World
RetroMagazine
future days are back

Issue 3 Year 1 - October 2020 - https://www.retromagazine.net - Free Publishing

FORTH: the secret weapon!

GOLDEN AXE WARRIOR
(Sega Master System)

Interview with Giuseppe Ettore Pintus
creator of Freedom Fighter - Rise of the humans!

SENSIBLE WORLD OF SOCCER 2020
(PC / AMIGA)

Alien Attack!
a new game in Locomotive Basic

- RetroMath: how to transform an image
- Minigrafik: a graphical extension for the Vic-20 BASIC
- Cross-programming in C on the Olivetti M20
- Introduction to ARexx – part 2

Japan 13^ episode: Nintendo G&W,
a challenge to immortality

RUFF 'N' TUMBLE (Amiga - CD32)

The driving force of retrocomputing

The passion that we all share and call “retrocomputing” (including perhaps not entirely rightly the term “retrogaming”), like any other passion, leads us to spend a lot of time in the search for vintage hardware, original games and software, old and modern accessories and peripherals, bibliographic material, programming books and magazines.

And yet, guided by this deep passion of ours, we sometimes employ resources as precious as time to achieve what we personally (or collectively) consider the Holy Grail of retrocomputing. It may be a particular disk-drive, considered rare because few pieces have been put on the market in the past, or a unique version of a chip, or even a home computer with a low serial number on the original manufacturer's plate. Whatever the object of desire, as for many other forms of collecting, from the most moderate amateur up to the “serial hoarder”, the fact is that we often find ourselves willing to open our wallets and spend lots of money, paradoxically even more money than the initial market price. To give you an example, the launch price in USA and Europe of a “breadbin” C64 costed about half a salary of your father or even more, do you remember that?* And today, in the summer of 2020, after almost 40 years and despite the 17 million units sold worldwide, there are people who ask and obtain amounts around 250-300 EUR for the same model, if in good and working condition and possibly including the original box. If you just make the conversion from EUR to your national currency, you’ll get an amount not very far from the original sale price of many years ago. And if we were talking about other Commodore models (i.e. Amiga) or other brands less successful on the market then and consequently less easy to find today, the prices of classified retro-stuff would far exceed the original retail price.
So we are in presence of an upward price tendency that has kept going on, especially in the last ten years, with a trend of slow but continuous growth. Of course, these increases also attract "investors", who are keen to buy and resell later, speculators and even scammers, who, by exploiting the weakness of real enthusiasts and the security lacks of online sales systems, definitely contribute to the fact that the match between supply and demand can only be found higher and higher on the price scale.

Fortunately, some niches of real enthusiasts do resist. They are populated by those who care that their hobby does not die under the blows of blind market trends and still practice the direct exchange of hardware, without involving payments in money. Fortunately, most of the software emulators for our beloved systems are available for free or even open-source. Fortunately, new hardware products (mostly FPGA-based) that are inspired by or reproduce old machines come to market at quite affordable prices while offering an experience not unlike that of the original systems. Otherwise, despite the increasing number of people (re)approaching the world of 8- and 16-bit systems, home and micro computers or video game consoles, this passion of ours made of collecting, using, putting our hands back on the original hardware to save it from oblivion, would remain the prerogative of a few "well-off" people and in time would disappear completely.

Because it is the energy of passion we all share that keeps retrocomputing alive, not money.

David La Monaca

(*) USD 595, LIT 973,500, DM 1495, GBP 325: equivalent to about USD 1,576 in 2019 - source Wikipedia

SUMMARY

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

- Alberto Apostolo
- Giuseppe Fedele
- Michel Jean
- David La Monaca
- Gianluca Girelli
- Davide Bucci
- Giorgio Balestrieri
- Michele Ugolini
- Carlo N. Del Mar Pirazzini
- Federico Gori
- Marco Pistorio
- Querino Ialongo
- Daniele Brahimi
- Leonardo Miliani
- Flavio Soldani
- Francesco Fiorentini
- Graphic support Irene G. Valeri
- Cover Flavio Soldani
- Proof-reading Francesco Fiorentini, David La Monaca, Robin Jubber, Giorgio Balestrieri, Alberto Apostolo, Gianluca Girelli, Michele Ugolini

Interview with Giuseppe Ettore Pintus

Author of Freedom Fighter - Rise Of The Humans!

by Francesco Fiorentini

For a few weeks now, MSX users have been able to get their hands on a new game, which has struck everyone for its extremely fluid scrolling and difficulty. We are of course talking about Freedom Fighter - Rise of the humans!, created by the talented Giuseppe Ettore Pintus.
The game, which participates at the 2020 edition of the MSXDev, has all the cards to be one of the absolute protagonists of this competition. Its programmer is easily met in many Italian FB retro-groups and we at RMW could not miss the opportunity to ask him a few questions. A nice interview came out, full of interesting ideas and to be read in one breath!

Hello Giuseppe and thank you for accepting our invitation for this interview in the pages of Retro Magazine World. Before we start talking about your game 'Freedom Fighter - Rise of the humans', let's start with a few ritual questions to get to know the man before the programmer.

Do you want to tell us something about yourself? Who is Giuseppe Ettore Pintus and what does he do in his life?

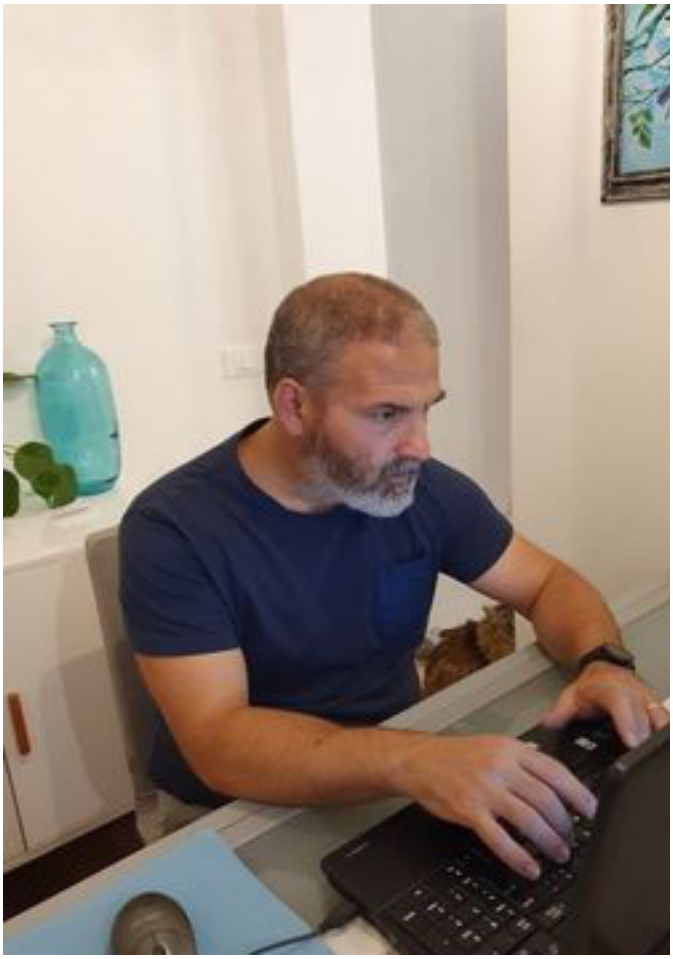


Figure 1 - Giuseppe Ettore Pintus working at his game

Hello everyone and thank you for the invitation. Giuseppe in life is first and foremost a husband: if you have Freedom Fighter in your hands you owe it to the patience of my wife, who supported (or endured?) my passion and my project. I am an electrician by profession (a bit reductive as a description, because I do not only deal with electrical systems, but more or less makes the idea).

The game you created is for the MSX standard, a computer that in Italy did not have the success of the Commodore 64 and Spectrum. Why did you approach this machine?

The MSX was the computer with which I grew up, unique among many Conmmodore 64 friends and some owners of Amstrad CPC (at the time the one distributed by Schneider together with a computer courses was popular on our side). I had a few "tape exchanges" with classmates of friends but then I met only a couple of msxists in person.

In case the MSX wasn't your first computer, what was it and what do you remember?

The first computer to enter my house was a Commodore Vic20. I still own and love the original of my childhood! The passion for programming was born with him. It came home on my 10th birthday. It arrived "naked and raw." No games, no tape. So every day I used to type one (but sometimes all) of the three listings at the bottom of the manual (they were three basic games of Duane Later). How many syntax errors in that period! Then at dinner time I turned off the Vic, and since I couldn't record, I knew I'd have to do it all over again the next day!

How did you get to know the world of programming? Are you a self-taught person, like many of us in the 80s/ 90s, or have you taken any special training courses?

Absolutely self-taught. Like I said, it all started with Vic20. By typing, reading and re-reading the manual (and all the examples contained) codicng became familiar. Then over time we bought the datassette and the first magazines. I used to copy and study lists. When I put the Vic-20 aside to replace it with the MSX, I had three/four BASIC games in place that I designed and programmed from scratch: a Breakout clone, a ten-line-like racing game, a PacLand platform (with single screen scrolling) and a racing game similar to the GiG Game & Watch. I continued with THE MSX BASIC (and bought the assembly book that I never

used in the past because I couldn't find an assembler program anywhere!) and then moved on to Amiga. I did a lot of things in Amos, including a complete RPG engine similar to Eye of the Beholder (but more fluid than the Amiga version, more similar to Black Crypt to be clear)

What is your favorite programming language back and for what reason?

At the moment the Z80 assembly, I have programmed my first official and international game!

What other computers do you like to program besides MSX?

All the ones I mentioned earlier (I got caught in the heat).

How do you approach retro-computing? Do you own many retro machines or prefer emulators and/or FPGAs?

At the moment I own a VCS2600 Jr (loose and under repair), a Boxed Vic20 Commodore, 3 MSX VG8020 (one of them under repair, the first one I had) a Amiga 500+ (mine, also under repair due to Varta curse...) and recently I bought a MSX2 VG8235 (the dream I had before the Amiga). I use emulators for convenience, especially to test the software quickly...

Let's talk a little bit about 'Freedom Fighter - Rise of the humans', a vertical scrolling shoot'em up, why did you choose this format for your game?

Eh! Actually, I just wanted to program a smooth vertical scroll to show that the MSX can do it (and that I can do it). But once the almost final version of the scrolling routine was ready, after presenting it on the MSX Resource Center forum, I was pushed by the other coders to move forward. I didn't even know which way to start and which cross assembler to use: I started using an assembler written in basic on emulated MSX, which they provided in the first

Figure 2 - Freedom Fighter's intro screen

cassette of the magazine C16/MSX in newsstand...

What games inspired you to create Freedom Fighter?

Definitely the first one I can think of is Zanac from Compile. A milestone in MSX shooting 'em up. In fact, many users during the development compared the two games. I paid tribute to Zanac in the first level music (derived from the original one) and in one of the three secrets of the game. Unlocking the second secret, in fact, you can change your spaceship with one of the most famous shooting 'em up for MSX: Zanac, Twinbee, Star Soldier, Star Force and Hype.

What difficulties did you encounter while writing the code?

Debugging an assembly project on your own, especially if it's your first project, you've learned all the way down the road and started directly with an ambitious megarom project (1 Megabit, i.e. 128KB of cartridge game) is not a piece of cake. For the rest, when I had some doubts I had an entire community of coders to ask for advice!

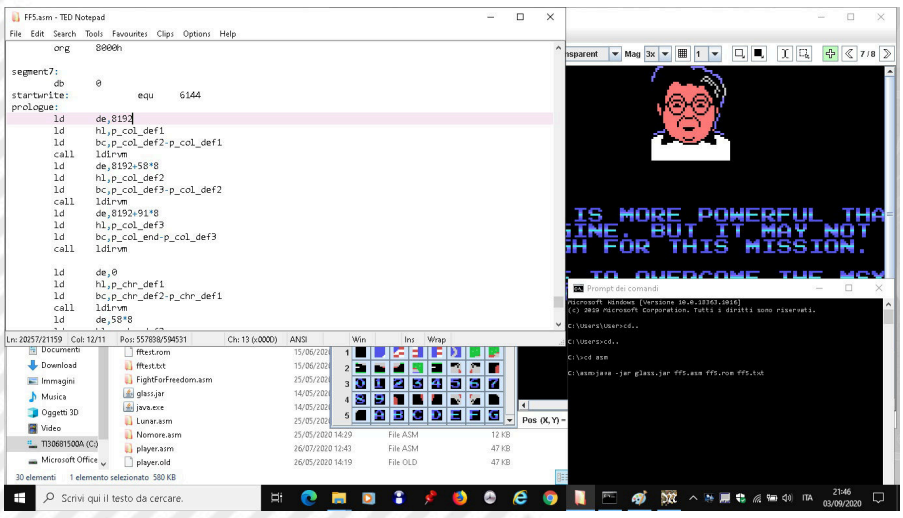


Figure 3 - Freedom Fighter's developing environment

I have not a deep knowledge of MSX programming, but I often read about scrolling problems on this machine. By contrary your game is very fluid, how did you achieve this result?

It's not that the MSX has scrolling problems... It doesn't have scrolling at all! It is a feature that its video chip (meaning the same as the TI99/4A and Colecovision) does not have. So everything that flows is purely a software-based trick. You need to know well the hardware you're dealing

Figure 4 - first level of Freedom Fighter

with.

If you could go back and approach work with hindsight, what would you do differently?

I don't know if I'd change anything. I had a great time making it happen. Perhaps I would directly avoid accepting the help offered by doubtful people (who in fact were excluded from the project) and perhaps I would have looked for some MSX artists in the community above (although I seem to have managed discreetly with all that graphics and code!). What I'm sure I'd do again is work with Phaze101 on the audio part. It was a fantastic experience, I think it came very close to what it had to be like working in a software house in the 80s (...or at least as I imagined it).

I noticed that you listen carefully to the feedbacks you receive from players and that you have posted several updates to your game. Some of these were aimed at lowering Freedom Fighter's initial difficulty. Did you deliberately create a game that was difficult to be completed to pay homage to the format of the 80s games?

Yeah, I initially thought, "Hey! 8-bit games in the '80s were frighteningly difficult! I need to recreate that feeling. Besides, it's only five levels. If I make it too easy, it'll be over in no time. Besides, I played and finished the first level quietly without losing a life. But then I read THAT EVERYONE really found it too difficult, so I listened to the players' voice and started smoothing here and there...

We have arrived at the funny moment of the interview: "Ask yourself a question and give yourself an answer".

“Who made me do it? ” can that be okay as a question? The answer is for everyone: boys (grown up, in fact, since you are lovers of retrocomputing), passions must be followed. Took me two years to develop the game. But it was my first game. The assembly was an unknown territory to me but I really wanted to do it. Do it, too! If you want to program and you don't know how to do it, you simply have to start. You can also receive satisfaction from the very slow BASIC. You learn to think as a programmer, to solve problems, to exploit hardware. With the current “widespread knowledge” between coders to ask and books of the past to download in pdf, manuals and everything that once was not there, you can really do it. Do it! Do it!

You can also join our Facebook group, Retro Programmers Inside, here:

https://www.facebook.com/groups/RetroProgrammersInside

Before I leave you, I'd like to thank you again for your time and especially for releasing your freeware game to the community.

Good luck with your participation in MSXDev 2020!

Thank you and long live the MSX!

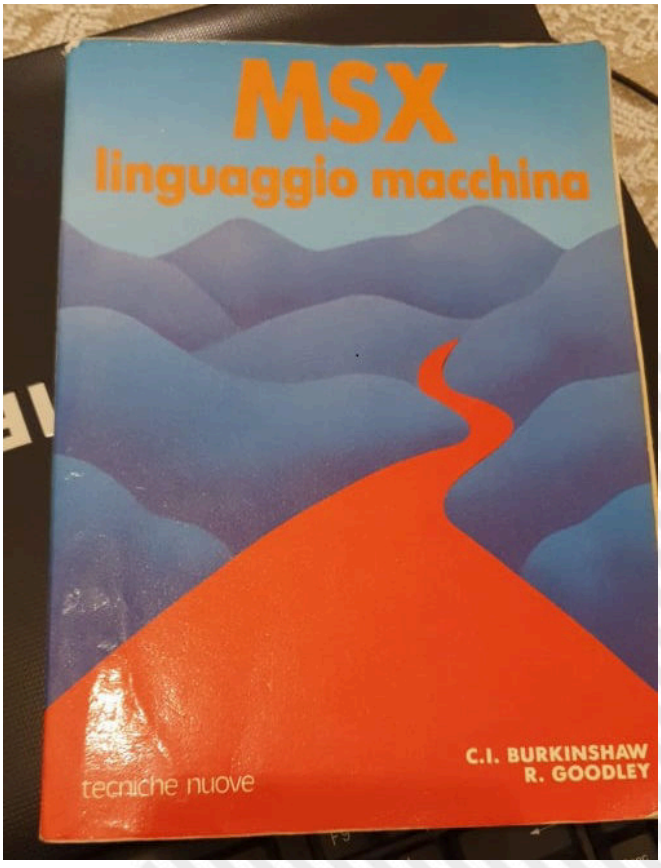


Figure 5 - An MSX Assembly book

How to unprotect GW-BASIC programs

by Alberto Apostolo

Reading Giuseppe Fedele's brilliant article on Cryptography (RMW #23-IT, RMW #01-EN), I remembered when (in 1998) I was commissioned to carry out the porting of some old programs on behalf of a company that produced textile machines in Pordenone (Italy).
The programs had been written to GW-BASIC by an employee of that company, but the paper lists had been lost and saving with the P protection option made the source files unreadable.

At the time, I had not been able to find documentation on how to carry out the unprotection (having been authorized by the owners of the software, I operated in an absolutely legal way). I could only dump the source files with the good old PCTOOLS.EXE on my 386 to try to figure out what the encoding was.

A brute-force solution

Imitating the legendary British cryptographers of Bletchley Park, I had to try to exploit a weakness of many encodings and observe the behaviour when dealing with a very long string formed by the same character.
As a first test case, I tried to securely save a simple program (List 1) and display it with a Hex Editor (in this article I used the free HxD program).
Figure 1 shows that the character "A" is coded differently depending on the position in the row. At the same time it is noticeable that after 143 bytes the encoding is repeated (a real stroke of luck!).
Repeating the same exercise for each character you can obtain a decoding matrix 256 x 143 (in Figure 2 you can see only a small part). Since there are much simpler and more elegant methods, I will spare you the publication of the programmes written at the time

```
1 REM AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA
```

Listing 1

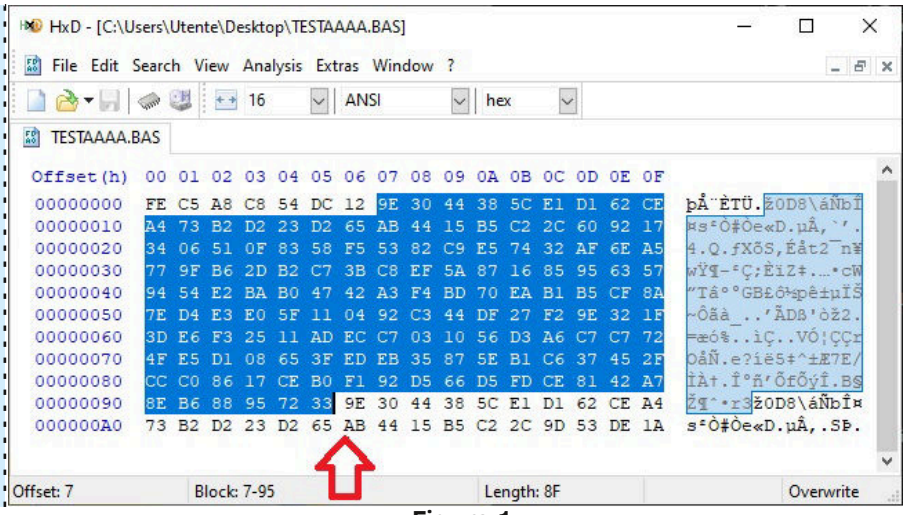


Figure 1

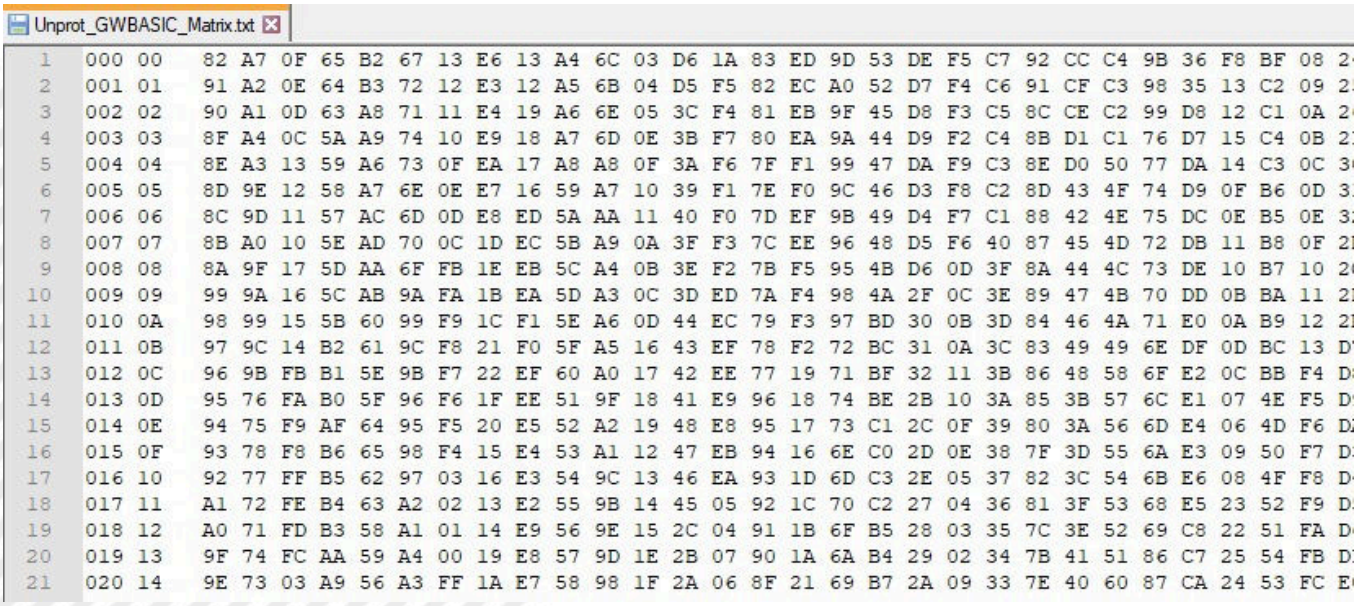


Figure 2

in COBOL that allowed me to arrive at the result.

Solution with a thrust of foil

By searching the Internet nowadays, you can find web pages on any topic. So I was able to experiment with a simple trick to unprotect a program by "tricking" the GW-BASIC interpreter.

First you need to create a dummy program in GW-BASIC consisting of only two bytes: 0FFh and 1Ah. In my case I used the Hex Editor HxD (Figure 3).

Then from the interpreter it is necessary to perform the sequence of operations (Figure 4):

1) load the protected program,
2) load the dummy program,
3) save the program (which will now be unprotected).

The deception occurs because the first byte of a saved program is 0FFh if the program is unprotected (with the keywords "tokenized"), 0FEh if it is encoded. The dummy program only overwrites the first byte and GWBASIC will diligently decode the subsequent instructions in memory.

This simple trick has the disadvantage that can be applied on one program at a time, activating the GW-BASIC interpreter.

The mystery revealed

While searching for sources to write this article, I found an interesting post written by Christophe Lenclud [Len18] showing the decryption system used by GW-BASIC.

The XOR between each byte of the program and two keys embedded in GWBASIC.EXE file applies. The keys are 13 bytes long and 11 bytes long respectively, so encryption is repeated every 13 x 11 = 143 bytes (List 2).

In [Len18] there is a (incomplete) list of the versions on which this system is applied that is based on the search of the keys within the executable file of the various BASIC interpreters:

1) BASICA.EXE (size 54272 bytes, 13 May 1983 12:00:00, MD5 = 28E22CAA7EC534A78D37AA3314690758) from "The COMPAQ Personal Computer DOS, Version 1.11" Rev E.

2) GWBASIC.EXE (size 59728 bytes, 05 June 1984 01:25:00, MD5 = 2FB3EB25944C27267626836435DE7369) "BASIC

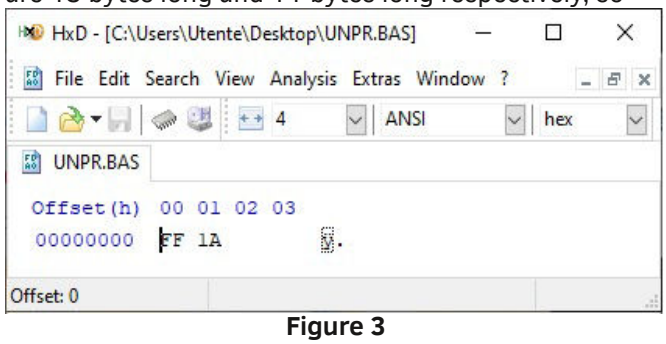


Figure 3

```
; Input: DS:SI -> Data to be deciphered.
;          DS:DX -> After end of data.
Decipher_GWBASIC proc near
    mov     cx, 0D0Bh
    mov     di,si
    mov     bh,0
    cld
@@LoopDecipher:
    cmp     si,dx
    jz      short @@EndOfFile
    ; Decipher one byte...
    mov     bl,ch
    lodsb
    sub     al,cl
    xor     al, [bx + offset Key1 - 1]
    mov     bl,cl
    xor     al, [bx + offset Key2 - 1]
    add     al,ch
    stosb
    ; Next byte...
    dec     cl
    jnz     short @@NotZ1
    mov     cl,0Bh
@@NotZ1:
    dec     ch
    jnz     @@LoopDecipher
    mov     ch,0Dh
    jmp     @@LoopDecipher
@@EndOfFile:
    ret
Decipher_GWBASIC endp

Key1    db    9Ah, 0F7h, 19h, 83h,  24h, 63h,
43h, 83h,  75h, 0CDh, 8Dh, 84h, 0A9h
Key2    db    7Ch,  88h, 59h, 74h, 0E0h, 97h,
26h, 77h, 0C4h,  1Dh, 1Eh
```

Listing 2

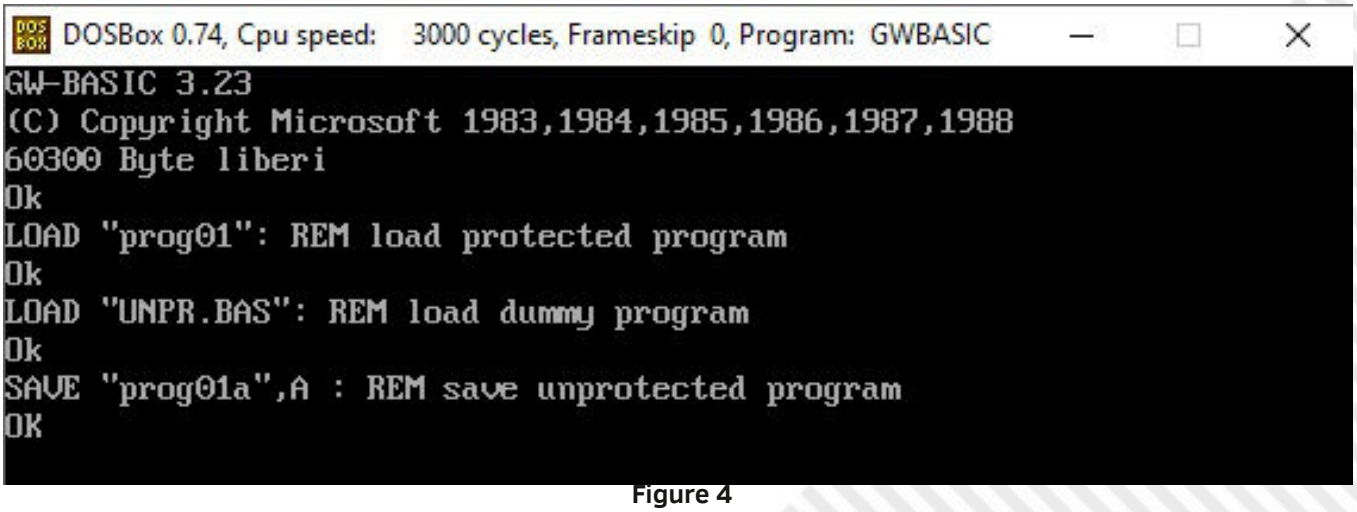


Figure 4

Interpreter - Version 1.12.03 - Copyright (C) 1984 Corona Data Systems, Inc" from MS-DOS 1.25.
3) Floppy disk images of Compaq MS-DOS 1.10, 1.11, 1.12, 3.00, 3.31.
4) Floppy disk images of MS-DOS 1.25, 2.11, 3.10, 3.30.

