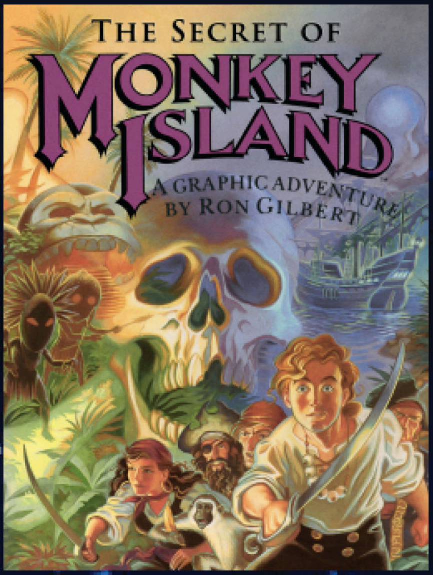
RetroMagazine World

future days are back

Issue 4 Year 1 - November 2020 - https://www.retromagazine.net - Free Publishing

Software compatibility between ZX 81 and Spectrum: monsters and ghosts

PC MS DOS...



Happy Birthday Monkey Island!

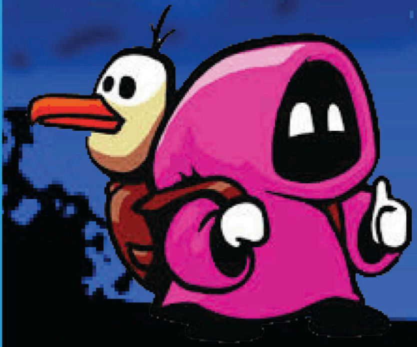
Video chip TMS9918A

CANNON FODDER

Amiga / PC

The Haunted House
...or how to play the same game on 3 different platforms!

A bit of rarity:
A treat without trick



MICRO MAGES (Nes)

Luigi's Mansion (Game Cube)

A "therapy" for Covid-19? We have the solution!

When we were kids, even cartoon monsters could scare us to death. Did this ever happen to you?

And nowadays, now that we are grown men and women, this virus has scared us in springtime and still frightens us. And it paralyses us in many different ways, because it is also a "social" and a "multimedia" virus. A virus that first spread through rumours and then actually became part of our lives. How can we get out of this situation? At the moment, we don't have a vaccine, a safe therapy. This will require the help of time and medical science. Meanwhile, we can at least "fight" the psychological part of the virus with our beloved video games and our glorious vintage computers.

According to the World Health Organisation (WHO), the world of video games has an effective "therapeutic" power for all of its users. Dedicating yourself to video games allows you to get distracted, you end up engaging yourself in activities that involve and bring social interactivity (just think about the multiplayer platforms such as Kaillera). Playing together and sharing experiences prevents contagion. In fact, it prevents a particular contagion, which is underestimated but equally dangerous: loneliness, depression and anxiety. It is not just a question of "playing games", but interacting with this incredible universe, thanks also to the many initiatives that arose during this crazy and unexpected pandemic.

In the world of retrocomputing, we have witnessed with interest the several online challenges on the old classics, the programming competitions using languages such as BASIC (for example, the competition based on the theme of Alien Attack! that we at RMW have personally promoted) and the relentless retro-hardware recovery of so many enthusiasts who repair and regenerate real gems of the past. All this has allowed us to stay away from those terrible psychological monsters, even more frightening than those of horror films or the damage to health that the virus itself can cause to us and our beloved ones. It is these fears that we must fight, those same monsters that have accompanied us over these last hard months.

Well, just a few years ago the WHO stated alarmistically enough that addiction to video games was to be considered as a disease, they named it "gaming disorder". And now something once labelled dangerous is becoming a valuable tool that can help us deal with something much more dangerous. We consider it very positively that the deep qualities of the video games are being rediscovered , now seen as a means of communication designed to unite rather than isolate. Especially in such a difficult time where we are all indiscriminately subjected to fear and uncertainty.

Come on, then! The time has come to defeat this "monster" too. We now know what to do! Let's start by pressing the power button of our console or favorite home computer. You'll see that slowly you can turn into a mustache plumber, a spaceship pilot, a knight in armor, a warrior... We can be all these characters. And we'll also know how to defeat the end-of-level monster. After all, think about it, haven't we done it hundreds of times yet? ;-)

Carlo Nith Del Mar Pirazzini

SUMMARY

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

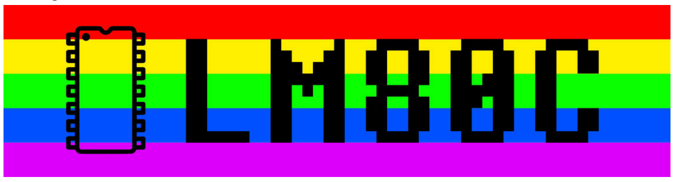
- Alberto (*Ghostbuster*) Apostolo
- Gianluca Girelli
- Michele "*Conte Ugolino*"
- Carlo N. Del Mar Pirazzini
- Daniele Brahimi
- Flavio Soldani
- Francesco Fiorentini
- Attilio Capuozzo/*RPI ITALIA*
- Querino Ialongo
- Leonardo Miliani
- Edoardo Ullo

- **Graphics support** Irene G. Valeri
- **Cover** Flavio Soldani
- **Proof-reading** Francesco Fiorentini, David La Monaca, Alberto Apostolo, Gianluca Girelli, Carlo N. Del Mar Pirazzini, Michele Ugolini

LM80C Color Computer

A 2019 self-built Z80-based home computer - part 3

by Leonardo Miliani



In the previous part, we analyzed how, thanks to the address decoders, the CPU can select certain chips to exchange data with. It is in fact the whole set of integrated circuits that work together that form a computer, it is not just the CPU. The CPU (Central Processing Unit) is the main component but is not the only one. The CPU is basically like an orchestra conductor. The orchestra is composed of many instruments that are played in a synchronized and harmonious way with the others precisely thanks to the conductor who decides the rhythm and the moments when each and every instrument has to play. Bringing the analogy to the level of a computer, therefore, the many instruments are the peripherals that compose it and the CPU our "conductor".

Choosing certain instruments instead of others leads to the composition of a certain type of orchestra, just as integrated circuits create the characteristics of a system as a whole. The choice I had in front of me for the LM80C was the same as in the late 70s and throughout the 80s of the 20th century the manufacturers of computers and consoles: to make a computer completely on their own, also manufacturing the various individual components, or to choose the way of assembly and create a system using parts produced by others. For some manufacturers, the first was a paved road: for example, Commodore owned MOS Technology, and so his computers were based on chips designed and built "in-house". Texas Instruments, with its TI-99 series computers, also followed the same path, being a manufacturer of integrated circuits. Others, on the other hand, relied on what they found on the market: the most famous example are the well-known MSX computers, based on components available on the price lists of various manufacturers. Since I do not yet own my own production plant, I have opted for the assembly of a machine, but resorting to components produced not exclusively for a single system. This has allowed not only to choose chips that are still easily available in circulation but also to draw on resources such as documentation and software that have accelerated and simplified the design of the computer. Furthermore, if I had opted for chips created exclusively for a certain system (for example VIC-II or the SID of Commodore 64) it would not only have linked me to a certain architecture but, given the demand on the second-hand market for these highly sought after chips and no longer in production for years, it would have put me in front of a large economic outlay that I did not want to face in any way.

Figure 1: The Atari 800XL mounted the custom ANTIC, GTIA and POKEY chips to lighten the CPU from video and sound management.

As we have already seen in previous articles, the LM80C is a system based on the Zilog Z80 microprocessor. Alongside this I have added auxiliary chips that manage different computer subsystems, following the philosophy in vogue in the 80s, namely to lighten the work of the CPU by delegating part of its tasks to other subsystems. One of the supporters of this approach was Jay Miner, a US engineer who designed the graphics chips at the base of the Atari 2600 console and the Atari 8-bit computers of the 1980s (figure 1) as well as the creator of the Amiga project. Assigning burdensome tasks such as, for example, managing audio and graphics to specialized chips not only allows you to recover a good slice of computational power from the CPU to be diverted to other tasks but also engages in those specially designed chip tasks. Many inexpensive computers, such as the ZX80, went a different way, optimizing costs but not performance: the CPU, in fact, also dealt with video signal generation, and this task drastically reduced the performance of the system. By introducing the FAST instruction in the ZX81, which deactivated the video while performing particularly heavy tasks, Sinclair tried to put a piece to this problem, allowing the CPU to disengage from image generation and devote itself completely to mathematical calculations. Precisely because the generation of graphic elements of the image is one of the heaviest tasks, some manufacturers did not just design chips capable only of generating the video signal but also worked to integrate additional functions into their products. One of these is the management of particular elements that were very popular at the time because they were widely used in games: sprites.

When I laid the foundations for the LM80C project, I immediately wanted the computer not only to be able to generate a reproducible video image on a common TV set but also to manage colorful graphics and sprite. I therefore needed a chip dedicated exclusively to the video sector. However, it must also have been a chip at the time quite widespread so that bones are now easily available at an economic price. The choice fell on the TMS9918A from Texas Instruments, which meets all the requirements. The TMS9918A was born in 1979 as a graphics chip for the Texas Instruments TI-99/4 computer. The first version of this chip is simply called TMS9918 (without the final "A") and is a good chip at the time. It has 3 video modes: a 40x24 character textual mode, a 256x192 pixel graphics based on "tile" and a 64x48 pixel multicolor mode.

Shortly afterwards the project is reviewed and the chip, which becomes TMS9918A (figure 2), sees the addition of a graphical mode of type "bitmap" (ie with individually manageable pixels). The chip is referred to as "Video Display Processor", VDP, in the official documentation.

The chip is easily interfaced to any CPU. In fact, it requires only 3 electrical signals to be driven: CSW/CSR/MODE, i.e. a "write" signal, a "read" signal and one that sets the mode, i.e. whether the current operation involves VRAM memory or one of the internal registers.

Figure 2: TMS9918A Video Display Processor

The VRAM, the video memory, is separate from the system memory and the only way the CPU can access it is through THE VDP. Usually the connection is made by banally connecting the CSR and CSW pins of THE VDP to the RD and WR pins of the Z80. To make the signals safer, instead of making these connections directly, I used the logical ORs, the 7432 chips. The choice of ORs is not random: in Boolean logic, the result of an OR is true when at least one of the two operands is true. Similarly, a 7432 returns a high signal with even a single high input. Since the signal is active low, I need a low output: for the output to have this level it is therefore necessary that both inputs are at a low level. Since the activation signals are all at a low level, the ORs ensure that CSW or CSR pins are only selected when a correct combination of signals on the inputs is composed. Figure 3 helps us understand this better.

The first OR on the left is used as a "main switch": only if THE VDP is selected from the address decoder (seen the previous time) and the CPU is actually trying to access an I/O device (state defined by the IORQ pin) will we have two low level inputs and, consequently, also the output of the OR at low level. At this point the output of the first OR becomes one of the 2 inputs of both downstream ORs that oversee the activation of the read or write signal. Let's take a practical example. Imagine you want to send data to THE VDP (so we want to write on the graphics chip). The signal on VDP_SEL is low thanks to the upstream action of the device selection decoder. Now, since this is an I/O operation, the signal on IORQ is also low: having 2 inputs at low level, the output is also at low level. This signal enters the 2 OR downstream. Since we intend to send a data to THE VDP, the signal on WR ("write") becomes low and at the same time the signal on RD ("read") becomes high. It is easy at this point to check that only the CSW signal is low since only the output of the OR port marked as U20C (pin 8 in the figure) is low because only here do you have 2 signals at low level on both inputs. CSR is instead at a high level because of the WR signal which is high.

Ultimately, we analyze how THE VDP "MODE" pin enters

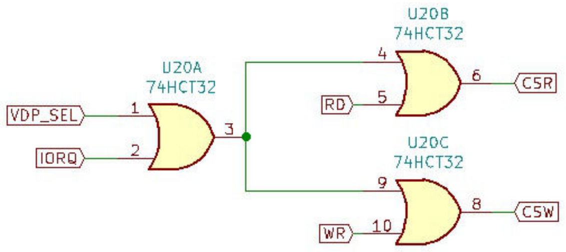


Figure 3: Logical ORs for selecting read and write signals for VDP

the field (figure 4). This pin, depending on the level it assumes, indicates whether the operation refers to the VRAM or if it is directed to one of the internal registers of the video chip. The first case occurs when MODE assumes a low level, while the second when the level is high. At this point it is easy to have this behavior by connecting the pin MODE to a CPU address bus pin, so as to obtain the status change with the simple setting of the I/O port number. In the case of the LM80C the connection is made with pin A1. Since THE VDP is mapped to logical port 48 (0011 0000 in binary), reading or writing to that port sets the pin MODE to low level, thus enabling data exchange with the VRAM. If instead we access port 50 (0011 0010 in track: note the status of the 2nd bit, highlighted in bold), the selection of the VDP does not change but the presence of a high signal on pin A1 simultaneously generates a high level on the pin MODE, which the VDP interprets as an exchange of data with one of its registers.
Surely there would be much more to say about the VDP but the purpose of this article was to examine how the CPU manages to exchange data with the graphics chip, not as much as this works.

Well, again this time we have arrived at the greetings, I

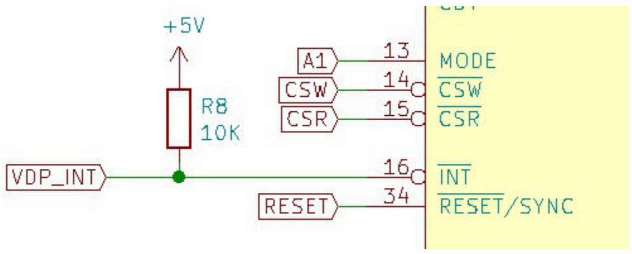


Figure 4: VDP pin "MODE"

will wait for you in the next issue.

USEFUL LINKS

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

The TMS9918A video chip

by Leonardo Miliani

Starting from this issue, we introduce a series of articles about the components that have formed the backbone of our beloved retro-systems, be they computers or consoles.

What made a certain system distinctive was its hardware architecture: unlike today, where computers share the same architecture (basically a direct evolution of that PC marketed by IBM in 1981), in the 70s and 80s of the last century each manufacturer marketed a computer by elaborating custom projects according to its own choices. Starting from the selection of the CPU up to the different custom chips, each component went to assemble a system that, good or bad, stood out from the others and was, so to speak, unique. Despite this, many computers resembled each other because, as mentioned in the previous series of articles dedicated to my LM80C self-built computer, not all manufacturers were also manufacturers of ICs, so those who were not, they were forced to select what the market offered then. And after the CPU, one of the definitely most important components was the video processor, which was entrusted with the task of generating the video image to be sent to the monitor or the TV connected to the system. The video processor often had the task of managing not only the text modes but also the low- and high resolution graphic modes. The chip we are going to analyze in this first article of the series, is the TMS9918A from Texas Instruments, used in lots of computers and consoles of the time.

TMS9918 Video Display Processor

In the second half of the 70s, the 8-bit computers on sale were no longer intended purely for amateurs because they had to be assembled by the user (such as the Apple I, the Altair 8800 and the like). They finally were complete user-friendly systems, with their own case equipped with a keyboard and ready to use: in fact, the user only had to connect the power supply and a monitor or, more widely, a common TV at home. So the Commodore PET, the Apple II and the TRS-80 invaded the homes of millions of users. Texas Instruments (TI), noticing the remarkable success obtained by those home computers, decided to enter the market and began, in 1977, the development of its own home computer called the TI-99/4: this was similar to a console, it had the ability to run cartridge games. Later it was also added the connections for a keyboard and an expansion port in order to connect additional peripherals. The TI-99/4 needed a video processor and so TI developed the **TMS9918** (note the absence of the ending "A" in its name) and they called it **Video Display Processor** (VDP), a chip with good technical specifications: the TMS9918 generated a 60 Hz video signal in standard NTSC with a graphical resolution of 256 rows, 192 columns and 15 colors. It could also handle a 40x24 character text mode and a 64x48 pixel multicolor mode. The chip designers decided to ship it with a VRAM (video memory) separated from the 4 KB system RAM: due to this choice, the CPU could only write to the video buffer through the VDP. The advantage proposed by this architecture, however, was the fact that the VDP did not "steal" memory from the system in order to manage the video buffer.

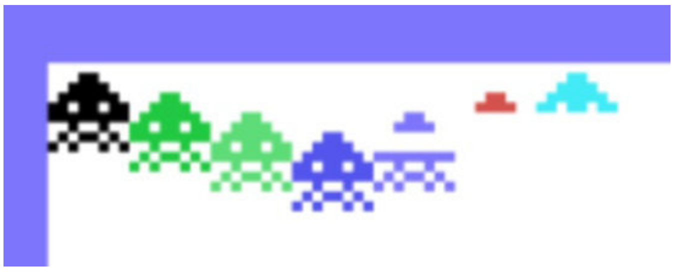


Figure 1 - TMS9918 displays only 4 sprites per scan line

Sprite

The TMS9918 supports hardware sprites, special graphical objects often used in games to represent elements such as the player's character, enemies or other moving animations. Interestingly, the term "sprite" was coined by an IT manager, Dave Ackley, and officially appeared for the first time in the VDP documentation. Sprites are graphical objects managed by the video chip that move around the screen independently of the background, without altering it. Initially it had been decided to support only 4 monochrome sprites for technical reasons (limited VDP access time to the VRAM and little bandwidth of the latter to pass more information) but Matthew Hagerty and Pete Macourek, a couple of the VDP designers, managed to expand the number to 32, adopting a hack based on a "sprite stack" and a "pre-processing sprite": during the course of the video raster to the edges on the side of the screen (an area of the screen with no information to display) the chip retrieves the sprite information (pre-fetching) and saves it into a stack. When the image is generated, the VDP checks the stack for the 4 sprites that can be displayed for each individual row and retrieves their data. However, the limit of the 4 sprites per row remains: in Figure 1 you can see how the parts of the 5th sprite onwards are not correctly rendered on screen because they occupy the same scanning line as the first 4.
The sprites have dimensions of 8x8 or 16x16 pixels (they are actually composed of 4 side by side sprites), which can be doubled: in this case the pixels assume a dimension of 2x2 pixels, so the dimensions of the sprites become, respectively, 16x16 and 32x32 pixels.

TMS9918A

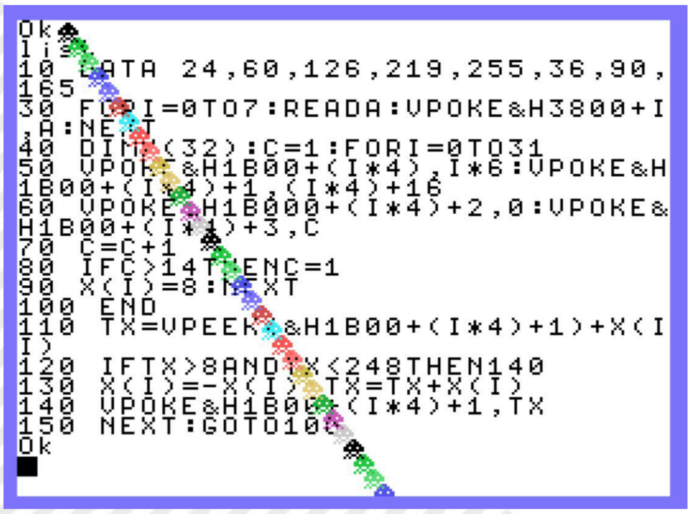
The TI-99/4 was not a very successful computer. So TI improved the design and released the TI-99/4A in 1981, a version revised in many respects and also equipped with a new graphic chip called TMS9918A. The "A" meant that, if compared to its predecessor, it has a new operating mode called “Graphic 2”, which can handle the image in bitmap mode, i.e. every single pixel is individually addressable. The TMS9918, in fact, can only handle graphics in tile mode, where a tileset composed of 256 tiles of 8x8 pixels each, is used to manage graphics. The limitation of this mode is that the elements, repeating on the screen, do not allow to draw complex graphics or images. In order to support this new bitmap mode, the TMS9918A must be equipped with 16 KB of VRAM, all necessary to keep in memory not only all the information related to the video buffer but also all the data related to the sprites.

Versions and usage

The first version of the chip, the TMS9918 (without the "A”) was only used for the TI-99/4. The **TMS9918A** (with the "A”) was initially used for the TI-99/4A but then licensed to third parties: thanks to this commercial move, TI sold many processors and they were widely used in many systems of the time. It has been mounted on ColecoVision, CreatiVision and Sega SG-1000 consoles, Spectravideo computers and, most importantly, the first series of MSX home computers. Some more modern consoles, such as the Sega Master System, Sega Game Gear and Mega Drive mounted graphics chips derived from the TMS9918A. To adapt the VDP to different systems and different video signals, specific versions have been released: the TMS9928A always generates an NTSC signal in YPbPr version (with colour components) but for countries with PAL standards the TMS9929A has been created. Subsequently the project was revised and the new TMS9118, TMS9128 and TMS9129 were released: these chips are identical to the models they replace, apart from the function of a pin and a different management of the VRAM.

N.B.: henceforth “TMS9918A” for convenience will mean globally both TMS9918A and all its derivatives, since they share the technical characteristics.

Technical specifications



```
Ok
lis
10  DATA 24,60,126,219,255,36,90,
165
30  FORI=0TO7:READA:VPOKE&H3800+I
,A:NEXT
40  DIMX(32):C=1:FORI=0TO31
50  VPOKE&H1B00+(I*4),I*6:VPOKE&H
1B00+(I*4)+1,(I*4)+16
60  VPOKE&H1B000+(I*4)+2,0:VPOKE&
H1B00+(I*4)+3,C
70  C=C+1
80  IFC>14THENC=1
90  X(I)=8:NEXT
100  END
110  TX=VPEEK(&H1B00+(I*4)+1)+X(I
I)
120  IFTX>8ANDTX<248THEN140
130  X(I)=-X(I):TX=TX+X(I)
140  VPOKE&H1B00+(I*4)+1,TX
150  NEXT:GOTO100
Ok
```

Figure 2: sprites rendered above the text, which occupies one of the top planes of the image

The TMS9918A communicates with the CPU through an 8-bit data bus and only 3 other signals connected to as many chip pins: read, write and mode. The first two have clear meanings while the last one is used to indicate to the VDP whether the data is to be exchanged with the VRAM or with one of its internal registers. These registers are 8 in total and write-only. They are needed to program the functions of the graphics processor, such as the graphics mode or in which portions of the VRAM to store video buffer and sprite data. One register is read-only, the status log, which provides information about, for example, the collision between sprites or the interrupt. The video image is divided into "display planes”. These “levels” are in number of 35, from 0 to 34: the lower the number, the closer the plane is to the observer and has higher priority than the planes below. The first 32 planes are occupied by sprites, the 33rd floor contains the background image, the 34th is the "backplane”, which is as wide as the entire screen image and forms the background of the entire image and is generally seen as an edge on the central screen sides. Behind it there is a particular plane called "external VDP” that would allow the video signal generated from an external source to be introduced into the image: the VDP in fact features a video input (only the TMS9918A model actually does) that, in the plans of the IT designers, would have served to overprint the objects and text on the image generated by another VDP, even though, to my memory, I am not aware of systems that have used this functionality. The last plane, called “default backplane”, is a mere black image that appears for example when the VDP is powered but is not programmed by the CPU. Figure 2 shows an example of sprites overlapping each other and the text, which occupies one of the last planes of the image. There are 15 colors available, although they are sometimes referred to as 16: actually the 16th color is “transparent” and it's needed to show the image of the plane dedicated to the external signal. In the event that there is no pixel on in any of the planes below and, in the absence of an input signal, then the pixel of the last plane (“default backplane”) will pass through: this is why it is often thought that this color is also black, because it passes the color of the last plane which, in fact, is black.
Regarding VRAM management, the VDP divides the video memory into various “tables” where, depending on the selected graphics mode, the data necessary for the generation of the video image and sprites are stored. The "pattern table” contains the character patterns, the "name table” contains the pattern codes to load for each cell of the screen, the "color table” the color information. The sprite data are also stored in 2 separate tables: the "sprite attribute table”, which contains the main sprite data (such as X and Y coordinates and color) and the "sprite pattern table”, which instead contains the pixel data that must be turned on or off to compose the objects.
A major chip deficiency is the absence of hardware scrolling, both vertical and horizontal. When it was designed, at the end of the 70s, the games were single-screen based and this feature was not required. Then at the beginning of

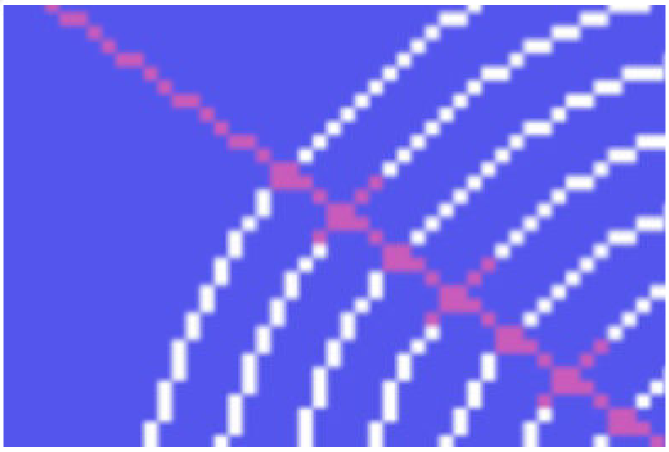


Figure 3: the defect known as "color spill"

the 80s arcade conversions became popular and they often featured scrolling as an important element of the games: all the conversions were greatly affected by this limitation, because replacing the hardware scrolling with software routines in most cases produced not enough good results.

Graphic modes

The TMS9918A, as mentioned, supports 4 graphics modes: text mode, tileset graphics, bitmap graphics and multicolor.

Text mode:

in this mode the screen is divided into 40x24 cells of 6x8 pixels, for an image resolution of 240x192 pixels. To manage this mode, the VDP uses a “pattern table” of 2,048 bytes (256 patterns per 8 bytes) and the “name table”, large 40x24=960 bytes, where each byte represents a memory cell: each of these cells contains the character code (from 0 to 255) that the VDP will then pick up from the pattern table for display. There are only 2 colors available and they apply to the entire screen: the main color, used for the bright spots of the individual patterns, and the background color, used for the off spots and for the edges of the screen. Because the video cell is only 6 pixels wide, the 2 most significant bits of the patterns are ignored.

Graphics Mode 1:

this mode is a graphical mode that can also be used to represent text. The image is divided into 32x24 cells of 8x8 pixels, for a resolution of 256x192 pixels. The pattern table is large 2,048 bytes (256x8), the name table is 768 bytes because the allowed cells are, as mentioned, 32x24. Unlike text mode, colors are handled differently in this mode, and the color table is 32 bytes long: each byte contains the color of pixels on and off for a block of 8 characters. Border color is independent. Since the available patterns are only 256 while the cells are 768, this mode does not allow you to address the individual pixels of the image: on the other hand, this mode is also used as a text mode, because you can define patterns with ASCII codes and then display them on the screen with a simple byte. Sprites are supported.

Graphics Mode 2:

Graphics mode 2 is a pure bitmap mode, where the graphics chip can independently handle each individual pixel and handle an image of 256x192 pixels. Unlike graphics mode 1, in this mode the image is managed with 3 tables of 256 cells of 8x8 pixels each, for a total of 768 different patterns. The color is also handled differently: each byte (8 pixels) of the image allows you to use a different primary color and one for the background. In this mode the pattern table occupies 6,144 bytes: since each pattern occupies 8x8 pixels, it takes 8 bytes, and since each table contains 256 patterns, it takes 8x256=2,048 bytes for each table. The same size (6,144 bytes) occupies the color table since, as mentioned before, each byte of the table indicates the colors for an 8-pixel video segment. Finally, the name table is 768 bytes long (32x24 cells): the screen is divided into 3 horizontal areas (top, middle and bottom), and each area is composed of 256 cells that can contain 256 different patterns. Graphics mode 2 also supports sprites. Since the colors of the pixels of each byte of the image (8 horizontal points) are stored in a single byte, where the color of the pixels on and that of the pixels off are saved, it is understood that for each cell of 8 pixels you can only have 2 colors at the same time: if the user tries to draw something with a third color, the latter alters the primary color of all the pixels of that byte, as can be seen from the image in figure 3. This defect is known as "color spill”.

Multi-color graphics mode:

This mode allows you to have a medium resolution image of 64x48 blocks, where each block actually occupies 4x4 pixels on the screen and can take any of the 15 colors available. The name table and pattern table are used for this mode: the color table is not used because the color information is contained in the pattern table. The name table consists of 768 bytes pointing to an 8-byte segment in the pattern table (1,536 bytes). Of this segment each area of 4x4 blocks indicates the colors for different lines of the screen and only 2 bytes are used: the first indicates the colors of the 2 upper blocks and the second that of the lower blocks. This mode also supports sprites.

Undocumented modes:

“playing” with the settings of registers you can get some undocumented VDP modes. For example, you can get an intermediate mode between graph 1 and graph 2, where you only have 256 patterns as in graph 1 but with the ability to manage the colors of the individual bytes of the patterns as in graph 2. Or set a different way for each of the 3 pattern areas on the screen, for example having the first 2 areas managed in bitmap mode and the last one in tile mode, thus allowing you to draw detailed graphics for the game frame on the first two thirds of the screen and texts or patterns on the last one, for scores, messages and more.

Well, that's it for this article. See you next time.

Software compatibility between ZX-81 and the Spectrum: monsters and ghosts

by Alberto (Ghostbuster) Apostolo

As a boy, I used to analyze ingenious little programs written for ZX 81 (almost always games) but I often encountered some difficulties in converting instructions. There is no 100% compatibility between ZX 81 and ZX Spectrum.

Hardware Differences

The circuits are obviously different. 16 KB memory expansion for ZX 81 is not usable on ZX Spectrum. Programs saved on cassettes for ZX 81 cannot be read on ZX Spectrum. Only the ZX Printer can connect to both models [Bon83].

The organization of memory is different and consequently POKE and PEEK commands will also be different (as well as the machine language routines and system variables).

The memory area reserved on ZX 81 for the screen consists of 768 bytes (24 rows by 32 columns of characters) and low-resolution graphics (64 x 44 pixels).

The ZX Spectrum model has 6912 bytes divided into:

1) 6144 bytes of graphics (256 x 192 pixels or 24 x 32 characters),

2) 768 bytes (24 rows by 32 columns) reserved for font colors and other video attributes (brightness and flashing).

The character set for ZX 81 has its own encoding while the characters from 0 to 127 in the set for ZX Spectrum are ASCII. For example, in the ZX 81 model the "A" character and the "A" character in reverse (Fig. 2) have code 38 and 166 respectively [Bon82] while in the ZX Spectrum model they have the same code 65 (only the video attributes change). Some graphical characters not in the ZX Spectrum set can be implemented through User Defined Graphics (UDGs).

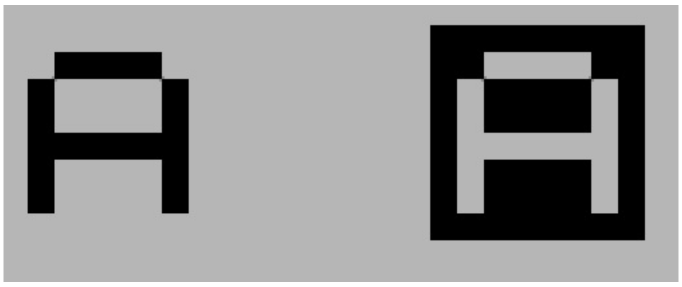
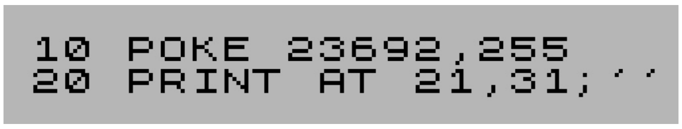
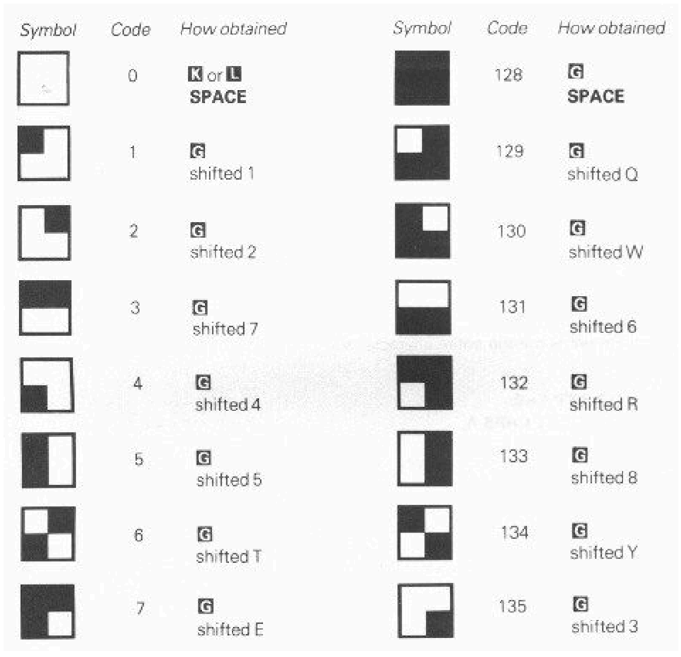


Figure 1 (above) - Figure 2 (below)



```
10 POKE 23692,255
20 PRINT AT 21,31;''
```

Figure 3: apex simbol is obtained with Symbol_Shift+7.



