Future days are back

RETROMAGAZINE WORLD

ISSUE 26 - WINTER EDITION - WWW.RETROMAGAZINE.NET

SEGA MASTER SYSTEM

ATLANTEAN UNDER THE SEA ON THE PC ENGINE

PROGRAMMING - HISTORY - INTERVIEWS - REVIEWS - INSIGHTS

The important thing is to participate, but we want to win!

Welcome to this latest issue of RetroMagazine, where nostalgia and vintage technology come together to offer you a fascinating journey into the past of computing. As the world gathered in Paris to celebrate the 2024 Olympics, we took the opportunity to remind you how computers of the past have influenced the way we experience everyday events.

The Olympics have always been a global stage for innovation, not only in sports but also in technology. From electronic timing to advanced race management systems, computers have played a crucial role in ensuring that these events run smoothly and accurately. The timing of sporting events has undergone significant changes since the 1980s, evolving thanks to technological innovations that have made the process more accurate, reliable, and data-rich.

Similarly, software development has benefited from this technological evolution, which has enabled developers and enthusiasts to meet, even virtually, bringing a wave of innovation to the way programming is approached. Examples of this are the dozens of games that are periodically released for our retro machines and, in this specific issue, the conversions of Prince of Persia for Plus4 and VIC20. Hands up anyone who would have imagined this little miracle possible 40 years ago...

I would also like to invite you to reflect on how retrocomputing is not only a celebration of past technology, but also a way to better understand continuous innovation, analyzing how the fundamental principles of programming and hardware design in the 1970s and 1980s laid the foundations for the advanced solutions we see today.

As Paris hosts the 2024 Games, we celebrate the resilience and creativity of the minds that shaped the world of retrocomputing.

We hope this issue continues to inspire you to rediscover the charm of computers of the past and recognize their lasting impact on contemporary technological innovation.

Francesco Fiorentini

Publication Note: this editorial and all of the contents were first released in Italian in July 2024 - ref. issue 48-IT

TABLE OF CONTENTS

The following people contributed to this issue of RetroMagazine World (in no particular order):

- Carlo N. Del Mar Pirazzini
- Daniele Brahimi
- Francesco Fiorentini
- Giulio Fieramosca
- Roberto Del Mar Pirazzini
- Giampaolo Moraschi
- Leonardo Miliani
- Takahiro Yoshioka
- Barbara “Morgana” Murgida
- Eugenio Rapella
- Diamanti Maurizio
- Marta Rossmann
- Gianluca Girelli
- András Vajda
- Fredrik Ramsberg

- Cover image: **Giuseppe Mangini**
- Cover layout: **Carlo N. Del Mar Pirazzini**

Original issue was first released in Italian in July 2024

Sega Master System

by Leonardo Miliani

Not only stung by the fact that Nintendo had unveiled its Famicom on the same day as the launch of its SG-1000 console, but also by its lack of commercial success compared to its rival, Sega decided to revamp its console hardware and unveiled an 8-bit product with mixed fortunes, the Sega Master System (fig. 1). Dominating the PAL markets, it was defeated in the NTSC markets but, despite this, the console served as an incentive for the Japanese manufacturer to continue investing in the home gaming systems market for several years to come.

From the SG-1000 to the Master System

As already seen in a previous article, the SG-1000 was created on the proposal of Sega's Japanese subsidiary which, in 1982, seeing the decline that was beginning to affect the arcade game market, especially in North America, suggested that the US parent company focus on the production of a home computer. During the development of what would become the SC-3000, Sega learned that Nintendo was investing in the development of a home game console, and the SC-3000 project was revised to become a console itself, the SG-1000. The hardware of the two systems is common, based on a Zilog Z80 CPU alongside a Texas Instruments TMS9918A graphics chip and an SN76489 audio chip. These were discrete components which, due to their average specifications, were unable to compete with the Famicom. However, at the end of 1983, a few months after its launch, sales of the SG-1000 were going fairly well, because the Famicom had technical problems and Nintendo not only recalled the consoles already sold but also delayed the distribution of those in stock in order to correct these defects. As a result, the Sega console enjoyed a few months of glory due to the absence of competition, but as soon as the Nintendo product returned to the market, it began to rack up sales again.

Meanwhile, Sega was going through a critical moment in its existence: West & Gulf, the multinational company to which it belonged, decided to dispose of its unprofitable divisions, and Sega, which was going through the famous 1983 video game crisis in America, was not doing well and was broken up, with the arcade division sold to a rival company while the American research and development division and the Japanese subsidiary were temporarily

