World RetroMagazine
future days are back
Issue 5 Year 2 - January 2021 - https://www.retromagazine.net - Free Publishing

Happy New Year 2021

from all our editorial staff





Inside this issue

- NINTENDO 64 Mario's 3D revolution
- Introduction to the MEGA65
- The MOS VIC videochip
- SymbOS - Windows on the Amstrad CPC!
- Exclusive interview with Randall Flagg/Razor1911

Games

The Real Ghostbusters Arcade Fangame (NEW)
Metamorphosis (Preview)
Holiday Lemmings
Mighty Final Fight
Mario Kart 64
Wiz Quest for the Magic Lantern (NEW)
Final Fantasy VII
and many more!

Over 80 pages!

8-bit Christmas Holidays

After a year like this one, spent mostly indoors or with a mask over your face, you may not feel much like celebrating the Christmas holidays, if for no other reason than the staggering number of victims of this damn virus. It is now mid-December, but it is not the same atmosphere as in previous years. This pandemic has psychologically drained us, mentally exhausted us. If last year, as always, the horizon of the future seemed clear and without any particular threatening clouds, now, despite the imminent arrival of vaccines which should mark the 'point of return' to a renewed normality, everything seems to us to be shrouded in a cold transparent cloth from which we have yet to escape. And it is not the typical cold of the winter season that is looming, it is not the snow that has already covered some places in many parts of Europe. The whole planet is hurting, our cities seem dull, all the lights and decorations in our cities seem less festive, the snow that has fallen a few days ago no longer brings the cheerfulness and light-heartedness typical of the end-of-year holidays, the cold temperatures of the incipient winter do not have the flavour that makes us say that Christmas is in the air.

It's a bit like living in an 8-bit world, with low resolution graphics, few sprites, limited colours or even black and white only. Think about it: only one year ago the world as we know it looked like a very powerful PC with 64-bit multi-core CPU, 4K display, super-fast GPU capable of displaying millions of colours, full-stereo sound card. And, let's face it, with not much soul either. The coming holidays show us a more unadorned, austere, less consumerist and less glitzy reality than that we are used to.

Do you remember how much fun and fantasy lit up the reality of the 80s, when our beloved 8-bit computers, with their 16 colours, croaking sound chips and rubbery keyboards, filled our days? It wasn't all ready and available before our senses. Half an hour to load a cassette game, floppy disks that were often unreadable after only a few uses and joysticks that broke all the time. Everything was more frugal and modest and we had to use a lot of imagination to believe that certain sprites or graphic backgrounds had at least a semblance of reality. And do you recall game covers? Fantastic images that made our imagination run wild, often to compensate for the graphical, sound and animation poverty of the video game included. Every day, though, was like a revelation, a step forward, a passion renewed by the constant desire to learn new things and make new discoveries.

So, in these 8-bit Christmas holidays of 2020, let's try to start again by using our inventiveness, creativity and passion to light up the colours of our holidays once again. One way to do that is to enjoy this outstanding issue #05-EN of RetroMagazine World, with over 70 pages full of surprises, news and exclusives.

Happy holidays to all and a special wish for the coming year. May it bring a true renaissance and lead us to a new era of solidarity and friendship. Then we will celebrate the next Christmas holidays in the right atmosphere.

David La Monaca

SUMMARY

People involved in preparing this issue of RetroMagazine World (in no particular order):

- Alberto Apostolo
- Gianluca Girelli
- Michele Ugolini
- Carlo N. Del Mar Pirazzini
- Daniele Brahimi
- Flavio Soldani
- Francesco Fiorentini
- Attilio Capuozzo
- Marco Pistorio
- Leonardo Miliani
- David La Monaca
- Giovan Battista "giomba" Rolandi
- Antonino Porcino
- Phaze101
- Starfox Mulder
- Simone Battaglioni
- Hakim Rezki
- Roberto "Il Bardo" Del Mar Pirazzini
- Cover by Flavio Soldani

LM80C Color Computer

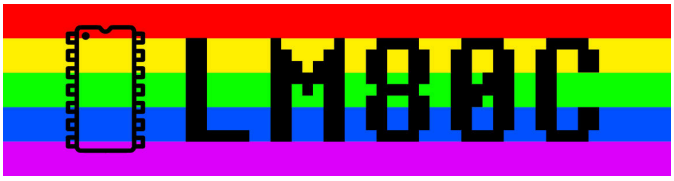
A 2019 self-built Z80-based home computer - part 4

by Leonardo Miliani

The topic of this article is slightly different from the others because we are going to look at computer analysis more from a software point of view while last time we focused on hardware.
The LM80C computer is a complete and functional machine, equipped with interesting features that, as we have seen in previous articles, are respectable and would have allowed it not to disfigure among the 8 bits in vogue in the 80s, being able to side by side with the most popular systems of that period. But what makes the LM80C a fun computer to use is its ease of use: it integrates an operating system that allows you to use its distinctive features such as graphics and sound at power up, without the need for additional software. To ensure, in fact, that home computers could be used as soon as they were removed from the box even by the most inexperienced users, they were equipped with an integrated programming language that also fulfilled the functions of command interface, thanks to which it was possible not only to insert programs but also to drive the hardware without resorting to routines in machine language. The complex software that oversees the management of the machine is therefore, in a home computer, as important as the hardware itself: it was precisely their ease of use that, among other things, allowed it to be widely used.

The operating system of an 8-bit home computer is generally structured on multiple levels. The highest level is, as mentioned, represented by an input/output interface thanks to which the user can type commands in a pre-loaded language and instruct the computer to perform operations, such as printing the result of a mathematical calculation, reading a program from a mass device or emitting a sound. The underlying layer consists of the installed language parser, which analyzes the commands entered and translates them into a series of even more elementary tasks that are performed by the system to perform the operation requested by the user. If the operation to be requested also involves the hardware, even lower level routines are called that communicate directly with the system components. The LM80C is also structured like this, as we shall see. But let's start with language.

The Integrated BASIC INTERPRETER
The LM80C BASIC, a dialect of THE POPULAR BASIC language, is pre-installed in the LM80C (see figure 1 for an example list). Take for example the following two instructions in LM80C BASIC:




```
SCREEN 2: CIRCLE 128,96,50,8
```

They tell the computer to switch to graphics mode 2 and draw a circle of radius 50, centered on the screen, using the color red. How does the system interpret these instructions and generate the result on the screen? It succeeds because in its ROM (non-volatile memory, which does not lose data when the power supply to the machine is removed) an interpreter has been inserted that analyzes the commands typed by the user and transforms them into a series of machine language operations executable by the CPU. BASIC, like other programming languages, is actually a facility for us humans: it allows us to use a communication system that is easy to understand and write because the CPU understands only the “machine” language (it is not by chance called that), that is, the values 0 and 1. To simplify programming, instead of having an endless series of numbers inserted that are meaningless to us humans, the assembly was initially invented (not to be confused with “assembler”, which is instead the assembler program, which transforms the assembly source into machine language), a very low level language of the "mnemonic” type because it consists of acronyms that help to “remember” the instruction they represent. An example of assembly code Z80 with, next to the "meaning” for the programmer:

```
INC A → INC (Remittance) A
CP $10→ C (om)P(are to) $10
JP NZ,$8000 → J(um)P (if) N(ot) Z(ero to) $8000
```

Instructions mean, in order: increment counter A; compare its value to 16 (the decimal equivalent of hexadecimal value $10); if it is not equal, skip to memory address 32768 (decimal for $8000). I spoke of "assembly Z80” because each CPU, as it is different from the others, has a very precise set of instructions and, consequently, its own assembly. For example, assembly statement 6502 LDX #$10, which means "load the decimal value 16 into the X registry”, makes no sense in assembly Z80 either because of the different syntax of the instructions themselves or because it does not have the X registry. It is obvious that writing in assembly is very complex: to simply assign a value to a variable you have to break down the task into

```
10 A=INKEY(10)
20 SCREEN 2:GPRINT "Press any key for next demo...",1,23
30 DRAW 127,95,RND(1)*256,RND(1)*192,RND(1)*13+1
40 IF INKEY(0)=0 THEN 30
70 CLS:GPRINT "Press RUN/STOP to stop",5,23,15,1
80 FOR I=0 TO 255
90 DRAW 127,95,I,0,RND(1)*13+1:NEXT
100 FOR I=0 TO 191
110 DRAW 127,95,255,I,RND(1)*13+1:NEXT
120 FOR I=255 TO 0 STEP -1
130 DRAW 127,95,I,191,RND(1)*13+1:NEXT
140 FOR I=191 TO 0 STEP -1
150 DRAW 127,95,0,I,RND(1)*13+1:NEXT
160 GPRINT "Press any key to exit",5,23,1,15
170 A=INKEY(5):IF A=0 THEN 170
180 SCREEN 1
```

Figura 1: An example of a program in LM80C BASIC

many small operations (see figure 2 for an example of a portion of the firmware of the LM80C in assembly Z80). In order to meet the earliest users, but also to speed up the writing of the programs to the more experienced ones, the so-called high-level languages were developed, that is, languages that were easier to write and more intuitive to read and understand, which were also integrated into the memories of computers of that period.

The language that between the late 70s and early 80s of the 20th century went for the most was BASIC! The language was born in 1964 (yes, it is more than 50 years old!) at the University of Dartmouth (USA) by Professors John Kemeny and Thomas Kurtz, who developed this language to facilitate the learning of computer science for their students. THE BASIC spread very quickly thanks to several factors but, mainly, because its inventors decided not to patent it and keep it confined in their university: they also did an intense work of dissemination also at other institutions so that many people began to use it. The fact that it was a language that did not require much resources (the first interpreters occupied a few KB of memory) made it perfect for use on microcomputers that began to spread in the 70s and the use of a generic language led people to write programs for the most varied tasks and to publish them also in the magazines and newsletters of computer enthusiasts clubs of the time. Microsoft itself began its commercial ascent by selling a BASIC interpreter, initially for the Altair 8800 and later distributing it for each type of machine. Microsoft BASIC became so popular that it eventually established itself as a standard and many computer builders, such as Comoodore, Apple, and Atari, decided to adopt it directly or use a BASIC interpreter derived from it. Using A BASIC interpreter was an obvious choice at the time: it allowed you to have a very common language when turning on the machine that could be used after a short learning period.

The LM80C BASIC is derived from Nascom BASIC, which is derived from Microsoft BASIC. Nascom BASIC was a BASIC interpreter that Nascom Computers, an English manufacturer active in the late 70s of the 20th century, distributed for its computers and terminals. It was derived from the Z80 version of Microsoft BASIC and maintained great compatibility with it. And so it is for the LM80C BASIC: apart from the differences related to different hardware architectures, a program in Microsoft BASIC can run with a few modifications on the LM80C. I chose Nascom BASIC as my starting point because Nascom's computers were so widespread that in the early 1980s a thriving community had formed with several dedicated publications: the interpreter's complete and commented source was published on one of them. The LM80C BASIC, like Microsoft BASIC, is an interpreted language (although BASIC was born as a compiled language): this means that behind there is no compiler that first transforms the source into machine code executable by the CPU but there is an interpreter that analyzes each command entered and, once recognized, executes the corresponding procedures in machine language.
Returning to our initial example, the interpreter will first encounter THE SCREEN 2 statement. Once the command

is recognized as one of the valid ones, the interpreter will call the corresponding procedure which, after verifying the correctness of the syntax (for example, it will check that the requested video mode is valid), will set the requested video mode by running specific sub-procedures, always in machine language, thanks to which the necessary data will be sent to the video chip. So here are the different layers of system software that we mentioned earlier. Once the instruction has been executed, the interpreter will continue the analysis of the command line received, and will find the character of the 2 points: this is interpreted as a kind of warning that signals to the interpreter that "there is something else", that is, that the interpretation of the line is not finished. Moving on, YOU'll find THE CIRCLE 128,96,50,8 statement. Here the situation is slightly more complex because the parameters are not only higher but the last one is also optional, i.e. it can be passed or not: in thelatter case the computer must use the default color. Once the syntax analysis is finished, the execution of a very complex machine language routine begins that calculates the coordinates of each single point constituting the circumference. Once the coordinates are found, a second procedure is called, of a lower level, which displays a point given a pair of X,Y coordinates. The first analysis that it makes concerns the control of the visibility or not of the point, that is, whether the past coordinates fall on the visible portion of the screen or not: in this case the point is not drawn. If the coordinates are valid, another procedure transforms the coordinates into the address of the video memory cell that contains the pixel to display and then turns on the corresponding bit and sets the required color. All this is repeated for all points of the circumference. At the end of this command, the interpreter will parse the last line and, finding no other command, return the control to the user. It will be returned to what is called "direct mode", a special mode where each command entered is executed immediately. Actually things are a little more complex than that...

Firmware

It is not true that only by pressing the "RETURN" button (or "ENTER" on the machines that came from the United Kingdom) the system executes the entered commands, because there is a program that runs continuously, and it is the one that manages the interface with the user: it continuously analyzes all the input peripherals waiting for a data coming from outside. For example, the system reads the keyboard at regular intervals and displays the character corresponding to the key pressed on the screen. If the program is requested to start in memory, the control passes to the interpreter, who will execute the list line by line. But even in this case the firmware stays "on the alert" because it continues to manage the display of printed characters on the screen, moves any sprites, generates sounds when required, reads the keyboard for input and so on. The complex software of a computer stored in THE ROMA that serves its operation is called firmware: this English word, which derives from the union of "firm" and "software" and which in Italian sounds almost like "permanent logical part", indicates the software preinstalled in a system that, usually, the end user cannot or is not able to modify and that deals with the startup of the

```
KEY:    dec     HL              ; dec 'cos GETCHR INCs
        call    GETCHR          ; Get next character
        jp      Z,LSTKEYS       ; jump if nothing follows
                                ; change FN keys
        call    GETINT          ; get a number
        and     A               ; is it 0?
        jr      NZ,KEYCH        ; no, jump over
        push    HL              ; yes - reset FN keys to defaults
        push    DE              ; store HL & DE
        ld      HL,DEFFNKS      ; pointer to default FN keys texts
        ld      DE,FNKEYS       ; pointer to destination
        ld      BC,$0080        ; 128 chars to be copied
        ldir                    ; restore default texts
        pop     DE              ; retrieve DE
        pop     HL              ; retrieve HL
        ret                     ; return to caller
KEYCH:  cp      $09             ; is it >= 9?
        jp      NC,SNERR        ; yes - syntax error
        dec     A               ; FN key in range 0~7
        add     A,A             ; multiply A...
        add     A,A             ; ... times 4...
        add     A,A             ; ... to get the correct...
        add     A,A             ; ... offset fo FN key text
        ld      (TMPBFR1),A     ; store FN key offset...
        xor     A               ; ...in a...
        ld      (TMPBFR1+1),A   ; ...16-bit register
```

Figura 2: The beginning of the routine related to the KEY command that allows to manage the tasks assigned to the function keys.

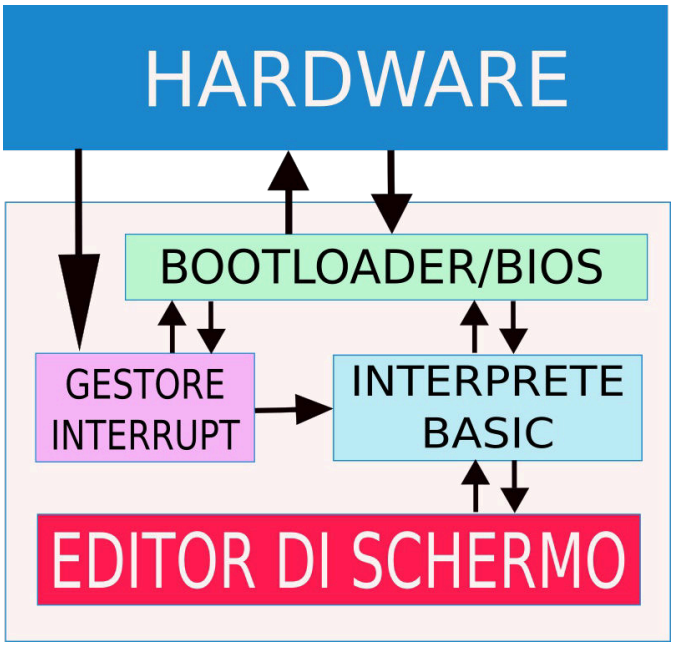


Figura 3: The structure of the firmware

machine and the management of its basic functions.

The firmware of the LM80C is not only made up of THE BASIC interpreter which, however, occupies much of the embedded code. There are several portions (Figure 3):
• the bootloader, which is called when the computer is reset and is responsible for initializing the hardware;
• the BIOS, composed of a whole series of low-level functions that allow direct access to computer hardware;
• the interrupt manager, which manages both the system interrupt, which is called at regular intervals and which deals with all the timed operations of the system such as the periodic reading of the keyboard, the deactivation of the playback of sound tones, the flashing of the cursor, and other, as well as the interrupts raised by the peripherals of the computer;
• the actual BASIC interpreter that can work both directly and indirectly;
• the screen editor, which manages the printing of characters on the screen and the movement of the cursor, acting as a real interface between user and machine.

The bootloader is a few lines of code. It directs the CPU to a specific point on THE ROMA after a reset and from there the system starts starting: the logo is displayed, the system timer used as a timer for the operations to be repeated at regular intervals is set, the interrupt vectors for the peripheral chips and the standard video mode are set, and then the control is left to the interpreter. The interpreter does not dialogue directly with the user, but the user interacts with the computer through the screen editor. The editor allows you to move the cursor and insert commands at each point of the video. The keys are read by a routine called by the interrupt handler, which scans the keyboard at regular intervals to see if the user has pressed anything. If it is a common key, the character matcher is displayed in the cell occupied by the cursor: to do this several BIOS routines are called that dialogue directly with the video chip. These wheels transform the X and Y coordinates of the cursor into the address of the corresponding memory cell and video and then write the data to be displayed in it. If the “RETURN” key is pressed then all the text on the line is passed to the interpreter who analyses it and executes any commands. Commands also rely on BIOS to perform their functions: for example, to play a note or write a character.

The core of THE BASIC interpreter was available on the network but, as mentioned, it was developed for a computer of terminal type, that is, with which it is possible to interface through a serial link and which accepts and returns data on that channel. For this reason, the initial firmware was just under 8 KB and did not offer many features. Drafting the screen editor took several weeks of work and KB of code (including character maps for 6x8 and 8x8 pixel fonts, drawn entirely by hand). Added to this was the work to make the system serial independent and transform it into a real home computer, with input passed from the keyboard and output displayed on the screen. In addition to this you add all the code written to manage all the new graphical and audio commands of the computer that, of course, were not present in a terminal. In the end the final work led to 19 KB of firmware, and the work of drafting the code was longer and more demanding than that for the realization of the hardware part. This is to make it clear that if you decide to go down the road of creating a computer from scratch you have to consider that this process is not only about choosing chips and designing a motherboard but also awaits you a lot of work of drafting the code to make it work and be able to use it and make the most of it.

Useful links
• Project reference website:
http://www.leonardomiliani.com/en/lm80c/
• Electrical diagrams and firmware source code:
https://github.com/leomil72/LM80C
• Hackaday page:
https://hackaday.io/project/165246-lm80c-color

NINTENDO 64 - Mario's 3D revolution

by Carlo N. Del Mar Pirazzini

The Nintendo 64 was Nintendo's latest cartridge console, the fifth generation console born as heir to that Super Nintendo that revolutionized the world of video games during the 1990s.

The N64 was produced by Nintendo between 1996 and 2002, first released in Japan and later in the United States (in September 1996). It arrived on the European market in March 1997. For many children born between the late 1980s and early 1990s it was the most coveted gift for Christmas and for a time it was the number one rival of Playstation 1, but then came second in the console war of that generation (annihilating the Sega Saturn in sales).

The development of titles for this console ended in 2002 with the publication of The Legend of Zelda Wind Waker and Resident Evil 0, also released in an improved version on GAME CUBE (we talked about the GAME CUBE on RMW 22 Italian issue, ndR).
With 32.93 million specimens sold worldwide, it is still an impressive number. The Nintendo 64 actually marked a real revolution in the field of video games, offering the first real three-dimensional gaming experience with high quality titles and still authentic milestones of their kind, and had positive results all over the world. However, in the last years of sale it was surpassed by rival PlayStation, which set a record selling over 70 million units, but with results significantly higher than those of the Sega Saturn whose sales reached only 9 million units.

The machine is presented as a rectangular box of black colour (but over the years special editions were also released characterised by different colours and also transparent, ndR) with rounded edges and which has a slot on the upper side for inserting the cartridges containing the games and on the front 4 connectors for gamepads, without buying external peripherals.

Function, as we said, by means of cartridges that are directly inserted into the machine. This system allows to store less data than CD-ROMs, but allows minimum loading times, possibility of saving without the aid of memory cards, presenting a cartridge recognition system through an electronic circuit called Checking Integrated Circuit flanked by a Serial Peripheral Interface type bus. During the period she was criticized for two things. The high cost of cartridge games and precisely the non-use of CD ROMs. It was a historical period where the CD ROM represented the future, it was very cheap and it was possible to use consoles equipped with a cd rom player as dvd players.
Features four gamepad outputs, so you can play with multiplayer titles for four players without having to buy additional media like Sony's Multitap. Below the gamepad there is a slot that allows you to insert additional devices. In front of the console there is a slot to insert a new RAM bank. Unlike its predecessor (the Super Nintendo), the Nintendo 64 cannot output an RGB video signal, nor, in the European version, even S-Video. This means that a European N64 has composite video as the best possible output video signal. Below the console there is a slot used to connect the console with the Nintendo 64DD, hardware expansion output only in Japan.

Fig. 1 - Nintendo 64 and its controller

True innovation comes through the joypad, among fans called tricorn because of its particular shape with 3 handles to grab it, presents:

An analog lever in the center of the controller, with octagonal base, on the center handle.
A digital cross to the left of the controller.
Six front keys, two of which are called A and B (blue and green, respectively), the other four are yellow, the middle of which is the letter C and on each of them there is a directional arrow (in the Nintendo GameCube controller they were then replaced by a second analog lever, called stick C).
3 back keys, L and R on the left and right respectively, and Z on the back of the controller's central handle.
START button in the middle of the controller.

Expansion at the top of the back of the controller, designed to accommodate three additional optional accessories:
the Rumble Pak, which allowed a vibration function.
the Pak Controller, an additional memory used by some games, which allowed the saving of optional data.
the Transfer Pak, an accessory that allowed you to transfer some game data between the Nintendo 64 and the Nintendo Game Boy, the Nintendo portable console. It was poorly supported, as it allowed the transfer of photos taken with a Game Boy accessory, the Game Boy Camera, and the

transfer of data between the Pokémon Stadium and Pokémon Stadium 2 titles with the three Game Boy Pokémon titles present at the time (Pokémon Red, Blue and Yellow).

It can be used in two configurations: in the first you use it with your hands on the external handles, neglecting the analogue lever and the Z key, in the second you prefer the central handle going to exploit these two elements, neglecting the digital cross and the L key. However, it was generally the games that "imposed" one of the two game configurations. The game The Legend of Zelda: Ocarina of Time introduced a game technique, currently used in recent games, extremely innovative, which allowed, for example in a fight against an enemy, to "hook" the target in question, or have the camera automatically frame it in every game situation, regardless of the movements made by the protagonist character and the target itself; it is called Z-targeting, precisely depending on the fact that this"hooking" is done by pressing the Z key.

The other two major innovations of this controller were the introduction to the mass video game market of the analogue lever and the vibration function (optional); given the importance of these two new features, Sony later also equipped the PlayStation with a revision of its controller, in fact creating a second version, the DualShock, which now had two analogue levers and the integrated and self-powered vibration function.

DEVICES

During the Nintendo 64 lifecycle, several official console add-ons were produced:
Pak controller: a 256 kB memory that was inserted under the pad, divided into 123 "pages". Subsequently Nintendo, realizing the minimum size of this memory card, decided to create other versions from the size of 1 to 4 MB.
Expansion Pak: This is a 4MB RDRAM bench (Rambus DRAM) that goes alongside the original 4MB bench mounted on-board, this allows you to reach 8MB of RAM, allowing you to use high game definitions and many other improvements. Although it is supported by several games, some of these can only work if the Expansion Pak has been installed on the console, while others need the Expansion Pak in order to activate some extra features of the game.
Rumble Pak: it is an accessory that vibrates in the gamepad during the game following events generated in the game environment. This accessory is now considered a standard available for all consoles of the latest generations. It is in fact a motorcycle powered by 2 mini-style batteries.

Fig. 2 - Nintendo 64 can count on a very wide range of titles with several Nintendo exclusives

Technical Specifications

Central Processing Unit:
4.6 million transistor @ 93.75 MHz NEC VR4300-64 bits
24 kB top-level cache
MIPS/R4300i RISC architecture
Total CPU Math Capacity: 93.0 million operations per second
Manufacturing process at 0.35 µm (micrometers)

Graphics Processing Unit:
Silicon Graphics-RPC Reality 64-bit @ 62.5 MHz Coprocessor
Vector unit for 8-bit integers
Peak graphical rendering equal to 150,000 polygons per second with all processing[8], 600,000 monochrome polygons per second.

Processing:
Z-buffering
Anti-aliasing
Texture mapping
Linear filter (bilinear and trilinear)
Mip-mapping
Gouraud shading
Fillrate in pixels of 30 megasamples per second with Z-buffer
16.7 million colors (32,768 on screen)

Resolutions: 320 × 240/640 × 480 pixels
GPU Performance: 600 MegaFlops

Memory:
4 MBytes RDRAM
System Bandwidth @562.5 MB/sec
Latency of 640 nanoseconds

Sound:
16-bit stereo
100 Linear PCM channels (maximum 16-24 high quality). Each channel occupies an entire CPU cycle
Sampling frequency at 48.0 kHz
Supported formats: MIDI, MP3, ADPCM and Tracker

Storage Media:
Electronic cartridges from 4 to 64 MBytes
Controller with analogue stick, vibration function and input for memory card (Controller Pak)

Input/Output:
4 ports per controller
Expansion Pak Port

Transfer Pak: is an accessory that inserted in the controller allows you to transfer game data between the Nintendo 64 and the Game Boy, the Nintendo portable console. Actually, this accessory had only one purpose, namely to transfer the images captured with the Game Boy Camera to the Nintendo 64 but was later used in Pokémon Stadium games to pass data from the Game Boy to the Nintendo 64.

The Nintendo 64 can count on a very large stock park with several exclusive Nintendos that were the real engine of the CONSOLE and that allowed you not to fall prey to the overwhelming SONY (which prevailed at that time).

The bad notes that didn't allow Nintendo to overrank his rival Sony? The inaccessible price of cartridges, the policy of never lowering the prices of older stocks and the lack of some third-party stocks that landed in Sony (Final Fanasy 7 was a real Killer Application for Sony, ndR).

This does not detract from the fact that there were many remarkable games for this console and below, since we are at Christmas you will find the list of the 5 basics for this console.

CORE TITLES

MARIO 64 - Mario's first 3D adventure and perhaps one of the most loved adventures by players. He fights the palm with Super Mario World of the best Mario ever. A very big game full of secrets. Absolutely worth a try.

Mario Kart 64 - You will find the review in this issue ;).

The Legend of Zelda: Ocarina of Time/Majora Mask - Two different JRPGs in style but beautiful! The 3D experience of the Nintendo saga is wonderful. Maybe the highest gameplay peaks in the series.

Goldeneye - FPS based on 007 series. It was revolutionary. The title developed by Rare, sold more than 8 million copies attesting to being the best-selling third party title on Nintendo 64, but it was the fully explorable three-dimensional levels and multiple goals to complete freely that conveyed a feeling of immersiveness never experienced before in a video game.

Perfect Dark - Another FPS spiritual sequel to Goldeneye. It takes what's best in the Rare game, expanding it and adding an incredible plot. Graphically beautiful at the time.

Mario 64

The Legend of Zelda: Ocarina of Time

The Legend of Zelda: Majora Mask

Perfect Dark

UNO2IEC HOST interface cable

A simple Arduino UNO-based interface for connecting 8-bit Commodores to your PC

by David La Monaca

0. Introduction

The UNO2IEC HOST is a low-cost project, and therefore somewhat limited, designed to simulate the presence of a 1541 drive connected to all 8-bit Commodore computers equipped with a 6-pin DIN serial port. Initially the CBM-1541 floppy drive was designed to be a worthy companion to the C64, the best-selling computer in Commodore history. At first it was a "single-sided" Floppy Disk Drive (it means that it only uses one side of the floppy at a time) with 170KB of space on 5¼" disks. Not so different from what was on the market at the time; the problem was that it had no real interface. It was connected through the serial (IEC) port of the C64, which used an I/O controller (the integrated MOS 6522) that was fast enough on its own, even if it had some bugs when trying to increase performance. But to make matters worse, since the project was well behind the launch of the C64, Commodore boss Jack Tramiel and his marketing department decided that the 1541 had to be backwards compatible with the VIC-20. This forced Robert Russell (the design engineer) to reduce the speed even more than planned. His 1541 drive, initially designed to be used with a high-performance serial line MOS 6526, might have been the fastest of its era, but it turned out to be among the slowest, if compared to the 8-bit competitors of the 80s. It took more than two minutes to load 64KB of data into memory, which was unconceivable. But the story didn't end that way. In fact, as many people know, the CBM-1541 is basically a complete computer because it has a 6502 CPU, dedicated ROM and a little RAM, therefore only the video chip is missing! So, over time, some programmers used the disk drive and its hardware as a co-processor, entrusting it with some computing tasks or turbo-loader management for their programs or demos!

1. Hardware implementation

Let's get straight to the point. The idea is to connect an Arduino UNO (or NANO) board to a C64 - later we will see that the compatibility of any 8-bit Commodore with a standard serial port is total - through the IEC interface to simulate the presence of a 1541 disk drive. The UNO2IEC project I refer to is Lars Wadefalk's [R1] which I discovered recently although it dates back to 2013. Divided into three phases, the project requires the assembly of a cable, the compilation of a program that will act as a "disk image server" and the preparation of an Arduino board as a host between the PC and the Commodore. As said, the same project works on C16/Plus4, VIC-20, C64 and C128 and can also be used with a FileBrowser that accepts SD2IEC commands, whereas compatibility with FastLoader or JiffyDOS cartridges has not been implemented yet.

Let's start with the list of components needed to build the cable:

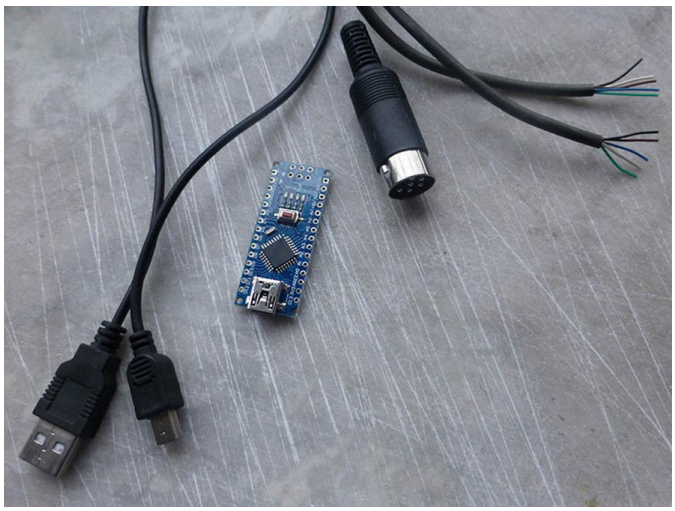


Fig. 1 - The components to build the cable

- A piece of 5-wire cable
- 1 DIN 6-pin male connector
- 1 Arduino UNO board (or NANO, less expensive)

Assembling the cable is quite simple: the IEC connector has 3 signals we need (CLK, ATN, DATA) and an optional one (RESET). These signals can be connected to any digital pin on the Arduino. Just note them down and then modify them appropriately in the server program interface. In my

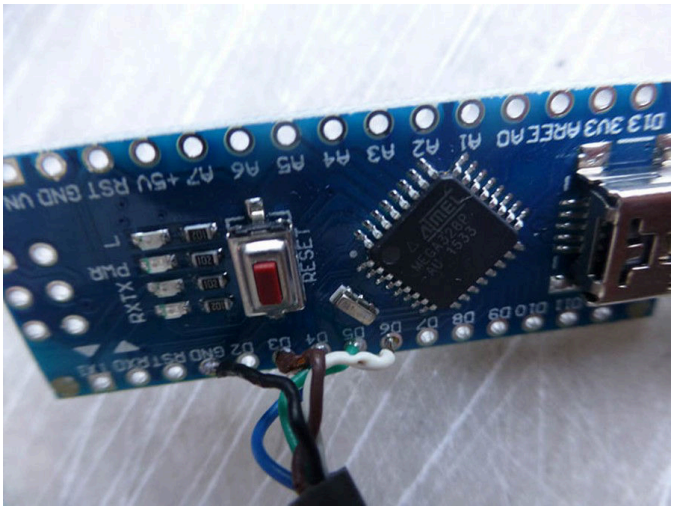


Fig. 2 - The wires soldered on the Arduino UNO board

case, digital pins 6, 5, 4 and 3 were used, for the sake of simplicity.

As far as programming the Arduino board is concerned, it is necessary to have the Arduino build environment available (download it from www.arduino.cc). It's all very simple, just follow the excellent "Getting Started" guide on the [R3] site.
Unzip the project from [R2] into a folder on your system and look at the folder named "uno2iec" which contains

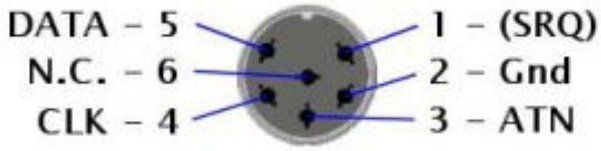


Fig. 3 - UNO2IEC cable finished and ready for use

the sources needed to compile and upload the necessary firmware to the Arduino board. Open the folder and double click on the file "uno2iec.ino": the file will open in the Arduino build environment. Then just press CTRL+U to start the compilation and automatic upload to the board previously connected to the PC through the mini-USB port.

2. Software installation

The UNO2IEC HOST control program is an open-source software [R2] that can be compiled for the major platforms: MS Windows (from XP to W10), Mac OS X, Linux and RPI. The package for Windows is already available in binary version [R4]. For the other platforms the sources can be compiled, taking care to include the Qt graphic libraries needed to manage the visual interface of the program.
The UNO2IEC HOST software is the tool with which we interface our 8-bit Commodore computer with the PC through the use of a USB port (USB 2.0 and 3.0 ports are supported). The program allows us to select the D64 disk image we want to use to load and save programmes, play games and generally transfer data to and from our beloved

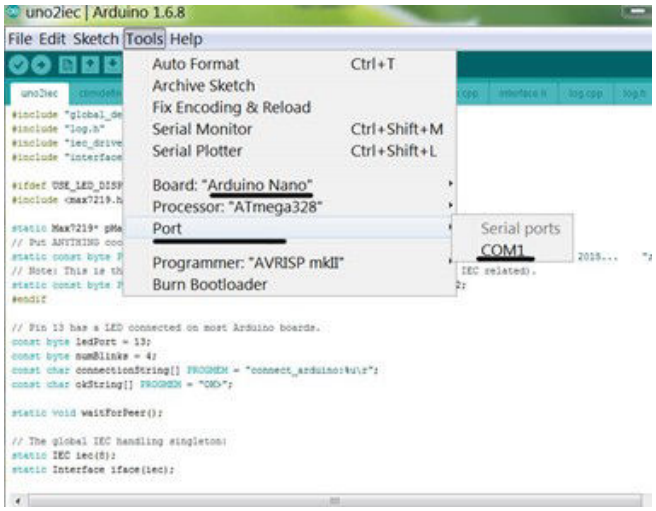


Fig. 4 - Compiling and uploading code to Arduino

Commodore. Once the disk image is selected, it's possible to load applications and programs inside the D64/T64 files using the standard commands (LOAD, LIST, RUN, DLOAD, DIRECTORY, etc.) provided by the different BASIC implementations on the 8-bit Commodore machines.
The actual installation of the "server software" is as simple as can be. For Windows, just unzip the zip package to a folder of your choice and run the executable "uno2iec.exe". There is nothing else to do. The same applies for other systems: once compiled with the Qt libraries, for OS X you will find a file "uno2iec.app" and for Linux an executable "uno2iec" in the main folder of the compressed package, no installation procedure required).

3. COM port configuration

Before using the UNO2IEC cable it is necessary to configure the USB serial port to which the cable is physically connected. And before configuring it you need to... find it! Nothing complicated, though. Just go to your OS device panel and read the port name. For Windows just open the device configuration and find out which COM port the operating system has assigned to the C64 communication cable. Specifically just open the Control Panel (Start > Control Panel) and then select Device Manager > Ports (COM and LPT). At this point in the list you will see the serial port reserved by the operating system, e.g. USB-SERIAL CM340 (COM5).

In your case, a different COM port may be assigned (COM6, COM7, COM9, etc.) depending on the configuration of the PC you are using. Mark down the COM port that Windows has selected for you and proceed to configure the UNO2IEC software.