Symbol	Code	How obtained	Symbol	Code	How obtained
	0	K or L SPACE	█	128	G SPACE
▘	1	G shifted 1	▟	129	G shifted Q
▝	2	G shifted 2	▙	130	G shifted W
▀	3	G shifted 7	▄	131	G shifted 6
▖	4	G shifted 4	▜	132	G shifted R
▌	5	G shifted 5	▐	133	G shifted 8
▞	6	G shifted T	▚	134	G shifted Y
▛	7	G shifted E	▗	135	G shifted 3

Figure 4

BASIC distinct...

Hardware differences affect their respective versions of BASIC wired in ROM.

BASIC for ZX Spectrum can be considered an extension and revision of that for ZX 81.

FAST and SLOW commands are missing on ZX Spectrum because it operates at FAST speed and transmits video data in SLOW without interfering with each other's operations and it makes no sense to emulate them (video management is independent).

PAUSE 0 command emulates PAUSE 40000 for ZX 81 [Lis84b].

SCROLL command is missing because on ZX Spectrum "scrolling" is automatic but with user control with the request "scroll ?" which appears at the bottom of the screen. POKE command 23692,255 (system variable "SCR CT") followed by PRINT AT 21,31;'' emulates SCROLL command for ZX 81 (Fig.3).

UNPLOT command for ZX 81 has been replaced by PLOT OVER 1 command, but care must be taken with the graphical resolution and programs for ZX 81 in

which PLOT and UNPLOT modify the graphical characters (Fig.4) in place of PRINT AT commands.
SAVE command for ZX 81 saves programs and data to the same file while on ZX Spectrum there are multiple options for saving arrays and byte sequences (screens, machine language routines). Another difference is saving BASIC programs with autostart. On ZX Spectrum you have SAVE s LINE k command where s is a string and k is the line number to skip to after loading with LOAD command. On ZX 81, however, it is a bit more complicated (Fig. 5). The example is taken from the original manual for ZX81 (chapter 16). After starting the recorder in REC mode, RUN 100 command will save the program (as a benign side effect, the last letter of the name will appear in reverse). After repositioning the cassette belt, LOAD command "USELESS" must be given to load the program. After loading, processing will continue to line 110.

As an alternative for a RUN command, you can give a GOTO command if you do not want to delete the variables (while GOSUB is not recommended because it does not work properly).
Manage collisions in games written in ZX BASIC
Generally, in a game that uses alphanumeric characters as graphics, collisions are handled with BASIC statements and not with strange machine language routines.

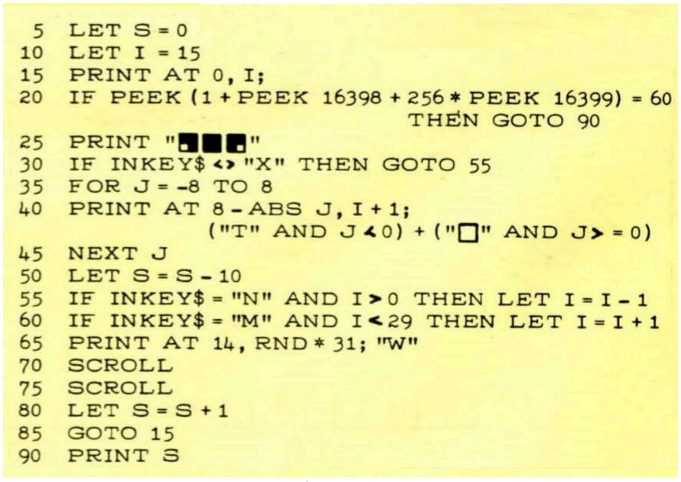
One difficulty that can be encountered in converting a program written for ZX 81 is due to the technique of using POKE and PEEK to place characters in the memory area reserved for the screen.

Fig. 6 shows a small game for ZX 81, which appeared in the Italian magazine LIST no. 1 [Lis84]. The spaceship at the top of the screen must dodge the enemies (symbolized by the letter "W"). The commands are "N" to go left, "M" to go right and "X" to shoot (each shot subtracts 10 points from the variable S that stores the score). Fig.7 shows the version for ZX Spectrum (with changes to lines 20, 70, 75 and THE PAUSE instruction added to improve gameplay). To manage the collision between the center of the spacecraft and the enemies, a SCREEN$(row,col) function was used on line 20 to return the printed character to the coordinates (row,col) of the screen.

However, a SCREEN$ function has a weakness. For example, if a high-resolution plotted pixel "soils" the character printed at the position (row,col) then SCREEN$ will return an empty string because it cannot recognize it.

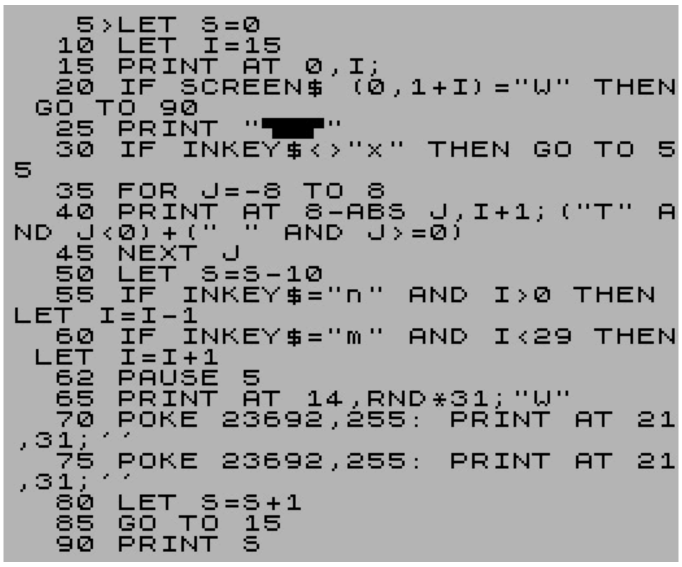
```
  5 REM "USELESS"
 10 PRINT "THIS IS ALL IT DOES"
 20 STOP
100 SAVE "USELESS"
110 GOTO 10
```

Figure 5



```
 5  LET S = 0
10  LET I = 15
15  PRINT AT 0, I;
20  IF PEEK (1 + PEEK 16398 + 256 * PEEK 16399) = 60
                                THEN GOTO 90
25  PRINT "███"
30  IF INKEY$ <> "X" THEN GOTO 55
35  FOR J = -8 TO 8
40  PRINT AT 8 - ABS J, I + 1;
          ("T" AND J < 0) + ("□" AND J >= 0)
45  NEXT J
50  LET S = S - 10
55  IF INKEY$ = "N" AND I > 0 THEN LET I = I - 1
60  IF INKEY$ = "M" AND I < 29 THEN LET I = I + 1
65  PRINT AT 14, RND * 31; "W"
70  SCROLL
75  SCROLL
80  LET S = S + 1
85  GOTO 15
90  PRINT S
```

Figure 6



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

Figure 7

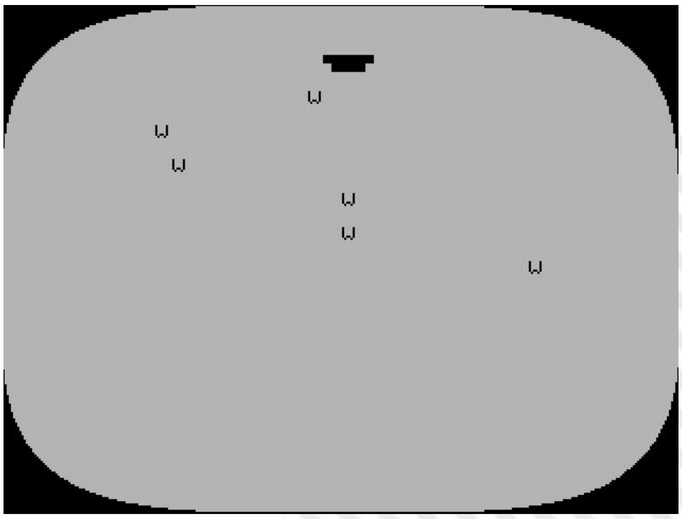


Figure 8: screenshot of the game in Fig.7

Curious facts
In [Bon83] at page 230, there's a mistake. Addresses written in row 20 are for ZX81 not for Spectrum!

It's a shame SCREEN$ only works with ASCII characters and can't recognize UDGs and other graphics characters. The legendary Dilwyn Jones suggested to circumvent the obstacle by coloring the graphics and using an ATTR (row,col) function that returns a number associated with the position (row,col) equal to Flashing*128 + Brightness*64 + Paper*8 + Ink. Additionally, ATTR function processing is 25% faster than SCREEN$ function. Fig.9 shows a small program written by Dilwyn Jones for ZX Spectrum in which UDGs are used [Jon83]. The keys to command the spaceship are: "5" to move left and "8" to move right.

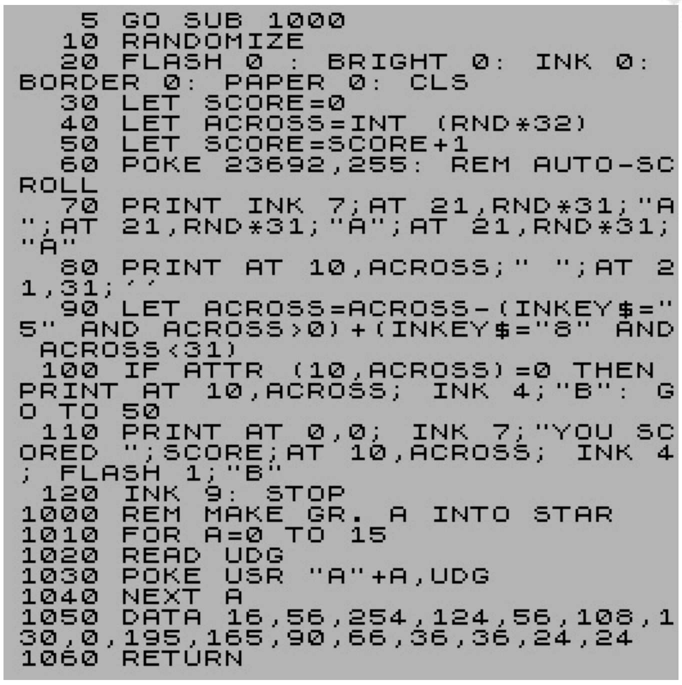
Conclusions

Why translate a program from ZX 81 to ZX Spectrum today?

In the world of retrocomputing there are programming competitions to create software with the least number of program lines. The small programs created for ZX 81 are a good gym to increase your skill.

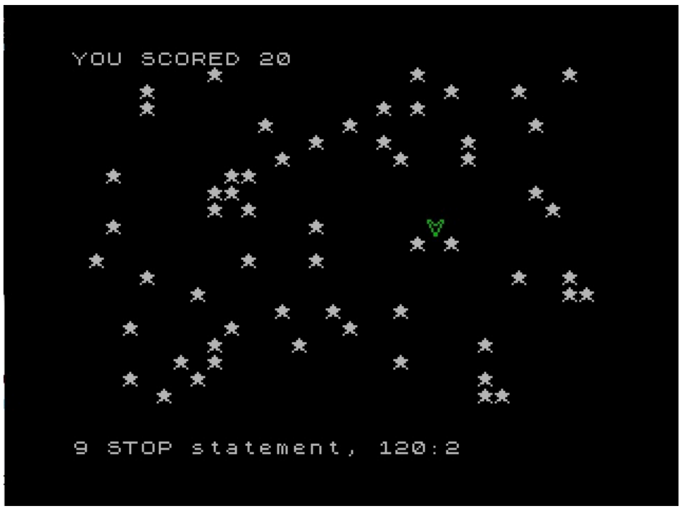
Or you want to run a rewritten program on ZX Spectrum (real or emulated) to take advantage of the superior features (color, graphics, sound).

There are no fixed conversion "rules". In order not to hunt monsters and ghosts during the debugging phase, a preliminary "reverse engineering" is convenient to understand the behavior of the program and subsequently improve only the parts related to the Input-Output, leaving unchanged those consisting of calculations.



```
   5 GO SUB 1000
  10 RANDOMIZE
  20 FLASH 0 : BRIGHT 0: INK 0:
BORDER 0: PAPER 0: CLS
  30 LET SCORE=0
  40 LET ACROSS=INT (RND*32)
  50 LET SCORE=SCORE+1
  60 POKE 23692,255: REM AUTO-SC
ROLL
  70 PRINT INK 7;AT 21,RND*31;"A
";AT 21,RND*31;"A";AT 21,RND*31;
"A"
  80 PRINT AT 10,ACROSS;" ";AT 2
1,31;''
  90 LET ACROSS=ACROSS-(INKEY$="
5" AND ACROSS>0)+(INKEY$="8" AND
 ACROSS<31)
 100 IF ATTR (10,ACROSS)=0 THEN
PRINT AT 10,ACROSS; INK 4;"B": G
O TO 50
 110 PRINT AT 0,0; INK 7;"YOU SC
ORED ";SCORE;AT 10,ACROSS; INK 4
; FLASH 1;"B"
 120 INK 9: STOP
1000 REM MAKE GR. A INTO STAR
1010 FOR A=0 TO 15
1020 READ UDG
1030 POKE USR "A"+A,UDG
1040 NEXT A
1050 DATA 16,56,254,124,56,108,1
30,0,195,165,90,66,36,36,24,24
1060 RETURN
```

Figure 9



```
YOU SCORED 20

9 STOP statement, 120:2
```

Figure 10: screenshot of the game in Fig.9

Bibliography

[Bon82] R.Bonelli, "Guida al Sinclair ZX81", Gruppo Editoriale Jackson, 1982,
https://archive.org/details/guidaalsinclairzx81zx80enuovarom

[Bon83] R.Bonelli, "Alla scoperta dello ZX Spectrum", Gruppo Editoriale Jackson, 1983,
https://archive.org/details/allascopertadellozxspectrum

[Hew84] A.Hewson, "ZX equivalents", Sinclair User Magazine, n. 25, Apr.1984, pagg.122-124,
https://archive.org/details/sinclair-user-magazine-025

[Jon83] D.Jones, "Delving deeper into your ZX Spectrum", Interface Publications, 1983.

[Lis84] LIST n.1, Jan-Feb 1984, pag.28,
https://archive.org/details/LIST1984-01

[Lis84b] LIST n.5, Sep-Oct 1984, pag.92,
https://archive.org/details/LIST1984-05

LIST issues (not complete):
https://archive.org/details/listprogrammi

Alien Attack!

from Amstrad CPC to Commodore 64 through Simons' Basic

by Francesco Fiorentini

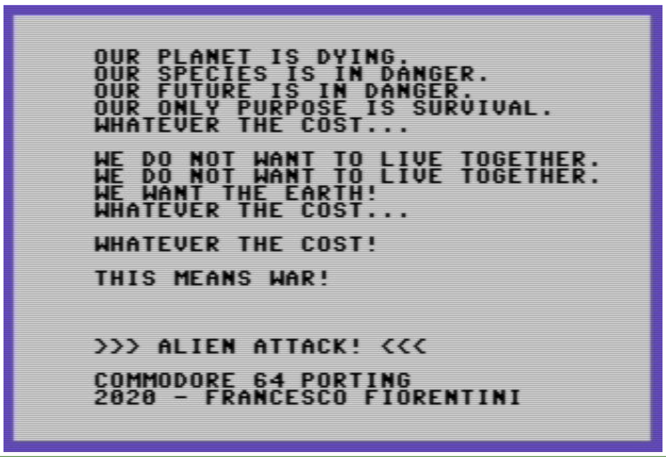
After the article in the last issue dedicated to the game Alien Attack! written in Locomotive Basic for Amstrad CPC, I realized that it's been a while since I wrote anything for my first computer, the Commodore 64. It was a gap that had to be filled and full with goodwill I decided to convert that game for the Breadbin. At first glance it might seem a rather simple operation, in the end they are two 8bit machines, almost contemporary, very similar in text screen resolution and both equipped with Basic. But, as the wises teach, the devil hides in detail.

As I have already written several times, the Locomotive Basic that equips the Amstrad CPC is a truly advanced Basic dialect, nothing to do with the limited capabilities of the Basic V2 that equips the Commodore 64. The commands that the two Basic dialects have in common are few and, obviously, I had used in my game most of those specific to the Locomotive. After a quick analysis I was still reasonably convinced that I would be able to transfer all the code fairly easily, with the sole exception of characters redefinition.

Redefining the characters on Commodore 64 is a rather long and tedious operation, while it is definitely simple on the Amstrad CPC thanks to the SYMBOL statement. Obviously I would have avoided redefining the entire character set, but to generate the tiles of the game and the spaceship, as well as the bomb, I would still have to redesign some characters. When I was resigned to making this effort, in order to effectively convert the game, I came up with an idea. An idea called **Simons' Basic**.

Simons' Basic

The limitation of the Basic V2's commands and the consequent relative ease they can be added, has caused, over the years, the proliferation of several Basic extensions for the Commodore 64. Undoubtedly one of the most famous is Simons' Basic.
Created by a talented 16-year-old boy, this software was released in 1983 and has been incredibly successful ever since. Anyone who has ever owned a C64 has heard of Simons' Basic at least once.
The program, generally distributed on cartridges, is now easily available even in D64 format and adds 114 powerful commands to the standard Basic!



```
OUR PLANET IS DYING.
OUR SPECIES IS IN DANGER.
OUR FUTURE IS IN DANGER.
OUR ONLY PURPOSE IS SURVIVAL.
WHATEVER THE COST...

WE DO NOT WANT TO LIVE TOGETHER.
WE DO NOT WANT TO LIVE TOGETHER.
WE WANT THE EARTH!
WHATEVER THE COST...

WHATEVER THE COST!

THIS MEANS WAR!


>>> ALIEN ATTACK! <<<

COMMODORE 64 PORTING
2020 - FRANCESCO FIORENTINI
```

The purpose of this article is not to illustrate all the capabilities of Simons' Basic, a book would not suffice, but to indicate some peculiarities that have been useful to me for the porting of the game.

Characters redefinition

As mentioned above, the redefinition of the characters on the Breadbin is quite a long operation; fortunately Simons' Basic greatly facilitates this task: not at the levels of the Locomotive Basic, but quite close.

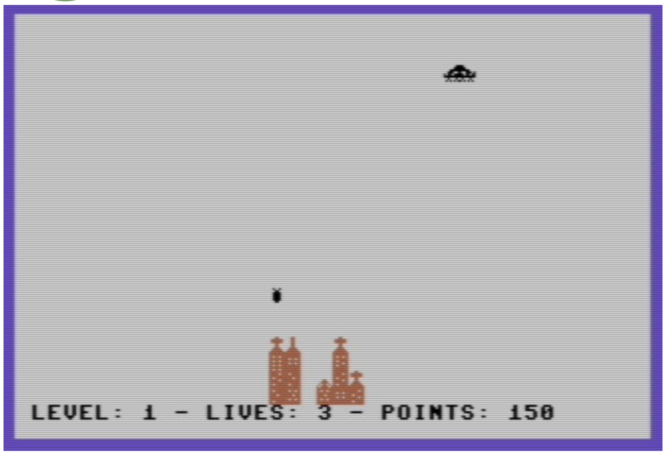
Before the characters can be redefined, however, they must be copied from ROM to RAM. This operation, which is quite tricky in Basic V2, translates into Simons' Basic in just the **MEM** statement.

As reported in Simons' Basic' manual: *the MEM command transfers a copy of the character set from ROM to RAM, under the control of the Kernal operating system*. **The screen is moved to location $CC00**.
Keep the highlighted part in mind, you'll see the reason later.

After copying characters from ROM to RAM, they can be redefined using the **DESIGN** statement and the connected @ command. See the attached code from rows 6000 to 6200. Not as simple as the Locomotive SYMBOL command, but close enough.

***NOTE**: I also created an Excel spreadsheet to automate the writing of rows needed to redefine a single character: if you are interested to receive it, just let me know.*

PRINT AT



LEVEL: 1 - LIVES: 3 - POINTS: 150

Another peculiarity of the Simons' Basic is the PRINT AT command. Completely missing in the Basic standard, it is comparable to the Locomotive's LOCATE command.
The choice of Simons' Basic was also made for this reason, since my Amstrad game makes extensive use of this command to print objects on the screen. Unfortunately due to poor performance, I could benefit of this command only in some parts of the code. For game animations and collisions control I had to rewrite the logic from scratch.

PEEK and POKE
In the Amstrad CPC program, the movement of the spaceship is generated by a series of characters printed on the screen and positioned with the LOCATE command.

Although it was possible to do the same in Simon's Basic, using the PRINT AT statement, to increase the speed I preferred to print the characters on screen by writing them directly in memory using the POKE command. You might think it's hard to calculate all the correct memory locations, but what at first glance might seem like a complicated operation is actually quite simple if you have the Commodore 64 video memory map in mind. Commodore 64's video memory consists of 25 rows and 40 columns that generally start from the $400 (1024) memory location. But if you remember, as per the annotiation of the MEM command in the Simons' Basic manual, after executing this command the video memory of the Commodore 64 is moved to the location **$CC00 (52224)**.

The movement of the alien spaceship is managed by two FOR cycles, one for rows and one for columns. It's pretty obvious that, in order to print the alien spaceship on the screen you can use the coordinates provided by the FOR cycles and sum them to the initial value of the video memory. Nothing really complex. See the variable AM in the line 1110.

Once the position of the spaceship has been calculated, simply write the corresponding character directly into the video memory using THE POKE command.
See line 1200 which prints the front of the spaceship.

Line 1220 prints the back of the spaceship (the spaceship consists of two characters). Line 1230, cleans up the trail of the spaceship, printing the character 32 (a blank space) on the screen. Repeating this operation creates the effect of the movement, from left to right, of the spaceship.
The same logic was adopted for the movement of the bomb, this time vertically.

What about the collisions?
In Locomotive Basic collisions were managed using the COPYCHR$() command which reveal the value of a character in a defined position indicated through the LOCATE function.
But wait a minute, I already know the position of my spaceship... I don't need the LOCATE this time. Maybe I can read the value of the character in that particular position by directly reading the value of the video memory.
How? Using the PEEK statement of course.
See again the line 1110.

Once done, it is sufficient to check this value and, if different from a blank space (character 32) or the spaceship itself (characters 254 and 255), it means that there is a collision.
See line 1140.

IF THEN :ELSE
Another reason that led me to use Simons' Basic is the possibility of using the IF THEN :ELSE construct.

As we all know, Basic V2 lacks of the ELSE condition in the IF statement. In my opinion, this is a rather important deficiency because it forces the programmer to carry out unnecessary checks. Additionally it makes the already poor readability of the Basic programs even poorer. Luckily Simons' Basic covers this gap by implementing the ELSE condition through the syntax :ELSE.

LEVEL: 1 - LIVES: 3 - POINTS: 800

I had to rewrite all the IF - THEN - ELSE conditions to fit the new syntax, but at least I didn't have to rewrite the logic of this part of the code too.

Well, the game's ready now. Porting it to the C64 cost me only a couple of afternoons of adaptation, thanks mainly to the flexibility of Simons' Basic.

Useful links:
From this page you can download the Simons' Basic:
https://www.c64-wiki.com/wiki/Simons%27_BASIC

Here you can find the manual in Italian:
https://archive.org/details/simonsbasic_en

Ah, I forgot... The bomb can be shoot with the '1' key.
Have fun!

```
10 rem ***********************************
11 rem alien attack! by francesco fiorentini
12 rem commodore 64 - simons' basic
15 rem September 2020
16 rem ***********************************
17 hi=5000
18 gosub 10000: rem intro
20 print chr$(147)

29 rem chars redefined
30 gosub 6000
37 rem np=initial palaces numbers - th=top high
38 rem vm=starting point of video memory after mem
39 rem it is 52224 but my iterations start at 1...
40 np=6:th=6:lv=1:lf=3:pt=0:ex=15000:vm=52223

69 rem create the background
70 gosub 7000
99 rem initialize game's parameters
100 bf=0:i=0

998 rem ship movement
999 rem r=row am=alienmovement
1000 for r = 1 to 23
1100 for i = 1 to 40
1110 am=vm+i+((r-1)*40): ck=peek(am)
1120 rem collision control
1140 if ck<>32 and ck<>254 and ck<>255 then goto 3000
1200 poke am,af: poke am+3072,0
1220 if i>1 then poke am-1,ab
1230 if i>2 then poke am-2,32
1239 rem controllo della bomba
1240 gosub 5000
1280 next i
1285 poke am,32: poke am-1,32
1290 pt=pt+50: if pt>=ex then 1291:else: goto 1298
1291 ex=ex+15000:lf=lf+1
1292 print at(1,23) "level:"; lv ;"- lives:"; lf ; "- points:"; pt
1293 goto 1299
1298 if lv< 10 then print at(30,23) pt :else:print at(31,23) pt
1299 if sf$="0000000000000000000000000000000000000000" then goto 1330
1300 rem print at(1,23) sf$
1301 next r
1309 rem level completed!!! move to next one
1310 gosub 6500
1320 goto 70
1329 rem level completed before reaching the ground
1330 pt=pt+(23-r)*100: goto 1310

3000 rem crash
3005 cr$="***":pr$="   ":x=i-2:if x<=0 then x=1:cr$="**":pcr$="  "
3010 print at(x-1,r-1) cr$
3020 lf=lf-1
3030 if lf=0 then goto 3500
3040 for t=1 to 1000: next t
3050 print at(x-1,r-1) pr$
3060 print at(1,23) "level:"; lv ;"- lives:"; lf ; "- points:"; pt
3061 rem print at(10,25) "- hi-score:"; hi; "-"
3070 goto 99
```

```
3499 rem game over
3500 print chr$(147)
3510 print at(9,6) "----  game over  ----"
3511 print at(13,8) "score: "; pt
3525 for t=1 to 1000: next t
3530 print at(9,10) "-- play again? y/n --"
3531 if pt>hi then hi=pt
3550 get re$
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 game end
4999 end

5000 rem bomb routines - br=bombrow - bl=bombline bf=bombflag(0/1=n/y)
5005 get kp
5010 if kp=1 and bf=0 then 5020:else: goto 5025
5019 rem a new bomb
5020 br=r+1: bl=i: bf=1: goto 5190
5024 rem check if a bomb already exists
5025 if bf=0 then 5200:else: br=br+1
5030 if br=24 then goto 5210
5040 poke vm+((br-2)*40)+bl,32
5190 poke vm+((br-1)*40)+bl,bo
5200 return
5210 br=0:bf=0:poke vm+(22*40)+bl,32:sf$=inst("0",sf$,bl)
5220 goto 5200

6000 rem chars redefinition
6001 rem **** IMPORTANT *****
6002 rem $0400-$0757 video-ram before mem
6003 rem video memory starts at 1024
6004 rem $cc00-$cfff video-ram after mem
6005 rem video memory starts at 52224
6009 mem
6010 design 2,$e000 + 254 * 8
6011 @......bb
6012 @.....bbb
6013 @...bbbbb
6014 @..bbb..b
6015 @b.bb.bbb
6016 @bbbbbbbb
6017 @..b...b.
6018 @.b.b.b.b
6020 design 2,$e000 + 255 * 8
6021 @bb......
6022 @bbb.....
6023 @bbbbb...
6024 @b..bbb.b
6025 @bb.bb.bb
6026 @bbbbbbbb
6027 @.b...b..
6028 @b.b.b.b.
6030 design 2,$e000 + 253 * 8
6031 @........
6032 @..b..b..
6033 @...bb...
6034 @..bbbb..
6035 @..bbbb..
6036 @..bbbb..
6037 @..bbbb..
6038 @...bb...
6039 rem palace floors char (230,231,232,233,234,235)
6040 design 2,$e000 + 230 * 8
6041 @bbbbbbbb
6042 @bb.bb.bb
6043 @bbbbbbbb
6044 @bb.bb.bb
6045 @bbbbbbbb
6046 @bb.bb.bb
6047 @bbbbbbbb
6048 @bb.bb.bb
6050 design 2,$e000 + 231 * 8
6051 @bbbbbbbb
```

```
6052 @bbbbb.bb
6053 @bbbbbbbb
6054 @bb.bbbbb
6055 @bbbbb.bb
6056 @bbbbbbbb
6057 @bb.bbbbb
6058 @bbbbbbbb
6060 design 2,$e000 + 232 * 8
6061 @bbbbbbbb
6062 @bb.bb.bb
6063 @bbbbbbbb
6064 @bb.bbbbb
6065 @bbbbb.bb
6066 @bb.bbbbb
6067 @bbbbbbbb
6068 @bbbbbbbb
6070 design 2,$e000 + 233 * 8
6071 @bbbbbbbb
6072 @bbbbbbbb
6073 @bb.bb.bb
6074 @bbbbbbbb
6075 @bb.bb.bb
6076 @bbbbbbbb
6077 @bbbbbbbb
6078 @bbbbbbbb
6080 design 2,$e000 + 234 * 8
6081 @bbbbbbbb
6082 @bb.bb.bb
6083 @bbbbbbbb
6084 @bbbb.bbb
6085 @bbbbbbbb
6086 @bb.bb.bb
6087 @bbbbbbbb
6088 @bbbbbbbb
6090 design 2,$e000 + 235 * 8
6091 @bbbbbbbb
6092 @bb.bbbbb
6093 @bbbbbbbb
6094 @bbbb.bbb
6095 @bbbbbbbb
6096 @bbbbbbbb
6097 @bb.bbbbb
6098 @bbbbbbbb
6099 rem draw palace top1 (240)
6100 design 2,$e000 + 240 * 8
6101 @........
6102 @........
6103 @........
6104 @........
6105 @...bb...
6106 @..bbbb..
6107 @.b.bb.b.
6108 @b.bbbb.b
6109 rem draw palace top2 (241)
6110 design 2,$e000 + 241 * 8
6111 @b......b
6112 @b......b
6113 @b......b
6114 @b......b
6115 @bb....bb
6116 @bbb..bbb
6117 @bbbbbbbb
6118 @bbbbbbbb
6119 rem draw palace top3 (242)
6120 design 2,$e000 + 242 * 8
6121 @...bb...
6122 @.bbbbbb.
6123 @.bbbbbb.
6124 @...bb...
6125 @...bb...
6126 @..bbbb..
6127 @.bbbbbb.
6128 @bbbbbbbb
6129 rem draw palace top4 (243)
6130 design 2,$e000 + 243 * 8
```

```
6131 @...bb...
6132 @...bb...
6133 @...bb...
6134 @...bb...
6135 @...bb...
6136 @..bbbb..
6137 @.bbbbbb.
6138 @bbbbbbbb
6139 rem af=alienfront-ab=alienback-bo=bomb
6190 af=255:ab=254:bo=253
6200 return

6499 rem dinamic level management
6500 np=np+1:lv=lv+1:pt=pt+500:th=th+1
6501 if pt>=ex then ex=ex+15000:lf=lf+1
6504 if th>14 then th=15:if np>19 then np=19
6505 print at(9,4) "----------------------"
6510 print at(9,5) "-- go to level";lv;"---"
6520 print at(9,6) "----------------------"
6525 for t=1 to 1000: next t
6530 print at(9,7) "-- r u ready? y/n ---"
6540 print at(9,8) "----------------------"
6550 get re$
6560 if re$="y" or re$="Y" then goto 6591
6565 goto 6550
6591 print chr$(147)
6600 return

7000 rem draw the background
7001 rem al=aleatorio - np=numero palazzi - p=colore della penna
7002 sf$="0000000000000000000000000000000000000000"
7003 print at(1,24) "level:"; lv ;"- lives:"; lf ; "- points:"; pt
7004 rem print at(1,25) "- hi-score:"; hi; "-"
7010 x=20-(np/2)
7020 for n=1 to np
7022   rem change the top of the palace x=int(rnd(1)*(high-low))+low
7025   a=int((rnd(1)*4)+1)
7026   pp=239+a
7030   hi=int(rnd(1)*(th-2))+2
7031   ox=x
7041   sf$=inst("1",sf$,x)
7042   x=ox
7050   for i=hi-1 to 0 step -1
7051     rem cambia il piano x=int(rnd(1)*(high-low))+low
7052     a=int((rnd(1)*6)+1)
7053     pf=229+a
7063     pk=vm+((23-i-1)*40)+x
7065     if i=hi-1 then 7066:else:goto 7070
7066     poke pk,pp: poke pk+3072,2
7067     goto 7090
7070     poke pk,pf: poke pk+3072,2
7090   next i
7091 x=x+1
7100 next n
7300 return

10000 rem write the intro
10001 ? chr$(147)
10060 print at(5, 2) "our planet is dying."
10061 print at(5, 3) "our species is in danger."
10062 print at(5, 4) "our future is in danger."
10063 print at(5, 5) "our only purpose is survival."
10064 print at(5, 6) "whatever the cost..."
10065 print at(5, 8) "we do not want to live together."
10066 print at(5, 9) "we do not want to live together."
10067 print at(5, 10) "we want the earth!"
10068 print at(5, 11) "whatever the cost..."
10069 print at(5, 13) "whatever the cost!"
10070 print at(5, 15) "this means war!"
10075 for i=1 to 2000:next i
10080 print at(5,19) ">>> alien attack! <<<"
10085 print at(5,21) "commodore 64 porting"
10090 print at(5,22) "2020 - francesco fiorentini"
10100 for i=1 to 3000:next i
10110 return
```

Bombs Away!

by Gianluca Girelli

As Francesco Fiorentini wrote in his article on RMW25 ITA, a couple of months ago our retro coder hearts beat very hard during the "challenge" to create a clone of “Air Attack”, an old game from the 1980s possibly developed for every platform of the time. After so many years, it's amazing how this simple game has not only withstood the passage of time but has actually become an icon that can attract interest in the modern world as well.
It is even more extraordinary how in non-suspicious times (about a month before the challenge) I decided to create my own version to be inserted as a mini-game inside a larger game of next-gen derivation.

