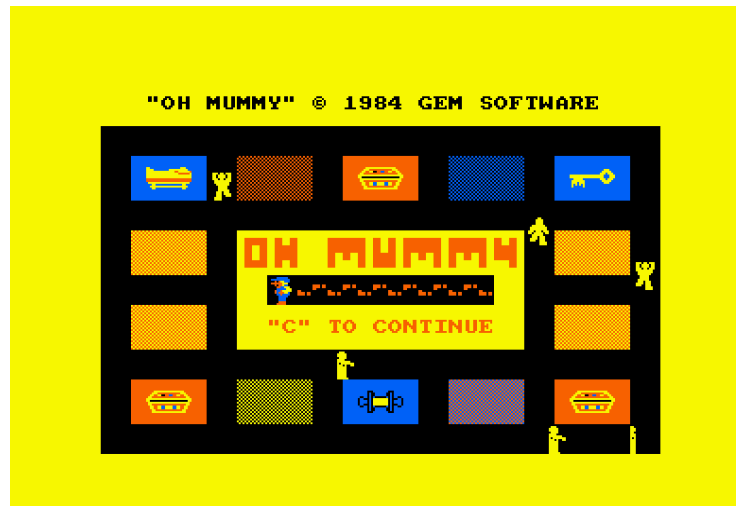


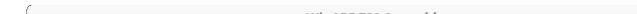
## Creating Oh Mummy Resurrected

Another great game that came free with the Amstrad is Oh Mummy. Its simple yet addictive gameplay made it a delight to play as a child. Its reliance on the keyboard for menu control means it cannot run on the GX4000, so I thought I would disassemble it and see if it is possible to convert it.

LOADing the BASIC loader into memory and disabling the final command that runs the game enables me to load the entire game into memory and then disassemble it using WinApe's Z80 disassembler. Then it's a case of separating the various functions by making a line break after every RETurn or JUmP command. This is purely to make the code easier to read, so we are able to identify the main functions of the game.



After disassembling the code and improving the formatting, I look for areas of code that should be marked as DATA in the disassembler. These look like ‘nonsense’ Z80 commands when disassembled, so they are not too difficult to spot. I.E:-



The screenshot shows the WinAPE Z80 Assembler interface. The menu bar includes File, Edit, Assemble, and Help. The assembly code is displayed as follows:

```
org #0100
nolist
```

```
ld h,a
ld h,l
ld h,l
etc
```

We know straight away this is data as it would be very inefficient programming if it was a real function. So we mark the appropriate area as DATA in WinApe's disassembler and then disassemble it again and paste in the resulting DB statements into the appropriate place. Thankfully Oh Mummy's code is well structured, and there were only two main data areas in the disassembled code. The first one contained all the text for the instructions, and the second one contained the sound, images, object map and other variables. For the instructions data area, I worked out the corresponding letters for the hexadecimal numbers and inserted them instead, to make it more accessible.

```
WinAPE Z80 Assembler
File Edit Assemble Help
; NEED INTERRUPT TO WORK
K1_time_please equ $bd0d
K1_test_key equ $bb1e
K1_read_char equ $bb09
K1_char_return equ $bb0c
sound_set_env equ $bcbc
sound_set_ent equ $bcbf
sound_queue equ $bcaa

; NEED EXTRA FONT DATA, ETC
txt_output equ $bb5a
txt_set_window equ $bb66
txt_set_cursor equ $bb75
txt_set_paper equ $bb96
set_char_limits equ $bca7
txt_clear_window equ $bb6c
scr_dot_position equ $bcd1
txt_set_pen equ $bb90

start:
    call set_char_limits
    ld a,1
    ld hl,soundenvelope1
    call sound_set_env
    ld a,1
    ld hl,soundtone1
    call sound_set_ent
    ld a,2
    ld hl,soundenvelope2
    call sound_set_env
    ld a,2
    ld hl,soundtone2
```

## Firmware calls

Once we identify the areas of data, we can look to see if there are any firmware calls in the code. As the Plus cartridges do not contain any firmware, these will need to be replaced with our own functions. So we look to see what calls are made, usually any command starting with &BB, &BC or &BD is a firmware command. So we look up the firmware manual to see what these do and define names for them. This will give us an idea of what commands we will need to create to get the game to run.