Fig. 1 - The Sega Master System (image: Evan-Amos – source: Wikimedia Commons)

retained. Unfortunately, things did not improve and, in early 1984, the latter was also put up for sale. Fortunately, former administrator David Rosen and current president Hayao Nakayama, with the financial support of the CSK Corporation group, took over the company and began a relaunch plan. They redesigned the SG-1000 to make it more aesthetically appealing, solving some of the design problems of the first version, such as the joysticks that could not be detached from the main body. The SG-1000 Mark II went on sale in mid-1984, but sales were still poor. Nintendo implemented an aggressive commercial policy, tying video game developers to itself: anyone who wanted to publish games for the Famicom had to release them exclusively for its console. Unable to offer the same games as its competitor, Sega's offering consisted mainly of conversions of its arcade games. Conceptually, the hardware was the same as the previous system, based on the TMS9918A, which was objectively inferior to the Famicom's video chip: the latter offered many more sprites, which were also multicolored, and a much richer palette. Sega therefore decided to revise the TMS9918A design and came up with a chip capable of displaying 32 colors simultaneously on the screen from a palette of 64. The number of sprites that could be managed increased to 64, each with 16 colors. The audio chip was the same as the SG-1000 but, to save space and components, it was integrated directly into the graphics chip.

On October 20, 1985, two days after Nintendo released the international version of the Famicom in the US, renamed the Nintendo Entertainment System (NES), Sega presented the Mark III in Japan, an evolution of the SG-1000 Mark II with which it shared very little, being a completely different console (fig. 2).

Although equipped with superior technical features, the console failed to compete with the NES, mainly due to the aforementioned exclusives that Nintendo managed to secure from video game producers. Despite lackluster sales at home, Sega decided to export the console, thinking that it would do better on international markets than it was doing in Japan. In order to operate more effectively in the prolific US market, a subsidiary called Sega of

Fig. 2 - The Sega Mark III, evolution of the SG-1000 Mark II and progenitor of the Master System (image: Muband – source: Wikimedia Commons)

America was founded, headed by Bruce Lowry, who came directly from Nintendo of America, where he had served as vice president of sales. The choice of the company name was also his: he believed that "Sega" and "America" were two words that went well together, and, coming from a company called Nintendo of America, the choice seemed obvious. Sega of America initially worked on redesigning the console to suit Western tastes: the white case was abandoned in favor of a darker one, made of smoother plastic with more pronounced corners, cleaning up the angular appearance of the upper part and imprinting graphics reminiscent of cassette players.

The console was named Master System following a survey of Sega of America employees. However, final approval came from the president of the Japanese parent company, Okawa, who also approved it because of its reference to the fact that in any competition there is always only one 'Master', the one who stands out above all others, and this was seen as a good omen for the console's future.

Marketing

The console was unveiled at CES in Chicago in July 1986 and went on sale in September of the same year. The white, checkered box contained the console, pompously named the "Power Base," an optical gun, two controllers, and a multi-game cartridge, all for $200. Sega invested around $15 million to promote the console, but the results were not as hoped: at the end of the Christmas season, when the console was on sale for around $150 to counter NES sales, Sega had sold around 250,000 Master System units, less than a quarter of what Nintendo had sold. Although Sega pushed the quality of its games with the slogan "the graphics on the box are the graphics on the screen," indicating that what you see on the game packaging are not retouched images but the actual appearance of the titles as they appear on the screen, the NES continued to sell much better due to Nintendo's exclusive agreements and the limited supply of games from other developers. In practice, in North America, games for the console are produced not only by Sega, but also by Activision and Parker Brothers. To boost sales, former NASA astronaut Scott Carpenter is hired, much like Commodore did a few years earlier when it took on William Shatner, Captain Kirk of Star Trek, as its spokesperson to advertise its VIC-20.

In an attempt to boost sales, Sega of America reached an agreement with the Tonka toy chain, granting it exclusive marketing rights in the US, in the hope that sales of the console, which could also be purchased in ordinary toy stores, would pick up. Unfortunately, Tonka had no experience in managing sales of electronic gaming devices and sales continued to languish. At the end of 1987, Sega also released the Master System in Japan, but here too, things continued to go badly, failing to compete with the NES. The Master System fared much better in other markets. In Europe, the console arrived in mid-1987, initially in the UK, France, and Germany, but with a marketing launch marked by a hiccup. Due to Sega's slowness in fulfilling pre-orders, local distributors suffered heavy financial losses and terminated their contracts. In early 1988, Virgin Mastertronic took over distribution across the European market and promoted the console by leveraging its superior graphics capabilities compared to the Commodore 64 and Sinclair Spectrum. Bolstered by the poor penetration of the NES in Europe, the Master System quickly established itself as the best-selling gaming system: in 1990, Virgin Mastertronic sold just under 1 million Master Systems. Encouraged by these figures, several European game developers decided to release their titles for the Sega console, with the result that the PAL version of the Master System beat the NTSC version: around 100 games were produced in the US, compared to 300 in Europe.

The console was also launched in South Korea, where it was well received and remained the best-selling console until at least 1993. The numbers were also very good in Australia. At the end of 1989, the Master System was also distributed in Brazil by TecToy, which did a great job of marketing the console successfully, promoting it not only through advertising but also by localizing the games for the console into Portuguese.