Instead, encryption takes place with a similar system where SUB and ADD registers are inverted [Kit18]:
1) SUB AL, CH
2) XOR AL,Key1(pos. CH)
3) XOR AL,Key2 (pos. CL)
4) ADD AL, CL

Conclusions

Listing 3 contains my GW-BASIC version of the decryption algorithm (with some adjustments due to simulation of the AL registry with GW-BASIC).
The unprotected file is still "tokenized" but you can upload and then save with option A of conversion to text mode.

Lenclud does not say this, but decryption starts from the second byte onwards (the first one is reserved for the encoding flag).

```
1000 REM UNPROT2.BAS
1010 REM UNPROTECTING GW-BASIC PROGRAMS
1020 DIM K1(13):DIM K2(11)
1030 FOR J=1 TO 13:READ K1(J):NEXT J
1040 FOR J=1 TO 11:READ K2(J):NEXT J
1050 REM NAME OF PROTECTED PROGRAM
1060 OPEN "test1.BAS" FOR INPUT AS #1
1070 REM NAME OF UNPROTECTED PROGRAM
1080 OPEN "test1u.BAS" FOR OUTPUT AS #2
1090 CH=13:CL=11
1100 AL=ASC(INPUT$(1,#1))          :REM SKIP FIRST BYTE
1110 PRINT #2,USING"!";CHR$(255); :REM WRITE 0FFhex
1120 IF EOF(1) THEN 1280
1130 AL=ASC(INPUT$(1,#1))
1140 AL=AL-CL                      :REM SUB AL,CL
1150 IF AL < 0 THEN AL=AL+256:REM IT'S BASIC NOT ASSEMBLY
1160 U=AL:V=K1(CH):GOSUB 1310:REM XOR AL,K1(CH)
1170 AL = X                        :
1180 U=AL:V=K2(CL):GOSUB 1310:REM XOR AL,K2(CL)
1190 AL = X                        :
1200 AL = AL + CH                  :REM ADD AL,CH
1210 AL = AL MOD 256               :REM IT'S BASIC NOT ASSEMBLY
1220 PRINT #2,USING"!";CHR$(AL);
1230 CL=CL-1
1240 IF CL = 0 THEN CL=11
1250 CH=CH-1
1260 IF CH = 0 THEN CH=13
1270 GOTO 1120
1280 CLOSE #1
1290 CLOSE #2
1300 END
1310 REM X = U XOR V
1320 X=0
1330 FOR J=0 TO 7
1340 BITU = U MOD 2:BITV = V MOD 2
1350 X = X + (BITU-BITV)*(BITU-BITV)*(2^J)
1360 U=INT(U/2):V=INT(V/2)
1370 NEXT J
1380 RETURN
1390 DATA 154,247,25,131,36,99,67,131,117,205,141,132,169
1400 DATA 124,136,89,116,224,151,38,119,196,29,30
```

Listing 3

Bibliography

[Bas85] AA.VV. "GW-BASIC User's Manual", 1985.
[Kit18] S.Kitt, "How were Microsoft GW-BASIC “protected” files encoded?",
https://retrocomputing.stackexchange.com/questions/7104/how-were-microsoft-gw-basic-protected-files-encoded
lastly consulted on 23.05.2020.
[Len18] C.Lenclud, "Deciphering GW-BASIC / BASICA protected programs",
https://slions.net/threads/deciphering-gw-basic-basica-protected-programs.50/
lastly consulted on 23.05.2020.

RetroMath: How to transform an image...

by Giuseppe Fedele

In computer graphics, many applications require to manipulate an image by changing, for example, its size, position, and orientation. This can be done by applying a geometric transformation to the coordinate points of the image. In this article we want to analyze some basic geometric transformations in 2D space.

Types of transformation

Translation

One of the simplest transformations is to move the image to a new position. In Fig. 1, the points of the original image are plotted in black

$$P_1=\begin{bmatrix}2\\-5\end{bmatrix} P_2=\begin{bmatrix}3\\6\end{bmatrix} P_3=\begin{bmatrix}7\\10\end{bmatrix}$$

while the corresponding translated points are plotted in red.

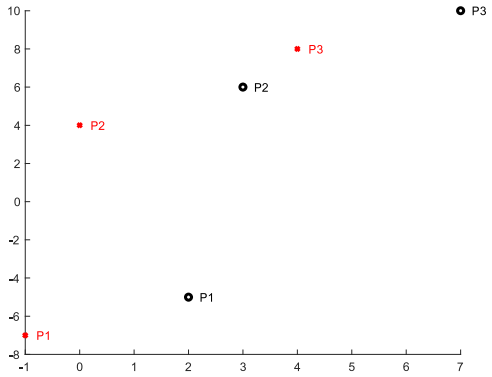


Figure 1. Transfer.

In order to translate a point it is necessary to add a constant value to *the x* and *y* coordinates of the point, as shown in Fig. 2. The new coordinates of the point are given by

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$

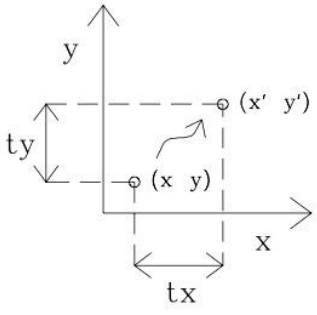


Figure 2. Move a point.

Scaling

A scaling transformation is used to change the size of an object. Scaling about the origin of the axis of the *xy* plane is achieved by multiplying each component of the vector by a scale factor:

$$\begin{cases} x' = x \cdot s_x \\ y' = y \cdot s_y \end{cases}$$

If ¿s_x∨¿ and¿s_y∨¿, i.e. the scale factor modules, are both greater than 1, then the effect is of increasing the size of the object, while a reduction of the image is obtained if ¿s_x∨¿ and¿s_y∨¿ are less than 1. In Fig. 3, for example, the vector from the origin of the plane to the point P_1 is scaled using the factors $s_x = \frac{1}{4}, s_y = \frac{1}{2}$ (size reduction) while the vector relative to the point P_2 is scaled with $s_x = \frac{3}{2}, s_y = \frac{6}{5}$ (size increase).

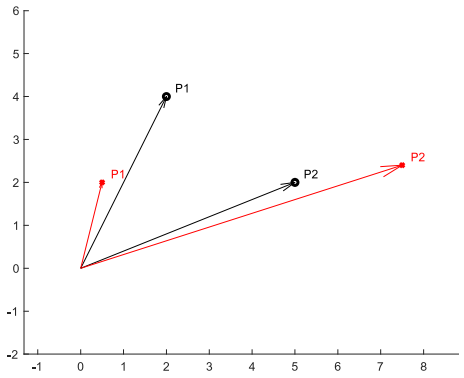


Figure 3. Scaling.

If the scale factors are the same, $s_x = s_y$then we have a symmetrical scaling, i.e. the image is reduced or expanded by the same amount in each direction (Fig. 4).

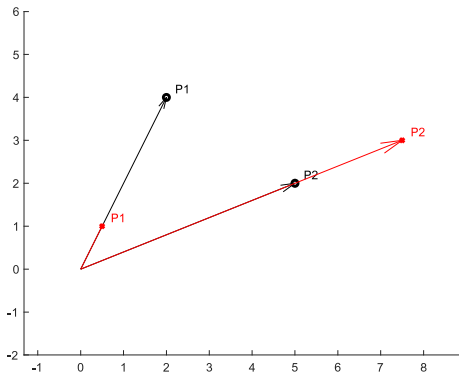


Figure 4. Symmetrical scaling.

If the scale factor in x is negative ($s_x<0$), then the object is reflected with respect to the y axis. Similarly if $s_y<0$then the object is reflected with respect to the x axis. In Fig. 5, the vector relative to P_1 has scaling factors $s_x=\frac{-1}{2}, s_y=1$ while the vector relative to P_2 has factors $s_x=1, s_y=\frac{-1}{2}$.

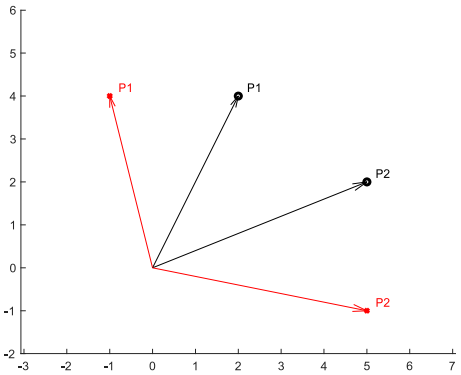


Figure 5. Scaling with negative scale factors.

Rotation

Another common type of transformation is rotation, which is often used to orient objects. Rotation of a point of the object is shown in Fig. 6.

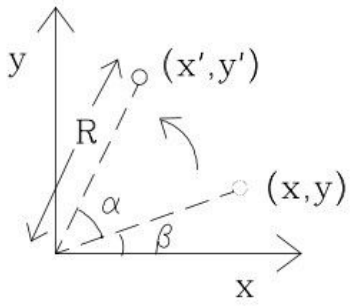


Figure 6. Rotate a point.

The segment of length R joining the point (x, y) with the origin, forms an angle β with the x axis, and then:

$$\begin{cases} x=R\cos(\beta) \\ y=R\sin(\beta) \end{cases}$$

After rotation of an angle α, the new point has coordinates

$$\begin{cases} x'=R\cos(\alpha+\beta) \\ y'=R\sin(\alpha+\beta) \end{cases}$$

Expanding the trigonometric terms and replacing the values of x and y, gives:

$$\begin{cases} x'=R\cos(\alpha)\cos(\beta)-R\sin(\alpha)\sin(\beta)=x\cos(\alpha)-y\sin(\alpha) \\ y'=R\sin(\alpha)\cos(\beta)+R\cos(\alpha)\sin(\beta)=x\sin(\alpha)+y\cos(\alpha) \end{cases}$$

Shearing

This transformation has the effect of distorting the shape of an object. The new coordinates of a shearing point are given by:

$$\begin{cases} x' = x + a\,y \\ y' = y + b\,x \end{cases}$$

If $b=0$, the effect is to distort the image by translating the coordinate x as a function of the height of the object (with $a>0$, as the ordinate of the point increases, the object undergoes greater distortion on the x axis). Fig. 7 shows a distortion along the x axis with a shearing factor $a=\frac{3}{2}$.

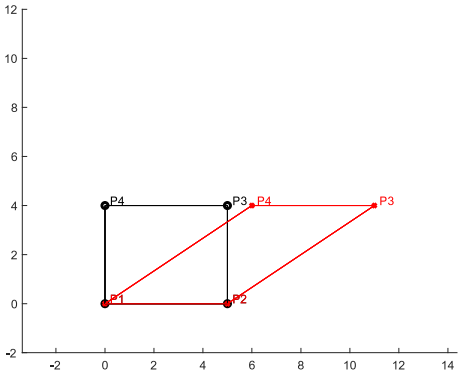


Figure 7. Shearing along the x-axis.

Similarly, a distortion along only the axis y is obtained by choosing $a=0$ and a value of $b>0$ (Fig. 8).

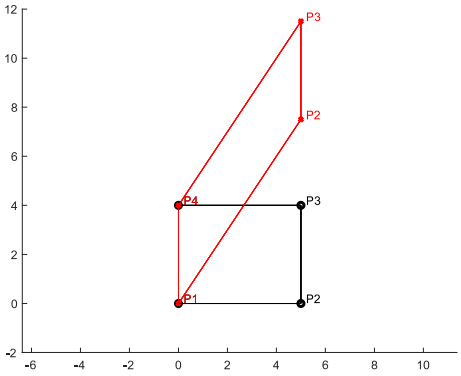


Figure 8. Shearing along the y-axis.

Fig. 9 instead shows a distortion on both axes with $a=\frac{3}{2}, b=\frac{4}{3}$.

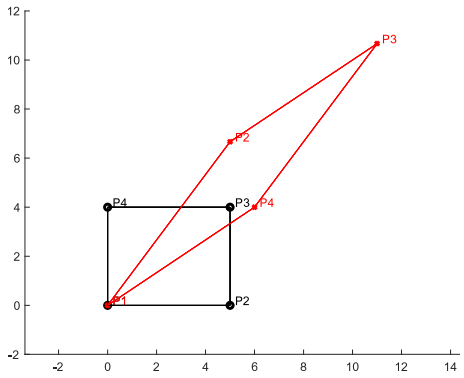


Figure 9. Shearing on the x and y axes.

Matrix representation

A general formula that includes all of the transformations discussed above is

$$\begin{cases} x' = a\,x + b\,y + c \\ y' = d\,x + e\,y + d \end{cases}$$

where a, b, c, d, e, f are constants. With the artifice of inserting a fictitious equation, the above formula can be rewritten, in matrix form, as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The four transformations can be rewritten with this notation as:

- Transfer

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Rotation

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Shear

$$\begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Obviously the matrix that leaves the object unchanged is the identity matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since the matrix product is not commutative, i.e. $M_1 M_2 \neq M_2 M_1$, then the order in which the transformations are performed is important. Fig. 10 shows, on the left, the application of a translation and then a rotation with $t_x = 2, t_y = 4, \alpha = \frac{\pi}{4}$, while, on the right, the effect obtained with the inverted order of the transformations.

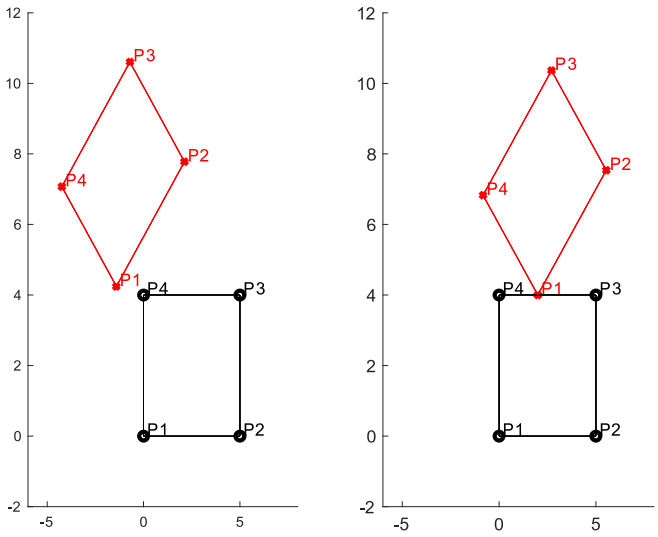


Figure 10. Non-commutativity of transformations.

Bibliography

[1] J. Vince, Mathematics for Computer Graphics, fifth ed. Springer, 2017.

[2] D. Kothari, G. Awari, D. Shrimankar, A. Bhende, Mathematics for Computer Graphics and Game Programming, Mercury Learning and Information, 2019.

FORTH: the secret weapon!

di Michel Jean

In September 1982, the British company Jupiter Cantab, founded by two members of the Sinclair Company staff, launched the **Jupiter Ace**, a direct competitor to their former company's ZX-81 and Spectrum.

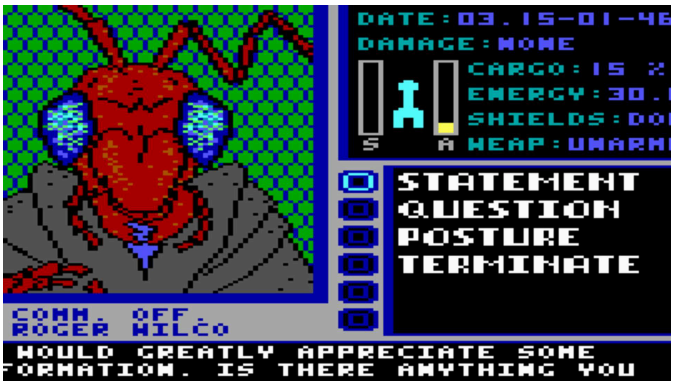
The Ace has a similar case to the Spectrum, the same microprocessor, but the small company has a joker in hand: it is programmable in **Forth**.
While BASIC is the language used by all the competitors at this time, the British company was taking the bet of distributing its machine with the Forth language. This language at occupies half the memory space and allows an execution six to ten times faster than BASIC, making the Jupiter Ace the most efficient machine in it's price range.

At the time, Forth is not widely used in microcomputing, but it's not totally unknown.

Already in 1980, Byte magazine devoted an issue to it. It suggests that the Atari company is developing a version that will allow them to develop their arcade games faster : «(...) Atari has developed its own custom version of the language, called game-FORTH, that is awaiting its first use to replace machine code as the language used to create arcade games. Someday soon, you may play a coin-operated game without knowing that you are actually running a FORTH program».

This statement gave birth to the legend, still circulating today in the Forth community, that Forth would be Atari's secret weapon. Although, there is no evidence that major video game production companies made extensive and secret use of Forth , several software have taken advantage of the speed of development that Forth allows.

For example, it is documented that the game **Starflight**, published by Electronic Art in 1986, was developed in Forth: "The team coded the game mostly in Forth with a few key routines written in x86 assembler.

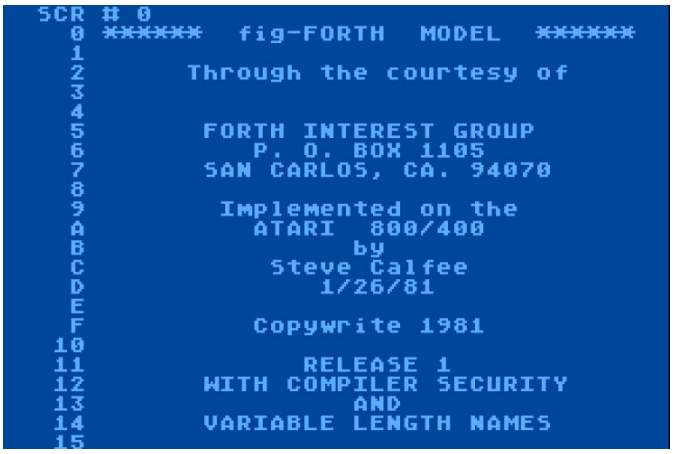


Forth was chosen since it is easier to use than assembler and more compact. This was important because the game had to fit into 128K of RAM ».

But what is Forth? Forth is a language developed in the sixties by Charles H. Moore. In charge of calculating satellite trajectories for an observatory, Moore was looking to develop a toolbox that would allow him to facilitate his daily work. After several years of maturation, this toolbox became, in the early seventies, a powerful, elegant and addictive language.

Forth's strength is that it is only a core that can be easily modified and improved according to its needs, similar to Lego blocks building the shape we want. The best argument in favour of Forth is its use and I will present some elements of the langage. The goal is not to provide a programming course, but to give an overview of the possibilities of the language. For a more complete initiation, I advise the essential: **Starting Forth** by **Leo Brodie** available for free online .

There are Forth implementations for literally any platform. Whether it's a Commodore 64, a Atari 800,



a PDP-11 or your Linux 64 bit. Contrary to what many would like to believe, Forth is a fairly simple language, but to approach it we have to leave aside many reflexes that we developed with other languages. Forth is a compiled and interpreted language. So, a bit like in BASIC, we can test each function or procedure (a word in Forth) directly in the interpreter. But that's where the comparison ends, no GOTO, no line number in Forth.

Forth is a very different language from Basic, Fortran, Pascal, C++ or Java. It is closer to Lisp, APL, Prolog or Smaltalk. Although it looks like a functional language, we can't define it that way, not everything is function in Forth. It's a language where programming consists in building a software tool from "primitives". These tools are then used to create new ones, and so on until the application itself. Two other fundamental characteristics distinguish Forth from traditional languages. First, the intensive use of the stack, which avoids creating a multitude of variables and constants. Second, the fact that it works in Reverse Polish Notation (RPN).
This notation which makes the notoriety of the HP calculators of the 70s and 80s. For example, if we want to add 12 + 5 we will enter to the interpreter :
12 5 + .
12 and 5 are then introduced in the stack, the [+] being a function (in Forth we would say a Word) which takes the last two elements of the stack to add them and returns the result in the stack. The point [.] then displays the element at the top of the stack.

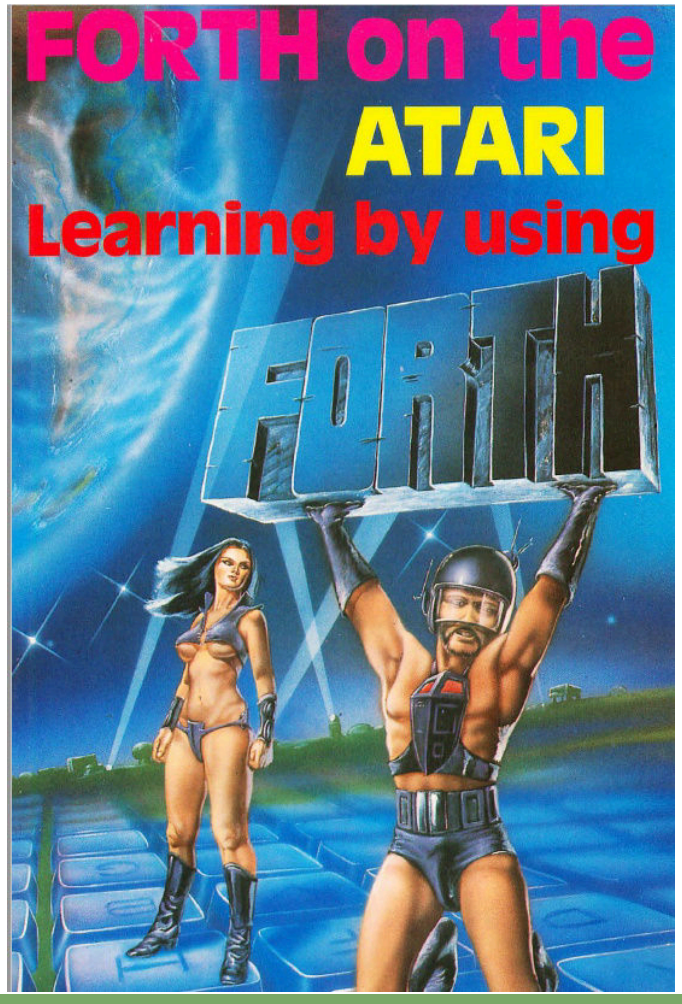
This may seem complex, but, as HP calculator users know, it quickly becomes natural and saves the need for parenthesis and questioning the prioritization of operations. The expression 2 * (3 + 6) becoming 2 3 6 + * The addition is applied to 3 and 6, 9 is then sent to the stack, the multiplication is then applied to 2 and 9.

Forth defines a wide variety of special operators to manipulate the stack data, to reorganize it (ROT, SWAP), to delete elements (DROP), to generate elements (DUP, OVER).

For example, the following expression calculates the square of 10 using the word DUP which duplicates the value at the top of the stack and sends the result to the stack:
10 DUP *

We could use this piece of code to create a new operator called SQUARE. This would be done by telling the compiler that we want to define a new word using the colon [:] followed by the name of our word. We will indicate the end of the compilation with the semicolon [;] .
Thus we get :
: SQUARE DUP * ;

We then defined a new word that will take the first element of the stack, duplicate it and multiply it by itself. So,
4 SQUARE
will send 16 into the stack.



Why stop here ? From this new word, I can create another one : CUBE

: CUBE DUP SQUARE * ;

4 CUBE will send 64 into the stack. (Ex.1)

```
OK
: SQR DUP * ;

OK
4 SQR .
16
OK
: CUBE DUP SQR * ;

OK
4 CUBE .
64
OK
```

Forthers generally favour short definitions, and a short definition can be surprisingly effective. Here Euclid's algorithm for determining the most greatest common divisor (GCD) fits in a single line. (Ex.2)

: GCD BEGIN 2DUP MOD ROT DROP DUP 0 = UNTIL DROP . ;

```
OK
: GCD BEGIN 2DUP MOD ROT DROP DUP 0 =
  UNTIL DROP . ;

OK
343 280 GCD
7
OK

578 442 GCD
34
OK

4144 7696 GCD
592
OK
```

Of course, we could criticize the lack of readability of this line of code, but nothing prevents us from documenting our code, which is usually done by parentheses, mainly by clearly indicating the effect of the Word on the stack.

In fact, such an obscure line should never be found in a good Forth code. As an exercise, we leave it to the reader to analyze this line of code, but you can see one of the many ways to introduce a conditionnal loop in Forth with BEGIN and UNTIL.