4. UNO2IEC software configuration

Ok, now launch the software and set the basic operating

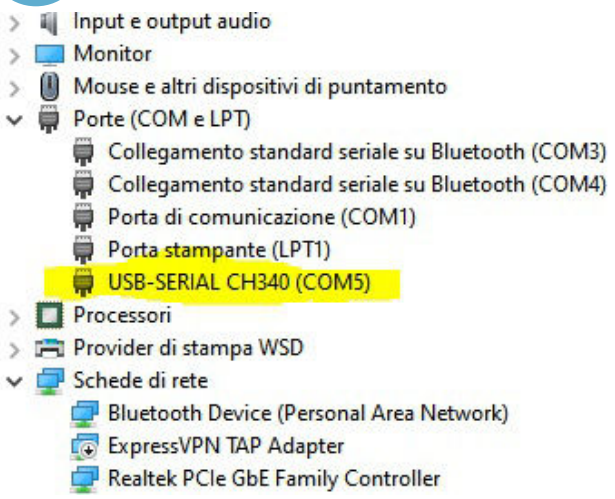


Fig. 5 - COM port detection in Windows

parameters. Once the program is open, select the display interface from those available. By default the theme is Commodore 64, but also VIC-20, C128 (also 80 columns) and PLUS/4 are available. From the Main menu select Directory Listing Theme > Machine > Vic-20 | C64 | C128 (80 col) | Plus4. This setting is nothing else than a way to set the theme of the software control window when using D64 files. The control window will display the contents of the virtual disk that is 'mounted', i.e. selected.

Main > Directory Listing Theme > Machine > C64 (default)

Let us now configure the communication port settings that we have found in the Device Manager of the Control Panel. Select Main > Settings (Ctrl+O). Set the COM Port according to what we have found (in our case COM5) and leave the Serial Speed at 57600 bit/sec for now.

In the next line we select the following parameters:

Device Number: 8 | 9 | 10... [4-30] - corresponds to the ID of the usable drive

Reset Pin (optional): 6 - not supported by all Commodore machines

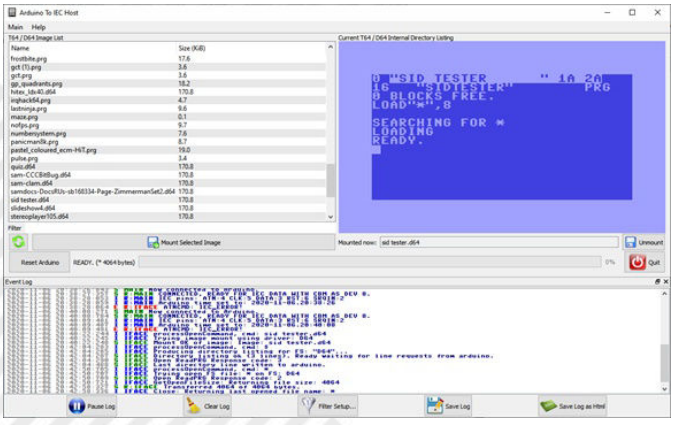


Fig. 6 - The UNO2IEC Host program in action

Clock Pin: 5, ATN Pin: 4, Data Pin: 3

All these parameters may already be in place, but if so, modify them as above. Basically we tell the UNO2IEC software how the pins of the Arduino UNO board on our cable are configured, precisely by function (reset pin, clock, data pin, etc.).

Then we choose the folder that contains the D64, T64, PRG, etc. files on our system. Basically the program needs to know where your collection of programs and games for your system is located. E.g. Folder for D64/T64 Collection: C:\TEMP\RETROGAMES\C64

5. Ordinary Use

Once everything is confirmed with the OK button, the left side of the main UNO2IEC software window will show the list of files (virtual D64 floppy disks, T64 tapes, individual

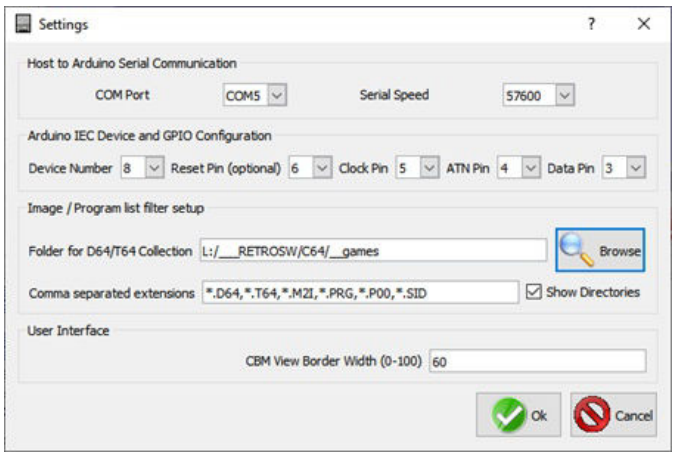


Fig. 7 - Communication and pin settings for the Arduino board and other parameters

PRG programs, etc.).

To mount a D64 or T64 file just select the file with a double click or press the Mount Selected Image button. The directory of the selected image file will appear in the control window at the top right. At this point you can switch to the computer keyboard and type in the commands to load the programs, exactly as you would do with a 1541 drive connected to the machine. So, for example, to load the directory of the virtual diskette type: LOAD"$",8 and press Return. A simple LIST followed by Return will display the list of programs. Obviously the drive ID depends on what you have selected in the UNO2IEC program settings. To

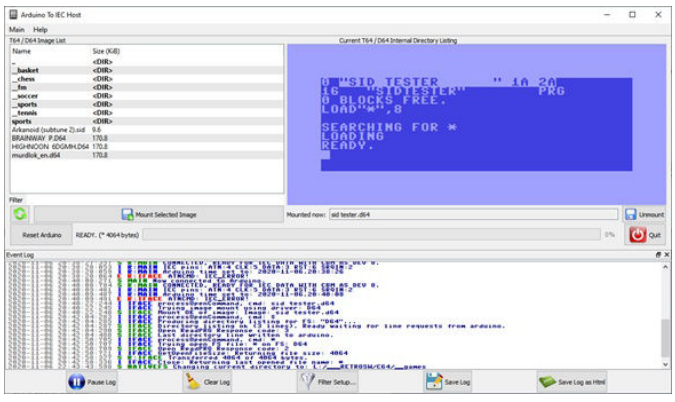


Fig. 8 - Selecting and mounting an image file

load a program use LOAD "*",8 or LOAD "*",8,1 followed by Return. Of course, the command syntax for the 1541 drive applies exactly as if you had a physical drive connected to the serial port of your Commodore. In case you use computers with advanced serial drive commands (like C16, Plus/4 and C128), these are fully supported.

Also in case of T64 file mounts, just issue the LOAD "*",8 command to load the program contained in the tape image. If you have ready-made PRG files instead, you must select the Windows folder that contains them and then switch to the 8-bit computer keyboard and load them with LOAD "programname",8. Remember that using the asterisk character means loading the last file that the computer had previously loaded. Obviously, if the computer has just been turned on or reset, LOAD "*",8 will load the first program in the directory of the mounted floppy disk. For the record, the correct command to make sure you load the first program in the disk image is LOAD":*",8 (i.e., adding the colon character before the asterisk).

Of course, the loading or saving speed is similar to that of a physical drive connected to the serial port or other SD2IEC interfaces, so we can expect nothing different. During the loading or saving phases, a handy control bar will help us to understand where we are, while also giving us an idea of the current transfer rate. The transfer rate can be increased by adjusting the Serial Speed parameter under Main > Settings, but it's hard to set a higher speed because of the 2KB of RAM on the Arduino (512 bytes of which are used as a data buffer to load the disk sectors). Even loading the PRG version of JiffyDOS before the necessary loads or inserting a FastLoader cartridge on our Commodore does not give better results in transfer rate, although the project developer [R1] and other users are still testing to improve transfer rate.

A useful feature of the program is the ability to reset the Arduino board, in case of application or PCB freezing. If we are using a C64 where the serial port supports the reset pin (early models had this pin assignment, the C64c does not), then our C64 will be reset as well: useful when we want to switch between games or D64 images.

6. Conclusions

Before the advent of USB technology, the connection interfaces between Commodore computers and PCs with serial and parallel ports represented a convenient and cheap solution (e.g. 64HDD, XA/XE cables, XUM 1541). Nowadays the UNO2IEC interface is probably not the easiest and most straightforward way to use "modern" mass storage on old Commodore computers, but it is certainly one of the cheapest, if not the cheapest ever. The material to build one is really basic, easy to find and inexpensive. Building the cable and 3D printing a box to protect the Arduino PCB can cost between 8 and 12 euros, including the case. Of course the basic performance is more or less the same as an original 1541 drive or one of the many SD2IEC interfaces on the market, but, avoiding the comparison with more powerful and elaborate interfaces like the 1541 Ultimate II+ or the Pi1541 (which guarantee a compatibility level with the original drive around 99%), for beginners or for those who do not want to spend money on an original drive and get tangled up with floppy disks, cleaning and misalignment of the heads, UNO2IEC HOST is undoubtedly a more than effective solution.

A special (and mandatory) thank you goes to **Carlo Piacentini**, well known in the Italian retrocomputing scene, with whom I have actively collaborated for this little project and who has made for me (and for himself) some samples of UNO2IEC cables of excellent workmanship, finding also the time to make a comfortable 3D printed protective case. I would also like to thank **Emmanuel Barraud** (aka **Klyde**) for his support and for having also built some UNO2IEC HOST interfaces for himself and for some retro-commodore friends.

Fig. 9 - The UNO2IEC cable with its protective case

References

R1 - The author of the project: https://github.com/Larswad
R2 - The source package of the project: https://github.com/Larswad/uno2iec/releases
R3 - The Arduino compiler: https://www.arduino.cc/en/Main/Software

R4 - The UNO2IEC HOST program for Windows: https://www.retromagazine.net/download/uno2iec/uno2iec_host.zip
R5 - The UNO2IEC HOST program for Mac OS X: https://www.retromagazine.net/download/uno2iec/uno2iec_osx.zip

Compiling the Uno2IEC Host software

The software part of the solution is developed in C with a QT graphical interface and it is available for Windows, OS X, Linux and Raspberry PI. Compiling the software is not complicated for any platform, except perhaps for Windows, for which a ton of programs must be downloaded including the graphic libraries, the C compiler and a suitable IDE. To make life even easier for all our interested readers, we have provided a Windows ready-made executable file with its static QT library. For Mac OS X, Linux and RPI systems, however, you can follow these quick guides.

Mac OS X

For the beginners, the OS X kernel is based on BSD, more precisely on a kernel called Darwin, which is free and also periodically updated and modified by an active community. Since it is a near-standard BSD, a group of programmers ported/created the BSD package manager for OS X; the project is called Brew and is used like Apt/Yum on BSD and Linux/Debian or Linux/CentOS distributions. We will use Brew to install the Qt5 libraries and then proceed to compile the project. Installing Brew and Qt5 is relatively straightforward but XCode is also required to be picked up and installed from the AppStore.

Once everything is set up correctly, download the "uno2iec-0.5.0.zip" package from [R2], unzip the contents of this file and a "uno2iec_src" folder will be created. Open a terminal, access the created folder and type the following commands:

```
$ brew install qt5
$ qmake "CONFIG+=release staticlib".
$ make
```

After several minutes and if all goes well, a new folder called "release" will be created. Inside this folder you'll find the executable "uno2iec.app", just copy it into the Applications folder of your Mac and run it.

As you can see, compiling under OS X is fairly straightforward but the whole process often depends heavily on your system configuration. I have therefore enclosed a binary version of the app already compiled under my VM Mojave [R5]. Download the file "uno2iec_osx.zip" and open it, placing the folder "uno2iec_osx" somewhere. A double click on the file uno2iec_host,app will start the host program. From the main menu, choose "RP2IEC > Preferences" to set the interface parameters. If the program does not start, make sure you have installed the Qt5 library and add the command:

```
$ ln -s /usr/local/opt/qt5 /usr/local/opt/qt55
```

This command is used to overcome a path problem in storing the Qt5 graphics library in the system, which is essential for the program to function correctly.

Linux / Raspberry PI

For Linux users things get even simpler! Using one of the Debian-based distributions that offer "apt" as a package manager (you can use any distro, as long as you know what you're doing), we just have to install Qt5 and the development libraries and then proceed with the compilation. In my case I used a LUbuntu 20.04 distribution, for which it was necessary to force the installation of the QtSerialPort module. The "qmake" command was also missing, but a "sudo apt install qtchooser" was enough to fix it.

Download the "uno2iec_src.zip" package from the [R2] site - specifically version 0.5.0.0 - and unzip the contents of this file into a directory in your home directory. A "uno2iec_src" folder will be created. Now open a terminal, access the newly created folder and type the following commands:

```
$ sudo apt install qt5-default
$ sudo apt install libqtSerialport5 libqtSerialport5-dev
$ qmake "CONFIG+=release staticlib".
$ make
```

As with OS X, a new folder called "release" will be created. Inside this folder will be the executable "rpi2iec". Just rename it and copy it wherever you want on your Linux (e.g. /usr/bin) and you will have the host program ready to use.

The compilation process for Raspberry PI (all versions) is similar to the above. We just recommend using the Raspbian distribution.

An introduction to **MEGA65**

by Gianluca Girelli

As this magazine testifies very well, our beloved retro-computers are experiencing a second (or perhaps third) youth. One of the reasons is probably that none of them used moving parts that could have suffered a mechanical failure, so they managed to resist more or less unharmed over time. However, all their storage peripherals had some failure, so in recent years many manufacturers have explored ways to make good use of our old companions again by means of state-of-the-art storage and network navigation peripherals.

The good thing about such devices (such as the Ultimate II cartridge, just to name one) is that they allow you to get your hands on old computers, but their capacitors and other components are still aging and will, one day, break. Thanks to many "visionaries", however, such as the guys of MEGA (Museum of Electronic Games and Art), a brand new 8-bit project was launched: building a computer based on the C65, working about 50 times faster than a C64 while remaining highly compatible. C65 design, mechanical keyboard, HD output, SD card holder, Ethernet, extended memory and other features will soon increase our fun without spoiling the 8-bit feel.

The MEGA65 is a completely open-source project and, unlike other FPGA-based computers, its implementation can be reviewed, modified and improved. This FPGA implementation is based on a reproduction of the large scale compatibility of the old MOS 6502 microchip, including illegal OpCode. This means that MEGA65 is designed to offer compatibility with both C64 and C65 and, as it is open-source, its compatibility will continue to improve over time.

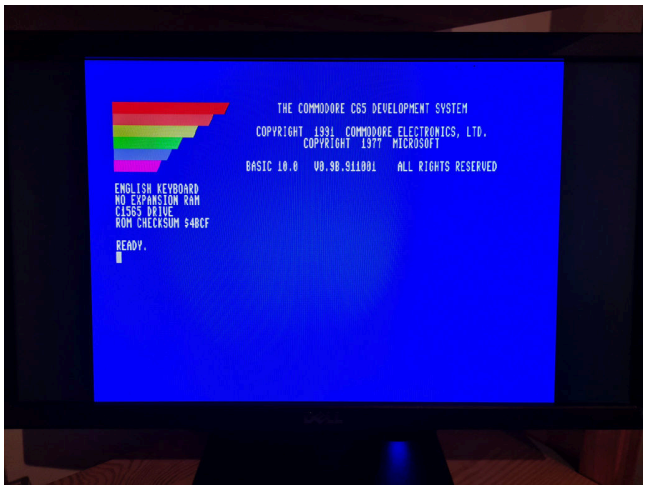
Unlike other products out there, this is not meant to make us float in nostalgia: although this is a very likely and very desirable result, you can really go back to programming as in the old days thanks to a large library of easy-to-learn programming languages, drawing programs and various tools focused on supporting our creativity.

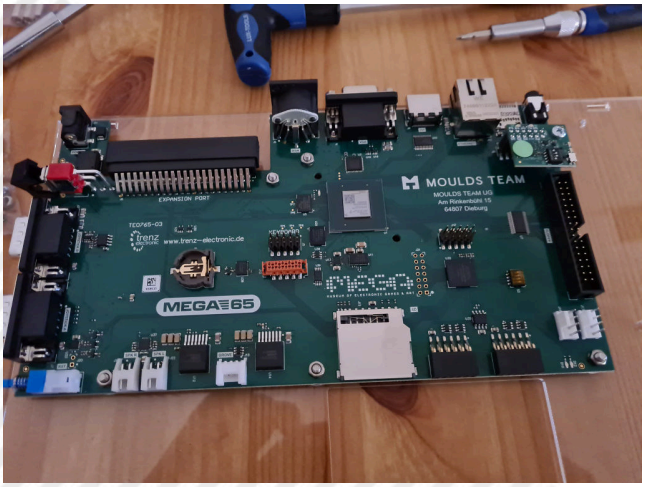
The project was announced in 2015 and was due to be completed within a few months. To date, a release date has not yet been announced, but developers are really trying their best and the results are now tangible. In addition, a portable version of the MEGA65 is under development and its features are simply unique: a dual 4G (and upgradeable 5G) mobile phone with unprecedented battery life (about 1000 hours standby), built-in 2W stereo speakers and 1 second shorter start-up time.

Like any project of this type, which relies heavily on direct support from fans, this too had a moment of pause, just as the relaunch campaign for the purchase of injection molding equipment to build the shells for the MEGA65 was launched. Such a machine is very expensive and even though the community donated over €20K, and a goal of 66K seemed almost unattainable, until... well, until the day that over €40K was raised on a weekend!

After the success of the fundraising campaign, development kits became a reality within a few months and were now sent to developers. Courtesy of Stefan Vogt, creator of cutting-edge multi-platform textual adventures, we now have the opportunity to take a closer look at this beauty.

Images in this article show the bottom plate and drive, motherboard and its body, with the drive mounted, along with the details of the board and keyboard.
The assembled devkit seems really magnificent, and although some people still think that all this is just an expensive toy for geeks and nerds (which I am proud of) let me point out that Tristam Island, Hugo Labrande's last textual adventure released just a few weeks ago, already runs on a MEGA65 and so it will be for Hibernated2 by the aforementioned Stefan Vogt. I can't close this article without thanking Paul Gardner-Stephen and everyone else at MEGA65.org for making this possible.





An **illegal opcode**, also called an **undocumented instruction**, is an instruction to a CPU that is not mentioned in any official documentation released by the CPU's designer or manufacturer, which nevertheless has an effect. Illegal opcodes were common on older CPUs designed during the 1970s, such as the MOS Technology 6502, Intel 8086, and the Zilog Z80. On these older processors, many exist as a side effect of the wiring of transistors in the CPU, and usually combine functions of the CPU that were not intended to be combined. On old and modern processors, there are also instructions intentionally included in the processor by the manufacturer, but that are not documented in any official specification.
https://en.wikipedia.org/wiki/Illegal_opcode

The MOS VIC video chip

by Leonardo Miliani

In this article we will discover the VIC MOS, processor responsible for managing the image of Commodore VIC-20 8-bit computers. VIC stands for Video Interface Chip, is such an important chip for VIC-20 that it also gave it its name but, on the other hand, its origins are not common to it: in fact, VIC is a chip born before the computer itself.

Origins

Let's go back to 1976. MOS Technology is a small integrated manufacturer founded in the late 1960s by Allen-Bradley as a secondary supplier of Texas Instruments chips. It came to know a certain fame in 1975 when, thanks to the work of several former designers of the Motorola 6800, including Chuck Peddle, who left the company and were hired en bloc by MOS Technology, it marketed the 6502, an economical CPU that immediately enjoyed enormous commercial success. Motorola, however, denounces MOS because the development of 6502 was in his opinion carried out with knowledge and technologies “exported” by Motorola from its former employees. Allen-Bradley, given the bad news (a Motorola lawsuit could prove very expensive) and also given the decline in the turnover of the computer chip industry during that period, sells its shares to the company's executives and abandons MOS Technology. At the beginning of 1976 MOS settled and paid a one-time fee to Motorola to close the case and agreed to take the licences of the peripheral chips necessary for the operation of 6502.

Despite the good sales of 6502, finances of MOS are in trouble at that time. At the end of 1976, therefore, MOS accepts the purchase proposal made by Commodore, who at that time is fighting a price war with the other manufacturers of electronic calculators and seeks a chip manufacturer to become independent of its supplier/commercial rival Texas Instruments.

Commodore, however, tries to expand its market by diversifying its offer and, in this perspective, is convinced by Peddle that calculators have now made their time and that home computers will be the future. Peddle proposes a system based on its 6502 development kit called KIM-1 that MOS offered to 6502 customers: the project is started and the final result is the PET, which Commodore launched in 1977. This computer is similar to a terminal, not offering the possibility of managing raster graphics but only having a set of semi-graphic characters with which you can draw program interfaces.

Research continues in MOS Technology. Al Charpentier has in the meantime created a graphics chip called Video Interface Chip, labelled 6560, intended to equip cheap CRT systems such as computer terminals, biomedical systems, or home gaming consoles, another sector of consumer electronics that is taking over at that time. The chip, however, is unsuccessful: MOS fails to place it with any of its customers. At the same time trying to improve the PET, Chuck Peddle and Bill Seiler designed a new system called TOI (The Other Intellect), an office computer with a 6564 graphics chip capable of displaying 80 columns.

TOI remains at the prototype level because 6564 requires fast but very expensive (at the time) static RAM (SRAM) due to its limited access times to video memory; in addition, the 80 columns require a special screen to be displayed. Both of these factors would greatly increase its selling price and it was decided not to continue its development. At the same time, a new PET model capable of generating

color images is also being developed thanks to another graphics chip 6562 which, however, has the same economic problems because it also needs SRAM memories. Both projects are set aside.

Towards the end of the 70s, a young engineer, Robert Yannes, was hired to make a cheap computer prototype called MicroPET at home. Together with Al Charpentier and Charles Winterble, Yannes presents this prototype to Jack Tramiel, head of Commodore, who immediately authorizes its development. The Vixen project, which will lead to VIC-20, takes shape. For graphics and sound it was decided to use the best of what had been developed until then: the original 6560 was taken to which were added the most efficient sound generator of 6562 and color management of 6564. The new VIC was born.

Technical specifications

VIC is a chip used for both image and sound generation. For its programming it uses 16 registers mapped to memory, then seen by CPU and computer as normal memory cells, at the addresses $9000-$900F.

VIC has 14 addressing lines and can therefore access up to a maximum of 16 KB of video memory, which it divides into 3 main areas: video memory, color memory and character memory. The latter, in the case of VIC-20, is on ROM because the character map is predefined and is contained in 4 KB where four distinct maps are saved: 2 normal maps, one with uppercase and semi-graphic characters and one with mixed uppercase/lowercase and semi-graphic characters, as well as the inverted versions (i.e. in "reverse") of both.

At the screen level, VIC manages an image whose dimensions can be set through its registers up to a maximum of 248x232 pixels in NTSC systems and 256x280 pixels in PAL systems, even if VIC-20 presets the image to 176x184 pixels, corresponding to 22 characters in width and 23 characters in height, respectively, for a total of 506 cells of 8x8 pixels.

This reduced horizontal resolution, due to the ratio of the video signal of 4:3, translates into pixels with a slightly rectangular appearance, being larger in width than in height (as shown in Figure 1). The chip does not allow bitmap graphics: this means that it cannot address the individual pixels of the image but only use the characters as graphics elements.

To create games with elaborate graphics, however, VIC

Fig. 1 - Images generated by the VIC appear with rectangular pixels

allows you to use custom maps, thus giving the programmer the possibility to create their own graphic elements.

The video memory corresponds to as many bytes as those set for the current image: in the case of VIC-20, which reserves 512 bytes for video memory, 506 bytes are used due to the screen resolution of 22x23 characters. Each byte contains the character code to display, whose array of pixels is retrieved from the character memory, going to "draw" in the currently active map. Such a map may be either in ROM (for the default characters) or on RAM (if you are using a redefined map).

The color of the individual pixels is contained in the color memory and is stored in nibble form (4 bits), where each nibble indicates the color code to be used. Colors are in all 16 but the chip has some limitations. The main colors of the fonts and border of the image can only be 8, chosen from the following: black, white, red, cyan, purple, green, blue and yellow. For the background there are also 8 other so-called auxiliary colours to choose from: orange, light orange, pink, light cyan, light purple, light green, light blue and light yellow.

VIC can operate in two modes, with "high resolution" or "multicolored" characters. The first is the standard mode: the characters are 8x8 pixels wide and each bit in memory corresponds to 1 pixel on the screen. The second handles characters with a double width of 16x8 bits where 2 bits in memory correspond to 1 pixel on the screen and is activated by setting the 4th bit of the color nibble. Up to 4 different colors can be used in this mode, and each color is selected from the bit pair value.

This pair does not directly indicate the color but from where to retrieve it between: the primary color, the border color, the background color, the auxiliary color. Note the chip has 4 pins intended exclusively for direct connection

with the color memory for direct recovery of the color nibble. Since multicolor graphics appear at half resolution due to the fact that it takes 2 bits to represent a pixel (see figure 2) this mode has not been widely used in games.

Map of the memory

Although the video chip sees the memory dedicated to it as a single contiguous block of 16 KB, in reality it is addressed in a rather particular way in the computer. Character map memory, which occupies 4 KB, is located in the $8000 to $8FFF area. Since the characters that can be managed by VIC are 128 and each character is 8x8 bit wide, it takes 4 areas of 1024 bytes each to contain map data integrated into VIC-20. As mentioned above, from $9000 to $900F the 16 registers that manage VIC in all its aspects are mapped: color settings, vertical and horizontal resolution, sound generation, etc...

Video memory is mobile. In a VIC-20 with no RAM expansion installed, it occupies cells from $1E00 to $1FFF (512 bytes). If there is an expansion then the video memory is moved to block $1000-$11FF: this is because the memory dedicated to programs must always be contained in a contiguous block of RAM.

The color memory is also placed in an address space that varies depending on whether the computer mounts an expansion or not: in the first case it is placed in block $9400- $95FF, while in the second case in block $9600-$97FF.

Strengths and weaknesses

As mentioned at the beginning, VIC does not only manage the video image. To contain the final price of the computer, its designers have included several features within it. The chip is also responsible for generating sound: it is equipped with 3 square wave audio channels and a fourth channel for white noise. Volume control is, however, only possible globally.

VIC embeds two digital/analog converters that allow it to read the position of the X and Y axes of a paddle. It can also handle an optical pen. Direct memory access (DMA) is also available to read and write to RAM independently.

In addition to bitmap graphics, the chip also lacks DRAM (Dynamic RAM) refresh circuitry, as it is designed to work with SRAMs. DRAMs are a type of memory widely used between the late 1970s and the early 1980s because they are cheaper than faster but more expensive SRAMs: the latter, however, unlike DRAMs, do not need continuous

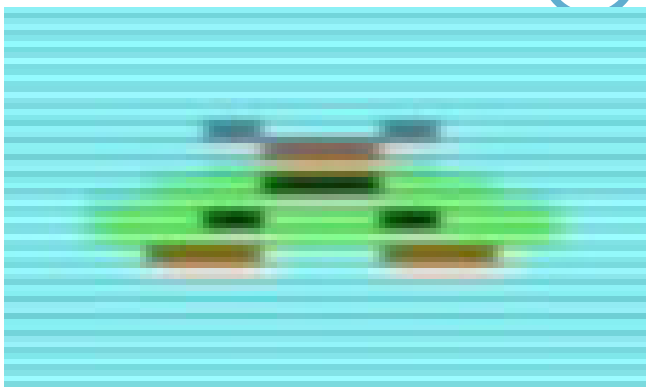


Fig. 2 - Example of multicolor graphics where two characters are placed side by side to create an object. Note the "block" pixels with horizontal resolution halved.

access at regular intervals (the so-called "refresh") in order not to lose the stored data.

VIC does not support sprites, present on other contemporary graphics chips such as Atari TIA (Atari 2600) and the TMS9918A (the latter was the subject of our previous paper), nor an interrupt raster. However, it has a log that contains the row currently drawn by the video brush.

Inheritance

Despite its limitations, VIC-20 was very successful, becoming the first computer to be sold in more than 1,000,000 units and surpassing the target of 2.5 million copies marketed.

And part of the credit goes mainly to its VIC chip, thanks to which the computer offered at a more than affordable price graphic and sound capabilities of some importance.

Work done on VIC was then put to good use. From it came the two chips responsible for the success of the best-selling computer ever, the C64: VIC-II and SID are in fact “descendants” of that first, winning, project.

Happy New Year to everyone and to the next article.

Flash News!

BASIC ON THE ZX SPECTRUM

(Alberto Apostolo)

Anja de Weerd, reader of RMW ENG, added a comment (Fig.1) to the article on the conversion of programs from ZX81 to Spectrum, published in RMW #26-IT / RMW #04-EN, concerning a small game for ZX Spectrum (RMW #26-IT Page 7 Fig.7, RMW #04-EN Page 9 Fig.7).
Thanking for the compliments, I accepted the directions and I reported them in the program in Fig.3. At line 75, the number of "scrolls" performed is equal to the number of quotes (SymbolShift-7) minus 1.

Notice: those who want to know more, can subscribe to the Facebook group BASIC ON THE ZX SPECTRUM (Fig. 2) where the comment appeared.

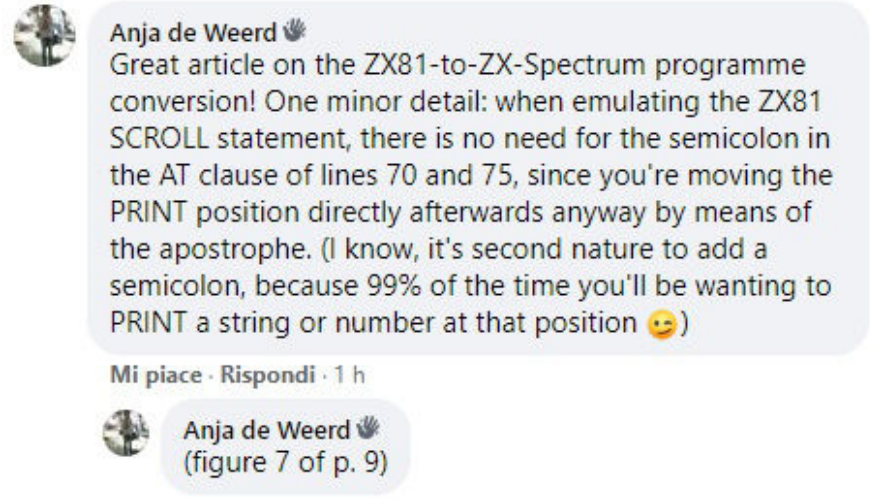


Fig. 1

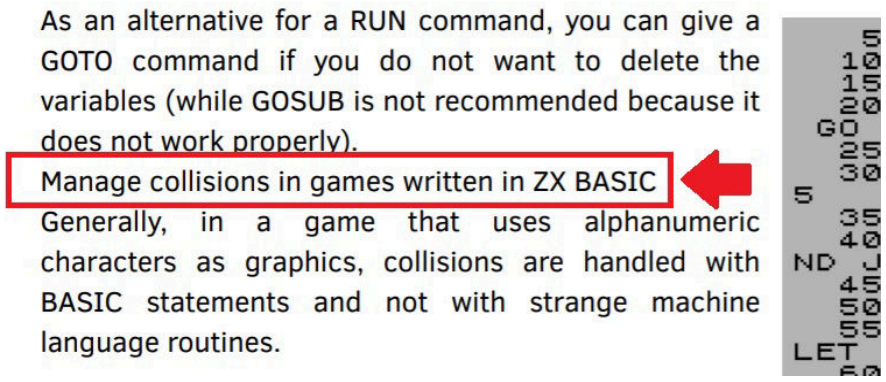
Fig. 2

```
  10 LET S=0: LET I=15
  15 PRINT AT 0,I;
  20 IF SCREEN$ (0,1+I)="W" THEN
 GO TO 90
  25 PRINT "■■■■"
  30 IF INKEY$<>"x" THEN GO TO 5
5
  35 FOR J=-8 TO 8
  40 PRINT AT 8-ABS J,I+1;("T" A
ND J<0)+(" " AND J>=0)
  45 NEXT J
  50 LET S=S-10
  55 LET I=I-(INKEY$="n" AND I>0
)+(INKEY$="m" AND I<29)
  60 PAUSE 5
  65 PRINT AT 14,RND*31;"W"
  70 POKE 23692,255
  75 PRINT AT 21,31;;;
  80 LET S=S+1: GO TO 15
  90 PRINT S
```

Fig. 3

PRINT HORRORS

In RMW #04-EN (page 9), released after Halloween, we missed a "horror" of printing.
The title of a paragraph had to be highlighted in bold (Fig.4).
We apologize with the readers.


As an alternative for a RUN command, you can give a GOTO command if you do not want to delete the variables (while GOSUB is not recommended because it does not work properly).

Manage collisions in games written in ZX BASIC

Generally, in a game that uses alphanumeric characters as graphics, collisions are handled with BASIC statements and not with strange machine language routines.

Fig. 4

A "TRICK" FOR ONE-LINERS ON ZX SPECTRUM

(Alberto Apostolo)

If you type a program line too long, with lots of instructions separated by ":", the ZX Spectrum "rebels" slowing down the cursor even more.

Then you should start from the bottom, entering the instructions in reverse (the cursor, always at the beginning of the line, will not be slowed down). To avoid confusion, first prepare a list of instructions to type on your beloved ZX.

A Snake's clone for the C64 on cartridge

by Giovan Battista "giomba" Rolandi

Introduction

Like many of RMW readers, I also love collecting what are today called "retrocomputers". The main reason I do this is to experience the pure contact between hardware and software: I like to go into the meanders of registers and memory locations, and then see the machine obeying every single bit of code I'm sending to it.

This is how, a few years ago, during a relatively quiet period, I started programming a small game for Commodore 64, a clone of the old and famous Snake which has become popular again with the versions seen on the first mobile phones with matrix screen. With very poor imagination, I decided to call it "Snake6502", honouring the famous processor that sits at the heart of millions of retrocomputers.

After trying it for a long time on VICE emulator, then on the physical machine using tape and .prg format, the natural next step was to bring it on cartridge format. And to do so, I had to face new challenges that I decided to share with the readers of this magazine.

Space

The Commodore 64 is designed to use "standard" 8KiB cartridges, although it can be used up to 16KiB and, with the addition of dedicated hardware, even beyond. My habit has always been trying to save memory spac as much as I can. In this project certainly I could do more, but with 4KB of SID, 2KB of custom characters, mazes and code, it is still early to fill in the 8KB available for the basic configuration.

The memory of each program can be divided into sections and conceptually three of them are recognized: text, data and bss. The text section contains the actual code that defines the program logic, the operations to be performed, the decisions to be made, the jumps to the subroutines (and the subroutines themselves) and so on; the data section contains data that has already been initialized, and possibly static, such as character maps, mazes, or lookup tables (i.e. a table containing pre-calculated values, to streamline runtime calculations); the bss section finally contains uninitialized data, and this is an area that is populated during program execution.

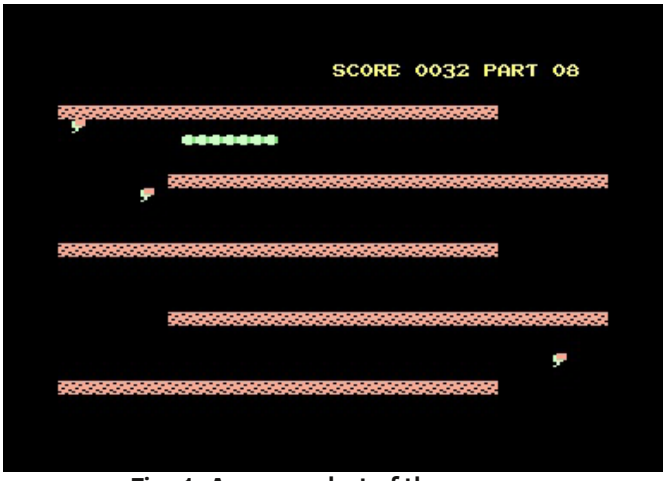


Fig. 1: A screenshot of the ganme

Knowing these basics, essential for those who try to program in assembly, you can immediately understand that certain sections of our program, such as the text and the data, which contain the program itself and the labyrinths, must necessarily be contained in our ROM, while the bss, which instead will contain, at runtime, the information on the status of the game, has no reason to occupy space in our limited ROM. Among other things, the content of ROM certainly cannot be changed during the execution of the game, unlike the .prg which is instead copied to RAM when loaded, so wasting bss space to put it on ROM makes no sense.

So, for example, the variables for the score, for locating the head and tail of the snake, and so on, will occupy no space (if put in the right section) other than a small reference that the assembler will keep in mind for us on the development machine.

The following table shows the memory map of the game before starting the conversion: making careful use of the sections, you can see, for example, how no reference in the zeropage is allocated in the .prg, and how all the essential bytes are concentrated in a small memory area around the address $1000.

Knowing your assembler is essential to organise the code

Address	PRG	Description
$0000 - $0001	no	hardware
$0002 - $00FF	no	zero page pointers
$0100 - $07FF	no	*free ram*
$0800 - $0FFF	yes	initialized data segment + BASIC autostart
$1000 - $1FFF	yes	SID tune
$2000 - $27FF	yes	custom charset
$2800 - $xxxx	yes	program logic
$xxxx - $CCFF	no	*free ram*
$CD00 - $CDFF	no	data segment (not-initialized vars)
$CE00 - $CEFF	no	list X
$CF00 - $CFFF	no	list Y
$D000 - $DFFF	no	I/O
$E000 - $FFFF	no	Kernal

Table 1: Memory Map

as intelligently as possible. For example, with dasm you can mark segments initialized with SEG and those not initialized with SEG.U, maintaining the convenience of references, as seen in the excerpt:

```
SEG.U zeropageSegment
org $02

; Generic src/dst copy pointers
srcPointer DS 2
dstPointer DS 2

; Music
SEG SIDSEGMENT
org $1000
INCBIN "music.sid"

; Program
SEG TEXT
lda (srcPointer), and
sta (dstPointer), and

; Lists
SEG.U listSegment
org $ce00
listX DS 256
listY DS 256
```

As you can see from the memory map, using this precaution allowed you to keep the program in a small memory region. Small, but not too much: at this point in fact the program amounts 8252 bytes, which is still 60 bytes too much to be able to enter the 8192 byte ROM, not to mention that you still have to add other bytes of program to manage the startup from cartridge.