In 1988, the Mega Drive, Sega's first 16-bit console, was introduced in Japan, and immediately all the company's efforts focused on the newcomer. The Master System soon fell into oblivion: the last game released was in 1989, followed by a slow decline until 1991, when it was withdrawn from the market. In 1992, it also disappeared from North America, without leaving much of a trace. In 1994, it also left the Korean market, while in Europe, thanks to games that continued to be released regularly, it held out until

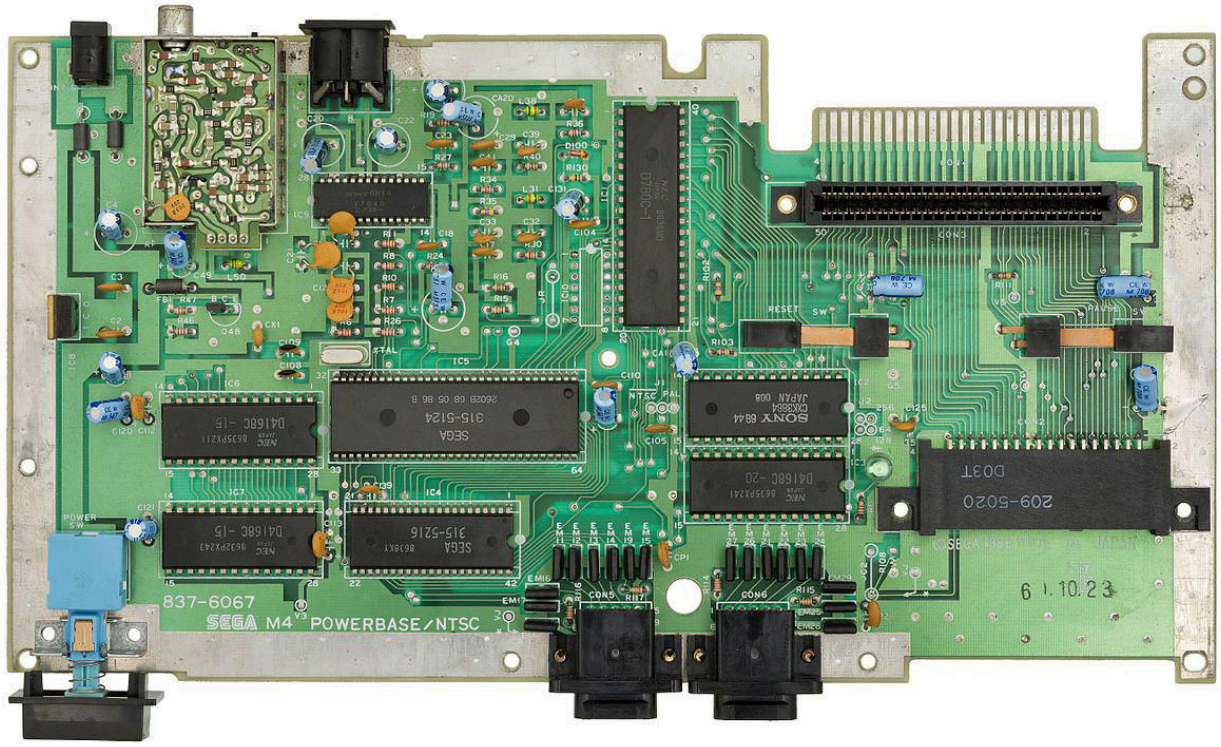


Fig. 3 - Sega Master System motherboard: note the Z80 CPU (the central chip positioned vertically) and the VDP (the large chip slightly to the left of center). On the right side are the Sega Card port (bottom) and the cartridge port (top) (image: Evan-Amos – source: Wikimedia Commons)

1996. Brazil remains a special case: after official marketing ended, TecToy continued to produce, at regular intervals, various consoles compatible with the Master System and, recently, even a device compatible with both the Master System and the Mega Drive.

Technical features

Developed to be better than the NES, the console succeeded admirably in this task, offering graphics with more sprites and colors than its Nintendo counterpart. As mentioned, the Video Processing Unit (VDP), the graphic heart of the Master System (fig. 3), derives directly from the TMS9918A, a Texas Instruments chip well known to lovers of 1980s computers, having equipped both the TI-99/4A and all computers in the MSX series. Starting from this basis, Sega's designers came up with a unit capable of displaying an image with many more colors: while the TMS9918A is limited to 16 colors (actually 15, because the 16th is actually a "non-color," a transparency), the VDP offers a palette of 64 colors, 32 of which are visible on the screen at the same time, according to a particular distribution: 16 colors are available for "tiles" (the 8x8 pixel tiles, or characters, that make up the background) while another 16 are available for sprites. The palette is indexed and the 32 colors are chosen from the 64 available and numbered from 0 to 31. In reality, there is a small clarification to be made: while for the background colors you can choose between the colors for the tiles (from 0 to 15) or those for the sprites (from 16 to 31), for the latter you can only choose between the colors reserved for them (from 16 to 31).

