The Mansion – Technical Aspects

2011-05-23 by Michael Steil

by enthusi

(This is a reprint from "Vandalism News" issue 52)

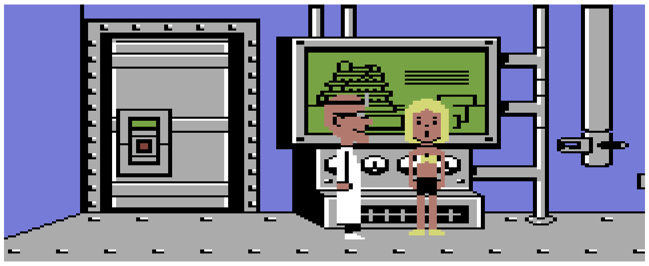
Maniac Mansion is a true classic – in many ways.

It's a classic example of smart game play versus photorealistic graphics that is lacking in so many modern productions and it's the first LucasArts Adventure that made use of the point and click verb parser. And it's of course eponymous for the famous SCUMM – the scripting utility for Maniac Mansion. This is reason enough to talk about it in some more detail and as a side effect advertise the NEOram version for C64 released just now. :)

MM has a nice Wikipedia page (especially in Finnish – check it out :) so I think I shall concentrate on some technical specialties:

The original *Scumm (version 0)* was designed for C64 in '87 and its idea was to be able to develop a game independently of the target machine and only port the SPUTM (script presentation utility TM (seriously)) to several architectures. However it started with C64 and for several reasons the C64 version was kind of special in comparison to all later ports. Hence Scumm version 0 is only used here. Even **Zak McKracken** already had important changes and fixes. The original version came on disk with some copy protection (not too bad, not many cracks are around) and hence there is no in-game copy protection unlike all other versions (except for the Apple). You may have seen that code door at 2nd floor on other computers and guess what: the door code is made of PETSCII graphics chars! Officially the protection was left out due to disk space. In fact the 2 disk sides are pretty crammed in the original. So the game design was done on PC (UNIX system) and assembled there. This explains the spaghetti like distribution of the data and some of the code. In general the data is split into rooms. A room contains all the data for a specific area in the game. I.e. the beginning with the beautiful moon in the background or the kitchen is one 'room' each. The data consists of one graphics charset in multicolour mode and its screen map (from 40×17 up to 160×17 chars big), one full charset for the clipping mask with its map (same size as screen map of course), a colourmap for the fourth colour per char (ranging only from 0-7) and then the scripts themselves. There is one room data set for each location and numerous scripts, one for each room and several additional ones.

The scripts are more or less freely distributed among the rooms by the original SCUMM. They differ a **LOT** between English and German versions, for example to have minimal waste of space at the end of a sector on disk. Beside the room graphics and script data there is also data for sound effects, the characters in the game and the objects. The people in the game are drawn as sprites, but not constructed from ordinary sprites but plotted into a semi static multiplexed sprite matrix in real time each frame (17 raster-splits per frame). The data for the characters is also attached to several rooms. Often the one they appear in first. The text charset is not the ROM font btw. Since usually $01 is set to $14, hence RAM, RAM, RAM…



Each actor has a vertical stripe of sprites covering the whole height and this allows for quite some actors on screen simultaneously. The sprite priorities are set according to the actors positions, so you can actually walk in front or behind other kids and even orbit them for hours… MM is so much fun. :)