60 bytes too much, however, are tempting, because yes, the memory map is compact, but only apparently. In fact, there are several memory locations that were chosen because they were forced, and that left a few bytes of empty space here and there.

For example, the BASIC autostart, which is used to start the game as soon as it is loaded, without the user having to hit RUN or, worse, some weird SYS, must necessarily be at the memory location $801, in order to be played automatically, while the music for SID has been written so that it can be played only if it is at the location $1000. Similarly, between SID and Character Map, which must be at $2000, a handful of bytes left.

How to quantify how much “gruyere” space is actually empty? Again, the assembler is our best friend to solve this type of doubt, automatically.

```
LASTINIT SET .
;
; ... code ...
;
ECHO "file.asm @ ", LASTINIT,"len:",(. -
LASTINIT)
```

With dasm you can divide the program into several files and insert, at the beginning and end of each of it, two lines like those shown, so that, during compilation, the space used by each piece of code is shown.
It turns out that, a hole here and a hole there, this leaves a hundred bytes of empty space, but difficult to fill: too little and too fragmented for the character map, enough for the code (at least at the beginning) but limiting in the long term, and certainly predisposed to make more spaghetti-code than necessary.

Although free space is apparently sufficient, it is in fact almost completely unusable due to fragmentation. But despite this, I then add the code for starting from cartridge (see below), and by running a little "knapsack algorithm", the code is broken and packed in every nook and cranny of memory, and the size of the program is increased to just 8190 bytes, that is, with a margin of just 2 bytes of free memory. It's a (partial) success!

Starting from cartridge

When the Commodore64 starts, it attempts to determine, through a combination of hardware and software, whether there is a cartridge inserted into the dedicated expansion port. From a software point of view, the Kernel checks if, at address $8004, the string “CBM80” is present in PETSCII, and, if it finds it, jumps to the address indicated in locations $8000-$8001 (i.e. the first word of the cartridge).

When the Kernal starts the cartridge, it fails to call the I/O initialization routines, which are essential to be able to use video, keyboard, joystick, mass memories, and so on. As you can see from the code excerpt, therefore, it is advisable to call them manually.

Before starting the actual program, then, it must be copied to the original memory locations, because some parts cannot reside elsewhere (e.g. music for the SID).

```
SEG CARTRIDGESEGMENT
```

```
org $8000

sUBROUTINE CARTRIDGE
WORD .COLDSTART
WORD .WARMSTART

; CBM80 in PETSCII
; (autostart signature)
BYTES c3, c2, cd, 38, 30

.coldstart:
six
stx $d016
jsr $fda3
jsr $fd50
jsr $fd15
jsr $ff5b
cli

.warmstart:
; copy to original location
jsr copy
; jump to program entry
jmp start
```

The cartridge is therefore ready and working, but at this point it is spontaneous to ask: but all that spaghetti code from before, is it a good thing? And all this work, did it do any good? The game just enters 8KiB: what if tomorrow is to be expanded?

RLE Compressor

One of the space-saving techniques is data compression, which exploits the inherent redundancies of data. Consider, for example, the labyrinth that the serpent must travel through: it is made by means of numerous tiles close to each other, all equal and forming repetitive patterns, such as vertical and horizontal lines. One of the compression algorithms that can be used to compress this type of data is the so-called Run Length Encoding (RLE), and the underlying concept is rather trivial: why store so many X tiles all the same, one after the other, when you could simply store more compact information like “there are N X tiles in a row”? A sequence like “AAABBB” would then become “3A3B”. With this simple and trivial observation, the labyrinth of an entire level, which in memory would occupy 40x24 = 960 bytes, can be reduced to a few dozen bytes. For example, in snake6502, the "Training” labyrinth, after being compressed with RLE, takes just 69 bytes - a 93% memory saving!
However, this type of compression had already been applied at the Snake6502 level well before thinking about making the cartridge, so the 8190 bytes are already net of compression. Basically, I'm always at the starting point.

One problem with RLE compression is that it is too simple and can only exploit gross, obvious redundancies. As we have seen, it therefore works very well for maps and mazes, but it is tremendously ineffective against code or character maps, which have finer redundancies. Using RLE on code could actually be highly counterproductive, increasing the final size of the program rather than decreasing it.

How can this happen? When you compress data, you do it to save memory, but at the expense of adding new data structures and a decompression subroutine, which take up space for things that were not there before: the hope is that the beneficial effects of compression are large enough to compensate for the loss due to the addition of this overhead.

With RLE compression, this compromise would not bring benefits: a smarter algorithm is therefore needed.

LZ Compressor

The LZ algorithm, named after Lempel and Ziv who designed it, recognizes not only when a byte repeats, but also when longer sequences repeat. For example, if RLE cannot compress the sequence “ABABAB” in any way, LZ recognizes that "AB” is a substring that repeats 3 times, and can therefore conceptually store something like “3AB”. In practice, the algorithm is more complex and there are different versions.

The basic idea of LZ is to see the data to be compressed as a stream of bytes, and as you scroll through it, to reuse the strings you have already encountered by pointing them back into the stream.

The following image shows a fairly trivial example in which we see how the previous 6 bytes were compressed into 4 using a special structure (offset;length), where the offset specifies the position of the original substring.
Clearly this is just one example that needs to be deepened more rigorously, but before doing so it is good to dwell on some details.

If the structure (offset;length) has a fixed size, how many bits can be dedicated to offset and how many to length? Dedicating many bits to offset would allow retrieving very distant bit strings spatially, but at the cost of limiting the length of these, and vice versa, dedicating many bits to

A	B	A	B	A	B
A	B	(-2;2)	(-4;2)		

Fig. 2: A sequence of bytes before and after compression with LZ

length would allow retrieving very long strings, but only close. The number of bits dedicated to offset and length can be chosen in such a way as to have the best compression compromise, which of course is different depending on the data you want to compress, and often also clashes with the technical limitations of the hardware: therefore, in essence, it depends on the specific implementation.

The liblzg library implements a fairly simple version of the LZ algorithm, such that it can be easily carried on modern embedded systems, but also on somewhat retro computers, such as the Commodore64. In fact, there is a “mini” version for the Motorola 68000 processor and, of course, for the MOS 6502. The library comes with a compression utility, written in C language, which can be run on any modern operating system, and which is particularly convenient for cross-development nowadays.

“liblzg-mini”, in order to allow the best possible compromise during compression, rather than employing a single structure (offset;length), employs 4, and can use the most appropriate one, depending on how far the redundant string of bytes is or how long it is. Figure 3, in which each square represents a bit, shows the four structures: near copy, short copy, medium copy and distant copy, which are chosen as needed.

Note how, to save as many precious bits as possible, they do not actually represent directly offset and length as binary numbers, but a variant thereof. For example, since offsets 0 to 23=8 can be easily represented by the "near copy” structure, the "short copy” structure, while having 6 offset bits, does not represent values from 0 to 26=64, but those from 8 to 64+8=72. Similarly, the 5 bits of length, which could represent only values from 0 to 25=32, are used instead to index a lookup table, which contains a selection of values of common lengths, from 2 to 128. All details can be found directly in the source code comments.

A final problem that has not been addressed is that of distinguishing the various data structures. To distinguish them during decompression, a prefix is applied to them, called in jargon marker: of course there are 4 types, as many as there are structures.

All bytes that are not markers or structures are treated as original, uncompressed bytes that are part of the stream as they are, and are called literals.
A sample stream of compressed bytes then appears as such (see Figure 4).

One last problem remains to be solved: what happens if a marker is equal to a literal? What if the marker was $77 and you wanted to literally represent a $77 byte instead? In this case, liblzg solves the problem by treating that byte as a marker, but placing the first byte of the corresponding structure at the special value $00, which would still be unusable for other purposes (offset = 0, length = 0). Using the marker plus a piece of structure, it is also possible to represent those literals equal to the markers. In these rare cases, however, a figure that would originally have been 1 byte large, compressed becomes... 2 bytes! This is a compromise that we must accept, given the technical constraints.

Fig. 3: The 4 data structures used by liblzgmini

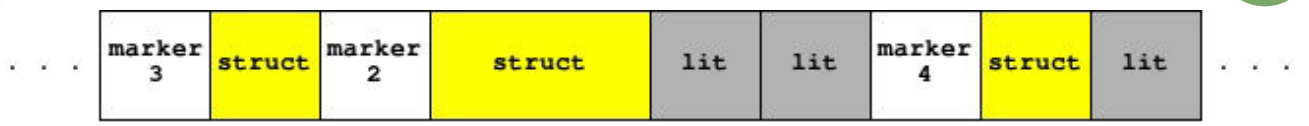


Fig. 4: A sample example of compressed byte stream

Using the LZ compressor for the snake6502, the final size of the program therefore goes from 8190 to just 5614 bytes, that is reduced to 69% of the original! However, the decompression routine, not exactly trivial, must still be placed on the cartridge, uncompressed, and takes about 400 bytes. Despite this, at the end of the day, the cartridge amounts 6029 bytes, so there are still more than 2KiB free for any future expansion - a great result! Here we go. Here we go.

Packaging

The game is then first assembled as usual, possibly even leaving a few empty spaces here and there for reasons of maintainability and readability – both LZ will take care of minimizing waste, then it is compressed with lzgmini in the development environment, becoming a nice compact snake.pak.lz file. This file is then chained to the decompression routine, and placed on the cartridge which, as soon as it is started, will unpack it in memory in the original location, and will finally allow us to play.

To make the decompression operation more scenic, you can add a simple $d020 statement to the routine to run each cycle, to get a few seconds of “loading bars”.

Hardware Cart

Prepared the binary file with all this cinema inside, and extensively tested on the emulator, it is finally time to physically transfer it to a cartridge.

The circuit to be created is quite simple, but if – as in the case of the undersigned – you are not a professional in the sector, you should rely on some ready-made solution, for example Versa64Cart.

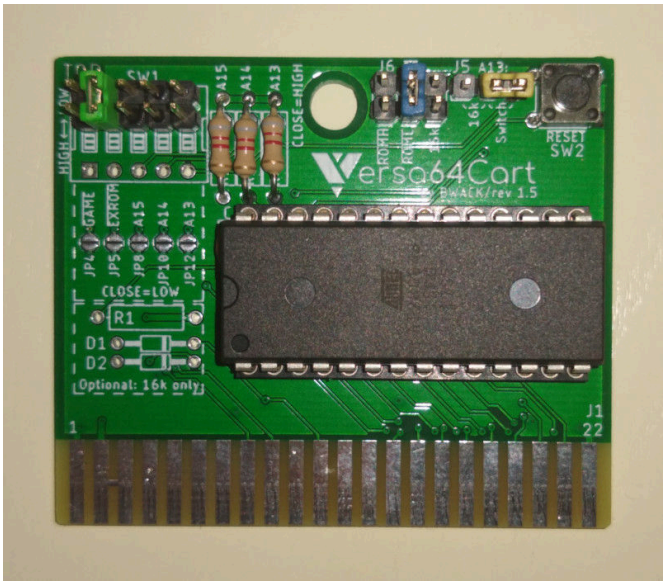


Fig. 5: The Versa64Cart with snake6502

Versa64Cart is a versatile card that allows you to make 8 and 16KiB cartridges for the Commodore64, both in standard and ultimax mode. For about 10 Euro you can have some copies printed at any printed circuit factory, and for another about 10 Euro you can buy (in abundance) all the components you need to weld it at home.

Technically all you need is:
- a circuit board;
- an EPROM, or better still a compatible EEPROM 27C64 (8KiB), 27C128 (16KiB), 27C256 (32KiB);
- a 100nF capacitor;

In fact, in order to enjoy the versatility of configuration, it is also advisable to obtain:
- a “wide” 28-pin DIP socket, practically essential if you do not want to weld the ROM permanently;
- button, convenient for resetting;
- 5-way DIP switch;
- Resistors (8-10 kΩ), pin headers, and assorted jumpers.

When the Commodore64 is turned on, this determines the type of cartridge attached via two hardware signals, EXROM and GAME, active low, i.e. when they are brought to 0V. In order for the cartridge to actually respond, when contacted by the Commodore, it is necessary to connect OE (Output Enable) pin of the ROM to the right ROM enable signal, which must be chosen between ROML for cartridges mounted at $8000 and ROMH for those mounted at $E000. To mount the snake6502 cartridge, which is 8KiB, standard mode, address $8000, only the EXROM signal (the second switch of the DIP switch) must be activated and the ROM OE connected to the ROML signal (jumper J6).

(In the attached image a jumper is used as an alternative to the DIP switch: all possible configurations are listed in the detailed manual that can be downloaded online, together with the circuit diagrams)

EEPROM Programmer

The last problem to deal with before you can actually play is writing the EPROM.

Although it is not exactly in line with the spirit of the time,

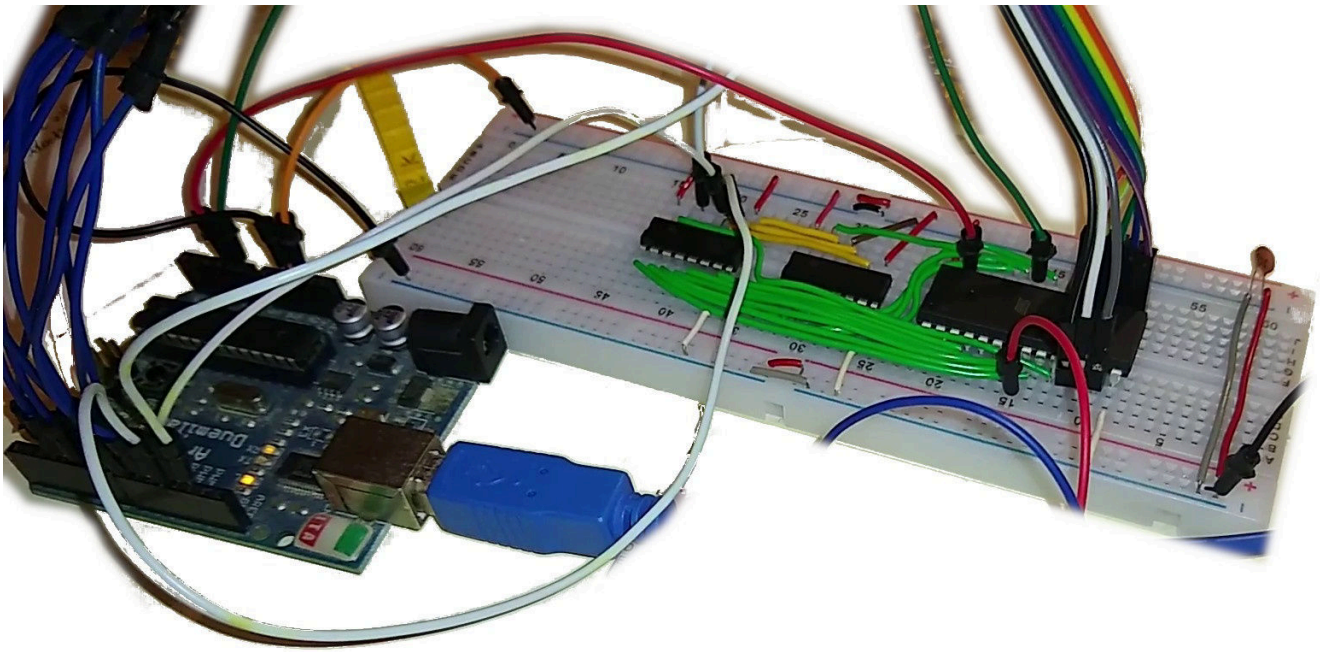


Fig. 6: The homemade EEPROM programmer built with Arduino

for simplicity it is advisable to use a "modern" EEPROM (Electrically Erasable Programmable ROM), so that it can be deleted and reused comfortably in case of errors, and also to try any new versions.

The EEPROM 28Cxx (i.e. 27Cxx compatible) can be purchased for a few Euros, while EEPROM programmers are not very expensive, but, amateurly, it may also be convenient to use any cheaper microcontroller, such as the famous Arduino, especially if you already have it, such as the undersigned. Ben Eater is famous in the amateur environment for having created a didactic and adaptable version, so it is best to take inspiration from his work.

A common Arduino Duemilanove/Uno has 20 feet, but to program the ROM they need more:
13 pins for addresses (213 = 8KiB)
8 pins for data (8 bits = 1 byte)
2 control pins
2 pins for serial communication

The most numerous address pins are fortunately only needed in output, so they can easily be replaced by a pair of shift registers, i.e. digital circuits which, after passing them a string of bits in series, show it in parallel on their 8 pins. The 595 series shift registers can be cascaded, allowing an arbitrarily large number of output feet to be obtained.

The firmware for Arduino, expanded, is therefore responsible for managing the writing of the EEPROM, selecting the addresses for writing through the shift registers, reading the contents of the cartridge that is passed to it from the development computer through the serial port.

Since the feet for the data are directly connected to Arduino, then, in the end, it is possible to reread the ROM to verify the integrity of what is written.

The EEPROM programmer pseudocode follows:

```
/* Writing */
```

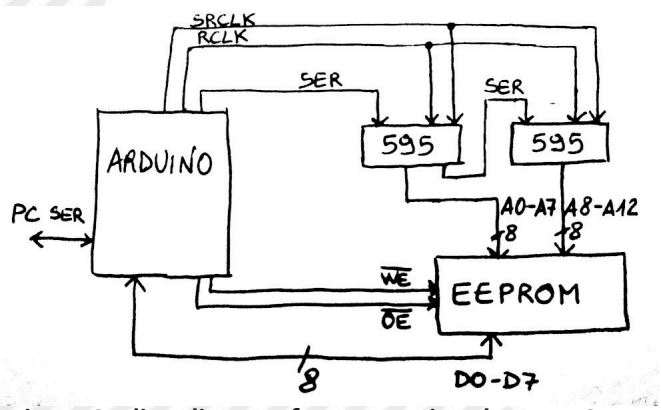


Fig. 7: Outline diagram for connecting the EEPROM to the Arduino

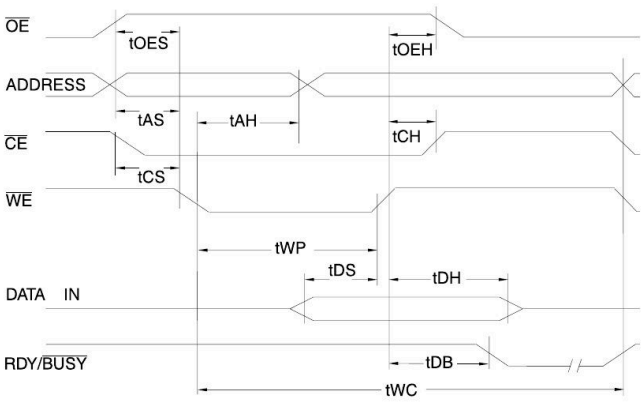


Fig. 8: Timing diagram for writing

Fig. 9: Snake6502 running on a Commodore 128D from cartridge (in 64 mode)

```
address = 0
for address
read from serial
write address on shift-register (A0-A12)
write data on feet (D0-D7)
write command (WE) (HIGH → LOW → HIGH)
address
next

/* Reading */
address = 0
for address
write address on shift-register (A0-A12)
output command (OE) (HIGH → LOW → HIGH)
feet read (D0-D7)
write data on serial
address
next
```

At this point all you have to do is mount the EEPROM on Versa64Cart, insert it into the Commodore64 and… play! ☺

Actually the picture portrays a Commodore 128D, I was wrong right at the end!

References

- dasm, https://dasm-assembler.github.io/
- EEPROM programmer by Ben Eater, https://github.com/beneater/eeprom-programmer
- liblzg, https://liblzg.bitsnbites.eu/
- snake6502, https://git.giomba.it/giomba/snake6502
- Versa64Cart, https://github.com/bwack/Versa64Cart
- VICE, https://vice-emu.sourceforge.io/

Coding without GOTO on the ZX Spectrum

by Alberto Apostolo

The BASIC language wired into the Sinclair ZX Spectrum ROM does not have the structured programming constructs UNTIL, WHILE, IF-THEN-ELSE.

However, you can still write structured programs, taking advantage of the possibilities offered by BASIC Sinclair.

Those who want to apply the techniques covered in this article to other BASIC versions should consult their reference manuals to verify feasibility.

The iterative structures UNTIL and WHILE

The BASIC Sinclair allows the use of conditional expressions that can return the integer 1 (TRUE) or 0 (FALSE) to be used in normal arithmetic expressions.

Also it is known that BASIC Sinclair allows you to alter the values stored in the control variables of a FOR-NEXT loop within the loop itself.

Other languages do not have this feature (for example, FORTRAN 77 and DO-CONTINUE loops).

Fig. 1 shows the conversion of structures UNTIL and WHILE in FOR-NEXT loops of BASIC Sinclair.

Of course, in case of nested structures, it is mandatory to use control variables with different names in FOR-NEXT loops.

The IF-THEN-ELSE structure

Each respectful "Sinclairist" knows well the arithmetic expression in Fig.2, often used in ZX Spectrum programming to calculate the position of characters along the screen abscissa moved with keys 5 (left) and 8 (right).

It's not the only way to implement IF-THEN-ELSE structure. For example, you can use the "calculated" GOSUB command (Fig.3) or, alternatively, use two consecutive FOR-NEXT loops (Fig.4). Also in Fig.4, note the variable A needed to "synchronize" the two FOR-NEXT loops.

Also here, in case of nested structures, it is mandatory to use control variables with different names in FOR-NEXT loops.

A game written in a single line of BASIC

The game shown in Fig. 5 can be rewritten without GOTO (Fig. 6) and even in a single line of BASIC (Fig. 7) as would a real "One-Liner" (who can write non-trivial programs in a single line of code).

```
UNTIL condition          FOR F = 0 TO 1
   [statements]              statements
                             F = condition
                          NEXT F

WHILE condition          FOR F = 1-(condition) TO 0
   [statements]              statements
                             F = -(condition)
                          NEXT F
```

Fig. 1

```
LET X = X - ( INKEY$="5" AND X > 0 )
          + ( INKEY$="8" AND X < 29)
```

Fig. 2

```
                          ... GOSUB 2000 -
                                    1000 * (cond)
IF cond                    ...
   THEN                     999 STOP
        statement_1_1      1000 statement_1_1
        ...                 ... ...
        statement_1_n       ... statement_1_n
   ELSE                    1999 RETURN
        statement_2_1      2000 statement_2_1
        ...                 ... ...
        statement_2_m       ... statement_2_m
END                        2999 RETURN
```

Fig. 3

```
REM IF
          LET A = cond
REM THEN
          FOR F = 1-A TO 0

             statement_1_1
             ...
             statement_1_n

          NEXT F
REM ELSE
          FOR F = A TO 0

              statement_2_1
              ...
              statement_2_m

          NEXT F
```

Fig. 4

Conclusions

It was realized what seemed impossible at first sight (eliminating the use of GOTO command) due to the possibility of modifying the control variables of FOR-NEXT loops and the use of conditional expressions in arithmetic calculation.

```
  10 LET S=0: LET I=15
  15 PRINT AT 0,I;
  20 IF SCREEN$ (0,1+I)="W" THEN
 GO TO 90
  25 PRINT "▀▀"
  30 IF INKEY$<>"x" THEN GO TO 5
5
  35 FOR J=-8 TO 8
  40 PRINT AT 8-ABS J,I+1;("T" A
ND J<0)+(" " AND J>=0)
  45 NEXT J
  50 LET S=S-10
  55 LET I=I-(INKEY$="n" AND I>0
)+(INKEY$="m" AND I<29)
  60 PAUSE 5
  65 PRINT AT 14,RND*31;"W"
  70 POKE 23692,255
  75 PRINT AT 21,31'''
  80 LET S=S+1: GO TO 15
  90 PRINT S
```

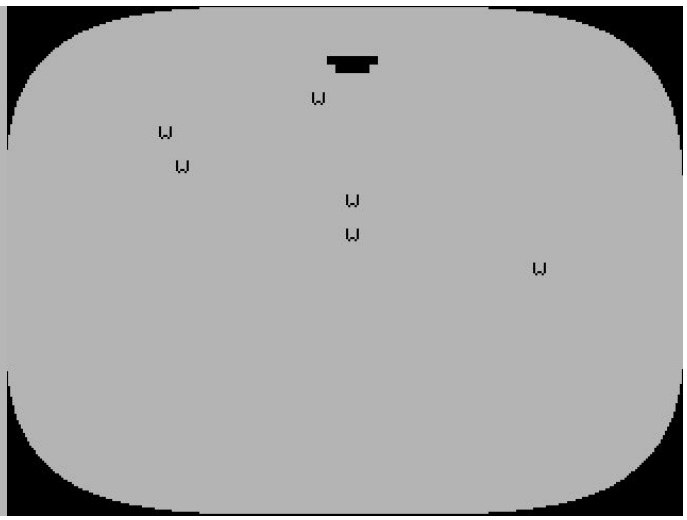


Fig. 5

```
  100 LET S=0: LET I=15
  110 PRINT AT 0,I;
  120 FOR F=0 TO 1
  130 PRINT "▀▀"
  140 FOR G=INKEY$<>"x" TO 0
  150 FOR J=-8 TO 8
  160 PRINT AT 8-ABS J,I+1;("T" A
ND J<0)+(" " AND J>=0)
  170 NEXT J
  180 LET S=S-10
  190 NEXT G
  200 LET I=I-(INKEY$="n" AND I>0
)+(INKEY$="m" AND I<29)
  210 PAUSE 5
  220 PRINT AT 14,RND*31;"W"
  230 POKE 23692,255
  240 PRINT AT 21,31'''
  250 LET S=S+1
  260 PRINT AT 0,I;
  270 LET F=SCREEN$ (0,1+I)="W"
  280 NEXT F
  290 PRINT S
```

Fig.6

```
1000 LET S=0: LET I=15: PRINT AT
 0,I;: FOR F=0 TO 1: PRINT "▀▀"
: FOR G=INKEY$<>"x" TO 0: FOR J=
-8 TO 8: PRINT AT 8-ABS J,I+1;("
T" AND J<0)+(" " AND J>=0): NEXT
 J: LET S=S-10: NEXT G: LET I=I-
(INKEY$="n" AND I>0)+(INKEY$="m"
 AND I<29): PAUSE 5: PRINT AT 14
,RND*31;"W": POKE 23692,255: PRI
NT AT 21,31''': LET S=S+1: PRINT
 AT 0,I;: LET F=SCREEN$ (0,1+I)=
"W": NEXT F: PRINT S
```

Fig.7

Eliminating GOTO makes it easier to write more compact programs, which can even be contained in a single line of code.

Finally, you can do without POKE commands on system variables pointing to the instructions to be executed (NEWPPC 23618/23619 and NSPPC 23620).

Appendix

In 1985, I also participated in a "One-Line" contest organized by the Italian magazine "Sinclair Computer" [SC85].

There were no prizes to be won but only the satisfaction of seeing their own program published (Fig.8)

in una riga

CARATTERI INGRANDITI

di Alberto Apostolo - Foligno PG

```
 100 LET q=2: LET a$="1234567890
": LET a=0: LET o=100: FOR l=1 T
O LEN a$: PRINT AT 0,0;a$(l): FO
R y=175 TO 168 STEP -1: FOR x=0
TO 7: PLOT (a+q*(x+(l-1)*8))*POI
NT (x,y),(o+(y-175)*q)*POINT (x,
y): DRAW q,0: NEXT x: NEXT y: NE
XT l: PRINT AT 0,0;" "
```

Fig.8

Bibliography

[Bon83] R.Bonelli, "Alla scoperta dello ZX Spectrum", Gruppo Editoriale Jackson, 1983.

[SC85] AA.VV., "in una riga", Sinclair Computer n.15, Jul-Aug 1985, Pag.36. https://archive.org/details/Sinclair-Computer-15

A bit of rarity

(rummaging here and there)

Using Sinclair fonts to create custom stickers

by Alberto Apostolo

Searching the Internet for "sinclair computer logo" you will find many beautiful images (official and unofficial) of the Sinclair logo that can be downloaded and printed on stickers to decorate our devices or self-built boards containers. But in addition to pictures, it would be nice to print stickers using Sinclair fonts.

Fonts used on Sinclair computers can be downloaded free of charge in TrueType format from dafont.com (Fig. 1, [DF20a], [DF20b]). Other sites allow you to download similar versions but not for free.

The "Sinclair logo font" that makes up the wording "sinclair" on the casing is located in [FS20] (Fig.2) but to download it you must register. However, it is noted that some letters ("o", "g", "l", "s","z") are confused with the numbers 0,9,1,5,2.

A similar problem is also found in [BA20] (Fig.3) where the letters "v" and "x" could be better treated graphically and the letters "o", "s","z" are confused with 0,5,2 respectively.

Another noteworthy version is the one in the British magazine Sync [SY20], used in no.2-6 of Vol.3 and collected in Fig. 4 (I was forced to add the missing letters "b", "q", "x", "v", "z", drawing them with Paint).

I couldn't find any material about the fonts used in the words "ZX 81", "ZX Spectrum" and the BASIC

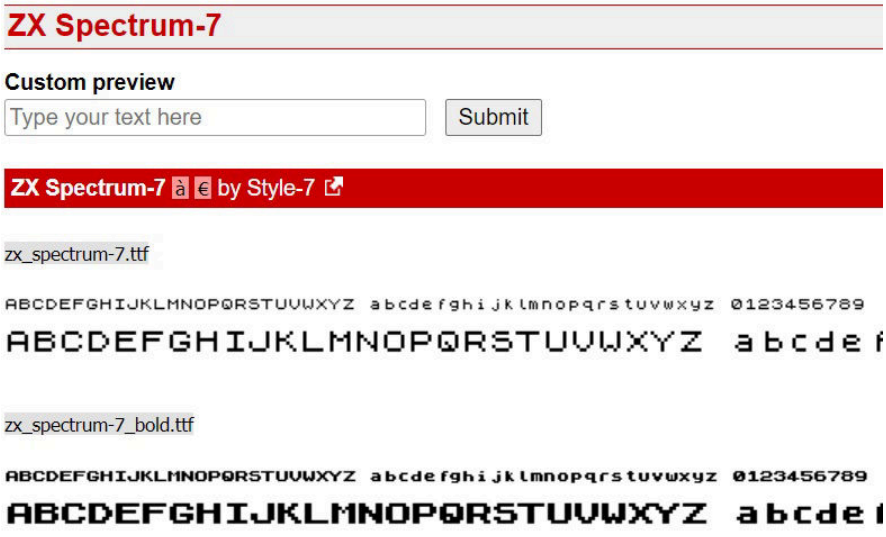


Fig. 1

Fig. 2

Download Bauhaus2015 here.

Oh and here's a font I just made based on the Sinclair logo! Use lower case only for best results. Named after Sinclair Research's founder, 'Uncle' Clive Sinclair. **Sir Clive** to you. Download here and I've also made a bold version called Sir Clive the Bold:

Fig. 3

commands on the ZX keyboards. However, it can be partially solved by using the Arial font in bold despite some letters (e.g. "G") having a very different shape.

In Fig. 5, the comparison between the Arial font (left) and the real ones written on the Sinclair computers (right).

Finally, looking for "Sinclair QL font", I came across the site of the legendary Dilwyn Jones.

In [Jon20] you access an updated section in March 2020, where you can download TrueType fonts suitable for the Windows and MacIntosh environment (Fig.6).

On the same web page there are also links to download versions related to ZX81 and ZX Spectrum.

Fig. 4

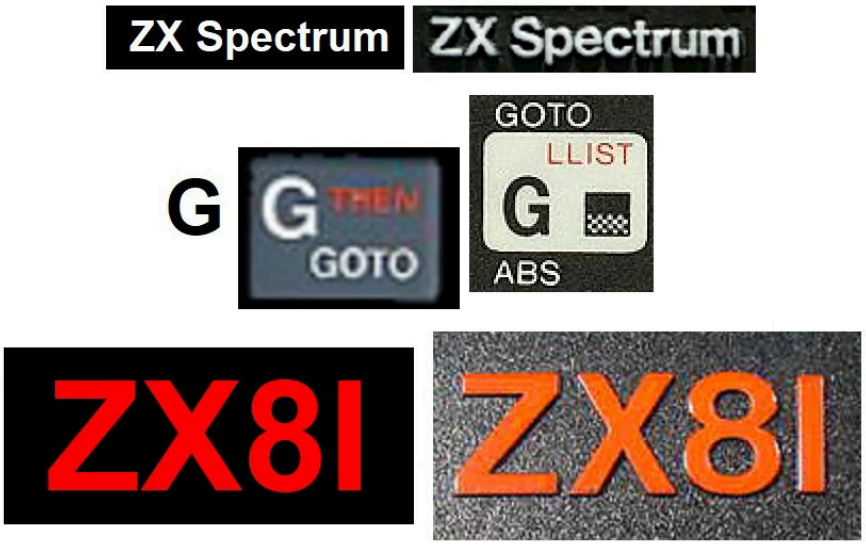


Fig. 5

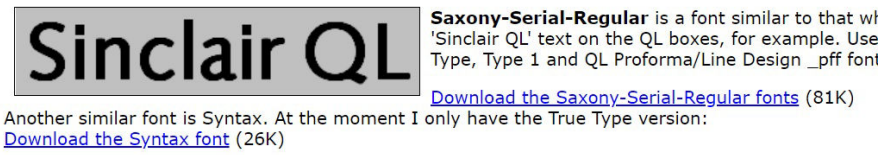


Fig. 6

Bibliography and references

[BA20] (2020-10-24) retrieved from http://www.suppertime.co.uk/blogmywiki/2015/08/bauhaus2015-bitmap-font/

[DF20a] (2020-10-24) retrieved from https://www.dafont.com/zx-spectrum-7.font

[DF20b] (2020-10-24) retrieved from https://www.dafont.com/it/zx81.font

[FS20] (2020-10-24) retrieved from https://fontstruct.com/fontstructions/show/245226/sinclair_logo

[Jon20] (2020-10-24) retrieved from http://www.dilwyn.me.uk/fonts/

[SY20] (2020-10-24) retrieved from https://archive.org/details/syncmagazine

SymbOS - Windows on the Amstrad CPC!

by Francesco Fiorentini

Unbelievable! Simply unbelievable!

I have no other words to describe this software.
By the way, calling it software is extremely reductive; SymbOS is a complete operating system, with utilities, games, browsers and networking tools.
Saying that doesn't make much of an impression nowadays, but try to think that this operating system runs on 8bit machines based on the Z80 processor.
Yeah, SymbOS runs perfectly on Amstrad CPC, MSX, Amstrad PCW and Enterprise 64/128...
Like I said at the beginning: incredible!

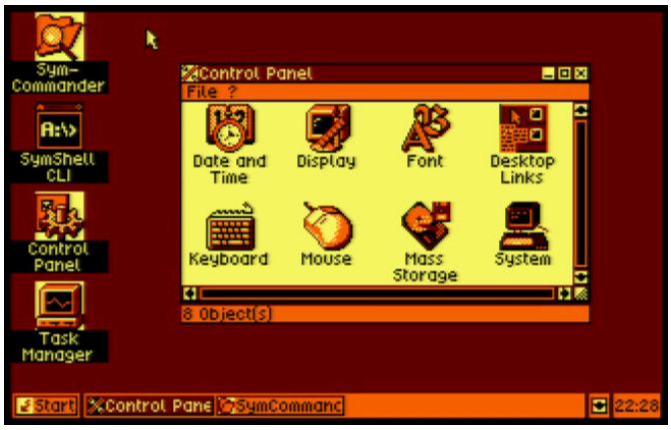


Fig. 1 - It is not Windows, but SymbOS on Amstrad CPC

A bit of history

In the 1980s, GEOS laid a solid foundation for a graphical operating system for Commodore 64. The OS developed by Berkeley Softworks was a real revolution on 8bit machines. Although many elements were only static, it was incredible how an 8bit machine with only 64K of RAM could run a graphical operating system like GEOS.
There were many attempts to reproduce the same result also on Amstrad CPC, but for one reason or another none of these were able to get anywhere near what was developed for Commodore 64.

It was thus that, towards the end of 2000, the author decided to create a real graphical operating system for the CPC. On the other hand, many CPCs had basic 128K of memory (the C64 only 64K), a 320x200 resolution with 4 colors (the C64 instead only 2 colors for each 8x8 area at 320x200 resolution) and many other advantages over the C64. Thus the idea of the SymbOS project was born.

SymbOS means "SYmbiosis Multitasking Based Operating System". SymbOS would have been a real modern operating system for Amstrad CPC, with real preemptive multitasking, dynamic memory management up to 1024K and a look and feel similar to MS Windows. In practice it should have been a demonstration of what could have been achieved on a CPC in the 1980s!
SymbOS development was not continuous, but continued from the end of 2000 until at least 2017, when version 3.0 of SymbOS for CPC & MSX & PCW & EP was released on the author's website. From that moment on you have no information about any other updates, but version 3.0 is perfectly functional and definitely impressive for care, functionality and performance!

SymbOS

SymbOS is available for:
- for the Amstrad CPC 464/664/6128 series,
- for MSX1 (+V9990), MSX2, MSX2+ and MSX TurboR
- for all Amstrad PCW, 8xxx, 9xxx and 10 series models
- for Enterprise 64/128 machines

It requires a minimum of 128K and a mass storage device (floppy disk, IDE, SD...).