The resolution is 256x192 pixels but can be increased to 256x224 and, in PAL systems, even to 256x240: in the latter case, there is no classic border around the image. The VDP manages a maximum of 488 different tiles for the background, which can be reflected horizontally or vertically. In addition, tiles can appear both in front of and behind sprites. There can be 64 sprites on the screen at the same time (twice as many as the original TMS9918A), but a maximum of 8 per video line. Compared to those managed by the chip from which it derives, the VDP sprites are multicolored and their dimensions can be 8x8 or 8x16 pixels: there is also the possibility of enlarging them by doubling their size: they become, respectively, 16x16 and 16x32 pixels. Like background tiles, sprites also draw data in 8x8 pixel blocks from memory. This means that the more sprites with different details you have, the more memory is taken away from the background tiles, and

Fig. 4 - A Sega Card with the Hang-On game (source: fullyretro.com)

vice versa. Finally, the chip also manages vertical and horizontal scrolling.

The audio chip is integrated into the same silicon as the VDP: it is the audio core of the SN76489, capable of 3 square wave voices and a noise generator. Nothing exciting, but overall a decent and widely used audio processor, being used on the TI-99/4A, the BBC Micro, the ColecoVision, and several arcade games. In the original Japanese version of the console, the Mark III, there is an additional audio chip, the Yamaha YM2413, for FM audio generation.

The CPU is the same Zilog Z80 as in the SG-1000, clocked at 3.58 MHz. The console offers 8 KB of RAM for storing game data and 16 KB of VRAM, exclusive to the VDP. Games are distributed on two different media: classic ROM cartridges and Sega Cards, a proprietary format introduced with the SG-1000 (fig. 4). The latter are smaller and cheaper to produce, resulting in games with a lower purchase cost than those distributed on classic cartridges but, on the other hand, they offer significantly less storage space: while the capacity of a Sega Card ranges from 4 to 32 KB, that of a ROM cartridge ranges from 16 to 512 KB. An interesting fact: as with other manufacturers, Sega also distributed its games by reporting the capacity of the medium in bits rather than bytes, so that they would appear larger and thus give the impression of containing more game data: this resulted in media with capacities of, for example, 256 Kbit (32 KB), 1 Mbit (128 KB), 2 Mbit (256 KB) or 4 Mbit (512 KB).

Aesthetically, the console looks like a wide, low black parallelepiped. At the front, there are two sockets for the controllers and the power button, while at the rear there is the composite video output, the RF output for connection to a standard TV, the TV channel selector for the signal output, and the power input. On the top are the ROM cartridge slot, the Sega Card port, and the game reset and pause buttons. The game controllers are rectangular, with a directional pad that can be fitted with a small stick to turn it into a small joystick, and two buttons.

Accessories and revisions

Several accessories were produced for the Master System. The most famous are the SegaScope 3-D glasses (fig. 5) and the Light Phaser optical gun (fig. 6). The former are special glasses that, when combined with specially developed games, allow for stereoscopic vision. The glasses connect to the console via the Sega Card port, and only eight games were developed for them, including OutRun 3D and Zaxxon 3-D. The Light Phaser is an optical gun modeled after the weapon used in the Japanese anime Zillion, and was used in 13 games, all published outside Japan. There are also reinterpretations of classic gaming peripherals, such as the Sega Control Stick, an arcade-style joystick, and the Sega Sports Pad (not released in Europe), a trackball designed for sports games but used in only three titles ("Sports Pad Football," "Sports Pad Soccer," and "NFL Sunday Ticket").

Sega also released a portable version of the console, the Sega Game Gear (fig. 7): it was launched in Japan in late 1990, then in Europe and the United States in 1991, and finally in Australia and New Zealand the following year. It is a battery-powered device with a small 3.2-inch diagonal color LCD screen: compared to the Master System, the resolution is 160x144 pixels and there are still 32 colors on screen at the same time, but its VDP has an expanded palette of 4,096 different shades. Using an

Fig. 5 - The Sega Scope 3-D
(image: Boffy B – source: Wikimedia Commons)

adapter called Master Gear, it is possible to play Master System games on the Game Gear, thanks to the fact that it shares virtually the same hardware as the console.

In 1990, the console was redesigned and, in order to keep production costs down, its size was reduced and some features were removed, including the composite video output, the reset button, and the Sega Card port. A game was integrated, which started when the console was turned on without a cartridge inserted: for NTSC versions, there was "Alex Kidd in Miracle World," while for PAL versions, there was "Sonic the Hedgehog." It was marketed as the Master System II and was well received by the public, especially in Europe and Brazil (fig. 8).

Thanks to the console's success in Brazil, TecToy introduced the Master System III Compact in 1992, which was actually just a rebranded Master System II (fig. 9). The console was distributed with a multi-game cartridge included in the package. TecToy also produced the Master System Super Compact, again for the Brazilian market only, which was smaller in size and equipped with a radio transmitter to send the signal to the TV. In 2008, TecToy finally released a version of the Master System III based on a

Fig. 7 - The Sega Game Gear
(image: Evan-Amos - source: Wikimedia Commons)

Fig. 6 - The Light Phaser, designed based on the laser gun from the anime "Zillion"
(image: Evan-Amos – source: Wikimedia Commons)

SoC with 131 integrated games.