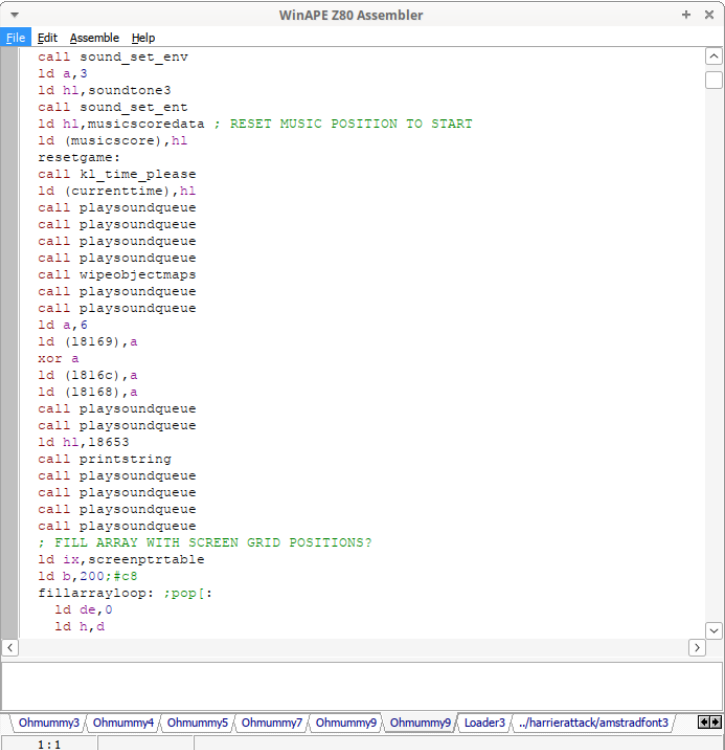
Unfortunately in Oh Mummy there is quite a lot of reliance on the firmware. The interface uses the firmware font, as well as pen, paper and window functions. These will need to be replaced by our own code. The sound effects and music use the Sound Manager envelopes, tone and sound queue commands. So these will need to be replaced as well. Then there are the keyboard reading functions and timer. The keyboard, sound and timer functions require the use of interrupts, which are built into the firmware, but we will need to be recreate them for it to work in cartridge form.

## Music

The music data in Oh Mummy seems to be a series of SOUND statements that are played sequentially. So we will need to download the Streets of Cairo music score from the internet and recreate this so it will work with my Cpsoundeffects.asm code. Interestingly, the original programmer did not use interrupts to check to see whether the sound buffer was empty in order to decide whether or not to play the next note. They just called their music function repeatedly throughout the game. So once we get our own music system working, we can remove this inefficient code.

## Sprites

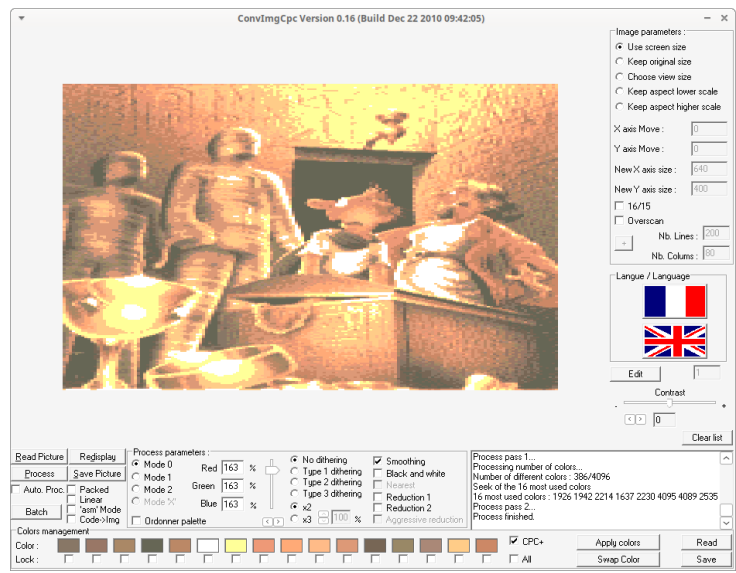
Identifying the main functions of the game helps us to know which data statements are sound, images or text. It is then just a case of labeling each data statement accordingly. If in doubt, we can for instance, set all the mummy graphics to a single image to see which one points left, right, up or down, etc. Once we get the game running with the Plus ASIC enabled, it will be possible for us to design multicoloured sprites that will replace these single coloured ones.



```
WinAPE Z80 Assembler
File Edit Assemble Help
call sound_set_env
ld a,3
ld hl,soundtone3
call sound_set_ent
ld hl,musiccoredata ; RESET MUSIC POSITION TO START
ld (musiccore),hl
resetgame:
call kl_time_please
ld (currenttime),hl
call playsoundqueue
call playsoundqueue
call playsoundqueue
call playsoundqueue
call wipeobjectmaps
call playsoundqueue
call playsoundqueue
ld a,6
ld (18169),a
xor a
ld (1816c),a
ld (18168),a
call playsoundqueue
call playsoundqueue
ld hl,18653
call printstring
call playsoundqueue
call playsoundqueue
call playsoundqueue
call playsoundqueue
; FILL ARRAY WITH SCREEN GRID POSITIONS?
ld ix,screenptrtable
ld b,200;#c8
fillarrayloop: ;pop[:
ld de,0
ld h,d
Ohmummy3 Ohmummy4 Ohmummy5 Ohmummy7 Ohmummy9 Ohmummy9 Loader3 ../harrieraattack/amstradfont3
1:1
```

## Loading Screen

I wanted to see what a loading screen would look like on the Amstrad Plus. So I scanned the inlay card of my copy of Oh Mummy, and converted it to the Plus palette and resolution. This is done using a program ConvImgCPC. The results are quite pleasing as the source image uses mostly shades of orange. One thing I did need to do is adjust the brightness up to 163%. This gives a more pleasing spread of colours. The Plus colour palette spread is uneven and biased towards lighter colours, so increasing the image brightness overcomes this problem.