While this idea was forming in my mind, the challenge took place on RPI pages and so (a FaceBook Italian group of Retro Programmers), instead of programming the game using "advanced" techniques (layers or double-buffering on a graphic screen), I thought of implementing it using as much code as possible that mimicked its original behavior (text screen and redefined characters).

Unlike Francesco, I did not have time to complete the work by the due date of September 24, but I hope that the effort I made can still be appreciated.

For the sake of brevity, I will focus here only on the differences of this specific port (written in Hollywood, the Multimedia Application Layer by Andreas Falkenhahn) while leaving to the reader the task to study in detail the original implementation (see: RMW Italian edition no. 25, p. 22).

Let us start with the constraints that have been set. As mentioned before, my implementation consists of a mini-game to be activated inside a more complex and larger game in order to progress with the story-plot. For this reason, this particular implementation of "Air Attack" (which I renamed "Blast Away!") could not constitute a real challenge in itself, otherwise the overall level of difficulty would have been too high for the occasional user. That's why I chose to restrict it to:
- 1 level only;
- 1 life only;
- a limited number of buildings to be destroyed;

Figura 1 - Bombs Away! splashscreen

- if the bomb goes off it will destroy the building in its entirety (unlike other implementations);
- level of difficulty set to medium easy (high score not important).

The next step was to define the graphics: using Gimp (but any other program would do) I designed the tiles and other graphic elements, trying to give them a look as pixellated as possible (see also Francesco's article on the subject).

As in the original (and Francesco’s implementation) I used different tiles for the basic shapes of the skyscrapers, a series of "roofs" that could further diversify them, a chequered base to simulate a lawn (or the basic footing of the city), and the "bomber" (our ship) which, given the spatial theme of the game in which “Blast Away!” is set, I decided to draw with the likeness of the "Colonial Viper" from the old TV series "Battlestar Galactica" of the 1980s. The work is completed by a stylized man with an open parachute in case the ship hits a building. The initial splash screen is instead an artwork by my good friend Marco Riva.

The code attached to this article is commented enough to be self-explanatory, but there are a few points that need to be clarified. First of all, there are two procedures, p_Update_input() and p_MouseState(), that are not present in the original BASIC implementation. The reason for their introduction is that today's systems are, of course, much faster than in the past and a simple mouse click actually corresponds to multiple "states" that are stored in the input/output buffer. The consequence, regardless of the user's speed in pressing and releasing the button, is that

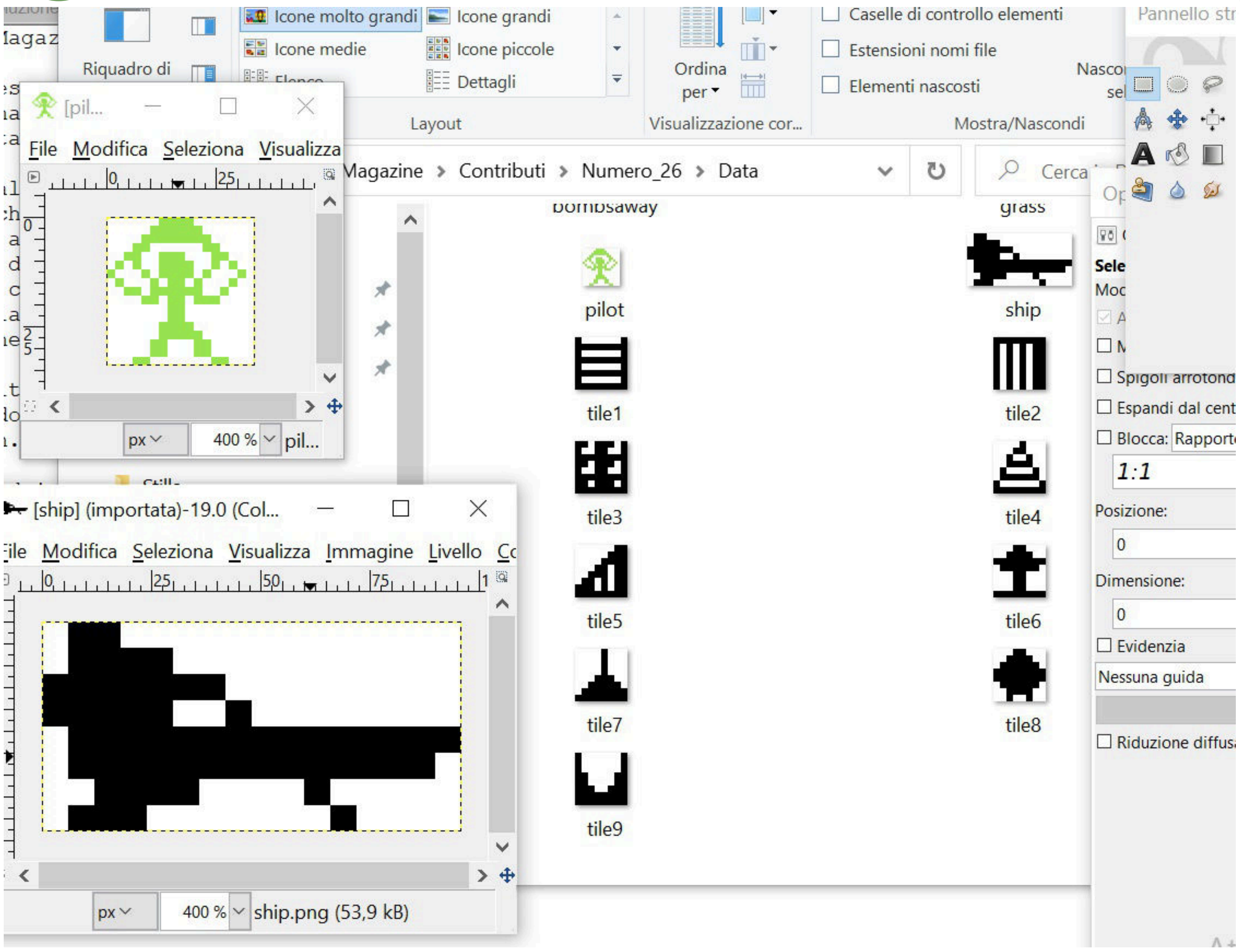


Figure 2 - The tiles used in the game

the system will continue to react as if the player were continuously clicking to shoot. Without going into detail, the aforementioned procedures are meant to limit the program to reacting only once per click.

Another thing that had to be limited was the "range" of horizontal positions from where the bomb can be dropped. Let me explain myself better: unlike the classic implementation on the text screen, where the ship moves from column to column, one at a time and over a limited number of columns (for example 40 on the C64/C128, each 8 pixels wide), in this case our bomb can fall from any point on the screen at the resolution of a single pixel, being such implementation related to a completely graphic environment. This turns into the fact that a building can only be partially destroyed because the bomb, not falling exactly inside the column containing the relevant building, actually "slice it" according to the vertical plan(see figure 3).

The result is a virtual multiplication of the number of on-screen buildings that eventually invalidate the initial assumptions. The solution I identified was to emulate, in a way transparent for the end-users, the effect of the existence of the individual columns: thanks to Gimp I so

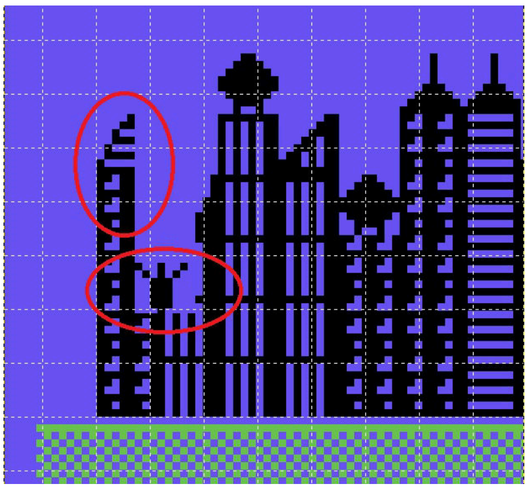


Figure 3 - A partially destroyed building

calculated and stored in a vector the coordinates of the left and right extremes of each individual building. The code then, when it detects the mouse pressure and decides that it is possible to release the bomb (no other bomb already in flight) detects its Y coordinate, compares it with those inside the array and, after having corrected it by an appropriate "offset", drops the bomb.

As for the main loop of the game, it is quite simple and follows the principles already theorized in my article on "Game Coding Notes" published on RMW ITA 17: the aim is to keep everything as simple and linear as possible, so as to facilitate maintenance and future expansions of the code. The gameplay control section is virtually detached from the way you implement it on screen, so you can more easily adapt the program to different development environments. In more detail, as a first thing you read all the inputs and calculate the reactions; only at the end you do create the graphics with a single rendering cycle. In this way the movement is more fluid and devoid of the "hang-ups" that are perceived when the rendering takes place in several phases (e.g.: calculate ship position; draw ship; calculate bomb position; if there is an impact show explosion animation; draw bomb) because during the execution of a single animation phase the rest is stalled.

Put into "natural language":

```
Repeat
read input
determine if it is possible to drop a bomb
calculate the next ship position
if the bomb is in flight, calculate the next position of the bomb
calculate new score in case of bomb/building collisions
render the screen (ship, bomb, explosion animation frame etc.)
check conditions for game over/game won
Until exit=True
```

With regard to the end-of-game conditions, the mechanism is very simple: if the next position of the ship is already occupied (this check is carried out by verifying the color of the first pixel beyond the nose of the ship: if it is black we have a building in front of us) then we are in collision situation(game over); otherwise, if all the buildings have been destroyed, we won. Like Francesco, I also decided to link this check to the content of an array whose number of cells is equal to the virtual columns on the screen, and the content of the single cell is "1" if a building exists and is still standing.
The code also contains additional controls, the explanation of which is not important here.

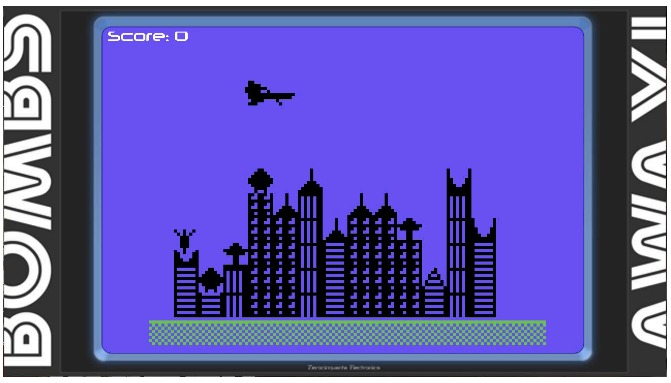


Figure 4 - In-game graphic

Finally, since modern games run in full-screen and it was necessary to simulate an old monitor with edges around the screen, I used the Hollywood "clip region" command:

```
CreateClipRegion(1, #BOX, 190, 53, 928, 623)
SetClipRegion(1)
```

This command is used to create and activate an area on the screen (which can also have complex shapes) that acts as a mask: everything inside it is displayed on the screen, while the rest is hidden. This way our ship will pop up from the left edge of the screen and then will gradually disappear as it crosses the right edge.

Francesco's code, related to the implementation of his “Alien Attack” on Amstrad, ends with the redefinition of the characters necessary to print instructions and scores on video. While a port of this code for C64 and C128 will be published in one of the upcoming issues of RMW, I have opted to use a third-party futuristic font. This font, called "Neuropol", is free and freely downloadable for non-commercial purposes from the link at the bottom of the page. Although the code shown has not been developed for retro systems, I hope you will still appreciate its spirit and I hope that the programming techniques shown here can be of use to you in the future. Blast Away! runs on various systems, the first of which is the AmigaOS4.1. Through Hollywood it can be easily compiled for "Classic" systems (AmigaOS3.x) as long as you lower the size of the graphics assets.

Source and compiled executable for Win32 systems can be downloaded from:
www.gdg-entertainment.it/rmw/bombs_away.zip
Have fun!

```
/*****************************************************************
**                                                              **
** Name:        Bombs Away!                                     **
** Author:      Gianluca Girelli                                **
** Original Code and Graphics: Gianluca Girelli                 **
** Additional Graphics: Marco Riva                              **
** Version:     1.0                                             **
** Date:        08.09.20                                        **
** Last update: 19.10.20                                        **
** Interpreter: any                                             **
** Licence:     Creative Common 4.0 (CC BY-NC-SA 4.0 INT)       **
** Function:    Hollywood implementation of "Air Attack"        **
**              8-bit videogame.                                **
** History:                                                     **
** 1.0: (22.09.20)                                              **
** -game complete                                               **
** 0.1: (08.09.20)                                              **
** -Initial test                                                **
*****************************************************************/

@DISPLAY 1, {X=#CENTER, Y=#CENTER, Width=1280, Height=720, HideTitleBar=True,
   Sizeable=True, ScaleMode=#SCALEMODE_AUTO, Mode = "ask", FitScale = True, KeepProportions = True}

@FONT 1, "neuropol", 36,{Engine=#FONTENGINE_INBUILT}

state=0 ;must be global variable

Function p_Update_input()
;store previous mouse state and read the new one
        laststate=currentstate
        currentstate=IsLeftMouse()
EndFunction

Function p_MouseState()
; compare present and previous mouse state to determine action to take.
; Events are triggered only if mouse button was just release (mouse state=3)
        If laststate=True
                If currentstate=True
                        state=4         ;still pressed
                Else
                        state=3         ;just released
                EndIf
        Else
                If currentstate=True
                        state=2         ;just pressed
                Else
                        state=1         ;still released
                EndIf
        EndIf
EndFunction

Function p_AdjustCoordinates(bomb_x_coord)
;mimics a text-only screen by creating "virtual columns" to channel bomb's flight
        If bomb_x_coord>=184 And bomb_x_coord<232 Then result=0
        If bomb_x_coord>=232 And bomb_x_coord<280 Then result=1
        If bomb_x_coord>=280 And bomb_x_coord<328 Then result=2
        If bomb_x_coord>=328 And bomb_x_coord<376 Then result=3
        If bomb_x_coord>=376 And bomb_x_coord<424 Then result=4
        If bomb_x_coord>=424 And bomb_x_coord<472 Then result=5
        If bomb_x_coord>=472 And bomb_x_coord<520 Then result=6
        If bomb_x_coord>=520 And bomb_x_coord<568 Then result=7
        If bomb_x_coord>=568 And bomb_x_coord<616 Then result=8
        If bomb_x_coord>=616 And bomb_x_coord<664 Then result=9
        If bomb_x_coord>=664 And bomb_x_coord<712 Then result=10
        If bomb_x_coord>=712 And bomb_x_coord<760 Then result=11
        If bomb_x_coord>=760 And bomb_x_coord<808 Then result=12
        If bomb_x_coord>=808 And bomb_x_coord<856 Then result=13
        If bomb_x_coord>=856 And bomb_x_coord<904 Then result=14
        If bomb_x_coord>=904 And bomb_x_coord<952 Then result=15
        If bomb_x_coord>=952 And bomb_x_coord<1000 Then result=16
        If bomb_x_coord>=1000 And bomb_x_coord<1048 Then result=17
        If bomb_x_coord>=1048 And bomb_x_coord<1096 Then result=18
        If bomb_x_coord>=1096 And bomb_x_coord<1114 Then result=19
        Return(result)
EndFunction

Function p_minigame()
        LoadBrush(140, "Data/44_2.jpg")
        LoadBrush(141, "Data/tile1.png", {Transparency = #WHITE})
        LoadBrush(142, "Data/tile2.png", {Transparency = #WHITE})
        LoadBrush(143, "Data/tile3.png", {Transparency = #WHITE})
        LoadBrush(144, "Data/tile4.png", {Transparency = #WHITE})
        LoadBrush(145, "Data/tile5.png", {Transparency = #WHITE})
        LoadBrush(146, "Data/tile6.png", {Transparency = #WHITE})
        LoadBrush(147, "Data/tile7.png", {Transparency = #WHITE})
        LoadBrush(148, "Data/tile8.png", {Transparency = #WHITE})
        LoadBrush(149, "Data/tile9.png", {Transparency = #WHITE})
        LoadBrush(150, "Data/ship.png", {Transparency = #WHITE})
        LoadBrush(151, "Data/bomb.png", {Transparency = #WHITE})
        LoadBrush(152, "Data/pilot.png", {Transparency = #WHITE})
        LoadBrush(153, "Data/bombsaway.png", {Loadalpha = True})
        LoadBrush(154, "Data/grass.png", {Transparency = #WHITE})

        Local x=328             ;coordinates of first tile to be drawn
        Local y=558
        Local altpal=0          ;height of current palace
        Local whichtype=0       ;defines which kind of palace to be drawn (3 different types)
        Local whichtop=0        ;defines which kind of top to be drawn (6 different types)
        Local ship_x=94         ;ship initial coordinates
        Local ship_y=100
        Local bomb_x=bomb_y=0  ;initial bomb coordinates
        Local exit=False
        Local fire_enabled=True
        ;all buildings standing + virtual columns
        Local a = {0,0,0,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,1,0,0,0,0}
        ;virtual columns horizontal boundaries (coordinate in pixels)
        Local b =
{184,232,280,328,376,424,472,520,568,616,664,712,760,808,856,904,952,1000,1048,1096,1114}
        Local column=0
        score=0
        sum=0                   ;transit variable for standing buidings

        DisplayBrush(140, #CENTER, #CENTER)   ;background
        DisplayBrush(153, #CENTER, #CENTER)   ;title

        WaitLeftMouse()

        Cls
        SetFont("neuropol", 48)
        SetFontStyle(#BOLD)
        DisplayBrush(140, #CENTER, #CENTER)   ;background
        CreateTextObject(1, "How to play")
        CreateTextObject(2, "Your ship is slowly flying into the ground. Destroy the buildings for a safe
landing. Press left mouse button to drop a bomb.", {Align=#JUSTIFIED, wordwrap=750})
        CreateTextObject(3, "Press left mouse button to start.", {Align=#JUSTIFIED, wordwrap=650})
        DisplayTextObject(1, #CENTER, 80)
        DisplayTextObject(2, #CENTER, 200)
        DisplayTextObject(3, #CENTER, 520)

        WaitLeftMouse()

        Cls
        DisplayBrush(140, #CENTER, #CENTER)   ;background

        SetFillStyle(#FILLCOLOR)
        SetFormStyle(#NORMAL)
        CreateClipRegion(1, #BOX, 190, 53, 928, 623)
        SetClipRegion(1)

        ;generating world
        For Local i=280 To 1000 Step 48
                DisplayBrush(154, i, 612)
                Wait(3)
        Next
        For Local i=1 To 13
                altpal=Rnd(5)+1
                whichtype=Rnd(3)+1
                For Local j=1 To altpal
                        DisplayBrush(140+whichtype, x, y)
                        Wait(3)
                        y=y-48
                Next
                whichtop=Rnd(6)+4
                DisplayBrush(140+whichtop, x, y)
                y=558
                x=x+48
        Next

        SetFont("neuropol", 32)
        SetFontStyle(#BOLD)
        SetFontColor(#WHITE)
        Locate(200,45)
        Print("Score: "..score)

        ;commencing game
        Repeat
                laststate=currentstate=0 ; flush mouse buffer

                ;read input
                p_Update_input()
                p_MouseState()

                ;react to input
                If state=3 And fire_enabled And ship_x>184 ;fire has been pressed and bomb is still
unreleased
                        column=p_AdjustCoordinates(ship_x+24)
                        bomb_x=b[column]
                        bomb_y=ship_y+48
                        fire_enabled=False     ;bomb released. no more firing until it hits ground
                        state=0                ;flush mouse buffer
                EndIf

                ;compute next ship position
                If ship_x<1115          ;1015
                        ship_x=ship_x+4
                Else
                        ship_x=94       ;190
                        ship_y=ship_y+48
                EndIf

                ;if bomb's away, compute next bomb position
                If bomb_y<607 And fire_enabled=False Then bomb_y=bomb_y+8

                ;updates score in case of impact with standing building (p_GameProgress())
                If fire_enabled=False And a[column]=1
                        a[column]=0
                        score=score+50
                EndIf

                ;render screen
                DisplayBrush(150, ship_x, ship_y)
                If fire_enabled=False Then DisplayBrush(151, bomb_x, bomb_y)
                Wait(1)
                Box(ship_x, ship_y, 96, 48, $6953f5)
                If fire_enabled=False Then Box(bomb_x, bomb_y, 48, 48, $6953f5)

                If bomb_y+48>=606
                        fire_enabled=True
                        Box(200, 45, 500, 48, $6953f5)
                        Locate(200,45)
                        Print("Score: "..score)
                EndIf

                ;termination conditions
                sum=0
                For Local i=0 To 12
                        sum=sum+a[i]
                Next
                If score=650 And sum=0 And bomb_y+48>=606    ;game won
                        ;move ship out of screen boundaries on current row
                        For Local i=ship_x To 1120 Step 4
                                DisplayBrush(150, i, ship_y)
                                Wait(1)
                                Box(i, ship_y, 96, 48, $6953f5)
                        Next
                        ;land ship
                        MoveBrush(150, 190, ship_y+48, 700, 558,{Speed = #SLOWSPEED})
                        Box(700, 558, 96, 48, $6953f5)
                        MoveBrush(150, 700, 558, 900, 558,{Speed = #SLOWSPEED})
                        SetFont("neuropol", 64)
                        SetFontStyle(#BOLD)
                        SetFontColor(#WHITE)
                        CreateTextObject(1, "Game Won !", {Align=#JUSTIFIED, wordwrap=750})
                        DisplayTextObject(1, #CENTER, #CENTER)
                        If level=1 Then arcade_game_won=1 Else arcade_game2_won=1
                        Wait(150)
                        exit=True
                EndIf

                If (ship_x+97)<1015    ;avoid the following checks go out of boundaries
                        If ReadPixel(ship_x+97, ship_y+26)=0  ;game over
                                Box(ship_x, ship_y, 96, 48, $6953f5)
                                While ship_y+34<607            ;pilot ejecting and gliding toward ground
                                        DisplayBrush(152, ship_x+34, ship_y)
                                        Wait(1)
                                        Box(ship_x+34, ship_y, 34, 34, $6953f5)
                                        ship_y=ship_y+2
                                Wend
                                SetFont("neuropol", 64)
                                SetFontStyle(#BOLD)
                                SetFontColor(#WHITE)
                                DisplayBrush(152, ship_x+34, ship_y) ;fix pilot on ground
                                CreateTextObject(1, "Game Over", {Align=#JUSTIFIED, wordwrap=750})
                                DisplayTextObject(1, #CENTER, 250)
                                Wait(150)
                                exit=True
                        EndIf
                EndIf

                WaitTimer(1, 40)  ; script timed at 25fps
        Until exit=True

        ;resets game state
        exit=False

        FreeClipRegion(1)
        For Local i=140 To 154 Do FreeBrush(i)

EndFunction

StartTimer(1)
p_minigame()
```

A bit of rarity

(rummaging here and there)

A treat without trick

by Alberto (Ghostbuster) Apostolo

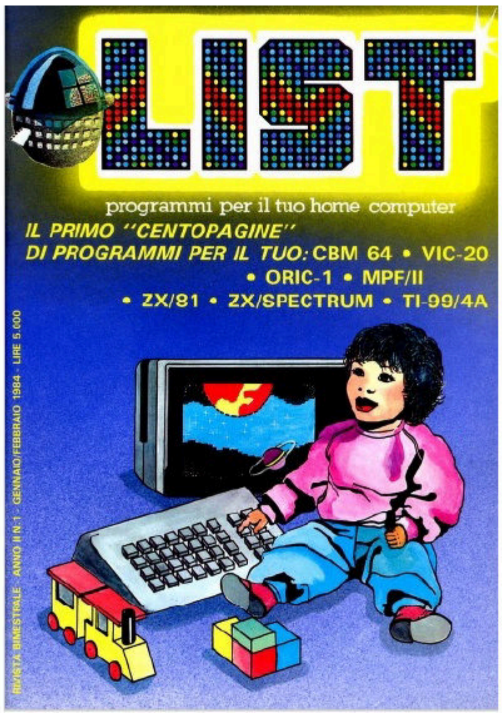
In the past, someone complained that RetroMagazine World often talks about Commodore and Sinclair computers only. We will never stop repeating that RetroMagazine World is a magazine open to anyone who has something interesting to tell about hardware and software of any computer model. For this issue I have composed a reasoned list of some programs and articles published in the Italian magazine LIST for ORIC-1, Sega SC-3000, Sharp MZ-700 and MicroProfessor II.

LIST was a very underrated computer magazine (when I was a high school computer science student, some professors advised against buying it). Almost all the articles published were easy to understand and some were didactically oriented.

I hope I have done some justice to LIST and provided interesting material to computer owners who have had little space in the market. Those who do not understand the Italian language can easily translate articles and wordings in programs with online automatic translators.

The (not complete yet!) collection of LIST issues can be found on the page:

https://archive.org/details/listprogrammi


LIST
programmi per il tuo home computer
IL PRIMO "CENTOPAGINE" DI PROGRAMMI PER IL TUO: CBM 64 • VIC-20 • ORIC-1 • MPF/II • ZX/81 • ZX/SPECTRUM • TI-99/4A
RIVISTA BIMESTRALE - ANNO II N. 1 - GENNAIO/FEBBRAIO 1984 - LIRE 5.000

https://archive.org/details/LIST1984-01

Pag. 08-08 Sega SC-3000 (technical sheet)
Pag. 50-51 ORIC-1 "La fontana" (The fountain, graphics)
Pag. 56-56 Sharp MZ-700 (technical sheet, part 1)
Pag. 57-61 ORIC-1 "Entertainer Rag" (music)
Pag. 64-67 MPF II "Oroscopo" (Horoscope, astrology)
Pag. 68-70 MPF-II "Enalotto" (Italian horse-racing pools)
Pag. 94-95 "Dentro il computer" (Inside the computer, education, part 1)

https://archive.org/details/LIST1984-02

Pag. 11-12 ORIC-1 "Tris" (Tic-Tac-Toe, game)
Pag. 15-16 ORIC-1 "Rally" (game)
Pag. 23-27 MPF II "High Driver" (game)
Pag. 37-38 MPF II "Banca" (Bank account, business)
Pag. 39-40 MPF II "Istogrammi" (Histograms, utility)
Pag. 45-45 ORIC-1 "La torre" (The Tower, math game)
Pag. 46-46 Sharp MZ-700 (technical sheet, part 2)
Pag. 47-48 Sharp MZ-700 "Mastermind" (game)
Pag. 48-49 Sharp MZ-700 "Armonia" (Harmony, astrology)
Pag. 60-62 ORIC-1 "Zhorick" (game)
Pag. 81-82 ORIC-1 "Mosca cieca" (Blindman's bluff, game)
Pag. 83-85 MPF II "O-X" (Othello,Reversi, game)

Pag. 86-86 ORIC-1 "Conversione da ISSUE binario a decimale" (bin-to-dec, utility)
Pag. 92-94 "Dentro il computer" (Inside the computer, education, part 2)

https://archive.org/details/LIST1984-03
Pag. 13-13 ORIC-1 "Coder-Decoder" (cryptography)
Pag. 14-14 ORIC-1 "Renumber Line" (utility)
Pag. 21-22 ORIC-1 "Falciatrice" (The mower, game)
Pag. 36-39 "Dentro il computer" (Inside the computer, education, part 3)
Pag. 40-40 Correzioni al programma "Tris" (Bug-List of program "Tris", see List n. 1)
Pag. 41-41 Sharp MZ-700 "Caccia al tesoro" (Treasure hunt, game)
Pag. 42-42 ORIC-1 "Istogrammi" (Histograms, utility, Sharp MZ-700 printed in header is a mistake)
Pag. 43-45 ORIC Atmos 48K (technical sheet)
Pag. 57-58 Sega SC-3000 "Cascatutto" (Everything Falls, game)
Pag. 59-60 Sega SC-3000 "Corso Basic" (BASIC course, education, part 1)
Pag. 69-75 MPF II "Fatturazione magazzino" (Inventory invoice, business, part 1)

https://archive.org/details/LIST1984-04
Pag. 18-18 ORIC-1 "Tabella" (printing price lists, utility)
Pag. 32-33 Sega SC-3000 "Corso Basic" (BASIC course, education, part 2)
Pag. 34-35 Sega SC-3000 "Bioritmi" (Biorythm, utility)
Pag. 38-38 Sharp MZ-700 "Istogrammi di percentuali" (Histograms and percentage, utility)
Pag. 39-40 Sharp MZ-700 "Riunisci...la musica" (Collecting music, music)
Pag. 42-43 "Dentro il computer" (Inside the computer, education, part 4)
Pag. 68-69 ORIC-1 "Calendario" (Calendar utility)
Pag. 73-74 MPF II "Quattro in fila" (Four-in-a-row, game)
Pag. 88-90 MPF II "Fatturazione magazzino" (Inventory invoice, business, part 2)

https://archive.org/details/LIST1984-05
Pag. 14-14 Sega SC-3000 "Totocalcio" (Italian football pools)
Pag. 30-30 Sharp MZ-700 "Risoluzione di equazioni con il metodo di Cramer" (Cramer's method, math)
Pag. 45-47 "Dentro il computer" (Inside the computer, education, part 5)
Pag. 64-65 MPF II "Snake" (game)
Pag. 72-73 Sega SC-3000 "Corso di Basic" (BASIC course, education, part 3,4)

https://archive.org/details/LIST1984-06
Pag. 20-21 MPF II "Viaggio nello spazio" (Space journey, game)
Pag. 29-30 ORIC-1 "Gran Prix" (game)
Pag. 31-34 Sega SC-3000 "Caccia al sottomarino" (Submarine hunt, game)
Pag. 35-36 Sharp MZ-700 "Incontri di calcio" (handling football championship results)
Pag. 56-57 "Dentro il computer" (Inside the computer, education, part 6)
Pag. 66-67 Sharp MZ-700 "Tombola" (handling the Bingo)
Pag. 77-78 Sega SC-3000 "Video Picture" (graphics)
Pag. 87-89 Sega SC-3000 "Corso di Basic" (BASIC course, education, part 5)
Pag. 90-91 Sega SC-3000 "Outline" (how changing your character set)

https://archive.org/details/LIST1985-01/mode/2up
Pag. 08-09 Sega SC-3000 "Morra" (Chinese Morra, game)
Pag. 10-11 Sega SC-3000 "Mastermind" (game)
Pag. 15-17 Sharp MZ-700 "Su e giù per la penisola" (part 1, education, geography of Italy)
Pag. 22-24 Sharp MZ-700 "Attacco aereo" (Air attack, game)
Pag. 27-29 ORIC-1 "Levrieri" (Greyhounds, game)
Pag. 45-48 MPF II "Piramide" (Marienbad, game)
Pag. 49-50 Sharp MZ-700 "Istruzioni POKE" (education)
Pag. 53-56 Sega SC-3000 "Corso di Basic" (BASIC course, education, part 6,7)
Pag. 74-77 ORIC-1 "Break-Out" (game)
Pag. 78-79 Sega SC-3000 "Astro War" (game)
Pag. 80-82 Sega SC-3000 "Paroliamo" (Countdown word game)
Pag. 89-90 ORIC-1 "Salta la rana" (Frog, game)

Pag. 91-93 Sharp MZ-700 "Electronic Mastermind" (game)
Pag. 94-96 Sharp MZ-700 "Saliscendi" (Rise-and-Fall, game)

https://archive.org/details/LIST1985-02
Pag. 26-26 Sharp MZ-700 "Load and Run" (education)
Pag. 28-31 Sega SC-3000 "Corso di Basic" (BASIC course, education, part 8,9,10)
Pag. 32-35 ORIC-1 "Char-constructor" (utility)
Pag. 40-42 ORIC-1 "Smash" (The Wall, game)
Pag. 43-45 Sharp MZ-700 "Su e giù per la penisola" (part 2, education, geography of Italy)
Pag. 57-58 Sega SC-3000 "Uova Spaziali" (Space Eggs, game)
pag. 59-61 Sharp MZ-700 "JAZZI" (Jahtzee, game)
Pag. 67-69 Sharp MZ-700 "Boxe" (game)
Pag. 70-72 Sega SC-3000 "Tiro a volo" (Shooting, game)
Pag. 85-88 Sharp MZ-700 "Slot machine" (game)
Pag. 89-90 Sega SC-3000 "Pianoforte" (music)

https://archive.org/details/LIST1985-04
Pag. 31-33 Sega SC-3000 "Super Master Mind" (game)
Pag. 37-38 Sharp MZ-700 "Sci alpino" (Skiing, game)
Pag. 40-41 Sega SC-3000 "Real golf" (game)
Pag. 46-48 Sharp MZ-700 "Dispersioni termiche" (Heat loss, science)
Pag. 49-50 MPF II "Contraerea" (Flak, game)
Pag. 53-54 Sharp MZ-7000 "Bioritmi" (Biorythm, utility)
Pag. 55-56 ORIC-1 "Cubi" (Cubes, game)
Pag. 57-58 MPF II "Mastermind" (game)
Pag. 84-85 Sega SC-3000 "Archer" (game)

https://archive.org/details/LIST1985-05
Pag. 37-39 Sega SC-3000 "Grafica in 3D" (graphics)
Pag. 40-42 Sharp MZ-7000 "Caccia all'U-boot" (U-boot hunting, game)
Pag. 48-49 ORIC-1 "Oric Sequencer" (music)
Pag. 52-57 MPF II "Black-Jack" (game)
Pag. 62-63 Sega SC-3000 "Decisioni" (Choices, mathematical logic)
Pag. 64-65 Sharp MZ-700 "Analisi" (Calculus, math utility)
Pag. 70-71 ORIC-1 "Equazioni di 1° e 2° grado" (math)
Pag. 72-73 ORIC-1 "Semina" (Seeding, game)

https://archive.org/details/LIST1985-08-09
Pag. 74-78 Sega SC-3000 "Math Software" (math utility)
Pag. 79-81 Sega SC-3000 "Fasi Lunari" (Moon phases, astronomy)
Pag. 82-83 Sharp MZ-700 "Ferma il totale" (Stop the total, math game)
Pag. 84-85 Sharp MZ-700 "Biglietti da visita" (business cards printing, utility)

https://archive.org/details/LIST1985-08-09-speciali
Pag. 06-07 Sega SC-3000 "Sega Graphics" (graphics)
Pag. 19-20 Sega SC-3000 "Simulatore di volo" (Flight Simulator, game)
Pag. 21-21 Sharp MZ-700 "Archivio Sharp" (Archive, utility)

https://archive.org/details/LIST1985-10-11
Pag. 67-70 Sega SC-3000 "Bosco maledetto" (The cursed wood, adventure)
Pag. 70-71 Sega SC-3000 "Calcolo dei solidi" (Solid geometry, math)
Pag. 72-75 Sharp MZ-700 "Elenco fornitori" (Suppliers' List, business)
Pag. 75-78 Sharp MZ-700 "Sette e mezzo" (7 and half Italian BlackJack, game)

https://archive.org/details/LIST1986-04
Pag. 52-54,63-64 Sega SC-3000 "Drawer" (graphics)
Pag. 65-69 Sharp MZ-700 "Gestione Magazzino" (Inventory management, business)

The Haunted House

...or how to play the same game on 3 different platforms!

by Francesco Fiorentini

In the closing article of the last issue I mentioned that I would enjoy porting Basic code on different platforms. What you have in your hands is only the first of these works and I hope it can meet your appreciation.