Games

As already mentioned, Nintendo secured exclusive rights to publish games for its console. This cut out several other consoles, including Sega's Master System, which failed to compete with its rival in several markets, such as Japan and North America, which have always been two of the most prolific for video games. Despite this, its popularity in other markets, such as Europe and Brazil, encouraged several developers to produce titles specifically for the Master System. This phenomenon was particularly noticeable in Europe, where production was so prolific that it far outstripped that of North America: many games developed in Europe were in fact exclusives published only for the PAL version of the console, such as 'Asterix' (fig. 11). As Nintendo did with Mario, Sega also sought fortune by creating a mascot and entrusting it with the role of representing the console itself. While this role was initially filled unofficially by "Alex Kidd," who appeared in "Alex Kidd in Miracle World" (fig. 10) in 1986, it was later officially entrusted to "Sonic the Hedgehog" (fig. 18), the blue hedgehog who first appeared in the 1991 game of the same name. In addition to the games based on these characters, the arcade conversions by Sega and other manufacturers, such as Bubble Bobble (fig. 15), are also very famous. There are games that have been able to bring out the best in the Master System more than others and have become true icons over time, such as Ninja Gaiden (fig. 13), those with Mickey Mouse as the main character, such as Castle Illusion starring Mickey Mouse (fig. 12) or Wonder Boy III (fig. 17). Thanks to the console's graphics capabilities, game production remained active

Fig. 8 - The Sega Master System II (image: Evan-Amos - source: Wikimedia Commons)

Fig. 9 - The Sega Master System III, a Brazilian exclusive marketed by TecToy (image: Neowoikkin2113 - source: Wikimedia Commons)

even after its discontinuation and the spread of new-generation systems, with conversions of games created for these systems, such as "Street Fighter II" and "Prince of Persia" (fig. 16).

Legacy

Although it did not achieve record sales during its commercial life, the Master System remains a milestone in Sega's history. It is the console that strengthened the Japanese company's image as a console manufacturer, building on the legacy of the SG-1000 and consolidating its presence in the industry, paving the way for the sales success that was the Mega Drive, and introducing the mascot Sonic, who became the very image of the company. Its graphics left their mark on thousands of fans, so much so that, years after its discontinuation, in some countries, consoles sold in somewhat dubious nostalgia sales turned out to be real hunts for console enthusiasts. At the end of the day, the 13-15 million consoles sold worldwide and the approximately 8 million units sold in Brazil alone are numbers that clearly express the love for an object that has remained in the hearts of many.

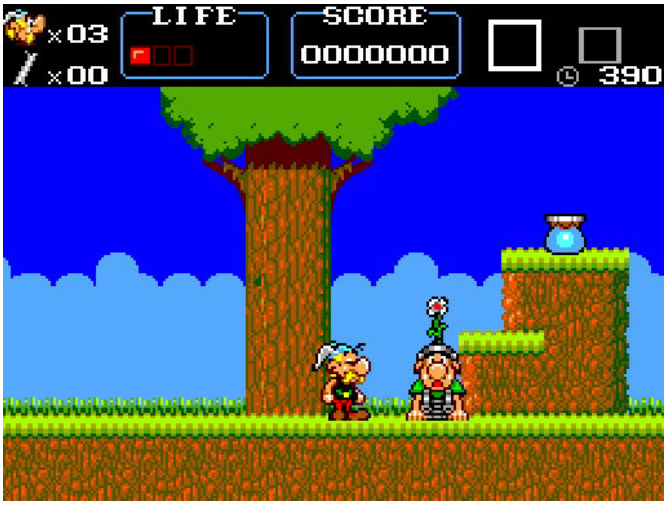


Fig. 10 - Asterix (source: Mobygames.com)

Fig. 10 - Alex Kidd in Miracle World (source: Mobygames.com)

Fig. 10 - Castle Illusion - starring Mickey Mouse (source: Mobygames.com)

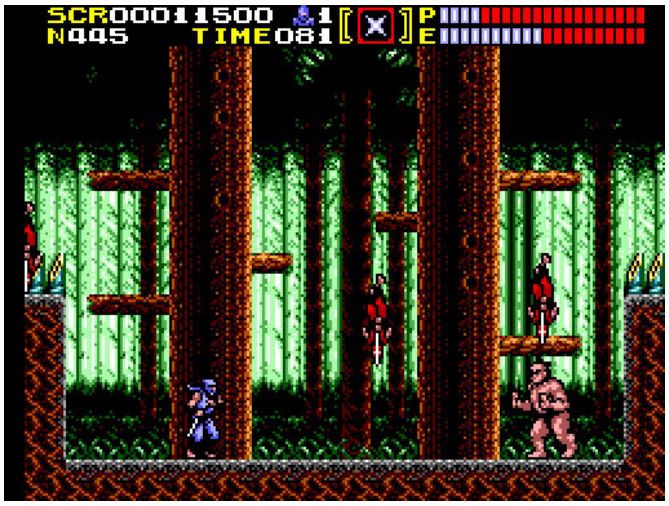


Fig. 13 - Ninja Gaiden
(source: Mobygames.com)