It is then just a case of saving the palette, and saving the image to an ASM file, which we will use as a loading screen. I may either add a title manually or use Plus sprites to create a title and place it on the image.

## Errors in disassembling

The main code for Oh Mummy is located at &6000. But I found whenever I relocated this to &1000 by changing the ORG command, graphical items in the menu would be out of sync. Also, the Guardian Mummy's tomb would not open correctly. Checking the code, I found the disassembler had mistakenly given labels to some hex values that were meant to be just coordinates. So the code was treating coordinates as a memory address to look up. Once this was corrected, the game could be correctly compiled at different addresses. Starting at &0100, I found I could compile the entire game within the first 16K bank of RAM (&0000 to &3FFF). This gives us plenty of spare memory to work with.

## The next stage

So the next stage will be recreating the text and interface commands to use our own functions. As the firmware functions can't be called from a cartridge, this testing will need to be done in WinApe's own assembler.

## Pen, paper, ink

The graphic functions I replaced using my own code were txt\_output, txt\_set\_window, txt\_set\_cursor, txt\_set\_paper, txt\_set\_pen and txt\_clear\_window.



Whenever I replaced txt\_output with my own text printing function from Harrier Attack Reloaded, I realized Oh Mummy was using a lot of ASCII control codes to execute additional graphics commands, such as setting the mode, border and ink colours, paper and pen colours and specifying cursor location. These are passed to txt\_output as if they are normal characters, only the firmware interprets them accordingly. So I needed to replicate these codes in my own function. Looking up what function each code performed, I soon had the locate function working, as well as defining and clearing text windows and setting inks and pens.

One particular problem I came across was setting the paper and pen for text. Using a combination of OR, AND and XOR while copying the character data to the screen, I was able to change the colour of the pen and paper of text. These commands invert and rearrange the bits in the byte, which changes the colour on the screen. I had to make a careful note of which combinations produced which colours. But how to allow the program to set the colour by pen and paper numbers was a more difficult problem to solve. In the end I decided to make a lookup table. The table contains the pure machine code to produce a particular pen and paper combination. Whenever the program wants to change the colour, it looks up this table, with PEN number on the vertical axis and PAPER number on the horizontal axis. The chosen code, just 4 bytes, are copied directly into the letter printing function at the appropriate places producing the correct colour for the particular combination. I'm not sure how the BASIC firmware accomplishes this task.

## Compiling the cartridge

As the main code for Oh Mummy can fit in the first 16k of memory, I was able to reuse the memory structure of the Harrier Attack Reloaded. This saves me having to redesign the boot sequence for the cartridges and the loader code for the discs and tapes. The file structure and the bash files for compiling the game can also be copied over without needing too many changes. I am able to just replace the loading screen with an Oh Mummy picture and change the filenames on the disc and tape and everything works the same. As the font data was causing the code to run past &4000, I moved it to the second block of code at &C100, where I will put all the extra functions I create for the Plus version. A jump block at the start of this memory enables us to call the functions from the main program as necessary.