Haunted House is a game, a textual adventure, published in the book '**Write your own Adventure Programs for your microcomputers**' published by **Usborne Publishing** in 1983. The book aimed to teach how to design and write a textual adventure in Basic, and Haunted House game was one of the most significant examples of the volume.

You can find the book on archive.org here: https://archive.org/details/Write_Your_Own_Adventure_Programs_1983_Usborne/

While the **GW Basic** version of Haunted House I found it at this address:
https://github.com/robhagemans/hoard-of-gwbasic/blob/master/AllBasicCode/ADV-OLD.BAS

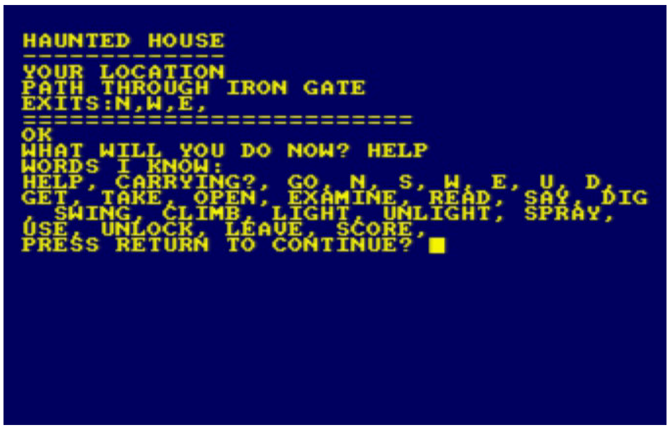
Amstrad CPC

After reviewing the code I realized that it could work without modification on the Amstrad CPC. I then copied and pasted the code into **WinAPE** and like a charm the adventure started smoothly.

I tried to play the game a bit and it seemed to go on without any particular problems.
We can therefore say that the porting for Amstrad CPC was really simple. :-D

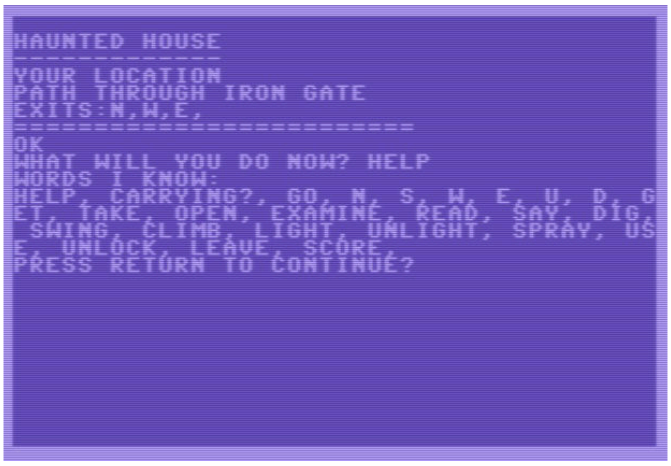
Commodore 64

On the wings of enthusiasm I decided to work on the porting for Commodore 64... And this is where the pain started.



```
HAUNTED HOUSE
-------------
YOUR LOCATION
PATH THROUGH IRON GATE
EXITS:N,W,E,
=========================
OK
WHAT WILL YOU DO NOW? HELP
WORDS I KNOW:
HELP, CARRYING?, GO, N, S, W, E, U, D,
GET, TAKE, OPEN, EXAMINE, READ, SAY, DIG
, SWING, CLIMB, LIGHT, UNLIGHT, SPRAY,
USE, UNLOCK, LEAVE, SCORE,
PRESS RETURN TO CONTINUE? █
```

Figure 1. Haunted House runs on Amstrad CPC



```
HAUNTED HOUSE
-------------
YOUR LOCATION
PATH THROUGH IRON GATE
EXITS:N,W,E,
=========================
OK
WHAT WILL YOU DO NOW? HELP
WORDS I KNOW:
HELP, CARRYING?, GO, N, S, W, E, U, D, G
ET, TAKE, OPEN, EXAMINE, READ, SAY, DIG
, SWING, CLIMB, LIGHT, UNLIGHT, SPRAY, US
E, UNLOCK, LEAVE, SCORE
PRESS RETURN TO CONTINUE?
```

Figure 2. Haunted House porting on Commodore 64

After pasting the code 'as is' into **VICE** I received a series of errors that forced me to review the listing and rewrite part of the logic.

Fortunately, the program already used only two characters long variables and therefore perfectly compatible with the limitation of Basic V2; the main problem of the code was that some program lines exceeded 80 characters in length.

Some of the lines could fortunately be compressed simply by removing unnecessary spaces, but others had to be split to be shortened and in some cases I had to slightly change their logic.
See, for example, the original line 270 that produced lines 270, 271, and 275 in Commodore 64.

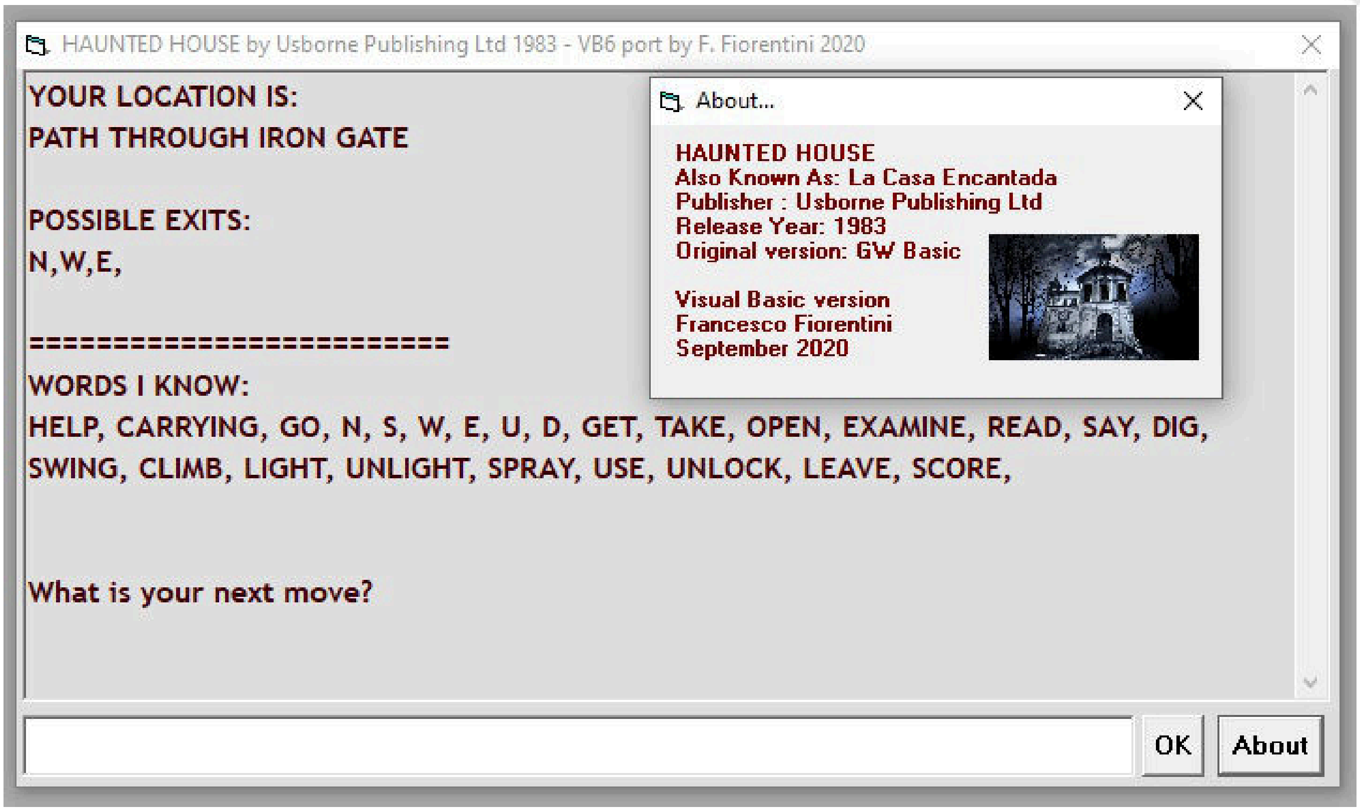
Another striking example is the original line 460, which handles a conditional GOSUB by the VB variable. In this case I had to split the row and also manage the value of the variable VB in order for the conditional jump to remain efficient.
See rows 460, 461, and 465 in Commodore code 64.

All in all, the conversion for Commodore 64 was quite easy, once we understood how and where to act.

Visual Basic 6.0

At this point I wanted to raise the difficulty bar and try to make a much more challenging porting: Visual Basic 6.0.

Some might argue that the VB6 is not exactly retro, but since it was released in 1998 and is already 22 years old, I would say that it can be considered retro in all respects.
Of the 3 portings this was definitely the most challenging.
Adapting the GW Basic code created as' deconstructed


HAUNTED HOUSE by Usborne Publishing Ltd 1983 - VB6 port by F. Fiorentini 2020

```
YOUR LOCATION IS:
PATH THROUGH IRON GATE

POSSIBLE EXITS:
N,W,E,

========================
WORDS I KNOW:
HELP, CARRYING, GO, N, S, W, E, U, D, GET, TAKE, OPEN, EXAMINE, READ, SAY, DIG,
SWING, CLIMB, LIGHT, UNLIGHT, SPRAY, USE, UNLOCK, LEAVE, SCORE,


What is your next move?
```

About...

HAUNTED HOUSE
Also Known As: La Casa Encantada
Publisher : Usborne Publishing Ltd
Release Year: 1983
Original version: GW Basic

Visual Basic version
Francesco Fiorentini
September 2020

Figure 3. Haunted House porting on Visual Basic 6 with the "About..." form

'programming to a Basic' event driven 'such as Visual Basic 6, I must admit that it was not a joke. However, I wanted to try this conversion to understand what level of difficulty it entailed and whether the result, once achieved, was worth the effort or not.

Here is a list of the actions I had to take:
- I had to make a distinction between the descriptive area and the command typing area
- due to the removal of line numbering, I had to modify all conditional jumps (GOSUB) to point to labels (to do this first I kept the line number as the label name)
- VB does not handle READ/DATA, so I had to transform DATA into Array
- old Basic distinguished between variable names if they contained the type indication; e.g. WA, WA$ and WA% were three totally different variables. In this case I had to rename all variables to allow the program to work properly
- I also added an 'About' form to give Usborne Publishing the credits due

All in all, the result is pleasant. The textual adventure reflects the spirit of the games of the past, but with a decidedly more modern look and feel.

In addition, the VB6 offers several potentialities to enrich our software, and with relatively little effort. Perhaps in one of RetroMagazine's subsequent releases we could also think of relaunching **Haunted House Enhanced**!
You could add a list to represent inventory, or add a series of buttons to run basic commands (instead of typing them in from time to time).
A self-fulfilling map could also be added as our hero advances in exploration. Not to mention transforming a textual adventure into a textual adventure with graphic, adding an image for each location.

Obviously these are just ideas that I threw down when I wrote the article. But the basis is there and implementing them, as I wrote earlier, would be relatively simple. My hands itch already to get me to work...: -D

I hope this first article on the porting of Basic programs can meet your interest and above all can be a stimulus to rediscover the old programs.

Coming back to Haunted House adventure: if you are experienced adventurers, solving it will be reasonably simple, however if you need any help, here you can find a comfortable walkthrough that will allow you to complete the game:
https://solutionarchive.com/file/id%2C9320/

So I just wish you all a lot of fun and forgive me for the excessive length of the enclosed code, but it is, I hope, for a good cause. :-)

Original GW-Basic code (MS DOS) - also compatible with Locomotive Basic (Amstrad CPC)

```
10 REM HAUNTED HOUSE ADVENTURE
20 REM ***********************
30 REM THIS VERSION FOR "MICROSOFT" BASIC
40 REM REQUIRES A MINIMUM OF 16K
50 REM SELECT "TEXT MODE" IF NECESSARY
60 REM ***********************

70 V = 25: W = 36: G = 18
73 DIM R$(63), D$(63), O$(W), V$(V)
77 DIM C(W), L(G), F(W)
80 GOSUB 1600 '----> DO INITIALISATION

85 REM DESCRIPTION AND FEEDBACK
90 CLS : PRINT "HAUNTED HOUSE"
100 PRINT "-------------"
110 PRINT "YOUR LOCATION"
120 PRINT D$(RM)
130 PRINT "EXITS:";
140 FOR I = 1 TO LEN(R$(RM))
150 PRINT MID$(R$(RM), I, 1); ",";
160 NEXT I
170 PRINT
180 FOR I = 1 TO G
190 IF L(I) = RM AND F(I) = 0 THEN PRINT "YOU CAN SEE "; O$(I); " HERE"
200 NEXT I
210 PRINT "========================="
220 PRINT M$: M$ = "WHAT"
225 REM INPUT AND INPUT ANALYSIS
230 INPUT "WHAT WILL YOU DO NOW"; Q$
240 V$ = "": W$ = "": VB = 0: OB = 0
250 FOR I = 1 TO LEN(Q$)
260 IF MID$(Q$, I, 1) = " " AND V$ = "" THEN V$ = LEFT$(Q$, I - 1)
270 IF MID$(Q$, I + 1, 1) <> " " AND V$ <> "" THEN W$ = MID$(Q$, I + 1, LEN(Q$) - 1): I = LEN(Q$)
280 NEXT I
290 IF W$ = "" THEN V$ = Q$
300 FOR I = 1 TO V
310 IF V$ = V$(I) THEN VB = I
320 NEXT I
330 FOR I = 1 TO W
340 IF W$ = O$(I) THEN OB = I
350 NEXT I
355 REM ERROR MESSAGES OVERRIDE CONDITIONS
360 IF W$ > "" AND OB = 0 THEN M$ = "THAT'S SILLY"
370 IF VB = 0 THEN VB = V + 1
380 IF W$ = "" THEN M$ = "I NEED TWO WORDS"
390 IF VB > V AND OB > 0 THEN M$ = "YOU CAN'T '" + Q$ + "'"
400 IF VB > V AND OB = 0 THEN M$ = "YOU DON'T MAKE SENSE!"
410 IF VB < V AND OB > 0 AND C(OB) = 0 THEN M$ = "YOU DON'T HAVE '" + W$
420 IF F(26) = 1 AND RM = 13 AND FIX(RND(1) * 4) <> 3 AND VB <> 21 THEN M$ = "BATS ATTACKING!": GOTO 90
430 IF RM = 44 AND FIX(RND(1) * 3) = 1 AND F(24) <> 1 THEN F(27) = 1
440 IF F(0) = 1 THEN LL = LL - 1
450 IF LL < 1 THEN F(0) = 0
455 REM BRANCH TO SUBROUTINES
460 ON VB GOSUB 500, 570, 640, 640, 640, 640, 640, 640, 640, 980, 980, 1030, 1070, 1140, 1180, 1220,
1250, 1300, 1340, 1380, 1400, 1430, 1460, 1490, 1510, 1590
470 IF LL = 10 THEN M$ = "YOUR CANDLE IS WANING!"
480 IF LL = 1 THEN M$ = "YOUR CANDLE IS OUT!"
490 GOTO 90

495 REM VERB 1
500 PRINT "WORDS I KNOW:"
510 FOR I = 1 TO V
520 PRINT V$(I); ", ";
530 NEXT I
540 M$ = "": PRINT
550 GOSUB 1580
560 RETURN

565 REM VERB 2
570 PRINT "YOU ARE CARRYING:"
580 FOR I = 1 TO G
590 IF C(I) = 1 THEN PRINT O$(I); ", ";
600 NEXT I
610 M$ = "": PRINT
620 GOSUB 1580
630 RETURN

635 REM VERBS 3 TO 9 INCLUSIVE
640 D = 0
650 IF OB = 0 THEN D = VB - 3
660 IF OB = 19 THEN D = 1
```

```
670 IF OB = 20 THEN D = 2
680 IF OB = 21 THEN D = 3
690 IF OB = 22 THEN D = 4
700 IF OB = 23 THEN D = 5
710 IF OB = 24 THEN D = 6
720 IF RM = 20 AND D = 5 THEN D = 1
730 IF RM = 20 AND D = 6 THEN D = 3
740 IF RM = 22 AND D = 6 THEN D = 2
750 IF RM = 22 AND D = 5 THEN D = 3
760 IF RM = 36 AND D = 6 THEN D = 1
770 IF RM = 36 AND D = 5 THEN D = 2
780 IF F(14) = 1 THEN M$ = "CRASH!  YOU FELL OUT OF THE TREE!": F(14) = 0: RETURN
790 IF F(27) = 1 AND RM = 52 THEN M$ = "GHOSTS WILL NOT LET YOU MOVE": RETURN
800 IF RM = 45 AND C(1) = 1 AND F(34) = 0 THEN M$ = "A MAGICAL BARRIER TO THE WEST": RETURN
810 IF (RM = 26 AND F(0) = 0) AND (D = 1 OR D = 4) THEN M$ = "YOU NEED A LIGHT": RETURN
820 IF RM = 54 AND C(15) <> 1 THEN M$ = "YOU'RE STUCK!": RETURN
830 IF C(15) = 1 AND NOT (RM = 53 OR RM = 54 OR RM = 55 OR RM = 47) THEN M$ = "YOU CAN'T CARRY A BOAT!":
RETURN
840 IF (RM > 26 AND RM < 30) AND F(0) = 0 THEN M$ = "TOO DARK TO MOVE": RETURN
850 F(35) = 0: RL = LEN(R$(RM))
860 FOR I = 1 TO RL
870 U$ = MID$(R$(RM), I, 1)
880 IF (U$ = "N" AND D = 1 AND F(35) = 0) THEN RM = RM - 8: F(35) = 1
890 IF (U$ = "S" AND D = 2 AND F(35) = 0) THEN RM = RM + 8: F(35) = 1
900 IF (U$ = "W" AND D = 3 AND F(35) = 0) THEN RM = RM - 1: F(35) = 1
910 IF (U$ = "E" AND D = 4 AND F(35) = 0) THEN RM = RM + 1: F(35) = 1
920 NEXT I
930 M$ = "OK"
940 IF F(35) = 0 THEN M$ = "CAN'T GO THAT WAY!"
950 IF D < 1 THEN M$ = "GO WHERE?"
960 IF RM = 41 AND F(23) = 1 THEN R$ = "SW": M$ = "THE DOOR SLAMS SHUT!": F(23) = 0
970 RETURN

975 REM VERBS 10 AND 11
980 IF OB > G THEN M$ = "I CAN'T GET " + W$: RETURN
985 IF L(OB) <> RM THEN M$ = "IT ISN'T HERE"
990 IF F(OB) <> 0 THEN M$ = "WHAT " + W$ + "?"
1000 IF C(OB) = 1 THEN M$ = "YOU ALREADY HAVE IT"
1010 IF OB > 0 AND L(OB) = RM AND F(OB) = 0 THEN C(OB) = 1: L(OB) = 65: M$ = "YOU HAVE THE " + W$
1020 RETURN

1025 REM VERB 12
1030 IF RM = 43 AND (OB = 28 OR OB = 29) THEN F(17) = 0: M$ = "DRAWER OPEN"
1040 IF RM = 28 AND OB = 25 THEN M$ = "IT'S LOCKED"
1050 IF RM = 38 AND OB = 32 THEN M$ = "THAT'S CREEPY!": F(2) = 0
1060 RETURN

1065 REM VERB 13
1070 IF OB = 30 THEN F(18) = 0: M$ = "SOMETHING HERE!"
1080 IF OB = 31 THEN M$ = "THAT'S DISGUSTING!"
1090 IF (OB = 28 OR OB = 29) THEN M$ = "THERE'S A DRAWER"
1100 IF OB = 33 OR OB = 5 THEN GOSUB 1140
1110 IF RM = 43 AND OB = 35 THEN M$ = "THERE'S SOMETHING BEYOND..."
1120 IF OB = 32 THEN GOSUB 1030
1130 RETURN

1135 REM VERB 14
1140 IF RM = 42 AND OB = 33 THEN M$ = "THEY ARE DEMONIC WORKS"
1150 IF (OB = 3 OR OB = 36) AND C(3) = 1 AND F(34) = 0 THEN M$ = "USE THIS WORD WITH CARE 'XZANFAR'"
1160 IF C(5) = 1 AND OB = 5 THEN M$ = "THE SCRIPT IS IN AN ALIEN TONGUE"
1170 RETURN

1175 REM VERB 15
1180 M$ = "OK '" + W$ + "'"
1190 IF C(3) = 1 AND OB = 34 THEN M$ = "*MAGIC OCCURS*": IF RM <> 45 THEN RM = FIX(RND(1) * 64)
1200 IF C(3) = 1 AND OB = 34 AND RM = 45 THEN F(34) = 1
1210 RETURN

1215 REM VERB 16
1220 IF C(12) = 1 THEN M$ = "YOU MADE A HOLE"
1230 IF C(12) = 1 AND RM = 30 THEN M$ = "DUG THE BARS OUT": D$(RM) = "HOLE IN THE WALL": R$(RM) = "NSE"
1240 RETURN

1245 REM VERB 17
1250 IF C(14) <> 1 AND RM = 7 THEN M$ = "THIS IS NO TIME TO PLAY GAMES"
1260 IF OB = 14 AND C(14) = 1 THEN M$ = "YOU SWUNG IT"
1270 IF OB = 13 AND C(13) = 1 THEN M$ = "WHOOSH"
1280 IF OB = 13 AND C(13) = 1 AND RM = 43 THEN R$(RM) = "WN": D$(RM) = "STUDY WITH A SECRET ROOM": M$ =
"YOU BROKE THE THIN WALL"
1290 RETURN

1295 REM VERB 18
1300 IF OB = 14 AND C(14) = 1 THEN M$ = "IT ISN'T ATTACHED TO ANYTHING!"
1310 IF OB = 14 AND C(14) <> 1 AND RM = 7 AND F(14) = 0 THEN M$ = "YOU SEE THICK FORREST AND CLIFF
```

```
SOUTH": F(14) = 1: RETURN
1320 IF OB = 14 AND C(14) <> 1 AND RM = 7 AND F(14) = 1 THEN M$ = "GOING DOWN!": F(14) = 0
1330 RETURN

1335 REM VERB 19
1340 IF OB = 17 AND C(17) = 1 AND C(8) = 0 THEN M$ = "IT WILL BURN YOUR HANDS"
1350 IF OB = 17 AND C(17) = 1 AND C(9) = 0 THEN M$ = "NOTHING TO LIGHT IT WITH"
1360 IF OB = 17 AND C(17) = 1 AND C(9) = 1 AND C(8) = 1 THEN M$= "IT CASTS A FLICKERING LIGHT": F(0) = 1
1370 RETURN

1375 REM VERB 20
1380 IF F(0) = 1 THEN F(0) = 0: M$ = "EXTINGUISHED"
1390 RETURN

1395 REM VERB 21
1400 IF OB = 16 AND C(16) = 1 THEN M$ = "HISSSS"
1410 IF OB = 16 AND C(16) = 1 AND F(26) = 1 THEN F(26) = 0: M$ = "PFFT!  GOT THEM"
1420 RETURN

1425 REM VERB 22
1430 IF OB = 10 AND C(10) = 1 AND C(11) = 1 THEN M$ = "SWITCHED ON": F(24) = 1
1440 IF F(27) = 1 AND F(24) = 1 THEN M$ = "WHIZZ - VACUUMED THE GHOSTS UP!": F(27) = 0
1450 RETURN

1455 REM VERB 23
1460 IF RM = 43 AND (OB = 27 OR OB = 28) THEN GOSUB 1030
1470 IF RM = 28 AND OB = 25 AND F(25) = 0 AND C(18) = 1 THEN F(25) = 1: R$(RM) = "SEW": D$(RM) = "HUGE
OPEN DOOR": M$ = "THE KEY TURNS"
1480 RETURN

1485 REM VERB 24
1490 IF C(OB) = 1 THEN C(OB) = 0: L(OB) = RM: M$ = "DONE"
1500 RETURN

1505 REM VERB 25
1510 S = 0
1520 FOR I = 1 TO G
1530 IF C(I) = 1 THEN S = S + 1
1540 NEXT I
1550 IF S = 17 AND C(15) <> 1 AND RM <> 57 THEN PRINT "YOU HAVE EVERYTHING": PRINT "RETURN TO THE GATE
FOR FINAL SCORE"
1560 IF S = 17 AND RM = 57 THEN PRINT "DOUBLE SCORE FOR REACHING HERE!": S = S * 2
1570 PRINT "YOUR SCORE = "; S: IF S > 18 THEN PRINT "WELL DONE! YOU HAVE FINISHED THE GAME": END
1580 INPUT "PRESS RETURN TO CONTINUE"; Q$
1590 RETURN

--------------------------------------------------------------------------
1595 REM GAME INITIALISATION ROUTINE
1600 REM DIM R$(63), D$(63), O$(W), V$(V)
1610 REM DIM C(W), L(G), F(W)
1620 DATA 46,38,35,50,13,18,28,42,10,25,26,4,2,7,47,60,43,32
1630 FOR I = 1 TO G
1640 READ L(I)
1650 NEXT I
1660 DATA HELP,CARRYING?,GO,N,S,W,E,U,D,GET,TAKE,OPEN,EXAMINE,READ,SAY
1665 DATA DIG,SWING,CLIMB,LIGHT,UNLIGHT,SPRAY,USE,UNLOCK,LEAVE,SCORE
1680 FOR I = 1 TO V
1690 READ V$(I)
1700 NEXT I

1705 '----> Possible movements (they are associated with the location, sharing the same index)
1710 DATA SE,WE,WE,SWE,WE,WE,SWE,WS
1720 DATA NS,SE,WE,NW,SE,W,NE,NSW
1730 DATA NS,NS,SE,WE,NWUD,SE,WSUD,NS
1740 DATA N,NS,NSE,WE,WE,NSW,NS,NS
1750 DATA S,NSE,NSW,S,NSUD,N,N,NS
1760 DATA NE,NW,NE,W,NSE,WE,W,NS
1770 DATA SE,NSW,E,WE,NW,S,SW,NW
1780 DATA NE,NWE,WE,WE,WE,NWE,NWE,W
1790 FOR I = 0 TO 63
1800 READ R$(I)
1810 NEXT I

1815 '----> Description of the location
1820 DATA DARK CORNER,OVERGROWN GARDEN,BY LARGE WOODPILE,YARD BY RUBBISH
1830 DATA WEEDPATCH,FOREST,THICK FOREST,BLASTED TREE
1840 DATA CORNER OF HOUSE,ENTRANCE TO KITCHEN,KITCHEN AND GRIMEY COOKER,SCULLERY DOOR
1845 DATA ROOM WITH INCHES OF DUST,REAR TURRET ROOM,CLEARING BY HOUSE,PATH
1860 DATA SIDE OF HOUSE,BACK OF HALLWAY,DARK ALCOVE,SMALL DARK ROOM
1865 DATA BOTTOM OF SPIRAL STAIRCASE,WIDE PASSAGE,SLIPPERY STEPS,CLIFFTOP
1880 DATA NEAR CRUMBLING WALL,GLOOMY PASSAGE,POOL OF LIGHT,IMPRESSIVE VAULTED HALLWAY
1885 DATA HALL BY THICK WOODEN DOOR,TROPHY ROOM,CELLAR WITH BARRED WINDOW,CLIFF PATH
1900 DATA CUPBOARD WITH HANGING COAT,FRONT HALL,SITTING ROOM,SECRET ROOM
```

```
1905 DATA STEEP MARBLE STAIRS,DINING ROOM,DEEP CELLAR WITH COFFIN,CLIFF PATH
1920 DATA CLOSET,FRONT LOBBY,LIBRARY OF EVIL BOOKS,STUDY WITH DESK AND HOLE IN WALL
1925 DATA WEIRD COBWEBBY ROOM,VERY COLD CHAMBER,SPOOKY ROOM,CLIFF PATH BY MARSH
1940 DATA RUBBLE-STREWN VERANDAH,FRONT PORCH,FRONT TOWER,SLOPING CORRIDOR
1945 DATA UPPER GALLERY,MARSH BY WALL,MARSH,SOGGY PATH
1960 DATA BY TWISTED RAILING, PATH THROUGH IRON GATE,BY RAILINGS,BENEATH FRONT TOWER
1965 DATA DEBRIS FROM CRUMBLING FACADE,LARGE FALLEN BRICKWORK,ROTTING STONE ARCH,CRUMBLING CLIFFTOP
1980 FOR I = 0 TO 63
1990 READ D$(I)
2000 NEXT I

2010 DATA PAINTING,RING,MAGIC SPELLS,GOBLET,SCROLL,COINS,STATUE,CANDLESTICK
2012 DATA MATCHES,VACUUM,BATTERIES,SHOVEL,AXE,ROPE,BOAT,AEROSOL,CANDLE,KEY
2014 DATA NORTH,SOUTH,WEST,EAST,UP,DOWN
2016 DATA DOOR,BATS,GHOSTS,DRAWER,DESK,COAT,RUBBISH
2018 DATA COFFIN,BOOKS,XZANFAR,WALL,SPELLS
2060 FOR I = 1 TO W
2070 READ O$(I)
2080 NEXT I
2090 F(18) = 1: F(17) = 1: F(2) = 1: F(26) = 1: F(28) = 1: F(23) = 1: LL = 60: RM = 57: M$ = "OK"
2100 RETURN
```