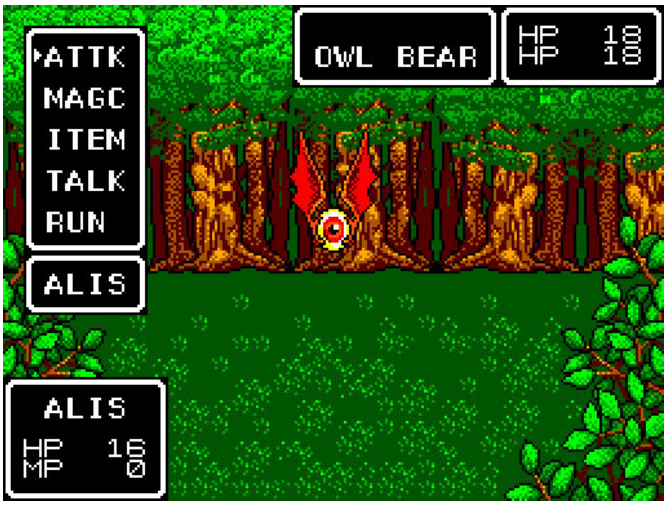


Fig. 14 - Phantasy Star
(source: Mobygames.com)

Fig. 15 - Bubble Bobble
(source: Mobygames.com)

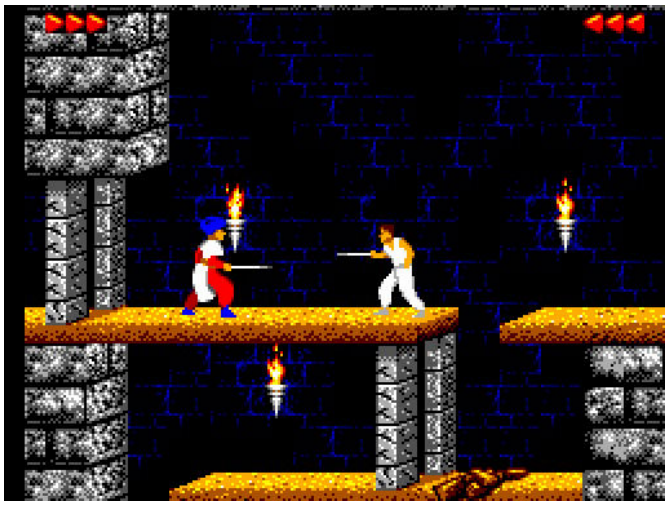


Fig. 16 - Prince of Persia
(source: Mobygames.com)

Fig. 17 - Wonder Boy III - The dragon's Trap
(source: Mobygames.com)

Fig. 18 - Sonic the Hedgehog
(source: Mobygames.com)

Tiny Disk II

by Giulio Fieramosca

About ten years ago, I came into possession of an Apple] [e, saved before it was thrown away. Its many problems - failed memory chips and a mechanically faulty floppy drive - made it unattractive for experimentation, and it remained in the garage of our "RetrOfficina" for years, only being taken out for rare exhibitions. That was until 2022, when I decided to devote a lot of time to restoring this interesting and historic machine.

To cut a long story short, the original floppy drive is broken and I haven't been able to repair it. This is a great pity, as most of the software for this machine was distributed on floppy disks. Not only that, but even though it was equipped with a tape interface, in many cases it was necessary to preload more advanced versions of BASIC or DOS... from floppy disks. I looked around to consider purchasing one of those almighty floppy emulators, such as floppyEmu or wDrive. They are nice but expensive. This prompted me to create my own version of a floppy emulator, with the intention of making the entire project available online.

Before reinventing the wheel, I looked around the web for a free project, and I came across a certain "SDISK II," for which both the schematics and source code are available online. I tried to build it without success, but I didn't put much effort into it: I wasn't entirely convinced by the hardware. In any case, the firmware was an excellent and very useful starting point for my project.

I had no idea how a floppy disk worked, let alone the Apple II one. So, first of all, I started reading up on the electronics and the protocol.

The protocol - Physical organization of data

The soft casing of 5¼" floppy disks protects a plastic disk covered with a thin magnetizable layer. Information is recorded digitally by applying a magnetic field to a small area, which remains imprinted on the disk. Reading takes place through the reverse process, capturing changes in the disk's magnetization.

Reading and writing are performed by a pair of heads placed close to the disk while it is rotating. Imagining the surface divided into regular intervals, a particular encoding is used to map the bits on the disk: the bit "1" is represented by a polarity reversal and "0" by the absence of polarity transitions.