Faithful to the passion that lately I have for the Amstrad CPC, I wanted to try just this version.
Not owning the real machine, the test has been run on WinAPE. At the beginning I thought I was doing an injustice to the author, but then I read on his site that also he, to preserve to preserve the real machines, he often uses WinAPE for development and debugging and I changed my mind.

First startup...

The accompanying SymbOS manual is very detailed regarding the installation of the MSX version, but rather lacks instructions compared to the CPC.
In the Amstrad CPC package we find 4 ROMs and 3 diskettes. Although you can run SymbOS directly from a floppy disk, I opted for a mixed installation.
We will then boot the operating system from ROM and then use the floppy disk to run programs that are not in memory.

Launch WInAPE and from **Settings** -> **General** -> **Memory** menu, let's set the options as in Fig. 2.
Select **RAM 64K + 256K RAM Expansion** and **256K Silicon Disk**.

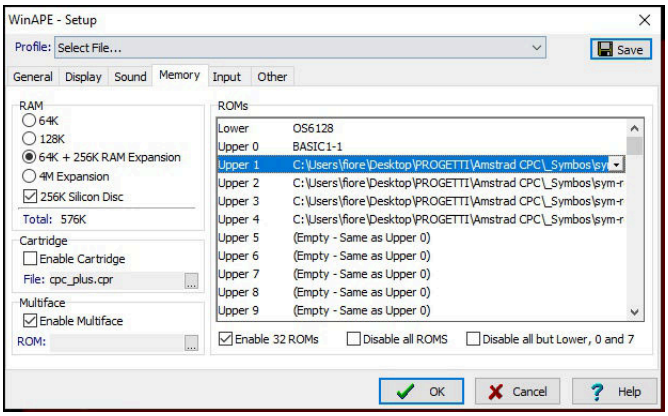


Fig. 2 - RAM and ROM Settings

Notice that I inserted the 4 SymbOS ROMs into Upper1 slots up to Upper4:
- Upper1: sym-romA.rom
- Upper2: sym-romB.rom
- Upper3: sym-romC.rom
- Upper4: sym-romD.rom

In the **Lower** and **Upper0** slots I inserted the ROM **OS6128** and **Basic 1-1** respectively.

Make sure there are no floppies inserted in drives A: and B: and reset the emulator: **Settings** -> **Reset**. If everything went well you should be faced with a screen as in Fig. 3.

Now, as specified on the screen, just type |Sym or |Symbos to start the Operating System.

Fig. 3 - Booting with SymbOS' ROM

That's awesome. A fully Windows like operating system has appeared on your Amstrad CPC, see Fig. 4. The first thing you'll notice is the mouse arrow that, surprisingly, moves smoothly across the desktop. As in Windows, just

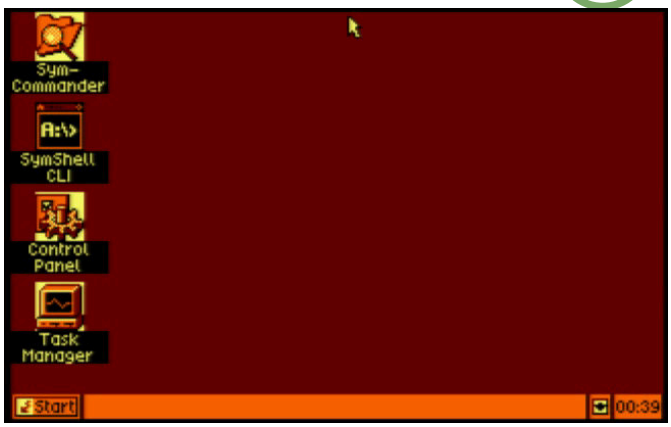


Fig. 4 - The first boot of SymbOS on Amstrad CPC

a double click on the icons to start the indicated program and a right click of the mouse to access the graphic settings.

The second thing you'll notice is the clock at the bottom right. Even this perfectly synchronized with the guest computer clock (Eh yeah, finding the time to write RMW articles between work commitments and an 18-month-old girl is a luxury I can only afford late at night. :-) - NdR).

So let's start the command prompt, which here is called **SymShell CLI**. Double-click on its icon and... An error message, see Fig. 5.

Fig. 5 - SymShell CLI error

What happened? What happened? We simply need to tell SymbOS where the programs we want to run reside.

Insert the **SymbOS-CPC-AppsStandard.dsk** disk (you can also find this inside the zip containing the Roms) into drive A with the command **File** -> **Drive A:** -> **Insert Disc Image**. Then from the **Start** menu choose the **Run...** command and in the next window click on the **Browse...** button. An additional window will open in which you have to choose an executable, in this case the **Cmd.exe** file is more than fine (see Fig. 6 on next page), and confirm the choice using the **Open** and **Ok** buttons.

After a few moments, the time to upload the file from the diskette, you should find yourself in front of a familiar looking window. Looks like exactly the DOS command prompt, see Fig. 7. But the surprises aren't over here. Try

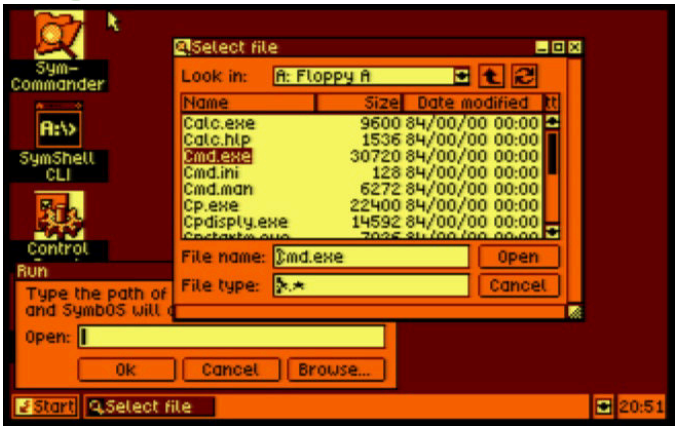


Fig. 6 - Start the executable Cmd.exe

typing **DIR**. Yes, the SymShell CLI is fully compatible with the Windows command prompt. I've tried various commands like **COPY**, **REN**, **DEL**... They are all perfectly implemented. You can even create directories with the **MD** command.

Fig. 7 - The SymbOS commad prompt

The next dir will display the directory correctly. See Fig. 8. Unfortunately, although it is possible to enter a directory using the **CD** command, the directory is only a visual artifact, since it does not work like the real directories of the DOS; in fact, running the dir command inside the directory you will also see the contents of the disk root. Perhaps it would have been asking too much...

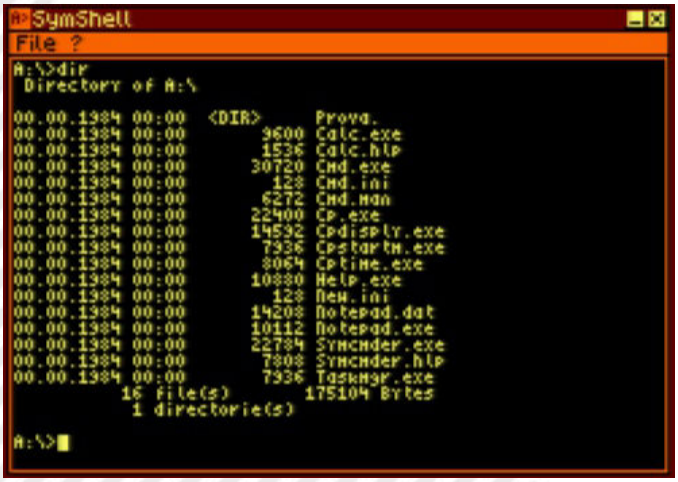


Fig. 8 - The DIR command; notice the 'Prova' directory

The wonders of SymShell CLI have not ended here, however, through the File menu you can configure the appearance of the CLI. And the options are certainly not few, see Fig. 9. We can choose the size of the shell: rows and columns, the color of the text, the background, the border and even decide whether to display the window in full screen.
All the settings you set will be stored in the cmd.ini file so that each time you restart the shell again, you will find it exactly as you configured it. I know I repeat myself, but this is just incredible.

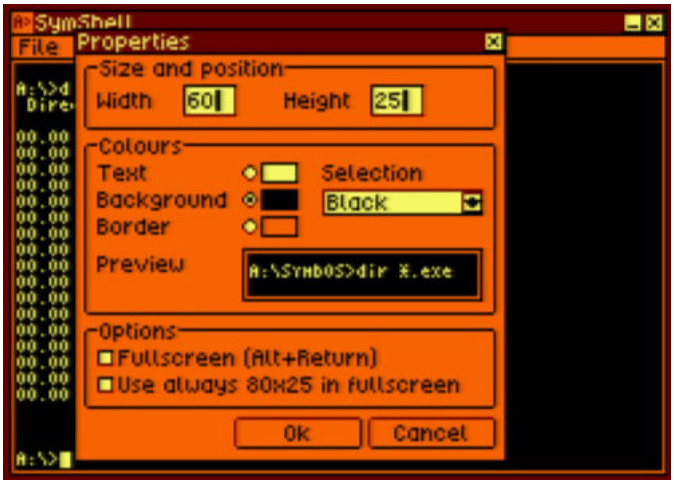


Fig. 9 - SymShell CLI Settings

Obviously, wonders don't stop here... Try double-clicking the Control Panel icon; it looks like it's right in front of Windows 3.1 Control Panel (see Fig. 10). The surprising thing is that all the icons on the control panel are functional to configure SymbOS; try clicking Mouse or Display and you will immediately realize the incredible number of options to configure the system (Fig. 11).

It is interesting to note that the graphic resolution is set by default to 320x200 at 4 colors but that it can also be set to 640x200 at 2 colors. So let's try setting this resolution to see the effect. The space for the windows increases considerably and it is possible to position even four in a

Fig. 10 - The control panel and time zone...

row, see Fig. 12, unfortunately the characters become less legible and the only 2 colors available are a limit, although not insurmountable.

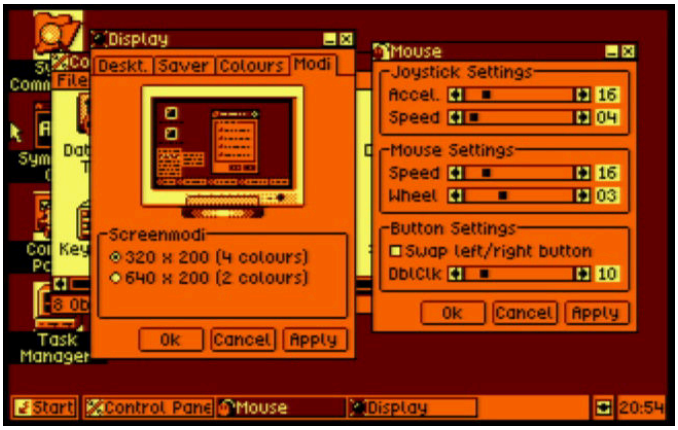


Fig. 11 - Display and Mouse Options

Let's end our first trip within SymbOS by talking about SymCommander, a File Manager clearly inspired by the famous Norton Commander. As you can see from the images in Fig. 13 and 14 many of the most common actions to be performed on files and disks, they are immediately reachable through convenient buttons placed under the graphical representations of the directories.

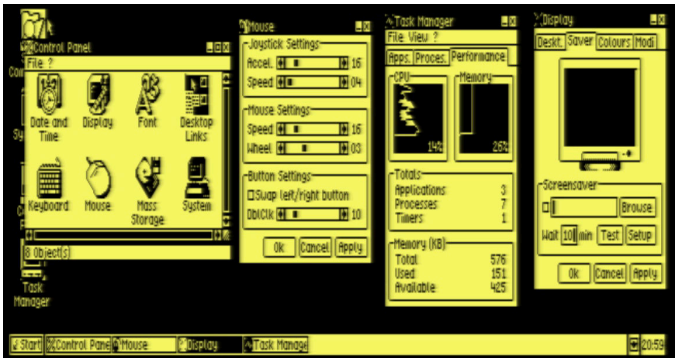


Fig. 12 - SymbOS set with 640x200 resolution

All the other commands, and not a few, can be reached from the convenient dropdwon menu. Of course, SymCommander can also be configured for user use and consumption. The SymCommander is also accompanied by a convenient help file, which can be retrieved via the Help menu.

Note, in Fig. 14, how the SymCommander icon and its Help have two different images in the Task Bar that allow you to immediately identify them.

Conclusions

Well, what more can I say? You would have guessed my opinion by reading the article. SymbOS is an incredible product. Careful down to the smallest details: try double-clicking on the watch in the task bar or pressing the icon

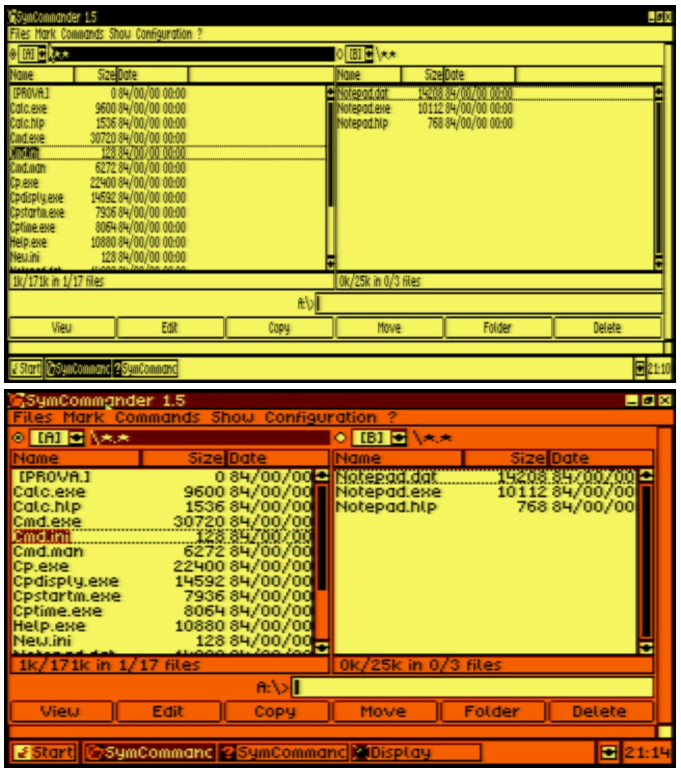


Fig. 13 e 14 - SymCommander at 640x200 and 320x200 resolutions

next to it (spoiler: it is the desktop display function); fast and reliable (during my tests I have never found crashes or strange behaviors, just a glitch in the SymCommander). You can change screen resolution on the fly, no need to restart or upload anything...

Had it been available on the market in the 1980s, Amstrad CPC would certainly have had its killer application for the office market.

Oh, and all this is just the tip of SymbOS' iceberg. On the author's website you will find dozens of other programs, games and even utilities for internet connection, which obviously we will not fail to test in subsequent articles of RMW. My regret is that I do not have a real machine to test it on original HW, but I invite those who have one to send us photos, comments and also a complete review. Obviously the invitation is open to all supported hardware, not just Amstrad CPC.

SymbOS can be downloaded from:
http://www.symbos.de/

Try it, you won't regret it!

When sprite collisions don't collide!

by Attilio Capuozzo Founder of RetroProgramming Italia – RP Italia

Talking about Sprite Collisions, a topic discussed, moreover, in part 3 of the Felice Nardella's Tutorial about how to create a Game in Full BASIC V2 (this can be found on the RP Italia group on Facebook), it should be noted that these are also events that generate an IRQ type Interrupt Hardware request as previously seen when we have mentioned the sources of Interrupt of the dedicated CIA #1 chip.

The VIC-II graphics chip, the C64 GPU, has 4 sources of Interrupt requests (see Fig. 1), including the Sprite to Sprite Collision and the Sprite to Data Collision.

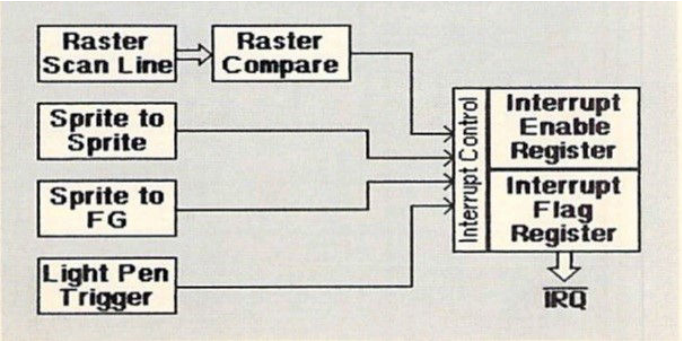


Fig. 1 - Interrupt Sources

Similarly to the 2 CIA chips, Interrupt sources from VIC-II are controlled by 2 Internal Registers (See FIG. 2): the IRQ Mask Register (IRQMASK) mapped to address 53274 ($D01A) and THE VIC Interrupt Flag Register (VICIRQ) mapped to address 53273 ($D019).
The Interrupt Flag Register reports when an Interrupt originates from VIC-II and indicates which source generated the Interrupt request.

Bit 7 and the corresponding bits responsible for the Interrupt request will be set to 1.

Following the occurrence of an Interrupt request, a "latch" (a sort of "lock") will be set in the Interrupt Flag Register so that no further Interrupt requests can originate from the same source until the latch is reset by writing 1 in the specific bit related, precisely, to the source of the Interrupt.

The IRQ Mask Register (or Interrupt Enable Register) is used instead to enable/disable the 4 sources of Interrupt requests of the VIC-II.

It is necessary to write 1 or 0 to one of the first 4 bits of Registry 53274 ($D01A) to enable or disable one of the Interrupt sources, respectively.

When there is a collision between 2 or more Sprites or between a Sprite and a Character or a part of the Bitmapped

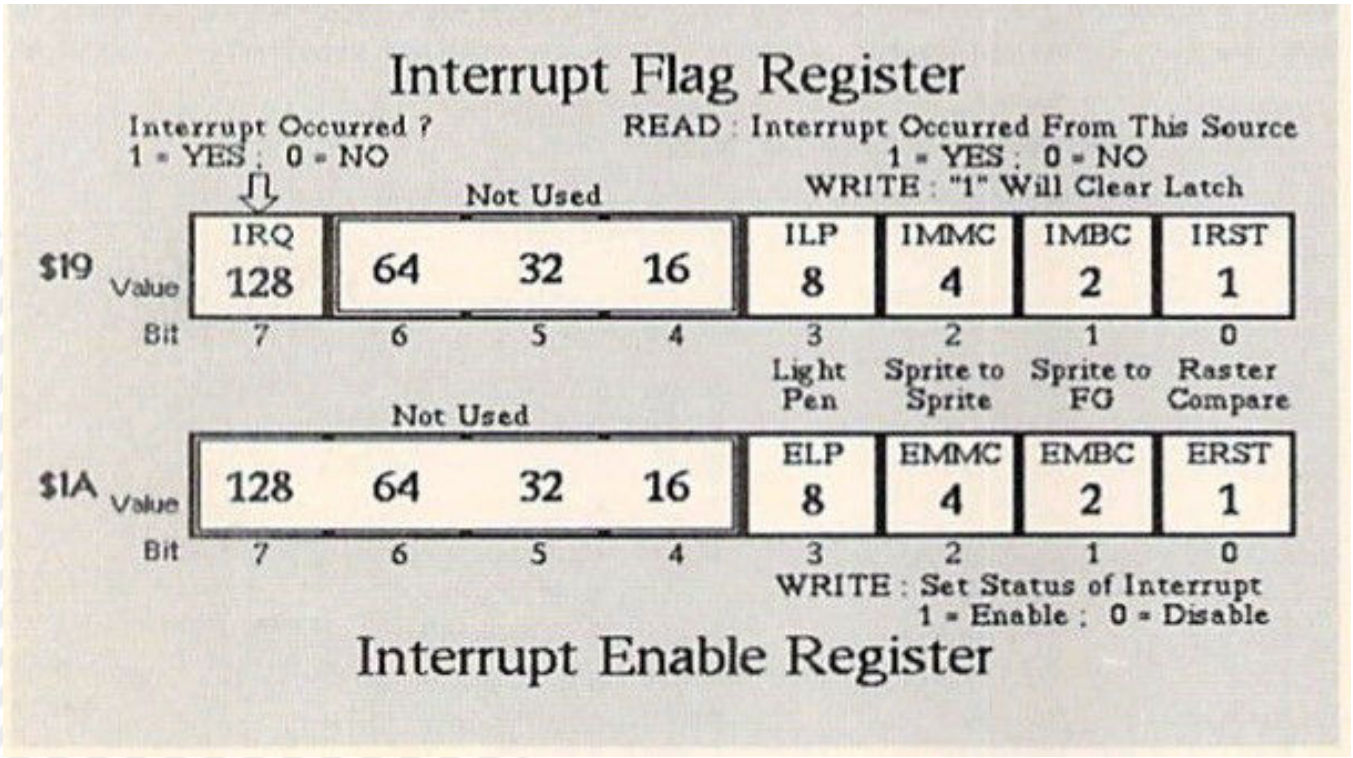


Fig. 2 - Interrupt Flag Register

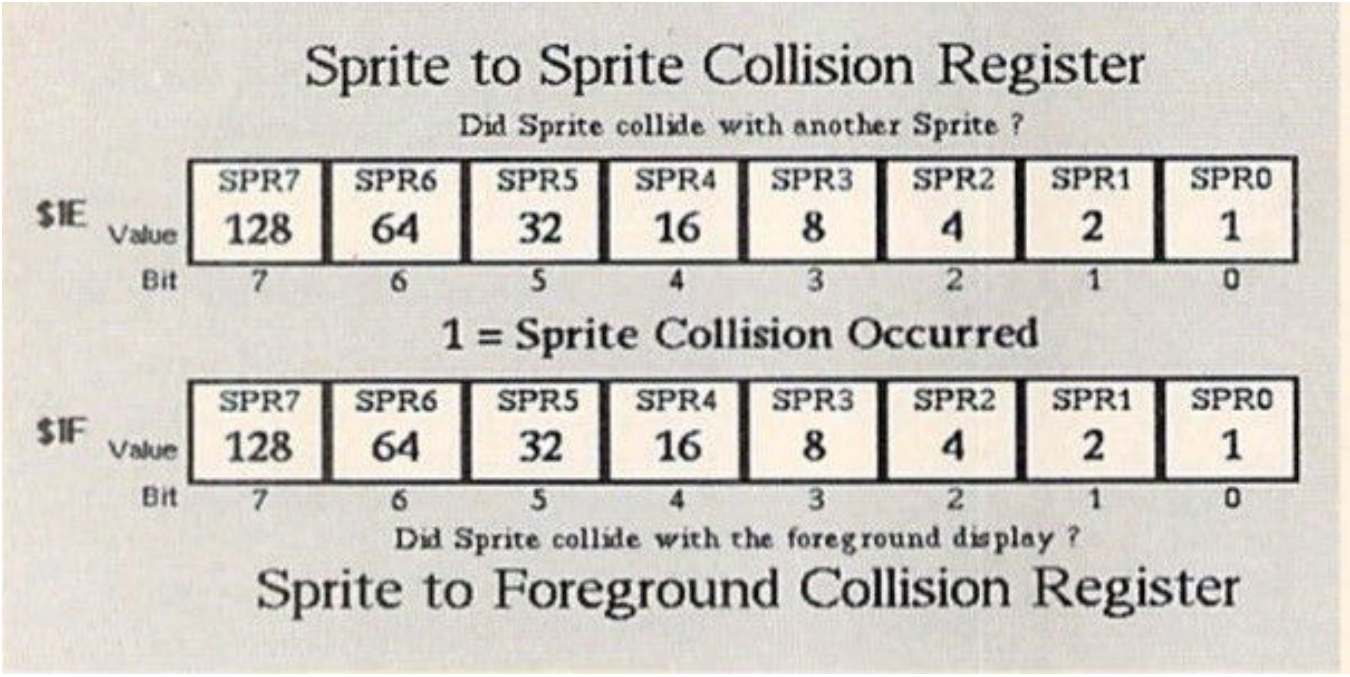


Fig. 3 - Sprite to Sprite Collision Register

Screen, another 2 Registers come into play (see FIG. 3) which are respectively the Sprite to Sprite Collision Register (SPSPCL) mapped at address 53278 ($D01E) and the Sprite to Data Collision Register (SPBGCL) mapped at address 53279 ($D01F), the latter, in the technical literature, is also called Sprite to Foreground Collision Register or again Sprite to Background Collision Register but we prefer to use the prevailing name proposed by the C64 PROGRAM'S REFERENCE GUIDE which we consider to be more relevant to the nature of the Collision examined and less subject to interpretative ambiguity by the reader (as mentioned, Sprite to Data Collision Register).

Each single bit of both Collision Logs corresponds to one of the 8 possible Sprites.

By reading these Records it is therefore possible to verify which Sprites were involved in a Collision as the respective bits will be set to 1.

Reading the Records will result in the content being reset.

With regard to Collision, it is essential to note that it will only occur when the position of a non-transparent pixel of the rectangular area of a Sprite (24x21 pixels in the case of a Normal Size Sprite) coincides with the position of a non-transparent pixel of another Sprite or when it meets a Foreground pixel of a graphic element of the Display (Character Block in Text Mode or Cell in Bitmap Mode).

In Standard Display Color Mode, non-transparent (Foreground) pixel is 1 and the Background pixel is 0.

In the MultiColor Mode (MCM), Foreground pixels correspond to the bit combinations “10” and “11”, while the Background pixels also include, exceptionally, the bit pair “01” in addition to the bit combination “00”.
Therefore, keep in mind that bit pair “01” DOES NOT generate Collision.

Remember, for the convenience of the reader, that the color corresponding to the bit pair “01” in a Sprite Multicolor is set by the Sprite Multicolor Register 0 (SPMC0) mapped to address 53285 ($D025).

In the Multicolor Character Mode, however, the color can be chosen through the Background Color 1 Register (BGCOL1) mapped to address 53282 ($D022) and finally in the case of the Multicolor Bitmap Mode, the color of the bit pair “01” can be set through the high nibble of the Screen Memory location corresponding to the Bitmap Cell.

That's all folks!

The group **RetroProgramming Italia - RP Italia** can be reached at:
https://www.facebook.com/groups/retroprogramming/

May the FORTH be with us - part one

by Francesco Fiorentini

Intrigued by **Michel Jean**'s article (see RMW 25 ITA and RMW 3 ENG) and mindful of a conversation with **Ermanno Betori** some time ago, where we had discussed the possibility of learning FORTH, I decided to give myself a chance and start studying this unusual language.

Introduction

I would like to say that I do not currently know the Forth and therefore it is not my intention to teach this language, but rather to create a series of articles with some personal considerations that could be useful to those who decide to follow this same path.

There are dozens of manuals, guides and code examples for any language on the web, and Forth is obviously no exception, but I personally wanted to enrich my paper collection by purchasing **Steven Vickers' FORTH Programming** book. This book is a reprint of the Jupiter ACE manual on the 35th anniversary (in 2017) of the launch of this computer, which happened in 1982.

The examples that I propose will therefore be compatible with the Forth version of the Jupiter ACE, although in theory it should be possible to run them on other machines. The TI99/4A for example has several implementations of Forth, some of which are very rich in commands.

The Forth Dictionary

As we read in Michel Jean's excellent article, Forth is a rather particular programming language. So, let us begin to look at some of these particularities together. The first and perhaps the most obvious is its dictionary.

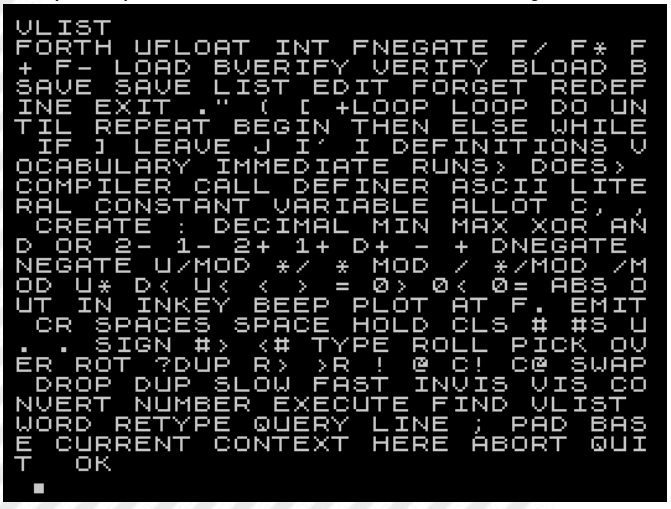


Fig. 1 - Jupiter ACE Forth Dictionary

Try this command:

```
vlist
```

If you ran it on the ACE Jupiter emulator, or even better on real hardware if you are among the lucky owners, you should see the result of [Fig. 1].

This collection of commands is known as the Forth Vocabulary. The idea behind Forth is to expand this list by creating new commands from existing ones. Once these additional commands are created, they will enrich what is known as the Forth Dictionary.

Obviously, these new commands can also be used to create new ones: you can easily understand the potential of Forth over other programming languages.

But let's take a quick look at an example of how easy it is to enrich basic language with new features.

Then type the command:

```
cls
```

to clean the screen and then type the following code:

```
: hello
CR. "hello world"
;
```

For the time being, let us not worry about the instructions, we will analyse them later, but concentrate on the result. Be careful to respect all the spaces, even the one between the first double apex and the h of hello, otherwise you will

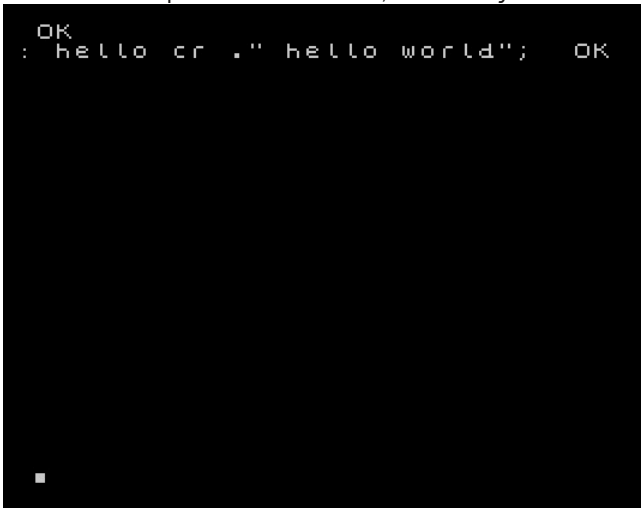


Fig. 2 - The HELLO program

certainly receive an error.
If you have done everything correctly you should be faced with a screen like the one in [Fig. 2]

Let's also try to run our program by typing:

```
hello
```

The result is certainly not exciting, but we have successfully created our first Hello World in Forth:

```
 OK
: hello cr ." hello world";  OK

hello
hello world OK
```

Let's try running the VLIST command again.
As we can see, we have created a new Forth command that has been added to the list of natively available commands. We have therefore contributed to enriching the Dictionary of our Forth.

```
vlist
HELLO FORTH UFLOAT INT FNEGATE F
/ F* F+ F- LOAD BVERIFY VERIFY B
LOAD BSAVE SAVE LIST EDIT FORGET
 REDEFINE EXIT ." ( [ +LOOP LOOP
 DO UNTIL REPEAT BEGIN THEN ELSE
 WHILE IF ] LEAVE J I' I DEFINIT
IONS VOCABULARY IMMEDIATE RUNS>
DOES> COMPILER CALL DEFINER ASCI
I LITERAL CONSTANT VARIABLE ALLO
T C, , CREATE : DECIMAL MIN MAX
XOR AND OR 2- 1- 2+ 1+ D+ - + DN
EGATE NEGATE U/MOD */ * MOD / */
MOD /MOD U* D< U< < > = 0> 0< 0=
 ABS OUT IN INKEY BEEP PLOT AT F
. EMIT CR SPACES SPACE HOLD CLS
# #S U. . SIGN #> <# TYPE ROLL P
ICK OVER ROT ?DUP R> >R ! @ C! C
@ SWAP DROP DUP SLOW FAST INVIS
VIS CONVERT NUMBER EXECUTE FIND
VLIST WORD RETYPE QUERY LINE ; P
AD BASE CURRENT CONTEXT HERE ABO
RT QUIT  OK
 ■
```

But that's not all. As we said, our Hello command is now an integral part of the Forth dictionary and can therefore be used to create new commands.

Type the following code:

```
: clshello
cls
hello
;
```

If you have done everything correctly you should receive the OK message at the end. Then try the new command:

```
clshello
```

That's right! It cleans the screen and then prints the Hello World message on screen.

```
hello world OK
 ■
```

If we repeat the VLIST command again, we'll see our CLSHELLO command at the top of the command list, followed by the previous HELLO.

The commands : e ;

As you have probably already guessed, the command **:** (colon) is used to declare the definition of a new word of the Forth Dictionary.

The **;** command (semicolon), on the other hand, is used to tell Forth that the definition is finished.
What if we want to review and correct our listings? How can we do that?

LIST, EDIT, and FORGET commands

We can use the LIST command:

```
hello list
```

```
 OK
list hello
: HELLO
 CR ." hello world"
;
 OK
list clshello
: CLSHELLO
 CLS HELLO
;
 OK
 ■
```

and the EDIT command respectively:

```
edit clshello
```

```
 OK
list hello
: HELLO
 CR ." hello world"
;
 OK
list clshello
: CLSHELLO
 CLS HELLO
;
 OK
edit clshello



 ?: CLSHELLO
  CLS HELLO
 ;
```

What if we wanted to delete one of the newly created commands instead? Nothing easier than simply telling Forth to forget the command:

```
forget hello
```

```
forget hello vlist
FORTH UFLOAT INT FNEGATE F/ F* F
+ F- LOAD BVERIFY VERIFY BLOAD B
SAVE SAVE LIST EDIT FORGET REDEF
INE EXIT ." ( [ +LOOP LOOP DO UN
TIL REPEAT BEGIN THEN ELSE WHILE
 IF ] LEAVE J I' I DEFINITIONS V
OCABULARY IMMEDIATE RUNS> DOES>
COMPILER CALL DEFINER ASCII LITE
RAL CONSTANT VARIABLE ALLOT C, ,
 CREATE : DECIMAL MIN MAX XOR AN
D OR 2- 1- 2+ 1+ D+ - + DNEGATE
NEGATE U/MOD */ * MOD / */MOD /M
OD U* D< U< < > = 0> 0< 0= ABS O
UT IN INKEY BEEP PLOT AT F. EMIT
 CR SPACES SPACE HOLD CLS # #S U
. . SIGN #> <# TYPE ROLL PICK OV
ER ROT ?DUP R> >R ! @ C! C@ SWAP
 DROP DUP SLOW FAST INVIS VIS CO
NVERT NUMBER EXECUTE FIND VLIST
WORD RETYPE QUERY LINE ; PAD BAS
E CURRENT CONTEXT HERE ABORT QUI
T  OK
 ■
```

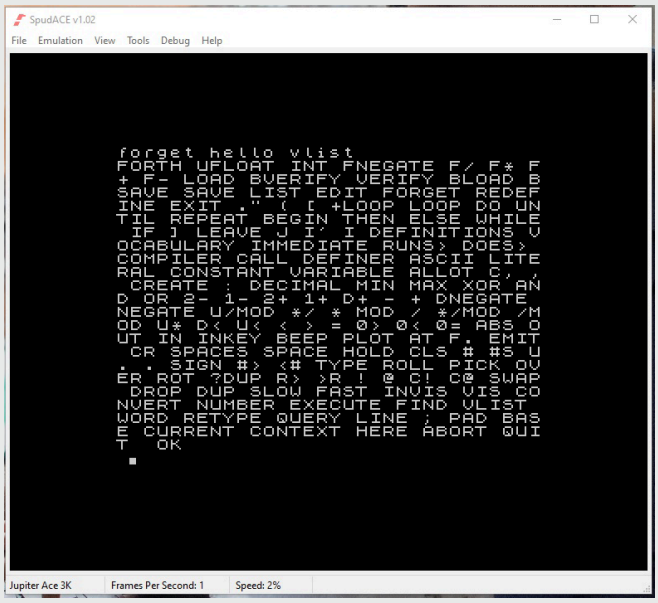
Note that the Forth is smart enough to forget the CLSHELLO command which was derived from the HELLO command.

We've come to the end of our first FORTH diary. I hope that I have aroused your curiosity a bit and that, like me, you want to discover the peculiarities of this language that, especially in Italy, went a little underrated in the golden years of the 8 bits.

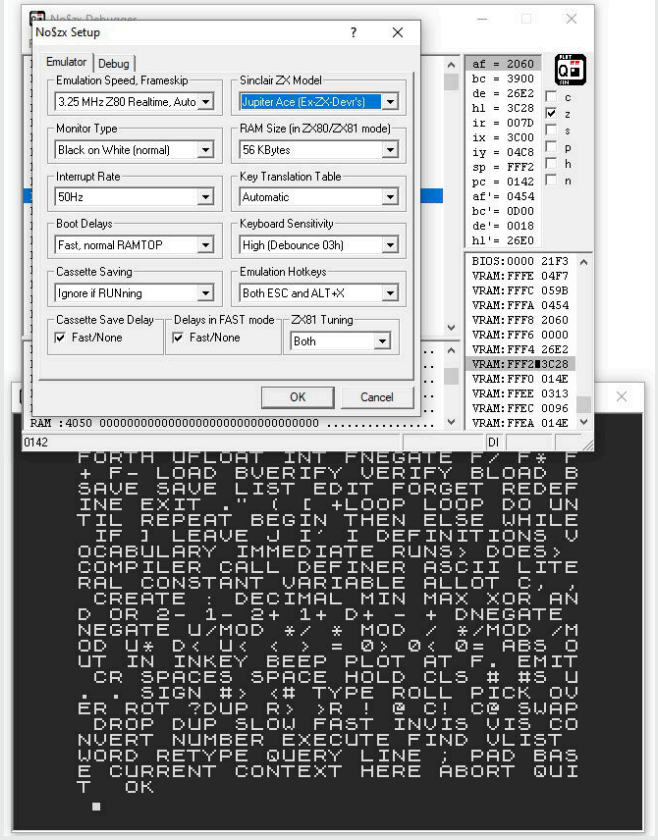
Before I drop you off at the box on the ACE Jupiter emulator, I want to wish a Happy New Year to all of you!