Data manipulation is not only done by using the stack. Forth also allows the use of variables and constants, but here again the language offers a flexibility that can be found almost exclusively in assembler. Thus, let's declare a variable (with the word Forth VARIABLE)

VARIABLE myvariable

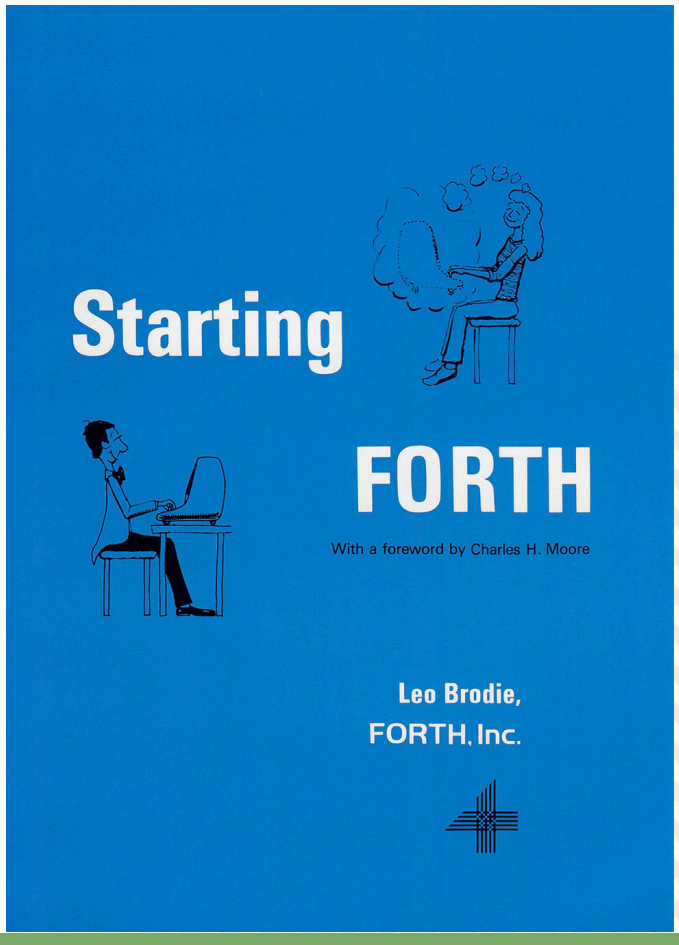
the word [!] associates a value for this variable.

36 myvariable !

It is important to realize the distinction between the address of the variable and the content of the address. If I type the name of my variable in the interpreter, what will be sent to the stack is the address of the variable. To get the content of the variable I use **@** (called fetch in Forth). Thus [myvariable @ .] gives the content which is found at the myvariable memory address, in this example 36.

You probably now have an idea of what we meant when we were talking about a toolbox. You certainly also get a glimpse of the continent that Forth programming opens up for us. Forth never had the notoriety of C, Pascal or BASIC. The Jupier Ace did not have the commercial success that these creators would have hoped for. But many of those who experimented Forth have developed an addiction for this langage and the Forth community is still very much alive.

Forth may not have been Atari's secret weapon, but it may be yours.



MiniGrafik: a graphical extension for the Vic-20 BASIC

by David La Monaca

One of the most refined amusements of us geeks and retro-fans is looking for so-called "hacks" on our old dear systems. Computer hacks are particular software solutions or pieces of code that, taking advantage of some hardware features, can solve a certain problem to speed up or simplify a process or to implement striking functions and eye-catching applications, usually something unexpected for a machine known for its limitations. This is the beauty of software, the "magic" that sometimes manages to overcome the inner limits of a hardware architecture. YouTube and some other sites collecting demos and intros are full of these hacks. A real-time colour video on the Atari 800XL? Done.[1] A chess program on ZX81 with only 1K of RAM? Got it![2] More than 16 colours simultaneously on a C64 bitmap image? Seen that too![3] What about Doom running on an unexpanded Vic-20? Beautiful![4] And a .MOD track that plays on Impulse Tracker and runs the Bad Apple animation on the four tracks? What a fantastic idea![5]

Well, I could go on for a while. The list could be very long and it could easily be the subject of a next, tasty article for RMW. Sometimes, in the category of “computer hacks” end up single pieces of code written in assembly or other languages (even in BASIC) that solve specific problems or constitute useful and effective solutions. Creativity in writing software, if it comes from right and wise hands, can result in small or big masterpieces. However, only experts and lovers of the art of coding can really appreciate and recognize these true strokes of genius, because they often are the result of a deep knowledge of a computer hardware, a single processor or an audio / video chip, a programming language or an entire system.

MINIGRAFIK for the Vic-20

In the category of hacks being at the same time surprising and effective falls a project by Michael Kircher, a German engineer, universally recognized as a sort of guru of the Commodore Vic-20, author/coder of many programs and games and among the most active users of Denial, the famous forum all dedicated to this little machine. Michael is also the author and maintainer of a graphics library (and its complementary high resolution pixel editor) that can be easily defined a "top solution" for the little Vic. Much more than that: Minigrafik (this is the name given to the library that extends the infamous Commodore BASIC V2 with some graphic instructions) is a real "software hack" providing the Vic-20 with primitive graphics and convenient instructions to fill the 160x192 screen with many beautiful coloured pixels to draw bitmap images, graphics and 2D/3D functions, design games backgrounds and more. A full-screen editor called MiniPaint is the practical complement to the MiniGrafik library. It allows you to draw "freehand", select colours and shapes to create backgrounds and bitmap images to be used in your games and applications or export them to a reusable format. MiniPaint itself is actually an application written partly in BASIC extended with MiniGrafik, which has now become a standard for creating BASIC or Assembly games and programs that make use of high or medium resolution graphics mode.

Figure 1. Minigrafik's logo, created with Minigrafik

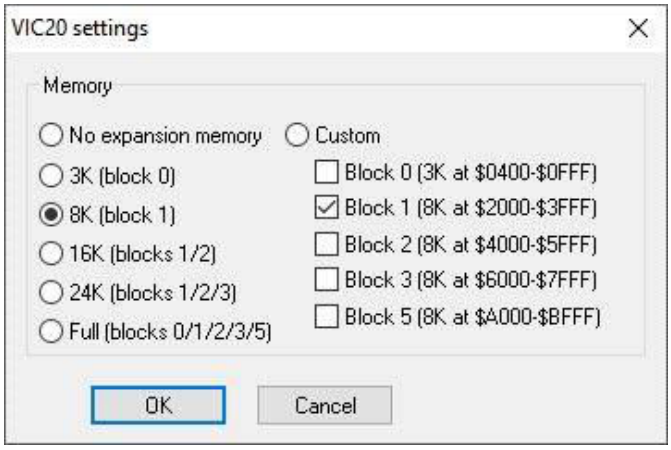


Figure 2. An 8K RAM expansion is needed to run MG

To extend the BASIC of the Vic-20 with Minigrafik all you need is 8K of RAM expansion and a floppy disk drive or a D64 [MG] image. Then you just load the library and run it. Once launched, the extension will automatically load into RAM, allocate the graphics memory and return control to BASIC by changing the opening message and updating the number of free bytes (a few less, of course). Alternatively you can use a boot loader, which in sequence loads Minigrafik, initializes the extension without returning to the boot screen and then proceeds to load a client program (application or game) that needs the graphics library.

Figure 3. The Vic-20 table of colors (0-15)

Using this method, the Minigrafik implementation becomes very simple: as long as the D64 disk image contains a copy of the library and all programs and games making use of MG can easily be launched after loading the extension. This modular system is much more comfortable than the method of embedding the library into every program. Any extension updates can also be done more quickly and effortlessly.

The Minigrafik extension consists of a single .PRG file that can be loaded from disk with a simple command: LOAD "MINIGRAFIK",8. After the RUN command, the CBM BASIC start message will appear again. As said, the number of bytes available for your BASIC programs will be reduced by the amount allocated for the bitmap screen and the extension itself, which basically adds 6 new custom commands and a function to the BASIC interpreter. All new instructions are preceded by the '@' character and should only be used in program mode and not as direct commands from the BASIC prompt. This is because normal text output can interfere with the high-resolution screen.

Figure 4. An image created with MiniPaint

Also, when writing code, after a THEN of an IF construct, you must be careful to always use the colon ':' before a Minigrafik command, otherwise the program will stop with an annoying '?SYNTAX ERROR'.

The additional commands

The new commands that MG adds to BASIC are: @ON, @CLR, @RETURN, @SAVE and @LOAD. The library is completed by the @() function that returns the status and color of a single pixel. Let's quickly see its features and usage:

@ON initializes bitmap mode at 160x192 pixel resolution, correctly centering the screen for both VIC chips, NTSC and PAL.

@CLR clears the screen in high resolution. The colour

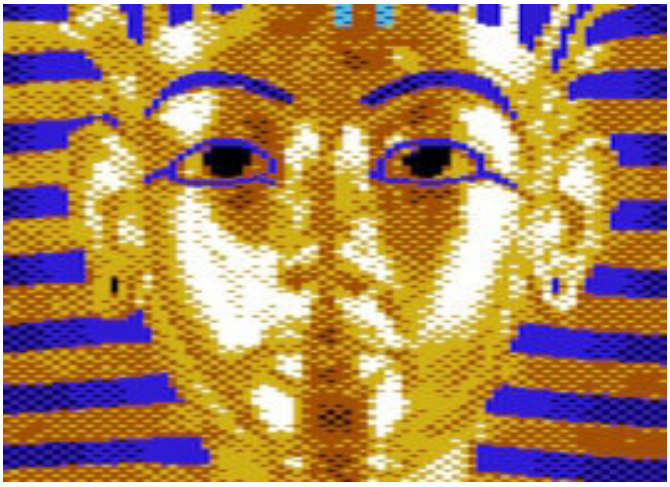


Figure 5. Another image created with MiniPaint

RAM is initialized to the foreground colour.

@RETURN returns to text mode. If an error occurs while running a program, this command is automatically executed before printing the error message on the screen.

@<color>,<x1>,<y1> [TO <x2>,<y2>] - draws a line with the color indicated by the screen coordinates x1,y1 to the coordinates x2,y2. If the TO part is omitted, a single pixel at the x1,y1 coordinates is plotted. The values of x1 and x2 range from 0 to 159, while y1 and y2 range from 0 to 191. The screen origin is fixed in the upper left corner. All arguments can be entered as numbers, variables or numerical expressions.

In medium resolution (or multi-color mode) the resolution is halved and therefore for all x even, x and x+1 coordinates indicate the same pixel. The foreground color (0 to 7) are set individually for each 8x16 pixel cell. In this graphic mode two extra colors are available in addition to the background (G) and foreground (F): the edge color (B) and the auxiliary color (A). The background color (0 to 15), the border color (0 to 7) and the auxiliary color (0 to 15) apply to the entire screen.

To assign colors on the screen in the two modes, you need to use some very direct POKE commands:

POKE 36879.16*G+8+B (to set background colour G and

Figure 6. A screenshot of Minipaint

border colour B)
POKE 36878.16*A (to set auxiliary colour A)
POKE 646.8*M+F (to set foreground colour [F] and enable [M=1] or disable [M=0] multi-colour mode)
Here is the complete VIC colour list:

0 Black	8 Orange
1 White	9 Light Orange
2 Red	10 Light Red
3 Cyan	11 Light Cyan
4 Purple	12 Light Purple
5 Green	13 Light Green
6 Blue	14 Light Blue
7 Yellow	15 Light Yellow

@SAVE [<filename> [,<device>]] - saves the bitmap screen to a device (e.g. disk, 8). The file name is optional when saving to tape. The file is saved with an initial SYS command that invokes a graphics library display routine. The routine then waits for a keystroke and finally restarts the VIC.

@LOAD [<filename> [,<device>]] - loads a bitmap screen from a device (disk or tape). The command without arguments loads the first bitmap screen from the tape. When the loading is finished, the image is automatically shown on the screen and the running program continues with the following instruction. There is no waiting for a button to be pressed.

@(<x>,<y>) - The only function provided returns the color of the pixel at the coordinates (x,y). The x argument varies from 0 to 159 while y goes from 0 to 191 and both can be entered as numbers, variables or numeric expressions. Depending on the graphic mode in use (M=0 or M=1) the function returns values from 0 to 3 (0 for background pixel, 1 for foreground pixel, 2 and 3 for an auxiliary colour pixel).

Figure 7. Bitmap image created with MiniPaint

Example listings

We warmly invite you to download and give a try to MiniGrafik and its many examples which come included in the distribution diskette. We add here below a couple of examples that make use of the library. In the first listing we refer to the articles already appeared on RMW about 2D function plotting, whereas in the second one we present a complete game, a Snake clone, written by Michael Kircher himself. Both listings are self-explanatory and quite easy

Figure 8. A simple clone of Snake

to understand for those who chew at least some BASIC. By examining them you will appreciate how the few commands of MiniGrafik are really well designed and easily integrate with the rest of the BASIC V2 instructions, thus providing a concise and effective tool to write programs that make use of the graphics mode. The results in terms of speed of tracing straight lines and curves, loading and saving screens, control and flexibility in handling the bitmap screen, are perfectly up to expectations. MiniGrafik is now a de facto standard and a stable reference point for all programmers of the beloved and unforgettable Vic-20.

-- Listato: mg-2d functions

```
10 rem vic-20 + minigrafik
20 rem 2d functions plot
30 rem ******************
120 sx=160
130 sy=191
140 hy=sy/2
150 printchr$(147)
160 print"    2d functions plot"
170 print
180 print"1.y=x*x*sin(1/x)"
190 print
200 print"2.y=x*sin(1/x)"
```

```
210 print
220 print"3.y=sqr(x*x+2)"
230 print
240 print"4.y=cos(x*exp(-x/5))"
250 print
260 print"5.y=6+2*x*x-x*x*x*x"
270 print
280 print"type in the number of"
290 print"the equation";
300 input n
310 ifn=1thendeffna(x)=x*x*sin(1/x)
320 ifn=2thendeffna(x)=x*sin(1/x)
330 ifn=3thendeffna(x)=sqr(x*x+2)
340 ifn=4thendeffna(x)=cos(x*exp(-x/5))
350 ifn=5thendeffna(x)=6+2*x*x-x*x*x*x
360 print
370 print"values of x range"
380 print
390 print"lowest value";
400 inputa
410 print
420 print"highest value";
430 inputb
440 print
450 ifa>=bthenprint"error-try again"
460 ifa>=bthengoto360
500 rem ***calculating range of y ***
510 print"calculating range of y"
520 c=(b-a)/100
530 m=1.0e-30
540 forx=atobstepc
550 ifx=0thengoto580
560 y=abs(fna(x))
570 ifm<ythenm=y
580 next x
590 print"ready to plot"
600 fori=1to1000
610 next i
620 printchr$(147)
630 gosub1010:rem prep screen
700 rem *** plotting
710 c=c/10:rem try c/5 or c/20
720 forx=atobstepc
730 ifx=0thengoto790
740 y=fna(x)
750 u=sx*(x-a)/(b-a)
760 v=hy+hy*y/m
770 ifv<0orv>sythengoto790
780 gosub1110:rem plot u,v
790 next x
800 rem *** ending
810 getg$:rem g$=inkey$
820 ifg$=""thengoto810
830 gosub1210:rem restore screen
840 printchr$(147)
850 print" another go? y or n"
860 getg$
870 ifg$<>"y"andg$<>"n"thengoto860
880 ifg$="y"then goto150
890 end:rem stop
1000 rem *** prepare hi-screen
1010 @on
1020 @clr
1030 :
1040 return
1100 rem *** plot function's dots
1110 @1,u,sy-v
1180 return
1200 rem *** restore screen
1210 @return:poke198,0
1220 return
```

-- Listato: mg-snake

```
1 REM *****
2 REM SNAKE
3 REM *****
10 DIMDX(3),DY(3):DX(0)=2:DY(1)=-
3:DX(2)=-
2:DY(3)=3:HX=80:HY=97:TX=HX:TY=HY:I=1
11
POKE36878,160:POKE36879,15:POKE646,13:@
ON:@CLR:@1,0,0TO0,191:@1,158,0TO158,191
12 FORY=0TO2:@1,0,YTO158,Y:@1,0,191-
YTO158,191-Y:NEXT:@2,HX,HY+1TOHX,HY-
1:GOSUB23:SC=0
13 GETA$
14 IFA$="X"THENI=0
15 IFA$=";"THENI=1
16 IFA$="Z"THENI=2
17 IFA$="/"THENI=3
18 @1,HX,HY+1:@I,HX,HY:@1,HX,HY-
1:HX=HX+DX(I):HY=HY+DY(I)
19 J=@(HX,HY-1):@2,HX,HY+1TOHX,HY-
1:IFJ=3THENJ=0:GOSUB23:GOTO21
20 K=@(TX,TY):@0,TX,TY+1TOTX,TY-
1:TX=TX+DX(K):TY=TY+DY(K)
21 IFJ=0THEN13
22 @RETURN:PRINT"SCORE:"SC:END
23
X=2*INT(RND(1)*80):Y=3*INT(RND(1)*64):I
F@(X,Y)<>0THEN23
24 @3,X,YTOX,Y+2:SC=SC+1:RETURN
```

References

[1] Real time video on the Atari 8-bit - https://www.youtube.com/watch?v=PAeYZWz15Ns
[2] ZX-81 1K Chess - https://www.youtube.com/watch?v=m0VAwqg9N0k
[3] HFLI picture on a C64 - https://www.youtube.com/watch?v=SgM6KVdae3Y
[4] Doom running on Vic-20 - https://github.com/Kweepa/vicdoom
[5] Bad Tracker - https://www.youtube.com/watch?v=SDvk3aL78fI
[MG] MiniGrafik Batch Suite - http://sleepingelephant.com/ipw-web/bulletin/bb/viewtopic.php?t=5179
MiniGrafik download - https://dateipfa.de/.Public/denial/minigrafik/minigrafik.zip
MiniPaint - http://sleepingelephant.com/ipw-web/bulletin/bb/viewtopic.php?t=5627
MiniPaint manual - https://dateipfa.de/.Public/denial/minigrafik/manual.zip

Alien Attack!

a new Locomotive Basic game

by Francesco Fiorentini

At the beginning of August, on the **Retroprogramming group Italy - RP Italia**, a challenge was called inviting attendees to reproduce a clone of the game Air Attack. The challenge proposed by the group, with the aim of bringing more and more people closer to retro-programming, was as follows:

A) The challenge is to play one or more clones of the game "Air Attack", using the following languages: Basic, or C, or ASM or even a combination of the above, such as Basic+ASM or C+ASM.

B) Such games may be programmed for any 8/16-bit computer (including 8088/8086 platforms).

C) Your work must in any case respect the following categories:

1) Category "Full BASIC"

2) Category "Full C"

3) "Full ASM" category

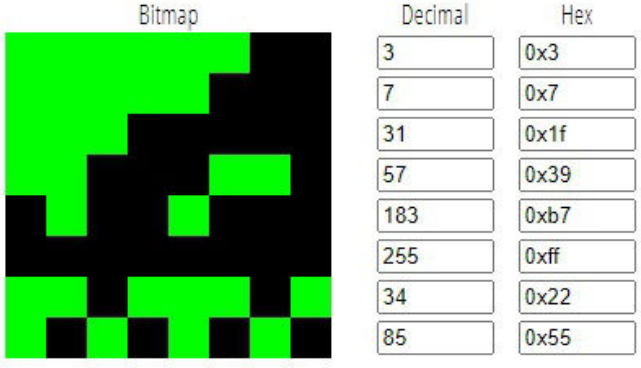
4) Category "Mixed C" (C + ASM)

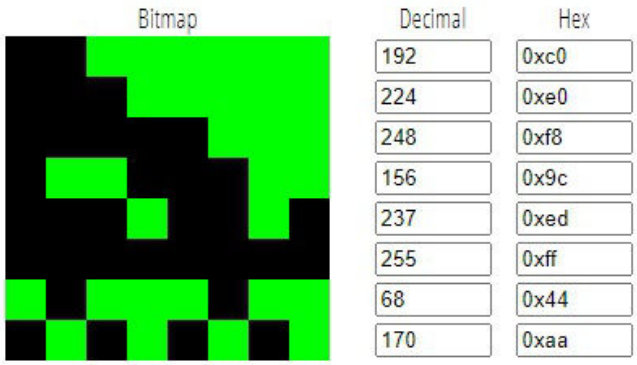
5) Category "Mixed BASIC" (Basic + ASM)

Initially the deadline for the delivery of the work had been set for the end of August and, overwhelmed by work and family commitments, I had initially exceeded it.
Towards the end of August, however, I notice that the deadline was postponed to September 24 and, driven by curiosity, I decide to write two lines to animate an airplane...

Obviously my poor propensity for graphics is not at all helpful in drawing an airplane, so I decide to focus on something more linear. Idea! I can replace the airplane with an alien spaceship; much easier to make.
However, the problem remains that in order to make a ship that is minimally credible, a single character is not enough, it takes at least two.

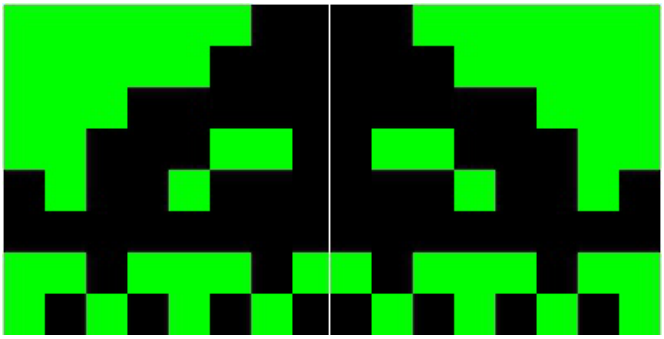
Armed with all my artistic flair, I create the two works as





you can see above.

Seeing them individually, magnified in the 8x8 editor, does not make a nice impression, but once they are side by side I have to admit that the effect is credible enough.
I had my spaceship.



Now all that remained was to animate it and see if the Locomotive Basic can handle the movement of two characters side by side quickly enough... (spoiler: it definitely can!).

Fortunately, the animation of the protagonist of the Air Attack game is quite simple. It is a matter of moving the object horizontally, in only one direction and, once you reach the extreme limit of the screen, starting from the opposite end only a little lower.

You can just draw the two characters side by side and move them one at a time (to the right or left), taking care to clear the first box opposite the direction of travel (easier to do than to say...).
Repeating this process for the entire length of the screen will have the effect of a unique object moving from one side of the video to the other.
I strongly suggest to those who have never tried to create a game, to try to animate an object on the screen to understand how simple this technique is. Once you've

done this experiment, you'll want to add a few more objects and in no time you'll have accomplished something that looks like a game. Try to believe!

Well, now that I've animated the main character, using the same principle, but vertically, I can animate the bomb. By the way, the animation of the bomb naturally leads to the destruction of the buildings, since the design of the bomb will overwrite ecah floor of the building any time it passes over. Easier than that!

The background drawing is probably the thing that took the most of my time. I wanted to achieve something that was random, but at the same time with a certain logic. Initially I tried to limit the number of buildings and their height by drawing them randomly along the entire length of the screen... The effect was horrible!
More than a city, they looked like a series of random stairs, without any logic.

I need a better idea... Think Francesco, think!

Idea! What if I draw the buildings starting from the center and expanding to the right and left as I add one?
I can always limit their number and height and increase these variables as you continue in the game, but at least the visual impact should be better.
I tried and... EUREKA! I found my background drawing routine.
The effect is not that bad, especially considering that everything is generated randomly and dynamically.
Also take in mind that the floors of the buildings and their tops are also randomly generated by choosing from a

group of 6 and 4 different tiles respectively and you can understand how it is actually difficult to play a game that is the same as a previous one.

I have achieved my goal, I have created a random background that, however, has a semblance of credibility resembling a city.

What remained to be done? Oh, yes! The collision control of the spacecraft with the buildings and the passage to the next level once all the buildings have been destroyed. Well, you won't believe it, but out of the two, the second one was the most complex challenge.

Collision control in Locomotive Basic is pretty trivial. This language is provided with the COPYCHR$() function which copies the character of the current position into a variable. It is therefore intuitive how it is enough to position yourself with A LOCATE in the position that our spaceship will assume and read its contents. If the character contained is a plan of the skyscraper or its summit, it means that the spaceship collided with a palace!

As for intercepting the destruction of all the buildings and the consequent passage to the next level, I used a trick. I could have memorized the location of each building in an array and cleaned the location whenever the bomb was on its vertical, but then I would have to scroll through the array to see if all the buildings had been destroyed...

I then chose to create a text string containing 40 times the character 0. Every time I draw a building I put a 1 in the corresponding position, while setting the value back to 0 in the position where the bomb is dropped. Every time the spaceship has reached the end of the screen I check the string variable and if it is formed by 40 times 0, it means that the level is cleaned and I can move on to the next one.

Now the frame of the game is practically complete and I can start working on its enrichment.

I drawn an image as an introduction, but I soon noticed that it lacks of grit. My game will be called Alien Attack!

and the mission of the aliens is to invade Earth... So I need something stronger. The current image, which sees an alien face to face with a human being, is exactly what I was looking for.

In the last issue I described how to import images into Amstrad and how to show them with Basic. I used that same technique to display the introductory screen and the Game Over image.

Every game must entice the player by rewarding him with a score and with the possibility to improve always a lttle bit by challenging himself.
Then I decided to add points:
- each passage of the ship without collisions is rewarded with 50 points
- completing the level rewards with 500 points, plus 100 points for each remaining row separating from the ground and an high score!

To make it even more interesting, and give the player the chance to always improve, an extra life can be won on every 15000 points (each multiples of 15000).

The gaming technique is obvious, but I can suggest targeting the tallest buildings at the first place and moving on to the lower ones later...

Ah, I forgot to mention that I used again the Future Set font I described in number 3.
Now I can say the game is complete!

The code, sufficiently commented, can be found here below, while the virtual image of the floppy disk (with the images) to be uploaded on your favorite Amstrad CPC emulator (I use WinAPE), can be downloaded from:
http://www.retromagazine.net/download/AlienAttack.zip

Give it a try and let me know what you think.