The magnetic field imprinted on the disk would be too weak to be digitized directly: for this reason, an amplifier is needed to keep the amplitude of the analog signal constant, so that both a newly written disk and a more damaged one can be read. Unfortunately, a long sequence of zeros gives an almost constant output, and Drive II will tend to "turn up the volume"... to the point of picking up tiny variations due to background noise. This is why a bit sequence can never contain more than two consecutive zeros. This is a major limitation that prevents us, for example, from transcribing text directly onto the disk:

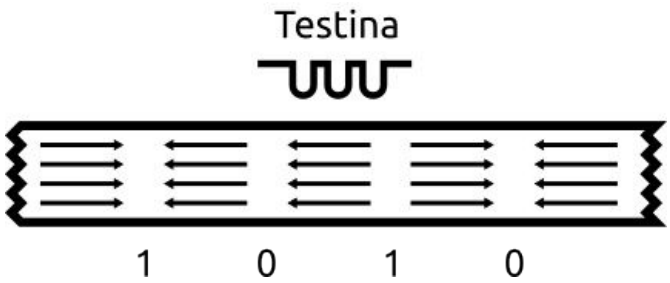


Fig. 1 - Representation of the magnetized surface of the diskette. Each polarity reversal is associated with a "1" bit.

the capital letter A, ASCII code 01000001, would be prohibited!

But the problems don't end there. During reading, the head picks up a continuous stream of bits, and the Drive II does not provide a mechanical means of signaling the start of a valid segment.

To overcome this inconvenience, the Drive II was designed to require that a valid byte must always have a "1" bit at the beginning. The capital letter A is therefore prohibited. Furthermore, to ensure correct alignment, a particular 12-bit sequence, equal to "1111111100", is placed before each valid data segment, which causes the reading system to always align itself with the first "1". This code is also called "self-synced FF".

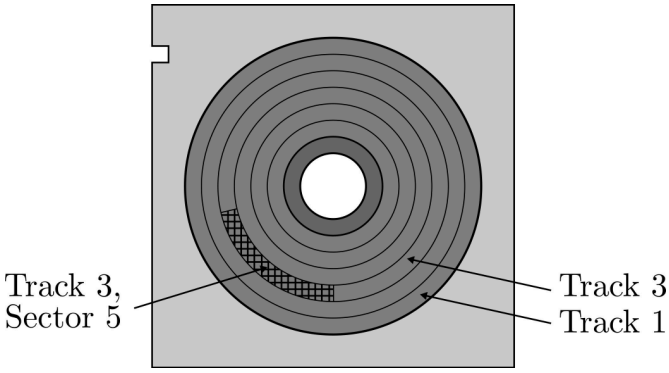
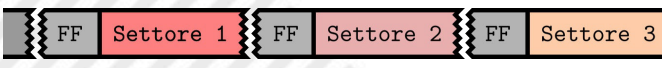
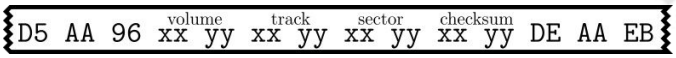


Fig. 2 - Geometry of the floppy disk

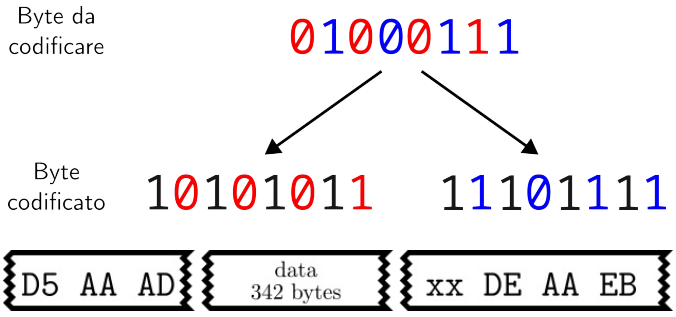
As with the latest 3.5" floppy disks and hard disks, the disk is divided into concentric tracks and for each track the data is segmented into sectors, so that reading and writing can be managed in smaller groups. The information content of a track will be similar to the figure below.



The unique feature of the Apple Drive II, which sets it apart from other models of the time, is the extreme simplicity of its hardware. On the other hand, the management of the disk geometry is entirely handled by the software.

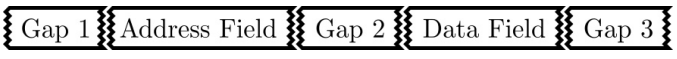


Therefore, it is necessary to distinguish one sector from another in some way. For this reason, each sector is associated with a description field, called an "address field." The address field contains information identifying the sector within the disk, for example, "track number 2, sector number 7." These values require one byte each, but cannot be written directly because they may not be representable on disk. For this reason, each number is first encoded using an "odd-even" approach: given the sequence "10101010," the odd bits of the byte to be encoded are replaced with "0." Starting from the same sequence, the same is done with the even bits, resulting in two distinct bytes. It is clear that, in this way, the first bit will always be 1 and there can never be consecutive 0s.