Jupiter ACE emulation

For a good Jupiter ACE emulation and execute the examples in the article, you can use the **SpudACE** emulator.



Alternatively the **no$zx** emulator, but having the precaution to choose the Jupiter ACE as ZX Model to be emulated from the main menu:
Options -> **Emulation Setup**.



Both emulators can be downloade from:
http://www.jupiter-ace.co.uk/emulators_win.html

LIFE: the game of life

by Marco Pistorio

While closing this issue of RetroMagazine World, we realized that, compared to previous issues, a few lines of code were missing.

In order to compensate for this deficiency, we've contacted Marco Pistorio who, basically in no time, provided us with not one, but two programs in CBM BASIC V2 ready to be formatted. Our choice fell on this algorithm of LIFE, the game of life.

We leave to the readers the task to analyze the code and possibly modify it to increase the number of generations.

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.[1] It is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves. It is Turing complete and can simulate a universal constructor or any other Turing machine.
From Wikipedia:
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

```
100 rem game of Life
101 poke53280,0:poke53281,0:poke646,5
102 printchr$(147);
105 :
106 rem define arrays
107 dim x$(9,9),y$(9,9)
109 :
110 fd=1:gosub 4000
115 :
130 x$(4,4)="*":x$(4,5)="*":x$(4,6)="*"
133 x$(5,4)="*":x$(5,6)="*"
135 x$(6,4)="*":x$(6,5)="*":x$(6,6)="*"
199 :
200 for gn=1 to 7
202 :
205 gosub 3000
299 :
300 rem calculates the next generation
310 for j=2 to 8
315 :
320 for k=2 to 8
321 :
325 gosub 900
326 :
327 cr$=left$(x$(j,k),1)
330 if cr$=" " then 400
331 :
335 if sv<=1 then y$(j,k)=" "
336 if sv>=4 then y$(j,k)=" "
337 if sv>=2 and sv<=3 then y$(j,k)="*"
350 goto 500
399 :
400 if sv=3 then y$(j,k)="*"
499 :
500 next k,j
600 gosub 2000
610 fd=2:gosub 4000
699 :
700 next gn
800 end
899 :
900 rem cell proximity control
901 sv=0
902 if x$(j-1,k-1)="*" then sv=sv+1
903 if x$(j-1,k)  ="*" then sv=sv+1
904 if x$(j-1,k+1)="*" then sv=sv+1
905 if x$(j,k-1)  ="*" then sv=sv+1
906 if x$(j,k+1)  ="*" then sv=sv+1
907 if x$(j+1,k-1)="*" then sv=sv+1
908 if x$(j+1,k)  ="*" then sv=sv+1
909 if x$(j+1,k+1)="*" then sv=sv+1
910 return
999 :
2000 rem --- copy arrays ---
2001 for j=1 to 9
2002 for k=1 to 9
2003 x$(j,k)=y$(j,k)
2004 next k,j
2005 return
2999 :
3000 rem --- drw arrays 9*9 ---
3001 :
3002 print"generation:";gn
3005 :
3010 for j=1 to 9
3020 for k=1 to 9
3025 :
3030 print mid$(x$(j,k),1,1);
3035 :
3040 next k
3050 :
3060 print
3070 next j
3080 return
3999 :
4000 rem --- fill the arrays ---
4001 for j=1 to 9
4002 for k=1 to 9
4003 :
4004 if fd=1 then y$(j,k)=" ":x$(j,k)=" "
4005 if fd=2 then y$(j,k)=" "
4006 :
4007 next k,j
4008 return
```

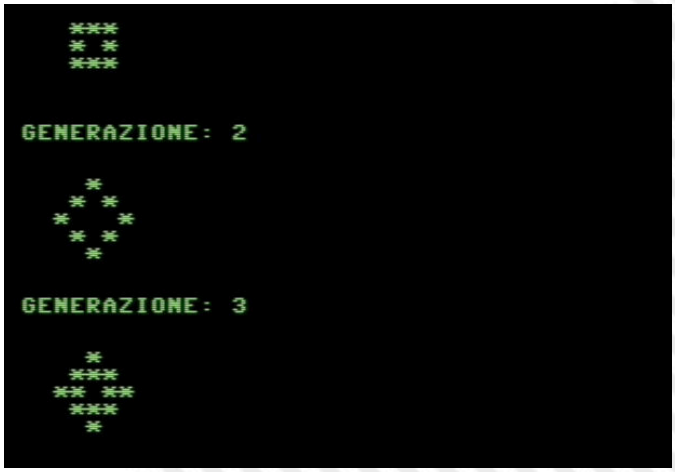


Fig. 1 - Output of the program

Accessing a PETSCII BBS from a web browser

by Antonino Porcino

In the 23rd issue of the Italian RMW, Francesco Sblendorio illustrated his project “PETSCII-BBS Builder”, a Java framework with allows you to build a complete BBS for Commodore computers connected to the Internet instead of through the analogue telephone line as it once used to be.

As he mentioned in the his interview, the project was fairly successful and the framework was used to create other BBSs in addition to the one created by Francesco himself, which currently runs at bbs.sblendorio.eu:6510.

To connect a Commodore 64 to the BBS, a modem card

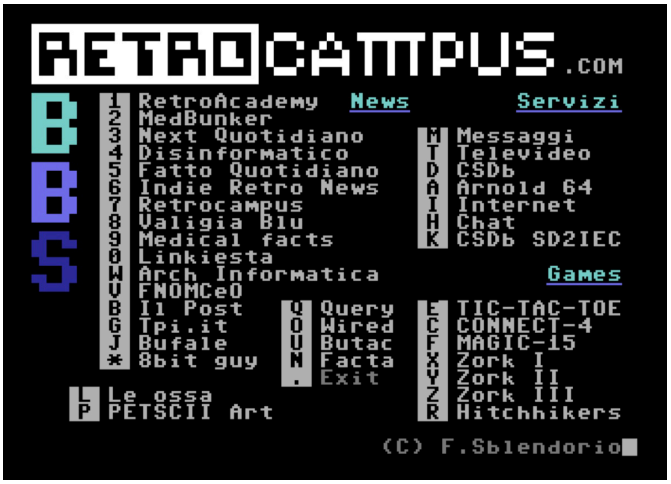


Fig. 1 - The BBS main menu

must be plugged into the user port, such as Pasquale De Luna's "KC64Wifi” card [1]. Unfortunately, however, not many C64 owners have such a modem and this somewhat limits the use of the same BBS by most of them.

Thinking about this problem, some time ago I came up with the idea of setting up a C64 emulator and having it connected in some way to Francesco Sblendorio's BBS. In fact, you could already do this with the good old VICE emulator, after having it properly configured; however, it is not really easy for inexperienced users. My idea, on the other hand, was to make available an online emulator that can be reached directly from an Internet browser, which would then be ready for use without any installation or configuration, thus allowing access to the BBS even by the occasional user who just wanted to browse for a little while.

The idea of the emulator in the browser has always fascinated me, for the ease of access that it involves, but also because using the JavaScript language the emulator becomes modifiable to your liking by opening infinite possibilities. For example, you can inspect or modify “open heart” memory locations, while the emulator is running without interrupting execution.

Over the last few years I have created several emulators running in a browser (Laser 500, LM80C, ChildZ, General

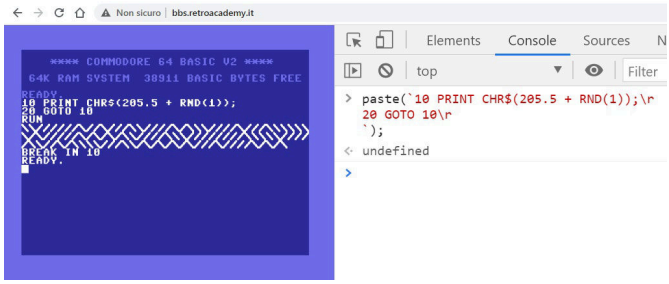


Fig. 2 - Example of interaction between JavaScript and emulator: JavaScript injects BASIC code into the C64 using the function "paste()"

Processor and VIC20), mainly for personal use, for example to debug my cross-compiled programs or to reverse-engineer some of the computers I have dealt with.

For the BBS, however, I didn’t want to start from scratch, both because writing a C64 emulator is quite complex due to its custom chips and because the Commodore 64 is fortunately one of the most emulated computers so far; so it made no sense to reinvent the wheel, it was easy to look around in the open-source world to see what was available.

The “chips” project

After examining different solutions, my choice went to Andre Weissflog's fascinating “chips” project, which you can see in action on the website "Tiny 8 bit Emulators” [2]. The idea behind the project is truly innovative: rather than creating a classic emulator, the author thought of simulating separately the individual chips that make up a computer in the form of C language files, and then "virtually" connecting them together. This way you can recreate different systems simply by changing the chips by which the system is composed.

The emulation takes place precisely at the single clock cycle (cycle-exact) and all pins are emulated as in the real chips. The implementation is very efficient and it is all written in standard C language; each chip resides in a “.h” file without the need for other dependencies and can be

included in your project, in fact that's how I emulated the Z80, the TMS-9929 and the AY-3-8910 of the LM80C computer [3].
Andre Weissflog's brilliant idea was to connect the chips together with a simulated 64-pin bus; with modern 64-bit architectures, this bus can safely reside in a CPU word and be exchanged in the C functions within a normal processor register, thus guaranteeing maximum efficiency. For example, the A0-A15 address bus lines reside in bits 0-15 of the word, while the 8 bits of the data bus reside in bits 16 to 23. Other control signals such as RW, IRQ, NMI, etc... reside in the subsequent bits. Writing and reading from this 64-bit word is a very fast operation as modern CPUs can move data and perform shift operations in the same clock cycle.
To use one of these emulated chips, simply perform the function called "tick()" which is equivalent to advancing the chip of a clock cycle. In this step, the internal registers and I/O feet are updated just as a real chip would.

In Andre Weissflog's project many of the integrated circuits of the 8-bit computers (Z80, 6502, etc.) were implemented. However, since it is a project carried out by a single person, the emulation is not always perfect and there are incomplete areas that the author unfortunately did not have time to finish. However, we are amazed at the enormous amount of work done and browsing the sources of the project is an extremely instructive exercise for the retrocomputing enthusiast.
In addition to individual chips, ready-to-use systems have been created in the project, with already connected circuits that therefore emulate a complete computer (VIC20, C64, ZX Spectrum, Amstrad CPC, etc.); for example, the C64 emulator resides in the file “c64.h” to which you only have to connect keyboard, audio and video -- which in my project I connected to an Internet browser.

However, there is one issue: the browser speaks JavaScript while “chips” is written in C. How to make the two communicate with each other? And here comes help from a recent HTML technology called WebAssembly. It is essentially a standardized virtual machine that runs in a sandbox inside the browser and is able to execute its byte code very efficiently. This byte code is not linked to a specific CPU architecture, but it is generic without even registers (it only uses the stack); moreover, as it is structured, it translates very well into machine language and the performance is close to the execution of the native code. The byte code is obviously not handwritten, but is generated by the various compilers, in my case by "emscripten” [10] the C/C++ compiler created specifically for the WebAssembly.

With WebAssembly we can run C programs within the browser, or even an entire C64 emulator. The browser keyboard, video and audio must be “connected” to the emulator using the appropriate JavaScript code. The screen is nothing more than a normal HTML canvas, while for the sound there are the "API audio”s of the browser, which essentially reproduce on the speakers the sample buffer that the emulator provides at regular intervals.
Having previously written an emulator for VIC-20 with Andre Weissflog's project, I simply adapted its code for C64, taking advantage of the similarity between these two computers. In a short time I created a first working prototype with the classic “READY” prompt and the 38911 bytes available to the user.
Compared to the original project, I made some minor changes, such as adding the RESTORE key (which becomes "PGUP” on the PC keyboard) that was not implemented. To do this, after consulting the C64 wiring diagram, I saw that RESTORE key line from the keyboard matrix goes directly to the NMI pin of CPU 6510, so it was just a matter to add the following lines in the emulator code:
Where “pins” is the 64-bit word described earlier that contains the entire bus. As you can see, to turn on the NMI pin it was sufficient to make a binary OR using the

```
static uint64_t _c64_tick(c64_t* sys, uint64_t pins) {

    // RESTORE key
    if(sys->kbd.scanout_column_masks[8] & 1) {
        pins |= M6502_NMI;  /* RESTORE key is pressed */
    }
```

pipe operator “|” of the C.

The virtual modem

Set up the emulator, how do we proceed for the BBS? My first idea was to simulate one of the various modems available in order to use existing communication programs. But I immediately realized the enormous effort required, so I opted for another quicker solution: implementing my own virtual modem. In the end it was not a big effort: the modem simply had to receive and send characters.
Since there are no I/O ports on the C64 (unlike the Z80, for example), my virtual modem had to be mapped to memory, as is already the case for all devices in the C64.
In the “c64_tick()" function, I went to find where the I/O mapped to memory is managed and subtracted the space

from $D7F0 to $D7FF from SID, reserving it for my virtual device:

At each reading in this range, I run the JavaScript function "modem_read()" and at each writing call "modem_write()", so as to intercept the modem accesses comfortably from the JavaScript side:

```
else if (addr < 0xD7F0) {   // was D800
    /* SID (D400..D7FF) */
    sid_pins |= M6581_CS;
}
else if (addr < 0xD800) {
    /* MODEM (D7F0..D7FF) */
    modem_access = true;
}
else if (addr < 0xDC00) {
    /* read or write the special color Static-RAM bank (D800..DBFF) */
    color_ram_access = true;
}
```

My virtual modem is structured as follows:

DATA_IN and DATA_OUT are simply the locations where the characters are transmitted and received from the modem.

```
else if (modem_access) {
    if (pins & M6502_RW) {
        /* modem read: chiama la funzione JavaScrip "modem_read(data)" */
        byte data = (byte) EM_ASM_INT({ return modem_read($0); }, addr);
        M6502_SET_DATA(pins, data);
    }
    else {
        /* modem write: chiama la funzione JavaScript "modem_write(addr,data) */
        uint8_t data = M6502_GET_DATA(pins);
        byte unused = (byte) EM_ASM_INT({ modem_write($0,$1); }, addr, data );
    }
}
```

DATA_REQ is a flag indicating whether there are characters in the receive buffer. CONNST returns the status connected/not connected to the BBS: I use it to change the color of the screen from black (connected) to red (not connected).

```
// porte del modem virtuale
dim MODEM_DATA_OUT    as byte at $D7F2    ; caratteri da inviare
dim MODEM_DATA_IN     as byte at $D7F0    ; caratteri in arrivo
dim MODEM_DATA_REQ    as byte at $D7F3    ; 1 se ci sono caratteri in arrivo nel buffer
dim MODEM_ACK         as byte at $D7F1    ; porta per l'handshake
dim MODEM_CONNST      as byte at $D7F4    ; status del modem (0=connesso, <>0 disconnesso)
```

As far as ACK is concerned, this allows a sort of handshake with the modem for the exchange of characters in reception. Initially there had to be no handshake because the data from the modem to CPU had to pass through the single I/O read clock cycle. Then, however, this system proved unreliable, sometimes some characters were inexplicably lost.

I did not understand exactly why, but rather than getting lost in the meanders of an infinite troubleshooting (considering that there is an emulator not very tested), I preferred to solve quickly with a brutal handshake mechanism: after the CPU reads the incoming byte, it sends on the ACK port first the value "0" and immediately after "1"; in this way the transition from 0 to 1 indicates that the character was acquired by the CPU and the modem can proceed with the next one.

The terminal program

Having developed the virtual modem, I needed a communication program to run on the emulated C64, since none of the existing ones could be compatible. To read and print the characters on screen, Francesco Sblendorio suggested that I simply use the routines CHROUT ($FFD2) and GETIN ($FFE4) of the C64 kernal, as is already the case in his software for the "Ultimate 64" card. I then wrote a small assembly program that essentially loops infinitely between reading incoming characters and transmitting keyboard typed characters to the modem.

Rather than using the bare and raw assembly, I have used a macro language of my invention that for years has accompanied me in all my projects for 6502. This tool (which I called "Asmproc" [4]) allows me to write an assembly that can be read using constructs like IF-THEN, DO-LOOP etc... avoiding the use of the classic branch-here and branch-there that completely obscure the comprehensibility of the code in 6502.

```
; stampa il carattere a video
jsr CHROUT

; notifica al modem che il carattere è stato ricevuto
lda #0
sta MODEM_ACK
lda #1
sta MODEM_ACK
```

Fig. 3 - The terminal program handshakes with the virtual modem by sending "0" and "1" bytes on the ACK port

Initially the BBS terminal program was really a handful of bytes, then it was gradually expanded to as many as 165 bytes, with the addition of the flashing cursor, the CHR$(7) bell and other small details. If you're curious, find the full source at the bottom of the article.

For my BBS purpose, the terminal program had to start

```
cursor_off:
    ldy $cc
    if zero then
        ldy #$01
        sty $cd
        do
            ldy $cf
        loop while not zero
    end if
    ldy #$ff
    sty $cc
    rts
```

Fig. 4 - Example of structured assembly programming with Asmproc: the implementation of the "cursor_off" routine; as you can see there are no labels or BEQ/BNE

automatically when the emulator started; but since on the C64, the memory is reset at boot time, it was necessary to "inject" the program into RAM only after the end of the same boot. One trick I came up with is to listen to location 204, which is the one that determines whether the cursor is present on the screen or off; when the cursor is on, it means that the "READY" appeared on the screen and then you can inject the program into RAM and then immediately after inserting the characters "RUN" + RETURN into the keyboard buffer, causing it to run.

WebSocket Tunnel

After the program on the C64 that sends characters to the virtual modem, how do these get from here to the BBS? In a normal world this would happen with a trivial TCP connection on port 6510 which is the one where the BBS turns. But no, there is an obstacle: for cybersecurity reasons the browser cannot open any TCP connection with the outside world.

To make up for this lack in recent years, a new standard called WebSocket has been introduced: it is a protocol with which connections similar to TCP sockets can be conveyed, channelling them through the normal port 80 of the HTTP service.

However, the BBS does not accept WebSocket connections, only TCP connections. So what should we do? The idea I had was to create a special utility to run separately that accepts WebSocket connections from the emulator/browser and forwards them to the normal TCP port of the BBS. That's what's called tunnelling in jargon.
So I wrote an application in server-side JavaScript, which is Node.js. With a lot of imagination I called it "websocket-to-tcp"; like everything else it is open source and you can find it on Github [5]. The utility can be run locally ("localhost") by launching it from the command prompt or you can simply use the one that has been installed directly on the BBS server so that the user does not have to perform any installation.

However, there is a further complication: since the connection to the WebSocket could originate from an emulator running on a site in HTTPS (HTTP Secure), it was necessary to implement this type of connection as well. Francesco Sblendorio then proceeded to certify the BBS website with "letsencrypt" [6] (an authority that issues certificates for HTTPS free of charge) and subsequently installed the

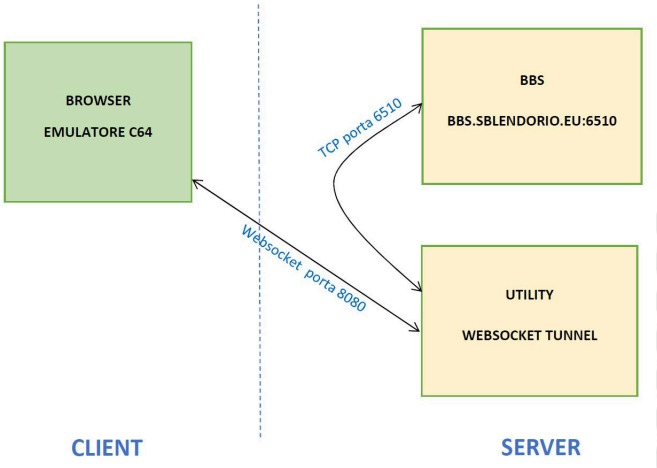
```
$ wstcp -t bbs.sblendorio.eu -p 6510 -w 8080 -n bbs
Fri Dec 04 2020 18:40:23 GMT+0100 (GMT+01:00) Server is listening on port 8080 protocol "bbs"
Fri Dec 04 2020 18:41:17 GMT+0100 (GMT+01:00) connection accepted from https://nippur72.github.io
Fri Dec 04 2020 18:41:17 GMT+0100 (GMT+01:00) TCP connection established
Fri Dec 04 2020 18:41:17 GMT+0100 (GMT+01:00) TCP ready
Fri Dec 04 2020 18:41:40 GMT+0100 (GMT+01:00) TCP connection closed
Fri Dec 04 2020 18:41:40 GMT+0100 (GMT+01:00) peer ::1 disconnected
```

Fig. 5 - The "wstcp" utility running on the BBS server creates a tunnel between WebSocket and TCP

relevant certificates on the server that is now able to establish encrypted connections (your chats on the BBS are therefore safe from spies!).

The tunnelling utility between WebSocket and TCP has been written in Node.js because it has the advantage of being available on all OS (Window, Linux, Mac) and of having libraries ready and easy to use. Its operation is very simple: it simply creates a web server on the specified port on which WebSocket traffic passes while waiting for a connection from the browser. Once connected, it opens a TCP connection to the BBS and exchanges data between the two. A hundred lines of code.

On the browser side, however, it is the virtual modem that initiates communication via WebSocket and transmits the characters it receives to the CPU.
The figure summarizes the connection system between browser and BBS:



Using the emulator

Not shown in the figure is also the Web server on port 80 that provides the HTML page containing the emulator when the user navigates to the BBS address.

In fact, you can open one of the following addresses with any browser:

http://bbs.retrocampus.com or http://retrocampus.com/bbs
http://bbs.retroacademy.it

Alternatively, you can access the emulator repository on Github and run the terminal program with the "load" parameter:
https://nippur72.github.io/c64-emu/?load=nippur72/terminal.prg

You can also use only C64 without BBS, omitting "load":
https://nippur72.github.io/c64-emu
The emulator can also be embedded within another web page with the following HTML code:
<iframe width="720" height="554" src="https://nippur72.github.io/c64-emu/?load=nippur72/terminal.prg"></iframe>

It is also possible to connect to a BBS other than the one created by Francesco Sblendorio. In this case, you will need to run the WebSocket tunnel locally, for example:
$ wstcp -t particlesbbs.dyndns.org -p 6400 -w 8080 -n bbs
(opens a WebSocket tunnel to BBS "particlesbbs.dyndns.org:6400" via port 8080)
to then invoke the emulator with the parameter "wstcp" indicating that the tunnel is active on localhost:8080:
https://nippur72.github.io/c64-emu/?wstcp=ws://localhost:8080&load=nippur72/terminal.prg

I recommend using the Chrome browser, which turns out to be the fastest, allowing it to run even on not-so-recent computers. In fact, you will notice that the browser window will tend to have a rather high CPU load; this is due to the fact that the emulation is cycle-exact, which means that you have to simulate the four C64 chips (CPU, VIC, SID and CIA) at a speed of 1 MHz using a single thread. Chrome is what does best in this business, while everyone else follows at a distance.
Other browsers have also given incompatibility problems, specifically with the "Blob.arrayBuffer()" function: on Safari 13 for Mac this function is not really implemented, while on Firefox 83 it occasionally caused bytes to arrive from the modem in the wrong order. To solve it I applied a patch found on the internet that simply re-implements the functions in JavaScript and inserts it into the class of the object "Blob" (since JavaScript is a dynamic language you can do this even at run-time). It is located in the file in the file "patch-arrayBuffer.js"

Final considerations

In this article I illustrated how I created a C64 emulator that runs in the browser, equipping it with a virtual modem, with a terminal program, and then connecting it to the BBS via the internet, through a WebSocket tunnel. Completing this project was a pleasant challenge that allowed me to confront some of the modern technologies, applying them to the world of retro-computing. I hope you found the result interesting.

Useful Links

[1] KC64Wifi by Pasquale De Luna: http://www.codingkoala.com/kc64wifi
[2] Tiny 8 bit Emulators: https://floooh.github.io/tiny8bit
[3] LM80C Emulator: https://nippur72.github.io/lm80c-emu
[4] Asmproc: https://github.com/nippur72/asmproc
[5] WebSocket to TCP: https://github.com/nippur72/websocket-to-tcp
[6] Let's Encrypt: https://letsencrypt.org
[7] BBS http://bbs.retrocampus.com and http://bbs.retroacademy.it
[8] The C64 emulator online https://nippur72.github.io/c64-emu
[9] C64-emu emulator and "terminal.prg" program sources https://github.com/nippur72/c64-emu
[10] Emscripten Compiler https://emscripten.org

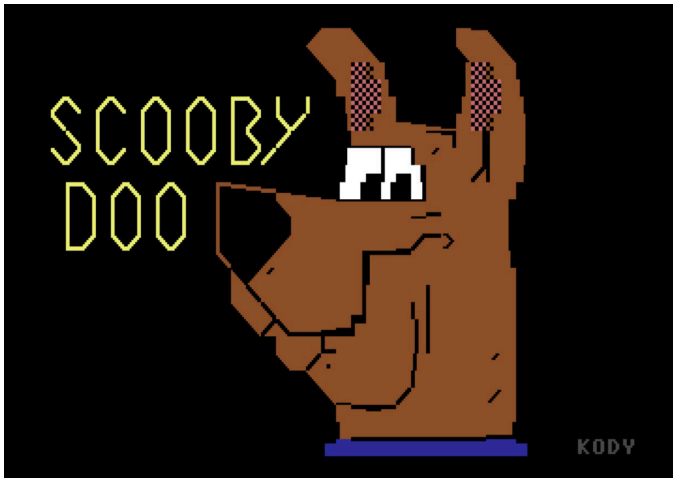


Fig. 6 - One of the images from the PETSCII-art section of the BBS ©Ivan KodydaKillah

--> On the next page the source of "terminal.lm"

```
   processor 6502

// routine del kernal
const GETIN  = $FFE4
const CHROUT = $FFD2

// porte del modem virtuale
dim MODEM_DATA_OUT    as byte at $D7F2    ; caratteri da inviare
dim MODEM_DATA_IN     as byte at $D7F0    ; caratteri in arrivo
dim MODEM_DATA_REQ    as byte at $D7F3    ; 1 se ci sono caratteri in arrivo nel buffer
dim MODEM_ACK         as byte at $D7F1    ; porta per l'handshake
dim MODEM_CONNST      as byte at $D7F4    ; status del modem (0=connesso, <>0 disconnesso)

    org 2049

basic start compact
   2020 sys {main}
basic end

main:
    lda #0   : sta 53280 : sta 53281 ; black screen
    lda #15  : sta 54296     ; max volume
    lda #14  : jsr CHROUT    ; lowercase
    lda #147 : jsr CHROUT    ; clr
    lda #5   : jsr CHROUT    ; white

terminal:
    ; controlla se ci sono caratteri da stampare nel buffer
    lda MODEM_DATA_REQ
    if a<>#0 then
        jsr cursor_off
        do
            ; esci da modalità QUOTE e INS
            ldx #0
            stx $D4
            stx $D8
            ; legge il carattere dal modem
            lda MODEM_DATA_IN
            ; se è CHR$(7) emette il suono della campana
            if a==#7 then jsr term_bell
            ; stampa il carattere a video
            jsr CHROUT

            ; notifica al modem che il carattere è stato ricevuto
            lda #0
            sta MODEM_ACK
            lda #1
            sta MODEM_ACK
            ; controlla se ci sono altri caratteri da stampare nel buffer
            lda MODEM_DATA_REQ
        loop while not zero
        jsr cursor_on
    end if

    ; legge un carattere da tastiera e lo manda al modem
    jsr GETIN
    if a<>#0 then sta MODEM_DATA_OUT

    ; aggiorna il colore dello schermo con lo status del modem (nero/rosso)
    lda MODEM_CONNST
    sta 53280
    sta 53281

    jmp terminal

; cursor on/off, term_bell (C) Francesco Sblendorio
; https://github.com/sblendorio/ultimateii-dos-lib/blob/master/src/samples/screen_utility.c

cursor_on:
    ldy #$00
    sty $cc
    rts

cursor_off:
    ldy $cc
    if zero then
        ldy #$01
        sty $cd
        do
            ldy $cf
        loop while not zero
    end if
    ldy #$ff
    sty $cc
    rts

term_bell:
    ldy #15  : sty $D418
    ldy #20  : sty $D401
    ldy #0   : sty $D405
    ldy #249 : sty $D406
    ldy #17  : sty $D404
    ldy #16  : sty $D404
    rts
```

Making music for a retrogame

by Phaze101

Getting the lead

Composing music for a game starts with a request, usually through a private message from our Facebook Page or Groups, Discord or from our website's contact details. It could be the person doing the game or someone in the team asking "Hi, could you do music for my C64 game?" or "I heard your music in a game, would you be interested in doing music for our upcoming game on the Amiga?". And if it is someone we know it would be something like "What about some music and sound effects for my new MSX game?". Many times, we are also actively following projects and instigate the request by offering to do music for them.

Before we accept, we do a couple of checks to see what effort would be required (such as the number of tracks needed for the game) to make sure that we can meet required deadlines. We also try to see that the game project is viable and is already in progress. Unfortunately, in the past we put effort in making music for games that were never released, and that's always a pity.

Obviously, some "requests" are for our own projects, and we treat these more or less the same way as any other project.

Starting off

Once we have decided to get on board, we try to get as much information on the game itself. Video clips, screen-shots and story-line all help us to capture the feel of the game. Sometimes we are provided with music that would fit the feel of the game, although this is not always the best approach and could be counterproductive.

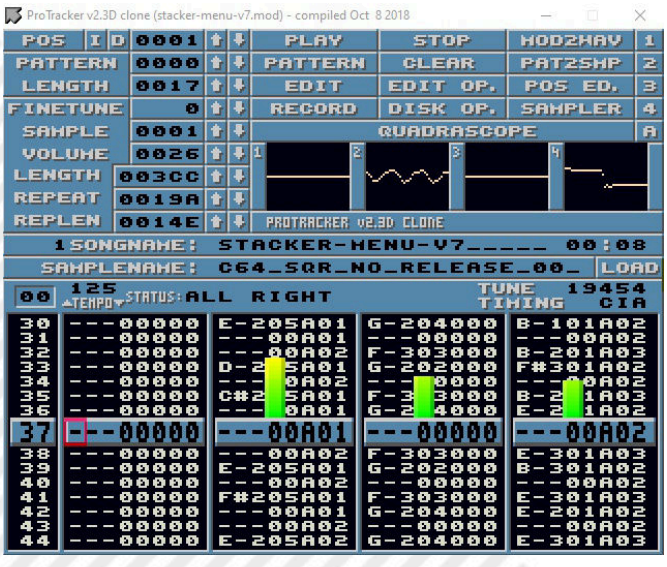


Fig. 1 - ProTracker Stacker 101

As already mentioned, the music itself is created specifically for the request. We believe that the most important thing for game music is to augment the user experience, to be part of the overall "feel" of the game. Having a memorable tune is great, but if it does not fit in the game's atmosphere it might actually make the game experience worse. In our music we try to capture the essence of the game, and this works even better when we have a free hand to take the direction which we feel would be best.

We also ask questions about the project to know more about the requirements, limitations and constraints. These would cover technical aspects such as file size limit, duration, looping, the number of channels to be used and whether music will share a channel with sound effects.

Making the music

Once we are all set to go, the creative aspect is one of the trickiest parts of the whole process. Usually, initially there would be small clips done to get the feel of the sound right. Lots of parts (and sometimes whole tunes) are scrapped (sometimes the very next day) to keep only those that sound right. There would be a lot of peer review going on within Phaze101 members at this stage. When we have created something that is good enough to start off with, feedback from the person requesting the music is also obtained, to make sure everyone is in sync before going through more iterations.

Getting the sound right can be a tricky bit. We embrace the limitations of the respective sound chip and try to make the most out of it. After all, this is the fun of doing retro music! For people used to composing music on modern platforms, where a single note or sample can be enough to create the wow factor for the whole track, this can be sheer frustration. Getting samples right on Amiga is not easy either, although there is more room for creativity there. All this is done within the constraints of the file

size / memory limitations imposed by the game requirements. Once the sound is more or less defined (although this would be constantly tweaked and sometimes completely changed), work starts on the music itself. Sometimes a draft version of the tune takes only a few hours to do. Sometimes the project is left on the side for days or weeks, waiting for a spark of inspiration to come by. This is especially true if multiple tracks are required for the same game since taking some time off the project can recharge us with new ideas. When inspiration comes, if a computer with a tracker is not handy, humming the tune and recording it on a mobile is a good alternative to make sure that the bright idea is captured. Sometimes trying out a tune on a piano keyboard or on a guitar before helps to trigger ideas which can then be transferred to a tracker in an already mature state.

As things progress, we try the music out with the game itself. As a start playing the music overlaid on existing gameplay video (or screen shots) help to see the product come together. We play the music over and over again (especially if the music would be looping multiple times within the game) to see if it quickly becomes annoying. This helps us to try to minimise the effect of repetition by doing changes that do not have a significant "cost" on the overall file size.

At this point we also start focusing on the sound effects, to make sure that everything fits within the overall feel of the game, and that the volume levels are right. Sometimes we are asked to do the music only with the sound effects being done by someone else. Other times we participate in projects where multiple tracks are done by different

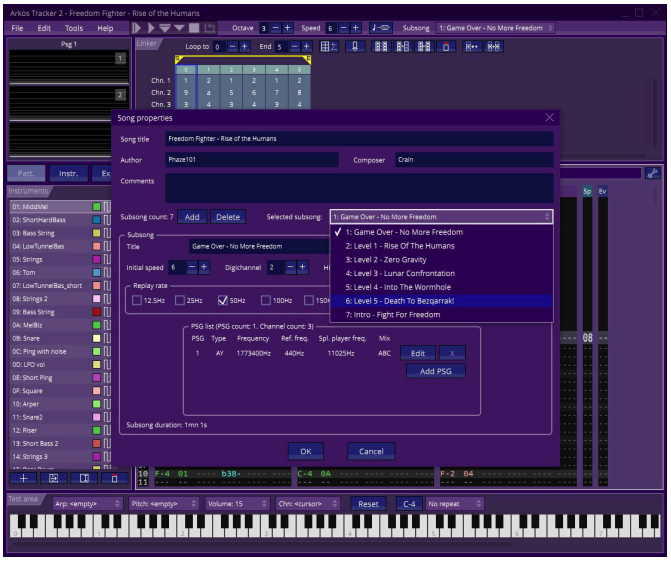


Fig. 3 - Arkos - Freedom Fighter

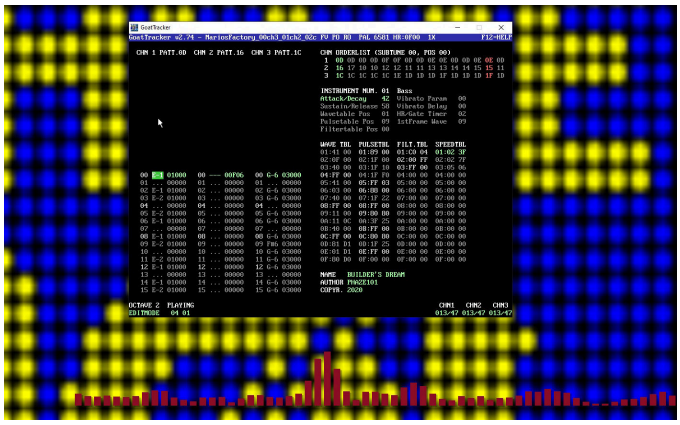


Fig. 2 - Goattracker - Mario Cement Factory

people. This makes it even more important to keep a holistic view of the product in mind, and to ensure that everything together sounds as it should.

While doing the music and especially when they are mostly finalised, we make it a point to hear the music on different devices. If it sounds right on normal cheap headphones, good expensive headphones, monitor speakers, external speakers, mobile phones and car stereo … then it will probably sound right on many devices. In collaboration with the person requesting the music, music is also heard and assessed both on emulators and on original hardware. For C64/128 we ensure that it sounds right on both 6581 and 8580 SIDs which is why we avoid heavy use of the SID filters.

