. If I'm going to introduce some kind of delay, then this should be at the track level, not instrument level.
I also think I can remove the exponentialBend part of a pitch event. Not sure what it does exactly, so I'll remind myself first, but it feels redundant.

I've removed the delay event.

Exponential bend is not even used, just added to the java class, so that should be easy to remove.

I also want to add a pitch bend dive on one of the sections, which is something I've not done yet.

```
new TrackVoicePattern(trackMainGameVoice2, patternArpeggio3, 0);
new TrackVoicePattern(trackMainGameVoice2, patternArpeggio3, 3);
```

I'd like to get to the bottom of the artefacts that are created when we first start the game.

I've got rid of the worst of the game start-up flicker… the main issue was caused by us starting a raster interrupt at the top of the screen at 40, rather than 210

Next up is the bug where if we die at the start of the next level, then we go back to the previous level. But we don't reflect that in the level count. So when I create a check point, I need to remember the level number.

To test this out, I need to be able to start on level n ideally.

Found the code that triggers that.

Basically, when we die, we need to restore certain values, such as screen, but also the grid number. Then we can add a variable to store the last grid number. We don't increment that until the portal opens on the new level.

Think I've found where the values are stored:

```
LDA_Absolute(WorronGridDigit0);
STA_Absolute(WorronGridDigit0Text);
LDA_Absolute(WorronGridDigit1);
STA_Absolute(WorronGridDigit1Text);
```

08/05/2021

I've been debugging why when we die going through a portal that lands us in space, when we go back to the previous level, the colours remain the same for the level we've just come from.

The variables seem to be set up correctly, but I think it is because of this:

```
// TODO: Need to also change colours of level at this point else
we don't see the change
LDA_Absolute(LevelIndexCheckpoint);
STA_Absolute(LevelIndex);
```

So we need to call the routine to set up the colours here and I think we're good to go.

I've also noticed that I've broken the open borders on the title screen due to my screen blanking code.

We'll come back to that shortly.

Ok - I've fixed the colour scheme issue. Now to fix the open border issue which I think I broke this morning with my attempt to blank the screen.

No - I broke this a few days ago as the video I took yesterday was also broken. Blah!

For whatever reason, we only seem to be storing a value at D011 on line 245 now.

I think this is related to the new music routine, which is running from 250-270 before we service other things such as sprite positions.

I think I've fixed it by rolling back to a previous version. Not sure what caused it, but maybe it was reading the joystick in between the 2x blank rasters?
Maybe that screwed it up? Something odd to do with reading the CIA chip maybe?

So the next bug to fix is that the man can't move if we get sent back to a checkpoint.

So I've fixed that, but the music doesn't restart when going back to a checkpoint.

Just tried to play through the game with infinite lives and it is fiendishly difficult. Is that a bad thing? Well - who is going to judge? Haha everyone. Trust me… I'm just as critical of how hard this is!

I do think we might need to start with more lives, however. A million?

I think I'm going to add a new life at the end of each level. It's the least I can do. Ok - done that. Let's check that it works if we get to the end of the first level with the maximum lives?

Ok - done that. We now seem to have an issue with an occasional crash when starting the game. It smells like an 8 bit raster line issue, so let's double check.

It might be because I've fixed the 40 vs 210 raster line issue on start-up. 40 is a valid raster line regardless of what the 8-bit value is.

The game complete music is not terribly interesting, so I'll compose something simple.

Let's use the crazy arpeggios music from midway through the main tune.

Ok - done that. So… I *think* I've done pretty much all I wanted to do, except for the bitmap image I want to add. So I'll tackle that next.

I also need to make sure that this fiendishly difficult game is actually winnable, albeit with infinite lives. I believe Scott Joplin could not play his own compositions, so there is precedence for writing impossible-to-play games that the developer cannot complete unaided. That's not an excuse, just an observation haha.

Let's add the code to display an image… need to decide where in memory this will reside first of all. There must be an 8k block somewhere?

40960 (A000) to 49152 (C000) seems to be empty, which is 8k. It should align with an 8k block. We need to fill it with junk first of all to confirm it isn't being used.