Have fun!!!

```
10 REM **********************************
11 REM AlienAttack! by Francesco Fiorentini
12 REM SETTEMBRE 2020
13 REM **********************************
17 HISCORE=5000
18 GOSUB 10000: REM INTRO
19 GOSUB 9020: REM LETTERS AND NUMBERS RE-DEFINITION
20 CLS

29 REM GAME CHARS RE-DEFINITION
30 GOSUB 6000
39 REM NP=NUMBER OF BUILDINGS - TH=TOP HIGH
40 NP=6:TH=6:LV=1:LF=3:PT=0:EXTRA=15000

69 REM BACKGROUND (THE CITY)
70 GOSUB 7000

99 REM INITIAL PARAMETERS SETTING
100 BF=0:I=0

998 REM ALIEN MOVEMENT
999 REM R=ROW
1000 FOR R = 1 TO 23
1100 FOR I = 1 TO 42
1110 LOCATE I,R:CK$=COPYCHR$(#0):CKV=ASC(CK$)
1120 REM CHECK COLLISION
1130 REM - VOLD CHECK - IF CK$=PT$ OR CK$=PF$ THEN GOTO 3000
1140 IF CKV<>32 AND CKV<>254 AND CKV<>255 THEN GOTO 3000
1200 PRINT AF$
1220 IF I>1 THEN LOCATE I-1,R: PRINT AB$
1230 IF I>2 THEN LOCATE I-2,R: PRINT " "
1239 REM BOMB CHECK
1240 GOSUB 5000
```

```
1280 NEXT I
1290 PT=PT+50: IF PT>=EXTRA THEN GOTO 1291 ELSE GOTO 1298
1291 EXTRA=EXTRA+15000:LF=LF+1
1292 PEN 2:LOCATE 1,24: PRINT"LEVEL:"; LV ;"- LIVES:"; LF ; "- POINTS:"; PT:PEN 1
1293 GOTO 1299
1298 LOCATE 30,24:PEN 2: PRINT PT: PEN 1
1299 IF SF$="00000000000000000000000000000000000000000000" THEN PT=PT+(23-R)*100:GOTO 1310
1300 NEXT R
1301 REM LEVEL COMPLETED!!! MOVE TO NEXT ONE
1310 GOSUB 6500
1320 GOTO 70

3000 REM CRASH
3005 CRA$="***":PCRA$="   ":X=I-2:IF X<=0 THEN X=1:CRA$="**":PCRA$="  "
3010 LOCATE X,R: PRINT CRA$
3020 LF=LF-1
3030 IF LF=0 THEN GOTO 3500
3040 FOR T=1 TO 1000: NEXT T
3050 LOCATE X,R: PRINT PCRA$
3060 PEN 2:LOCATE 1,24: PRINT"LEVEL:"; LV ;"- LIVES:"; LF ; "- POINTS:"; PT
3061 PEN 1: LOCATE 10,25:PRINT "- HI-SCORE:"; HISCORE; "-"
3070 GOTO 99

3499 REM GAME OVER
3500 MODE 1
3501 LOAD "GOVER.SCR",&C000
3510 LOCATE 11,6: PRINT"----  GAME OVER  ----"
3511 LOCATE 15,8: PRINT"SCORE: "; PT
3525 FOR T=1 TO 1000: NEXT T
3530 LOCATE 11,10: PRINT"-- PLAY AGAIN? Y/N --": PEN 1
3531 IF PT>HISCORE THEN HISCORE=PT
3550 RE$ = INKEY$
3560 IF RE$="Y" OR RE$="y" THEN GOTO 20
3570 IF RE$="N" OR RE$="n" THEN GOTO 4999
3580 GOTO 3550
4998 REM END OF THE GAME
4999 END

5000 REM BOMB MANAGEMENT - BR=BOMBROW - BL=BOMBLINE BF=BOMBFLAG(0/1=N/Y)
5010 IF INKEY(47)=0 AND BF=0 THEN GOTO 5020 ELSE GOTO 5025
5019 REM NUOVA BOMBA
5020 BR = R+1: BL=I: BF=1: GOTO 5190
5024 REM CHECK IF A BOMB ALREADY EXISTS
5025 IF BF=0 GOTO 5200 ELSE BR = BR+1
5030 IF BR = 24 THEN BR=0: BF=0: LOCATE BL,23: PRINT " ": MID$(SF$,BL,1)="0": GOTO 5200
5040 LOCATE BL,BR-1: PRINT " "
5190 LOCATE BL,BR: PRINT BO$
5200 RETURN

6000 REM ALIEN CHAR(254 e 255)
6010 SYMBOL 254, 3, 7, 31, 57, 183, 255, 34, 85
6020 REM ALIEN CHAR(255)
6030 SYMBOL 255,192, 224, 248, 156, 237, 255, 68, 170
6040 REM DRAW THE BOMB CHAR(253)
6050 SYMBOL 253,0, 36, 24, 60, 60, 60, 24
6060 REM BUILDING FLOORS CHAR (230,231,232,233,234,235)
6061 SYMBOL 230,255, 219, 255, 219, 255, 219, 255, 219
6062 SYMBOL 231,255, 251, 255, 223, 251, 255, 223, 255
6063 SYMBOL 232,255, 219, 255, 223, 255, 223, 255, 255
6064 SYMBOL 233,255, 255, 219, 255, 219, 255, 255, 255
6065 SYMBOL 234,255, 189, 255, 247, 255, 251, 191, 255
6066 SYMBOL 235,255, 223, 255, 247, 255, 255, 223, 255
6080 REM BUILDING TOP1 CHAR (240) -
6081 SYMBOL 240, 0, 0, 0, 0, 24, 60, 90, 189
6082 REM BUILDING TOP2 CHAR (241)
6083 SYMBOL 241, 129, 129, 129, 129, 195, 231, 255, 255
6084 REM BUILDING TOP3 (242)
6085 SYMBOL 242, 24, 126, 126, 24, 24, 60, 126, 255
6086 REM BUILDING TOP4 (243)
6087 SYMBOL 243, 24, 24, 24, 24, 24, 60, 126, 255
6189 REM AF$=ALIENFRONT - AB$=ALIENBACK - BO$=BOMB - PF$=PALACEFLOOR - PT$=PALACETOP
6190 AF$=CHR$(255):AB$=CHR$(254):BO$=CHR$(253):PF$=CHR$(230)
6200 RETURN

6499 REM DINAMIC LEVEL MANAGEMENT
```

```
6500 NP=NP+1:LV=LV+1:PT=PT+500:TH=TH+1
6501 IF PT>=EXTRA THEN EXTRA=EXTRA+15000:LF=LF+1
6504 IF TH>14 THEN TH=15:IF NP>19 THEN NP=19
6505 PEN 2:LOCATE 11,4: PRINT"--------------------"
6510 LOCATE 11,5: PRINT"-- GO TO LEVEL";LV;"----"
6520 LOCATE 11,6: PRINT"--------------------"
6525 FOR T=1 TO 1000: NEXT T
6530 LOCATE 11,7: PRINT"-- R U READY? Y/N ---"
6540 LOCATE 11,8: PRINT"--------------------":PEN 1
6550 RE$ = INKEY$
6560 IF RE$="Y" OR RE$="y" THEN GOTO 6570
6565 GOTO 6550
6570 REM FOR T=4 TO 8
6580 REM LOCATE 11,T: PRINT"                    "
6590 REM NEXT T
6591 CLS
6600 RETURN

7000 REM DRAW THE CITY
7001 REM NP=NUMBER OF BUILDINGS - P=PEN'S COLOUR
7002 SF$="00000000000000000000000000000000000000000000"
7008 PEN 2:LOCATE 1,24: PRINT"LEVEL:"; LV ;"- LIVES:"; LF ; "- POINTS:"; PT
7009 PEN 1: LOCATE 10,25:PRINT "- HI-SCORE:"; HISCORE; "-"
7010 X=20-(NP/2): PEN 2
7020 FOR N=1 TO NP
7021   REM LOCATE 2,2: PRINT X
7022   REM RANDOM TOP OF THE BUILDING X=INT(RND(1)*(high-low))+low
7025   A=INT((RND(1)*4)+1)
7026   PT$=CHR$(239 + A)
7030   HI=INT(RND(1)*(TH-2))+2
7031   OLDX=X
7041   MID$(SF$,X,1)="1"
7042   X=OLDX
7050   FOR I=HI-1 TO 0 STEP -1
7051     REM DRAW THE BUILDING FLOOR RANDOMLY X=INT(RND(1)*(high-low))+low
7052     A=INT((RND(1)*6)+1)
7053     PF$=CHR$(229 + A)
7063     rem P=INT((RND(1)*3)+1):PEN P
7065     IF I=HI-1 THEN GOTO 7066 ELSE GOTO 7070
7066     LOCATE X,23-I
7067     IF COPYCHR$(#0)<>PF$ THEN PRINT PT$: GOTO 7090
7070     LOCATE X,23-I: PRINT PF$
7090   NEXT I
7091 X=X+1
7100 NEXT N
7105 REM PEN 2: LOCATE 1,24: PRINT"LEVEL:"; LV ;"- LIVES:"; LF ; "- POINTS:"; PT: PEN 1
7110 PEN 1
7300 RETURN

9010 REM ******************************************
9011 REM * Future Set on Amstrad CPC
9012 REM * original code by Pete White
9013 REM * Popular Computing Weekly 713 August 1983
9014 REM *
9015 REM * Typed and corrected by
9016 REM * Francesco Fiorentini on June 2020
9017 REM * RetroMagazine World July 2020
9018 REM ******************************************
9020 SYMBOL AFTER 32
9030 REM Upper case chars
9040 SYMBOL 65,126,66,66,126,98,98,98,0
9050 SYMBOL 66,126,66,66,126,98,98,126,0
9060 SYMBOL 67,126,64,64,96,96,96,126,0
9070 SYMBOL 68,254,66,66,98,98,98,254,0
9080 SYMBOL 69,126,64,64, 120,96,96,126,0
9090 SYMBOL 70,126,64,64,120,96,96,96,0
9100 SYMBOL 71,126,64,64,102,98,98,126,0
9110 SYMBOL 72,66,66,66,126,98,98,98,0
9120 SYMBOL 73,60,16,16,24,24,24,60,0
9130 SYMBOL 74,126,8,8,24,24,24,120,0
9140 SYMBOL 75,68,68,68, 120,100,100,100,0
9150 SYMBOL 76,64,64,64,96,96,96, 126,0
9160 SYMBOL 77,126,74,74,98,98,98,98,0
9170 SYMBOL 78,98,82,74,102,98,98,98,0
9180 SYMBOL 79,126,66,66,98,98,98,126,0
```

```
9190 SYMBOL 80,126,66,66,126,96,96,96,0
9200 SYMBOL 81,126,66,66,98,98,106,126,4
9210 SYMBOL 82,126,66,66,126,106,100,98,0
9220 SYMBOL 83,126,64,64,126,6,6,126,0
9230 SYMBOL 84,126,16,16,24,24,24,24,0
9240 SYMBOL 85,66,66,66,98,98,98,126,0
9250 SYMBOL 86,66,66,66,66,66,36,24,0
9260 SYMBOL 87,66,66,66,98,106,106,126,0
9270 SYMBOL 88,102,102,36,24,36,102,102,0
9280 SYMBOL 89,66,66,126,16,24,24,24,0
9290 SYMBOL 90,126,4,8,16,32,64,126,0
9295 REM Lower case chars
9300 SYMBOL 97,0,0,126,6,126,70,126,0
9310 SYMBOL 98,96,96,96,126,98,98,126,0
9320 SYMBOL 99,0,0,126,96,96,96,126,0
9330 SYMBOL 100,6,6,6,126,70,70,126,0
9340 SYMBOL 101,0,0,126,98,126,96,126,0
9350 SYMBOL 102,60,48,48,120,48,48,48,0
9360 SYMBOL 103,0,0,126,70,70,126,6,126
9370 SYMBOL 104,96,96,96,126,98,98,98,0
9380 SYMBOL 105,24,0,24,24,24,24,24,0
9390 SYMBOL 106,6,0,6,6,6,6,6,126
9400 SYMBOL 107,96,96,102,108,120,108, 102,0
9410 SYMBOL 108,24,24,24,24,24,24,24,0
9420 SYMBOL 109,0,0,126,90,90,66,66,0
9430 SYMBOL 110,0,0,108,114,98,98,98,0
9440 SYMBOL 111,0,0,126,102,102,102,126,0
9450 SYMBOL 112,0,0,126,98,98,126,96,96
9460 SYMBOL 113,8,0,126,70,70,126,6,6
9470 SYMBOL 114,0,0,108,114,96,96,96,0
9480 SYMBOL 115,0,0,126,96,126,6,126,0
9490 SYMBOL 116,24,62,24,24,24,24,30,0
9500 SYMBOL 117,0,0,102,102,102,102,126,0
9510 SYMBOL 118,0,0,102,102,102,60,24,0
9520 SYMBOL 119,0,0,66,66,90,90,126,0
9530 SYMBOL 120,0,0,198,104,16,104,198,0
9540 SYMBOL 121,0,0,102,102,102,126,6,126
9550 SYMBOL 122,0,0,126,12,24,48,126,0
9555 REM Numbers
9560 SYMBOL 48,126,102,110,118,102,102,126,0
9570 SYMBOL 49,24,56,24,24,24,24,126,0
9580 SYMBOL 50,126,2,2,126,96,96,126,0
9590 SYMBOL 51,126,2,2,30,6,6,126,0
9600 SYMBOL 52,96,96,96,96,104,126,8,8
9610 SYMBOL 53,126,64,126,6,6,6,126,0
9620 SYMBOL 54,126,64,64,126,98,98,126,0
9630 SYMBOL 55,126,2,4,62,16,32,64,0
9640 SYMBOL 56,126,66,66,126,66,66,126,0
9650 SYMBOL 57,126,66,66,126,6,6,6,0
9680 SYMBOL 95,0,255,0,0,0,0,0,0
9690 RETURN

10000 REM INTRO
10005 MODE 1
10010 LOAD "ALIEN2.SCR",&C000
10015 PEN 2
10020 FOR I=1 TO 1000:NEXT I
10030 LOCATE 15,22:PRINT"ALIEN ATTACK!"
10040 LOCATE 8,24:PRINT "2020 - Francesco Fiorentini"
10045 PEN 1
10050 FOR I=1 TO 3000:NEXT I:CLS
10060 LOCATE 5, 5: PRINT "Our planet is dying."
10061 LOCATE 5, 6: PRINT "Our species is in danger."
10062 LOCATE 5, 7: PRINT "Our future is in danger."
10063 LOCATE 5, 8: PRINT "Our only purpose is survival."
10064 LOCATE 5, 9: PRINT "Whatever the cost..."
10065 LOCATE 5, 11: PRINT "We do not want to live together."
10066 LOCATE 5, 12: PRINT "We do not want to live together."
10067 LOCATE 5, 13: PRINT "We want the Earth!"
10068 LOCATE 5, 14: PRINT "Whatever the cost..."
10069 LOCATE 5, 16: PRINT "Whatever the cost!"
10070 LOCATE 5, 18: PRINT "This means WAR!"
10075 FOR I=1 TO 3000:NEXT I:
10080 RETURN
```

An introduction to ARexx – part 2

by Gianluca Girelli

This article first appeared on Bitplane pages in September 2012.

After the appetizer of the previous tutorial, published on issue 20 of RM and which led us to explore the basics of language, we'll now begin to analyze in greater depth the power of ARexx scripting. As thoroughly illustrated, what determined the success of REXX (and therefore ARexx) was the fact that it could truly be a glue among the most diverse applications and, since everything is based on effective inter-process communications, it is necessary to have constructs able to simplify and format the input/output flows to arrive at a fast and efficient validation of the text. That's why managing strings, i.e. alphanumeric sequences of characters read in input or produced in output, is one of ARexx's strengths.
For the purposes of our discussion, we will now introduce some language instructions related to string management that will allow us to create a (absolutely minimal) "syntactic parser" (see box).

Syntactic parser

In linguistics and computer science the term "parsing" or, more formally, "syntactical analysis", is the process of analyzing a sequence of symbols (tokens) or words to determine the congruence of the grammatical structure in relation to a given "formal grammar". More in detail, a parser is one of the components of an "interpreter" (or "compiler") and is used to check the syntactical correctness of the language and to build its data structures.

ARexx and strings

The absolute importance given to strings by the creators and first users of language is clear from the quantity and quality of the instructions that can be used to manage them. In ARexx we can have, for example, constructs capable of:
- edit: "compress (str, [list])" compresses the string by removing the characters in the "list" from "str". Other instructions deal with deleting, inserting or overlapping strings or parts of them;
- compare: "compare(str1, str2)" determines the position from which the two strings differ. Other instructions in this "class" control whether a string is an abbreviation of another or whether a string is contained in another;
- format: extract one string from another. This category includes the "left" and "right" instructions given in the following examples;
- work directly on the words: "words(str)" determines the number of words contained in "str", "wordindex(str, n)" determines the position from which the n-th word starts in "str" while "word(str,n)" extracts the word "n" from the string "str".

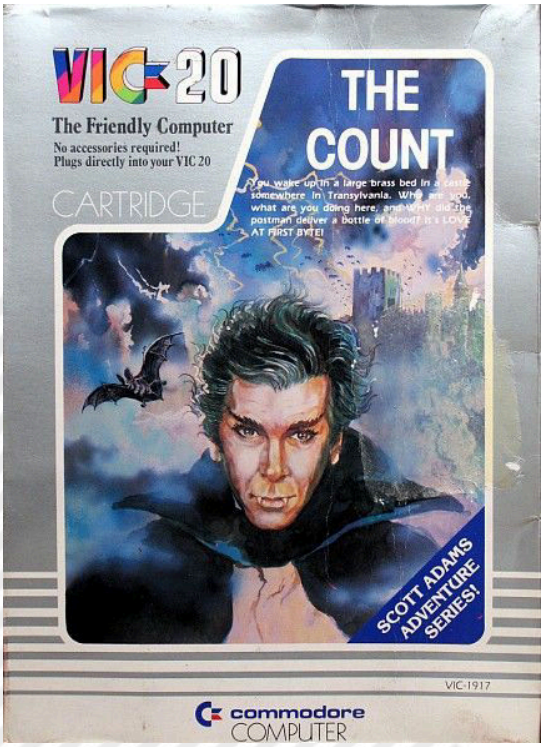


Figure 1: The Count for VIC20

Then there is a particular command,"parse", which has so many arguments that it can perform many of the functions just mentioned by using user-definable "templates". This greatly facilitates the usual mechanism that involves reading the alphanumeric sequence of characters and its subsequent elaboration and decomposition into the "significant" parts of the sentence. In total, the manual "Using ARexx on the Amiga" mentioned in the previous article, reports over thirty functions available only to effectively manage strings.

Let's start now with some examples (instructions in the box) to better clarify what has just been expressed. In these examples we assume for simplicity that we have to process a string that we already know to be composed of two words (we will then see why).

```
/* example 1 */
```

Using the examples

As reported in the article on number 3, to use the examples you have to save the script in text mode in the format "name_script.rexx". To launch the script just type from shell ">rx name_script.rexx" or, more simply, ">rx name_script".

```
pull phrase
verb=left (phrase, index(phrase, ' ')-1)
name=right(phrase,length (phrase)-
index(phrase,' '))
```

In the given example, after acquiring our string with the "pull" statement and storing it in the "phrase" variable, we divide it by assigning the first part of the string ("left") to the "verb" variable and the second ("right") to the "name" variable. The division is made assuming that the two words are separated by a space, whose position in the string is determined by the "index" statement. If the string is therefore "take rope", our two variables will assume the values of:

```
verb="TAKE" and name="ROPE".
```

Remember, in fact, that the "pull" statement is short for "parse upper pull" and automatically converts the string to uppercase.
The same identical result can be obtained much more easily using a "template":

```
/* example 2 */
parse upper pull verb '  ' name
```

where the template consists of the sequence: "variable1 ' ' variable2". Note that the two variables (in the code called directly "verb" and "name") are separated by a space enclosed in single quotation marks that, at run time, will remove the space that separates the two words within the string.

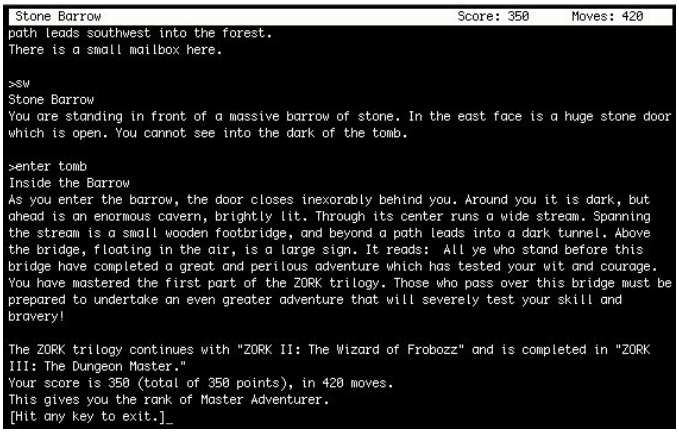


Figure 2: DOS version of ZORK

A more sophisticated way to extract too many spaces from a string without using the sequence of single quotes just mentioned is to combine the keyword "parse" with the argument "var". This argument specifies that the word after "var" is a variable and that all other words are instead the "template" to be used to divide the string. Because every word extracted, except the last remaining word, removes the white spaces before and after the word itself, we build our model by making the program believe that the string contains one more word than it actually is.

```
/* example 3 */
pull phrase
parse upper var phrase verb phrase rest
name = phrase
```

What happens in this case is that our sentence is read in input and inserted in the variable "phrase" ("pull phrase"). It is then converted to uppercase ("parse upper") and the first word is extracted and assigned to the variable "verb". Since "phrase" is also the second parameter of the pattern, it would now contain all the rest of the string, including the space between the two words.
By inserting a third variable (to be lost), the second word also removes unwanted whitespace. In the end, for the sake of clarity, the phrase content is assigned to the variable "name", although this would not be necessary.

Let's now look at other examples using the templates:

```
/* example 4 */
say "Enter name and age, for example: Gianluca
Girelli,50"
parse pull name "," age
say " You say you are" name"," age "years old."
```

As you can see when launching the program, when ARexx is waiting for input in the console, a simple blank line is displayed.
Often, however, it is more "user-friendly" to display a "prompt" (for example ">") to tell the user that they must type something. This can be done with the OPTIONS PROMPT statement as follows:

```
/* example 5 */
options prompt ">"
say "Enter name and age, for example: Gianluca
Girelli,50"
```

```
parse pull name "," age
say " You say you are" name"," age "years old."
```

A simple parser to play with

At this point, unless you are fond of programming languages for your own sake, you will be starting to get bored and, since our motto continues to be "Remember when computing was fun?", why not put some of these notions into practice by having fun?
Personally, I've always been a big fan of textual adventures ever since I played "The Count" with a friend's VIC20. As you may recall, the beauty of those games was that they told of worlds that took shape directly in our heads and their limit was just our imagination. "The Count" recounted the exploration of a ghost castle in search of the "Count" (Dracula, of course) and the "navigation" within the game was carried out by entering simple structured sentences as in the first example of this article. Then came "Zork" of the late Infocom and everything else went into the background [see photo], since Infocom's syntax analyzers could parse two lines of text....

Without wanting to mount our heads, however, thanks to ARexx and its powerful string management instructions, we can write our little parser and lay the foundations for what will eventually be our game engine for textual adventures!

Suppose we need to issue commands to our text avatar and have decided that these commands can only be in the form of "verb" or "verb+name". Our parser will therefore have to respect these rules and will basically take care of dividing the string into its components, automatically excluding empty strings (null command!) and those containing more than two words as they are not consistent with the rules of the given "grammar".

So let's look at the following subprogramme:

```
/*-------------------------*/
/* 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
otherwise say"I don't understand. try again"
  end
if words =2 then do
verb=left (phrase, index(phrase, ' ')-1)
name=right(phrase,length (phrase)-
index(phrase,' '))
end
return
```

The subroutine is identified by an initial "label" ("Parser:") that serves both as the name of the subprogram and as the logical address of reference for the call by the main program and ends with the "return" statement.
The logic of this routine reflects what has been said previously, therefore:
- if the string contains at least one word ("if words (phrase)=1") the program checks if it is in the list of known individual commands. In this case the command is assigned to the variable "verb", otherwise an error message is returned;
- if the string is two words, proceed to its division as already illustrated in the opening of the article.

We must basically note two things: first, because in ARexx the variables are global, that is, accessible by anyone at any time, if the string is composed of a single word, the subroutine also takes care of resetting the variable "name" so as not to trigger behaviors not foreseen by our logic; moreover, the parser could also directly invoke the actions to be performed once the string is deciphered, but we chose to keep the two things conceptually separate.

At this point our syntax analyzer is ready to be used within the main code which could look like the following and which, although not directly related to the processing of strings, has been inserted for the sake of clarity.

```
/* main code */
pull phrase
call Parser (phrase)
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()
otherwise say 'I don't know what it means '
|| verb
end
```

As can be seen from the code, once the string has been read in input and analyzed thanks to our parser, depending on the value of the variable "verb", the subroutine will be invoked, possibly passing as a parameter the value of the variable "name".
Note that in the case of "GO" the current position (contained in the variable "Pos") to be updated will also be passed to the navigation routine in addition to the "where" (eg: GO NORTH).

Conclusions

The problem of formatting the input/output is as old as the computer itself being intimately connatured with the reason why computers were created, that is to help solve problems quickly and effectively.
As we have seen, with ARexx this problem virtually does not exist given the number, power and flexibility of the instructions that deal with strings in this language.

I hope this article has shown how much fun working with strings can be. With the simple use of a few instructions we have in fact created our first parser, which could be refined and enhanced infinitely.
If this reading gradually made you want to exhume some old textual adventure, the links at the end of the pages will be very useful. However, don't miss the next issues of RetroMagazine as the "Programming with ARexx" section is about to take on the appearance of "Game Coding with ARexx". Deep down... do you remember when computing was fun?
Have fun!