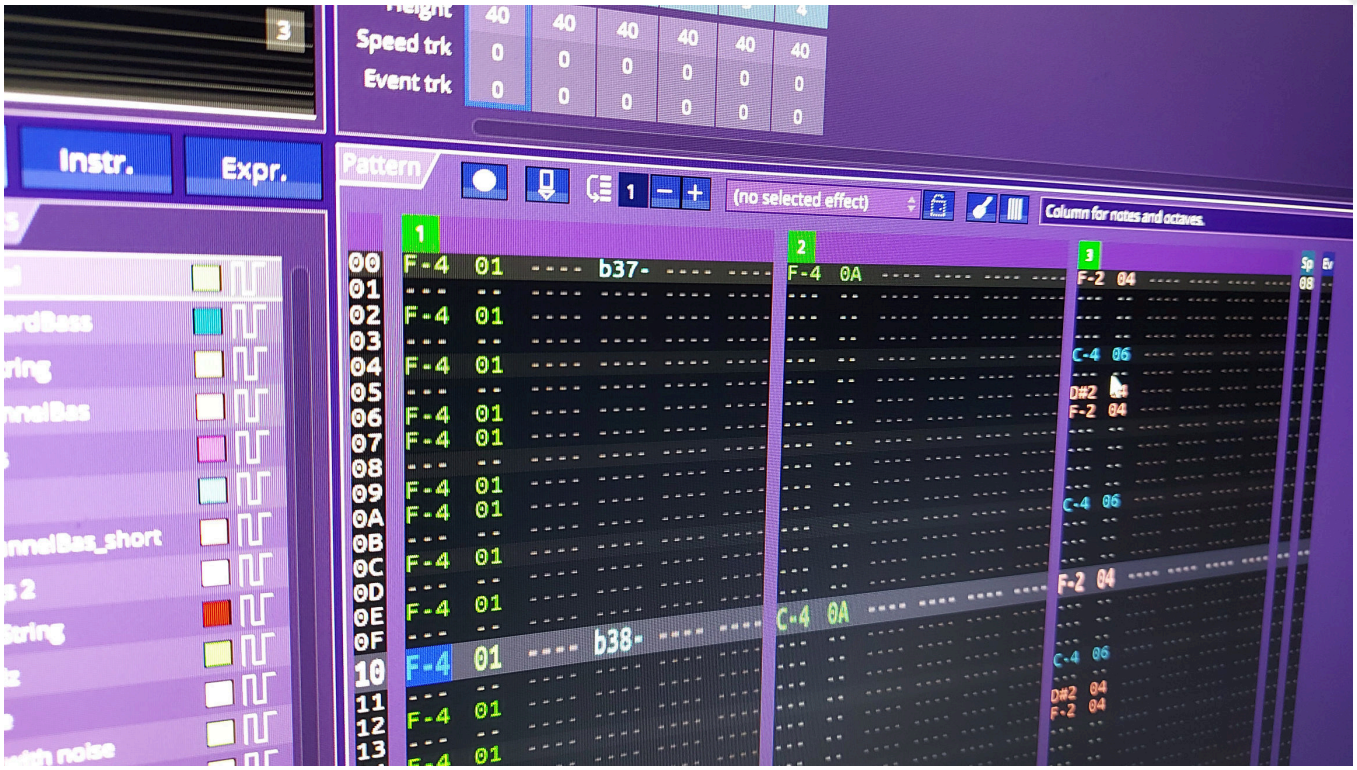
The tools

The main tool is obviously the tracker itself. We have worked with many different trackers over the years, but when doing music for others one of the main concerns is to use products that have a good stable play routine that can be used in the game itself.

We prefer Goattracker for the Commodore 64 as we believe it is currently one of the best trackers out there for the Commodore machines. We use Arkos Tracker to do music for the AY chip for MSX, Spectrum or CPC. It has a modern interface and is very user friendly as far as trackers go. When it comes to Amiga the tracker of choice is Pro Tracker (including the Windows clone by Olav Sørensen). For the Amiga we also have the help of Renoise on Windows with various VST instruments to generate samples, and tools such as Audacity to process them.

Occasionally we use other tools and are always happy to experiment and try out new stuff.

Projects in 2020



This year has been quite a busy one, as apart from our own ongoing projects we concluded the music for a number of games, with some others still in progress.

• Mario Cement Factory by Hayesmaker which is a port of the Game & Watch retro handheld game to the Commodore 64 (Reviewed in RM).

• Freedom Fighter - Rise of the Humans by Pintus Giuseppe Ettore (aka Geppo) on the MSX machines released in July 2020 (Reviewed in RM).

• Neptune Lander Elite by C64_Mark which is a variant of the classic lander game for the Commodore 64 that is bound to be released.

• Pik'n'Mix by Shallan50K on the Commodore 64 that will be released towards the end of this year.

When all is ready from the music's aspect, it sometimes takes weeks or months until the game is officially released. When it is released, we support the game and the author by putting a video on our YouTube channel or promote the game on our social media channels. It is always an honour and privilege for us to be invited to work on projects and feel proud to share the work in any possible way.

References

R1 – Phaze101 FB Page: https://www.facebook.com/Phaze101Games
R2 – Phaze101 FB Group: https://www.facebook.com/groups/Phaze101
R3 – Phaze101 Discord: https://discord.gg/tXnRe4XYV7
R4 – Phaze101 Web Site: https://phaze101.com
R5 – Phaze101 YouTube: https://www.youtube.com/c/Phaze101

Concluding the project

Introduction to ARexx – part 4

by Gianluca Girelli

GAME CODING WITH AREXX - PART 1

After analyzing the basic features of the language in the previous issues, let's now see how to implement a text adventure with ARexx. To better understand the topics that will be covered, I recommend retrieving the previous parts published in RetroMagazine numbers 20, 22 and 23, as well as the article "An introduction to Game Coding" published in number 17.

It is important to note that the proposed implementation is just one of the countless possible implementations of such a game engine. Although not necessarily the most efficient, it has been chosen for its ease of understanding even by the less experienced user, without worrying too much about issues related to memory usage both with regard to the world of play and the management of variables.

1. IMPLEMENTATION OF THE GAMING WORLD

In William Gibson's novels, the well-known "cyberpunk" writer, the term "matrix" has always been used to define the structure of cyberspace within which part of the protagonists' lives took place. In the trilogy "The Matrix", clearly inspired by these novels, the matrix was the construct within which the protagonists believed they lived. In accordance with these illustrious predecessors, here is our gaming world, defined as a two-dimensional matrix:

```
/* prepare matrix. 1= room exists */
matrix.=1
matrix.1.1=0
matrix.1.5=0
...
...
```

As we have seen in the previous numbers, the definition "matrix. =1" alone is sufficient to declare a multidimensional construct and to initialize it in order to indicate to the code which locations actually exist and which don't. Since not all "rooms" are accessible locations, as can be seen from the map in Figure 1, instructions such as "array.x.y=0" serve to "mark" some elements for other purposes. For example, the first location (indices 1.1), will be used to store the description for the "game over" text (we'll see later how). The map describes a dystopic world that revolves around a matrix of 7 rows and 8 columns. From the code point of view, using such a construct renders extremely easy to navigate it. If, for example, we assume we are in the location (x,y), going north will simply mean "move one row up", while going east will mean "advance one column". In other words:

```
when name='NORTH' then x=x-1
when name='EAST'  then y=y+1
```

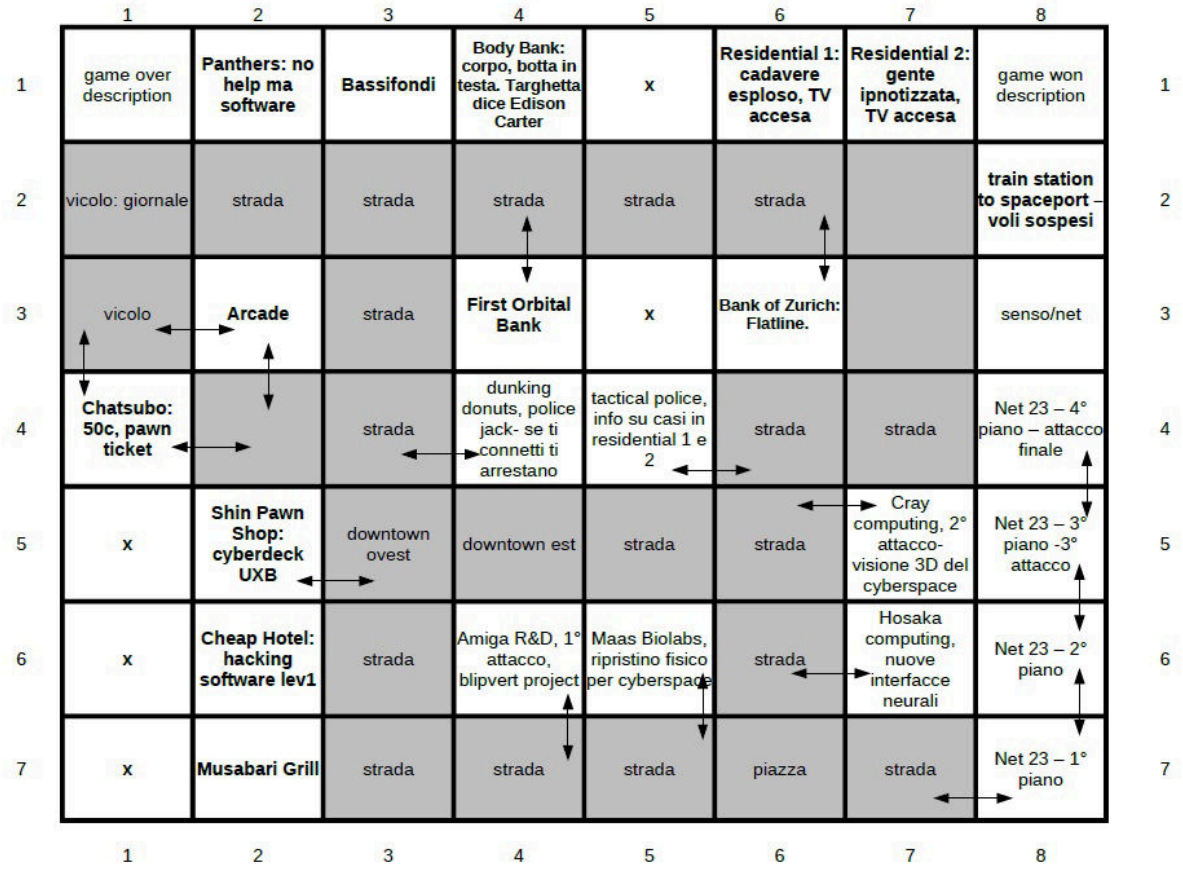
At this point, however, it is necessary to make some assumptions about how we want the world of play to be represented to the user and how we want him/her to interact with it. Again, I chose to use a multidimensional matrix, consisting of as many "columns" and "rows" as the total game locations (also counting those inaccessible to the user since they are used for other purposes). Below is an example with its explanation.

```
Situation.1.2='Panthers cove. '
Situation.2.2= 'Panthers are a group of
crackers living on the margins of society''.'
Situation.3.2='They are strongly politicized
and their purpose is, as often happens,'
Situation.4.2='to put the system in crisis.
Though you do not share the same ideals, '
Situation.5.2= 'they respect your ethics.
You will not get direct help, except in the
form of software.'
Situation.6.2= 'A war DIALER is currently
available.'
Situation.7.2= 'NORTH EAST WEST UP DOWN'
```

As you can see when comparing the map, the one reported above is the description for location Number 2.

In the game, each room is described by a 7-line text. When necessary, the penultimate line describes a creature or object. After killing the creature, or removing the object, this line must be "reset" (with a simple Situation.6.2='') to avoid giving false information.

The last line lists the directions that are not allowed for navigating rooms. After defining the variables "x" and "y" (current position) and the variables "maxrow" and "maxcol" (limits of the game world matrix), the description of the current location will be found in our matrix at the



	1	2	3	4	5	6	7	8	
1	game over description	**Panthers: no help ma software**	**Bassifondi**	**Body Bank: corpo, botta in testa. Targhetta dice Edison Carter**	**x**	**Residential 1: cadavere esploso, TV accesa**	**Residential 2: gente ipnotizzata, TV accesa**	game won description	1
2	vicolo: giornale	strada	strada	strada	strada	strada		**train station to spaceport – voli sospesi**	2
3	vicolo	**Arcade**	strada	**First Orbital Bank**	**x**	**Bank of Zurich: Flatline.**		senso/net	3
4	**Chatsubo: 50c, pawn ticket**		strada	dunking donuts, police jack- se ti connetti ti arrestano	tactical police, info su casi in residential 1 e 2	strada	strada	Net 23 – 4° piano – attacco finale	4
5	**x**	**Shin Pawn Shop: cyberdeck UXB**	downtown ovest	downtown est	strada	strada	Cray computing, 2° attacco- visione 3D del cyberspace	Net 23 – 3° piano -3° attacco	5
6	**x**	**Cheap Hotel: hacking software lev1**	strada	Amiga R&D, 1° attacco, blipvert project	Maas Biolabs, ripristino fisico per cyberspace	strada	Hosaka computing, nuove interfacce neurali	Net 23 – 2° piano	6
7	**x**	**Musabari Grill**	strada	strada	strada	piazza	strada	Net 23 – 1° piano	7
	1	2	3	4	5	6	7	8	

Fig. 1

address "Pos=(x-1)*maxcol+y". Consequently, "Situation.5 1.Pos "will contain, in the example described above, the"title" of our room (Panthers Cove).

We have therefore solved the first problem (representation of the gaming world) and its implementation in ARexx code is as follows:

```
/*------------------------------*/
/* Print location and situation */
/*------------------------------*/
Scenario:
say
do i=1 to 6
if Situation.I.Pos~='' then say
Situation.I.Pos
end
return
```

The above-described procedure simply displays the 6 lines of text describing the location on the screen (unless one of them is empty, for example because we have already picked up the object). The seventh line, which tells the code which directions are not allowed for navigation, will be explained in the following paragraph.

2. NAVIGATION OF THE GAMING WORLD

Having understood how we can create a game world and how to make it usable to the player, let's now see how to move within it.

In the previous article, published in issue 4, we saw how to implement a simple parser.
This parser was created precisely for this game and it works with a simple grammar based on the historical form "verb+object" typical of the very first text adventures. Inserted a phrase like "go north", the parser "breaks" the phrase in its components and, after checking that "go" is actually a word contained in its vocabulary, calls the corresponding routine.

This routine may take the form of:

```
/*-------------------------*/
/*          GO             */
/*-------------------------*/
Go:
if find(Situation.5 7.Pos,name)>0 then say
'you can''t go  '|| name
else do
select
when name='NORTH' then x=x-1
```

```
when name='SOUTH'  then x=x+1
when name='EAST'  then y=y+1
when name='WEST' then y=y-1
when name='UP' then x=x-1
when name='DOWN' then x=x+1
otherwise say 'I don't understand. Try again';
say; return
end
Pos=(x-1)*maxcol+y
do i=1 to 6
if Situation.I.Pos~='' then say Situation.I.Pos
end
end
return
```

At this point, the smartest readers will have already understood what the code does and how the notorious line 7 works:

- first, we check if the chosen direction is actually allowed: for example, if we want to go east while we are in the last rightmost column, this will not be possible ('You can't go to'). The "find" statement searches for a substring within a string: if the substring is found in line 7 of the current location, that direction is forbidden for navigation;
- if the desired direction is among those allowed, the variables "x" or "y" will be modified depending on the direction, actually moving the player to the new location;
- finally, after transforming the matrix coordinates into a scalar position, the description of the new position will be displayed on the screen.

As the last part of this article, let's now see how you can interact with the game world. The "navigation" routine is clearly an important part, but without something more complex there will never be a game, even if it was relegated to the rank of a simple "walking simulator". How many and what actions our parser will need to be able to understand depend on how we want to move our story forward, but as a minimum we will need to be able to process the following requests: "go", "take", "use", "examine" (to get more information about an object), "look" (to request that the description of the room be displayed again), "list" (or "inventory", to check our equipment) and of course "vocabulary" to give the user an idea of what they can do. In addition, our code may contain other undocumented instructions, usually used by programmers and testers when developing the software.

It is impossible, in fact, to demand that these figures restart the game from scratch only to test new features or to eliminate a bug noticed at an advanced stage. In the event that these features are released to the public once the development is completed, we are dealing with the so-called "cheat codes".

In the case of our game, "debug" recalls a routine used to move quickly from any location to any other without having to traverse the entire map, while "var" displays the status of the game variables on the screen to check that no wrong assignments are present.

3. INTERACTION WITH THE WORLD OF GAME

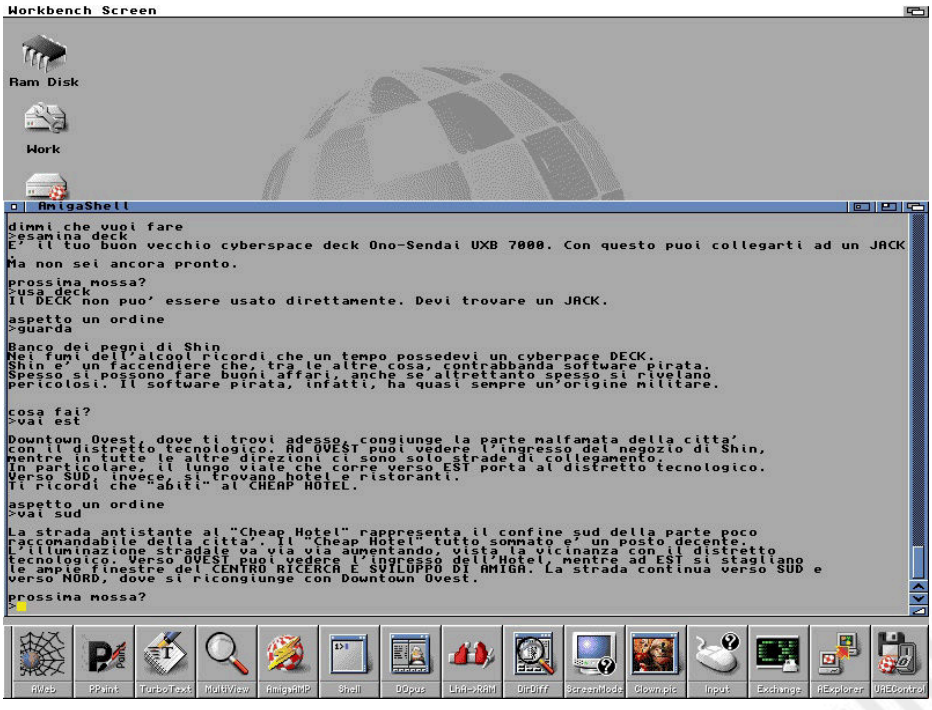


Fig. 2

Not all actions need arguments (ex: "list" or "look") and this is why the following parser is divided into two subparts: if the "phrase" contains only one word it is directly assigned to the variable "verb" (provided it is a valid command) and the variable "name" (our object) is reset, otherwise the phrase is broken down into its elements.

It is important to note that in a game like this it might not be necessary to systematically "reset" the variable "name". After all, if after the "LOOK" command (which does not need to be followed by a name) the "GO" command is invoked (for which the "name" variable will contain the

desired direction), there would be no conflict within the parser. In general, however, remember to always keep an active control over your entire variable system: as the complexity of the code increases, it may be extremely difficult to understand where a possible "bug" is generated and, consequently, to correct it.

```
/*-------------------------*/
/* Syntax parser */
/*-------------------------*/
Parser:
if words =1 then
select
when phrase='LOOK' then do
verb='LOOK'; name=''
end
when phrase='LIST' then do
verb='LIST'; name=''
end
when phrase='QUIT' then do
verb='QUIT'; name=''
end
when phrase='VOC' then do
verb='VOC'; name=''
end
when phrase='DEBUG' then do
verb='DEBUG'; name=''
end
otherwise say"I don't understand. try again"
end
if words =2 then do
verb=left (phrase, index(phrase, ' ')-1)
name=right(sentence,length (sentence)-
index(sentence,' '))
end
return
```

At this point, the "trick" is almost done. The next step is to invoke the "main code" and invoke, depending on the situation, the corresponding routine:

```
select
when verb='GO' then call Go(name, Pos)
when verb='LOOK' then call Look()
when verb='TAKE' then call Take(name)
when verb='EXAMINE' then call Examine (name)
when verb='USE' then call Use(name)
when verb='LIST' then call List()
when verb='VOC' then call Vocabulary()
when verb='DEBUG' then call Debug()
when verb='VAR' then call Var (name)
when verb='QUIT' then nop
otherwise say 'I don't know what it means '
|| verb
end
```

At this point, our job is practically complete. What is missing is to define how the management of events and objects within the game advances our story plot. In the next issue, we will analyze with a greater level of detail what we still need in order to finally enjoy our text adventure.

4. CONCLUSIONS

Nowadays there are many frameworks specifically designed to develop text adventures and using them is essential for the developer who needs to focus on the story plot rather than "wasting time" writing the code that manages it, especially when the intention is to publish on many different platforms, be they "retro" or "next-gen". For programmers who prefer to write their own code though, having an idea of how to implement a game world and finally be able to achieve it is, however, priceless.
In the next issue, we will continue to explore ARexx to discover how to write routines that will actually allow us to interact in a complex way to advance our story and discover every secret.

BIBLIOGRAPHY

- Mike Cowlishaw "The REXX Language: A Practical Approach to Programming" (1985) Prentice Hall. ISBN 0-13-780651-5.
- Chris Zamara, Nick Sullivan "Using Arexx on the Amiga" (1991) Abacus Software Inc. ISBN 1-55755-114-6.
- AmigaOS 2.0 System Manual
- http://it.wikipedia.org/wiki/Game_engine
- http://it.wikipedia.org/wiki/Porting

How to use the examples

As reported in the article on num. 20, to use the examples you must save the script in text mode in the format "script_name.rexx".
To launch the script just type from shell:
">rx script_name.rexx"
or simply:
">rx script_name".

JAPAN HISTORY
Oh no! More G&W!

by Michele Ugolini

"Oh no! More Lemmings!": that was the Psygnosis title that we used to play in the '90s. "Oh no! More G&W!" says the title of this article since these electronic wonders are returning to the market today as the famous little creatures.

The original Nintendo G&W was finally released for the 35th anniversary of the great "N" company. Have you already showcased it? How many minutes did you use it? Problems with your eyesight, or are you okay?

Regardless of these provocative questions, it remains a sacred object for us collectors. The feel of rubber keys is identical to the original G&W keys of 40 years ago.
The display, the design, the graphic arrangement, are all simply adorable elements. Unlike the old pill batteries, this time the battery is inserted internally, collecting appreciation and mood from the web: when the battery is exhausted , unfortunately, it will be extracted to avoid swelling or dispersion of the contents.

More bad news from the web?
The price has been heavily criticized, moreover, which is no small feat, in this version of the G&W there is no stand to keep it vertical in our showcase.
A cardboard stand was instead present in the remake of the 2010 G&W "Ball" that I had reviewed a few episodes ago, a cheap but robust paper product, functional and above all marked "G&W". Too bad, we'll have to buy an anonymous booth from the web!

Finally, the last trend coming from the Internet: the alarm clock. In fact, there is a clock with several wallpapers, but there is no alarm clock that can get us out of bed in the morning, perhaps with Mario's ringtone.
What a pity.

What are the benefits? Yes, of course many: we are playing an official Nintendo product and above all from the web have already come numerous tutorials, which obviously both RMW and I strongly advise against looking/viewing/applying, through which you can modify the start of Mario and run Doom.
More programs and games may be included in the future, but the sanctity of an object officially released by Nintendo will disappear!
Be aware, however, that many reckless people have already succeeded in reprogramming.
Doom is an epic title but the graphics of the 90s, combined with such a small display, generates several visual problems: I challenge anyone not to complain because of visual disturbances after half an hour of gameplay.
On the other hand, the taste of running various games/programs in this G&W, denotes the high talent of our most reckless friends. (see figure 1)

As we know, Nintendo is famous for generating excitement in the companies of its rivals. Rivals not in the true sense of the word since marketing in Japan only creates other marketing and each company enters the revenue through its own intelligent slice of "subfolder-style" services.

Fig. 1 - Doom on Nintendo Game & Watch

And here's the news from a few days ago on the web: Capcom will launch "Retro Station" on the market, through Amazon Japan.
It will be a mini cabin, with a very particular vintage style.
Personally, I find the design very provocative, in the sense that I still can't decide if I will love it or not digest it! The only certainty is that it will not go unnoticed.

In Osaka nobody jokes, Capcom will insert ten games, we can play Street Fighter (II, II Champion Edition, Super Street Fighter II, Super Street Fighter II Turbo, Super Puzzle Fighter II Turbo) and Mega man (The Power Battle, 2 The Power Fighters, X, Soccer, Man & Bass).
The dimensions are: 329 mm x 280 mm x 315 mm. The weight will be relevant: 2.1 kg. (see Figure 2)
We will find a display and stereo speakers positioned in front, from the web it is intuited that the audio will enjoy excellent quality.
In Japan it will be possible to buy it for the price of 21,780 yen, or about 176 of our euros. We will have to wait for the Western import which will be subject to the various customs surcharges.

This time the price is high and the object is very particular, will our heroes, from Capcom, be able to keep up with the skillful strategies of the Nintendo market?
It is not a useless question, in fact, we must remember the limited success of the mini console Capcom Arcade, launched in April 2019: the units sold did not arouse particular moods at Nintendo, the game this time is open and the variables are reset, only the future can tell us the past of this "nostalgia operation"... still perpetrated by the Japanese giants today.

Finally, do we want to talk about the Gig Tigers that have finally arrived on the Italian market?
Three versions currently highly appreciated: Marvel X-Men ProjectX , Sonic the hedgehog 3, Transformers generation 2. The prices recorded on the web have already begun to rise and fall. We have faced an anomalous Black friday, a numb Cyber monday and a very different Christmas awaits us than usual, unfortunately society had to interface with an insidious and silent enemy. Market strategies have also had to face new variables, some unsuspectingly fruitful, others incredibly bankrupt.

Sales of Tiger GIGS, it seems, are going well, on the website camelcamelcamel.it you can keep track of price curves.

https://en.camelcamelcamel.com/

Unexpected news recently came from Sega, the same Sega that allowed Hasbro to produce A GIG Tiger in honor of our beloved Sonic.
The sad news is that, although the pandemic has brought golden business to the gaming world, the ban on assemblies has wiped out the "arcade" sector.
Sega Sammy Holdings, a very famous sector in Japan for caring for the arcade, arcade and games room department, said it had suffered a significant decline in turnover.
Shukan Famitsū is the most important and respected

Fig. 2 - Capcom Retro Station

video game magazine in Japan, it mainly contains reviews and news from the world of video games and recently published the painful news of Sega: the sale of 85% of the shares to GENDA INC.

We all hope to be able to play in the future the magnificent Japanese cabins inside Tokyo, still branded Sega, but Akihabara's famous Sega building will disappear from the crowded street of electric town.
Will Laox be happy?
Or maybe Bic Camera?
I don't think so.
I think the building behind Sega, which houses Super Potato, will also be very sad.
Who knows what happens to the Akihabara Building Sega and the nearby VR area Sega?
(see Figure 3)

Japan is famous for reinventing itself and for being reborn from its problems and defeats. It is not at all unlikely that a competitor company (not an enemy, but only positioned in a congruent slice within the same marketing sector) will reach out to 15% of Sega.
Or will Microsoft reach out? If Microsoft decides to join Sega, after courageous marriages with Bethesda, Ninja Theory, Playground Games, Obsidian, it would only mean that the "PlayStation monopoly" should review the entire Sony home organization and operate more surgical actions in its immediate future!
In the meantime, I attach a link for a pleasant reading about the future video game world:

https://www.everyeye.it/notizie/microsoft-saw-marriages-ha-fare-new clue-greenberg-483922.html

Last indiscretion: in 2021 there will be important news regarding Sonic.
I repeat it from several RMW issues: the G&W world is not the end nor the world of retrogaming.
It is obvious that even Sega is transforming, the epidemic will allow the Japanese ability to invent something genius, to fuel a completely new and revamped post-pandemic marketing.
Aren't the same streets as the cities a year away?
We will never find shops, signs, directions and buildings identical to the previous year in which we visited Japan.

That's it, that's all. Stay tuned, a powerful rebirth is at the gates of Japan: a bursting return to feed us faithful consumers and collectors.

See you soon, all the best wishes for Happy Holidays!

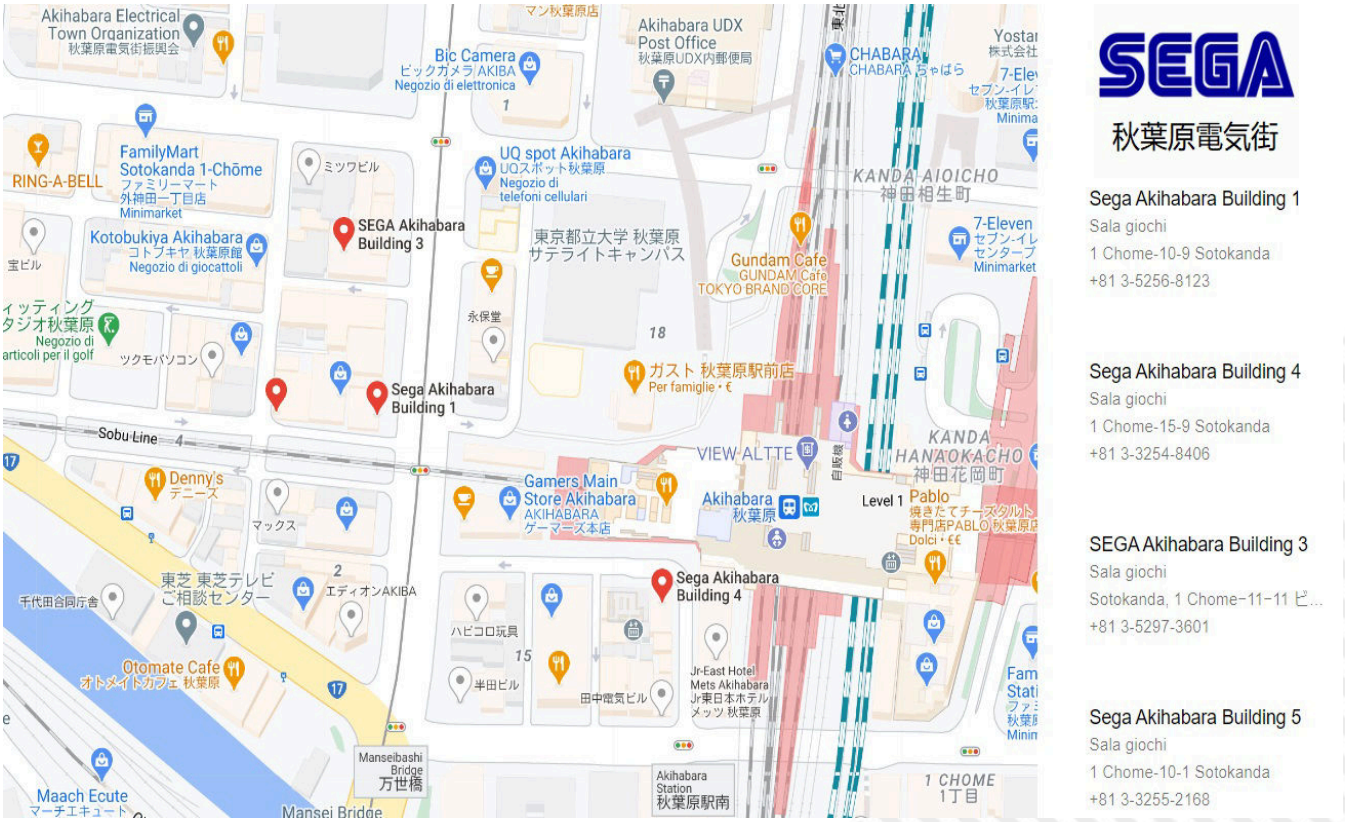


Fig. 3 - Tokyo, Akihabara district

Exclusive interview with Randall Flagg

Phreaker/cracker/hacker of the international DOS/Windows scene in the 90s and 2000s

by David La Monaca

1 - Intro

Who remembers The Day Of The Tentacle? Surely, many of our readers will have a good memory of it. For the very few who do not remember, it was a point & click adventure by LucasFilm Games, also known as Maniac Mansion II, based on the equally well-known SCUMM development engine.

What is not as well known to the public and to many retrogamers, is a little story about DOTT's crack, which we want to tell you below, a story that speaks about the competition between cracking scene groups of the 90s. The competition was all about fame, respectability and technical skills of the several members of the scene. Those who first managed to remove or overcome a video game protection gained notoriety and respect, in addition to the right to "defame" or mock other groups in .NFO text files that came with the cracks.

In DOTT's case, the story went something like this: it seems that Hybrid's people didn't have the necessary skills, so at some point they gave up. Even the Swedes couldn't crack the game. They even came to declare it "uncrackable", perhaps to buy some time on the other groups, discouraging them from participating in the competition and still trying to be the first, continuing to work hard on the crack. But this was one of the tricks well known to everyone on the scene and unanimously considered unfair.

So, at 9:40 on June 12, 1993, a member of Hybrid wrote this message: "Okay guys, here's the first disk of DOTT! So stop bitching. Yeah, that's right, it's a pain in the ass, so now we've all given it a fair chance to try, I think! There's probably more copy protection after that SPECIAL BATTERY shit, so try to figure something out! - Hoson/Hybrid 1993 (pissed, tired, cool, heavy, death metallist). P.S. At least we're men enough to admit our own defeats! P.S.S. We're still working on it!"

Fig. 1 - Randall Flagg aka Antonio Mazzanti

Meanwhile the team Razor 1911, European division (of course!), at 7:33 on the same day (June 12), CET, received the following note, written by a cracker acting under the nickname Randall Flagg: "Day Of The Tentacle - Maniac Mansion II - Cracked by Randall Flagg! Note: --- for public use--- '

Yeah, that's pretty clear, so at least it looks that way. But guess what? That's right, the race wasn't over yet! Other scene fellows, from the legendary group The Dream Team, and more precisely Dr. Detergent/TDT [R.I.P. :- (], seem to have already solved the game at 13:48 on June 11. Their NFO file reads, "We were focused on the game and we knew we would make it, but not WHEN! Finally now, your favourite cracking team has made it again! And a nice clean CRACK came out! TDT has nothing to do with charity and posting a crack patch would just be like giving the release to another group! After installing the game, be sure to copy the DOTTCRK.COM file and launch it EACH TIME before you start playing! As you read from the NFO from HYBRID, they told everyone it was NOT cracked so that means everyone could crack the game, but TDT came first!"

So the question here is WHERE the game was cracked (considering only Razor 1911 and TDT), discover the correct time zone and determine WHO came first to the hit!

To understand the level of competition between the different groups, let's take a look at the file .NFO from another team. This time it was the GENESIS*PROJECT, a part of the legendary group that operated on the C64 scene, which proudly continued to occupy a prominent place on the scene. On June 12, 1993, at 7:03 pm, Snacky/G*P wrote: "Well, we certainly had some problems inside GENESIS and also with some lamers who claimed to be members of G*P, but now I'm really proud to introduce you to a crack-patch for this fantastic game... It took me a few hours to develop it and I hope you all like it! Hey, you guys from TDT, what's going on with you...? Nice release eehhhh, another #6 disk release to spread around a crack-patch that DOESN't work... hahahah! Hey, you guys from Hybrid, what's on... Before you give up and say this game CAN't be cracked, next time wait a few more days and there you go! Be sure to grab the new trainer programs and releases

from your favorite 1993 entertainment group!"

So, even though the time of the messages reveals everything, this seemed to be the ULTIMATE crack-patch, not just any bugged release. Knowing Snacky's skills, no one doubted it, but comparing the time of the patch file, Randall Flagg/ Razor1911 was the first to reach the finish line, followed by the others. But don't forget that the crackers were also thoroughly testing the game, thus controlling the protection. After all, it was a game that used a fairly sophisticated protection that was activated during the course of the game, proceeding along the plot. This important fact seems to go beyond the issue of the release time to determine who really came first to the COMPLETE crack of the game.

So we could conclude that G*P had fully won the race. But there is a problem: in the crack file of G*P called GENESIS.GP (the crack consists of two small files, DOTTCRGP.COM and GENESIS.GP) you can clearly read: "RAZOR/Maniac Mansion II". Maybe it doesn't mean anything, does it? The only one who could really help us settle the matter would be Snacky himself, a cracker from the C64 scene who was really skilled and respectable, one who never stole a crack: it would have taken much longer than cracking a game all by himself. Unfortunately, we don't have a direct testimony from Snacky, but we found a comment from Randall Flagg dated 2016 on a thread dedicated to this crack: "Hey... I can only say it was a lot of fun to crack!"

Some claim that Snacky was a member of both G*P and Razor at the time... But the last word on the whole story is provided by Randall Flagg himself: "I can only add that in my crack file I also changed the text in which Dr. Fred triggers the protection by asking for the ingredients of the super battery (the ASCII text was masked via XOR as for other single-byte LucasFilm games, perhaps 0x69h if I remember correctly). Crack would completely erase that scene with Dr. Fred, letting the player through without requiring anything. But it also did so in case the player later looked at the battery plans. Nothing that complex. If I'm not mistaken, two or three other groups announced their crack, even 2-3 days later, and then they were wiped out virtually everywhere. The RAZOR version is the only one that can still be found around, even on torrent files. :)".

Conclusion: Randall Flagg's crack was the first to appear chronologically and was also ALREADY COMPLETE!

So let's go meet the author of this and countless other DOS/ Windows game cracks from the 1990s and 2000s. Italian readers of RMW will be proud that one of the most authoritative and skilled crackers in the whole cracking scene was their fellow countryman Randall Flagg, a.k.a. Antonio Mazzanti.

2 - .NFO

DLM – Hi Antonio and thank you for accepting our invitation for an interview. As we tried to explain in the introduction, you are a little-known character outside a certain circle of techies, hackers and IT enthusiasts. Yet many of those who are now reading this interview owe you the chance to have been able to play those PC DOS and Windows titles of the 90s that had a particularly articulate and difficult protection to overcome. In addition to the aforementioned DOTT, just think of the series of graphic adventures by LucasFilm "Monkey Island" and "Indiana Jones". So we ask you to briefly introduce yourself and give us some news about yourself.