Let's try putting the colour memory 1k before that at 39936 (9c00)

09/04/2021

Out having a walk in Hyde Park this morning and it struck me that what we call VibratoEvent can be re-branded as a FrequencyTickEvent as it can be used for changing The frequency of a note to a tick of a pitch.

Vibrato is just one use case.

Let's re-brand this as FrequencyTickEvent

Before we get to that, I think we need to remove frameLsb from the music routine so that random decisions are driven by a track index internal to the music routine.
Otherwise we don't get consistency as the frameLsb will vary depending on when we started the track.

I've also fixed the pitch bend on the main game track which now sounds much better as a result of re-purposing the vibrato routine so it can be used to express slides.

I also briefly ran out of the 4k block of memory that I've allowed myself for this tune, so this seems like a good place to stop messing about.

I need to shuffle around the code to try to find an 8k + 1k block in the same page, so I figure I'll make that the first page.

This is what the memory map looks like so far:

2000-12891
<blank>
16384-16662
<blank>
18432-20792
20792 - Sprite Definition - appear to be lots of blanks here
<blank>
25087 actual start of sprite definitions, so about 5k wasted
40960-49152 - 8k free - I think some code before this is free.
We have plenty of free space in the kernal rom too. About 7k

So - in conclusion, I think I should start moving code + data from the first 16k block into kernal rom area and then some more if I need to to 40960

Ok - I've managed to move a lot of code out of the first 16k in the 2nd 16k and now have free space from:
5119-16384, so enough for 8k of bitmap + 1k of colour.

We'll put the bitmap at 8192 and the colour memory at 7168

I need to move some more code around as the VIC cannot see data from 4096-8192 as this is character rom

10/05/2021

Yesterday, I moved a lot of code around and managed to find an 8k block from 8192-16384 for the bitmap image + set the screen memory for bitmap colour to be at 3172.

I can now focus my efforts mostly on the title image + scrolling message. I'm going to leave my immature scrolling message on the title screen as it is, but I'm going to add a new, longer scrolling message at the bottom of the bitmap image which will provide instructions about the game that would have appeared on any packaging + some details about the 2021 revisit. That way, everything is self-contained in the game, including the history of the game… unless some hacker changes it to say "hello" to a bunch of their mates!

Before I do that, however, I need to fix another bug, which is that when the game finishes, you have to press fire to move to the high score screen, but the fire is applied to the character A so that you immediately select the first character of the 3 initials without having any choice, so I need to leave a pause before I read the fire again. I do the same thing when completing the high score to prevent a new game from being immediately started, so should be able to use that same code.

Ok - I've fixed that.

11/05/2021

Tonight I've got the basics of my new scrolling message working. I can work on the content tomorrow. I've got oodles of memory left, about 8k, so I can make it nice and long and indulgent. Unlike the immature drivel on the original title, this will describe what the game was supposed to be about and the history of the game (if anyone is even remotely interested?) It'll be a split screen bitmap/text. Making this flicker free is an interesting challenge.

12/05/2021

This morning, I am writing the text for the new scrolling message. Because I've now switched off all the ROMs I've got a nice chunk of 8k to play with, which is pretty generous. May as well fill it up. Why not? Be indulgent!