BIBLIOGRAPHY

- For the textual adventures of Scott Adams visit:
http://www.msadams.com/index.htm
- For the textual adventures of Infocom visit:
http://www.infocom-if.org/games/games.html
- For "ZORK" (MS-DOS) visit the Infocom website:
http://www.infocom-if.org/downloads/downloads.html
- For "ZORK" on Amiga visit:
http://www.lemonamiga.com/
- For Amiga emulators visit:
http://www.amigaforever.com/

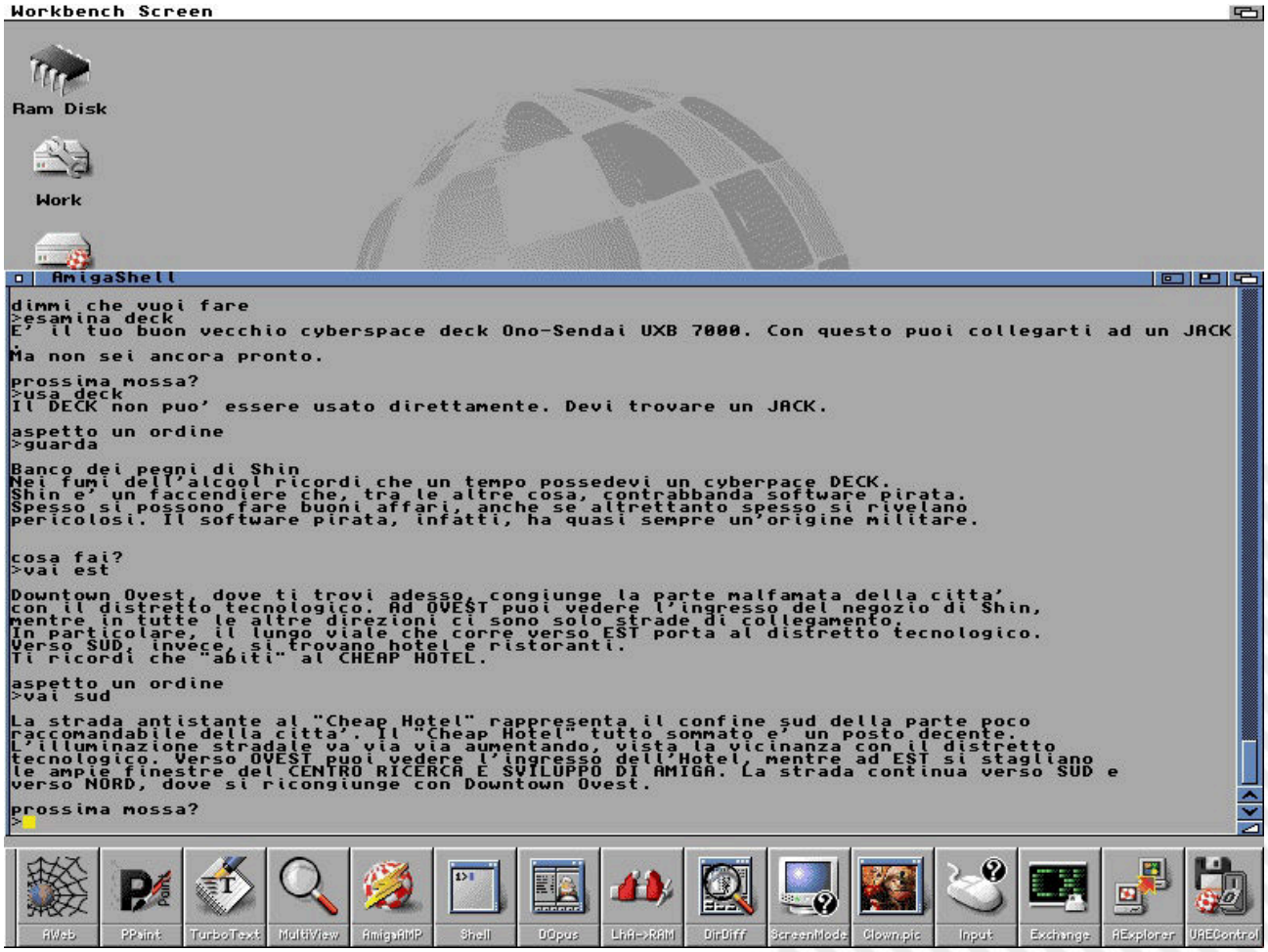


Figure 3: Cyber.rexx on AF12

Cross-programming in C on the Olivetti M20

by Davide Bucci

1 - Introduction

Last month, we saw an introduction to the Olivetti M20, a rather peculiar machine from 1982 [1]. We spent some time discussing the characteristics and the quirks of the Professional Computer Operating System or PCOS. This article describes how to use a comparatively modern C compiler (specifically, a version of GCC) to develop software for it.

One of the advantages of exploiting modern computers and tools to program vintage computers is that we now have beautiful text editors, excellent compilers and efficient languages. A purist may argue that the true "1980's experience" may be lost, but this is compensated by countless advantages, especially for relatively large projects. In the last few months, I have used this approach for developing text-adventures. I have been able to exploit the portability of the C language, targeting systems as different as the Sinclair ZX Spectrum and the Olivetti M20 with almost the same source code.

I will describe in this article how to cross-compile for the M20 in C on a Unix-like operating system such as Linux or macOS. I do not have any Windows machine around, but I think that for that operating system, programs such MinGW or Cygwin may be useful. A convenient strategy is to have on the developing machine both a compiler and an emulator running for each platform. The required tools (z8k-pcos GCC, m20disk, MAME) must be downloaded and in some cases built from sources, so I hope you will not be put off by things such as GNU make.

This article is organized as follows. I will start by briefly describing the compiler and the MAME emulator for the M20. Then, I will show how to use them to compile and run some introductory examples. I will finally discuss how to transfer the executables on the real hardware and run them there. I will finally discuss a nontrivial example (a small graphic demo) before drawing some conclusions.

2 - The C cross-compiler and the emulator

Many personal computers of the 1980's could be programmed in one of the many BASIC dialects available. The Olivetti M20 was no exception and came with a reasonably complete Microsoft interpreter, called BASIC-8000. Even if BASIC was a simple language and was easy to learn, it was painfully slow in some situations. Moreover, it was not very convenient for low-level operations, not efficient for large projects and severely limited in many areas. I started programming with BASIC on my VIC-20 when I was a child and I used it for many years on the PC too, but I am not very fond of it. An assembler suite for the Z8001 was available for the M20, but handling large projects in assembly is often tedious, cumbersome and the code is not portable, even if one can possibly write extremely compact and efficient programs.

From the modern perspective, the C programming language offers a good trade-off between execution speed, ease of coding and overall efficiency on limited machines, being a remarkably efficient compiled language. I will not describe here the strengths and pitfalls of the C language (many resources and tutorials are available on the Internet for that), but modern compilers targeting 8- and 16-bit processors exist. These are for example the cc65 for the 6502, the z88dk for the Z80, etc... For the Olivetti M20, much work has been done in this direction by Christian Groessler over several years. He created a version of GCC 2.9 dedicated to the Zilog Z8001 processor and PCOS, from a compiler originally put together in 1998 by the eCos group (then part of RedHat). His work included GNU binutils as well as newlib.

GCC 2.9 does not support all the bells and whistles of recent standards for C and C++, but it is still a very decent compiler, much more powerful than the original Microsoft BASIC available on the machine. Christian distributes the compiler along with its sources for many Unix systems on his FTP site [2], and wrote an introductory article, which is available at [3].

One possible strategy to install the compiler is to use one of the available binary distributions (Chris kindly prepared packages for many Un*x flavours), or directly compile it from sources. Once everything is done, you should install the executables in /usr/local/bin or make sure that they can be reached via current shell path. If the install has succeeded, typing the following command should yield the compiler version, as follows:

```
$ z8k-pcos-gcc --version
2.9-ecosSWtools-990319-m20z8k-3
```

The compiler suite is composed of a collection of tools that appear familiar if you are used to GCC. There are versions that are dedicated to COFF executables, but we will not use them on the M20. The tools dedicated to PCOS start with the z8k-pcos prefix.

Probably, the most convenient way to cross-develop for

a vintage computer is to have an efficient compiler paired with an emulator, both available on the modern machine. The second tool we are going to use is therefore MAME, as from version v0.212, it started to partially support the M20. The implementation is still slightly buggy, but remains quite useful to rapidly test simple programs.
Benjamin Eberhardt has written a very interesting article about how to use MAME to emulate an M20 [4].

MAME can be downloaded at [5] and among other persons, many of the efforts done to emulate the M20 have been done (once again) by Christian Groessler. After the download, you will need a copy of the boot ROM code that can be found at [6], as well as an image of a boot disk containing PCOS, such as the one that is present in the archives associated to this article [7]. Once MAME is installed on your system and you have put the M20 ROM in the current directory, it can be launched with a command that has the following structure:

```
$ mame m20 -bios 0 -rompath . -flop1 <image1>
-flop2 <image2> -window
```

Now that the main tools we need are ready, in the next paragraph we are going to discuss, compile and run some simple C programs on the emulator.

3 - Three 'Hello World' programs

Of course, the first program that one may use to test the compiler toolchain is the very well-known Hello World program:

```
#include<stdio.h>

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    return 0;
}
```

Figure 1: Hello world output in MAME

If we call this file hello.c, the command to compile it is:

```
$ z8k-pcos-gcc hello.c -o hello.cmd
```

A rather unpleasant surprise is that the executable is 16211 byte long. If it is tiny for today's standards, it is relatively large for a 1982 computer and this size is not acceptable for such a simple program. We must mitigate this problem.

The culprit is the standard library and in particular the implementation of the printf function. This function offers very flexible formatting capabilities, at the price of substantial code to be included in the executable. It is worth noting that, even if the C compiler supports floating point types such as double and float, the present implementation of scanf and printf does not handle it. For many practical purposes however, if one does not need the formatting capabilities, printf can be skipped completely. A more manageable 9963 byte long executable can be obtained with the following code:

```
#include<stdio.h>

int main(int argc, char **argv)
{
    fputs("Hello World!\n", stdout);
    return 0;
}
```

To further shrink the size of the result, an interesting technique (that makes the code non portable) is to exploit a direct PCOS system call:

```
#include<sys/pcos.h>

int main(int argc, char **argv)
{
    _pcos_dstring("Hello World!\r");
    return 0;
}
```

Once compiled, this code yields a much more manageable 2919 byte executable. This size is still much greater than the one that can be obtained with a pure assembly program, but can be acceptable. A list of the PCOS functions callable from C can be found in the pcos.h header, which closely follows the description done by Olivetti in the manual dedicated the assembly language suite [8]. The -Os and -O2 options of gcc can be used and tell the

compiler to optimize the code respectively for code or for speed. In both cases, the simple "Hello World!" program yields a 2897-byte executable. Note in the last example the use of the \r code, the newline used by PCOS in place of \n.

In my experience, it is a good practice in C to adopt a modular strategy and keep separated from the program core the routines related to input and output. When porting a relatively large program to a new computer, the latter often require an adaptation. Non portable code (such as PCOS system call) shall be confined in this part of the code.

If you would like to mix Z8001 and C code, or if you want to use the z8001-pcos-as assembler alone, this is perfectly possible. The compiler manual [9] includes some detailed instructions about how to do that and contains many example programs. If you are used to the Z80 assembly, you may find it interesting to learn the Z8001, as it was meant to be the 16-bit successor to the Z80, exploiting a segmented memory paradigm and preserving a certain degree of compatibility.

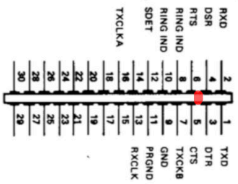
4 - Executing programs in MAME

In order to execute the Hello World program described above, we need to transfer it first into a usable disk image. MAME can read different types of disk images, the most useful file format to be used with the M20 has the extension IMG (in some older versions of MAME, only those in the MFI format could be written). There is a certain number of details to be considered when creating usable disk images, due to the head 0/track 0 format that is different from the rest of the disk in the PCOS disk organization. As said previously, a good bootable image that can be used with the emulator is the pcos20.img file, contained in the archive available from [7].

We are going to need the m20floppy utility described in [10]. Download and compile it with make, in order to obtain an executable called 'm20'. Once created and installed in your computer, to obtain a disk image called hello.img, type:

```
$ m20 hello.img new
```

By the way, m20floppy supports several commands: launch it with no arguments to obtain a brief description of each of them. At this point, the disk image is not yet usable, as the utility does not create the contents of head 0/track 0. They must be transferred manually from a disk image that contains them. The disk image example.img present in the same archive as the PCOS disk can be used for that:



Female DB9	M20 connector
2 (RXD)	1 (TXD)
3 (TXD)	2 (RXD)
6 (DSR)	3 (DTR)
4 (DTR)	4 (DSR)
7 (RTS)	5 (CTS)
8 (CTS)	6 (RTS)
5 (GND)	9 (GND)

Figure 2: RS232 cable pinout

```
$ dd conv=notrunc if=example.img of=hello.img
bs=4096 count=1
```

Benjamin Eberhardt suggests in [4] a simple way to check if a disk image contains the data corresponding to head 0/track 0 or not. You have to inspect the first bytes of the file to see if they are different from zero. If the dd command was successful, here is what you should obtain from an image that can be successfully used in the emulator:

```
$ hexdump hello.img |head -n 1
0000000 01 04 00 23 02 10 01 00 00 0a 00 c4
00 86 1e 00
```

and here is the result with an image that can not be used, as track 0 data is missing:

```
$ hexdump bad.img |head -n 1
0000000 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

When you have a complete image of an empty disk, you may want to copy the result to another file, to avoid having to repeat the process each time. We can then add the executable program to the disk image:

```
$ m20 hello.img put hello.cmd
```

You may check the contents of a disk image using the ls command of m20floppy:

```
$ m20 hello.img ls
hello.cmd
```

Once the disk image contains the executable, we are going to launch the emulator in a window, with a system disk pcos20.img in the disk drive 0: and the image hello.img in drive 1:. If both files are available in the current directory that also contains the ROM file m20.zip, the command to launch MAME is:

```
$ mame m20 -bios 0 -rompath . -flop1 pcos20.img
-flop2 hello.img -window
```

If you have put the files elsewhere, change their paths accordingly. In the emulator, after the machine has finished booting, we can type 'hello' (or simply 'he') and the Hello World program should be executed, as shown in figure 1. One may notice that we did not have to select the drive, as one of PCOS's quirks is that if a file is not found in the current drive, the other one is scanned, too. The last accessed drive becomes the current one. If you have problems with the keyboard layout, you can mitigate them by running 'sl' that allows you to change the current language. The command 'ps' saves the current PCOS configuration and the save will be permanent, as recent versions of MAME can write to file images in the img format. If you want to have descriptions of the error messages more explicit than a numerical code, you can use the 'ep' command, at the expense of 1240 bytes of free RAM. If you are getting mad at the backspace key apparently misbehaving like a Carriage Return, in [1] I suggested a simple fix for that.

The current state of the MAME emulation of the M20 is that many things can be done, but the emulation may be unstable (a warning message is in fact issued by MAME). The emulator is invaluable nonetheless for preliminary testing, as transferring files to a real machine is not entirely trivial and requires some time and effort, as we are going to see in the next paragraph.

5 - Transferring files to a real M20

There are different strategies available to transfer files towards a real M20. If you have an MS-DOS computer with a 360 KB floppy disk drive, you can use Dwight Elvey's wrm20 and rdm20 routines, as described in [11]. There are limitations, mainly because of the peculiar formatting of the track 0/head 0, that is not handled by many disk controllers in the PC world. Usually, a way to circumvent them is to format a disk on the M20 and write it on the PC using Dwight's tools, which simply skip the tracks that can not be written.

Figure 3: File transfer in action

In my case, I do not own a suitable PC and I preferred to make an RS232 null-modem cable to attempt data transfer with protocols such as XMODEM. Figure 2 shows the connections of the cable. I represented the numbering of pins in a male DB9 connector as they appear this way on the solder side of the female connector to be used for the cable. On the "modern" side, I used an USB-RS232 interface that I bought many years ago, working reliably with macOS. I wrote a small collection of utilities in BASIC described in [12] that can be used for this task. Starting from scratch may involve copying a XMODEM receive program on the M20 and then use it to transfer the more involved tools. Instead of directly typing the program, once the M20 is connected, one may redirect the input and output of the PCOS towards RS232 with following commands:

```
pl ci
rs
sc com:,9600,none,0,8,half,off,256
ci 0,o,0
+Scom:, +Dcom:
```

The first commands load the 'ci' utility as well as the RS232 driver into memory and configure the M20 for a 9600 baud 8N1 connection, with no echo nor XON/XOFF control. Then, a serial connection is open. Finally, the last command redirects input and output towards RS232. On your modern computer, if you configured the terminal program correctly (I use Minicom), you should see the PCOS prompt appearing in your terminal, replicating what the M20 writes on the screen. This is a quite convenient way to use the M20, as you can control the computer remotely. You can for example launch basic by typing 'ba' and copy/paste the whole xreceive.bas program. To do this, you should first configure your terminal program to apply a delay for each key. BASIC is not fast enough to process data continuously fed by a modern computer and the result would become mangled after a few lines. On Minicom for example, type CTRL+A, then T and set the 'TX delay' to 10 ms. You may save the transferred file (as 'xmodem.bas'), then restart the machine and reissue the first four commands seen above (as the I/O redirect must not be active to transfer files with XMODEM) and finally load and run 'xmodem.bas' within BASIC to transfer files.

Figure 3 shows a file transfer between my MacBook pro using Minicom and the Olivetti M20, thanks to a USB to RS232 interface and the cable I built. Figure 4 shows the Hello World program running on my machine.

6 - A non-trivial example: memory access for graphics

Of course, programming in C offers a great deal of

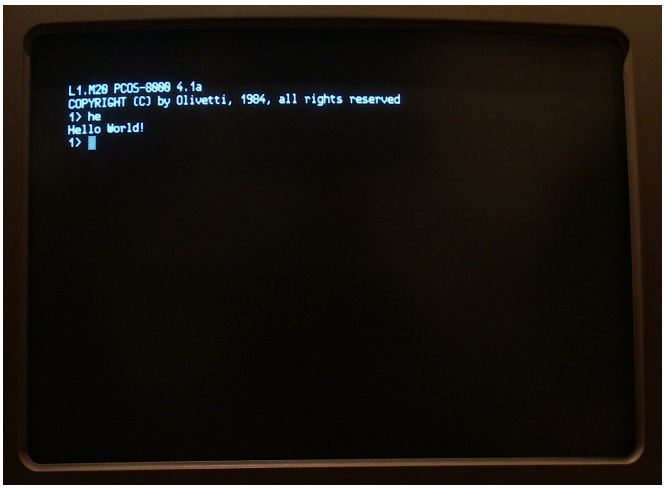


Figure 4: Hello world on a real M20

possibilities and the code in listing 1 shows two functions and a macro that can be used to draw on the screen by directly accessing the memory (on a B/W machine):

- The 'scrclear' function clears the screen (there is no difference between graphics and text modes on the M20, the screen always displays graphics).
- The 'PSET_M' macro draws a pixel on a grid of 512x256.
- The 'line' function draws a segment with the Bresenham algorithm [13].

The result can be seen in figure 5. Of course, such an implementation may be improved, but gives an idea of the expressiveness of the C language. If you really feel the need to get your hands dirty, the gcc manual [9] describes in detail the integration of C code with Z8001 assembly, taking for example different versions of the 'scrclr' function. As said earlier, if you already are familiar with the Z80 assembly language, you may find yourself at home with the Z8001, after all. Among the tools that come with the GCC compiler, the z8k-pcos-as assembler is quite powerful and convenient.

Conclusion

In this article, we briefly described how to cross-program the Olivetti M20 focusing on the C language. After a short introduction, we discussed the tools that we choose for the task, namely a special version of GCC tailored for the Z8001 processor and the PCOS operating system, as well as the MAME emulator.

We then introduced the classic Hello World program and we saw how to reduce the size of the executable produced by the compiler. We discussed how to execute it in the emulator and how to transfer files on a real machine. We finished our discussion by presenting an example of direct memory access. The compiler manual [9] written by Chris is definitely worth reading if you want to go beyond what I describe in this article.

By the way, I almost forgot! This article (as the one you read last month) has been entirely written using Oliword on my Olivetti M20. Text files have then been transferred using RS232 on a modern MacBook Pro, where the final editing has been done.

All the source code discussed in the article is contained in an archive available at [7]. It contains disk images of the discussed examples, as well as the Olivetti M20 version (they are available for many 8 and 16 bit computers) of two text adventure games I developed: The Queen's Footsteps and Two Days to the Race. Enjoy!

Acknowledgments

I would like to thank Christian Groessler for the amazing tools, the constant commitment to the M20, as well as for the countless fruitful discussions we had in the last fifteen years. Concerning the MAME emulator, I could never be able to emulate an M20 without the help of Benjamin Eberhardt, to whom I would like to express my gratitude. Benjamin also kindly prepared the PCOS disk images in the IMG file format and provided useful remarks on early versions of this article.

This paper would have been probably awkward to read without the kind and attentive proofread by Chris Carter. The remaining errors are mine.

BIBLIOGRAPHY

[1] D. Bucci "The Olivetti M20 and the history of a website" RetroMagazine World #2, August 2020.
[2] C. Groessler, personal FTP site: ftp.groessler.org
[3] C. Groessler, D. Bucci "Cross-programming for the Olivetti M20 using GCC," available at http://www.z80ne.com/m20/index.php?argument=sections/download/z8kgcc/z8kgcc.inc
[4] B. Eberhardt, "Emulating the M20 with MAME," available at: www.z80ne.com/m20/index.php?argument=sections/tech/mame_m20.inc
[5] MAME official website: https://www.mamedev.org
[6] Olivetti M20 ROMs available at https://wowroms.com/en/roms/mame/olivetti-l1-m20/89051.html
[7] http://www.retromagazine.net/download/m20inC_sources_and_disk_images.zip
[8] Olivetti "M20 Assembler language user guide," release 2.0, March 1983
[9] C. Groessler, "GCC Z8001 user manual," 2009, available at: http://www.z80ne.com/m20/sections/download/z8kgcc/z8kgcc.pdf
[10] C. Groessler, "Manipulate disk images," available at: http://www.z80ne.com/m20/index.php?argument=sections/transfer/imagehandle/imagehandle.inc
[11] D. Elvey "How to read and write disk images for

the M20 system," available at: http://www.z80one.com/m20/index.php?argument=sections/transfer/imagereadwrite/imagereadwrite.inc
[12] D. Bucci "Transferring files using a RS232 connection" http://www.z80one.com/m20/index.php?argument=sections/transfer/serial/serial.inc
[13] N. Johnson, "Advanced Graphics in C," ed. Osborne, McGraw-Hill 1987. http://www.z80one.com/m20/index.php?argument=sections/download/z8kgcc/z8kgcc.inc

Listing 1: C code for direct access to video RAM

```
/* Segment #3: begin of video RAM for a
B/W machine*/
unsigned short *screen = (unsigned
short *)0x3000000;

#define SCREEN_WIDTH    512
#define SCREEN_HEIGHT   256
#define SCREEN_SIZE     (SCREEN_WIDTH /
16 * SCREEN_HEIGHT)   /* words */
#define ABS(a) ((a)>0 ? (a):(-a))
#define MAX(a,b) (((a)>(b))? (a):(b))
#define SIGN(a) ((a)>0 ? 1 : ((a)==0 ?
0 : (-1)))
#define TRUE -1
#define FALSE 0

/*  Fills the screen memory with a
defined word. */
void fillscr(unsigned short p)
{
    unsigned short *s;
    for (s=screen; s <
screen+SCREEN_SIZE; ++s)
        *s = p;
}

/*  Just turn on a pixel by accessing
directly to the video RAM. */
#define PSET_M(x,y) *(screen +
(((y)<<5) | ((x)>>4)))|=1<<(15-((x) &
0x000F))

/*  Plot a line using the Bresenham
algorithm.
    from Nelson Johnson, "Advanced
Graphics in C"
    ed. Osborne, McGraw-Hill 1987. */
void line(unsigned short x1, unsigned
short y1,
    unsigned short x2, unsigned short
y2)
{
    short dx=x2-x1, dy=y2-y1;
    short ix=ABS(dx), iy=ABS(dy);
    short inc=MAX(ix, iy), plotx=x1,
ploty=y1, i, plot;
    short x=0, y=0;

    PSET_M(plotx,ploty);  /* Plot the
first pixel */
    for(i=0; i<=inc; ++i) {
        x += ix;
        y += iy;
        plot=FALSE;
        if (x>inc) {
            plot=TRUE;
            x-=inc;
            plotx+=SIGN(dx);
        }
        if (y>inc) {
            plot=TRUE;
            y-=inc;
            ploty+=SIGN(dy);
        }
        if (plot)
            PSET_M(plotx,ploty);
    }
}