Fig. 2 - Antonio in 1993 on the phone (note the trusty US Robotics HST modem on the sideboard, the chaotic PC workstation, and the drumsticks)

Thank you for the invitation. I'm not a guy who loves to talk or write about himself. When asked to tell about myself or my experiences in a formal way I often decline. I prefer a friendly chat, like you and I are doing right now, albeit in two different cities. I was born in Florence in 1972, I had a good childhood and I had a very strong and intelligent mother. All normal with primary school, then in high school I chose Accounting for Programmers, a choice more dictated by the desire to find a way to learn how to use computers in school. Obviously I've learned something, but to be honest, I have to admit, I've basically done everything myself. Driven above all by my innate curiosity, I always wanted to know what was behind the games and software I saw running on the computers I gradually owned. I have always been (and still am) interested in both hardware and electronics, as well as software and programming. After high school, I also booked myself in college, but I never finished it, I always felt myself too far ahead, eheheh.

DLM – Have you always been a computer enthusiast since you were a child? What attracted you to the magical digital world?

I can't say I'm a "fan" even today. As a kid, no electronics or computers until a ZX Spectrum came into the house, my first computer, bought by my far-sighted mom. Obviously I immediately looked for games and then slowly started experimenting with programs in BASIC. My first test was a

program to print the multiplication tables. The FOR cycle was enlightening to me and from then on I realized that that world was for me. I've only been interested and working with electronics for about ten years now and I've been reversing practically forever. At first I was attracted to intros that appeared on pirated copies of C64 diskettes.

DLM – Tell us more about your first computer? Do you still own it or have you taken many more over time?

My mother had a clear vision of the future of her children and for me she had thought about English and computer science. So as a boy I had a long study stay in England that later was so important for all my digital adventures and my professional life. One day a 16KB ZX Spectrum, complete with Philips cassette recorder, arrived home thanks my mom. I don't have it anymore, but in time, I practically bought back all the retro-computers that could arouse a little nostalgia in me, eheh.

DLM – When did you start getting interested behind the scenes in programs and games? That is, when did you start discovering machine language and assembly?

When Maniac Mansion was released for C64, I became passionate about it but, proceeding in the adventure at some point my copy didn't work. I was stuck and I couldn't see what was going on in the rest of the game. So I took the same game on PC DOS version. I was about 15 years old and discovered that through a simple debugger I could see what a program or game loaded into memory was doing. In high school I had experience with COBOL and its runtime module. I had learned the difference between interpreted and executable programs. So, using Turbo Debugger, I was able to find the JMP statement that I needed to bypass the protection. I remember that moment was particularly exciting. From then on I was also interested in the other games by Lucasfilm, including The Secret of Monkey Island and The Day Of The Tentacle, but by that time I had already switched to a more sophisticated and complete debugging tool like Soft-ICE.

My first real crack was for Motocross, a DOS game, for which, I remember, it was enough to strategically place an unconditional jump (basically only one byte). DOTT's crack, on the other hand, was among the most complex to deal with. Among other things, the protection was of two types: one at the beginning and the other during one of the phases of the adventure. Surely it was not enough to use some well-settled jump opcodes, DOTT made use of a stack machine to be circumvented with a series of push/pop and comparison opcodes. Not surprisingly, only a few groups tried to find crack and I was the first to post it around.

DLM – At the time there was no Internet, at most some very slow modem connection to BBSes. How did you actually learn to juggle programming?

I'm a complete self-taught. At the age of 14 I collected my savings and went to the University Library in Via San Gallo in Florence and ordered two books on the assembly 8086/8088 and 80386, obviously both in English. They were my gateway to low-level programming and understanding the protection routines I encountered on DOS games and programs.

DLM – In addition to the world of 8-bit and 16-bit computers, we know you've always been attracted to the world of telecommunications. When was this passion born that pushed you to browse telephone networks?

After joining the RAZOR 1911, I started exploring the world of telematics and the goal was to be able to make free calls everywhere so that I could communicate quickly with the other members of the group and to connect to the BBSes around the world. It was a question about racking your brain and squeezing your heads to discover the most unthinkable and creative methods of attacking phone systems and networks. Of course, I can't go into too much detail. ;-)

DLM – At some point you came into the cracking scene by joining groups on the Italian and international scene. Among others let's remember Dead Memory, Razor 1911, Eclipse and Hybrid. Names that remind many of us of carefree afternoons spent playing crazy titles and sometimes real masterpieces. How did you get into this universe, make it a little bit your way of life?

In the early 1990s I worked for a computer shop in my city and at SMAU in Milan I met some guys who were part of the Dead Memory who introduced me to the Italian cracking scene (basically the few productions of local games concerned Simulmondo titles and two or three more software houses). They were looking for crackers, so I started cracking games

Fig. 3 - In the foreground Sector 9, the founder of Razor 1911 at dinner with our Randall Flagg (left in the background)

in Italian for them. Some of these titles such as 3D World Tennis, released in November 1992, could also be interesting for other foreign groups. So I started accessing BBSes and exploring that new world by uploading the cracks of Italian games. One day I uploaded the “Indiana Jones and the Fate of Atlantis” adventure crack. I was noticed by the leader of the Razor (a.k.a. The Renegade Chemist) and, since Lucasfilm games were considered very complex to unprotect, he contacted me and offered to join their team.

DLM – Can you tell us about the atmosphere breathed in those days, when you were splitting between phreaking and cracking? How were you technically and logistically organized?

The organization was basically very simple: when a game needed a crack, the supplier (the one who provided the original) called through bluebox, credit card or other methods the first available cracker, who proceeded to download the game directly from the supplier. Then he tried to crack the title and if he could, he would write a file .NFO (a text file) with notes needed for distribution. The supplier then phoned the so-called "couriers" (distributors) and the cracked game quickly ended up on all our BBSes. From these, other freelance couriers uploaded it on the BBS of other groups. The HQs (Head Quarters) of the other groups had priority, so they could put the "seal" on the release. In all the steps he communicated mainly by phone or with messages on the HQ.

DLM – Going back to your training, how was your university experience? Were you able to study, work and engage in phreaking/cracking activities?

Well, yes, after graduation I attended university for three years, first in Pisa and then in Florence, but honestly the courses were a little tight. I thought they were talking about a past world. Everything I learned, I learned in the field, experimenting and staying in touch with other crackers of the scene, reading books and consulting technical documentation found on the BBS and then on the Internet.

DLM – Our magazine mainly deals with retrocomputing and retrogaming. Do you remember the first videogame you played? Do you still like using the physical hardware of the past?

I don't remember exactly the name of the first game I loaded and played on my home computer. It must have been a game for the Sinclair ZX Spectrum. I remember a Pong clone that my mother bought. Today I often use Amiga (the 1000, 500, 600, 2000 and 1200 models are all in my collection) and also many of the 8 and 16 bits of the 1980s. And even today I enjoy cracking MS-DOS and Windows games using real hardware (a 386SX PC and a 200 MHz Pentium).

DLM – We also know that you do electronics projects by

Fig. 4 - OUAS's De Gregorio (left) and Mazzanti (right) in a 2018 photo with a well-known Lucasfilm Games programmer/designer

profession and also do repairs of old machines for friends and enthusiasts. After so many years, do you still enjoy getting your hands on cards and motherboards from so many years ago?

Sure, sure. The basic idea stems from the desire to pay tribute to those who designed and built those machines! Seeing them still at work today gives you the creeps! I often repair the old home computers that I find around and then give them to friends and acquaintances. Or I fix those of friends who ask me for help.

DLM – Can you tell us a particularly compelling anecdote among all those that you surely have in your memory concerning phreaking activities (if you can tell)?

Eheheh... that's a question that brings back a thousand memories and a thousand stories in my memory. Stories too long to tell here, but I do not exclude taking some to the next OUAS meeting in Rome, as happened at the end of October at the OUAS 2019 in Milan.

DLM – What about any stories about your cracking business?

An interesting story to tell would be about a well-known programmer/designer and screenwriter of Lucasfilm Games, who during a meeting a few years ago forgave me for cracking his games (see photo of the message that he sent to me). For other episodes I invite everyone to watch my speech at the latest OUAS 2019, recently published on Youtube [R5]. I’m pretty sure you'll enjoy it! (Warning: the speech on YT is in Italian, probably you can activate English subtitles.)

DLM – How did you choose the titles to crack? Was it your free choice, or was it just what the suppliers provided?

Well, the organization of the groups was actually pretty strict. It all depended on a clear hierarchy. The cracker with the best reputation in the group was consulted first, then the others were called. But it often also depended on the name of the software house or publisher of the game to crack. A new game by Lucasfilm or Sierra always went to the best cracker in the group. A matter of prestige and respect.

DLM – Have you ever participated in coding an intro or

keygen of a game or program cracked by your group?

I worked on several keygens, for example the one for the 1:1 game via LAN that controlled the serial numbers of the games connected to each other. As for intros, I don't remember ever writing a single line of code. Sometimes I still write keygens today, but now only for fun and a bit of academia. And obviously I don't disclose them.

DLM – When faced with protections, what was the strongest motivation you felt in taking up the challenge?

The goal was always to be the number one, the best, the most skilled on the scene and bring prestige to the group I was part of. Because when you reached certain levels, you automatically got respect for others and you could even afford to mock other groups, send them ironic greetings on the files .NFO and even in some cases fuck with the so-called "lamer crackers".

DLM – Do you remember which software tools you used at the beginning, when you took the first steps in cracking? How have they changed over time and what are the current and modern ones?

I used Borland's Turbo Debugger at the beginning. Then I switched to Soft-ICE for DOS/WIN, then Sourcer and CodeView. Nor did I disdain the use of the simple debugging tool included in the DOS. Then I learned to compile with the TASM (Turbo Assembler) and as an editor I often used HexView. Nowadays, I use IDA/GHIRA as decompilers and OllyDBG and X64Debug as debuggers.

DLM – Still remaining in the ranks of legality, do you have any stories to tell us about some purely hacking venture in which you participated?

There are several of them. One of the most interesting is the System75 control units, telephone devices in use in the United States that contained 1200/2400 bps modems. These electronic devices could be connected and hacked (sometimes even with default factory passwords). If the hacking was successful, you could even create new phone numbers to use at will. You could also create PBXs to use to call other numbers (typically used to make long distance calls). In practice, you could make local calls (free of charge) and then bounce off PBX to call domestic or foreign numbers. The funny and really cool thing was that you could also call the 700 numbers to create real conference calls with reserved access (through a PIN code). In the end, this is how free audio conferencing was organised to coordinate us within the group or to call anywhere for free. The quality of the calls was often very good and we also used System75 to access BBSs around the world.

DLM – As mentioned in the introduction, you occasionally attend events to bring and preserve your experience in the phreaking/cracking/hacking scene. We at RMW were able to appreciate your speech last October in Milan, especially for the curiosities and the juicy anecdotes you told. The previous year, at OUAS 2018 held in Rome, you even held a workshop open to all in which you gave more technical information and challenged those involved to crack a specific game. How did you come up with this idea and how did it go?

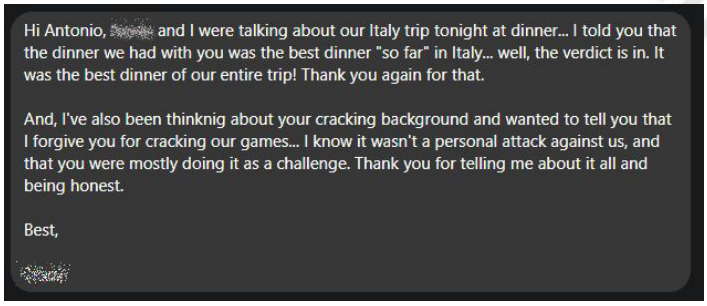


Fig. 5 - The message of forgiveness received by a Lucasfilm designer

I was asked to participate in OUAS 2017 to perform a classic speech but I proposed a real live cracking session. In practice, I showed cracking the protection of a game "live" (in detail it was Eric Zmiro's Prehistorik 2). On that occasion I armed myself with an old 386 laptop and performing everything live, including the problems of interfacing the VGA port with the screen and projector, I went through an operational analysis and the implementation of the crack patch.

We repeated the experience at the 2018 event, in which the cracking session turned into a real workshop attended by a good number of participants. The workshop sessions were about Zak McKracken's and Maniac Mansion's cracks while the actual exercise for the participants was about building the crack of “Loom”, simply using an appropriate debugger. I believe that the 2018 workshop was a unique event in Italy and perhaps also in Europe. The practical implications and technical elements touched upon during the sessions by the many participants, judging from the feedback received, proved to be very interesting and extremely educational.

DLM – Back to hard and pure cracking. Has there ever been a protection you really couldn't get over? And the one that, after you got over it, really impressed you with the brilliance of who invented it?

No, I don't remember ever running into anything ‘uncrackable’. Sometimes it took more time to overcome the protection built by programmers or real security experts hired by software houses, but I've always figured it out. :-)

Among the most difficult protections to get over I must mention those derived from the Protection Kit by Eric Zmiro, author of several PC/DOS games (including Prehistorik 2). Zmiro's protection was undoubtedly the best system I've ever seen. I remember he gave me a lot of trouble at the time and I wrote a complete unwrapper to try to get over it and there are still some parts of his kit that have remained

unsolved. I was in touch with Eric a long time ago, and even though he's always stuck to his hatred of all the crackers in the world, he even sent me an original copy of his software.

DLM – Do you still sit on your PC and try to crack modern games? Or to re-crack old titles, complete them or make them even easier for players?

Yes, I often do, although there is always little time to devote to these activities. I still receive many requests from friends on Facebook or from various MS-DOS groups. Do I crack modern games on PCs or other platforms or consoles? No, thank you, I categorically decline.

DLM – Over time you have certainly accumulated an impressive amount of information, techniques, know-how, etc. Have you ever thought that this whole mine of information should be preserved in some way? Without getting caught up in the stitches of the law (although at least in Italy you should enjoy some article related to prescription...), is there any way in your opinion to preserve all this information?

In due proportion, I would compare my skill and accumulated techniques over time to those of a surgeon or a craftsman. It is difficult if not impossible to describe the skills and techniques that are put in place to overcome or remove the protection of a game or program. It's even harder to write down things like intuition, creativity, or observation skills when using a hexadecimal editor or reading assembly code from a debugger. If this were possible then one might think of digitizing human imagination or creativity, which I find rather complicated at the moment. :-)

It was interesting to hold a workshop at OUAS 2018 in Rome. In that context, some basic principles and methods can be shown. Other information is difficult to transmit except with frequent attendance or direct contact with those who aspire to learn. Hacking is not exactly a subject of study. Yes, techniques can be classified, but for that any of the volumes dedicated to these activities is fine.

DLM - What do you think are the must-have references on the Net and not, that every good cracker or phreaker should consult or keep in their personal collection?

The Net is full of texts and books that talk about hacking and in some cases also cracking and phreaking. They are complex and articulated subjects, in reality, impossible in my opinion to enclose and box in books and tutorials. I always advise those who want to get closer to this world to get the book "Programming the Intel 80386" [R6] and start reading.

3 - Outro

DLM - One final question: why did you choose "Randall Flagg" as your battle name in the cracking scene?

I've always been a passionate reader of Stephen King's books. At the time "The Shadow of the Scorpio" struck me a lot and one of its darkest, wicked and ruthless characters, then turned into a synonymous of absolute evil in many subsequent books, was just...

There would still be a lot to ask, a lot to remember and perhaps a fixed address book on RMW full of anecdotes and curiosities with all the stories that R.F. could still tell us would not be enough. But for the moment we stop here, warmly thanking Antonio Mazzanti for his openness and sympathy. It was really fun chatting and remembering with him the old days of the '90s cracking scene. Don't forget to watch the video of Antonio's intervention at the latest Once Upon A Sprite 2019 [R5]. It's definitely worth it.

References

[R1] RAZOR 1911 official website - https://razor1911.com/
[R2] LucasFilm Games - https://www.lucasfilm.com
[R3] Snacky and Rubicon: https://ready64.org/articoli/leggi/idart/102/rubicon-rivelato
[R4] Eric Zmiro's Best Protection Kit: http://fabulousfurlough.blogspot.com/2008/08/eric-zmiros-best-protection-kit.html
[R5] OUAS 2019: https://www.youtube.com/watch?v=wec5tt9OPN0
[R6] Programming the Intel 80386: https://archive.org/details/programmingintel00smit

The Real Ghostbusters Arcade Fangame

Interview with **Giancarlo Schiano** and **preview** of the game

Hot Topic!!!

by Carlo N. Del Mar Pirazzini

"This is a dominant force of the underground world at the head of a legion of evil demons. The purpose of this is to invade and conquer the world of the living."
Dr. Egon Spengler

A few days ago, I got a notice on FB that stunned me. The Italian page dedicated to Ghostbusters talks about the future release of "The Real Ghostbusters arcade fangame". Obviously I rushed to see what it was and the amazement turns into "I must try this game!!!".
A mix of Metal Slug and Columbia Pictures' beautiful TV series. Excellent.
So I view the pictures and a small video and then I decide to write to the developers immediately.
And here we are with a new interview with Giancarlo Schiano, a piece of the heart of this developing game (the other part of the heart you will find out who it is, A/N).
An exquisite person with whom you can talk about everything, but above all a big fan of the ghostbusters and the TV series. Here follow the questions and the pictures of the game.

Nith - Hi, introduce yourself and present your project to our readers. Who you are and how the development of this game was born.
Giancarlo - First of all, thank you for this interview, which is an honour for me, a pleasure to do with you and present the project to your readers. My name is Giancarlo, I am 33 years old, I live in Ravenna and I have always, since I met them, since I can remember I am a crazy fan of The Real Ghostbusters, more generally in the world of ghost catchers but in particular I have followed with great passion the television saga of the heroes of New York. They always kept me company in the afternoons as a kid, they made me dream a lot because they expanded the world of ghost catchers beyond the film, giving new adventures and above all they gave you maybe some teaching, because for us boys of the time cartoons were a vehicle of instruction. And for me all this was why I decided to create the video game.

Nith - What sparked the creation of a GB arcade games in the style of Metal Slug?
Giancarlo - The stylistic choice to combine a cult game like Metal Slug with The Real Ghostbusters was born from the fact that many years ago, during my twelve years of age I spent the summer with a cousin of mine and he invented an incredible excuse to attract my attention and passion for the TV series. He invented a program with which you could edit the characters and enemies of a game to your liking. I already imagined creating Egon, shooting with the flow against ghosts, against zombies, mummies... then, of course, it turned out to be a colossal "bale" and this has left me over the years a small/large void inside. As a fan he left me this void and I wanted to fill it, years later, myself. Obviously we have taken inspiration from design, but we have completely recreated everything you will see in the finished game.

Nith - What system are you developing the game with? How many of you are on the development team?
Giancarlo - I want to tell you that there are two of us. Only

in two. Two people who have spent their free time creating this game for two years. Two people, Morgana Zaltron, who is my partner and I. She's the code writer and I'm the designer, dealing with graphics in total. We are developing the game with Game Maker Studio 2. I always thank Morgana for the work we're doing. The game at present is between 80 and 90% of the final realization. We're working on the final code optimization.

Nith - What platforms do you plan to consider and what kind of player target you have in mind (i.e. the game will have an old-fashioned arcade imprint or be more similar to modern products in terms of playability)?
Giancarlo - The game will be entirely "OLD-FASHIONED". Do you know the first Metal Slug in the early 1990s where you spent capital to finish it?? Well, that's exactly how it's gonna be. There will be no special effects, there will be nothing of the dynamics of the current games. It's all purely retro-arch. Classic! In some places it is also "tricky to play", but it is the beauty of historical arcades, where you had intrinsic difficulties during the game that did not make your life easy and that created the challenge. The desire to see even more, to get better. This was our idea, our imprinting to create the game.
For the moment we do not plan to do it on other platforms (remember the game will run on Windows systems, ndN) because we still have to close the game on PC. So our first step is to close it and make it FREE for everyone. Then if we can convert it and bring it to other platforms and devices for us it will be a plus that we reserve the right to do but do not guarantee at the moment.

Nith - Exit forecast? What outreach platforms do you want to use?
Giancarlo - I don't really think we're going to spend a penny on advertising! Let me get this straight. It's a game created by fans, me and my partner, for fans. We hope that the fandom linked to the series is so proactive as to do it himself by means of dissemination, by advertising for this project. We don't ask for anything. We donate it. Together with the game we are donating three years of our life and passion for this game and for The Real Ghostbusters without having anything in return but enthusiastic fans. And there's nothing better than an enthusiastic fan as a means of outreach!
As far as exit is concerned, we hope to get it out completely by mid-2021. Consider that we have been working on it since 2018 and it is becoming a necessity for us to close it. And to close it I intend to make it a playable product as soon as possible.

Nith - Tell us about the game itself. What about the gameplay?
Giancarlo - As I told you, it reminds you of Metal Slug, so imagine the game with The Real Ghostbusters themes. The entire game is a gigantic cameo of everything seen in the TV series and all the Kenner toys that came out after the TV series. We have also hidden numerous Easter Eggs of films that year made the history of cinema of the 80s/90s. I leave your imagination free. To wrap up the Gameplay thing, let me tell you something... I'm doing a little spoiler. There will be 5 levels plus an extra selectable from the menu called ECU (i.e. the ecto container) that will be used by players to visit all the entities they have captured within the game. A kind of Wall of Fame where the character can "interact" with the entities he has defeated within the game (and not only, ndN), just as it happens in one of the episodes of the cartoon, the one dedicated to the Christmas story by E. Scrooge by Charles Dickens (at this point in the interview Giancarlo unveils his knowledge of The Real Ghostbusters telling me the episode perfectly. This guy is a real ghost catcher encyclopedia! NdN). Each level has a different title. Each level is dedicated to a particular theme. The first is dedicated to a particular enemy of Egon, fans will understand and be enthusiastic! The second is dedicated to Ecto 1, in the background of New York City. Here we will find the enemies of the animated series and Kenner toys. The third level dedicated to Ecto 2, or flying in the skies

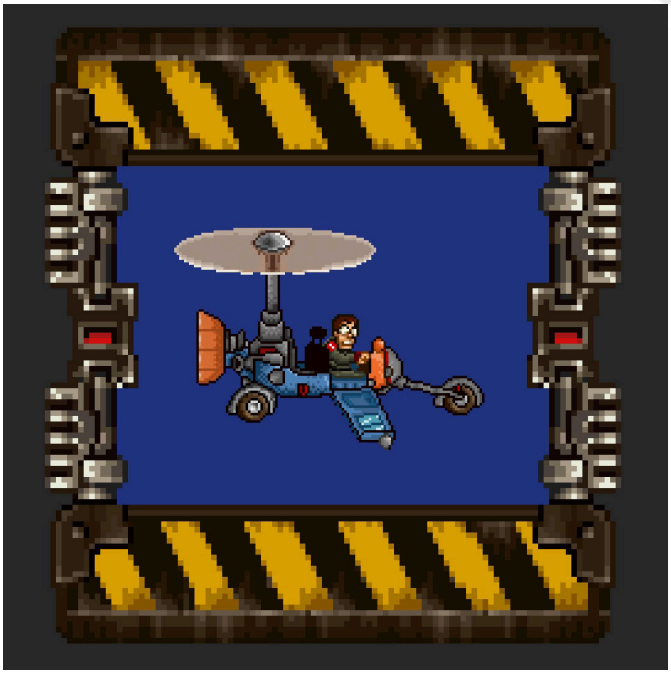
of New York.
The fourth level will be dedicated to an infested villa and will take us to the sewers of New York, already seen in the film Ghostbusters 2. And we will find ourselves at the end of this level facing four opponents within the ghost dimension! The fifth and last level... the final level. But that's all I'm saying.

Nith - Do you have a campaign to help support you?
Giancarlo - The problem here is very serious. We know that we are going to interact with a registered trademark. A brand that belongs to Sony and Ghost corps and we cannot ask for funds or crowdfunding to support the project, but we can leave open to donations the opportunity that any fan can make to support the project. But it's just a donation that would go to help support living expenses for development. In this sense we do not have and I do not want to create a campaign in this sense. We want our fans to be delighted by the product and its being free. Hopefully it will bring the fans back to a happy time, given the terrible time we're going through.

Nith - Thank you for your answers. If you want to add something, feel free to say what you want.
Giancarlo - Meanwhile I want to thank Morgana, without her the project could not have gone ahead. I want to thank the fans, true supporters of the project. In less than 4 days we have had almost 2000 followers, an unexpected thing and this proves the fact that word of mouth is more important than paid robotic advertising.

I thank you from Retromagazine World for giving us the space to talk about it, because it is wonderful for me to share for free something that was part of my childhood and that, in part, made me what I am today.

Having said that, I conclude with a personal reflection: Given the follow-up that a small project like this has had and is having on so many fans of the series and not fans, I have to say that budgets are not important, the money invested is not important but the heart and love that people put into it. Too often I see games with very high quality levels on a graphic level but with a poverty of plot and abysmal gameplay (and this is pure truth, to be printed on immortal boards. NdN). Finding a very old, in some ways cumbersome style of play, perhaps light years away from what is found in our consoles and computers today but still so appreciated makes us understand that, in the end, it is not the graphics, but the love, the passion, of putting every drop of effort into making everything perfect. For example, the game of Ghostbusters released in 2012 was very successful because you could see a continuity, a spasmodic love for the brand and for the characters. Nothing was left to chance.
The opportunity on this brand, on The Real Ghostbusters are endless, we fans hope, I first, that it is taken up. That it can expand and that it can have the same impact that it has given us in the past to the new generations.



I think we can stop here. Thank you. I want to make a small announcement, in addition to this playful project, we also have the film "THE REAL GHOSTBUSTERS – THE FILM", unfortunately stopped because of COVID, but we are ready to start again and carry it forward. Again you will find all the information on The Real Ghostbusters Live Action page. I am actively involved in the role of Egon. Thanks again to all of you and who knows, if fans will appreciate it, in the future the project could also have a follow-up. Make it a job? It's hard for now, but who knows?

We thank Giancarlo and Morgana for the work they are doing and for this comprehensive interview.

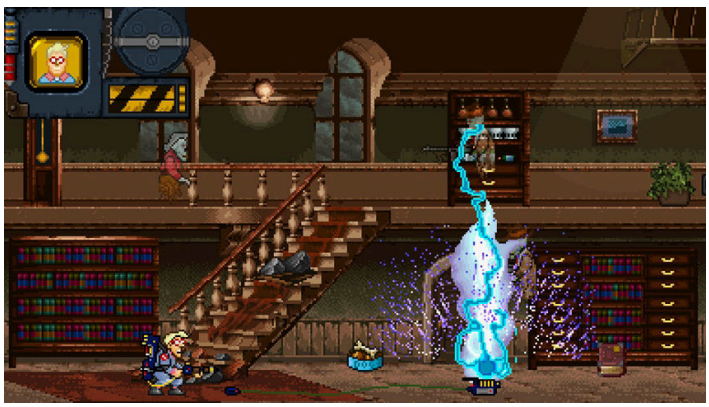
We invite you to follow the project on facebook at **The Real Ghostbusters Arcade fangame** page and leave you to the beautiful images of the game.
Personally, I can't wait to get my hands on the game and test it as a fan of movies and the TV series.
I'll leave you with a quote from the TV series.

"Remember, if you're not afraid, he can't hurt you. "
Winston Zeddmore


ECTO-1
F.D.N.Y.
AMBULANCE


ECTO-1


CITY OF DAMNED



Metamorphosis (ZX Spectrum 48kb/128kb) - PREVIEW

Evolution takes a step forward

by Starfox Mulder

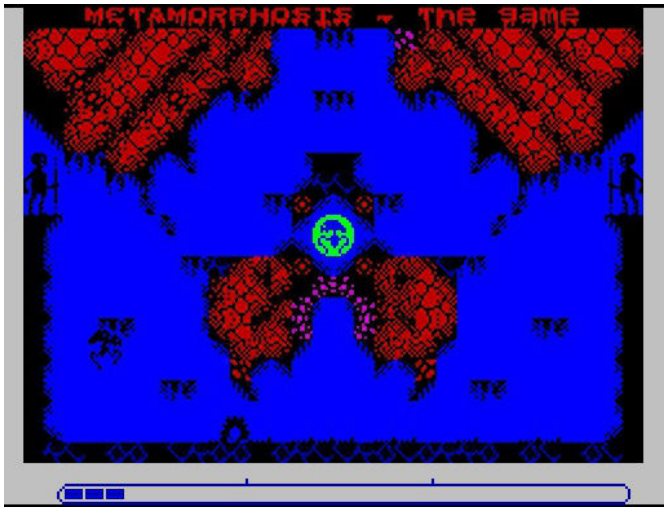
It's not often that you get to talk about retrogaming and get some exclusive news to share. If a hardware is retro, by definition, it is out of production and rarely receives releases of new video games. Although some machines enjoy a renewed youth, we are used to trying the new games once finished and published. This time we will make an exception.

Leonardo Vettori (graphics and gameplay) is one of us, from the RM Team, and is not new to the retrogaming world even as a developer. Remember Core 447/448? Retrieve issues 16 and 17 of this magazine (as they are always in the archive downloadable for free). Assisted by **Kees Van Oss** as programmer, **Pedro Pimenta** and **Mario Fanciulli** to the music, the team of wonders launched on the idea of a new platform for **ZX Spectrum**, with exceptional graphics and innovative concept design.

In Metamorphosis everything revolves around the concept of evolution. We will start the game with a being resembling an arachnid and a very limited energy bar. We can move

in both directions (right and left), jump or hit enemies with a particular spit attack.

The aim is not to kill them, but to bring them back to the previous state of existence, weakening them suddenly until they degenerate into worms and we will be able to eat them (thus regenerating our energy bar).



Initially the maximum number of energy points available to us will be very limited but with the game progresses we will be able to recover objects that, if placed within the energy radius present in the central room, will increase the maximum value of the energy bar, in addition to approaching the conclusion of the stage.

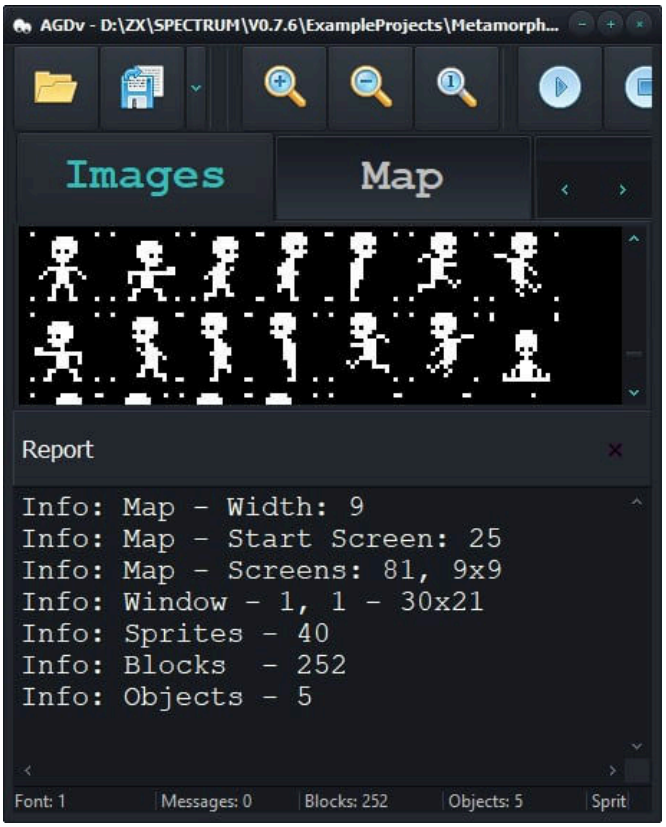
After a certain threshold, we will evolve into a human-arachnid mix, “and this is not even my final form” (a well-known Villain would have said), but aware that any blow immediately could lead us to go back to the previous form.

Why evolving?

First of all because it is the purpose of the game to become real children (today I have taken the quotations well) but

in terms of pure gameplay evolution will lead us to concrete advantages that I am not going to describe to you because... I only played (and finished) the Demo version.

In the version of Metamorphosis made available to me and finished on real hardware you can only conclude the first of the three levels that will then be present in the final version, so my judgment is concretely limited to what is worth "tutorial level".
Enemies are intelligently programmed and do not just attack us but also fight each other, sometimes evolving in front of our eyes after eating an opponent and becoming more threatening referrals.



I never remember this element being introduced into a game for Spectrum and it gave me great pleasure. The graphic impact then, as mentioned earlier, is among the most incredible things I have ever seen on the machine in question. Music is an excellent accompaniment and to finish the gameplay shows no side of any concrete criticism...except perhaps that of not being an original game system, but you have to explain it to me how to create something revolutionary for a computer having 38 years on its back.

In its proven dynamics, Metamorphosis still offers its best, leading us to alternate exploratory scenarios with pure combat stages, unlockable with special keys to be found along the levels. All with fixed screens, all capable of running perfectly on a 48Kb (but if you want music you need a 128Kb).

So what can I say in conclusion?
Leonardo is a friend and he asked me to be ruthless, but given the product that has been made available to me it is really difficult to find a defect. Too easy? Yes, obviously, but this is the first stage, I'm sure they'll raise the bar for the competition in the next two.

What will certainly not change is the sense of wonder effect due to an original and well-kept setting, capable of plunging us into a primitive-gygerian delirium never seen.

This is by far the game that created me most awaited (A.k.a. Hype) in the last period.

LOOM

Year: 1990
Editor: Lucasfilm
Developer: Lucasfilm
Genre: Avventura grafica
Platform: Amiga

"A Musical Journey"

Original article Simone Battaglioni, reissued for RMW by Gianluca Girelli

Loom is a historic video game from Lucasfilm Games released in the 1990s on many platforms. The review refers to the Amiga version.

Synopsis
With a gloomy medieval fantasy atmosphere, Loom's story tells the journey of a boy belonging to the "Guild of Weavers" to discover himself and what happened to his people. He will soon find himself involved, in spite of himself, in saving the fate of the kingdom.

Development
In the game we play Bobbin Threadbare, and our journey begins by learning the first rudimentary notions of how to apply the art of weaving. After leaving Loom Island and landing on the mainland we will meet the other Guilds, including those of Glassworkers, Shepherds, Blacksmiths...

Of some of them we will only hear the name, others will be more involved. We're also going to have some bad fights, including a hot female dragon. We'll discover a variety of hidden secrets and explore cities.
Finally, we will stop Evil and its evil plan.

Content of the various editions
As was used in the 90s, Loom was sold in a cardboard box with representations and a medieval fantasy style, with the "Book of Patterns" inside where you can write down the sequence of notes to be repeated in the game (explanation later).

These sequences of notes were not always the same in order to increase the longevity of the title.
The game was protected against copying with codes to be inserted at predefined times, taken from the manual, not photocopiable as it was written in blue with red lines...

An Audio-Drama on music told the story of the game.

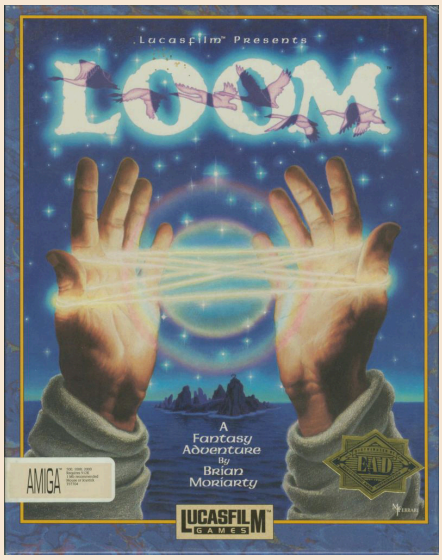
In addition, a leaflet with the technical information according to the version purchased was attached.

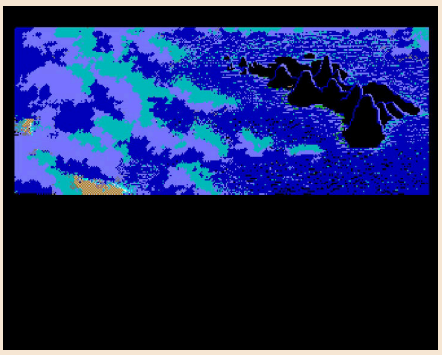
Technical explanation
Loom was the fourth game to use the SCUMM game engine written by Brian Moriarty, with the soundtrack of Tchaikovsky's music and the graphic backdrops created by Mark Ferrari.

Developed first on EGA PC it was then brought to various platforms including Amiga, Atari-ST, Macintosh.

Loom is a graphic adventure, but different from all the others: while in the others you use actions such as Open, Close, Talk and so on, here you use musical notes to create magic called "Textures" with which you can perform actions such as: Dye, Open, Fill etc.

The musical arrangement adapts well to the fantasy atmosphere, with music that repeats itself at the right times. This mechanics, which nowadays seems obvious, is actually quite difficult to achieve effectively, and the frustration of not having music synchronized with the game later led the programmers of Lucasfilm Games to develop the iMuse (Interactive Music Streaming Engine) system.





OUR FINAL SCORE

» Gameplay 85%

Thanks to the three difficulty levels it is well balanced and the puzzles are not so complex that they can't be overcome. One of the most beautiful adventures ever made.

» Longevity 90%

You won't get tired of playing it to find out if you left something behind or just for the sake of listening to the melodies again.

As an addition, the sound effects heard during the various locations accompany the player without making him forget where he is at that moment.

The graphics sector developed on EGA PC cleverly exploits dithering to be able to simulate on screen a greater number of colors, so until the VGA version came out in 256 colors with as much dubbing on CD, EGA version was maintained in all the previously mentioned platforms.

The latest versions were released for FM-Towns and PC-Engine TG16.
To the detriment of what one might think, Loom was fairly successful at the time: Brian Moriarty declared that he was willing to write a trilogy in which the second chapter should be called "Forge" and the third "The Fold". Later he put the idea aside to devote himself to other projects and there was no one else who thought or felt like continuing.