Commodore 64 version - adapted by F. Fiorentini

```
10 rem haunted house adventure
20 rem ***********************
30 rem this version for Commodore 64
40 rem requires a minimum of 16k
50 rem F.Fiorentini - Ottobre 2020
60 rem ***********************

70 v = 25: w = 36: g = 18
73 dim r$(63), d$(63), o$(w), v$(v)
77 dim c(w), l(g), f(w)
80 gosub 1600 '----> do initialisation

85 rem description and feedback
90 print chr$(147): print "haunted house"
100 print "-------------"
110 print "your location"
120 print d$(rm)
130 print "exits:";
140 for i = 1 to len(r$(rm))
150 print mid$(r$(rm), i, 1); ",";
160 next i
170 print
180 for i = 1 to g
190 if l(i) = rm and f(i) = 0 then print "you can see "; o$(i); " here"
200 next i
210 print "========================="
220 print m$: m$ = "what"
225 rem input and input analysis
230 input "what will you do now"; q$
240 v$ = "": w$ = "": vb = 0: ob = 0
250 for i = 1 to len(q$)
260 if mid$(q$, i, 1) = " " and v$ = "" then v$ = left$(q$, i - 1)
270 if mid$(q$, i + 1, 1) <> " " and v$ <> "" then goto 275
271 goto 280
275 w$ = mid$(q$, i + 1, len(q$) - 1): i = len(q$)
280 next i
290 if w$ = "" then v$ = q$
300 for i = 1 to v
310 if v$ = v$(i) then vb = i
320 next i
330 for i = 1 to w
340 if w$ = o$(i) then ob = i
350 next i
355 rem error messages override conditions
360 if w$ > "" and ob = 0 then m$ = "that's silly"
370 if vb = 0 then vb = v + 1
380 if w$ = "" then m$ = "i need two words"
390 if vb > v and ob > 0 then m$ = "you can't '" + q$ + "'"
400 if vb > v and ob = 0 then m$ = "you don't make sense!"
410 if vb < v and ob > 0 and c(ob) = 0 then m$ = "you don't have '" + w$
420 if f(26)=1 and rm=13 and fix(rnd(1)*4) <> 3 and vb <> 21 then goto 425
421 goto 430
425 m$ = "bats attacking!": goto 90
430 if rm = 44 and fix(rnd(1) * 3) = 1 and f(24) <> 1 then f(27) = 1
440 if f(0) = 1 then ll = ll - 1
450 if ll < 1 then f(0) = 0
455 rem branch to subroutines
456 if vb > 14 then goto 465
460 on vb gosub 500,570,640,640,640,640,640,640,640,980,980,1030,1070,1140
```

```
461 goto 470
465 on vb-14 gosub 1180,1220,1250,1300,1340,1380,1400,1430,1460,1490,1510,1590
470 if ll = 10 then m$ = "your candle is waning!"
480 if ll = 1 then m$ = "your candle is out!"
490 goto 90

495 rem verb 1
500 print "words i know:"
510 for i = 1 to v
520 print v$(i); ", ";
530 next i
540 m$ = "": print
550 gosub 1580
560 return

565 rem verb 2
570 print "you are carrying:"
580 for i = 1 to g
590 if c(i) = 1 then print o$(i); ", ";
600 next i
610 m$ = "": print
620 gosub 1580
630 return

635 rem verbs 3 to 9 inclusive
640 d = 0
650 if ob = 0 then d = vb - 3
660 if ob = 19 then d = 1
670 if ob = 20 then d = 2
680 if ob = 21 then d = 3
690 if ob = 22 then d = 4
700 if ob = 23 then d = 5
710 if ob = 24 then d = 6
720 if rm = 20 and d = 5 then d = 1
730 if rm = 20 and d = 6 then d = 3
740 if rm = 22 and d = 6 then d = 2
750 if rm = 22 and d = 5 then d = 3
760 if rm = 36 and d = 6 then d = 1
770 if rm = 36 and d = 5 then d = 2
780 if f(14)=1 then m$="crash! you fell out of the tree!":f(14)=0:return
790 if f(27)=1 and rm=52 then m$="ghosts will not let you move":return
800 if rm=45andc(1)=1andf(34)=0 then m$="a magical barrier to the west":return
810 if (rm=26 andf(0)=0) and (d=1 or d=4) then m$="you need a light":return
820 if rm = 54 and c(15) <> 1 then m$ = "you're stuck!": return
830 if c(15)=1 and not(rm=53 or rm=54 or rm=55 or rm=47) then goto 835
831 goto 840
835 m$="you can't carry a boat!": return
840 if (rm > 26 and rm < 30) and f(0) = 0 then m$="too dark to move":return
850 f(35) = 0: rl = len(r$(rm))
860 for i = 1 to rl
870 u$ = mid$(r$(rm), i, 1)
880 if (u$ = "n" and d = 1 and f(35) = 0) then rm = rm - 8: f(35) = 1
890 if (u$ = "s" and d = 2 and f(35) = 0) then rm = rm + 8: f(35) = 1
900 if (u$ = "w" and d = 3 and f(35) = 0) then rm = rm - 1: f(35) = 1
910 if (u$ = "e" and d = 4 and f(35) = 0) then rm = rm + 1: f(35) = 1
920 next i
930 m$ = "ok"
940 if f(35) = 0 then m$ = "can't go that way!"
950 if d < 1 then m$ = "go where?"
960 if rm = 41 and f(23) = 1 then goto 965
961 goto 970
965 r$ = "sw": m$ = "the door slams shut!": f(23) = 0
970 return

975 rem verbs 10 and 11
980 if ob > g then m$ = "i can't get " + w$: return
985 if l(ob) <> rm then m$ = "it isn't here"
990 if f(ob) <> 0 then m$ = "what " + w$ + "?"
1000 if c(ob) = 1 then m$ = "you already have it"
1010 if ob>0 and l(ob)=rm and f(ob)=0 then goto 1015
1011 goto 1020
1015 c(ob)=1:l(ob) = 65: m$ = "you have the " + w$
1020 return

1025 rem verb 12
1030 if rm = 43 and (ob = 28 or ob = 29) then goto 1035
1031 goto 1040
1035 f(17) = 0: m$ = "drawer open"
1040 if rm = 28 and ob = 25 then m$ = "it's locked"
1050 if rm = 38 and ob = 32 then m$ = "that's creepy!": f(2) = 0
1060 return

1065 rem verb 13
```

```
1070 if ob = 30 then f(18) = 0: m$ = "something here!"
1080 if ob = 31 then m$ = "that's disgusting!"
1090 if (ob = 28 or ob = 29) then m$ = "there's a drawer"
1100 if ob = 33 or ob = 5 then gosub 1140
1110 if rm = 43 and ob = 35 then m$ = "there's something beyond..."
1120 if ob = 32 then gosub 1030
1130 return

1135 rem verb 14
1140 if rm = 42 and ob = 33 then m$ = "they are demonic works"
1150 if (ob=3 or ob=36) and c(3)=1 and f(34)=0 then goto 1155
1151 goto 1160
1155 m$="use this word with care 'xzanfar'"
1160 if c(5) = 1 and ob = 5 then m$ = "the script is in an alien tongue"
1170 return

1175 rem verb 15
1180 m$ = "ok '" + w$ + "'"
1190 if c(3)=1 and ob=34 then goto 1195
1191 goto 1200
1195 m$="*magic occurs*": if rm<>45 then rm=fix(rnd(1)*64)
1200 if c(3) = 1 and ob = 34 and rm = 45 then f(34) = 1
1210 return

1215 rem verb 16
1220 if c(12) = 1 then m$ = "you made a hole"
1230 if c(12) = 1 and rm = 30 then goto 1235
1231 goto 1240
1235 m$ = "dug the bars out": d$(rm) = "hole in the wall": r$(rm) = "nse"
1240 return

1245 rem verb 17
1250 if c(14) <> 1 and rm = 7 then m$ = "this is no time to play games"
1260 if ob = 14 and c(14) = 1 then m$ = "you swung it"
1270 if ob = 13 and c(13) = 1 then m$ = "whoosh"
1280 if ob=13 and c(13)=1 and rm=43 then goto 1285
1281 goto 1290
1285 r$(rm)="wn":d$(rm)="study with a secret room":m$="you broke the thin wall"
1290 return

1295 rem verb 18
1300 if ob=14 and c(14)=1 then m$="it isn't attached to anything!"
1310 if ob=14 and c(14)<>1 and rm=7 and f(14)=0 then goto 1315
1311 goto 1320
1315 m$="you see thick forrest and cliff south": f(14)=1: return
1320 if ob=14 and c(14)<>1 and rm=7 and f(14)=1 then goto 1325
1321 goto 1330
1325 m$ = "going down!": f(14) = 0
1330 return

1335 rem verb 19
1340 if ob=17 and c(17)=1 and c(8)=0 then m$="it will burn your hands"
1350 if ob=17 and c(17)=1 and c(9)=0 then m$="nothing to light it with"
1360 if ob=17 and c(17)=1 and c(9)=1 and c(8)=1 then goto 1365
1361 goto 1370
1365 m$="it casts a flickering light": f(0)=1
1370 return

1375 rem verb 20
1380 if f(0) = 1 then f(0) = 0: m$ = "extinguished"
1390 return

1395 rem verb 21
1400 if ob = 16 and c(16) = 1 then m$ = "hissss"
1410 if ob=16 and c(16)=1 and f(26)=1 then f(26)=0: m$="pfft!  got them"
1420 return

1425 rem verb 22
1430 if ob = 10 and c(10) = 1 and c(11) = 1 then goto 1435
1431 goto 1440
1435 m$ = "switched on": f(24) = 1
1440 if f(27) = 1 and f(24) = 1 then 1445
1441 goto 1450
1445 m$ = "whizz - vacuumed the ghosts up!": f(27) = 0
1450 return

1455 rem verb 23
1460 if rm = 43 and (ob = 27 or ob = 28) then gosub 1030
1470 if rm=28 and ob=25 and f(25)=0 and c(18)=1 then goto 1475
1471 goto 1480
1475 f(25)=1: r$(rm)="sew": d$(rm)="huge open door": m$="the key turns"
1480 return
```

```
1485 rem verb 24
1490 if c(ob) = 1 then c(ob) = 0: l(ob) = rm: m$ = "done"
1500 return

1505 rem verb 25
1510 s = 0
1520 for i = 1 to g
1530 if c(i) = 1 then s = s + 1
1540 next i
1550 if s=17 and c(15)<>1 and rm<>57 then goto 1555
1551 goto 1560
1555 print "you have everything": print "return to the gate for final score"
1560 if s = 17 and rm = 57 then goto 1565
1561 goto 1570
1565 print "double score for reaching here!": s = s * 2
1570 print "your score = "; s: if s > 18 then goto 1575
1571 goto 1580
1575 print "well done! you have finished the game": end
1580 input "press return to continue"; q$
1590 return

1595 rem game initialisation routine
1600 rem dim r$(63), d$(63), o$(w), v$(v)
1610 rem dim c(w), l(g), f(w)
1620 data 46,38,35,50,13,18,28,42,10,25,26,4,2,7,47,60,43,32
1630 for i = 1 to g
1640 read l(i)
1650 next i
1660 data help,carrying?,go,n,s,w,e,u,d,get,take,open,examine,read,say
1665 data dig,swing,climb,light,unlight,spray,use,unlock,leave,score
1680 for i = 1 to v
1690 read v$(i)
1700 next i

1705 rem possible movements (associated with the location, same index)
1710 data se,we,we,swe,we,we,swe,ws
1720 data ns,se,we,nw,se,w,ne,nsw
1730 data ns,ns,se,we,nwud,se,wsud,ns
1740 data n,ns,nse,we,we,nsw,ns,ns
1750 data s,nse,nsw,s,nsud,n,n,ns
1760 data ne,nw,ne,w,nse,we,w,ns
1770 data se,nsw,e,we,nw,s,sw,nw
1780 data ne,nwe,we,we,we,nwe,nwe,w
1790 for i = 0 to 63
1800 read r$(i)
1810 next i

1815 rem description of the location
1820 data dark corner,overgrown garden,by large woodpile,yard by rubbish
1830 data weedpatch,forest,thick forest,blasted tree, corner of house
1840 data entrance to kitchen,kitchen and grimey cooker,scullery door
1845 data room with inches of dust,rear turret room,clearing by house,path
1860 data side of house,back of hallway,dark alcove,small dark room
1865 data bottom of spiral staircase,wide passage,slippery steps,clifftop
1880 data near crumbling wall,gloomy passage,pool of light
1881 data impressive vaulted hallway, hall by thick wooden door
1885 data trophy room,cellar with barred window,cliff path
1900 data cupboard with hanging coat,front hall,sitting room,secret room,closet
1905 data steep marble stairs,dining room,deep cellar with coffin,cliff path
1920 data front lobby,library of evil books,study with desk and hole in wall
1925 data weird cobwebby room,very cold chamber,spooky room,cliff path by marsh
1940 data rubble-strewn verandah,front porch,front tower,sloping corridor
1945 data upper gallery,marsh by wall,marsh,soggy path,by twisted railing
1960 data path through iron gate,by railings,beneath front tower
1965 data debris from crumbling facade,large fallen brickwork
1966 data rotting stone arch,crumbling clifftop
1980 for i = 0 to 63
1990 read d$(i)
2000 next i

2010 data painting,ring,magic spells,goblet,scroll,coins,statue,candlestick
2012 data matches,vacuum,batteries,shovel,axe,rope,boat,aerosol,candle,key
2014 data north,south,west,east,up,down
2016 data door,bats,ghosts,drawer,desk,coat,rubbish
2018 data coffin,books,xzanfar,wall,spells
2060 for i = 1 to w
2070 read o$(i)
2080 next i
2090 f(18)=1:f(17)=1:f(2)=1:f(26)=1:f(28)=1:f(23)=1:ll=60:rm=57:m$="ok"
2100 return
```

Visual Basic 6.0 code - adapted by F. Fiorentini

```
VERSION 5.00
Begin VB.Form About
   BorderStyle     =   3  'Fixed Dialog
   Caption         =   "About..."
   ClientHeight    =   2175
   ClientLeft      =   45
   ClientTop       =   390
   ClientWidth     =   4560
   LinkTopic       =   "Form2"
   MaxButton       =   0   'False
   MinButton       =   0   'False
   ScaleHeight     =   2175
   ScaleWidth      =   4560
   ShowInTaskbar   =   0   'False
   StartUpPosition =   1  'CenterOwner
   Begin VB.Image Image1
      Height          =   1000
      Left            =   2700
      Picture         =   "About.frx":0000
      Stretch         =   -1  'True
      Top             =   870
      Width           =   1680
   End
   Begin VB.Label Label1
      Caption         =   "Label1"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   0
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      ForeColor       =   &H00000080&
      Height          =   1935
      Left            =   200
      TabIndex        =   0
      Top             =   120
      Width           =   4095
   End
End
Attribute VB_Name = "About"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub Form_Load()
Label1.Caption = "HAUNTED HOUSE" & vbCrLf & "Also Known As: La Casa Encantada" & vbCrLf & "Publisher :
Usborne Publishing Ltd"
Label1.Caption = Label1.Caption & vbCrLf & "Release Year: 1983" & vbCrLf & "Original version: GW Basic"
& vbCrLf & vbCrLf & "Visual Basic version"
Label1.Caption = Label1.Caption & vbCrLf & "Francesco Fiorentini" & vbCrLf & "September 2020"
End Sub

VERSION 5.00
Begin VB.Form HauntedHouse
   BackColor       =   &H00E0E0E0&
   BorderStyle     =   3  'Fixed Dialog
   Caption         =   "HAUNTED HOUSE by Usborne Publishing Ltd 1983 - VB6 port by F. Fiorentini 2020"
   ClientHeight    =   5730
   ClientLeft      =   8295
   ClientTop       =   5670
   ClientWidth     =   10575
   ForeColor       =   &H00000040&
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   MinButton       =   0   'False
   ScaleHeight     =   5730
   ScaleWidth      =   10575
   ShowInTaskbar   =   0   'False
   Begin VB.CommandButton Command1
      Caption         =   "About"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   0
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   495
      Left            =   9600
      TabIndex        =   3
      Top             =   5160
      Width           =   855
   End
   Begin VB.CommandButton Command_Check
      Caption         =   "OK"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   0
         Weight          =   700
         Underline       =   0    'False
         Italic          =   0    'False
         Strikethrough   =   0    'False
      EndProperty
      Height          =   495
      Left            =   9000
      TabIndex        =   2
      Top             =   5160
      Width           =   495
   End
   Begin VB.TextBox InputString
      BeginProperty Font
         Name            =   "Verdana"
         Size            =   14.25
         Charset         =   0
         Weight          =   400
         Underline       =   0    'False
         Italic          =   0    'False
         Strikethrough   =   0    'False
      EndProperty
      Height          =   495
      Left            =   50
      TabIndex        =   1
      Top             =   5160
      Width           =   8895
   End
   Begin VB.TextBox Testo
      BackColor       =   &H00E0E0E0&
      BeginProperty Font
         Name            =   "Trebuchet MS"
         Size            =   12
         Charset         =   0
         Weight          =   700
         Underline       =   0    'False
         Italic          =   0    'False
         Strikethrough   =   0    'False
      EndProperty
      ForeColor       =   &H00000040&
      Height          =   5055
      Left            =   50
      Locked          =   -1  'True
      MultiLine       =   -1  'True
      ScrollBars      =   2  'Vertical
      TabIndex        =   0
      Text            =   "Haunted House.frx":0000
      Top             =   0
      Width           =   10455
   End
End
Attribute VB_Name = "HauntedHouse"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Dim V, W, G, I, VB, OB As Integer
Dim R, D, O, S
Dim Varray, RArray, DArray, OArray, LArray
Dim M$
Dim V_$
Dim W_$
Dim R_$
Dim U$
Dim Q$
Dim RM, LL
Dim FArray
Dim C(36)
Dim F(36)
Dim RL
Dim AddString As String

Private Sub Command_Check_Click()
Q$ = UCase(InputString)
M$ = ""

'---------- Controllo dell'azione
V_$ = "": W_$ = "": VB = 0: OB = 0
For I = 1 To Len(Q$)
    If Mid$(Q$, I, 1) = " " And V_$ = "" Then V_$ = Left$(Q$, I - 1)
    If Mid$(Q$, I + 1, 1) <> " " And V_$ <> "" Then W_$ = Mid$(Q$, I + 1, Len(Q$) - 1): I = Len(Q$)
Next I
If W_$ = "" Then V_$ = Q$
For I = 1 To V
    If V_$ = Varray(I) Then VB = I
Next I
For I = 1 To W
    If W_$ = OArray(I) Then OB = I
Next I

Rem ERROR MESSAGES OVERRIDE CONDITIONS
If W_$ > "" And OB = 0 Then M$ = "THAT'S SILLY"
If VB = 0 Then VB = V + 1
If W_$ = "" Then M$ = "I NEED TWO WORDS"
If VB > V And OB > 0 Then M$ = "YOU CAN'T '" + Q$ + "'"
If VB > V And OB = 0 Then M$ = "YOU DON'T MAKE SENSE!"
If VB < V And OB > 0 And C(OB) = 0 Then M$ = "YOU DON'T HAVE '" + W_$
If FArray(26) = 1 And RM = 13 And Fix(Rnd(1) * 4) <> 3 And VB <> 21 Then M$ = "BATS ATTACKING!": GoTo
Stampa
If RM = 44 And Fix(Rnd(1) * 3) = 1 And FArray(24) <> 1 Then FArray(27) = 1
If FArray(0) = 1 Then LL = LL - 1
If LL < 1 Then FArray(0) = 0

Rem BRANCH TO SUBROUTINES
On VB GoSub 500, 570, 640, 640, 640, 640, 640, 640, 640, 980, 980, 1030, 1070, 1140, 1180, 1220, 1250,
1300, 1340, 1380, 1400, 1430, 1460, 1490, 1510, 1590
If LL = 10 Then M$ = "YOUR CANDLE IS WANING!"
If LL = 1 Then M$ = "YOUR CANDLE IS OUT!"
GoTo Stampa

500:
Rem VERB 1
AddString = "WORDS I KNOW:" & vbCrLf
For I = 1 To V
AddString = AddString & Varray(I) & ", "
Next I
M$ = ""
AddString = AddString & vbCrLf
Return

570:
Rem VERB 2
AddString = "YOU ARE CARRYING:" & vbCrLf
For I = 1 To G
If C(I) = 1 Then AddString = AddString & OArray(I) & ", "
Next I
AddString = AddString & vbCrLf
M$ = ""
Return

640:
Rem VERBS 3 TO 9 INCLUSIVE
D = 0
If OB = 0 Then D = VB - 3
If OB = 19 Then D = 1
If OB = 20 Then D = 2
If OB = 21 Then D = 3
If OB = 22 Then D = 4
If OB = 23 Then D = 5
If OB = 24 Then D = 6
If RM = 20 And D = 5 Then D = 1
If RM = 20 And D = 6 Then D = 3
If RM = 22 And D = 6 Then D = 2
If RM = 22 And D = 5 Then D = 3
If RM = 36 And D = 6 Then D = 1
If RM = 36 And D = 5 Then D = 2
If FArray(14) = 1 Then M$ = "CRASH!  YOU FELL OUT OF THE TREE!": FArray(14) = 0: Return
If FArray(27) = 1 And RM = 52 Then M$ = "GHOSTS WILL NOT LET YOU MOVE": Return
If RM = 45 And C(1) = 1 And FArray(34) = 0 Then M$ = "A MAGICAL BARRIER TO THE WEST": Return
If (RM = 26 And FArray(0) = 0) And (D = 1 Or D = 4) Then M$ = "YOU NEED A LIGHT": Return
If RM = 54 And C(15) <> 1 Then M$ = "YOU'RE STUCK!": Return
If C(15) = 1 And Not (RM = 53 Or RM = 54 Or RM = 55 Or RM = 47) Then M$ = "YOU CAN'T CARRY A BOAT!":
```

```
Return
If (RM > 26 And RM < 30) And FArray(0) = 0 Then M$ = "TOO DARK TO MOVE": Return
FArray(35) = 0: RL = Len(RArray(RM))
For I = 1 To RL
U$ = Mid$(RArray(RM), I, 1)
If (U$ = "N" And D = 1 And FArray(35) = 0) Then RM = RM - 8: FArray(35) = 1
If (U$ = "S" And D = 2 And FArray(35) = 0) Then RM = RM + 8: FArray(35) = 1
If (U$ = "W" And D = 3 And FArray(35) = 0) Then RM = RM - 1: FArray(35) = 1
If (U$ = "E" And D = 4 And FArray(35) = 0) Then RM = RM + 1: FArray(35) = 1
Next I
M$ = "OK"
If FArray(35) = 0 Then M$ = "CAN'T GO THAT WAY!"
If D < 1 Then M$ = "GO WHERE?"
If RM = 41 And FArray(23) = 1 Then R_$ = "SW": M$ = "THE DOOR SLAMS SHUT!": FArray(23) = 0
Return

980:
Rem VERBS 10 AND 11
If OB > G Then M$ = "I CAN'T GET " + W_$: Return
If LArray(OB) <> RM Then M$ = "IT ISN'T HERE"
If FArray(OB) <> 0 Then M$ = "WHAT " + W_$ + "?"
If C(OB) = 1 Then M$ = "YOU ALREADY HAVE IT"
If OB > 0 And LArray(OB) = RM And FArray(OB) = 0 Then C(OB) = 1: LArray(OB) = 65: M$ = "YOU HAVE THE " +
W_$
Return

1030:
Rem VERB 12
If RM = 43 And (OB = 28 Or OB = 29) Then FArray(17) = 0: M$ = "DRAWER OPEN"
If RM = 28 And OB = 25 Then M$ = "IT'S LOCKED"
If RM = 38 And OB = 32 Then M$ = "THAT'S CREEPY!": FArray(2) = 0
Return

1070:
Rem VERB 13
If OB = 30 Then FArray(18) = 0: M$ = "SOMETHING HERE!"
If OB = 31 Then M$ = "THAT'S DISGUSTING!"
If (OB = 28 Or OB = 29) Then M$ = "THERE'S A DRAWER"
If OB = 33 Or OB = 5 Then GoSub 1140
If RM = 43 And OB = 35 Then M$ = "THERE'S SOMETHING BEYOND..."
If OB = 32 Then GoSub 1030
If OB > 0 And M$ = "" Then M$ = "NOTHING REALLY USEFUL FOUND..."
Return

1140:
Rem VERB 14
If RM = 42 And OB = 33 Then M$ = "THEY ARE DEMONIC WORKS"
If (OB = 3 Or OB = 36) And C(3) = 1 And FArray(34) = 0 Then M$ = "USE THIS WORD WITH CARE 'XZANFAR'"
If C(5) = 1 And OB = 5 Then M$ = "THE SCRIPT IS IN AN ALIEN TONGUE"
Return

1180:
Rem VERB 15
M$ = "OK '" + W_$ + "'"
If C(3) = 1 And OB = 34 Then M$ = "*MAGIC OCCURS*": If RM <> 45 Then RM = Fix(Rnd(1) * 64)
If C(3) = 1 And OB = 34 And RM = 45 Then FArray(34) = 1
Return

1220:
Rem VERB 16
If C(12) = 1 Then M$ = "YOU MADE A HOLE"
If C(12) = 1 And RM = 30 Then M$ = "DUG THE BARS OUT": DArray(RM) = "HOLE IN THE WALL": RArray(RM) =
"NSE"
Return

1250:
Rem VERB 17
If C(14) <> 1 And RM = 7 Then M$ = "THIS IS NO TIME TO PLAY GAMES"
If OB = 14 And C(14) = 1 Then M$ = "YOU SWUNG IT"
If OB = 13 And C(13) = 1 Then M$ = "WHOOSH"
If OB = 13 And C(13) = 1 And RM = 43 Then RArray(RM) = "WN": DArray(RM) = "STUDY WITH A SECRET ROOM": M$
= "YOU BROKE THE THIN WALL"
Return

1300:
Rem VERB 18
If OB = 14 And C(14) = 1 Then M$ = "IT ISN'T ATTACHED TO ANYTHING!"
If OB = 14 And C(14) <> 1 And RM = 7 And FArray(14) = 0 Then M$ = "YOU SEE THICK FORREST AND CLIFF
SOUTH": FArray(14) = 1: Return
If OB = 14 And C(14) <> 1 And RM = 7 And FArray(14) = 1 Then M$ = "GOING DOWN!": FArray(14) = 0
Return

1340:
```

```
Rem VERB 19
If OB = 17 And C(17) = 1 And C(8) = 0 Then M$ = "IT WILL BURN YOUR HANDS"
If OB = 17 And C(17) = 1 And C(9) = 0 Then M$ = "NOTHING TO LIGHT IT WITH"
If OB = 17 And C(17) = 1 And C(9) = 1 And C(8) = 1 Then M$ = "IT CASTS A FLICKERING LIGHT": FArray(0) =
1
Return

1380:
Rem VERB 20
If FArray(0) = 1 Then FArray(0) = 0: M$ = "EXTINGUISHED"
Return

1400:
Rem VERB 21
If OB = 16 And C(16) = 1 Then M$ = "HISSSS"
If OB = 16 And C(16) = 1 And FArray(26) = 1 Then FArray(26) = 0: M$ = "PFFT!  GOT THEM"
Return

1430:
Rem VERB 22
If OB = 10 And C(10) = 1 And C(11) = 1 Then M$ = "SWITCHED ON": FArray(24) = 1
If FArray(27) = 1 And FArray(24) = 1 Then M$ = "WHIZZ - VACUUMED THE GHOSTS UP!": FArray(27) = 0
Return

1460:
Rem VERB 23
If RM = 43 And (OB = 27 Or OB = 28) Then GoSub 1030
If RM = 28 And OB = 25 And FArray(25) = 0 And C(18) = 1 Then FArray(25) = 1: RArray(RM) = "SEW":
DArray(RM) = "HUGE OPEN DOOR": M$ = "THE KEY TURNS"
Return

1490:
Rem VERB 24
If C(OB) = 1 Then C(OB) = 0: LArray(OB) = RM: M$ = "DONE"
Return

1510:
Rem VERB 25
S = 0
For I = 1 To G
If C(I) = 1 Then S = S + 1
Next I
If S = 17 And C(15) <> 1 And RM <> 57 Then AddString = AddString & vbCrLf & "YOU HAVE EVERYTHING":
AddString = AddString & vbCrLf & "RETURN TO THE GATE FOR FINAL SCORE"
If S = 17 And RM = 57 Then AddString = AddString & vbCrLf & "DOUBLE SCORE FOR REACHING HERE!": S = S * 2
AddString = AddString & vbCrLf & "YOUR SCORE = " & S: If S > 18 Then AddString = AddString & vbCrLf &
"WELL DONE! YOU HAVE FINISHED THE GAME": End
Return

1590:
Return

Stampa:
'----------- Stampa risultato del comando -----------------
Testo.Text = "YOUR LOCATION IS:"
Testo.Text = Testo.Text & vbCrLf & DArray(RM)
Testo.Text = Testo.Text & vbCrLf & vbCrLf & "POSSIBLE EXITS:" & vbCrLf
For I = 1 To Len(RArray(RM))
    Testo.Text = Testo.Text & Mid$(RArray(RM), I, 1) & ","
Next I
Testo.Text = Testo.Text & vbCrLf
For I = 1 To G
    If LArray(I) = RM And FArray(I) = 0 Then Testo.Text = Testo.Text & "YOU CAN SEE " & OArray(I) & "
HERE"
Next I
Testo.Text = Testo.Text & vbCrLf & "========================="
If AddString <> "" Then
    Testo.Text = Testo.Text & vbCrLf & AddString
    AddString = ""
Else
    Testo.Text = Testo.Text & vbCrLf & M$
End If
Testo.Text = Testo.Text & vbCrLf & vbCrLf & "What is your next move?"
InputString.Text = ""
InputString.SetFocus
End Sub

Private Sub Command1_Click()
About.Show
End Sub

Private Sub Form_Load()
'Load parameters
```

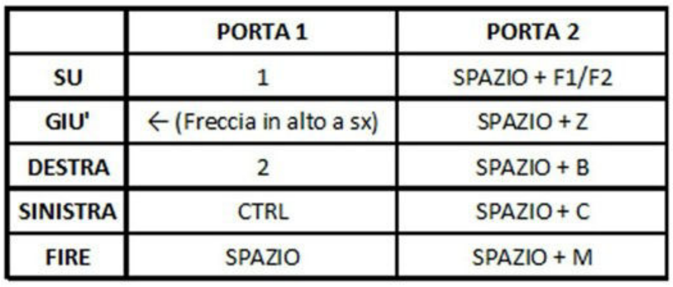
```
V = 25: W = 36: G = 18
LArray = Array(0, 46, 38, 35, 50, 13, 18, 28, 42, 10, 25, 26, 4, 2, 7, 47, 60, 43, 32)
Varray = Array("", "HELP", "CARRYING", "GO", "N", "S", "W", "E", "U", "D", "GET", "TAKE", "OPEN",
"EXAMINE", "READ", "SAY", "DIG", "SWING", "CLIMB", "LIGHT", "UNLIGHT", "SPRAY", "USE", "UNLOCK",
"LEAVE", "SCORE")
RArray = Array("SE", "WE", "WE", "SWE", "WE", "WE", "SWE", "WS", "NS", "SE", "WE", "NW", "SE", "W",
"NE", "NSW", "NS", "NS", "SE", "WE", "NWUD", "SE", "WSUD", "NS", "N", "NS", "NSE", "WE", "WE", "NSW",
"NS", "NS", "S", "NSE", "NSW", "S", "NSUD", "N", "N", "NS", "NE", "NW", "NE", "W", "NSE", "WE", "W",
"NS", "SE", "NSW", "E", "WE", "NW", "S", "SW", "NW", "NE", "NWE", "WE", "WE", "WE", "NWE", "NWE", "W")
DArray = Array("DARK CORNER", "OVERGROWN GARDEN", "BY LARGE WOODPILE", "YARD BY RUBBISH", "WEEDPATCH",
"FOREST", "THICK FOREST", "BLASTED TREE", "CORNER OF HOUSE", "ENTRANCE TO KITCHEN", "KITCHEN AND GRIMEY
COOKER", "SCULLERY DOOR", " ROOM WITH INCHES OF DUST", "REAR TURRET ROOM", "CLEARING BY HOUSE", "PATH",
"SIDE OF HOUSE", _
"BACK OF HALLWAY", "DARK ALCOVE", "SMALL DARK ROOM", "BOTTOM OF SPIRAL STAIRCASE", "WIDE PASSAGE",
"SLIPPERY STEPS", "CLIFFTOP", "NEAR CRUMBLING WALL", "GLOOMY PASSAGE", "POOL OF LIGHT", "IMPRESSIVE
VAULTED HALLWAY", "HALL BY THICK WOODEN DOOR", "TROPHY ROOM", "CELLAR WITH BARRED WINDOW", "CLIFF PATH",
" CUPBOARD WITH HANGING COAT", _
"FRONT HALL", "SITTING ROOM", "SECRET ROOM", "STEEP MARBLE STAIRS", "DINING ROOM", "DEEP CELLAR WITH
COFFIN", "CLIFF PATH", "CLOSET", "FRONT LOBBY", "LIBRARY OF EVIL BOOKS", "STUDY WITH DESK AND HOLE IN
WALL", "WEIRD COBWEBBY ROOM", "VERY COLD CHAMBER", "SPOOKY ROOM", "CLIFF PATH BY MARSH", "RUBBLE-STREWN
VERANDAH", "FRONT PORCH", _
"FRONT TOWER", "SLOPING CORRIDOR", "UPPER GALLERY", "MARSH BY WALL", "MARSH", "SOGGY PATH", "BY TWISTED
RAILING", "PATH THROUGH IRON GATE", "BY RAILINGS", "BENEATH FRONT TOWER", "DEBRIS FROM CRUMBLING
FACADE", "LARGE FALLEN BRICKWORK", "ROTTING STONE ARCH", "CRUMBLING CLIFFTOP")
OArray = Array("", "PAINTING", "RING", "MAGIC SPELLS", "GOBLET", "SCROLL", "COINS", "STATUE",
"CANDLESTICK", "MATCHES", "VACUUM", "BATTERIES", "SHOVEL", "AXE", "ROPE", "BOAT", "AEROSOL", "CANDLE",
"KEY", "NORTH", "SOUTH", "WEST", "EAST", "UP", "DOWN", "DOOR", "BATS", "GHOSTS", "DRAWER", "DESK",
"COAT", "RUBBISH", "COFFIN", "BOOKS", "XZANFAR", "WALL", "SPELLS")
FArray = Array(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0)
FArray(18) = 1: FArray(17) = 1: FArray(2) = 1: FArray(26) = 1: FArray(28) = 1: FArray(23) = 1: LL = 60:
RM = 57: M$ = "OK"

Testo.Text = "YOUR LOCATION IS:"
Testo.Text = Testo.Text & vbCrLf & DArray(RM)
Testo.Text = Testo.Text & vbCrLf & vbCrLf & "POSSIBLE EXITS:" & vbCrLf
For I = 1 To Len(RArray(RM))
    Testo.Text = Testo.Text & Mid$(RArray(RM), I, 1) & ","
Next I
Testo.Text = Testo.Text & vbCrLf
For I = 1 To G
If LArray(I) = RM And FArray(I) = 0 Then Testo.Text = Testo.Text & "YOU CAN SEE " & OArray(I) & " HERE"
Next I
Testo.Text = Testo.Text & vbCrLf & "========================="
Testo.Text = Testo.Text & vbCrLf & M$
Testo.Text = Testo.Text & vbCrLf & vbCrLf & "What is your next move?"
M$ = "WHAT"
InputString.Text = ""
End Sub

Private Sub InputString_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then
    Call Command_Check_Click
End If
End Sub
```

C64: How to disable the keys corresponding to joystick switches

by Attilio Capuozzo Founder of RetroProgramming Italia – RP Italia

In the 4th part of the Tutorial about how to develop a game entirely in BASIC V2, which you find on the group RetroProgramming Italia – RP Italia, Felice Nardella has inserted a very useful Table that, for convenience, we attach to this article (Fig. 1).