int main(int argc, char **argv)
{
    int i;
    fillscr(0);
    for (i=0; i<512; i+=10) {
        line(0,0,i,128);
        line(0,255,i,128);
        line(511,255,i,128);
        line(511,0,i,128);
    }
    return 0;
}
```

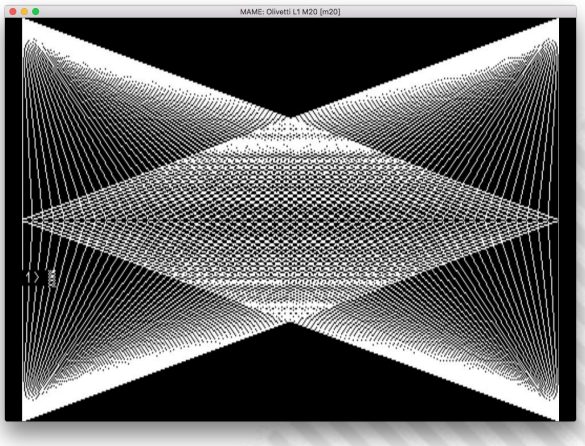


Figure 5: The result produced by listing 1

Japan 13th episode:
Nintendo G&W, a challenge to immortality.

by Carlo N. Del Mar Pirazzini, Giorgio Balestrieri, Michele Ugolini

Dear readers, welcome to this special review of RMW25.

Here is the playlist of the topics covered:

A) Introduction
B) Giorgio Balestrieri's collaboration with MADrigal (aka Luca Antignano)
C) Mario's Cement Factory porting
D) GIG Tiger returns
E) Cuphead porting
F) Are there other simulators and emulators?
G) Nintendo G&W 35th Immortality Challenge
H) Conclusions

We will talk about the past, present and future of G&Ws. We will talk about the immortality of these electronic wonders, born of Japanese genius and powered by numerous enthusiasts who, on a planetary level, continue to invest energy and money in these sacred objects. This time we will talk in two: the article will be written by us two editors: Michele Ugolini and Carlo N. Del Mar Pirazzini.
A third editor of RMW , on this occasion, will be interviewed: the lights will be focused on our Giorgio Balestrieri.

A) INTRODUCTION

Michele:

Talking about Nintendo in Japan means talking about the solid building bases of their building. Base based on reinforced concrete. Structures that rest on a sturdy base, although isolated from it. Constructions that rise to the sky using highly technological synthetic materials, anti-seismic, made in Japan, equipped with a certain year of birth and a predetermined year of demolition. Studies of materials that every year improve their technology, reinventing themselves, after absorbing previous mathematical models. This is tremendous from an Italian point of view: in Italy the same homes are frequently repaired "in saecula saeculorum", often with the same techniques and materials as in previous years. Why am I dwelling on these points? The reason is simple, Japanese constructive dynamics reflect the same "modus operandi" also on G&Ws.

As a good curious man I have always looked and I have always been fascinated by Japanese construction sites, since I have had the "pleasure" of absorbing so many earthquakes. In fact, often, finding myself at various levels of height of a building, facing the windows, I was muted, frightened and amazed to see the skyscrapers swaying among them, not as thin grass, but as imposing mental challenges that contrasted, live, the powerful physical rules of Nature. Equally solid and perfectionist seem to be the constructive foundations of the G&Ws. Deep down, the G&Ws are anchored to their sacred origins, "inescapable" just like the soul of the Japanese. On the surface, they experience the evolutionary stress of impressive graphics (see Playstation and Xbox, see figure 1),

Figura 1

taking blows on blows, oscillating because of the earthquakes of modern marketing. Shocks on shocks, without bowing to failure. Unbelievable.
I have always been fascinated, in addition to their buildings, also by the "stylized immortal soul" of the G&W, so impressive in its functionality, equipped with such bold security that, with a straight back, it can challenge the powerful and inexorable laws of Time! In fact, the date of demolition, or rather, abandonment, of this ambitious and imperturbable project of cyclical reinvention of G&Ws has not been defined. Therefore, it seems to be a challenge to immortality: here is the title of this review. In this article, with the collaboration of Carlo N. Del Mar Pirazzini and Giorgio Balestrieri, various topics related to Game&Watch will be discussed: we will discuss the solid development that to date, 2020, seems to be heading seriously towards the future.

Carlo N. Del Mar Pirazzini:
Early '80s, roaring years. Nintendo released that series of "scacciapensieri" (italian word that can be roughly translated as 'free your mind') known as Game & Watch.
I loved Mario and Donkey Kong. I loved these magic boxes that made us dream of worlds, adventures, challenges...

A time box that will always be in my memories. These memories of mine were awakened thanks to the passion of many fans. Today I analyzed two ports, just released: Mario's Cement Factory converted to C64 and the tribute to Cuphead.
The timebox has been opened!!

Have a nice trip.

B) GIORGIO BALESTRIERI'S COLLABORATION WITH MADRIGAL (AKA LUCA ANTIGNANO)
Michele:
RMW: "Giorgio, welcome to this interview with RMW25, today the honors go to you who collaborated with the brilliant Luca Antignano for the G&W porting. How was your collaboration born?"

GB: "Hi Michele and thank you for giving me the opportunity to experience being 'on the other side' in an RMW interview. The collaboration with Luca began in the early months of 2017, when I wanted to start playing again with the G&W who had cheered up my days as a teenager (i.e. 30-35 years ago). I had known Luca's crazy project before, but I had never really tried it before those fateful months when nostalgia took over. At that time I used Linux, in the form of Ubuntu Mate, as the main OS for some years now but the MADrigal emulator package was distributed only for Windows systems. After a few tests, I discovered that I could easily run them on Linux using Wine and I wrote to Luca to give him feedback on my experiments. Talking to him, I discovered that there was a G&W core per bookshelf, created by Andre Leiradella and released as an opensource, which I could use to run the simulators natively under Linux. Hence the idea of trying to make them available also for RetroPie, the main distribution for retrogaming on SOC systems and beyond."

RMW: "What porting have you tried? Which software and hardware assets did you adopt and above all what problems or bugs did you have to deal with?"

GB: "Actually, I didn't do any porting, the real

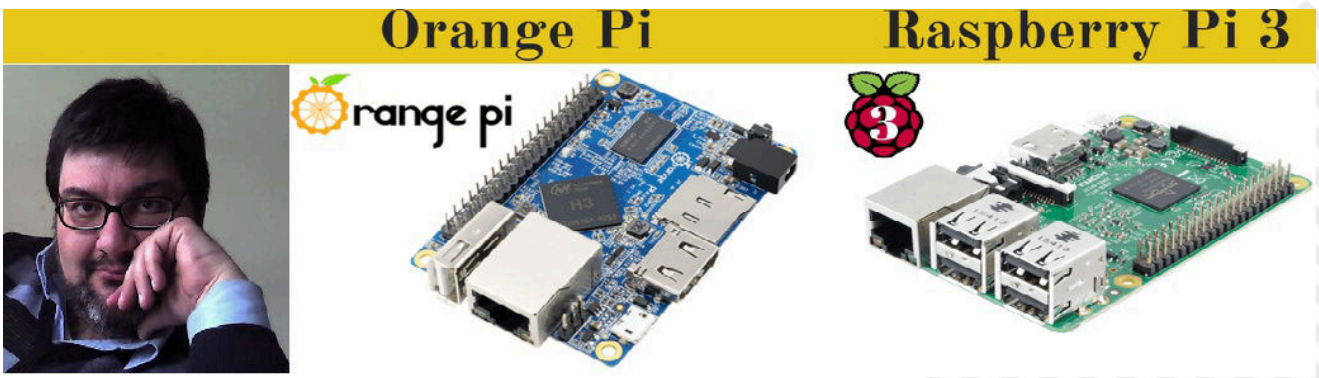


Figura 2

transposition work had already been done by Andre, I just incorporated it on RetroPie and, while I was there, on OrangePie (the version of Retropie for SOC Orange). Andre had created the G&W core to be included in libretro, the RetroPie's main multisystem emulator, and a transpiler to convert the Delphi sources of Luca's games into LUA that, once compiled, could be seen as "roms" and run thanks to the libretro G&W core. As I said, my contribution was simply to package everything so that it could be installed and used through RetroPie."

RMW: "Was there anything curious or noteworthy you noticed during your work? Minimal or important differences between the two Orange/ Raspberry systems?" (see Figure 2)

GB: "No, actually getting a working version of the simulators was a fairly straightforward process on both platforms. Both RetroPie and OrangePie are based on Debian Linux, so once you retrieve the libretro-gw sources from GitHub, compiling them was pretty simple; the maximum difficulty was finding the libraries needed for the process, but absolutely not complicated, Andre did a great job. Among other things, in today's versions of RetroPie the need to recompile the G&W core is no longer necessary since this core is already available in the distribution by default, but at the time only a rather old version was included and in order to run the full Luca's package it was necessary to compile everything by hand. The idea behind my Retro/OrangePie package actually comes from here, from the desire to quickly and easily make available the G&W simulated by Luca to all users of these retrogaming distributions. To simplify the installation as much as possible, I was able to create a single executable file of a few tens of megs to be copied to the target machine, launch and wait for it to finish its work. At that point we found ourselves with an updated G&W core, all the simulators installed and all the snapshots available, there was nothing left but to play. I'm talking about the past because the last time I took care of updating my installation package was in 2019; I should probably take a look at it and see if anything else needs to be touched up. Luca in the meantime has stopped producing simulators but libretro has moved on, maybe there are some adjustments to be made."

RMW: "Do you have any anecdotes or curiosities related to the world of G&W? Do you collect them and if so, do you have any rarities or are you hunting any G&W in particular (maybe some readers could help you in the hunt)?"

GB: "An anecdote yes, involving both G&W and myself. I remember that at the age of 16 or 17 I was on the beach with my parents and I had brought a G&W to spend time under the umbrella. It was one of the multiple-screen ones, that type of G&W that were able to show different game frames depending on the level, turning on and off some graphics of the screen appropriately. This in particular had a small pirate as the protagonist (see Figure 3) who in a first phase of the game had to transport a series of bombs from the pirate ship to a cave along the screen from left to right and in the second one load the treasures stolen from the cave on the ship traveling from right to left, showing a different

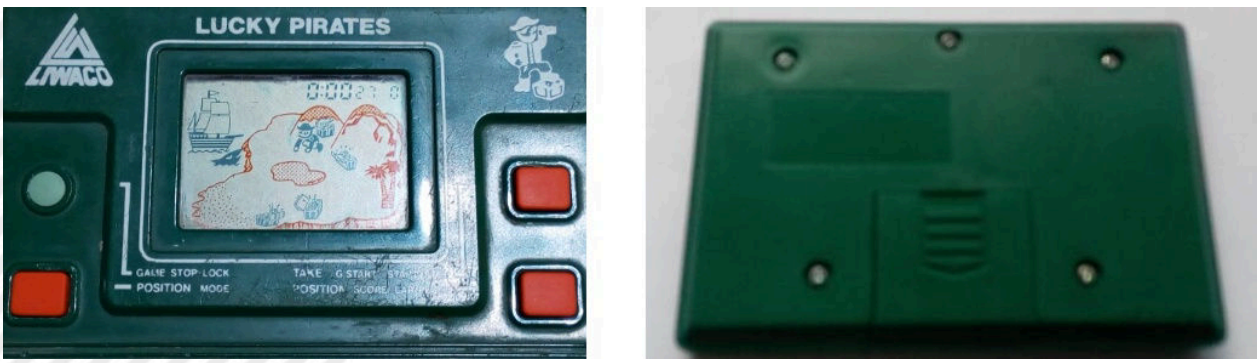


Figura 3

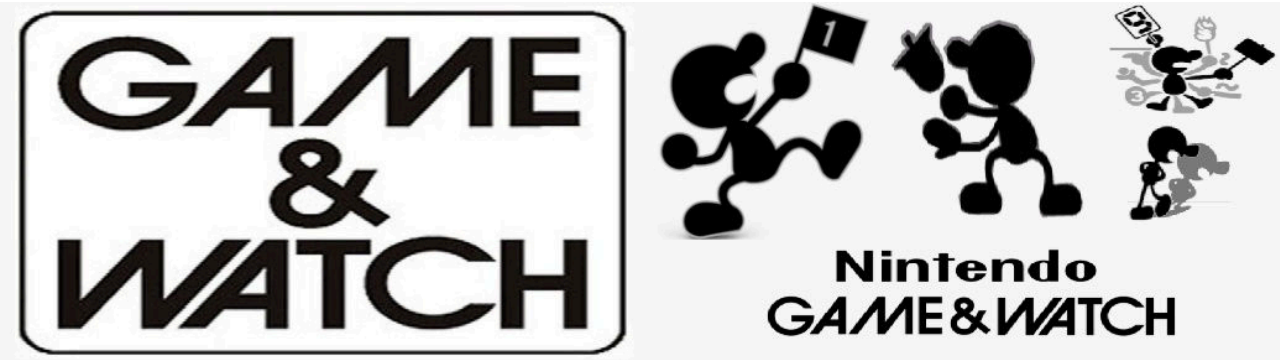


Figura 4

screen for each phase thanks to the “trick” mentioned above.
Well, the point is, after a while I was so focused on the game that I literally forgot where I was and what was around me, so caught up in the action.
After quite some time (I think a couple of hours), I managed to beat the game, accumulating so many points that I zeroed the score and stopped.
Reacquainted with the knowledge of reality, I realized that practically all the umbrella neighbors were watching me with expressions between the curious and the amazed: following the game rhythm, faster and faster, I had taken an absurd pose on the deckchair and had started to move my hands and fingers frantically, with very fast clicks, which attracted the attention (and concern) of those around me. Today such behaviour is not surprising, given the sometimes pathological relationship with smartphones, but at the time I seriously risked someone throwing a bucket of water at me to make me “recover”...

As for the G&W collection, I do have some, I'm a console collector and I've got most of them from Channel F to PlayStation 4, but I don't collect G&W, the ones I own I just took to change the fleet a bit. I'm basically only oriented towards consoles, both because some of them I can program, and because it's thanks to video games that I discovered what I wanted to do in life.

My first console was an Atari 2600 on which I spent, like many of my generation, an unworthy number of hours playing.

Then, as a good pre-teenager, I wanted to understand what video games were like, which led me a few years later to discover the wonderful world of programming, whose art later became my profession."

Oh, that's good. We thank our Giorgio for the funny anecdote, the valuable interview and the important work done, donated to the G&W community (see figures 4 and 5). Now the road map continues with other curious and delicious stops.

Figura 5

C) MARIO'S CEMENT FACTORY PORTING
Carlo N. Del Mar Pirazzini:

Mario's Cement Factory (see Figure 6)
Publisher: hayesmaker64
Year: 2020
Genre: Game & Watch/Platform
Platform: Commodore 64

Iconic. Difficult to find a different word for Mario and Game & Watch.
This conversion is based on the original game, released in 1983 developed for Nintendo Game & Watch under the guidance of that genius Gunpei Yokoi.
Converted for Gameboy, Gameboy Advance and for Nintendo DSi is considered one of the strangest games in Mario's franchise.
The player controls Mario at work in a cement factory, where he will have to fill concrete mixers with concrete falling from the assembly line.
You will have to do it quickly and without making them overwhelm under the eyes of the employer. To do this, you will need to activate the levers on three levels that will allow them to descend into the truck.
This C64 port was made in Assembly and I must admit it maintains the same frantic spirit as laptop gameplay.
The difficulty curve is always gradual, and you let yourself play pleasantly.
Fast and well structured, I'm sure it will give a good dose of fun to those looking for a quick game to charge to relax.
The only flaw is obviously the repetitiveness that affects global longevity, but it is to be considered an excellent port.

Final judgement

Gameplay: 80%
It embodies the spirit of Game & Watch and puts it in a C64. Well developed.

Longevity: 65%
It has the spirit from Game & Watch, it must keep you glued together for a few minutes. Otherwise in the long run can be tiring.

D) GIG TIGER RETURNS
Michele:

Dear readers, there are great news. The Gig Tigers, the portable video games that made the history of the 1990s, are coming back.

Here's the preview from "The Verge":
https://www.theverge.com/2020/2/19/21136607/hasbro-tiger-electronics-lcd-handheld-games-xmen-sonic-transformers

Hasbro has announced that they will be available in the United States starting next summer 2020. Stop! Stop! Everybody stops! It's already 2020 and it's autumn! In fact, this very fresh news is dated February 20, 2020.
We haven't talked yet about this due to damn Covid19. Then it all fell into oblivion.
Tiger Electronics, portable video games that became famous in the 1990s and known in Italy as Gig Tiger, unfortunately did not arrive in summer 2020. The initiative promoted by Hasbro included four games: The Little Mermaid, Transformers Generation 2, X-Men Project X and Sonic the Hedgehog 3. They were

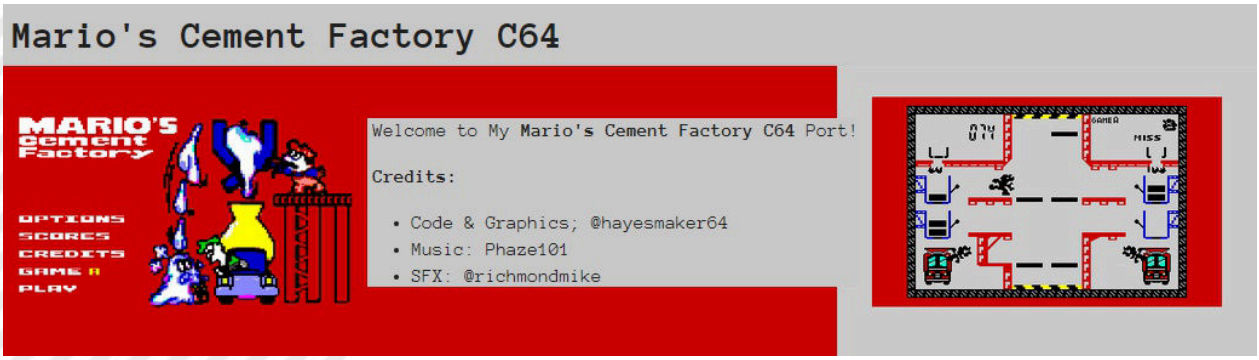


Figura 6

bookable by GameStop in the United States for $14.99 each. Probably in Europe the price would have been 13.99 Euro. At the time, the price of the glorious Tiger GIGS was twenty thousand lire. The price of the remake seems good.

No information had been released regarding the sales expansion in Europe, the official news would only be provided at the toy fair in New York. As Morpheus said in the Matrix: "Fate, as we know, does not lack a sense of irony." The updates of these games were soon overwhelmed by the tragic news that we all know.

In this link the official information so far not updated. (see Figure 7). Available, but unfortunately still frozen:
https://nypost.com/2020/02/19/hasbro-bringing-back-beloved-1990s-toy-tiger-electronics-hand-held-games/

Tiger Electronics created, many years ago, also the "Furby", the "Game com" console, the interactive robot "2-XL"(based on audio cassette), the "R-Zone" and finally this line of handheld LCD, similar to the famous Nintendo G&W. The Tigers were very small consoles with a monochrome Lcd display and a sturdy shell: they looked indestructible. Each console was equipped with a single video game with attached mini speaker for music and effects. Ergonomically they were well designed, the solidity was undisputed, the audio was acceptable, and the price was competitive. Nintendo had found a worthy rival in this sector. Will the presence of THE TIGER GIGS in those days have stirred upheaval in the house of the great "N"? My opinion is: no. Nintendo was proud about other clones, in fact in the house of the great "N" there was obviously a Japanese philosophy that recited: marketing only recalls further marketing.
At the time, the Tiger versions were numerous: Street Fighter, Double Dragon, Golden Axe, Castlevania, Robocop, Hook, Spider-Man, Batman, Flash, Snow White and the Seven Dwarfs, The Little Mermaid and even a game dedicated to Michael Jordan! From the early eighties to 2003, the year of disposal, 184 games were created, which can be consulted in the catalogue of the Handheld Museum website:
http://www.handheldmuseum.com/Tiger/index.html

Tiger Electronics was founded in 1978 and produced its LCDs independently until 1998, when it joined the large Hasbro family. Today, in 2020, Hasbro himself sensed the widespread nostalgia for these sacred objects (at least for us as collectors). Here it is, finally in 2020 the Hasbro Tiger LCD was born. Unfortunately, it was born in a very complex historical moment. We all hope that an effective solution will be found against this bloody epidemic, so that they can restart all the world's productions. Let us not despair, everything has been slowed down, the damage is great, the losses and pain are immense, but not everything is lost. In the meantime, to deceive the time, I'll let you know of a glamorous curiosity: many Tiger GIGS are perfectly emulated thanks to Jason Scott who with his team has made available "Handheld History", an archive on the web that allows you to emulate about 60 of those titles on our browser.
The project is updated from time to time with new games, many of which seem to be as perfect as the

Figura 7

original ones. Here is the link:
https://archive.org/details/handheldhistory

E) CUPHEAD PORTING
Carlo N. Del Mar Pirazzini:

Cuphead: Game and Watch Edition
Publisher: independent
Developer: Simon Delavenne, At0mium, Heelio
Genre: Platform
Platforms: Web

A tribute to a jewel that itself proves to be a precious gem. (see Figure 8)
This web version of Cuphead is a pleasant surprise to all fans of G&W and the title developed by MDHR.
A game developed with the Unity WebGl engine, which recreates on your screen the typical G&W of the past and puts you at the helm of the funny Cuphead grappling with a terrible carnivorous plant that throws feathers to jump and, at the same time, will try to capture you with its branches.
Well made, with few commands (right, left, on top of a key used for the start of the game) and above all very playable.
What can I say, maybe a little monotonous, but definitely effective.
A tribute to one of the best run'n gun games of the last 10 years.

Final judgement

Gameplay: 85%
Simple and linear. Well developed.

Longevity: 65%
The level of difficulty increases and unfortunately also the desire to try somethingelse. But fun.

F) ARE THERE OTHER SIMULATORS AND EMULATORS?
Michele:

In the last issue we talked about the brilliant Luca Antignano (aka MADrigal).
Have you visited all its valuable links and checked all the material on the sites it has provided us with?
It's an extraordinarily voluminous job, isn't it? Amazingly there is still G&W emulation and simulation material on the web ready to meet our further needs.
Mame is probably the best-known easy to use platform. Maybe it's the most convenient choice ever. He deserves a specific chapter. I will probably talk about it in the future, when a dedicated review is created, comparing Mame together with the particularities of G&W such as unofficial porting, clones, rarities, etc.
In this paragraph I will only talk about the possibility of playing with a G&W/Handheld through platforms independent of MAME.
Obviously not all possibilities: this fantastic world of G&W is changeable, some projects are born, others perish, others are merged, it is extraordinary to think that these small objects can enjoy such health and excellent foresight in marketing! (see Figure 9)

We can start from Retropie/RetroArch abundantly present on the web.
E.g. lr-gw. It's a simulator and not an emulator. This means that the games you can play with are not

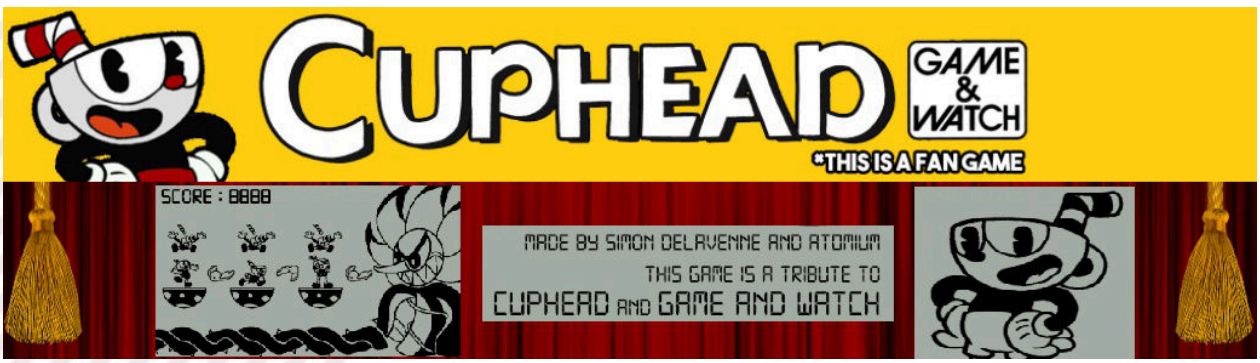


Figura 8

actually the original games, but recreations of the games. Here's the link:
https://retropie.org.uk/docs/Game-%26-Watch/
Here you can find the emulator:
https://github.com/libretro/gw-book

Here you will find explanations for emulation:
https://youtu.be/DzbsfCC77IQ

Want to play with G&W via Java? Nothing more immediate. LCDgame.js is an open source JavaScript library that currently supports authentic representations of Donkey Kong II and Mario Bros.
Here's the link:
http://bdrgames.nl/lcdgames/

Being all opensource you can easily collaborate on the beautiful and streamlined project:
https://github.com/BdR76/lcdgame.js

Let's turn to another very interesting project: HQ.
Here's the link:
http://www.emulator3000.org/hq.htm

Handheld Quake, simulator of Soviet and foreign portable games.

Honestly, I find this project nice because it can be used with both Linux and Windows. The list of Russian games is: Mysteries of the ocean, Just you wait!, Merry cook, Explorers of space, Autoslalom, Merry arithmetics, Space flight, Fisher tom-cat, Hockey, Fowling, Space bridge, Rhythm.
The list of Nintendo games simulated through this platform is as follows: Chef, Egg, Octopus, Fire, Mickey Mouse.
Try it. You're gonna love it. Let's hear it for the developers. Congratulations very much!

Let's not forget about Pica Pic, Hippopotamus.
They are G&W that can only be played online, taking advantage of Adobe Flash.

The creators are Polish and have created a fun and immediate site for a fast gameplay! Here's the link:
http://www.pica-pic.com/

Their old site was this:
http://www.hipopotamstudio.pl/

Want to talk about Android?
Every now and then G&W/Handheld is added to Google play. Just type "Alice game & watch" and one of the G&W I love comes up! The creator "datsuryoku_k" of Kanagawa, Japan, has developed many games and some of them are really fun. I will leave you the pleasure of browsing the full list of results. Some developers have been very good, others very questionable.
Other creators to mention? Definitely: AviSoft, Pixelegend, Lapigames, Mascot1039, Yanmania, Million Rabbit Studio, Short2Games, etc.

The more you navigate the suggestion links and the further you move away from the purist concept of G&W... it's up to you when to stop your journey to... well... the dark meanders of bad taste!

Shall we talk about Zophar? Here's the link:
http://www.zophar.net/gw.html

Figura 9

You'll find plenty of games ready to download. They work with Windows. The list is attractive: Armor attack, Ball, Balloon fight, Banana sbang, Camelot, Circus circus, Donkey angler, Donkey kong, Donkey kong circus, Donkey kong 2, Donkey kong jr, Dungeons and dragons.

Finally, pay attention to some broken links and abandoned projects. For example:
http://www.handheld.remakes.org

Is that enough? I'm pretty sure I haven't listed all the gameplay possibilities for these amazing G&Ws. I'm sure that in the next review there will be more material. Help me in the hunt, dear readers!

G) NINTENDO G&W, IMMORTALITY CHALLENGE
Michele:
Just as I was writing this article, this announcement from Nintendo appeared on the afternoon of September 3, 2020: "The classic console that revolutionized the world of video games makes its return! On the 35th anniversary of Super Mario Bros., don't miss Game & Watch: Super Mario Bros."
It seemed incredible, I was writing an article entitled "G&W, Challenge to Immortality" and Nintendo revealed the news of an official G&W remake, as well as other games, for the 35th anniversary of the birth of Super Mario Bros. These are real projections into the future!
Here are some links:
https://www.nintendo.it/Notizie/2020/sepmbre/Guarda-subito-il-Super-Mario-Bros-35th-Anniversary-Direct--1836547.html
https://youtu.be/a8DJpeCy8CQ

The advertisement reads: "Overcome chasms, step on the Goombas and enter the tubes just as you remember, but with even more precise controls thanks to the console's + push-button panel. Play alone or alternate with a friend for some healthy competition!"
Then again: "Looking for a bigger challenge? Then don't miss Super Mario Bros.: The Lost Levels! If you prefer a more relaxed experience, Ball (with a special touch of Super Mario) is for you! "
Finally it concludes: "The included digital clock can play 35 different animations, which also include some of Mario's friends and enemies! Keep an eye on the console even when you're not busy saving Princess Peach."

Exactly, that's right, Nintendo will celebrate its most famous brand: Super Mario Bros. (see figure 10)
An official, exclusive, limited G&W dedicated to Mario will be released on November 13. Along with the game, in the G&W there will be some levels "The lost levels" that we remember. In Japan they came out with the title "Super Mario Bros 2" and finally there will be the classic version of Ball with Mario's face instead of "Mr. G&W".

We'll find our funny Mario-juggler ready to intercept the balls falling from above. The watch will also be present. A very special watch, apparently.

A note dear to all of us collectors. You know the FamiCom pads? Well, this G&W will have the shape and color of those pads! It seems that there will also

Figura 10

be various software customizations, graphic setups and especially adorable easter eggs to be discovered as we found Mario's underground pipes.
Nintendo has officially released on November 13th. The price seems to be around €50. Surely the preorder will become a hot place of battle, therefore, open eyes dear readers. Besides, it seems that FamiCom will be the official adopted style, but remember that some versions of the past Nintendo Mini had different shells and games depending on the place of sale: Asia rather than America rather than Europe. I am very curious to know the sales volume of this remake. (see figure 11).
Stay tuned to this news and keep in mind the rules of the game around which, in a very short time, millions of dollars will revolve:

Sony: Playstation 5,
Microsoft: Xbox Series X,
Nintendo: Game&Watch.

Well, we shouldn't laugh. In the house of the great "N" nothing is done by chance. However, only the future can tell us about the past and above all the outcome of the G&W challenge... to immortality!

H) CONCLUSIONS

Michele:
We talked a lot. The G&W world is so vast that more delicious news will arrive shortly. I'm sure of it. Moreover, for us collectors, given the recent news, it is a hectic period.