Conclusions

Loom is really a particular video game, not too enigmatic or complex, but that knows how to dose carefully and entice the player to discover what they can offer. Every step taken is seen as a personal achievement. When it came out at the time it was innovative and much appreciated, so much so that it is still considered a Cult among all nerds...

Curiosity

It is noteworthy that Loom was released in English, French, Spanish and German. Over the years several users have tried their hand at the Italian translation of the video game.
Simone "SimonPPC" Battaglioni, editor of this article, made the last release in a period of time and tried to keep the author's original work unchanged.

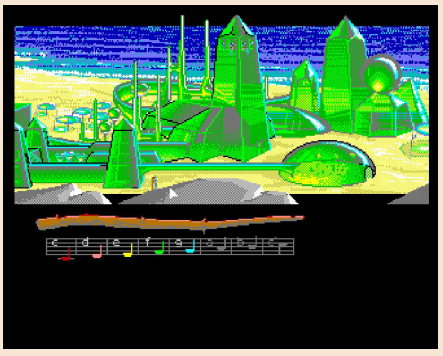
At the time of writing he was working on the VGA version, but he confided that he was thinking of putting his hand back on the previous versions adapted by him, namely PC EGA, Amiga, Atari ST and Macintosh.

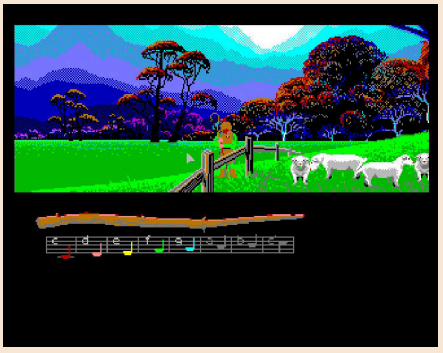
As a cherry on the cake, among the various photos included in this review you can admire Bobbin's mother's grave, which being a graphic part took a long time to figure out how to modify it.

However, Simone has also succeeded in this endeavour thanks to his collaboration with a foreign user.

Before leaving you, we want to propose this video of Loom's longplay made by Simone Battaglioni together with friends of Retro Edicola Videoludica:
https://www.youtube.com/watch?v=DmSyp_OETVO

by **Gianluca Girelli** and **Simone Battaglioni**





HOLIDAY LEMMINGS

Years: 1991-1994
Platform: Commodore Amiga (basically all...)
Genre: puzzle/platform
Reviewed version: Holiday Lemmings

If today the graphic quality of a nextgen title is measured by the polygonal complexity and massive presence of ray tracing, when Commodore Amiga still dominated the world of video games, the most demanding players measured the artistic quality of a game with the quality of its pixel art.

The Lemmings series really gave its best since the rodents in question had been expertly made by DMA Design software house using just a handful of tiny pixels! Lemmings was the kind of puzzle you used to play perhaps after a furious space battle at Project X or after an entire grand prix at Geoff Crammond's polygonal Grand Prix, wasting your hours in the complicated and delicious attempt to bring most cute little beings out to a series of increasingly complicated and challenging levels.

As many of you will know, Lemmings is a series of video games that has been able to entertain millions of players even on tablets and smartphones in recent times, but was born in the early 1990s on the flagship of the Commodore house.

The game levels were for the Lemmings tribe an immense fatality in the sense that if the player had not given them the right skills at the right time they could have stayed there to walk back and forth infinitely or they would have died falling en masse from a height that their slender physique could not bear. In other words, in Lemming, squashing was the order of the day!

The game interface consisted of a multi-directional scrolling time level, a small map, and a series of icons indicating the available skills level variables. The player had to assess the situation based on the morphology of the level and the skills available to complete the mission. Typically the goal was to get a certain percentage of Lemmings to the exit portal but this percentage could be set by programmers to 100%!

A feat not always easy to complete since the artificial intelligence of the cute little beings was limited to making them constantly walk in a certain direction reversing it in case of contact with obstacles that cannot be overcome with their normal walk.

However, using the skills by clicking on the right Lemming at the right time (which is not always easy given their size) we could make them perform even rather complex actions such as building bridges, digging the ground and rocks, using umbrellas as parachutes, blocking the advance of the other tribe members (so that we can proceed without too many losses) or even committing suicide!

Yeah, in Lemmings sometimes sacrifice was also indispensable for the good of the whole tribe and this is how a fun puzzle game that in its Holiday version delighted us with delicious Christmas scenery (but do not miss the main chapters) had managed to gain a special position in the heart of many players of those years and with the new mobile vesicles still today! Oh, no!

by **Flavio Soldani**

OUR FINAL SCORE

» Gameplay 90%

Lemmings is pure strategy, puzzles and brain teasers. If you let yourself be won over, you won't give up anytime soon!

» Longevity 96%

The first and perhaps most rewarding chapters born on Amiga are still available today, ready to be emulated on any type of platform including 8 and 16 bit consoles. Also available versions adapted for mobile platforms.

WEIRD DREAMS

Year: 1989
Editor: Rainbird
Developer: Rainbird
Genre: Horror
Platform: Commodore 64

Dear Dreamers, who of you experienced awakening to try and remember in detail the dreams you just had? Well, I did. And surely many of you, myself included, then went to the dream interpretation sites to get some idea of dreamlike symbols and general meaning, especially if dreams were actually real nightmares. One of the most recurring dreams, which happens at least once in your life to everyone, is the one in which you fly over the skies of our city like a bird. Weird Dreams will allow us to relive some of these strange dreams in front of the PC or smartphone screen. The protagonist in the chess pyjamas of this game, in fact, will find himself living in these bizarre products of the mind because of the sleep induced by an anaesthesia suffered in the operating room of a hospital.

It all begins on a rather confusing screen, where it is also difficult to understand what needs to be done. Once we understand the mechanism of play, we will find ourselves in many other fixed screens with many new worlds, including a carnival and many enemies and puzzles to face, such as giant bees and girls armed with kitchen knives.

The world of dreams is not only singular but also full of mysteries. And this game almost represents it perfectly, thanks to the strange but beautiful settings. Of course solving puzzles is essential to moving forward in the game. Gameplay is a bit slow, and it takes a while to respond to commands, especially when you have to turn around to escape from an enemy. One of the strongest reasons to continue playing (and to complete it), despite the inherent difficulty of gameplay, is the quality of the music theme, different for each screen. The use of the sound chip is truly masterful and deserves the publication of the soundtrack!

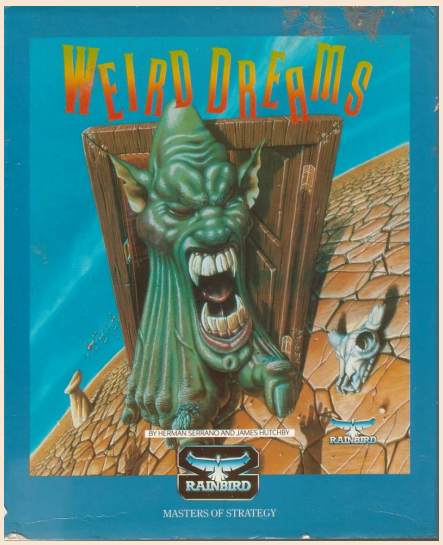
The game didn't get much press coverage when it came out, although in fact, it was judged by good reviews in specialist magazines. He probably went a little deaf because of the concomitant success of the many smash-hits of the time.

As a result, he quickly disappeared from the leaderboards and spotlights of the press and word of mouth. In addition, few stores had it on shelves or in the shop window. Weird Dreams was also converted for 16-bit computers, gaining much more success thanks to the greater graphics and sound capabilities of machines such as Atari ST and Amiga. But in all versions, in addition to the soundtrack, it is the puzzles proposed that entice us to load and proceed through the various game screens. In the C64 version, I highly recommend the disk version for convenience and to avoid continuous loading, as we will not struggle to lose the five lives available (hard to win against a giant bee and a creepy little girl armed with a kitchen knife).

After the loss of a life you will be able to see a scene of the situation out of the dream, that is, the surgeons who are operating on us. Their eyes do not promise anything good about the outcome of the intervention. Are they the authors of all this estrangement that for us means wandering in the nightmare of the game? I do not deny that some people may find the game a little disturbing given the overall atmosphere and suggestion, but it is still worth dusting it down, taking it for a spin and keeping it in your collection as a relic. Pay due attention to him, as I did in this article, because in my opinion he deserved and deserves something more in terms of appreciation by the players.

As always, I wish all of you a peaceful holiday despite these inevitable limitations to which we have been forced for almost a year now. Let's console ourselves with RetroMagazine World and our beloved retrogames!

by **Daniele Brahimi**



OUR FINAL SCORE

» Gameplay 60%
The controls seem a bit slow and delay the action and our reaction to hazards.

» Longevity 75%
The soundtrack and puzzles help extend the life of the game.

MIGHTY FINAL FIGHT

Year: 1983
Editor: Nintendo
Developer: Capcom
Genre: Beat'em up
Platform: Nintendo NES

Final Fight is a classic arcade loved by the video gaming community since it was released as an arcade in 1989. It was brought to many platforms including SNES and also received two sequels, but perhaps not everyone knows that it was also converted in 1993 on NES.

In July of that year in fact Capcom published Mighty Final Fight for the 8-bit Nintendo, a more "carefree" version of Final Fight with a colorful super deformed chibi style.
It is a classic porting of the original arcade and follows the same plot. Everything takes place in Metro city, a metropolis invaded by criminality due to the Mad Gear Gang.

The gang leader fell in love with Jessica, daughter of Metro City Mayor Haggar. Mayor Haggar gets a phone call in the middle of a training session with Cody, Jessica's boyfriend, and Cody's training partner, Guy. The villain on the other side of the line informs Haggar that Jessica has been kidnapped and will be forced to marry the Mad Gear Gang leader. Without a moment to lose, Haggar, Cody and Guy take action and prepare to take on the Mad Gear Gang once and for all to get Jessica home safely.

As you might expect, Mighty Final Fight is a side-slip fighting game. Unlike classic FINAL FIGHT, this version is "only one player".

The absence of the second player unfortunately makes the game lose the title of the biggest 8-bit hitter. But as in the main title, also in this version we can select all three major heroes: Cody, Guy and Haggar. Everyone has their own unique abilities and a variety of different moves.
This version for NES also stands out from the other games in the series by introducing a level system for character progression. The player receives experience points for defeating an enemy and the amount of experience you gain depends on the move you end up with your opponent, the more complicated moves make, the more points you earn.

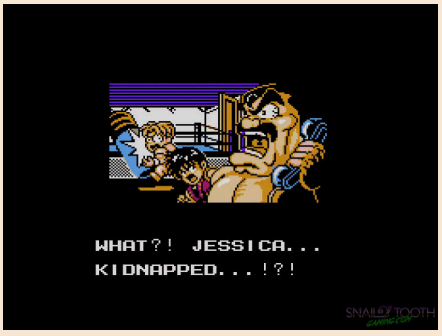
Once you've gained enough experience, your character will level up and unlock new fighting techniques. The game is generous enough to give the player 6 lives per game. There are also bonus phases during the game where you can earn extra lives, health boosts, player specific weapons and even continuous extras. Even with all these lives and power-ups, the game is not a walk in the park.

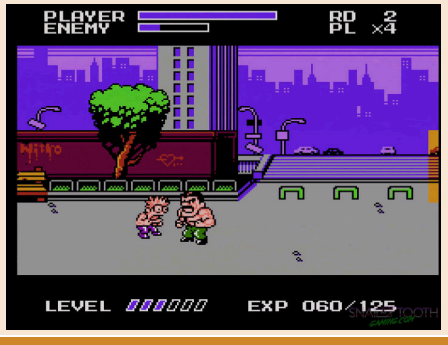
It has a fair amount of challenges, enough to keep you busy but not enough to end up in the same category of games as Ninja Gaiden. At first it may seem too difficult, but the excellent control is easily mastered with practice and gives you the chance to play the game to the end, just like I did in the video below. Overall, the gameplay is excellent. It's fluid, responsive, satisfying and, most importantly, very fun.

The graphics of this game are superb. A great example of what the NES is capable of. The animation of the sprite are some of my favorites ever on the little Nintendo. I love the way enemies react to being hit and the way their faces express shock after being punched.

The backgrounds are well detailed and colorful, and I like the Super Deformed art direction that developers have decided to adopt for this title.

The only problem is the flickering that sprites suffer from. The image processing unit in the NES can only display eight sprites per scanline at





a time. Character models often consist of multiple sprites and when there are too many sprites on the screen at the same time, the hardware must prioritize what needs to be shown.

This translates into the flickering of sprites which is a feature of so many fantastic NES games. This often happens in Mighty Final Fight as well.

It bothered me a little when I first entered this game, but after playing it for a while I stopped noticing it and it left me almost completely in my mind because I was enjoying the gameplay so much.

Trembling sprites are better than watching the NES slow down and suffer; another common event in NES games, and Mighty Final Fight never slows down!

In conclusion, Mighty Final Fight is a fantastic fighting game that is unfairly overlooked and often forgotten. I think this is due to his late release on NES. At this point the SNES had already been released since two years and Final Fight 2 was about to be released on that system. Today, this late release means it's rare and expensive by the retro video game collecting scene.

Fortunately the Famicom version is much cheaper and contains no major changes other than language in the cut scenes.

I recommend following that direction if you want to play this game with an original cartridge. Overall, I find hard to think of anything negative about Mighty Final Fight.

It's a little short and doesn't have a cooperative mode for two players, but it's still a very fun and stimulating single-player experience. Some bosses are repeated and the sprites flash a lot, but the gameplay, control, soundtrack and art are so good that you forget about these flaws.

I highly recommend this game to both occasional NES fans and avid collectors, and there are ways you can play this game without spending your money on an expensive copy of the NES cartridge. Test it on an emulator or on Switch via virtual shop.

There are plenty of options to get out and bring peace back to Metro City.

by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 90%

Exceptional gameplay with an innovative character advancement system for this type of games. Too bad about the lack of the second player option.

» Longevity 75%

Fun, well developed and with a good level of difficulty throughout the game. It is not very long.

MARIO KART 64

Year: 1996
Editor: Nintendo
Developer: Nintendo
Genre: Driving
Platform: Nintendo 64

In Mario's world, everything is fantasy; nothing can be considered real, nothing makes sense.
But what happens when the characters from Mario's world are put in the driving seat?

Mario Kart 64 is an exceptional sequel to Super Mario Kart (SNES).

The original Super Mario Kart contained four game modes: Mario GP, Versus Mode, Battle Mode and Time Trial - available for 1-2 players only.

On N64 Mario Kart takes a big leap forward in its gameplay. Now, 1 to 4 players can challenge each other on all the sparkling new slopes. 16 Grand Prix slopes divided into four cups: Mushroom Cup, Flower Cup, Star Cup and Special Cup.

As in the first beautiful game for the Super Nintendo (still an absolute masterpiece of gameplay after almost 30 years) players will be able to challenge themselves in 50cc, 100cc mode and in the furious and very fast 150cc category. The difficulty is obviously based on these speed classes, the lower it is and the simpler it is and, of course, the higher it is, the more complicated the challenge will be.

Obviously there are no shortages of game opponents, 8 characters all with their own personal characteristics (from light and fast but fragile Toad to Bowser, heavy and destructive), there is no shortage of power ups and weapons to launch to make way.

In 2-player GP mode, they can join forces or compete against six other drivers. Versus mode, on the other hand, brutally presents the challenge between 2-4 players on the slopes of the game. No cups or prizes. Simple brutality between rivals.
They are present as we said 8 playable characters: Mario, Luigi, Princess Peach, Toad, Wario, Yoshi, Donkey Kong and Bowser. Classified as light, medium and maximum weights.

Therefore each weight class (as we said a few lines above) has a strength and a weakness and we will be forced to choose with a little practice the most suitable character for us.

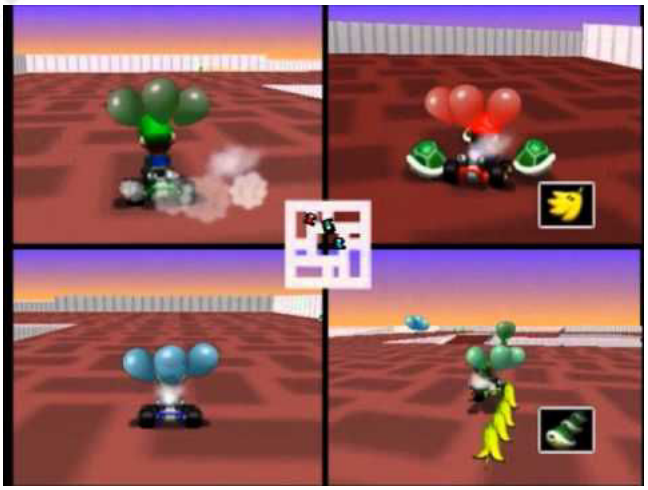
I prefer Toad and Yoshi, for example, but the former suffers for the tonnage and the second for the handling.

There are the classic weapons of the first Mario Kart such as banana peel, green shell and red shell, the time ghost that allows us to disappear momentarily and the star.

Add to those the multiple bananas and the set of shells, that can be collected around and which allow us to have more bullets than normal.

There is also the fake power up block which, if used correctly, can become devastating and very annoying.

The real value of the game, however, is in the Battle mode, available for 2-4 players at the same time, which



allows you to play a game in four different arenas.

In this mode each player receives three balloons. The aim is to avoid getting these balloons destroyed by the opponent by trying to blow them up to him.

This mode is perhaps the most fun and destructive part of the game. Capable of keeping us glued to the screen and breaking down loves or friendships given the high challenge rate.

The generational leap between the Super Nintendo and the Nintendo 64 is clearly visible in graphics and its development. A very detailed 3D environment where sprites take on more “courage” in movements, the tracks become more colourful and detailed and there are numerous graphic effects.

A beautiful and colourful 3D fantasy world in pure NINTENDO style. It certainly doesn't take full advantage of the machine's hardware like other games (Mario 64 or Zelda), but it's a great look and spins beautifully.

The soundtrack and sound effects are also customised. Each character makes different sounds or screams.

Game controls are perfectly balanced and never frustrating. Accelerate with "A” and brake with “B”. The “Z” key selects the object and shoots it, the back keys “L” and “R” blow up the character.

Pad "C" changes the displays, the analog lever is used for driving. Convenient and affordable predefined controls, but you can always customize them.
In conclusion, Mario Kart 64 is ideal for playing against your friends.

It is the type of game to insert during the Christmas period when (Covid's restrictions allowing) relatives at home get bored.

We turn on the Nintendo 64 and take on an unprecedented 4-man challenge.

A compelling gameplay and perfect nintendo style gameplay structure make multiplayer gameplay the most beautiful part of the game.

You're probably wondering why I didn't write a few lines or a review for the first Super Mario Kart.

Because I consider this version to be the turning point of the series.
The evolution that has allowed brands to continue in their future incarnations.

A diamond thanks to the multiplayer version.

Mario Kart 64 is a well done, fun, noisy and almost perfect game. Even after all these years.

by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 95%

Numerous game modes in both single and multi player, difficulty system calibrated through go kart classes, perfect controls.

» Longevity 95%

How can you rate a game that has made on-screen multiplayer its strong point. Up to four players in GP mode is fun, in VS mode is violent and rough, but it is with the Battle mde that becomes legendary! Immortal!

NEW GAME!!!

WIZ QUEST FOR THE MAGIC LANTERN

Year: 2020
Developer: Mutation Software
Genre: Platform
Platform: Amiga (AGA)

Wiz Quest for the Magic Lantern is a new 2D side-scrolling action platform game for Commodore Amiga. In Wiz, take control of the last old wizard and you need to find the Magic Lantern before it's too late. You are not unarmed in your research as you have your magic book and potions at your disposal.
You will need every trick to complete this search as the world ahead of you is dangerous.
A classic start to a new game.

Good old-fashioned action platform
When we think about 2D action platform games, we probably think about Mario and company. I do, too.
The Nintendo franchise is synonymous with the genre and for many good reasons. What should make this new Friend game different from the classics? Wiz Quest for the Magic Lantern has that extra touch of charm that most 2D games lack.

For example, the set of motion animations of our protagonist, for example we take the animation on foot, with extra frames for nose and hat. Both move at every step bringing some realism to the game, but not only is the entire set of animations really well cared for.

The devil is in detail as an old proverb says. Fortunately, Wiz Quest for the Magic Lantern pays attention to them. The developers have developed the game in two ways. The classic version in physical version that features manual, box and discs (and several extras, ndr) and the digital download version for anyone who is not lucky enough to own a Friend 1200 or 4000 in physical form.

Apart from specifications and features, what we have is a good solid platform. The short development time and the fact that this is the first (of many) new games for Mutation Friend means there is not much to push on what the hardware is capable of. Wiz is not a fast and contracting game in any way. There is no timer, so you can spend your time passing through the levels. And it is a game that does not allow mistakes, it is really an “old school" product.
Only doubt. And if it wasn't for the

beautiful AGA graphics, I'm sure the game would easily run on a 68000 Amiga. This does not mean that Wiz is a bad game, as it is not, but perhaps by optimizing the code better you could have enjoyed even with less performing processors than 68020 or higher. It's a cute, cute and unusual platform with smooth scrolling and it deserves to be in any collection of Amiga owners.
As I said, the levels are not very many, but they are all quite difficult to complete without the necessary attention. This will give you several hours of play.

The verdict can only be positive. It's only $9.99!
You can order it here:
http://softwareamusements.com/MutationSoftware/index.html
by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 75%
The protagonist may be tender and delicate, the monsters are cute... but the game is quite difficult and requires practice (it is really pixel perfect). Well developed control system and physics of jumps.

» Longevity 80%
A nice full 8! The level of challenge is high and will make you spend several hours in front of the screen.

NEW GAME!!!

ZETA WING

Year: 2020
Editor: Protovision
Developer: Sarah Jane Ivory
Genre: Shoot'em up
Platform: Commodore 64

A perfect arcade game. This is Zeta Wing.

A technically perfect vertical shooter, fluid, with tons of parallax, no slowdowns and a compelling game play.

A product developed with a lot of love towards the commodore scene and for the love of its players.
The plot is simple. Clean the planet with our spaceship through 7 levels

with the possibility of improving our weapon system by advancing through the levels.

The advancement takes place with 10 enhancements of our defense system and, joy and triumph, even if you die you do not lose everything as in many shoot em ups, but you go back just by an enhancement. That's because the game is well balanced, but it's bloody hectic and full of challenges.

To all this let's add an atmospheric music for this genre, three levels of difficulty and saving the score on the disc.

A game worthy of the best shoot em ups on the Commodore machine, but also worthy of the beautiful arcade games of the same kind.

Beautiful and also inexpensive. Only $3.99 through the author's website.

Don't miss it.

2020 is a troubled year for the world, but on the Commodore 64 side it has seen a scene rich and full of small jewels.

Beautiful!

by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 90%
A perfect arcade game!
Balanced, playable, fluid.

» Longevity 90%
It will keep you busy for quite some time and the difficulty level is well balanced.

SUPER MARIO 64

Year: 1996
Editor: Nintendo
Genre: Plaform
Platform: Nintendo 64

In 1996, with the overbearing entry of the "polygon" into the world of video games, all the developers of the time set out to produce anything using three-dimensional graphics engines for all the machines in the market.

A difficult and not always successful undertaking. The generational leap had not yet been fully somatized by programmers and, very often, the results were “strange” and disappointing.

The Nintendo 64 also saw the change of the company's main saga land in a new three-dimensional guise, but thanks to Shigeru Miyamoto's capabilities and ideas, this jump from 2d to 3d was very successful.

And so came a new chapter of Mario also on Nintendo 64 and revolutionized the world of platforms, influencing it for future generations.

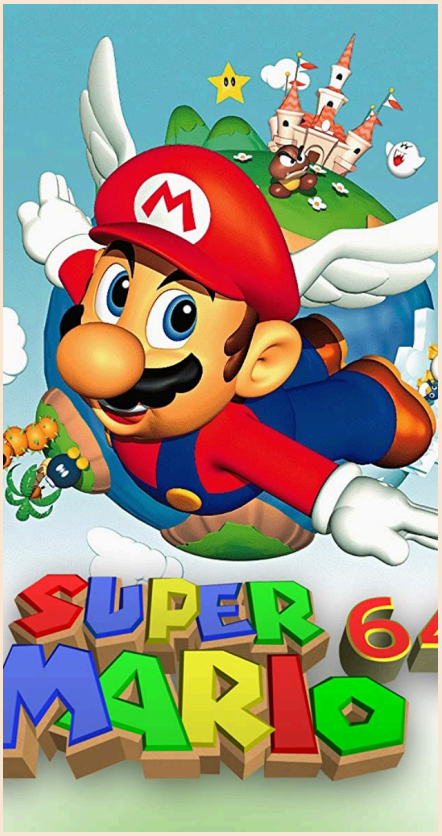
Nintendo hit the spot once again, creating a perfect balance between control system, graphics, gameplay and a sense of exploration that had not yet been seen before. This made Super Mario 64 an authentic universally recognized masterpiece.

When I was young, holding the "TRICORN" pad and taking control of the plumber with moustache I could sense the feeling of this new adventure of Mario, opening who knows what doors, into the future of video games. And it really was.
With 35 years since the birth of the Italian-American plumber saga, Nintendo has decided to make three titles for Switch: among them of course there is Super Mario 64.

But since in this issue Nith talked about the original console (in the Hardware section: Nith), I will talk about this version. Super Mario 64 is a pleasure today, the search for 120 stars can still become an obsession, in fact if it were up to me I would have added even more.

In the modern eyes of the classic video player, the camera that follows you may be cumbersome (in reality I found it a bit annoying already in 96: Nith) and the graphics a little "retro", but nevertheless from the technical point of view, SM64 still defends itself quite well and makes proper use of the hardware of the Nintendo 64.

The control system is perfect. Mario replies to commands like a charm and we are really free to do what we want. This has also changed the approach to the saga in the next chapters such as Sunshine for GameCube and the beautiful Galaxies for Wii.



For anyone who wants to dive into one of the most important and fun video games of all time I strongly recommend recovering it.

You can try it on N64 with the original physical version or in a Rom version for Everdrive. Or maybe on an emulator or even in the beautiful collection on Switch!

Try it! There are many beautiful games, but there are few titles capable of creating a new path for the future. Super Mario 64 created the way for the future. A masterpiece by Miyamoto that was the springboard for the last cartridge console in history.

Before I leave, I'll explain my vows. If there was a global, I'd give 95% off because it's the whole thing. The votes I have expressed are a “desire”, that today I would like much more. I would like more stars, more levels, more Mario...

by **Hakim Rezki**

OUR FINAL SCORE

» Gameplay 90%
A small miracle of a game, with an incredible control system. Still a pleasure to play today.

» Longevity 85%
120 Stars is a lot. But I wanted more.

FINAL FANTASY VII

Year: 1997
Editor: Sony
Developer: Square Soft
Genre: JRPG
Platform: Playstation 1/PC

In the world of video games, the history and setting of Final Fantasy VII have been elevated to the realm of myth.

First published in 1997 for Sony PlayStation, it ushered in a new era of Japanese role-playing games and, in the meantime, its story where the hero Cloud Strife and a disorderly gang of idealists and minions fight the captivating and evil Sephiroth for the sake of the planet, has become bigger than life for many players.

Cloud, Sephiroth, the floating city of Midgar: these are significant in the world of games that go far beyond the game from which they come and exist with power outside their context.

Final Fantasy VII was the absolute killer application for the Sony machine and perhaps the most dazzling visual experience for that console and consoles of that period.

High-quality movies blended seamlessly with game graphics and animations to create the surprisingly realistic world of the game.

But what struck the most about the game was the plot that helped to weave this aesthetic game.

The moving plot of Final Fantasy VII is influenced by some of the greatest films and science fiction literature, including Frank Herbert's Dune, Mary Shelley's Frankenstein, and even Godzilla.

Just this year, on the wave of countless similar operations, the Remake for the next generation consoles came out. A remake made graphically and beautifully, which changes very little the game mechanics and the trend of the plot.
But let's focus on this primordial 1997 version.

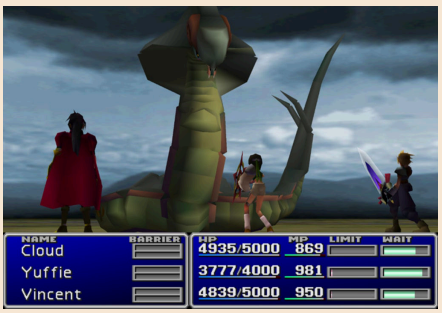
As with previous and beautiful episodes featured on Nintendo consoles, this episode was presented to the general public who were learning to appreciate video games. A much wider audience than the previous 90s Wars console, which featured neophytes literally electrocuted by the well-structured gaming system and the perfect narrative.

Glued to the screen for hours, excited by the stories. Think I've even seen kids cry as the story unfolds.

But after all these years, what can we say about this version? Is it still good?

It resents age aesthetically, but when you reopen the box and turn on the first Playstation again the magic will be there with you again.
Bard's word.

by **Roberto "Il Bardo" Del Mar Pirazzini**



OUR FINAL SCORE

» Gameplay 90%
Intuitive and enjoyable control system. The perfect narration will keep you glued to the screen.

» Longevity 95%
Three cd roms of puzzles, fights, and stories. Do you need more?

NEW GAME!!!

SYDNEY HUNTER AND THE CAVERNS OF DEATH

Year: 2020
Editor: CollectorVision
Developer: CollectorVision
Genre: Puzzle/Platform
Platform: Super Nintendo

Last year, I gladly played Sydney Hunter and the Curse of the Mayan for Switch and loved it, so imagine my surprise when I found out that its predecessor is actually available as a Super Nintendo game!
Once I got my hands on it, I was immediately brought back to the mid-1990s, when SNES games dominated the wars console. Wonderful '90s!

Let's start with packaging. The box is beautiful, complete with fantastic covers by Joe Simko (Joe Simko is an illustrator from New York City who contributed to the collectible cards of Topps Garbage Pail Kids and Wacky Packages, Nth) and is actually better than the boxes of the "old-fashioned" game because it is made from a durable material rather than a fragile cardboard. Inside, the game cartridge is in a transparent plastic holder which, once again, is probably better than the classic SNES game packaging. Last but not least, the manual that contains valuable information and, of course, a page to write down your passwords. It really is an extraordinarily well made package.

So, what about the real game? Sydney Hunter is a geologist on a mission to explore Mount Fate. Soon he is trapped in the mountain and the goal is to guide him towards freedom. You do this by running, jumping and climbing through 12 complicated labyrinthine levels while you launch your boomerang to the terrible creatures that reside in caves.
Many rooms are surrounded by darkness, but fortunately Sydney has a flashlight that he can hold and illuminates the room. However, he can't run with the flashlight, so you're forced to memorize where the enemies and dangers are. The resulting formula is quite challenging and tests your ability to progress slowly through the rooms.

One thing this game does exceptionally well is to offer scenery that rewards exploration as there are many treasures hidden everywhere. You will need to explore to progress as most steps include blocks that do not open unless you place certain objects on their pedestals. Sometimes, removing objects gradually raises lava, so you'll need to quickly return the object to its pedestal, otherwise Sydney might just be the next sacrifice for the island's deities.

On a graphic level, Sydney's animations are very beautiful. Watching him swing on the ropes and run and jump around is a pleasure. However, the environments never really change, so get used to seeing many contours of blue rock, red lava and green vines. On the bright side, the music is well made and provides some tribal songs mixed with old-school synthesizers as you explore.

In a particular period like this, Sydney is a great game to relax and get lost in the caves. You can find it on the site https://collectorvision.com/ along with other interesting titles for many systems (Amiga, Intellivision, Coleco, Snes...) at the cost of 40 dollars in full version or 25 dollars loose.

It deserves.

Oh, I forgot. Sydney Hunter's saga is featured on several gaming platforms: you can also find it for Sega Master System, Coleco Vision, Intellivision and Nes. They are all very fun games that duly exploit the potentials of the machines on which they run. Bard's word!

by **Roberto "il Bardo" Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 80%
First-rate packaging, perfect for retro game collectors. Challenging but simple gameplay. Good music and nice animations.

» Longevity 65%
Fairly short without much replay value. Environments unchanged throughout the adventure.

RISTAR

Year: 1995
Editor: Sega
Developer: Sega
Genre: Platform
Platform: Sega Megadrive

"All the stars are coming out tonight
They're lighting up the sky tonight..."
Take That - Rules the World.

The stars are on our Sega Megadrive and they are in excellent game.

One of the latest platforms produced before the console was dismantled.

The protagonist Ristar has the arduous task of saving the Valdi galaxy from the plans of the evil alien Greedy. He is certainly not the strongest hero or the most technological, but he is the most heroic! Our hero has no super rotations or super jumps but a pair of extendable arms with which he can grab objects, cling to stairs, climb the most unthinkable places and, of course, beat enemies.

The game presents itself as the classic two-dimensional scrolling platform, similar to Super Mario or Sonic (in this case because of the same development team, the Sonic Team, ed.).

Ristar's extendable arms are used as the main means of attacking enemies; extending arms, grabbing the enemy and pushing toward them in a "headshot " movement to defeat him. The same movement also allows you to open treasure chests containing various objects or to hit different parts of the environment, such as falling trees. Furthermore his elastic arms can also be used for grasping and/or throwing objects.
In addition to attacking, Ristar's arms are also used as a method of projecting it across levels. Many pole-like structures are present to swing Ristar from side to side, through empty spaces, or to climb up or down vertically from platforms.

Ristar is also able to cling to enemies and objects in the air and swing on them. The "Star Handles" are placed in the levels, where the player must grab Ristar and use momentum to swing it in a 360 degree circle.

Letting go throws him in a certain direction, depending on the time of release. If you gain enough momentum, the sparks appear behind Ristar and he makes a move called "Meteor Strike", which makes him invincible and able to defeat any enemy by touching him.

When enough momentum is lost, usually a few seconds, the flight ceases and he falls to the ground returning to his normal state, although this can be extended by bouncing against walls and ceilings during the flight.

Each level ends with a special "Star handle", which is used to start Ristar until the end of the level. Bonus points are awarded based on Ristar's altitude when you fly off the screen, similar to

how levels end in Super Mario Bros.

The game gone almost unnoticed, at a particular time of the generational change between consoles.

The development team has created a truly excellent product from a graphic point of view. Colourful and superbly animated, this game is a real eye joy.

Less excellent is the sound compartment, but still enjoyable.

Innovative in game play but perhaps less intuitive than Sonic. Cumbersome in maneuvering the character movements.
There are six planets to clean up, in each of them there are two levels, with a miniboss in the first and a real one in the second. Once the planets are completed, we cab go to the final clash with the wicked Greedy.
It is not an impossible undertaking and therefore the final longevity is average.

Final tip, somehow retrieve this niche product and play it.
You will spend a few hours relaxing ... among the stars!

by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 80%
It's a Sonic Team product, so it's well developed in all respects, but it's not as intuitive as the Sonic series. It takes a bit of practice to better manage the character's abilities.

» Longevity 80%
Six worlds to explore and a longevity typical of this genre of games.

Here are the numbers!

No, I don't want to talk to you about COVID numbers, to which we are all sadly accustomed; in the last issue of the year, in the last article on the last page, I would like to share with you some numbers and facts that have defined this and the first 4 years of RetroMagazine World.

This year approaching its conclusion (in the moment I was writing the article), apart from all the difficulties that have characterized it, has seen our magazine transform itself deeply: RetroMagazine has changed its name, becoming **RetroMagazine World**.
This change was necessary because we wanted to open ourselves up to the world by publishing the English version as well.
Maintaining two publications in two different languages, takes time and considerable effort by the entire editorial staff, for this reason you may have noticed a slight thinning of the releases... But don't worry, we're always alive and kicking. Today more than ever, in spite of the COVID!

Those who follow our Facebook page will undoubtedly noticed the change of pace we made this year: every week we publish dozens of posts, curiosities and previews of new games under development. Kudos for this decisive change goes to **Nithaiah** who is tirelessly striving to find information to share with all of you.

Well, for those of you who hadn't noticed, we also opened a **Twitter** channel. If you're more comfortable on that social media site, follow us there too.

But didn't you want to give us a few numbers? Sure! Here they come!
In these 4 years we have published 27 Italian issues in Italian and 5 English ones (including this one). More than 480 articles have been published on these pages, including 190 game reviews and more than 100 articles dedicated to coding. 16 There were the RetroMath columns and 24 interviews. We also hit more than 15 programming languages and more than 70 different computer models.
Obviously these statistics are not an end in themselves, but they allowed us to discover we have given little visibility to Atari machines and especially to Atari ST. It was certainly not our intention to snub this computer; unfortunately nobody in the editorial staff is a true expert and we have never received external contributions. We therefore publish an appeal:
Are you an expert on Atari 8-bit and Atari ST and would you like to collaborate with us? Contact us, the editorial offices are open!

Well, the cover of this Christmas issue is special and is a clear tribute to a famous Italian magazine. Let's see who guesses both the magazine and the issue that inspired our graphic artist **Flavio Soldani**.

Before leaving I want to wish you, on behalf of the entire editorial staff of RetroMagazine World, Happy Holidays with the hope, in 2021, to see the COVID emergency end soon because we want to return in person to our retro events!

Francesco Fiorentini

RetroMagazine World-English
Year 2 - Issue 5 - JANUARY 2021

Chief Editor
Francesco Fiorentini
Managing Editor
David La Monaca
Editing Manager
Marco Pistorio
Web Manager
Giorgio Balestrieri