The graphics are stored in sprite like format (3 bytes per line) but with a successive height of 16 pixels normally. To save space each person is only stored from front, back and its right side. Whenever someone is facing left, the graphics data is mirrored via table and then plotted. Oh, and all the facial expressions, well the mouth and the chin (being one pixel) are not part of the data but added during plotting. Btw, the explosion in the intro, chucky the plant and the mummy are stored and handled as actors as well. You can see that when you leave the bathroom: the mummy disappears just before the rest does. To prevent the secret phone number from being revealed the background graphics is distorted of course. :) The graphics replacements are stored among the objects. So an open radio is a replacement of the object graphics radio. Stored as x, y position in char steps inside the room area and its width. Each object has an owner attribute and a position x, y in chars where to walk to (which is not necessarily identical to its position). Also there is a byte containing the height of the object in pixels (using 6 bits) while the upper 2 bits give the direction the actor is facing when reaching the object. So when Dave opens the fridge we see his back but when reaching a door, he’s facing right for example. The object’s name is also given along with a pointer to the script dealing with it. Those pointers are 24bit wide. 1 byte for the room it is attached to (again, these data change between different language versions) and another 16bit offset inside that data chunk.

The scripting for MM is rather complex. The memory resident interpreter understands 82 opcodes and handles 256 global variables! There is lots of things are stored and swapped around. Like whose friend with Ed or the Tentacle or whether the highscore at the arcade is already set... It even has fancy stuff like a random number generator (yes, the numbers in the game like phone numbers, codes) are not always the same! (Look for the cheat in the NEOram version that displays Edna’s phone number, the safe combination, the radio frequency and the highscore once it’s played).

A typical script contains commands like these:

```
(0035) (42)   startScript(103)
(0037) (02)   startMusic(185)
(0039) (CE)   putActorAtObject(Var(225),162)
(003C) (82)   stopCurrentScript()
(003D) (04)   unless (Var(61) <= 73) goto 5A96
(0042) (66)   Var(66) = getClosestObjActor25(113)
(0045) (02)   startMusic(186)
(0047) (CF)   setState02()
(0048) (E2)   stopScript(Var(162))
(004A) (82)   stopCurrentScript()
(004B) (03)   doSentence(62,74,55)
(004F) (5B)   Var(69) = getActorBitVar(103,Var(64))
```

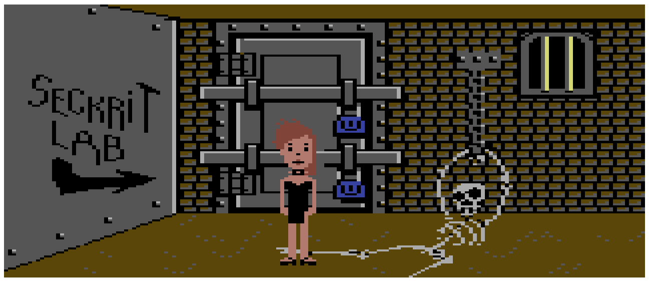
Unlike in later Scumm revisions, version 0 for C64 has quite some C64 specific hard coded stuff inside the data. A nice example: ALL actor graphics are stored the same way (you can simply try a PC based multicolour sprite ripper on the disk images) except for the breasts of Sandy =D They are stored completely separate. Maybe to prevent early nude-hacks? Back to the data formats. There is quite some graphics in this game - there is a total of 52 rooms, including the initial kid selection screen, save game screen or cut scenes which are all handled by the same interpreter and format. To make all this fit onto two disk sides, there is a specific RLE encoding. The four most common bytes of the data to follow are given for each data chunk. These were derived by the SCUMM on the PC during compilation already. The encoding is arranged as following:

- A data byte of < $40 gives the number of different single unpacked bytes to follow.
- $3f to < $80 represents the length of the run of the byte to follow (+$3f of course).
- $7f to < $a0 is the number of runs of most common byte 1,
- $9f to < $c0 and $bf to <$e0 and $df-$ff for common byte 2, 3 and 4 respectively.