13/05/2021

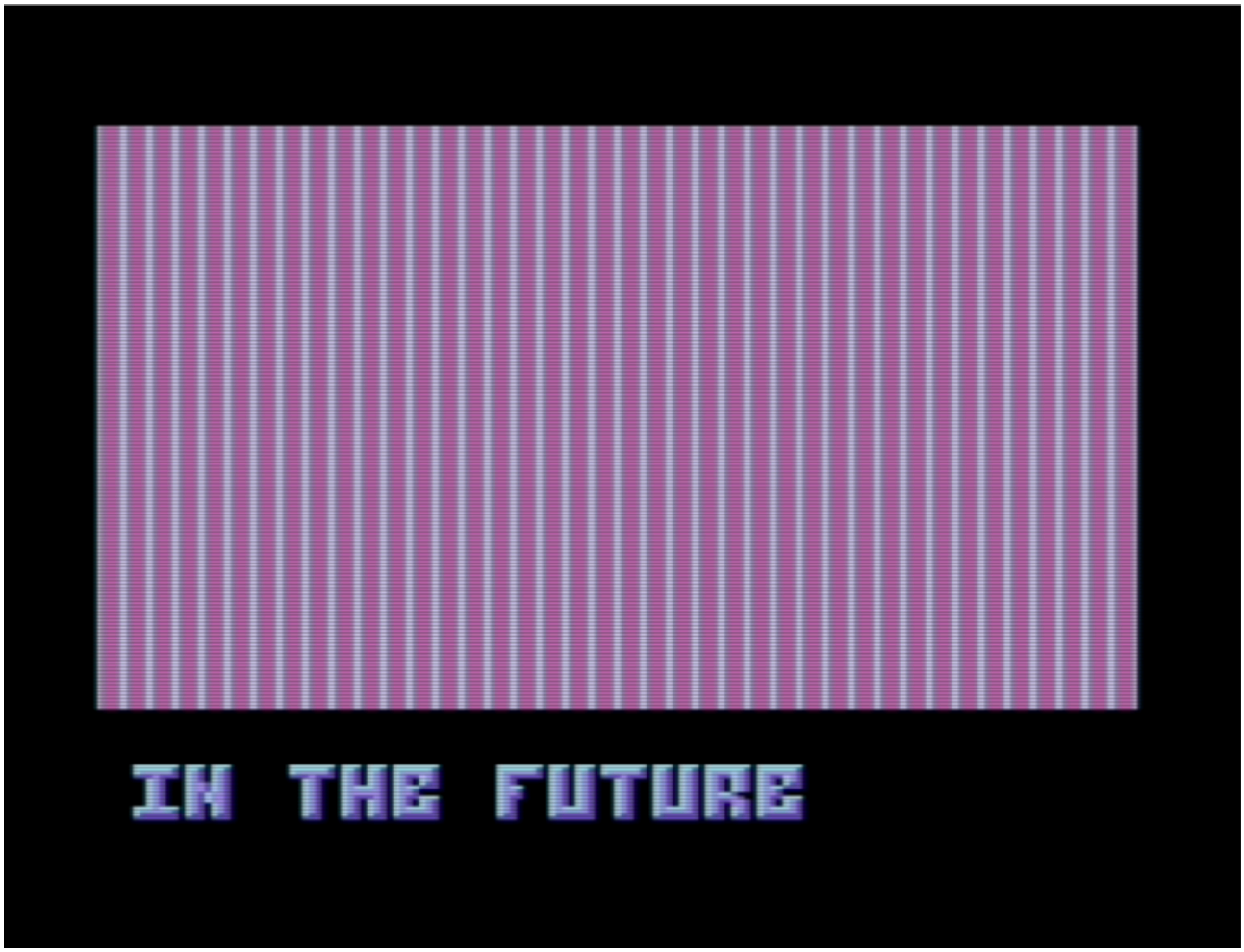
Need to track down a bug when playing the main game music on the loading screen. The filters are going out of whack. Not sure why.

I think it it because I'm not setting the tempo early enough… so I'm reading it before I set it.

Yay! Fixed it.

14/05/2021

Spent some time fixing the glitches on my bitmap/text raster split. Fixed it with a bit of timing:



Now to set up my bitmap editor with a new image.

I’m going to sketch an image of the man being surrounded by all the hazards:

I might base my images on some of these:

(Yes - I’m a Rush fan!!! Something I’ve in common with Andrew Braybrook I imagine as one of his Paradroid levels was called “Red Barchetta")

(Blagger Goes to Hollywood - bought a copy of this at a Commodore Show in Hammersmith - very disappointing after the excellent Son of Blagger)

15/05/2021

So I've started on the bitmap image:

Starting to flesh out some more of the ideas:

Ok - I think I've finished (I hadn't):