The Table shows the keys – or combination of keys – corresponding to the 5 switches of the Joystick connected to Control Port 1 or 2 of the C64.



	PORTA 1	PORTA 2
SU	1	SPAZIO + F1/F2
GIU'	← (Freccia in alto a sx)	SPAZIO + Z
DESTRA	2	SPAZIO + B
SINISTRA	CTRL	SPAZIO + C
FIRE	SPAZIO	SPAZIO + M

Figure 1

The 5 switches are the 4 directions of movement (North, South, East, West) and the Fire Button.

The purpose of this trick is to completely disable the use of these keys in programs that manage movements through the Joystick.
Let's start by saying that the first operation to be done is to disable the scanning of the keyboard, which usually takes place every 1/60 seconds.

Keyboard scanning, as well as cursor blinking management as well as updating the so-called Jiffy Clock, the precise C64 software clock per second (the Reserved Variables TI and TI$, of THE BASIC V2, refer to the Jiffy Clock), are managed through an Interrupt Routine.

To generate ("trigger") the Interrupt is a 16-bit counter, called Timer A, which constitutes one of the 5 Interrupt sources of one of the 2 CIAs – the CIA #1 – acronym for Complex Interface Adapter (6526), the specialized chips assigned to the interview with the I/O peripherals, that is, Input/Output.

CIA chip #1 is connected, via an IRQ type line, to Pin 3 of CPU 6510.

IRQ stands for Interrupt ReQuest and is one of the 2 types of Interrupt Hardware covered by the C64 architecture.
The other Interrupt Hardware is the NMI acronym for Non-Maskable Interrupt; CIA #2 is connected, via an NMI line, to Pin 4 of the 6510 microprocessor.

The 5 Interrupt sources of CIA # 1 can be detected via one of the 16 internal chip registers, the Interrupt Control Register (CIAICR) outlined in Fig. 2.

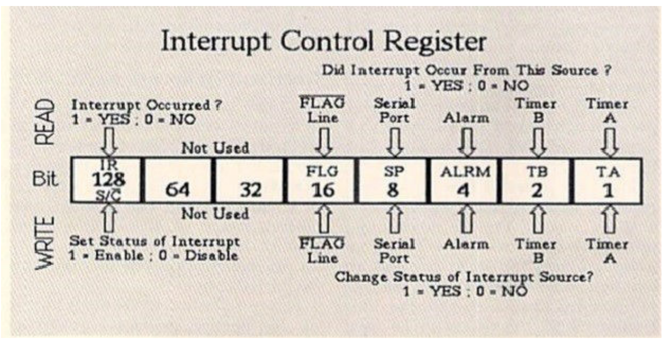


Figure 2

The Interrupt Control Register is mapped in memory to location 56333 ($DC0D).

To disable an Interrupt source, we will set the high bit of the Registry (bit number 7; bit value 128) to 0 (i.e. reset) and write a 1 in the bit corresponding to the Interrupt that we intend to disable.

In this specific case, our purpose is to disable the generation of the IRQ Interrupt by Timer A, which in Register 56333 is controlled by bit 0.
Then you must use the following statement:
poke 56333.1

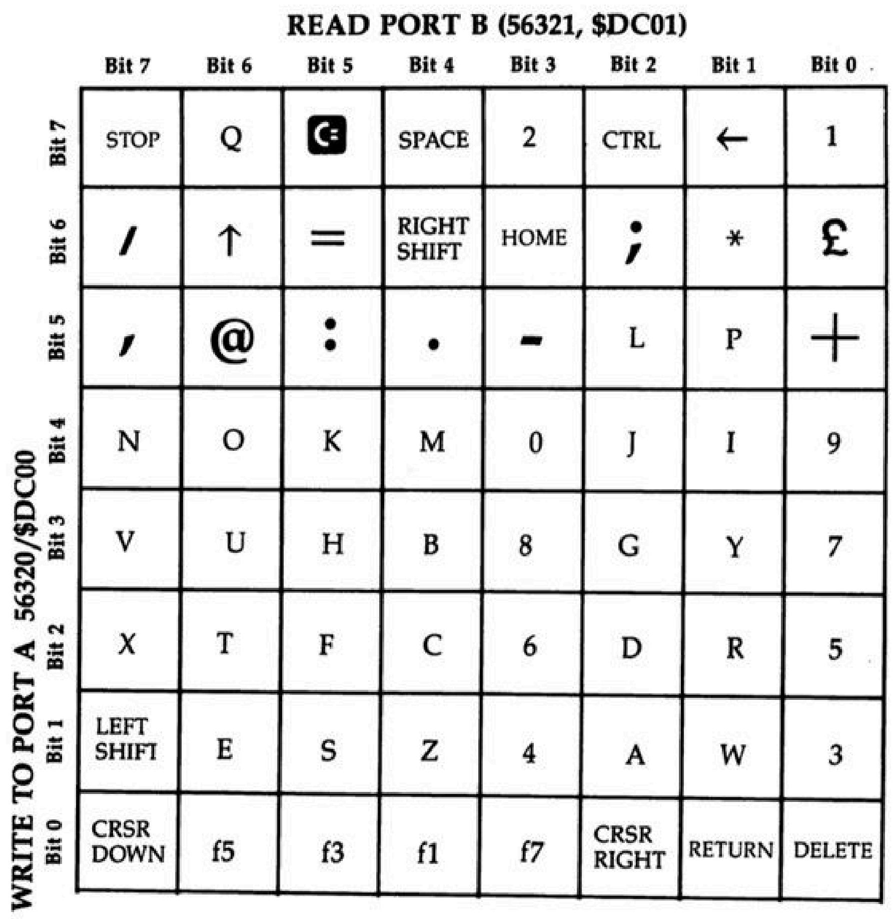
Let's briefly remember that Timer A counts down from a default start value – called LATCH VALUE – up to 0. When it reaches 0, if the corresponding bit in the aforementioned Registry is enabled, the Interrupt request will be generated.

By default CIA # 1, as mentioned, generates an IRQ Interrupt (via Timer A) 60 times per second.

Before leaving the program we will need to remember to re-enable Timer A (and ultimately re-enable keyboard scanning) by typing a 1 in CIAICR bit 7 and, again, a 1 in bit 0 that represents our Interrupt source: poke 56333,129

Changing the value contained in the Interrupt Control Register is not yet sufficient to achieve our purpose, namely to ensure that the user cannot use the combinations of keys alternative to the movements of the Joystick connected to one of the 2 Control Ports of the C64.
The 64 physical keys of the Commodore 64 are contained in an array of 8 rows for 8 columns, i.e. the Keyboard Matrix (See Fig. 3).

The Keyboard Matrix is connected to the CIA #1 via 2 of its 16 internal registers: Data Port Register A mapped to



READ PORT B (56321, $DC01)

WRITE TO PORT A 56320/$DC00

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Bit 7	STOP	Q	C=	SPACE	2	CTRL	←	1
Bit 6	/	↑	=	RIGHT SHIFT	HOME	;	*	£
Bit 5	,	@	:	.	-	L	P	+
Bit 4	N	O	K	M	0	J	I	9
Bit 3	V	U	H	B	8	G	Y	7
Bit 2	X	T	F	C	6	D	R	5
Bit 1	LEFT SHIFT	E	S	Z	4	A	W	3
Bit 0	CRSR DOWN	f5	f3	f1	f7	CRSR RIGHT	RETURN	DELETE

Figure 3

address 56320 ($DC00) and Data Port Register B corresponding to memory location 56321 ($DC01).

Actually, the physical keys of a C64 are 66 in all; the Keyboard Matrix lacks the SHIFT LOCK key that does not need to be detected as it is replaced by the normal left SHIFT, the LEFT SHIFT (whose pressure it simulates), and the RESET key whose pressure generates an NMI type Interrupt.

The 56320 Registry (Data Port A) is used for writing to select the Columns to be read from the Keyboard Matrix while the 56321 Registry (Data Port B), used for reading, is used to read the Matrix Rows for the Column previously selected through the Data Port Register A.

Just to complicate the life of us C64 RetroProgrammers, the same Registers store - in the first 5 bits - the values corresponding to the 5 switches of the Joystick as well described by the aforementioned Tutorial by Felice Nardella. So when we go to read the contents of one of the 2 aforementioned Registers we will have no way of knowing if the bits of the Registers have been modified by the Joystick movement (and/or by pressing the Fire Button) or by pressing 1 key (or a combination of keys) simulating the 5 switches of the Joystick.

In order to be sure to read, in one of the 2 Data Port Registers, a value that comes from the Joystick, we only have to write in advance in the 56320 Register (the Data Port Register used, as mentioned, in writing) a 1 in its 8 bits corresponding to the 8 Columns of the Keyboard Matrix. In fact, writing 1 in the bits of the 56320 Registry will ignore the reading of the corresponding Columns of the KeyBoard Matrix (the opposite effect is obtained by setting the bits of the Registry to 0).

In BASIC V2, therefore, we will type the following statement:
poke 56320,255

Ultimately, the 2 instructions that will be executed before the next readings of one of the 2 Control Ports to which the Joystick is connected (Register 56320 = Control Port 2 or Register 56321 = Control Port 1) are the following:
poke 56333.1:poke 56320.255

We have finally achieved our long-awaited goal!
That's all folks!

If you want to join **RetroProgramming Italia - RP Italia** group:
https://www.facebook.com/groups/retroprogramming/

An introduction to ARexx – part 3

by Gianluca Girelli

This article first appeared on Bitplane pages in May 2013.

MATRICES, CONDITIONAL INSTRUCTIONS AND PROCEDURES

After learning in the previous tutorial how to work with strings or, rather, how powerful the string management instructions are in ARexx, let's go into the details of procedures, matrices and, above all, conditional instructions.

PROCEDURES

Unlike other languages (e.g. Pascal), where the difference between functions and procedures is strictly and syntactically formal (as well as conceptual), ARexx handles subprograms very easily. From the beginning, in fact, both the author of the language and the community of developers preferred the simplicity of use and ease of reading to the formal complexity of structured languages.
Without wanting to go too far into the theory, we will simply remember that, while a procedure generally aims to perform actions/operations, a function "returns" a result. In other words: a procedure is usually invoked simply by writing its name, while a function (returning a result) is assigned to a variable.

For our purposes, it is sufficient to know that ARexx subprograms are invoked with the command "CALL subprogram_name (expression,...)".
The list of parameters in parentheses has no length limitations and is evaluated from left to right. If our subprogram is actually a function, then the last expression might be the variable that contains the final result. In this case, it might be more convenient (and conceptually correct) to use the following syntax: "result = subprogram_name (expression,...)".

As seen in the previous issue of the magazine, the subroutine is identified by an initial "label" that acts both as the name of the subprogram and as the logical reference address to be invoked by the main code and ends with the "return" statement:

```
; call example
pull phrase          ; reads the variable "phrase"
in input
call Parser (phrase)          ;call the subprogram
"Parser" passing the variable "phrase"

; subprogram structure
Parser:
statement 1                  ;these instructions
use the content of "phrase"
.....
instruction n
return
```

ENGINE (GAME)

The graphics engine is the software core of a video game or any other application with real-time graphics.
It provides basic technologies, simplifies development, and often allows the game to run on different platforms such as consoles or operating systems for personal computers. Basic functionality typically provided by a graphics engine includes a rendering engine ("renderer") for 2D and 3D graphics, a physical engine or collision detector, sound, scripting, animations, artificial intelligence, networking, and scene-graph.

Having said that, we will henceforth use the generic term 'subprogram' for simplicity, avoiding, if possible, the term 'procedure'. The reason is this: as we know, in ARexx the variables are all "global" and you do not need to declare a variable before using it. Nonetheless, the variables used within a subprogram are generally known only to the calling process. To change the scope of visibility of a variable, the reserved word "PROCEDURE" was then added to the language.
This statement can be used within a procedure to make certain variables "unknown" to the rest of the code.
“PROCEDURE”, however, can also be used, in conjunction with "EXPOSE", to modify this behavior. Before looking at an example, let's introduce another topic: matrices.

MATRICES

We have just mentioned that in ARexx neither the explicit declaration nor the initial assignment of values is necessary. However, we must remember that if the variable is not initialized, it will contain the character (s) that make up its symbol, transformed into uppercase letters. In other words, the value of the variable "amiga" will initially be "AMIGA". This type of variable is called "simple". A variable can also be a compound variable, such as "amiga.2000".

In this case, the beginning part is called "stem" and includes the variable name up to the dot (included), while the rest of the variable is called "node". An uninitialized compound variable contains the value of its node. In other words, if not initialized, "amiga.2000" will initially contain the value "2000".

One interesting thing about composite variables is that if the stem is initialized with a given value, all possible compound variables that begin with that stem will have the same value. That is, if to "amiga." I assign the "mythical" value, even the variables "amiga.500","amiga.1200","amiga.3000"e"amiga.4000" will contain "mythical. "

It is evident the enormous potential, even conceptual, that this type of initialization has compared to the traditional one. Assuming you want to initialize a vector of 100 elements to zero, just do:

```
vector. = 0
```

instead of having to use the classic:

```
for i=1 to 100
vector.i = 0
next i
```

Surprises don't end there, though. Since the "node" of a stem can also contain other dots (ex: a.b.c), the stem also represents the name of the multidimensional matrix associated with it. In this case, assuming you want to initialize a two-dimensional array of 100x100 elements to zero, the operation:

```
matrix. = 0
```

is equivalent to:

```
for x=1 to 100
for y=1 to 100
array.x.y=0
next y
next x
```

Does anyone still have doubts about the power of ARexx?

At this point, after talking about subprograms and simple and composite variables, it will be clear how the following code works:

```
/* this is the main program */
j=1; x.1= 'a'
call some_variables
say j k m       /* will display "1 7 M"  */
exit

/* this is a subprogram */
some_variables:  procedure exposed j k x.j
say j k x.j    /* will display "1 K a"          */
k=7; m=3       /* note that "m" is not exposed */
return
```

Please note the call to the subroutine "some_variables", the implicit initialization of simple and compound variables, the use of the stem as well as that of the instructions "PROCEDURE" and "EXPOSE".

Porting

Porting indicates in computer science a process of transposition, sometimes even with modifications, of a software component aimed at allowing its use in a different execution environment from the original one.

Two activities linked to but distinct from porting are emulation and cross-compilation.

CONDITIONAL INSTRUCTIONS AND ITERATIVE CYCLES

Let’s now close the present tutorial with a summary of conditional instructions (already introduced in the previous issue with some examples of syntax analyzers) and iterative cycles.
Like all the more advanced languages, ARexx implements constructs for iterating statements, but instead of having different types of statements, it uses the following syntax:

```
DO [repeater][condition];
[instruction list]
END
```

where:
- "repeater" can be: FOREVER, a single expression (to be evaluated) or a complex expression in the form "name=espr1 [TO espr2][BY espr3][FOR espr4]”;
- "condition" can be: [WHILE esprA] or [UNTIL esprB];
- and "instruction list" can be any sequence of language instructions.

With regard to conditional constructs, they are presented in the following forms:

```
IF  expression[;]  THEN[;]  statement  [ELSE[;]
statement]
```

or:

```
SELECT when_list [OTHERWISE[;][instruction_list]]
END;
```

where:
- "when_list" is composed of one or more constructs in the form "WHEN expression[;] THEN[;] instruction";
- and "instruction list" can be any sequence of language instructions.

In the code snippet below you can see the use of the just-mentioned constructs as used in the textual adventure "cyber.rexx", written for educational purposes by the author of this article. To introduce its scope, this is the subprogram that manages "navigation" within the game world, for the sake of simplicity implemented as a two-dimensional matrix of "X" rows and "Y" columns.

We have so clarified how the navigation works: we modifiy X,Y coordinates according to the desired destination. As an example, “go east" means moving one location to the right and it translates into the statement y=y+1 (while x remains untouched).

```
== = ====== = taken from cyber.rexx == ==============
Go:
if find(Situation.5 7.Pos,name)>0 then say ‘you
can't go '|| name
else do
select
when name='NORTH' then x=x-1
when name='SOUTH'  then x=x+1
when name='EAST'  then y=y+1
when name='WEST' then y=y-1
when name='UP' then x=x-1
when name='DOWN' then x=x+1
otherwise say ‘I don't understand. Try again’;
say; return
end
Pos=(x-1)*maxcol+y
do i=1 to 6
if Situation.i.Pos~='' then say Situation.i.Pos
end
end
return
==============================================
```

CONCLUSIONS

At the end of this first block of three tutorials, we should have achieved all the knowledge needed to write our first, truly complex program. In the near future, also thanks to the article published in issue 17 and related to "Game Coding", we will start building a game "engine" [BOX] for textual adventures. Although ARexx was born as a "scripting" program, the code will be written with the most modern programming paradigms in mind, in order to facilitate the eventual "port" [BOX] in other languages.

Further insights into the use and syntax of ARexx can be found in the texts quoted in the literature.

So don't miss the next round of articles: "Game Coding with ARexx".

Using the examples

As reported in previous articles, 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 simply,
`">rx name_script"`

BIBLIOGRAPHY

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

Happy Birthday Monkey Island!

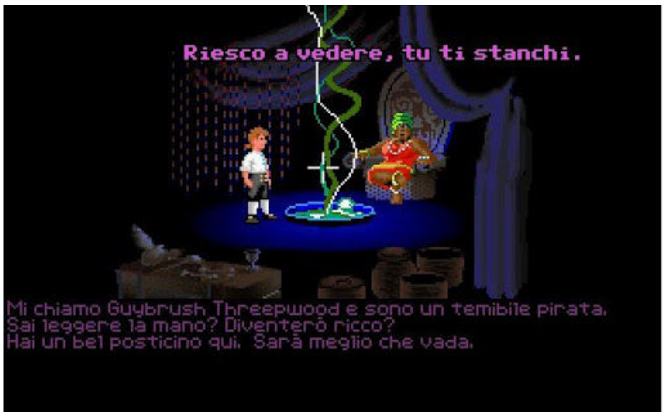
by Edoardo “Edward Scissorhands” Ullo

There are important anniversaries that cannot be overshadowed. Thirty years ago, it was October 15, 1990, **LucasFilm** published **The Secret of Monkey Island**, a point-and-click adventure by **Ron Gilbert** and other developers who would then leave a mark in the industry (see Tim Schafer and Dave Grossman).

The Secret of Monkey Island
TM & (c) 1990 LucasArts Entertainment Company.
All Rights Reserved.

This is a game that, in its four diskettes of the Amiga version but also in its Pc edition, has entered history for many reasons. Starting with the crazy protagonist, that **Guybrush Threepwood** who carelessly wanted to become a fearsome pirate but who – punctually – was mocked by everyone. Yelled in every way and made fun of both the old merchant and the guardian of the island of Melée (the fairytale island in the depths of the Caribbean that serves as the background to the first part of the game) who thanks to his falcon view like Mr. Magoo was able to immediately recognize the person in front of him apostrophe with a "Hello dandy”! Not to mention the music. Plenty of atmosphere and catchy to the very strange dialogues, puzzles, duels and various things.

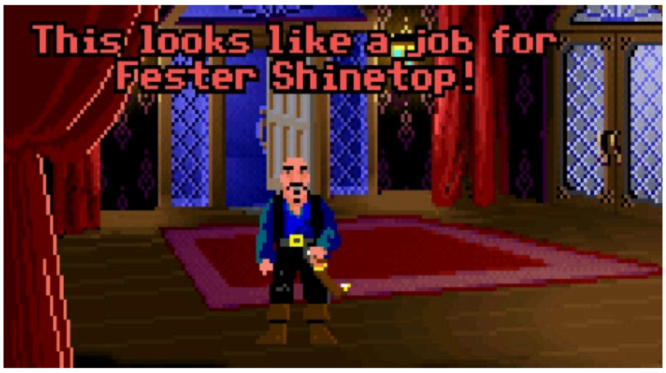
But, you may wonder, what does The Secret of Monkey Island have to do with a Halloween issue? We'll give you the answer immediately. Apart from the importance and



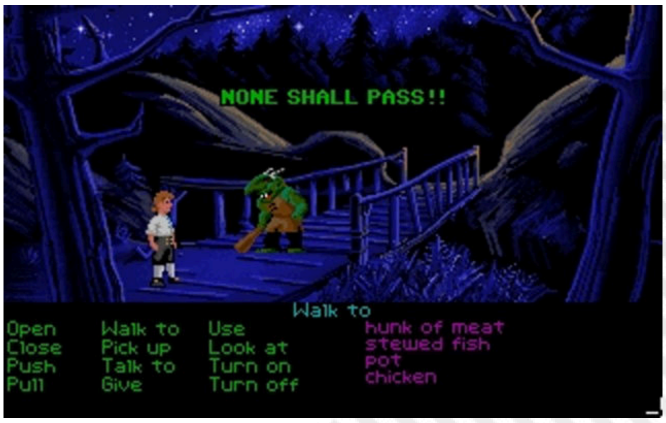
Mi chiamo Guybrush Threepwood e sono un temibile pirata.
Sai leggere la mano? Diventerò ricco?
Hai un bel posticino qui. Sarà meglio che vada.

weight of this absolute masterpiece, in fact, the game has some of its horror moments. Be it, of course, mitigated by a lot of humor: “No animal was harmed during the production of this game," one reads in a warning.

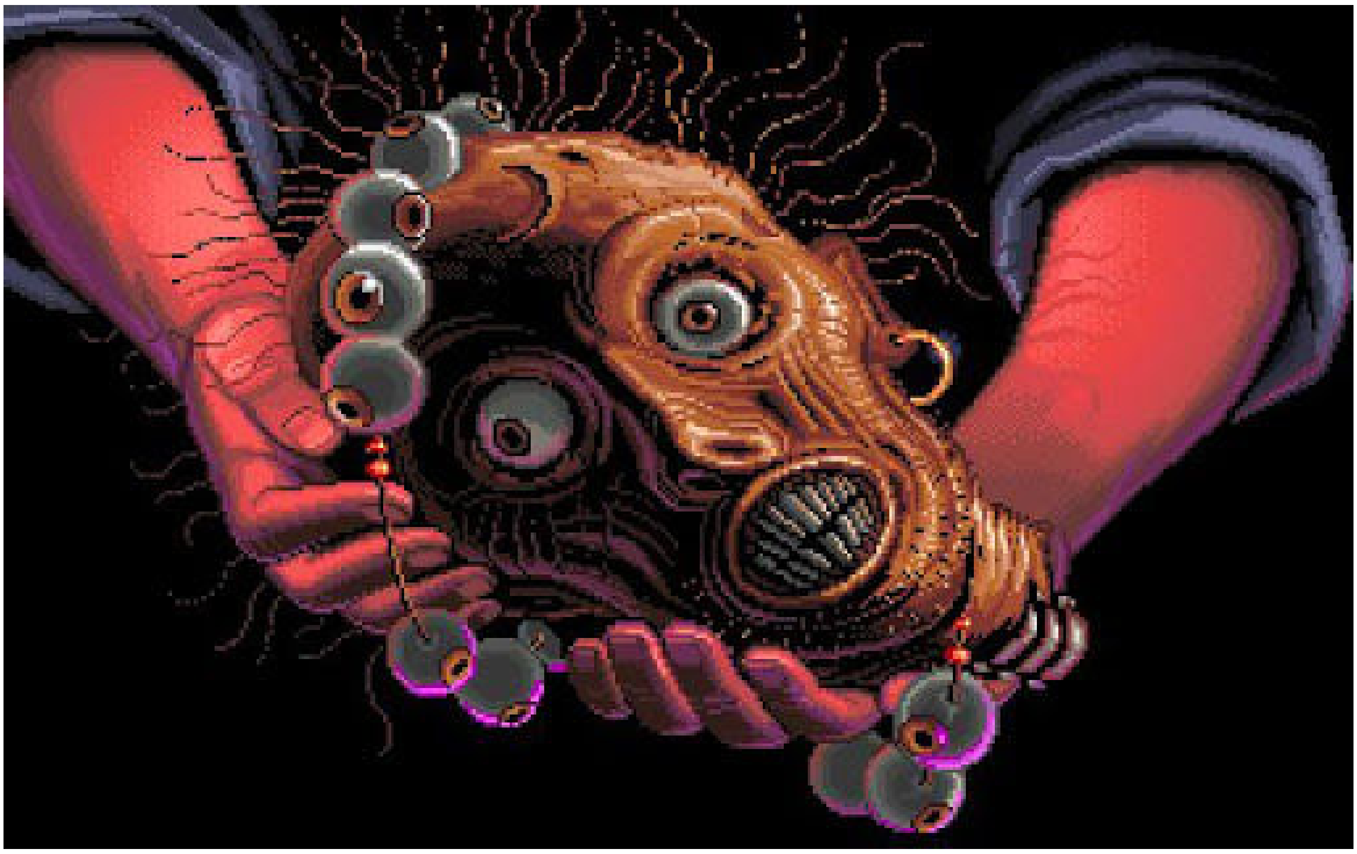
Remember the **splatter** scene where the antagonist of the series, **ghost pirate** LeChuck, dismembers the body of Sheriff Fester Shinetop whose looks he took to get rid of our “hero”? Well, if that's not horror. In addition, although in a comical and very light way, there is a lot of talk of voodoo. And that's how our blond hero manages to take the rampaging ship to Monkey Island despite the crew's very nice self-mutiny. We certainly do not explain how: the older ones will remember it, the younger ones will be able to play it and have fun.



There is also a **troll**... or something like that that hinders us and does not let us cross the bridge, except to keep us going after a "lavish” meal.



What about **cannibals**? They're usually to be feared. And of course: they are ruthless, fierce enemies, who eat other human beings. But in The Secret of Monkey Island we find them healthy and attentive to salt and cholesterol. Commendable. However, our Guybrush will also have to face a hellish labyrinth but to overcome it he will need a

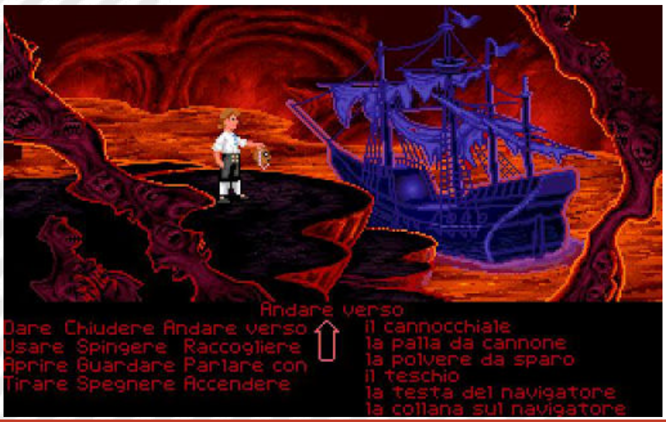


special compass: the **navigator's head**. It is the only one able to find its way in this Dante Alighieri's maze where mushrooms grow. This is truly a talking head kept alive by a special magic of Monkey Island cannibals.

What's next? Another potentially horrendous nuance is undoubtedly the one you experience in the passage where you arrive at the pirate ghost ship LeChuck to save Elaine Marley making in spite to a party crew. You have to become invisible to rip off these undead pirates so lucky to have met LeChuck alive that he killed them and enlisted them as undead.

In short, everything that traditionally should be on Halloween: ghosts, cannibals, a talking head, voodoo magic.

Ingredients that if mixed together would logically result in afirst-rate horror game or production. In The Secret of Monkey Island, on the other hand, we find them mixed with irony and carefree.


Andare verso
Dare Chiudere Andare verso
Usare Spingere Raccogliere
Aprire Guardare Parlare con
Tirare Spegnere Accendere
il cannocchiale
la palla da cannone
la polvere da sparo
il teschio
la testa del navigatore
la collana sul navigatore

What's the success due to? Super writing, fantastic, brilliant dialogues, written with both heart and head.

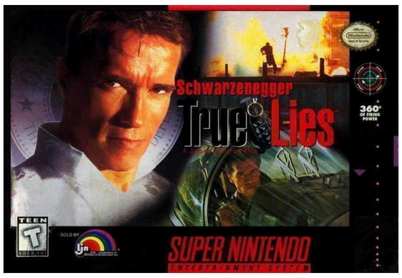
This is the prototype of the perfect point-and-click adventure, at least for productions from the early 1990s.

The sequel is equally spectacular and more long-lasting but perhaps has a little less charisma than its predecessor we celebrate in our Halloween issue.

TRUE LIES

Year: 1994
Publisher: Acclaim
Platform: Snes
Genere: Platform

In my last review, I talked about a revived passion for 16-bit consoles, so I want to present another title for Supernintendo discovered a little by chance: I'm talking about True Lies.



The game was released by Acclaim Entertainment in 1994, in conjunction with the release of the homonymous movie starring Arnold Schwarzenegger.

In fact, our hero Harry Tasker looks like the legendary American actor and our adventure follows, level by level, the plot of the movie.

From a graphic point of view, it is a top down game, with the classic isometric view from above that resembles both The Chaos Engine and Shadowrun.

At the beginning of each mission we are armed only with a gun but during our adventure we can also collect a pump rifle, a machine gun, a flamethrower, grenades and anti-personnel mines and we can easily switch from one weapon to another with a simple button.

In addition to weapons, we can also collect ammunition, keys to open hidden passageways and first aid boxes to recharge our energy.
With another key instead we can make a leap forward to escape the bullets or to find ourselves face to face with the enemy.

During our journey it is easy to encounter civilians who are often on our line of fire and who we absolutely must try not to hit. In fact, after three civilians were killed, we lost our lives.

The graphics are well cared for with colourful sprites and always new and varied settings. The many explosions are amazing as well.

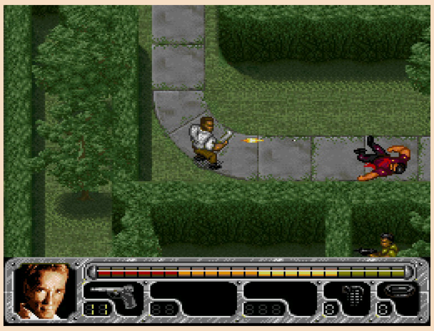
Levels are very long, deep and rich in detail and perhaps, being nine plus a bonus, in the long run they could be a bit repetitive.

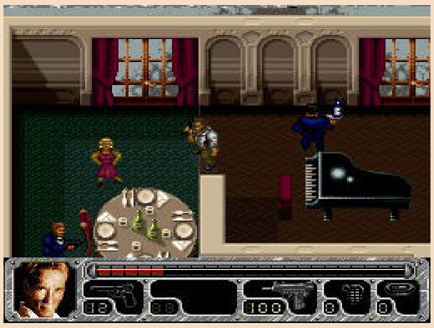
Nothing to say even on the sound compartment with listenable and different music for every level and effects always appropriate.

To enrich everything there are screenshots of the film, naturally pixelated, which introduce the theme of each mission.

In conclusion, if you do not know this title I strongly recommend you rediscover it. Moving our Schwarzy through bullets, explosions and various ambushes is really fun and we find ourselves immersed in a gaming experience that has not been replicated in other titles of this kind for the SuperNintendo.

by **Querino Ialongo**





OUR FINAL SCORE

» Gameplay 90%
Compared to the megadrive conversion, this one for super nintendo has controls that respond impeccably, offering a really fun gaming experience.

» Longevity 80%
It takes several hours to finish this title and maybe nine levels, plus a bonus, are a bit too much and they could result repetitive for some people.

BILLY MASTERS WAS RIGHT

Year: 2020
Developer: Paco Garcia
Genre: Point-and-click adventure
Platform: Windows

Nobody believes Billy Masters.

A 16-year-old American in the 80s and 90s accused of taking drugs in high school and slandering his teacher for serious crimes.
Tired of his outrageous attitude, his parents imposed the most complete curfew on him until he apologizes to his teacher.

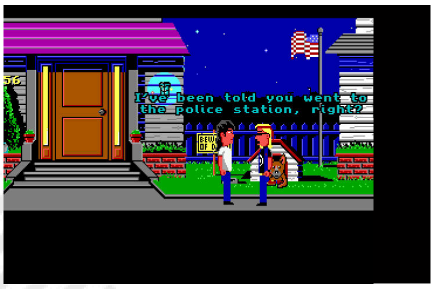
Problem is, his teacher is a real psychopath and he's up to something in the neighborhood.
Billy's in the corner, and he's gonna do everything he can to make his case... Until it's clear that Billy Masters was right.