Instead, I bet Nintendo is in a moment of relaxation: enjoy your earnings before announcing new ultra-powerful consoles, ready to challenge Sony and Microsoft with elegant lightness. Do G&Ws challenge immortality? There's no doubt about it.
Will they win this challenge too? Who knows?

Carlo N. Del Mar Pirazzini:
Wipe the dust off these items, install or try the games we reviewed here. It is a dive into the past and a leap into beautiful memories... and above all a challenge against yourself!!

Are you still able to beat the yourself of 40 years ago??

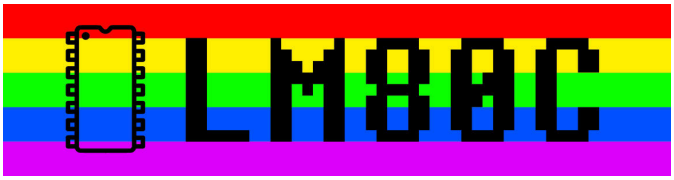
Figura 11

The LM80C Color Computer

A 2019 self-built Z80-based home computer - part 2

by Leonardo Miliani

After the first introduction of the previous article (part 1), where we saw how a computer is structured and what are the basic components that serve its operations, in this second part we will analyze how the CPU interfaces with memory and Input/Output (I/O) devices.
The data communication within a computer takes place



through digital electrical signals, which can take a “high” or “low” value, indicating a value of “1” or “0”, respectively. This type of signal is not only used for the exchange of information but also for the activation or deactivation of devices external to the CPU: since a computer worthy of its name is composed of several chips, it is necessary to design a process so that the processor can select from time to time the chip with which it intends to exchange data. Fortunately, we do not have to invent anything because there are people who did it for us: it is a family of well known integrated products and used for a long time, namely the 7400 family, presented in 1966 by Texas Instruments as an economical version (in plastic packages) of the 5400 family, introduced 2 years earlier. This integrated TTL family contains AND, OR, NAND operators, address decoder, flip-flop, buffer, inverter and more logic port chips. When you hear that computers from the '60s and ‘70s were built using TTL logic, well you may find that they were assembled using just tens, hundreds, of these integrated singles. IBM, Olivetti and other manufacturers built entire computers with TTL logic. A famous computer based on TTL logic was Computer Terminal Corporation (CTC) Datapoint 2200: its CPU, instead of being composed of a single chip (both Intel and Texas Intruments had initially been asked to build a CPU on a single integrated but neither company was able to meet CTC's timing and technical requirements) was composed of 100 7400 logics (Figure 1). I mention this computer, built from 1970 to 1979, because the logical scheme of its CPU was the seed of Intel's x86 architecture.

Figure 1: Datapoint 2200: its CPU consisted of 100 7400 series integrators
(photo: Wikimedia Commons)

Back to the 7400 family, I selected the IC 74HCT139, a 2-to-4 address decoder, meaning that with 2 inputs you can select one of 4 outputs available. This is easy to understand using the powers of 2: since a digital signal can only take the values of 0 and 1, that is 2 distinct values, 2 inputs can direct 2^2 = 4 outputs. The abbreviation of the integrated is composed of 3 parts: 74-HCT-139. “74” is the series, “HCT” indicates the family and “139” identifies the integrated chip. The family identification code is very important because over the years several series have been produced, each compatible with the levels of transistor outputs used in that period. In order for a computer to work, it is important to use integrated devices of the right family, whose types and voltage levels are compatible with the main computer chips (CPUs, audio and video chips, peripherals). Since all the aforementioned chips in the LM80C have been selected as CMOS (the construction technology of the transistors that make them up), I have selected 7400 series integrators belonging to the HCT family, an acronym that indicates integrations made in high-speed CMOS technology and low power consumption. It is good not to mix up different families: so if you want to replicate the computer, but also buy the components to repair yours, get 7400 logics and integrated all of the same type, in the case of the LM80C type CMOS (they can be recognized because in their acronym they have a “C”, for example Z84C0008PEG for a Z80 version CMOS at 8 MHz maximum frequency). For convenience, from now on, 7400 series integrators will be indicated without center letters, assuming they are all from the HCT family.

Figure 2 shows the decoder with the essential connections to its operation. Pin “E” (for “Enable”) is the pin used to enable, i.e. select the IC; pins A0 and A1 serve as selection inputs and the combination of signals on them determines which output, showed by pins O0..O3, it's activated. Let me point out one thing here: you can see that on some pins there is a circle-shaped symbol (in other CADs or on other built-in profiles you can find other symbols, for example a triangle). That symbol indicates that the signal is “active LOW”: this means that for the IC to recognize it as active it needs to be at a low level, or “0” to be clear. Let's go back to the previous article and the pinout of the Z80. You will remember that among the control pins of the CPU system there were 2 with the abbreviations MREQ and IORQ: the first is active when the CPU wants to access the memory, that is, the address presented on the address bus is intended for a read or write operation in memory,

while the second is active when the CPU wants to perform an operation involving a peripheral chip. We have seen that both the Z80 integrated family and the 7400 series integrated ones recognize as active signals those that have a low value: it is easy to understand that all we need to do is connect the Z80 MREQ pin with the 74139 enable pin: when this line goes low, that is, the Z80 will request access to memory, the low level on this pin will enable the 74139 decoder.

If you remember the specifications of the LM80C computer, the memory consists of 32 KB of ROM and 32 KB of RAM. The first one fills the address space (in hexadecimal) from $0000 to $7FFF while the second from $8000 to $FFFF (this is because the Z80, after a reset, starts to execute

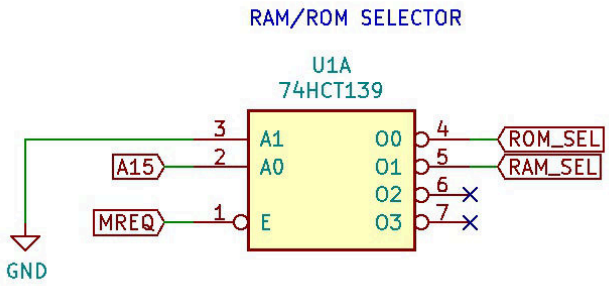


Figure 2: 74139 used as address decoder, to select RAM and ROM

the instructions starting from the address $0000, that's why the ROM must be in the lower part of the address space). To address 32 KB of memory, or 32,768 cells, you need 15 lines: in fact, 2 ^15 from your own 32,768. Now, since the address bus consists of 16 lines, it's easy to understand that we can use the 16th line as the selector of the 32KB memory bank that we want to access. In fact, the addresses $0000 and $7FFF, assigned to ROM, in binary these become 0000.0000.0000.0000 and 0111.1111.1111.1111 respectively; similarly, the addresses $8000 and $FFFF, assigned to RAM, are coded in binary as 1000.0000.0000.0000 and 1111.1111.1111.1111. It is easy to understand therefore that the sixteenth bit (highlighted in bold) assumes the value 1 only when we access an address belonging to the upper half of the address space (i.e. from the cell $8000 included onwards) so we will make sure that pin A15 of the CPU selects the ROM when it is at a low level, and the RAM when it is at a high level. This is done by connecting it directly to pin A1 of the IC 74139. The A0 pin, on the other hand, we can connect it directly to ground, so that it always has a fixed value and the selection takes place exclusively with the change of status of the other pin. On the input pins of the IC 74139 we can therefore only have 2 combinations: 00 or 01. In the first case the integrated will enable the O0 output, connected to the enable pin of the ROM memory chip, while in the second the O1 output, connected to the RAM. Figure 3 shows the RAM and ROM chips (for convenience side by side) with their enabling lines highlighted in red. Because the decoder emits a low signal on the selected output and simultaneously a high signal on the other, only one chip at a time will receive the enable signal. At this point, the signals on the Z80's "RD" (“read”) and "WR" (“write”) pins will be used to indicate to the selected memory chip the operation the CPU intends to perform. It goes without saying that for the ROM (indeed, EEPROM), it will only be possible to read: if you see, the relative “WR” pin is in fact connected directly to the 5V, because it is only used during firmware programming.

We just saw the address decoder in action for the Z80's selection of RAM and ROM chips. As in this case, even when communicating with peripheral units, those signals come into play to select the different chips and others in order to exchange data. Let's go back to our wiring diagram for a moment and see the I/O decoder, a chip similar to

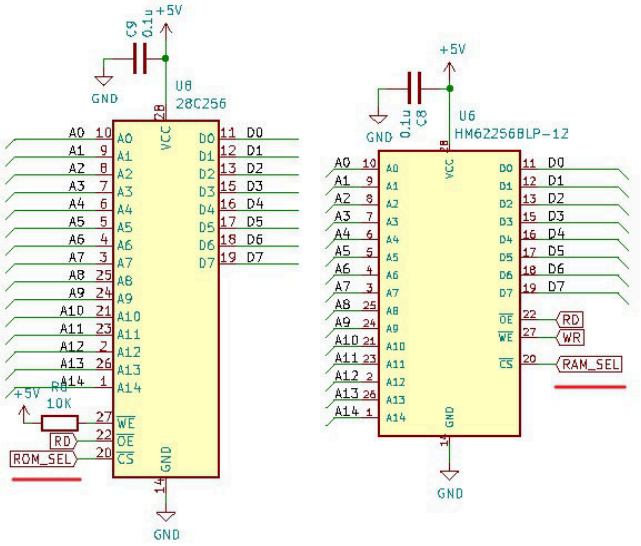


Figure 3: ROM and RAM memories with selection lines highlighted

the one used to select the memories but a little more complex. In fact, this chip has 3 inputs thanks to which it can select up to 8 independent units (in fact, 2^3=8): the IC is called 74HCT138. Why didn't they use the same embedded memory? Because 4 outputs alone were not enough for us, as the peripherals of the LM80C were well 5 (with the possibility in the future to increase further, expanding the computer): we have a chip for parallel communication, one for serial communication, a timer used to generate clock and interrupt signals, and finally the video and audio chips. Each of these embeds must be uniquely selected, otherwise the computer will not work.

Figure 4 shows the connection diagram of the 74138. Unlike the 74139, as previously seen, the 74138 has, in addition to the aforementioned 3 input lines, as many as 3 selection lines of the integrated circuit itself. Thanks to this big number of inputs we can use several 74138 and activate only one when a certain combination of signals occurs on their enabling inputs. Going back to the previous article, where I analyzed the reasons that led me to opt for the Zilog Z80 as the CPU of my computer, you will remember that in his favour there was the management of an I/O address space separate from the memory address space. What's that supposed to mean? Let's briefly explain this by comparing it to a well-known CPU that does not have a separate I/O addressing space, the 6502 (and its derivative chips). Those who owned a Commodore computer will remember that to access the video or sound registers they used PEEK and POKE made to specific memory

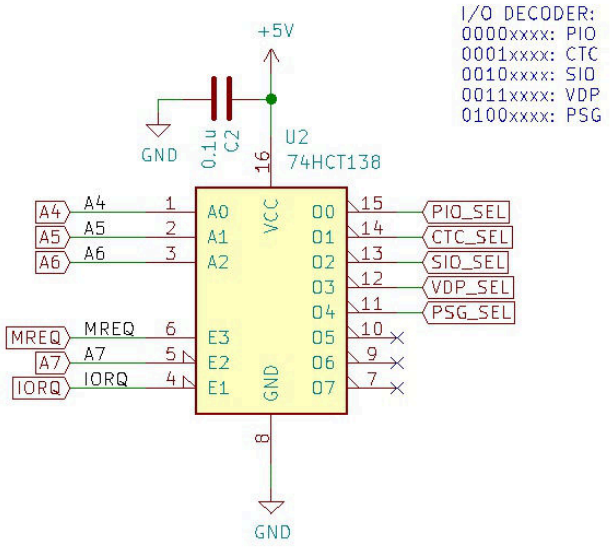


Figure 4: 74138, used as an I/O decoder

addresses: in this case it is said that the registers are mapped in memory, indicating with this phrase the fact that certain addresses are no longer associated with memory cells but serve to exchange data with the peripheral chips: the designers of the computer have built decoding circuits similar to those shown in this article thanks to which, when the CPU selects a particular combination of signals on the address bus, the hardware of the machine goes to select a specific chip. This sacrifices more or less large portions of memory, as the affected address blocks cannot be used more for memory access. A CPU that instead manages 2 separate addressing spaces for memory and I/O allows you to use all the memory addressing space: the Zilog Z80 directs 64 KB, and in fact the LM80C does not lose even one of these bytes, having direct access to 65,536 individual memory cells. In addition, it provides 256 unique addresses for accessing as many I/O drives (it can actually address 65,536, but I don't want to put too much meat on fire, referring to the reading of the CPU data sheet for the affected reader). The discriminant is made by the 2 simple electrical signals already seen previously: MREQ and IORQ. The first one we used to activate the memory decoder, and we will also use it now, in conjunction with the second one, to activate our 74138. Why do we use both signals, if we only used the MREQ signal to select 74139? Because, as mentioned, the 74138 has 3 integrated enable inputs, indicated in figure 4 with the pins called E1/E2/E3. The combination that the 74138 recognizes on the "enable" pins is LOW/LOW/HIGH, respectively for E1, E2 and E3: this means that in order for the integrated to understand that we want to dialogue with him, we must simultaneously ensure that the signals with the aforementioned values reach the 3 feet indicated. As can be seen from the diagram, on E1 and E3 the signals of the IORQ and MREQ pins arrive, respectively, emitted of the opposite value: IORQ is low and MREQ is high when the CPU wants to access an I/O device, while IORQ is high and MREQ is low when it wants to access memory. Consequently, it is easy to understand from the value they take on pins, that the 74138 is enabled when it recognizes the levels of the first combination. Finally, the third pin E2 is connected to pin A7 of the CPU address bus. Why would you do that? A help comes from the decoding table in the image: if you look at the combinations shown, pin A7, which is the eighth pin of the address bus and therefore carries the value of the highest bit of the first byte that makes up the address, always assumes a value of 0: this means that all devices whose address ranges from $00 to $7F, in binary from 0000.0000 to 0111.1111, will be managed by this 74138. This serves to simplify matters for us in the event that in the future we want to expand the peripherals of the computer and assign to another 74138 decoder the task of managing these peripherals or if we assign to some of them an address in the range $80- $FF, in binary 1000.0000 – 1111.1111: here you can see that the eighth bit, highlighted in bold, always takes value 1 (unlike the previous case, where it always takes value 0). Once the enabling pins have been arranged, the input pins that select the outputs of the integrated remain. Referring back to the table, we connected the 3 lines A0/A1/A2 to the A4/A5/A6 lines of the address bus, so that the incoming triplet selects the correct chip: with 000 we select THE PIO (parallel device), with 001 the CTC (timer), with 010 the SIO (serial device), with 011 THE VDP (video chip) and with 100 the PSG (sound chip). We chose the A4/A5/A6 lines of the address bus simply because all the peripherals contain multiple operating units or because they need one or more signals to select the operating mode, reserving these bits as real selectors. For example, the PIO has 2 ports on which you can operate in 2 different modes, and the same is true for THE SIO; the CTC instead has 4 internal channels, while THE VDP and the PSG use an additional pin to indicate to the chip whether the value indicates the selection of a register or whether it should be interpreted as data.

Well, in this article I have dealt with many topics that, I hope, have been interesting and have helped you understand how computers address the various internal components. In the next article we will discuss in detail the use of some peripheral chips.

Useful links

The LM80C project reference web page:
https://www.leonardomiliani.com/en/lm80c/

Electronic diagrams and firmware source code:
https://github.com/leomil72/LM80C

The Hackaday page dedicated to the LM80C:
https://hackaday.io/project/165246-lm80c-color-computer

RUFF 'N' TUMBLE

Editor: Renegade
Developer: Wunderkind
Year: 1994
Genre: Platform/Shoot'em up
Platform: Amiga/CD32

Perfection is not being perfect, but continually striving for it. - JOHANN GOTTLIEB FICHTE

Gentlemen, we are close. Very close to perfection in a video game and everything inside a 1.44 Mb disc (when the world was still simple, right?, NdN).

In 1994 Ruff'n'Tumble on the Amiga market, a market shaken by the failure of Commodore, imposing itself as one of the best securities of that period and perhaps of all time.
Nowadays, the game not only confirms the qualities it showed at the time, but it will blur many that went unnoticed at that time, but that nowadays are worth dusting.

The game is developed on four worlds (divided into four sections) and is a platform game with a strong shooting 'em up vein, today it could be classified between a Run n Gun and a Metroidvania. Developed by the duo known as Wunderkind (Robin Levy and Jason Perkins) it immediately attracts us for its technical realization and transports us into a game with furious but at the same time complex and varied mechanics.
The protagonist is a chubby boy named Ruff Rogers, who must collect a number (indicated by appropriate counters at the top of the screen) of red, green and blue marbles to finish the levels. Without these you cannot access the next level. Meanwhile, blow up everything in front of him!
The moment of "destruction" is possible thanks to five different types of weapons ranging from the simple machine gun to the most powerful laser, through the rocket launcher or the devastating flamethrower. One of the greatest innovations of the game lies precisely in the management of the arsenal: armament has infinite ammunition but does not last forever. A special bar decreases rapidly with the passage of seconds independent of the blows exploded. Once you reach the end you return to the dignified but not "enjoyable" classic shot. To avoid the situation the only thing our fat hero can do is collect the numerous power ups present in the levels. It may seem like a limitation but it

actually makes the action frantic and the fun curve always at its best, allowing you to explore the game in every possible hurdle.

Also remarkable is the conformation of the levels, the disposition and the behavior of the enemies. Designed specifically to force the player to face the path in speed, but also allowing them to explore maps (which are really vast) and face pitfalls with a minimum of tactics.
The intensity of action then becomes buzzing, dispensing entertainment in industrial quantities!

What about the difficulty curve? Ruff'n'Tumble is strict but fair. Never incorrect and always allows you to continue offering, thanks to an excellent control system, various ways to do so. And the control system is the real must-have on which the Wunderkind built this jewel. Even with a single button and despite the excessive inertia, the care with which it was designed allows us almost total control over Ruff, who runs, jumps, swims, climbs and blasts everything with precision and naturalness; which is very rare in many other similar products of those times (and also in many modern games). The result? Excellent gameplay, enhanced by an impact soundtrack!

I will conclude by talking about graphic creation. Clean, full of colors and light effects (remarkable those of transition between light and dark), always perfect animations and an impressive attention to detail. This was possible when programmers were left to work peacefully and passionately and this was what they knew how to get out of the Friend.

Ruff'n'Tumble, my lords, is the swan song of Commodore Amiga. A magnificent song that approaches perfection and demonstrates how that machine was able to go beyond the mismanagement of its managers.
Masterpiece!

by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 95%

Well designed and immense levels, excellent control system, strict but fair and immediately playable.

» Longevity 85%

A great product that will keep you busy for a very long time and that proves that it was possible in a 1.44 floppy disk. You can't miss it now.

GOLDEN AXE WARRIOR

Year: 1991
Editor: Sega
Developer: Sega
Genre: Action Rpg
Platform: Sega Master System

Between the late 80s and early 90s, the SAW felt a strong pressure in front of the Nintendo home playground. Kyoto's most famous brands were Super Mario and Legend of Zelda.
If Mario has to wait a little longer (1991 with the arrival of Sonic), Sega decides to develop a competitor of the well-known Rpg Zelda on the 8-Bit platform called Sega Master system. Between 1989 and 1990, thanks to the Golde Axe brand and its enthusiastic leverage, he created this Golde Axe Warrior.

In those years I went a little deaf, bisected by the magazines of the sector or almost, but I must admit that it is a very good product.

If you are familiar with Zelda, Golden Axe Warrior will feel like home. The plot is classic, we will play in the role of the hero, who came to destroy Death Adder (historical enemy of the Golden Axe saga) and to end his reign of terror in the world of Firewood. In short, the plot is similar to Zelda and the millions of other adventure games. To get to the bad guy on duty, we need to collect nine crystals hidden in nine different dungeons scattered all over Firewood. A further advantage to continue the game is the plethora of hidden secrets, mini-games, cities to explore, side quests to solve and shops.
There are so many areas to explore and locate and, literally, a secret in every world that we will visit. In some places we will be able to explore areas using objects such as canoes, miniboats or other that will allow us to reach areas that are difficult to reach without. This was missing from the first Legend of Zelda, a huge world to explore freely. Each world is well characterized and specific and there is also a beautiful full-bodied section dedicated to magic. Like Zelda for Nes, we can save game progress here. I mean, it's a nice game, classic but compelling.

Graphically it was light years compared to Nintendo's rival. Very careful graphics and well detailed. With excellent use of Saw 8-bit paddles. Very fluid and well animated. Also beautiful is the rather epic opening introduction. In short, a nice display of the graphic capabilities of the Sega Master System.

Very little sound, the music is not negative in itself but in the long run it is nerve-wracking and repetitive and seemed unsuitable in some game points. The Boss music is beautiful and the sound effects in general are excellent.

Graphics
Here's one of the areas where the game really shines compared to Zelda, it has just wonderful graphics. Some of them are not very innovative (paddle exchange bosses always bother me and some may have used a few more

details), but they are still wonderful. Lots of colors and fluid movement, it's well done. The opening story is decent, but the title screen is a real showcase of what SMS can do.

Sound and music
Here we find the game a little lacking. One of Zelda's most memorable things was the music, which fit perfectly. Although the music in Golden Axe Warrior is not necessarily negative, it simply becomes a bit annoying and repetitive and sometimes doesn't really fit. The music of the dungeons could have been more disturbing, even if the music of the bosses is beautiful. The sound effects, however, are good, much better than what you'll find in Zelda.

The controls at the beginning are a little wobbly, the character has a tendency to move a little in the wrong direction until you get used to it, but it's not a big deal. Some enemies require greater precision and putroppo the collision is not the best and will affect the gameplay as a whole. However the very simple configuration and interface with different elements are not bad. It may have been a good idea to have two separate slots, one for a weapon and one for an instrument / magic, but since you never really need to use two at the same time, it doesn't change much.

The degree of challenge of the game will keep us busy for several hours and will not bore us.

I have to admit, it's a good game! That's very good. One of the few ARpgs by SMS that I enjoyed playing to the end. Although it is very similar to the Zelda saga, it is very well done technically and presents a good degree of challenge. It does not improve the genre but lets itself be played with pleasure. Too bad Sega hasn't continued to develop this Spin Off series on other consoles as well because the charm of the Golden Axe world is always remarkable. Try it and play with it.

You will find the cartridge at "scandalous" prices and my advice is to try researching or uploading the Roma on an everdrive or emulator. There is no localization in Italian, but it does not affect much.

by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

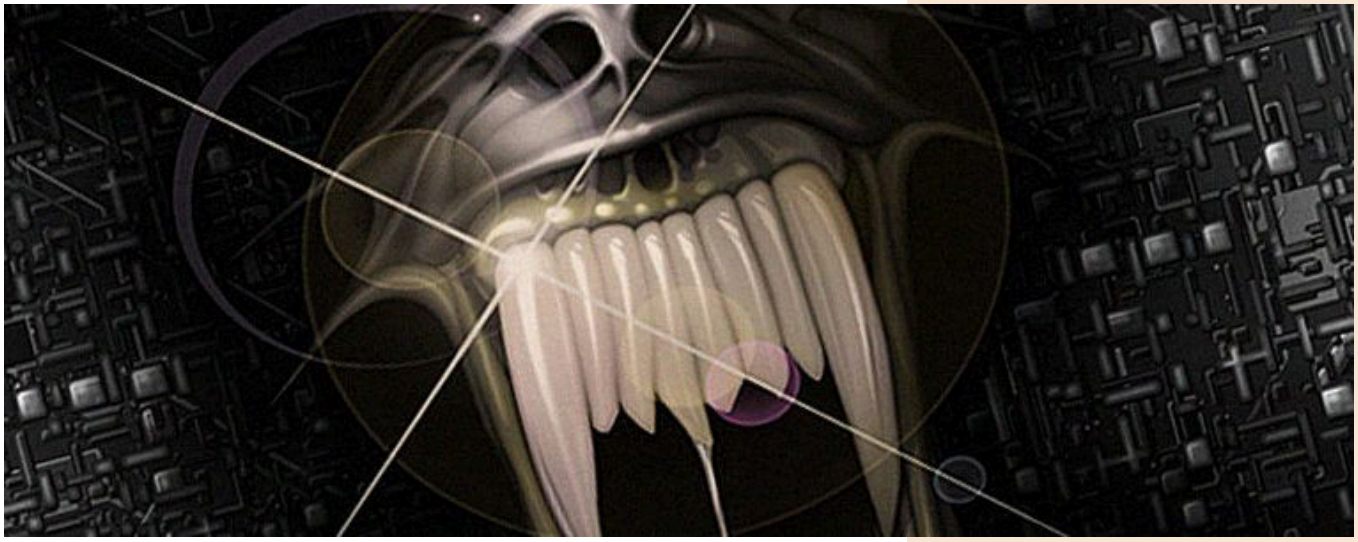
» Gameplay 70%
Controls not always perfect do not help you take immediate confidence with the game. Too bad because the game system and the double slot for weapons and magic are not bad at all.

» Longevity 90%
Many sidequests, many objects to recover and secrets to unveil all accompanied by wonderful graphics will keep you busy for quite a while.

ALIEN BREED

Platform: Commodore Amiga
Genre: Action

Release date: 1991
Reviewed version: Alien Breed Special Edition 92 – Amiga ECS



Being a teen in the 80s not only meant discovering increasingly innovative computers and coin-ops day after day, but also witnessing the birth of real cornerstones and masterpieces of cinema and music.

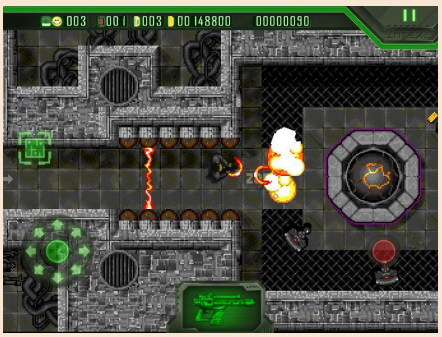
And maybe while we were copying from MC Microcomputer the latest list for our Vic20 or Commodore 64 while listening on the radio to You Shook me All night long AC/DC on TV they were going to transmit one of the last science fiction films released in the previous two/three years.
Obviously if we wanted to travel through time we looked Back to the Future but when we needed a bit of healthy terror and adrenaline, the choices were not lacking at all.