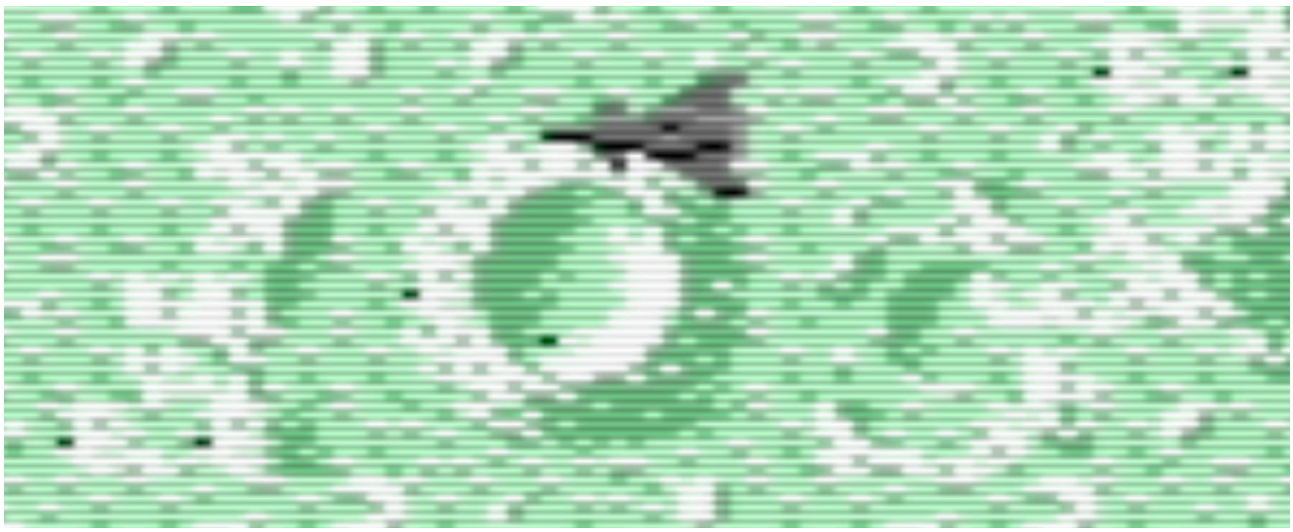
I've also fixed a bug with the scrolling message that meant it was not looping around correctly.

So I think I've nearly finished the game. Now to turn on infinite lives and see if I can actually complete it!

Bugs:
- When we go back to the checkpoint, it flickers a lot when we're standing on it as though it goes through the checkpoint cycle multiple times. We should suppress that.
- I've seen some character corruption on the first green level with a crater… we appear to be writing into the character set, possibly due to a counter overflow?
- I managed to start the level with the music off because I managed to move off the portal before the portal sound had completed
- When transported into space, you need to move before the collision code kicks in - which seems wrong
- Timer ran out just as I went onto a portal, which caused me to go back to the previous checkpoint and get locked so I could not move. We should stop the countdown on a portal whilst transporting.

So I couldn't complete the game, but it was looking promising, if not the most annoying game ever.

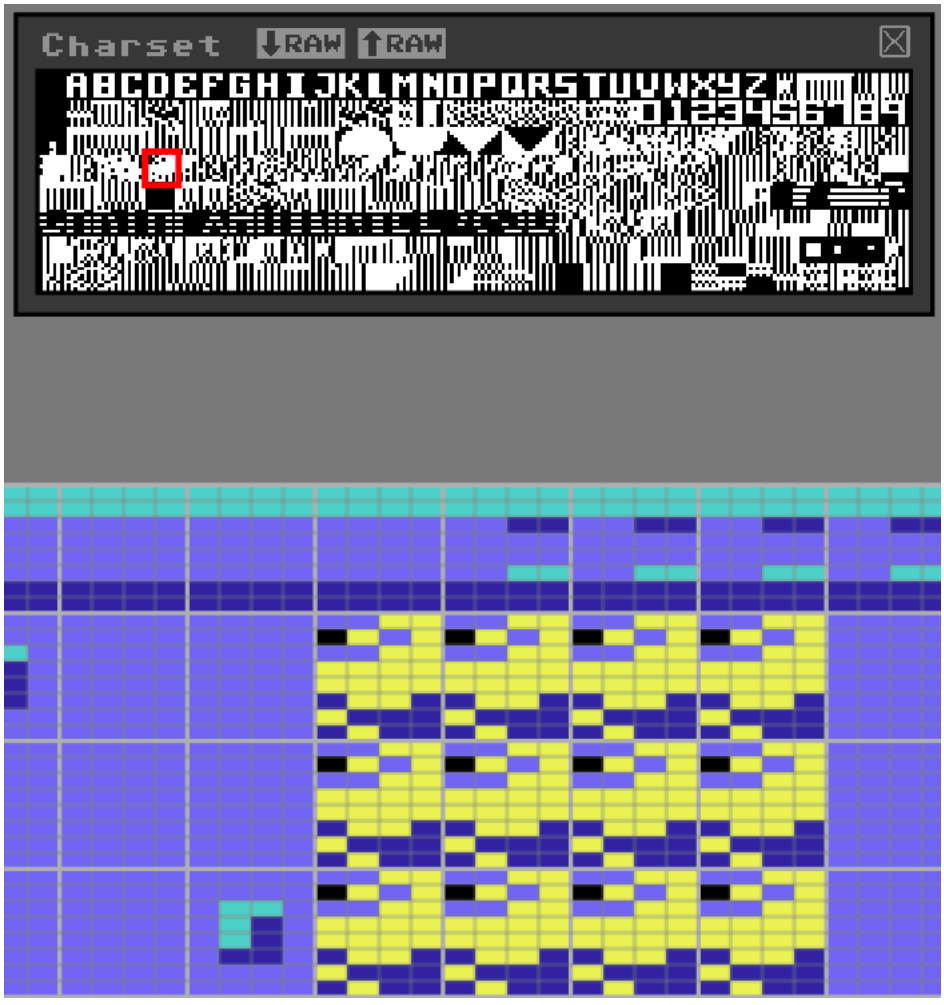
This is the crater bug:



Appear to be setting part of the character to background colour.

Need to find what this character is?

I think it is row 4, character index 4, so (32*4)+4



So it seems to be the 2nd byte.

30720 + (((32*4)+4)*8) + 1 = 31777 = 7c21

It seems to happen when the main game starts up

Here:

```
label(MainGame8969);
LDA_AbsoluteY(ARRAY32320);
STA_AbsoluteY(ARRAY31776);
```

It is character 93 (5d)

So this should by the byte at:
30720 + (93 * 8) + 1 = 31465 = 7AE9

Actually, it's the 4th row down not the character above, so:
7AEC = 31468

It might be that it has always glitched and I've never really noticed it. I could just hack it and set it to a certain value.

The value of 7AEC is 3f or 00111111

Let's just set it to 255

Or maybe we'll just leave it as it is.

In fact, we should be able to prove that this is nothing new by looking at an older version of Worron before I started ripping it.

Yes - it's been there forever, so let's forget about this one.

That leaves:

Bugs:
1) When we go back to the checkpoint, it flickers a lot when we're standing on it as though it goes through the checkpoint cycle multiple times. We should suppress that.
2) I managed to start the level with the music off because I managed to move off the portal before the portal sound had completed
3) When transported into space, you need to move before the collision code kicks in - which seems wrong. There is an issue of not dying if standing on a background item.
4) Timer ran out just as I went onto a portal, which caused me to go back to the previous checkpoint and get locked so I could not move. We should stop the countdown on a portal whilst transporting.

Let's tackle 3) first of all. We can walk onto a spike when it is down and nothing happens when it comes up. So this is the main issue here.

```
label(CheckForCharacterCollision);
```

This code does not get called unless we're trying to move.

We should be able to call the routine above as it checks the value of CharacterUnderMan which appears to have the correct value at reset.

I think I might have fixed this issue. This was the worst of the issues. The others were quite edgy. I'll need to play through the game to test it out.

Let's try to fix 1) next.

Looks like IsStandingOnCheckpoint = 0 which is the cause of the flicker.

I've fixed the race condition with the countdown timer. The bugs remaining are:

Bugs:
1) When we go back to the checkpoint, it flickers a lot when we're standing on it as though it goes through the checkpoint cycle multiple times. We should suppress that.
2) I managed to start the level with the music off because I managed to move off the portal before the portal sound had completed

16/05/2021

I'm getting very close to finishing Worron. I'm going to fix the character crater issue as it bothered me, so let's get rid of it. I'll just manually hack the data.

So this leaves the following issues:

1) When we go back to the checkpoint, it flickers a lot when we're standing on it as though it goes through the checkpoint cycle multiple times. We should suppress that.
2) I managed to start the level with the music off because I managed to move off the portal before the portal sound had completed
3) Fix the crater issue

So, tackling 3) I see that character 64, row 2 also has a dodgy pair of bits that shows the background.