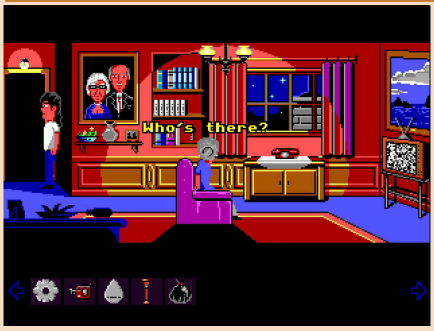
What a great game! Perhaps not a masterpiece, but a breath of fresh air (which then so fresh, it recalls the good old days) between the incredible fps, rpg and ultra-technological partners made of absurd framerate and 4k.
Are you really having fun with these numbers? Are you really interested in knowing if having fun is at 60 fps or less? All right, let's just drop it and get back to the game.
Billy Masters Was Right is a short adventure game in the Maniac Mansion graphic style and a plot inspired by films such as "The Burbs" or "Disturbia".
The atmosphere of the game is a mixture of nostalgia of the 80s, classic suspicion films of the same period and a teenage movie.
Classic point-and-click that recalls the system designed for Maniac Mansion by Lucas Film and I must say that it still works perfectly.
Funny, never trivial, well matched in puzzles... irreverent in dialogues and plot and in some places even disconcerting (a cartoon graphic with very strong themes, resounding!).
Nice technical realization that catapults us on a Commodore 64 of the times of Maniac Mansion, but fast and precise.
The strength for me is instead a soundtrack suitable for gaming opportunities, it really got me involved.
Unfortunately, it is neither long nor impossible, but it will take you a few hours to play.
Find it free on the developer's website:
https://postmoderdventures.itch.io/billy
or by giving a small amount to the programmer.

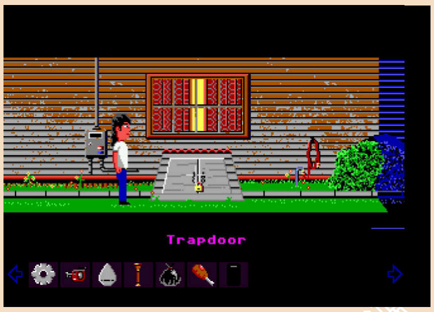
Available for Windows in English, Spanish, Catalan and German (locations are great).

This Halloween upload this game and remember that... Billy was right!

by **Carlo N. Del Mar Pirazzini**



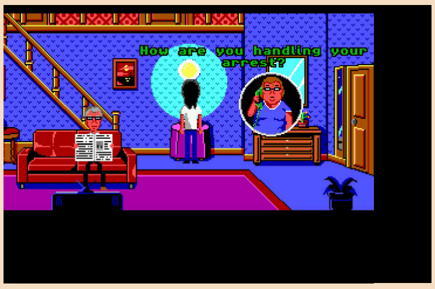




OUR FINAL SCORE

» Gameplay 80%
A well proven point and click game which reminds of Maniac Mansion.
Well structured plot, good puzzles. Beautiful.

» Longevity 60%
Not very long unfortunately, but fun. Passed with full marks.



PROJECT FIRESTART

Year: 1989
Developer: Dynamix
Publisher: Electronics Arts
Genre: Action/adventure/survival horror
Platform: Commodore 64

"Captain's log - February 13, 2061 - PROMETHEUS research spacecraft.

The mission funded by the Science System Foundation was a failure.
Experiments to create "super men" with extraordinary physical abilities have been a complete failure. How could I have agreed to be a part of this crap?
The passage of corporations in the name of power and money created monsters. We're all dying and... in the silence of these walls no one can hear us, but we hear them and their claws.

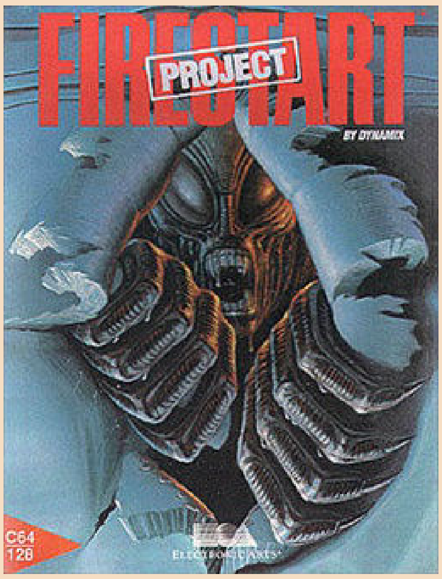
Here they are, they're coming in..."

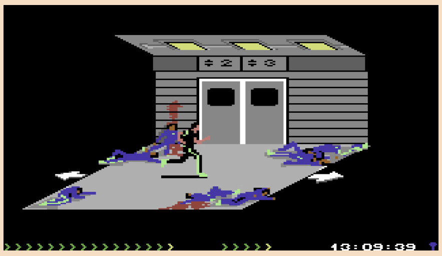
This could be the start of Project Firestart, set aboard the abandoned Prometheus and full of corpses. The lead agent, Jon, will move within the facility trying to figure out what happened through the logs, trying to save the only survivor and run away before detonating it.
An incredible plot for an incredible game, often forgotten but which really opened a genre years before Resident Evil or Silent Hill.
Project Firestart is an action game with pseudo 3D isometric graphics. The spacecraft consists of several rooms, fixed and sliding. Our hero can move freely on the ship and with the possibility of defending himself with a very effective laser gun but with a limited number of ammunition available.

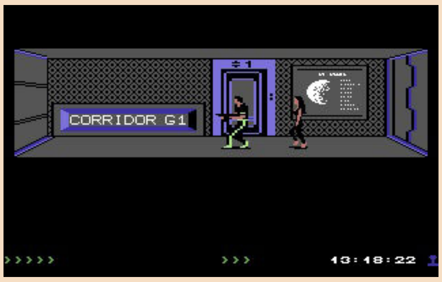
Some terminals that can be encountered during the game contain diaries and staff journals of the Prometheus, through which the player will learn what happened inside the spaceship. You can also find some objects that can recharge the character's energy or provide new ammunition or other objects useful for resolving the game.

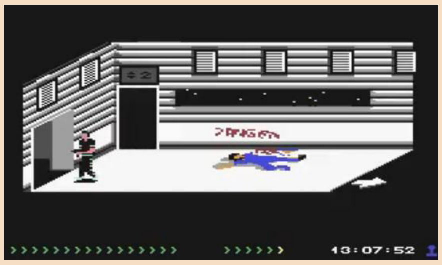
Occasionally, following a player action or at predetermined points, the game pauses to show the player a screen showing the protagonist, another character or one of the monsters, giving the player the opportunity to better see their appearance.

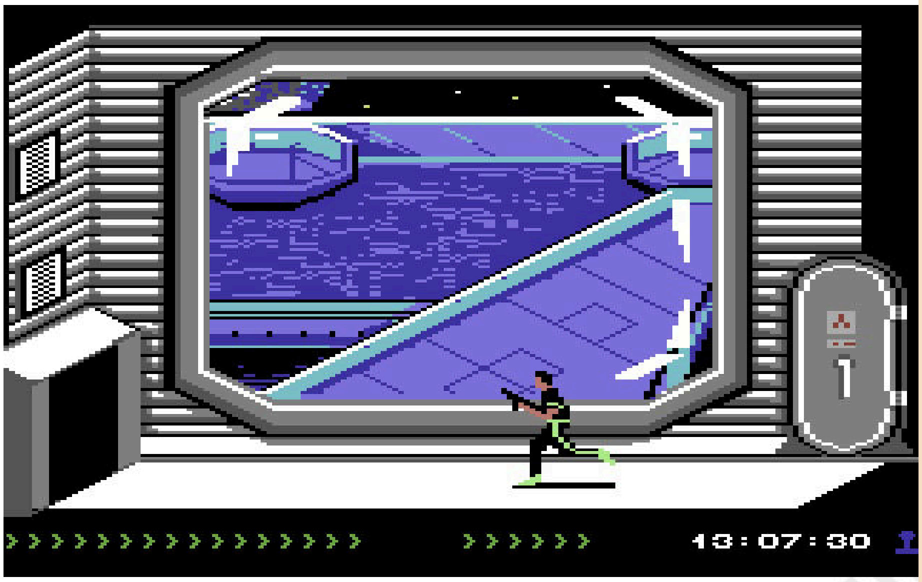
As we said, Project Firestart is an action video game created on Commodore 64.
It was designed by Jeff Tunnell and

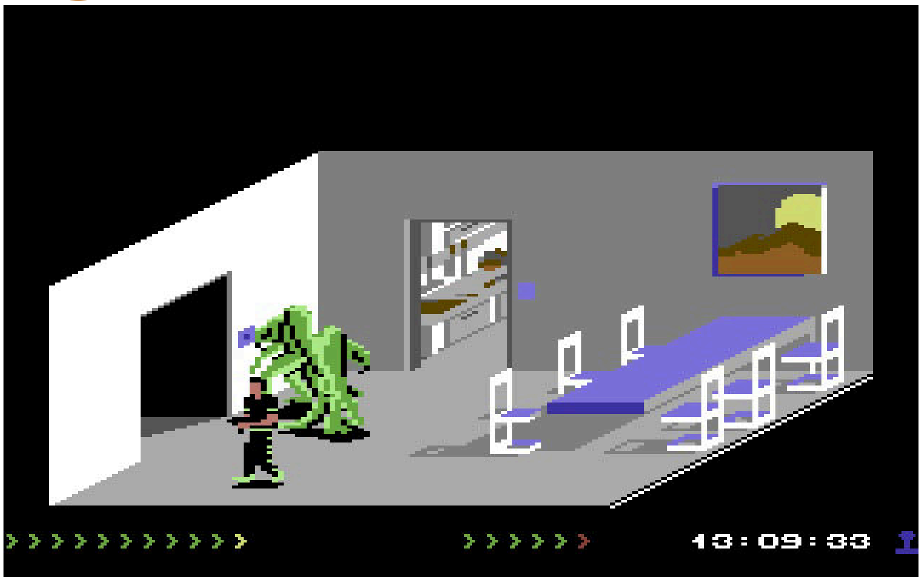












OUR FINAL SCORE

» Gameplay 85%
Difficult and not suitable for classic arcade "purists". Once inside the game you will not be able to stop easily.

» Longevity 85%
The Prometheus will not be released in a moment. Many hours of play.

Damon Slye and published by Electronic Arts in 1989.

To all intents and purposes it is to be considered the first example of survival horror with multiple endings, conceived ten years before the most famous games of the genre.

The gloomy atmosphere, the feeling that the enemy is hiding around every corner and that sense of powerlessness when you're out of ammunition made it fast at the time and gave it a halo of mystical mystery. The magazines praised him for the concept and for the game system, the players were astonished at the realization but then complained about the anomalous difficulty and a totally different approach.

Graphically stunning, varied and made with this isometric graphic that gave the player ample breath, something unprecedented in other titles. A title that reminded us of the atmospheres of horror films and skis of the time (Alien above all, but above all the thing). Players of the time were not accustomed to alien enemies so disgusting and leathery compared to the standard.

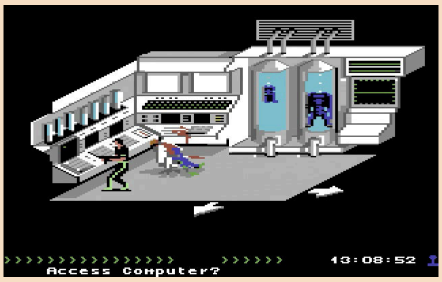
Difficult, violent, distressing... Beautiful.

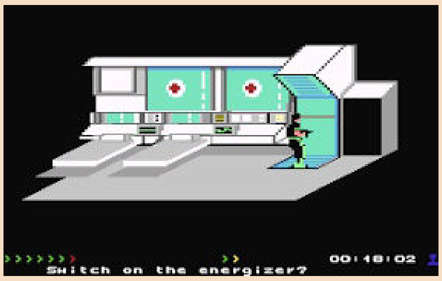
A real killer application for Commodore 64. Never been out for other systems, envied by owners of more powerful machines.

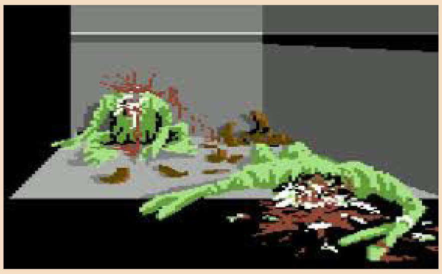
Try it locked in your room... With only the C64 on, your joystick and the silence of the space station where...

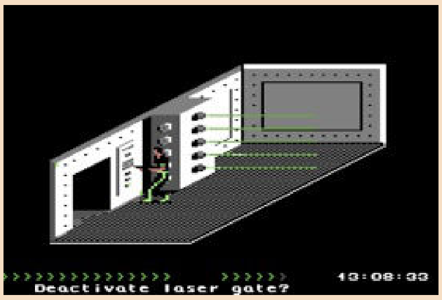
No one will hear you scream.

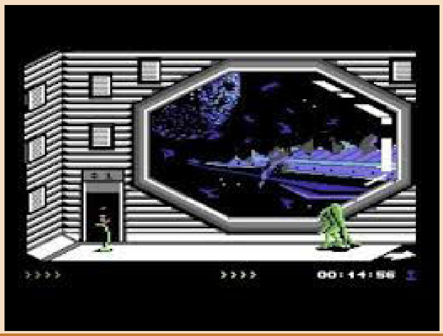
by **Carlo N. Del Mar Pirazzini**

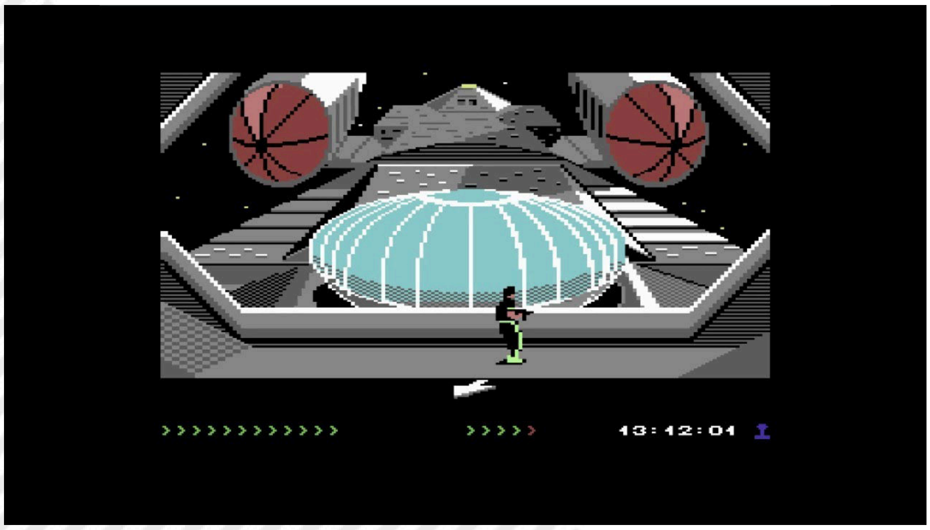












LUIGI'S MANSION

Year: 2001
Publisher: Nintendo
Developer: Nintendo
Genre: Survival Horror/Action
Platform: Game Cube

"Mariooooooooo..... Mario! "
How many times by pressing the A button of the Cube did we make Luigi repeat this sentence? Me very often. Luigi's Mansion is a title conceived by "Mamma" Nintendo that gave life to a saga very often underestimated, but with an appeal, a style and an excellent gameplay. This first chapter was released in 2001 (2002 in Italy) as the first title for the newborn Game Cube.

It's part of the big series of titles devoted to Mario and it had a remake on DS and a beautiful third episode on Switch.

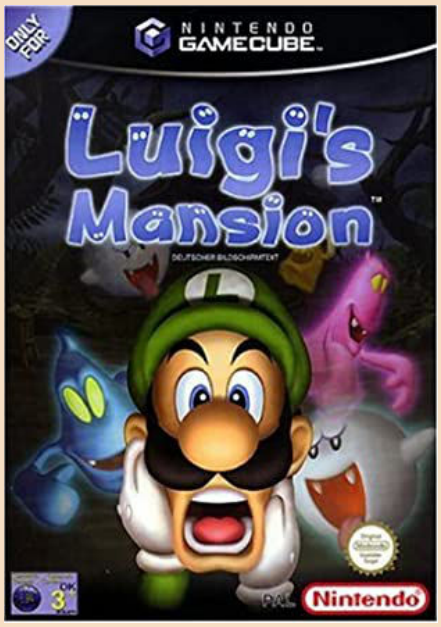
Upon its release it was criticized by magazines and game experts, who superficially labeled the title "childish" and simple. But everything improves over time (wine, for example, ndN). Let's go with order.

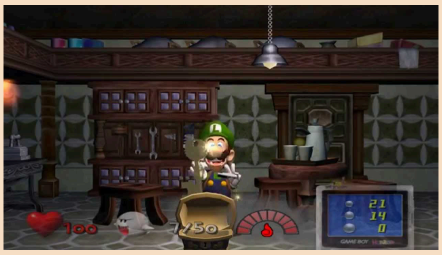
The game is fundamental to the history of the brand, as not only valorizes a character often in the background compared to Mario; Luigi who becomes an unexpected protagonist. This game basically does two things: detaches itself from the concept of the series of a classic platform and above all it mimics the numerous survival horrors that prevailed in that period.
An experiment never attempted before by Nintendo and that was not understood by the magazines. This is something that should be considered innovative and incredible!

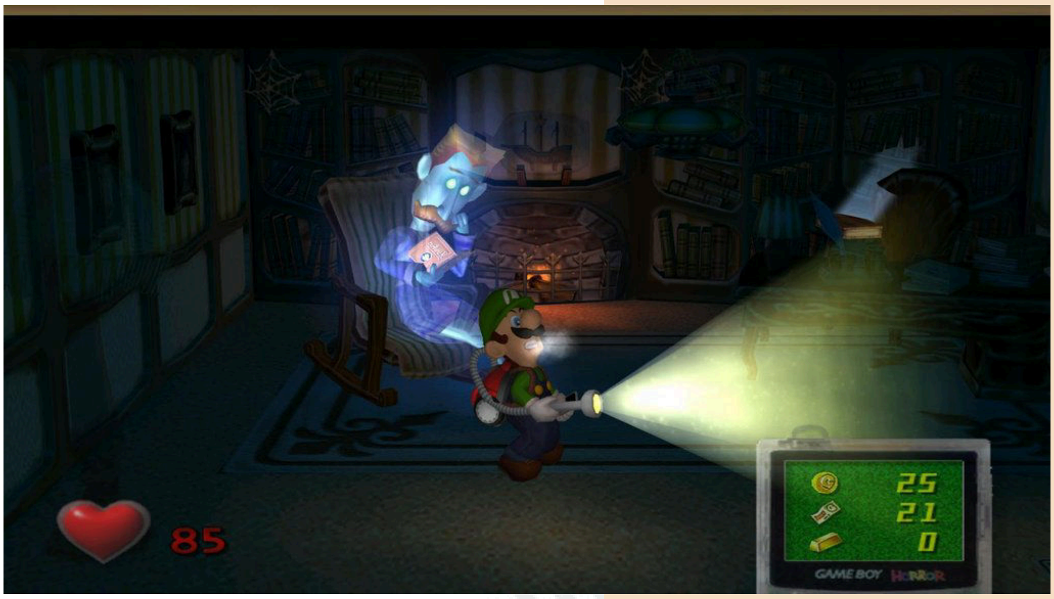
Few people know, however, that Luigi was not the first time he put himself in the role of the protagonist. Ten years earlier, he starred in "Mario is Missing!"

The story is simple: Mario was kidnapped by the Boo, the popular ghosts of the series. For this characteristic, in the wake of the Ghostbusters' films, Luigi will find himself facing them with a vacuum cleaner and a Game Boy Horror with the aim of exterminating the more than 50 rooms of the haunted palace to free up his brother after having obviously defeated the evil King Boo, the final boss. Each room contains secrets, traps or moments of fear (up to a certain point, ndN).

It is in the gameplay that this first episode presented itself as various and original, with ideas that still make it incredibly modern and never boring. The same formula proved to be successful and remained almost identical in the new chapters. This is an incredible strength. The use of the wonderful Nintendo cube joypad is masterful. The controls are very comfortable and complete with the balanced use of all the buttons and







the analogue stick.

As in the legendary Ghostbusters movie, Luigi, rather than fighting or jumping on his opponents, sucks them using the Poltergust 3000, which sucks up all sorts of ghosts, from simple generic ghosts to those in "frames" that haunt the house up to the most complicated ghosts and the very fast Boo.

The use of the vacuum cleaner is not limited to simple vacuuming, but you can use the Poltergust to move objects, suck in curtains and blankets, vacuum coins, bonus objects and keys (essential for opening rooms to explore).

The real strength of this chapter certainly remains the use of the objects and elements that serve our protagonist to advance in the game and higher in the villa. Starting with the simple vacuum cleaner it can be leveraged with extraordinary powers up to the Game Boy Horror, a device that allows us to observe the surrounding environment and receive information on the objects we encounter on our way; but also acts as a map of the entire building, helping the player by indicating the locked rooms, those already completed and those accessible (because we have perhaps found a special key to access them).

This particular object can also indicate the presence of ghosts and even have the function of inventory, indicating the amount of money, gems, spectra that we possess.

Critics, as we said, criticized the game by branding it as a beautiful title but "short and for kids".
Certainly the missions will not be billions and it will certainly not take thousands of hours to finish it, but it is a beautiful, serene and relaxing game (despite the ghosts).

Beautiful to play in front of TV in the dark right on Halloween, maybe with your son, a scaring ghost, and a frightened face of Luigi laughing all together at home. This makes it special and, finally, the critics have reassessed it.

If it were an alcoholic it would be a good brandy to enjoy calmly by the fire, but it is a video game and it is well made, playable and very beautiful to watch.
A perfect example of how Nintendo is "different."

If you have a Game Cube, I can't not

recommend it, but it also runs in emulation with the Dolphin.

You can find it in the remake version on DS and on Nintendo stores.

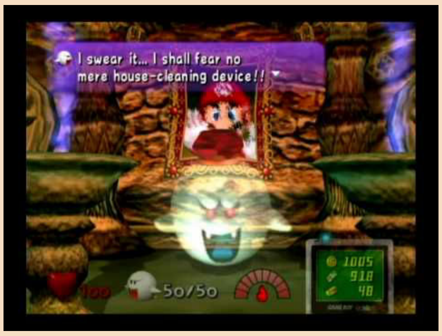
I mean, **play it**!

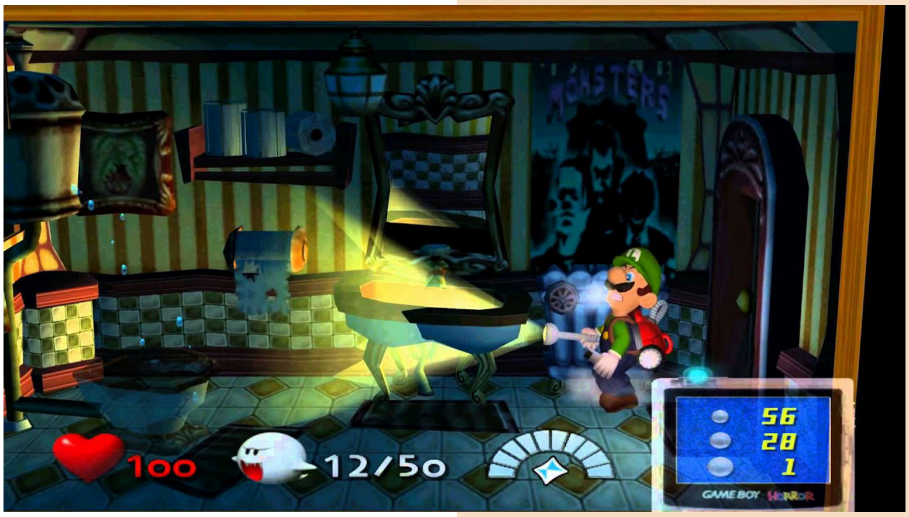
by **Carlo N. Del Mar Pirazzini**

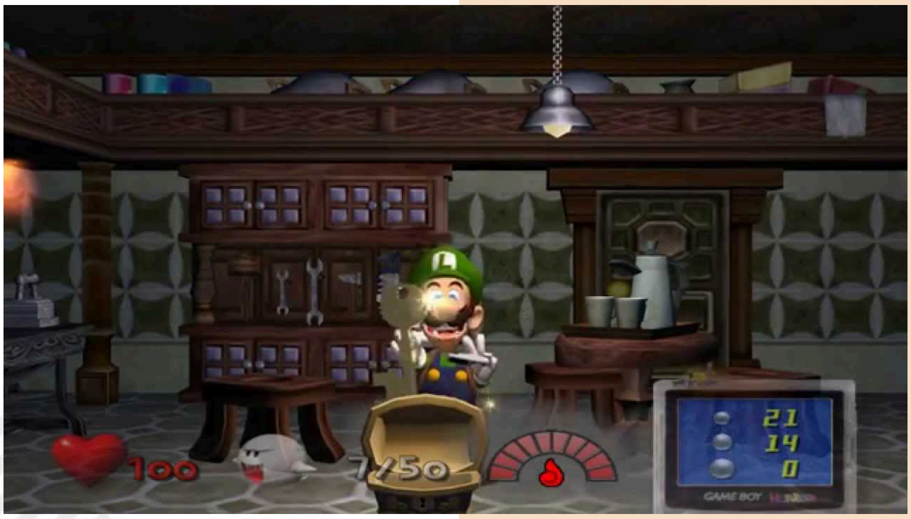
OUR FINAL SCORE

» Gameplay 95%
It is a well designed and playable product. The gameplay is well calibrated, the story fast and captivating and you will immediately feel immersed in the mission in search of Mario. Excellent the use of the joypad.

» Longevity 80%
It is not an blockbuster, but it is a title that will keep you busy each time you take it off the shelf and put it in the Game Cube.

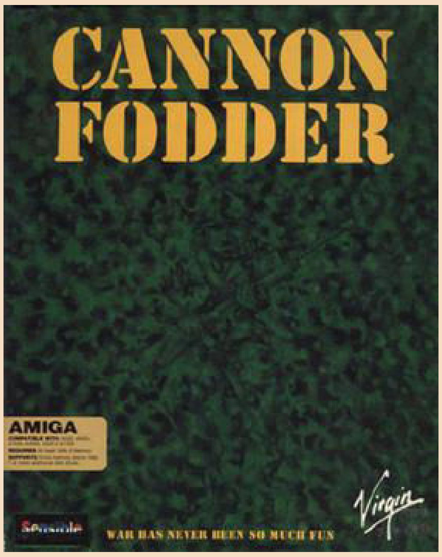






CANNON FODDER

Year: 1993
Publisher: Sensible Software/ Virgin
Developer: Sensible Software
Genre: Shoot'em-up/Real-time strategy
Platform: Amiga

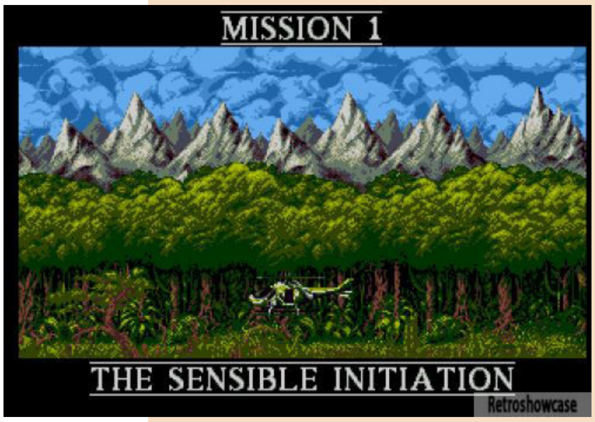


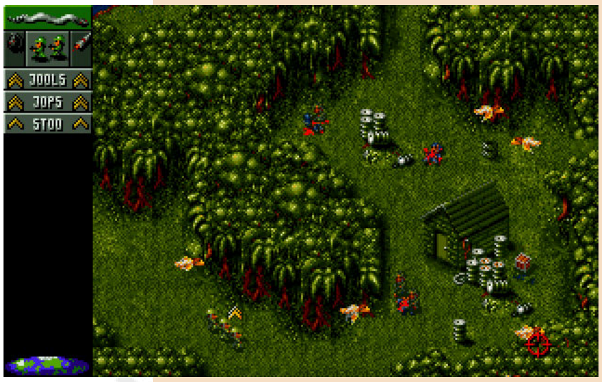
Those who owned a Commodore Amiga in 1993 could not help but add to their video game collection this small masterpiece created by those geniuses who had given Europe (and the world) the definitive football game: Sensible Soccer.

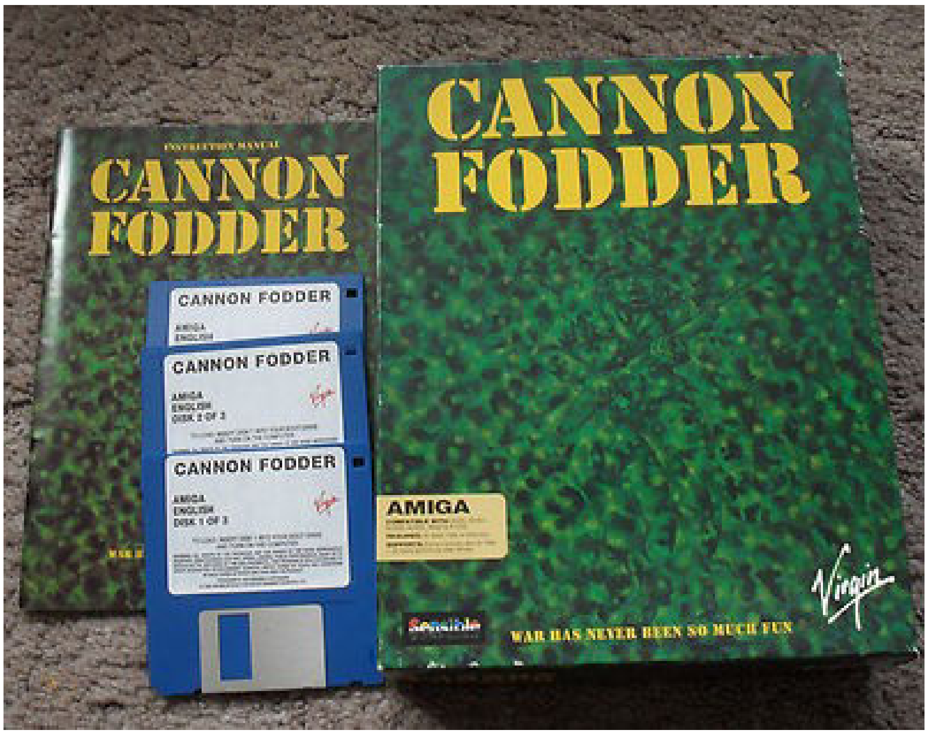
The English coders had another fantastic idea: to recruit thousands of small men in delicate pixel-art (as indeed were the players of Sensible Soccer) and place them in equally delicious scenarios designed with a view from above to make them become as the title of the game says “cannon fodder” or a goliardic and direct way to make us try directly on our monitors that the war had never really been so fun "War has never been so much fun".

Playability at the top within series of missions of increasing difficulty where obviously the purpose was in most cases to eliminate the enemy's presence on the territory (including bases and military installations) while trying to lose as few soldiers as possible (the survivors were increased from mission to mission) brilliantly using Commodore Amiga's trusty mouse as the only control system.

The game was not really a real-time strategy (genre that was born in those years due to Dune 2 on Amiga) but a lot of quick strategy behind our arcade actions would still make the difference between the life and death of our







little digital soldiers.

A masterpiece of playability (and game design) that contemplated the use not only of standard firearms but also rockets, tanks and everything that could have been useful to get to the promotion of our troops avoiding making them become small crosses scattered on the hill where new resources were recruited between one level and another.

A masterpiece of game design that pushed us to exploit the territory in our favor.
In the jungle they could camouflage us among the trees (but enemy soldiers could also do it) while in the Arctic areas we had to pay attention to the ice without counting the mines, the traps hidden in the vegetation, and dozens of other classy touches (such as the possibility of injuring only the enemies who then writhed and screamed from pain waiting to be finished).

That made the world of Cannon Fodder an immersive and exciting experience by treating a raw theme like war in a detached and fun way as it was actually announced in the initial theme song composed and sung by game designer Jon Hare or the song "War has never been so much fun", the icing on the cake for a product really worthy of being replayed again today.