This encoding is rather effective for the MM graphics. Walkable areas for a room are composed of a list of multiple boxes with varying size and positions and stored after the graphics data. You probably noticed the slowdown during scroll in the game. This is due to the double buffered scroll which waits for all the clipping and room data to be moved into the correct RAM areas. The colour scroll of $d800 is done with some sort of line based speed code. Rastertime and memory are both used almost to the max. At start-up some own drive code is loaded into the floppy (not in my NEOram version of course). Later on a room is loaded by checking which room we want to see next, then getting its corresponding disk side from a table, checking for correct disk, getting the track and sector at which the room data starts from another table, calling a function to set this t/s offset, then calling another function with the target address in memory and another one with the length to be transferred. Only then will the room data be actually loaded. Even worse, usually only the first few bytes are read since they contain the offset to the table of other offsets (actors, objects, and scripts) which are then loaded to the corresponding memory areas. As soon as the drive code is initialised that RAM area on C64 side is overwritten by data AND code again. So the only memory available for extra code in this version was the drive interface and some other routines handling disk sides etc. A major gain was for example the now obsolete "wrong disk" message. This game really uses all the RAM there is. Most of the stack is occupied by code as well. Some of the JMPs instead of JSRs in the interpreter I didn't get at first were due to that fact - to keep the stack pointer high (that was kind of sucky to debug).

I was able to rewrite some interpreter routines as well to gain more space for the mouse driver and save game selection.

And, err... I removed the checksum check for each room data. That was also quite some speedup btw and it probably prevented lots of people from making trivial changes to the game. Yet it is hardly possible to apply big changes in the game logic. All offsets are hard-coded to the data. Even changing the length of the name for i.e. the 'fridge' by 1 char, causes dozens of offsets to change (for costumes, scripts, sounds, other rooms and so on). Even the interpreter looks rather different from language to language. And the RLE nature of the data, makes it pretty hard to change graphics as well since they have to keep the same size.



For the *NEOram* version I only redistributed the room data and completely replaced its track/sector/disk side structure.

So unlike all other patches before this one no longer 'emulates' disk access anymore. MM handles the pointer on screen very arborescent and there is no such thing like an "open" function call. Instead pointer positions are stored as x, y chars on screen, x, y relative to room origin in chars, x, y as pixels on screen, x, y as pixel coordinates (half the x resolution 0-159) and as a delta to last frame. All of these are set by different functions. So when a verb is highlighted and you press fire, multiple addresses are set already in the background. The call itself is a self modified JMP in the end. To allow the OPEN and PICKUP command on the right mouse button I rewrote the button routine instead. The input from the mouse is faked as if it would point to the OPEN verb, the pointer is hidden (more or less), the old position stored and the mouse driver is skipped for the next 3 frames while the joystick readout is rerouted to an own routine that returns button pressed for the next frame, then 1 frame of button release and still no mouse action, one frame to return old pointer coordinates and run the routine to convert the mouse coordinates to all those mentioned before and finally enable pointer sprite and toggle mouse control back in. Sounds a bit weird but otherwise there isnt enough rastertime left. This is also why you shouldn't use the mouse on an NTSC system. The mouse is read every 2nd frame only for the very same reason. The evaluation of the new coordinates for the game engine (x, y in absolute and relative chars and check for verb activation) are only carried out when the mouse driver itself is skipped. This was the only way to get things done before the next rastersplit is due. Also the mouse is disabled as long as a button is pressed or actions with objects or scrolling the inventory would take too much rastertime. And sometimes you see a little jerky screen during loading when the I/O area has to be switched in to set NEOram registers and the rasterline hits the IRQ to switch back to text mode. I chose this part of the screen since its the least disturbing and usually works quite ok. MM is and was great fun - as soon as I've had some rest I will look into Zak McKracken deeper. Almost can't wait to do so (imagine both games on one cart). Have a lot of fun and try all the different solutions to the game.

Yes, Jeff HAS some more or less special ability and you can use the integrated room viewer to check for things yet unseen ;-)

Did you every finish the game without shooting that poor meteor (why is no one calling it accurately 'meteorite'?) into space? Go ahead and enjoy another game of Maniac Mansion. The NEOram image will work well under vice by the way but if you consider assembling your own NEOram card just look for schematics and part-lists on the web or contact maniac`enthusi.de - Quite likely there are some PCB boards still around.

Ok, let's go rescue Sandy!

/enthusi