30720 + (100 * 8) + 1 = 31521 = 7B21

The current value is $37 which is: 00110111

Let's try 01110111 $77

That looks good.

So there were 2x bytes with dodgy characters and I've confirmed that they pre-existed all the work I've done. Perhaps a result of some earlier corruption. Who knows?

Let's tackle the issue with the portal allowing me to move off… maybe this only happens when I die and go back to the start?

Yes - this is the bug. We stop the man moving when we start the level, but not if we die and go back to the portal.

So it seems as though the joystick is not read at the start of the level, but is when we go back, so there must be some flag we need to set.

So it looks like we read the joystick whilst the man first arrives at the portal, but we must suppress its use later.

The bug is not that we can move when we enter the portal a second time, it is because there are a few frames in between the portal sequence starting where we are able to move

The bug actually happens if we move right at the start of the level without even dying.

I think I need to patch this up by having another countdown that is started after the n frames

Ok - I think I've fixed that issue.

I've noticed a new bug which is that after we've been to a checkpoint, we can go back and use an open portal as a checkpoint, but I think I might just leave that as a feature haha. Maybe it's a big in the portal itself! And other excuses.

This just leaves the flickering issue between being on a check point and displaying the Worron grid.

Surely we can use this:

```
LDA_Absolute(IsStandingOnCheckpoint);
```

When we're trying to display the Worron grid comment and ignore that code? Let's try that.

I think I'm going to leave the final feature, the flicking on some checkpoints as a 'feature' (I actually went back and fixed this as the final issue)

So - I think I've fixed all the issues I want to fix.. can I actually complete the game? Let's try again….

The craters are too hard to move through. I need to make these easier by allowing the man to walk through more of the crater characters.

I need to determine what characters make up the craters.
Crater seems to be characters 84-112 roughly.

So I managed to play the game all the way through… frustrating as hell. There was only 1x bug I noticed which I think I need to fix.

When changing screen, if you immediately land on a character that you can't normally navigate to, you get stuck and can no longer move. You have to wait until the countdown runs down to zero, which is a bit crap. Ideally, we need to relax this check the moment we arrive on a new screen. Or at least we need to investigate what is causing it. First we need to reproduce it.

17/05/2021

So thinking about the bug. We should be able to reproduce it by manually poking a wall character under the man and seeing if that does it.

Found a way to reproduce it. Moving the man here:

And poking: 7480 = 26

Let's look at:

```
LDY_Absolute(CharacterUnderManScreenMemoryIndex);
```

This value will change depending on direction, so if stationary, perhaps we ignore walls

Right = $13
Up = $10
Down = $0d

Oooh - I think I might have fixed it. I'll only find out when doing another run through the game.

Ah - I think I've found another bug when the skulls drain the time. Get some dodgy glitch. Let's see what happens when we normally run out of time.
It seems to be when the countdown is already quite low…

So I think this was caused by decrementing the same counter 2x and then rolling over into FF and corruption land, so I've put a sticky plaster around the DEC code for skulls so we don't DEC if already zero.

Seems to have done the trick.

I now declare it complete.

34 years in the making! Haha.

So I think this is the end of the diary.

21/05/2021

Oh no it's not!

So I sent the game to Frank Gasking of www.gamesthatwerent.com and also to Darren Melbourne ex-Paranoid, who now is involved in an online retro gaming platform www.antstream.com

Darren has offered to put Worron up on the site, which is quite amusing as it'll mean that it finally gets published after all these years.

He also asked me about another game I started writing for Paranoid in 1987, Exodus… of which I still have the graphics… watch this space…. Maybe I'll write it.

It's been a cathartic experience finishing this game after 34 years.

The most interesting part was learning how to finally write SID music. As someone who dabbles in music, it was interesting to learn how SID music requires building your own synth out of the raw SID components and then your own sequencer on top of that. I now fully appreciate why each of the famous SID composers had their own 'sound' as they had different ways of tackling the various problems.

What I continue to love about the Commodore 64 is that it is a limited sandbox in which to create. Its very limitations are what make it such an exciting tool to create in. Yes - you could write an iPhone game with mb of mp3 sound data… but where is the fun in that?

Doing this project has made me want to do more. After 3 decades or so away from the Commodore 64, I feel drawn back to its simplicity.