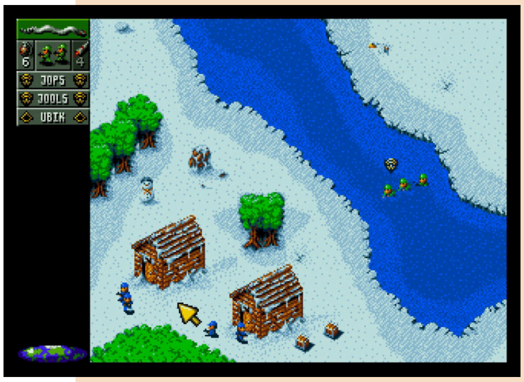
by **Flavio Soldani**

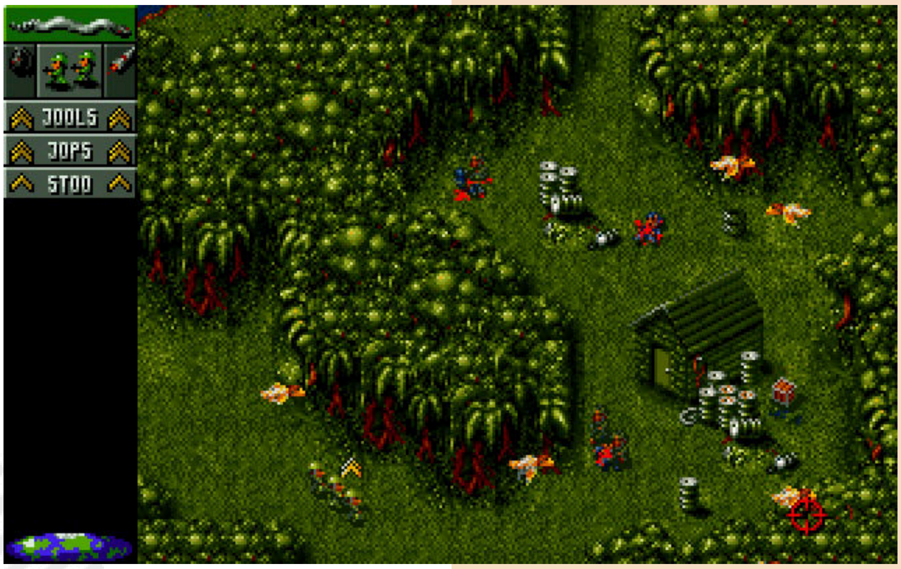
OUR FINAL SCORE

» Gameplay 96%
A successful mix of real time strategy and shooter, with so many classy touches and a playability so high that can hand down win the challenge with time.

» Longevity 95%
As said above Cannon Fodder is a brilliant video game that, along with many others, has helped to fix in our hearts the creativity that permeated the early '90s by making us love the Commodore Amiga systems.

The game was then converted on most of the platforms of the time, had a sequel called Cannon Fodder 2 and a 3D remake not made by Sensible Software in 2011 called Cannon Fodder 3. Unfortunately the sequels were not able to reach the quality of the first chapter.





MICRO MAGES

Year: 2019
Developer: Morphcat Games
Genre: Platform
Platform: Nintendo NES

"Do not be a magician - be magic!"
Leonard Cohen

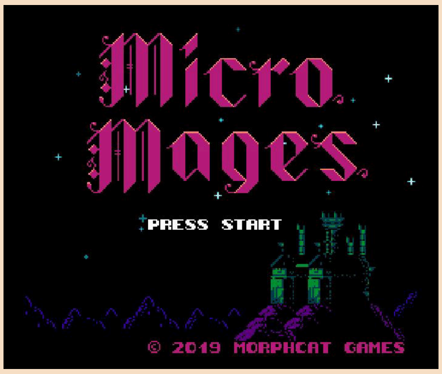
Micro Mages for the Nintendo Entertainment System, developed by Morphat Games, is a vertical sliding platform game that sees us as a magician just arrived in an ancient demonic fortress to save the princess from the forces of evil.

The journey inside the fortress will place our hero in pink tunic in front of great treasures, horrible creatures and deadly traps. Only thanks to his magical skills and his climbing skills the little magician can reach, with our help, the top of the tower.

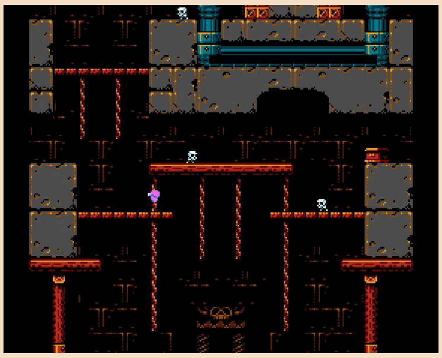
There are 4 towers to pass:
The Haunted Dungeon - an old lost tower full of bats and skeletons and where a huge ghost screams at its victims to kill them.
Valhalla Tower - where the Goblins are building a structure that leads to Valhalla.
The Jungle Temple - built inside a dormant volcano and ruled by the Prince of Darkness... and finally
Last Tower - The last tower is the somewhat mysterious headquarters of the supreme evil that trapped the princess.

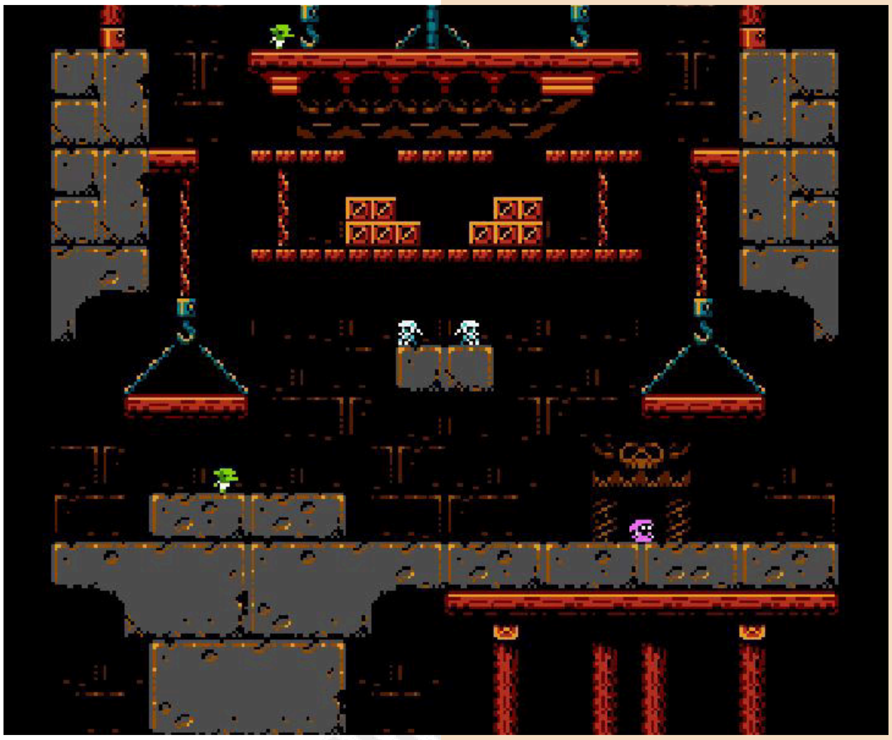
The main mechanic of the game is to cast spells against enemies with the A key of the joypad and use the jump (on the B key) to climb up to the top of the various levels. The jumping action on the wall is easy enough to be performed thanks to responsive controls and a perfect control of the main character in every situation.

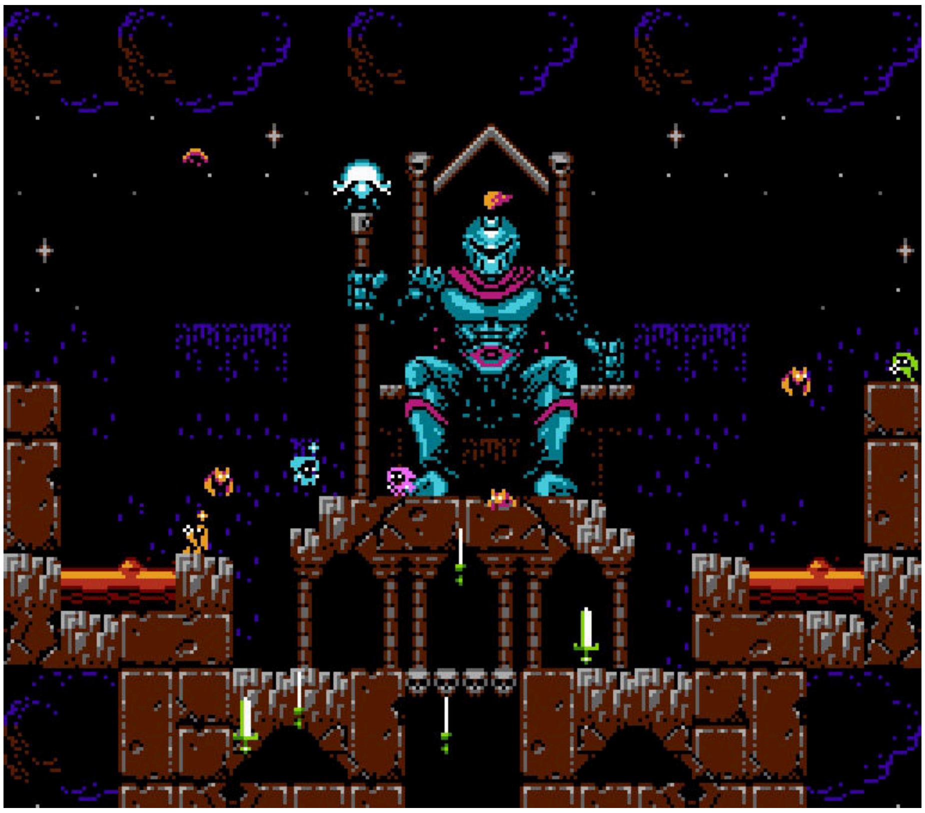
The level of challenge is quite various. During the journey we will meet several opponents and traps that will make the game never boring and very compelling. Initially we will only need one shot to kill the opponents, but increasing the levels we will face more " coriaceous" opponents who will need several shots.
During your exploration, you'll find chests and treasures to help you. In each box we will find several useful bonuses like a fairy who will give us a shield to protect from opponents shots or a feather that will give us the chance to fly… And much more.



Every 16,000 points (collected with bonuses or destroying the enemies) we will be awarded an extra life.
There are also checkpoints, as in any respected platform, usually located at the middle of the level and really useful during the longest and most complicated routes (the third world for example).
The game also has a convenient password system to save the location. Passwords will be provided at the end of each tower.
Once we reach the top of the towers we'll be faced by their big bosses.







Classic monsters which represents a challenge that is not simple but even not too difficult, once you have learned the attack patterns.
This is a big plus. This well-balanced difficulty makes Micro Mages a compelling one.

There is also a multi-player mode, very well implemented and quite fun. As long as at least one player is alive, we won't lose the character's life. Players killed along the way will turn into ghosts with the power to freeze opponents and help survivors. They can also destroy crates and scribes with the hope to find a fairy or a bonus that will bring them back to life.
As in the most classic platform of the 80s, once the a round is completed and the final boss is defeated, it will be possible to start over with an increased level of difficulty.
What can I say? A new game, well structured and with great level design and perfect gameplay.

The graphic is impressive. Detailed, animated with care (the little magician is featured with incredible animations, ndN) and everything moves smoothly. A product that makes very good use of the Nes's capabilities and that does not even have a slowdown or flickering as we often see in games for the 8-bit Nintendo.

The soundtrack is simple and fits perfectly to the game. Not obsessive or repetitive, it serves as background for the adventure.

Painful notes... Few, but there are. The game is not impossible once you understand the patterns of levels, opponents and traps and only 4 worlds seem too few. Too bad because with a few more levels it would be the absolute "masterpiece" of new productions for the 8-bit NINTENDO.

The game is available in download on Steam and Itch.Io and runs on all emulators for the Nintendo console (Mugen, Nostalgia...) but I recommend you to do as I did: buy the cartridge equipped with the manual and a very well crafted package!

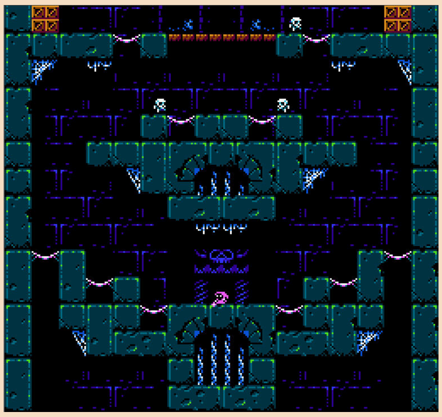
The cartridge can be ordered at:
https://www.brokestudio.fr/product/micro-mages-nes/

by **Carlo N. Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 90%
Smooth, dynamic and well balanced. It won't bore you and will keep you hooked to the joypad to complete all the 4 worlds of the game.

» Longevity 65%
...4 Worlds that are a little bit few and once you get the patterns of opponents and the logic of the puzzles it will be easy and fast to finish the game. Such a shame...



DOOM

Year: 1993
Publisher: Id Software
Developer: Id Software
Platform: Pc MS DOS, Atari Jaguar, Super Nintendo, Sega Mega32x, Sega Saturn, Sony Playstation, 3DO, Amiga.
Genre: FPS

Hunting rifles, chainsaws and demons. All this has been and is Doom, and although this concept seems silly and fanciful, it is actually one of the key ingredients of this legend of video games.

But why all this success?
In 1993, a number of controversial titles had problems. Most parents and trade associations fought the violence in Mortal Kombat or the controversial Wolfestein 3D (Doom's father). They were the violent and bloody titles that were on the market.
Did the controversy also break out on Doom? Yes, but in smaller quantities. The title was released on the shareware market and also spread in the world by fans word of mouth.

The trade associations were silenced by the sales of the game and especially by the players themselves! It was a just niche product for the period.
Zero plot (okay, a little bit, ndN) and is the protagonist relevant? Not even a little.

Doomguy, our general space marine was an example of experiential storytelling, through the player himself instead of simply being exposed to it. Obviously, the element that made it so engaging (in addition to the first-person perspective) were its fantastic visual effects, similar to a Brutal Death Metal cover; they were banal but fantastic ...And violent ...Oh, they were ultra violent!!!

For a period of time, this pixelated tribute to action and horror movies has been the next generation of video games. Every subsequent FPS game on PC, console and even Amiga was in the same style as Doom.
Despite his age, it is still exceptional. The graphics appear realistic, almost gritty, whether it is a simple wall or an ultra-decorated surface with a diabolical appearance.

The various levels of brightness give rise to claustrophobic corridors, with flickering or oscillating lights or even black like pitch.

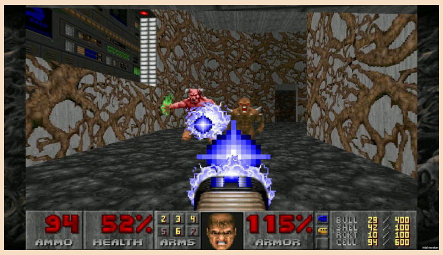
Movies like Alien were obviously key influences, the main theme of the franchise has always been inspired by HR Giger, immoral marriage of meat and machines and, like many great previous games, level design is abstract rather than realistic, and surprising rather than trivial. The enemies look fantastic, they have incredibly detailed and gruesome death animations, and because they are actually photographed sculptures, they have an almost realistic feeling. A heterogeneous group of undead and demons that has become almost iconic as the Goomba in Super Mario Bros or the ghosts in PAC MAN.

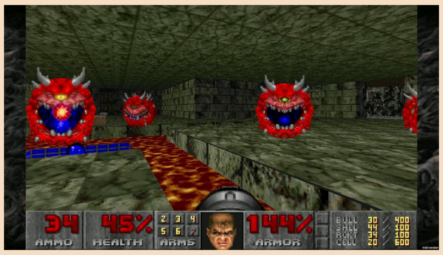
Appearance is only part of the equation, with the sound completing the package in the best possible way.

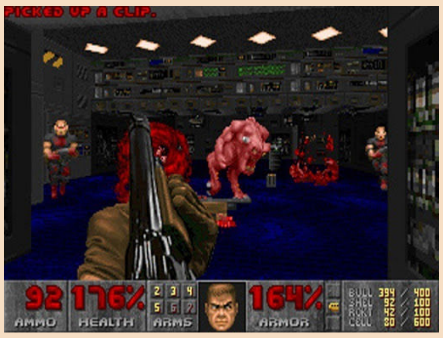
In addition to a classic MIDI soundtrack (remember MIDI?) inspired - and slightly torn - by metal legends such as Black Sabbath, Metallica, Megadeth and Judas Priest, among others, as well as some slower ambient and mood-enhancing melodies such as the cherry on the cake.

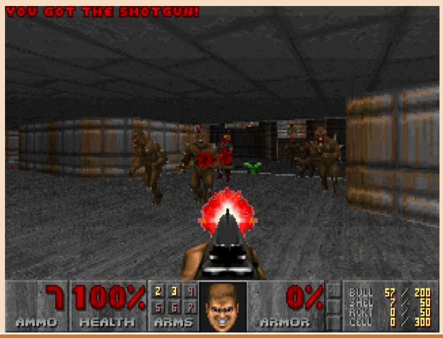
All the sound effects, from the powerful explosions of weapons, to the screams and growls of alarmed enemies, are just perfect - and wait for a carnivorous infernal bull made of steroids who wants to finish you off! Hearing the echo of the roar of the gigantic Cyberdemon; sounds like coupling ferocity and pain.

But in all this pureness we find a stain, a stain that, especially in these times,









makes us understand the age of this product. What is it? Well... Our hero can't tie his shoes, look at the mouses... I mean, he can't lower his eyes or even raise them because three-dimensionality was still "a Prototype" and not yet fully implemented (it will arrive a few years later with Quake and Descent).

That makes Doom old, makes it a masterpiece from another era.
Where we haven't concerned by photorealism, 60 fps, 4k/8k or ultra-realistic engine.
We were just thinking of using chainsaws to gut every Cacodemon in the world.

At the time of its release, Doom was criticized arguing that it was a simple game in which waves of enemies were shot down; enemy after enemy. It was far away from the truth.

This is not the usual Left for Dead, but a game in which we will find a complex design of fantastic levels, full of traps, with intelligently positioned enemies, tons of secrets and many situations in which you will need to think to not being slaughtered by demons.

The keyword when it comes to high-level design is "balance." The enemy count is high, but not so high as to make repetitive firefights; there are many traps, but they leave enough room for the player to react and confront them; ammunition is limited, but enough to do the deed and, finally, while part of the fun is trying to find a key to open a door or a switch to lower an elevator, it is managed in such a way that you never feel bored, especially since every time a key is collected, some areas will be repopulated.

Over the years, fans of the saga have created hundreds of maps, mods, additional layers and systems to make this product eternal. Some of these products are incredibly manufactured like Brutal Doom from which the series restarted again.

It was just adrenaline! No regenerative health, no guidance on where to go next, no artificial intelligence with scripts and no specific covers to hide behind something.

Run like a madman between levels, shoot down demons after demons, collect useful objects.

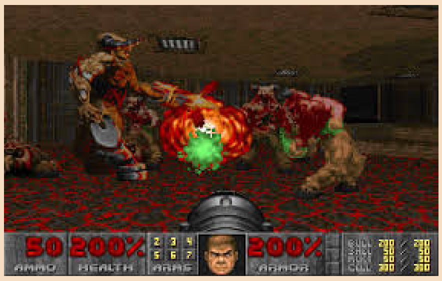
This was DOOM and this is DOOM again.

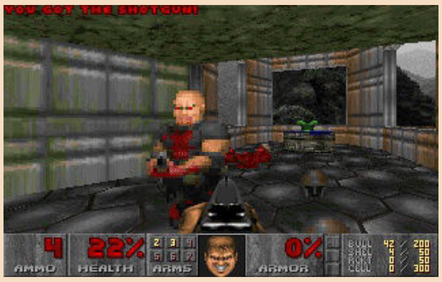
by **Carlo N. Del Mar Pirazzini**

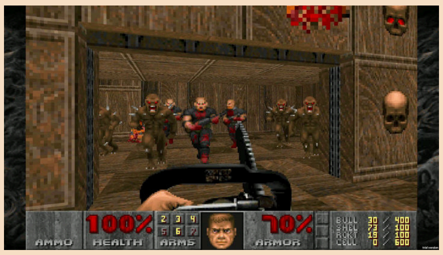
OUR FINAL SCORE

» Gameplay 95%
Immediate, intuitive, fast, frenetic... Many weapons, levels and hidden bonuses. Lets you play and makes you play like crazy.

» Longevity 95%
Many believe that Doom is just a relic from the past; an important relic, of course, but nothing more. False! The creation of Id Software is not only pioneering, but also one of the best video games ever made; a brilliant work of a company whose mentality was practically quite similar to what independent developers do today. A game of players for players. It's beautiful, it's fast, it's deeply compelling, immense in single and superb in multiplayer.
It's Doom, and it's here to stay until the Hell burns!

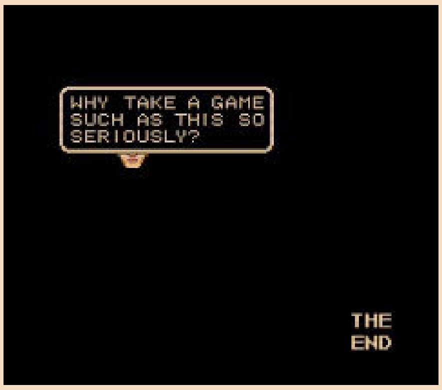


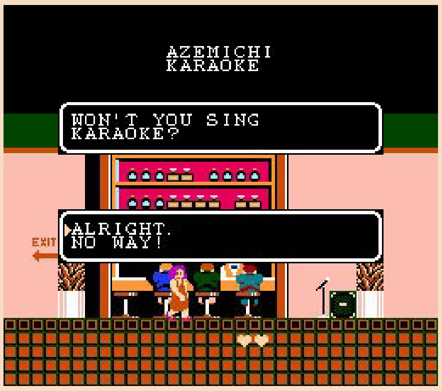




NINTENDO VS TAKESHI

Year: 1986
Developer: Taito
Genre: Platform
Platform: Nintendo FamiCom





Welcome to this new episode dedidated to Japan; according to our internal "Halloween-calendar" it is the second part of the 666th episode which officially corresponds to the 14th episode on Japan on RMW! Enjoy the reading and when you have finished reading the article, I hope you will agree with my full belief that the review of this video game is particularly relevant to this special issue!

We are talking about a video game inspired, indeed created, by the sparkling mind of the famous "Beat Takeshi" (two beats: in the sense of the rhythm of comedy because at the beginning he performed in comic duets).
Takeshi Kitano is one of the world's most famous film characters, especially in Japan. He's a very authoritative public figure. He's an actor, director, screenwriter, editor, film producer, television host, television author, radio host, comedian, writer, painter, singer, and video game author.
He is considered one of the most important living oriental directors due to the unmistakability of his style, the radicality and innovative strength of his filming. Often his movies are focused on Japanese mafia (Yakuza), not always appreciated or taken too seriously.

Given his fame, it was mandatory to create a video game dedicated to his interesting character.
Did I say video game? Or a nightmare with open dreams? In any case, it is present in the top ten of the worst Japanese games.

In fact, the game has even won the first prize as the worst game for Nintendo FamiCom.

Some parts of the game are traced back to the philosophy of his famous TV show "Takeshi's Challenge" or "Takeshi's Castle" known in Italy as "Never Say Banzai", commented by the great "Gialappa's Band".

https://youtu.be/kyLzDuAkqn8

https://youtu.be/8t2T4jGI6G0

https://youtu.be/tsuj4XkadUg

The program was animated by reckless Japanese who were striving for impossible, comic - unsafe challenges: the ultimate goal was to conquer Takeshi's virtual castle. This series was a success, but the video game inspired by Takeshi's works was very problematic.
Back to the video game. At the dawn of this story, Taito was licensed to create a video game dedicated to Takeshi's identity.

The problem is that Takeshi was involved too much in the game processing activity, leaving no room for the video game maker experience: a very dangerous choice!

It is said that Takeshi created the whole plot of the game in one night, at the bar, the famous Izakaya bars, where a thousand colourful saucers and glasses with surreal shapes and contents mingle mercilessly inside the stomach, causing bizarre and unpredictable digestive as well as psychosomatic alchemies.

Taito producers in the bar recorded everything Takeshi proposed. Takeshi wanted a completely different game from the others of the time: he succeeded perfectly in the intent, unfortunately in the negative sense of the phrase.

The presence of such a famous celebrity would surely have sold infinite copies of that video game, regardless of the quality of the game itself. I assure you that I possess it and that I love retrogaming, but at the same time, unfortunately, I too feel disgusted by this unfortunate work. It is useful to remember that Takeshi hated electronics, mobile phones, emails and his ideas were always mixed in a bizarre combination of tradition and madness. Takeshi has really created an awkward cartridge for Nintendo FamiCom.

Moreover, during the development of the game, Takeshi was involved in an extramarital scandal and ended up in the papers.
The editor of the newspaper that published the news was even beaten by Takeshi himself (along with his followers) during a violent rapture of madness: from that day on the great Takeshi who belonged to the historical comedic duo "Beat Takeshi" was enriched with the qualification of "Beat Takeshi" in the sense of "to beat up".

Away from television for a few months, he obviously returned more famous than before and completed his unfinished business, scheduling numerous future engagements.

The game consists of a very simple platform, with several elements and bonuses to discover. In short, it is a platform where you have to make the classic decisions to advance at the level. The music is frustrating, a loop without harmony, not listenable at all. The English translation, fortunately, is excellent.

https://youtu.be/j_RH518LyOk

There are many peculiarities of the game, I will not be able to list all of them since I have played it three times and I will certainly not play it again in the future:

1) if you throw punches in the air 20,000 times on the title screen you get directly to the end of the game
2) there are levels where you have to attach the second pad equipped with a microphone and sing like in the Karaoke halls
3) we can divorce and/or beat the shit out of wife and children (remember that the game is officially released by Nintendo!)
4) we can beat our employer
5) we can beat the cops and at the same time be beaten by them, losing energy
6) we can beat everyone like crazy and be hated by everyone and above all be beaten by people who hate us
7) there are porn situations, you can drink whiskey and play Pachinko
8) inside the Pachinko room we can only advance to the next levels if we kill the people who are beating us
9) we will be able to fly aircraft, even an aeroplane, but it will not be able to land and therefore we will crash to the ground unnecessarily, resulting in game over
10) the end of the game is a small drawing with Takeshi's face that says "congratulations"
11) plus numerous other strange things that I absolutely do not want to remember or describe...

It seems like a tragicomic, violent and absurd work, created by a person who hates video games: a logic typical of Takeshi who really hates electronics and video games.
Perhaps this logic was really wanted by Takeshi, with a macabre conscience. No one will ever know. No one will ask Takeshi for an explanation. Few will remember this troublesome game. I think Nintendo is not even happy with the presence of this article, patience, I will try to make it up to you with a letter of apology!

Even the creators of the "Strategy Guide" to complete the game have “cold sweated”, facing such an absurd, irrational and illogical game.

If you will find this cartridge in Tokyo's thrift stores, you absolutely must take it, it is a collectible monstrosity, with the consequent monstrous logic of collecting prices. Unfortunately, the manual is even rarer. I don't own it either.

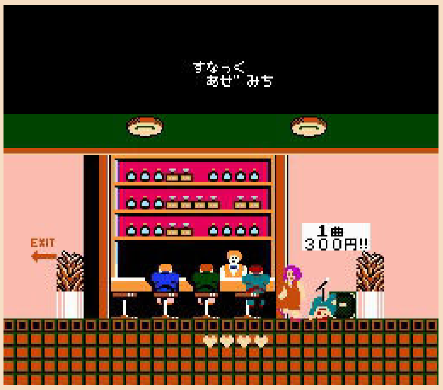
Well dear readers, I very much hope that this review has respected the "horror" soul of Halloween, goodbye to the next issue, I promise you that we will return to normal talking about Japanese madness!

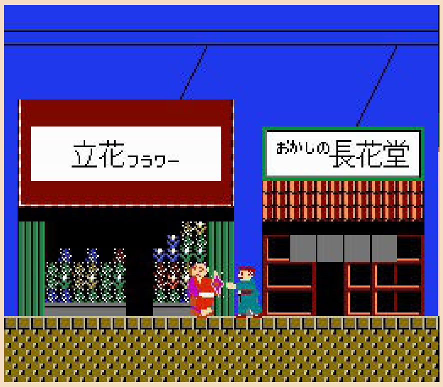
by **Michele "Conte Ugolino"**

OUR FINAL SCORE

» Gameplay 1%
It's a traumatizing experience!

» Longevity 1%
You can't wait to turn off the FamiCom!





STORMLORD

Year: 1989
Developer: Nick Jones, Raffaele Cecco, Huges Binns
Platfom: Commodore 64
Genre: Platform

Another autumn although very different and particular, another Halloween coming up with some last-minute limitations and another review of a game in theme with the Halloween night.

A Night that is perhaps one of the few that makes you feel the spirit and the greed; since every year entertains many Europeans and Americans and especially us, video players, thanks to the horror titles that were already popular on the "breadbin" at the time. I had played and tried many of them, but one that struck me particularly, especially for the soundtrack that scared me, was Stormlord.

Even this game I discovered by pure chance, assuming that case exists, on a newsstand, whose name I do not remember in Italian (but I do the English one!).

After the welcome screen, the game presents itself as a sliding platform, in which you impersonate an elderly warrior wizard who jumps, fires flames and teleports (and all despite his age).

The purpose of the game, as well as to overcome each of the four tough levels, is to free up the imprisoned fairies and solving puzzles with the right objects; for example the key to open the doors, honey moves the swarm of bees etc...
Meanwhile dozens and dozens of demonic creatures are ready to block the way and make the game even harder.

Fortunately, since the beginning we have eight lives to complete the entire game and, believe me, it will not be a piece of cake, like most games of the time (sorry if I'm a bit repetitive).

At the end of each level there is a bonus that will earn you several points and perhaps even an extra life.

It's up to you to find out what kind of bonus it is and how to collect points and lives and then move on to the next level and free other fairies: fairies that grow numerically from level to level.

Completing your task will not be easy; I remember perfectly those hateful worms that came out of the ground at the speed of light and made me lose a third of the lives and also some riddles solved in the wrong way, which forced me to start again since the very beginning.

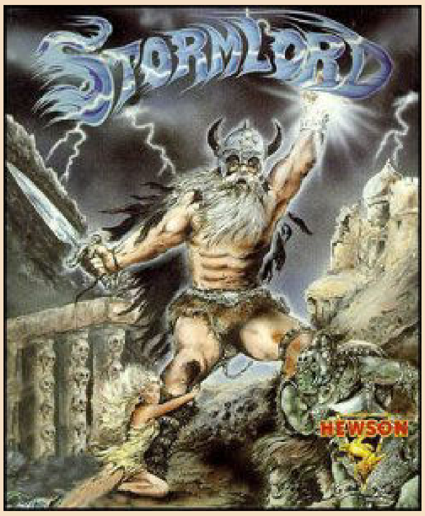
The game came out on different platforms and by playing the various versions I was able to make a comparison of the graphic and music; they are the two elements that most characterise this game and as always the music of the Commodore 64 version is the best one!

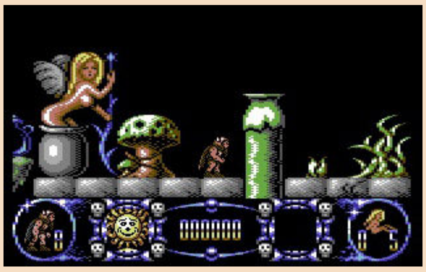
Stormlord also had a sequel very similar to the first chapter but with linear scrolling and end level bossese; maybe it was not as successful as the original one but it was still worth our time and our energy.

If you are forced to observe the curfew on Witches' Night, get this game together with many other beautiful and ugly horrors that have made the history of the C64 and be sure to turn off the lights!

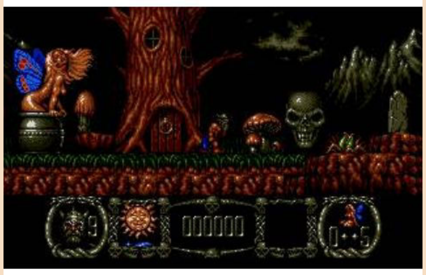
Happy Halloween!

by **Daniele Brahimi**





Commodore 64 version



Amiga version

OUR FINAL SCORE

» Gameplay 60%
It's easy to die... But as soon as you get used to...

» Longevity 70%
Only four levels? Play it, then you will tell me!

100,000 times, Thank You!

A few days ago we made an announcement on our official Facebook page to celebrate the **100,000 downloads of all our publications**.

Unfortunately, Facebook posts are rather termporary and disappear pretty quickly, swallowed up by other posts and the frenzy with which we approach the everyday news. Together with the entire editorial staff, we therefore felt necessary to come back again on this subject and to Thank our readers once again!

Those who have been following us for some time may know that we do not like to report numbers and statistics. We rather prefer the human aspect and the emotions that we can arouse with our articles. Feedbacks and/or compliments received for a particularly appreciated article, are our most important award.

But this time I'd be lying if I say this number didn't make any sense to us. This number, big, round, high-sounding, awakened our ego and... We wanted to celebrate!

It is relevant to say that this number is only a part, certainly the largest, of the absolute total. Some sites mirror our magazines and we have also found some of them on archive.org. This is certainly another point of pride for all of us.

But let's stop talking about what we have achieved.
Reached objectives are already part of the past, while we, apart for computers :-D, are always oriented towards the future!

During the last internal editorial staff meeting, we discussed many points that we would like to complete in the near future. There are several projects we hope to be able to announce very soon, but what we would like to see more and more is the active participation of our readers with their contributions.

Now more than ever there are so many people working on projects dedicated to retrocomputing. New hardware devices, emulators, games and programs are released almost daily and it's hard to keep up with everything given their amount and the speed at which they are announced.

If you, who are reading, are part of one of these projects, please contact us. We would like to give everyone visibility and a voice.

As we have always said: **RMW is your magazine!**

Francesco Fiorentini

RetroMagazine World
Year 1 - Issue 4 - NOVEMBER 2020

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