## Menus

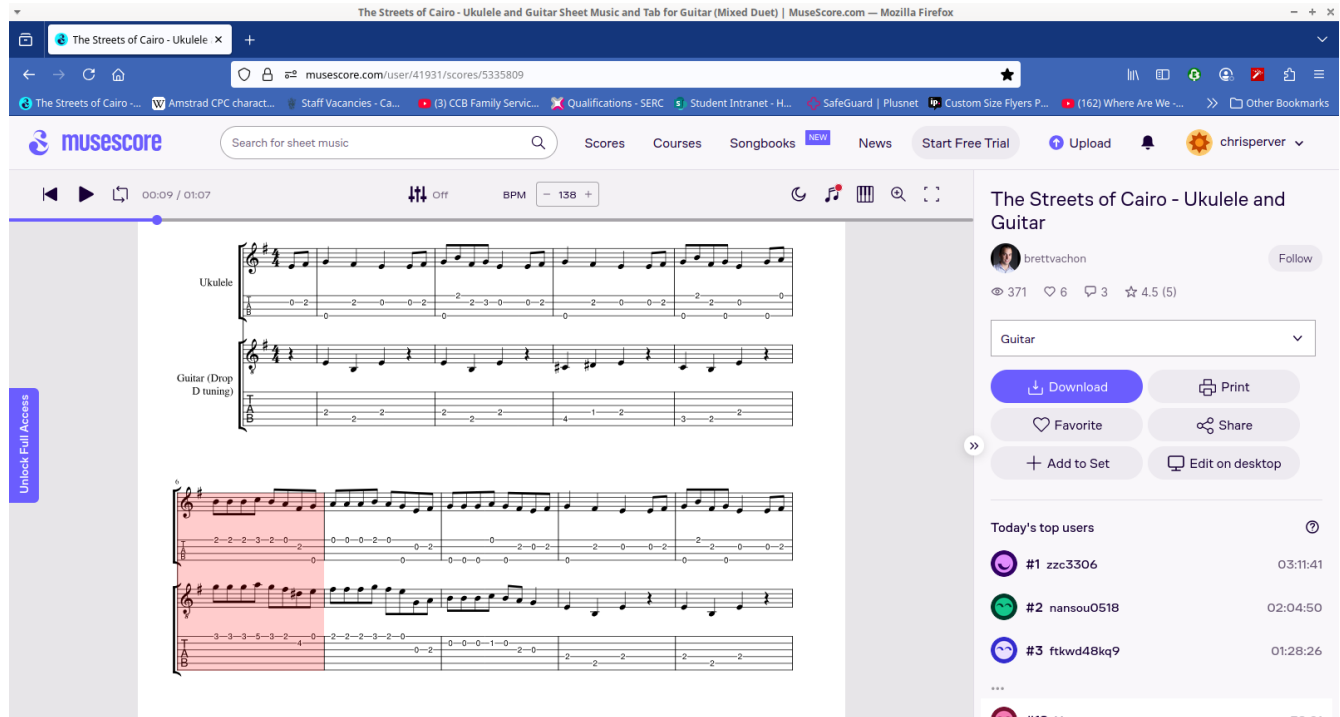
In order to allow joystick control of the menus, I thought it would be best to move the main menu options into the centre of the screen, using the high score table layout. So I crushed up the high scores, and used a > sign to allow the user to select the menu items. The option screen also had to be changed so it could be accessed via the joystick, so I modified the layout so all the options are displayed on the high score table and can be changed using the left and right directions.



## Blinking cursors

Rather than spend a lot of time redesigning the code of the scoreboard name entry to allow input via joystick, I decided to just copy and paste my code from Harrier Attack Reloaded straight into Oh Mummy. I didn't think it would work but it did, straight off, just by telling it where to position the text and where the name is stored in memory. The only problem is, the mummies do not move in the background while the user is entering their name. I found when I enabled them, the text would get printed in the wrong colour. So I will need to look at that.

## Music score

The screenshot shows a web browser window displaying the MuseScore website. The main content area shows a music score for 'The Streets of Cairo' for Ukulele and Guitar. The score is written in 4/4 time and features a mix of eighth and sixteenth notes. The Ukulele part is in the upper staff, and the Guitar part is in the lower staff. The guitar part is marked 'Drop D tuning'. The score is displayed in a clean, modern layout with a light blue background. On the right side of the page, there is a sidebar with the title 'The Streets of Cairo - Ukulele and Guitar', the user 'brettvachon', and various interaction buttons like 'Download', 'Print', 'Favorite', 'Share', 'Add to Set', and 'Edit on desktop'. Below the sidebar, there is a section for 'Today's top users' with a list of usernames and their scores.

As I have no access to the Amstrad firmware Sound Manager, I need to recreate the music from scratch. I imported the CPSoundEffect.asm functions I created for Harrier Attack Reloaded for making the music, and looked for a suitable score for Streets of Cairo online. I look for one that makes good use of the AY's sound channels. Not too complicated, but has a few chords, and still leaves a channel free for sound effects. I eventually found one. Then it is a case of manually working out which notes correspond to which frequencies on the Amstrad and typing them into our music data block.

I may need to modify it, as I find the rhythm in the original version more catchy than this rendition.

## Redefinable keys

I imported my redefinable keys functions from Harrier Attack Reloaded and inserted the appropriate code into the option screen in Oh Mummy. I included the two player key detection code just in case I decide to make a two player version of Oh Mummy later on. I will need to amend the main character movement function so it uses my own keypress detection routines.

## **Mummies in the menu**

There is a bit of a problem with the game speed and the mummies moving in the menu. There is a delay loop which slows the mummy and player movement down to the appropriate speed. The problem is, the mummies moving in the menu also depend on this loop to gauge their speed. Because of this, the menu selection code is also being slowed down. I may need to move the mummy and player speed code into an interrupt so it doesn't affect the menu selection. The game speed function may need to be reworked.