Alien's saga, which recently evolved into Prometheus and Covenant, managed to bring to our screens the purest terror coming from space in the form of imposing, treacherous and lethal creatures designed by Giger's genius, animated by our Rambaldi with a nemesis, the young Sigourney Weaver, determined and sexy at the same time, able to understand before the rest of the crew that those beasts found on a remote Earth star colony should never arrive on earth.

And while technologies in the late 1980s went from 8 to 16 bits the desire to relive that cinematic epic in the form of a video game grew stronger but only with the advent of the never forgotten Amiga computer (made by Commodore) an English programming team managed to give to the community of video players a game able to recreate the tension seen in film.

Team 17 for Amiga was synonymous of quality and (often) also difficulty but in the case of Alien Breed it was clear that it could not be a walk to escape from a Xenomorphi-infested space base without losing a ton of lives.

The game didn't have the original license, but what was there was enough to recreate on our monitors the adrenaline of hunting giant fucking bugs with acid instead of blood.
The missions were quite simple: reach the exit, reach the exit within a total



number of seconds after destroying some reactors but the labyrinthine structure of the levels was not so easy at all.

The corridors were often locked by electronically activated doors, they needed keys to be opened and scattered across the levels there were never enough. It was therefore necessary to access some terminals located in specific locations in order to access the central computer of the base (or the never forgotten Intex System) and using recovered credits you could buy keys, upgrades, new weapons, ammunition and medikits essential to save our skin from continuous and incessant alien attacks.

The Alien Breed series received numerous follow-ups: Alien Breed 2 The horror continues and Alien Breed Tower Assault where the two-dimensional graphic of the game was brought to the uppermost level with the help of the AGA chipset (mounted on the latest Amiga models 1200 and 4000) plus two 3D chapters that tried to push the Amiga architecture beyond its limits (however requiring additional expansion cards to be able to enjoy everything in full screen with a decent framerate).

Team 17 were the author of other memorable titles such as the Worms saga on Amiga,PC and Console, as well as the Project X shooter, the Superfrog platform and the Body Blows.

Recently, Team 17 tried to exhume the brand from the past with new 3D episodes on PC and Console without, however, being as incisive as for the first 2D chapters on Amiga.

Of the original Alien Breed there are also remakes published on Playstation 4 and PS Vita although, personally, I recommend paly the original saga in Amiga emulation on PC with WinUae to really understand what it felt like back then, in the early 90s, when the fucking aliens came out of the walls of our bedrooms!

by **Flavio Soldani**

OUR FINAL SCORE

» Gameplay 92%

Finally a game where surviving hordes of aliens can recreate the tension of the cinematic blockbuster. The first chapter of an unofficial series able to give the player playability and satisfaction with a bit of strategy that did not hurt.

» Longevity 96%

The game was not very long but given the difficulty to get to the end was no a piece of cake. We recommend the Special Edition 92 version that made the game slightly easier. If by longevity we also mean durability I would say that even after 30 years a healthy game of Alien Breed in emulated version is still satisfying today!

FRANTIC FREDDIE

Year: 1983
Developer: Commercial Data Systems

Platform: Commodore 64
Genre: Platform

After a long and sultry August spent entirely in Milan, sweating under a breathing mask and looking for some titles of the past that passed a little unnoticed although very cool to play with, I made my umpteenth discovery thanks mainly to old Italian newsstand cassettes of the 80s. They often included small games that were almost impossible to find in their original version. Still they deserved some attention given the playability and entertainment they gave us during the boring summer and holiday afternoons and also during evenings entirely dedicated to our video game station.

This time it's the turn of a phone techie, whose name is Frantic Freddie! This is a game that presents itself as the classic bi-dimensional platform on several levels with lots of stairs, in this case telephone poles on which to climb and descend to avoid enemies and collect the essential items to finish the level. You also have to grab as many bonuses as you can to increase your score.

That being said, this could only be a simple game capable to entertain you a dozen minutes, but the difficulty is certainly not lacking, so tha challenge becomes more and more interesing as you proceed! The stairs could not have gone up or down both sides, plus they could block the passage to go further, putting us in a dead end waiting only for death.

In short, to complete each level you have to recover all the sparkling pots scattered around the level and avoid the monsters called "Greeblies" that change appearance in each level. When you have successfully reached higher levels of play, you will also see nice detaches, very typical of the platoforms in the 80s. They sort of give that extra touch of entertainment and reward our big efforts.

The music soundtrack is very good thanks to tunes from successful and popular songs like Boogie Fever, Don't Bring Me Down, etc. Maybe I'm being repetitive, but the C64 always offered the most beautiful music in my opinion and I sometimes wonder how they could have gotten all those successful songs and converted them into eight bits!

For those of you, lovers and experts of the platform genre, will you definitely find some originality in this game or perhaps it was better to say limitations?

Well, like any game worthy of respect, once you get used to its gameplay, it will be pure fun! The games almost impossible to finish were counted on the fingers of one hand and I won't list all the titles here since you will surely have noticed and played them so many times.

My hope, on the other hand, is that all of you had a good time on these recent holidays surely a little different from usual. Our attention was no different in finding some obscure and good games. So was our passion for retrogaming that kept us company not only during the lockdown months, but also on the sea beaches and mountain chalets, etc.

Before saying goodbye I recommend you to always tell yourself when take on a new game: "Nothing is impossible!"

by **Daniele Brahimi**

OUR FINAL SCORE

» Gameplay 70%
Right level of difficulty and very fun once you get used to it!

» Longevity 75%
Not so long to finish, but surely a source of good company!

PUNCHY

Year: 1983
Developer: Mr.Micro
Platform: Zx Spectrum, Commodore 64, Commodore 16, Commodore VIC-20, Amstrad CPC, MSX, Tatung Einstein
Genre: Platform

Take control of Bobby the cop in a nightmare landscape.

As the clouds roll threateningly over your head, your goal is to cross fifteen screens to reach the booth of Punch and Judy, where Judy was locked up by the bad guy on duty, Mr. Punch.

Obviously, Mr. Punch has put various obstacles in Bobby's way: there are holes to climb over with ”perfect pixel” precision, flying tomatoes and custard cakes, magic carpets to ride and, of course, Mr. Punch himself, who is more than ready to stab you with his sword to prevent you from moving forward in your game path and completing it.

I have played both the C64 and C16 versions and, although the C64 has more RAM and a better audio/video compartment on its side than its 'smaller brother', I think that on the latter the performance is better, the game is nicer and more playable.

In C64 the choice of colours did not seem right to me and something is lost in terms of gameplay.

In the C16 on the contrary the graphic is pleasant, the colors are not excessive and Bobby's character is well drawn, although sometimes you notice during the game some chromatic defects known as "color-clash", typical also of Spectrum games.
The sound is functional and there is a nice melody played as a reward for reaching the end of each screen.

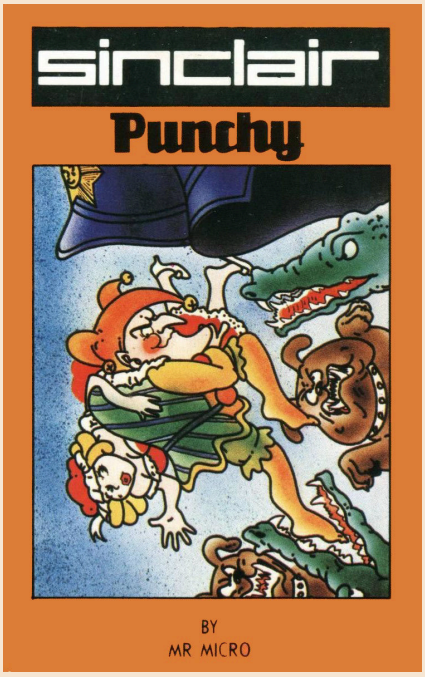
The controls are simple and respond smoothly and the difficulty, continuing on the various game screens, grows in the right way.
Obviously, the more you get into the game, the more difficult it is.

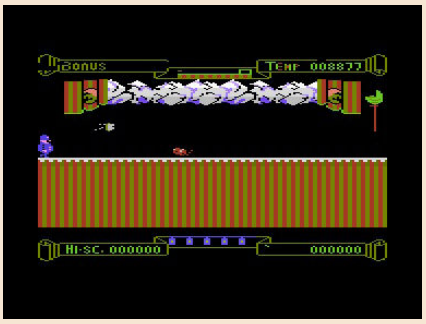
Sometimes play can be frustrating, especially when jumps need to be timed with extreme precision to avoid flying debris, and exactly at the right point on the screen.

There is also a time limit to completing each screen, but the game is still very playable, and with a little practice it can be completed. Although Bobby can never rest - just as he manages to save Judy, Mr. Punch arrives and pushes her away again, and the game returns to the beginning!

With its strengths and weaknesses, I still consider it a good game, and it is also one of the first I have played in my "green years", so I feel particularly attached to this game also for this reason.
Greetings to all and... see you next time!

by **Marco Pistorio**





OUR FINAL SCORE

» Gameplay 90%
Smooth play, with a well-calibrated increasing degree of difficulty. It is the ideal game for those who have a good eye, fast reflexes and the right amount of patience.

» Longevity 70%
It is probably difficult to play it many times, because the game never ends but always starts again from the beginning.

TINY BOBBLE

Year: 2020
Developer: Abyss

Genre: Platform
Platform: Amiga

The group called Abyss in recent years has given us a series of remake games of great classics, all called Tiny.

A remarkable job that has always been very successful among users.

In this hot “abnormal” summer they released Tiny Bobble, a “perfect arcade” conversion (as the developers call it) for Amiga.

The original version released in the 80s, although pleasant and effective, lacked similarity with the dining room machine and was a port of the Atari ST version, precisely why the Abyss collective handed over the original game code and pulled this version out. Before the final version, they put on the market (for free) several beta versions that gave hope.

What have they changed or improved? Here are some improvements from the original game for Amiga:
- It only needs a single 178kb "lowercase" file (the original needed a full disk)
- 50 fps (instead of 25 fps)
- 32 colours (instead of 16)
- 150 power ups (instead of 40)
- Original screen height of 224 pixels (was only 200 pixels)
- Almost all sprite animations (had only about 20% animations)
- Progress screen (not available)
- Big Enemies animation every 16th level (not available)
- Player Two Join animation (not available)
- Big Score images (not available)
- Extended animated screen (not animated)
- Animated Potion Screen (not animated)
- Animated introduction (not animated)
- Animated Boss fight (it was a still image)
- Multiple Finals (had only one ending)

All premises maintained.
Qualitatively it is a beautiful sight and also from the audio point of view we have a beautiful conversion and an audio quality of all respect.

All encoding has been performed with Amiga C/C + + Compile and can be viewed on the developers' site.

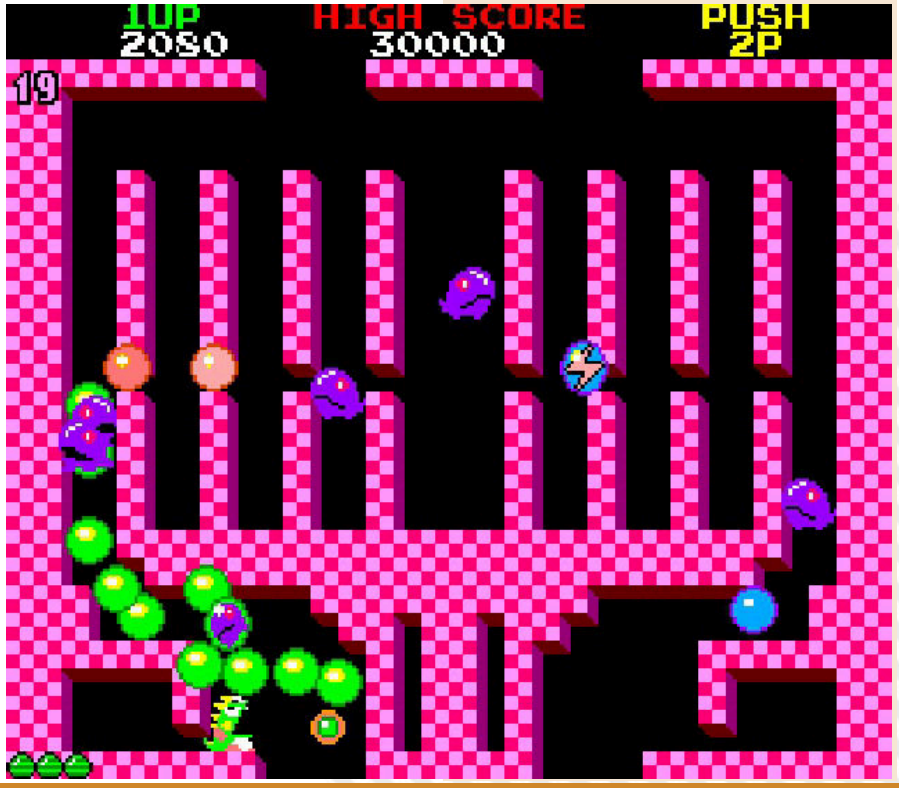
But... And yes, there is a but...

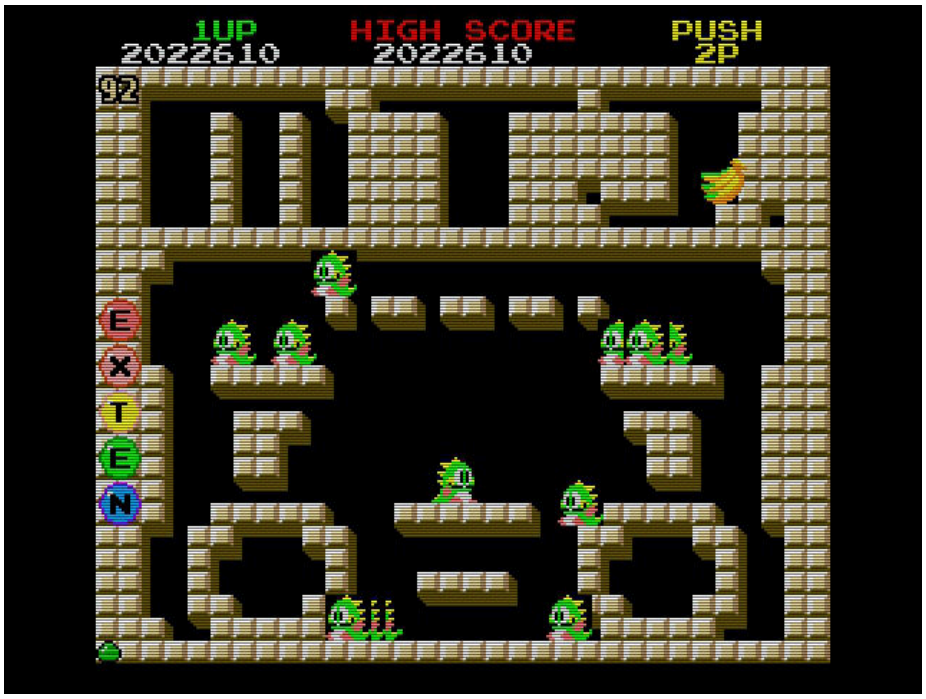
It's not really perfect arcade.

It's very nice, very similar, well developed but it's excessively easy. Simplified.

Some power ups and bonuses are absent or in incorrect positions (and I know, I'm picky) and there are no secret rooms (real resistance challenge of the original game).

Too bad, because the development is really almost perfect and totally





OUR FINAL SCORE

» Gameplay 90%

It is an almost perfect experience.
Playable as the most classic Bubble Bobble but in some places not quite as "perfect" as promised.

» Longevity 90%

It's still Bubble Bobble, but it's much simpler and with many gaps (LACK of the secret rooms!! nbN).

similar to the original.

The decision to release the final version a week after the last beta is also critical. I would have lost a few more weeks.

Caution is not a failure. He's always a beautiful Bubble Bobble.

The game is available in ADF format for all Amiga with at least 1 mb of ram, it also runs perfectly emulated on WinUAE and with the Cores on the MysterFPGA.

by **Carlo N. Del Mar Pirazzini**

BLACK THORNE

Publisher: Interplay
Year: 1995
Platform: Snes
Genre: Platform

My experience as a video player was born in the arcades of the 80s and early 90s. At my house, however, I always preferred computers and so I went from the legendary commodore 64 directly to my first PC 386. I basically completely skipped the console phase, especially the 16-bit ones.

Thanks to computers I discovered the phenomenon of emulation and this allowed me, and still allows me today, to play gems from the past.
Among these titles today I want to tell you about my favorite, which unfortunately did not have the consideration that it fully deserved: I am talking about Black Thorne Super Nintendo version.

The title was developed by Blizzard for Interplay in 1994 (PC and Mac) and 1995 (SNES and saw 32x ???) and this already is a hell of an origin. But beyond its noble origins, Black Thorne sports an original gameplay that is incredible to this very day.

Let's start with some history. In this game we play in the shoes of Kyle, heir to the throne of a kingdom that was conquered by the wicked Sarlac who, thanks to an army of orcs, reduced the Androthi people to slavery. Starting from the usurper's mines, we must climb the various levels to the final clash with Sarlac himself and bring order back to the kingdom.
The game is a platform in perfect Prince of Persia and Another World style, with moves and dynamics that also remind of the beautiful Flashback. Our goal is to get past every screen by moving through stairs, bridges, platforms, jump holes and monsters to kill.
Our hero is armed with a powerful pump rifle that boosts beyond levels and can also collect and inventory grenades, keys, radio-controlled bombs and healing potions.
In order to overcome each screen we must always think carefully about the right move to make and just so often we will have to solve riddles or find hidden passages.

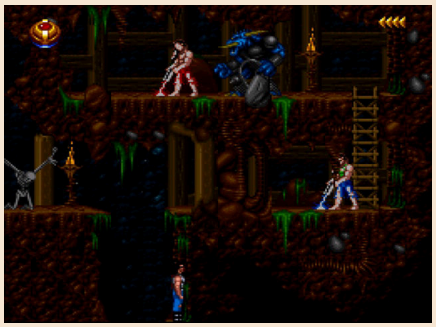
During our journey we will encounter orcs and monsters with whom we must engage in deadly duels. We can also interact with Androthi prisoners who will give us important information or give us objects to use in our adventure.

Once arrived in the throne room, to defeat the wicked Sarlac we must use our best moves, namely jumps, dodging and rolling, otherwise we will never succeed on the monster.

Blach Thorne has beautiful graphics, with detailed and colourful sprites. Excellent is also the sound compartment with music and effects that help create the right atmosphere.
Black Thorne is definitely rough and violent, as evidenced by the continuous bloodshed or the possibility of even shooting and killing prisoners. Unfortunately, this contributed to the criticism of this title and also hampered the design of a possible follow-up, which would have been absolutely welcome.

Nonetheless, I really recommend you to retrieve this masterpiece that you will never find in the rankings of the best games of the Super Nintendo, but that nevertheless offers a truly unique gaming
experience.

by **Querino Ialongo**



OUR FINAL SCORE

» Gameplay 90%
Black Thorne takes the playability of Prince of Persia and Flashback to the highest level. If you like the genre you will still love it nowadays.

» Longevity 90%
Every jump, every movement, every shot must be thought and calculated. This makes the game very long. In the end you will find that you will need several hours of play to finish it.

OBSCURITY CORNER
STURMWIND

Publisher: Duranik
Year: 2013
Platform: Dreamcast
Genre: Shoot'em up

If there's one thing computers and consoles enthusiasts have learned, it's that obsolescence sooner or later comes. It could be in a year, five or ten, but in the end a system is inevitably left on the bench, while other more powerful and versatile machines take its place. But what happens to those that leave the market? Are they destined to disappear into memory, remembered only by a group of nostalgic enthusiasts?

Not necessarily.

There are people who still believe in the past and actively support it; this is why "historic" consoles and computers are enriched by new annual releases, developed with passion and dedication by true romantic heroes.
One of the most interesting examples, in this sense, is Sturmwind, a horizontal scrolling shooter created by the German team Duranik.

Originally born on Atari Jaguar back in 1997, under the temporary title "Native", the game was then moved first to Nuon, one of the darkest consoles ever, finally finding a home on Dreamcast. Released under the label Red Spot Games in 2013, Sturmwind is a demonstration of how skill and stubbornness can overcome the challenge of time.

After a choreographic introductory full motion video, we are introduced to the main menu, through which we can enter the live game. Sixteen truly action-packed levels await us, divided into seven worlds, with a hundred different enemies waiting for us and about twenty giant bosses to face. Considering the average lifespan of such a shooter, we are at absolutely remarkable levels.
Let's start with the aesthetic implant: graphically Sturmwind will make your jaw drop to the ground.

The programmers squeezed the Dreamcast like a lemon, and you can see the result.

The graphics engine features perfectly fused two-dimensional and three-dimensional elements together, interactive backdrops, particle effects, reflections, fluid simulation and tons of enemies simultaneously on screen.

The levels are surprisingly varied, both for settings and themes, and are characterized by impeccable fluidity, which remains constant even when we meet the gigantic bosses, beautifully animated. Watching everything move on screen makes a certain impression, especially when our beloved Dreamcast is attached via a VGA cable to a CRT monitor.

The audio department also defends itself very well, with a soundtrack of excellent quality (as expected from German composers), always fitting and integrated into the action. The

effects are also well realized and loud to the right point.

One of the most interesting aspects of Sturmwind is that it was not designed for an audience of just sneakers (???). Even those who do not have much experience in the genre will be perfectly comfortable, thanks to the possibility of choosing the level of difficulty, as well as being able to save progress on each new chapter reached. In this way the challenge is more accessible and makes the game really for everyone. All types of controllers are supported, from standard pads to arcade sticks, the VMU and even the Rumble Pack (which is also configurable).

In addition to the normal mode there is also the arcade mode, where we have to overcome six levels without continuing, and a whole series of extras such as trophies (similar to those of modern games), which open a section dedicated to preparatory drawings, sketches and models.
There is also the possibility to enter your score on the official game site.

So, what do you say? Sturmwind is a true act of love for Dreamcast, which no sliding shooter (???) should miss. If you are in the mood to spend and want to look for the limited edition, you will also find a CD with the soundtrack and one of the bosses (a giant octopus) in a very cute soft toy version. For those without the Sega home console, you can still recover the latest EX version, which is available on PC, Xbox One and Switch.

Sturmwind is therefore an example to follow and demonstrates once again that our beloved machines can continue to amaze us, despite of the market logic and the principle of obsolescence. The past has passed, but it can always live again: the important thing is not to stop believing in it.

See you next time!

by **Federico "Arzak1" Gori**

OUR FINAL SCORE

» Gameplay 90%

Thanks to a remarkable variety of environments and situations, as well as a scalable level of difficulty, the game is very immediate and fun, making it suitable for both newcomers and fans of the genre.

» Longevity 85%

The amount of levels, modes and options, make the game much more enduring than the average of the genre, giving you a complete and lasting experience.

SENSIBLE WORLD OF SOCCER 2020

Year: 2020
Editor: FREEWARE – SENSIBLESOCCER.DE
Developer: Sensible Software (original version) – Many more developers since then
Genre: Sports simulation
Platform: PC/AMIGA

The most beautiful game in the world? Football? No, Sensitive Soccer!

Fifa 2020 and PES are only successors to what is the absolute king of computer and console football games. Originally released on Amiga and later ON ATARI ST and PC DOS, Sensible Soccer changed the way we played football on our gaming platforms.

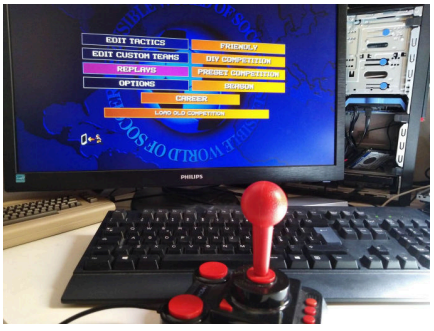
It is perhaps the most converted and imitated game of its kind and, although it has a lot of years on its back, it has a really impressive fan base.
Impressed enough to make him immortal. Over the years it has been continuously updated and finished, played and replayed by all fans.

In this unfortunate 2020 the boys of sensiblesoccer.de come out with this "definitive" version.
Sensible World of Soccer 2020 is a Windows application made by disassembling version Two of the Sensible World of Soccer 96/97 game. Enlarged and expanded, updated with all players in the current market. Immense is the database of players and eligible nations (2400 teams and 26,000 players).

What else they've added. Support for state-of-the-art USB controllers, new rescue system, multiplayer gaming capabilities and even a free team and player db upgrade system.

The game hasn't changed. It's always that beautiful product you came out in '94. Fast, dynamic and all based on the player's "dynamic" ball control.

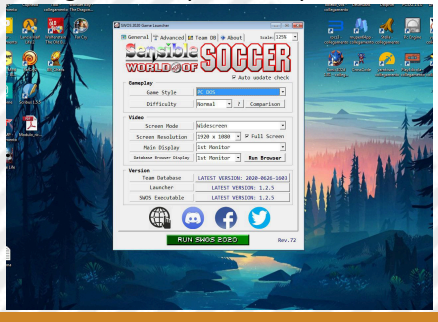
Playable and with eternal longevity, especially in this version that is always updated and updatable.



After all, the version is also available for Amiga (A600 or A1200 with 4 MB of FASTRAM, exotic but it can fit).

Look it up and download it (it's on sale on GOG for just under 2 euros). Football has never been so fun.

by **Carlo N. Del Mar Pirazzini**



OUR FINAL SCORE

» Gameplay 95%

Perfect controls and gradual level of difficulty. Excellent AI of the teams. Impressive updates and the possibility to stay up to date in the future.

» Longevity 100% (Eternal!!!)

I cannot use numbers. Sensible Soccer has gone beyond time and history and has a really incredible fanbase. Always updated, always long-lived... so ETERNAL.

The Basic that lived twice... Or even more!

Porting is the process of transposing, sometimes even with modifications, a software component, aimed at allowing its use in a different execution environment from the original one. Regarding the passage of code from a Basic language to another, however, we usually don't use the term "porting" and we prefer "conversion" or "adaptation", according to the Basic "dialect" used.

How many of us in the past have tried to adapt Basic code, specific to a computer, so that it can be used on another one? Nowadays, since we have all the software at hand, it is an almost lost practice, but in the past it was a real need. Think back when you bought magazines and found listings in Basic for a number of machines but you only owned one of them... how many programs, mostly games, wasted. So why not try to adapt the code, created to run on another machine, to our Basic implementation?
Those who have ever tried this process should have realised its difficulty. Far from being as complex as a real porting, where the adaptation of the machine code was mandatory, still Basic conversions are no joke at all.

Basic, unlike other programming languages, has seen a myriad of dialects proliferate, sometimes configurable as real distinct languages.
Take a look at this list just to get a rough idea:
https://en.wikipedia.org/wiki/List_of_BASIC_dialects
And the list, however extensive, is far from being complete.
Try, for example, to think of all the Basic extensions that over the years have been created to upgrade the Basic V2 of the Commodore 64; in this list you can find only few of them...

All this long introduction just to tell you that I am personally working to transfer some Basic listings created for specific platforms and 'take them' to others.

Why all this effort? What's the point?

First of all because it's fun and educational. Given the substantial differences of the many dialects, it is sometimes necessary to rewrite the logic of the program to make it work like the original. This way I will make sure I well understand the code and above all that I am familiar with the target dialect.
Secondly because I like to give a second life to the code that would otherwise end up forgotten.
And both reasons fit perfectly with the mission of our magazine.

In addition, this exercise lays the foundation for a RMW project that is still in the pipeline, but that we plan to grow as soon as possible.

I also invite you to try this same exercise, especially if you are an early stage programmer. I can tell you that an entire world will open up for you.

Do you want to know more? Keep following us, you won't regret it!

Francesco Fiorentini

RetroMagazine World
Year 1 - Issue 3 - OCTOBER 2020

Chief Editor
Francesco Fiorentini
Managing Director
David La Monaca
Editing Manager
Marco Pistorio
Web Manager
Giorgio Balestrieri