The address field is followed by the data field, which contains the actual information. The data is not encoded using the "odd-even" technique because this would double the space occupied. Other more efficient encodings, called "5-and-3" and "6-and-2," have been preferred, but their implementation goes beyond the scope of this article, so I refer you directly to the manual "Beneath Apple DOS" for more information. It is sufficient for us to know that we can store 256 bytes in a sector, which, by adopting the most efficient encoding, increases to 342 bytes.

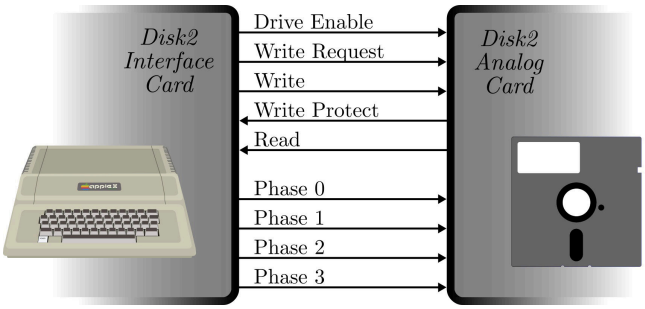
It is important to note that both fields have a unique three-character prologue, which is important for distinguishing and decoding them correctly. The first two bytes of both fields, "D5 AA", are reserved codes that cannot be used elsewhere and serve as an additional alignment system.



As mentioned, each field is preceded by "self-sync FF" sequences. In addition to serving as synchronization patterns, these "gaps" act as buffers for writing.

Since the write operation is implemented in software and

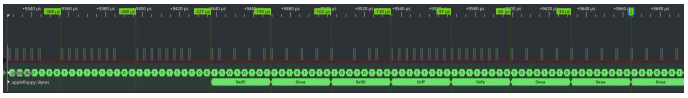
is performed "blindly," these buffer areas serve to provide sufficient margin to prevent the rewriting of a field from overwriting portions of adjacent sectors.



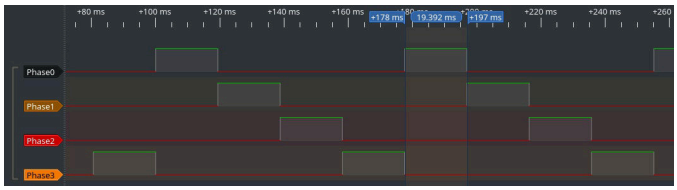
The Apple II floppy drive system basically consists of two cards: the drive contains one, called the "analog card," which handles the conversion from analog to digital; the second card is contained within the central unit, called the "interface card," and handles high-level coordination. Among other things, it includes a ROM for the bootstrap code and a series of chips needed to "serialize" and "deserialize" the bytes to be exchanged with the processor. In fact, the floppy emulator will have to pretend to be the analog card.

The drive remains on standby until the "drive enable" signal is deactivated. Activating the drive means putting it into read mode, and from that moment on, the data read by the head will begin to flow down the read wire. The digitization logic cleans up everything the head is able to pick up and returns a square wave signal consisting of a train of pulses, one for each "1" bit detected, with a spacing of 4Âµs.

The drive switches to write mode as soon as the "write request" signal is also activated. In this phase, the write head is turned on and its magnetic field overwrites whatever was there before. Switching or leaving the "write" unchanged determines the new content of the disk, i.e., whether to write a "1" or a "0" bit.



What remains is the management of the head positioning on the track, in fact by means of a stepper motor controlled directly by the four phases with sequential activation. To change tracks, it is actually necessary to activate two consecutive phases, probably to ensure greater precision in the motor-track alignment. The "intermediate tracks" were used for special copy protection mechanisms, which I was not interested in emulating.

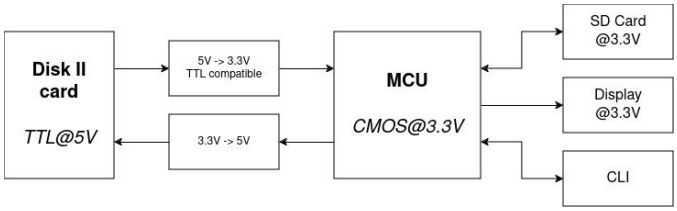


The emulator

To implement the emulator, I followed in the footsteps of the SDISK II project and used only a microcontroller. The disk images to be served are taken from a common FAT-formatted SD card and served on the disk interface.

To select the image and monitor the drive status, I first set up a serial command line interface, then a semi-graphical interface via OLED display and selector, which does not require a second computer.

The microcontroller chosen was based on AVR architecture - the same as the first Arduino series - because I had a development board with a more powerful variant to use for the first tests. In addition, the SDISK II is also based on AVR, and its code was a good starting point.



The entire "modern" section operates at 3.3V, a constraint imposed by the SD card and the display. The interface with the Apple II requires particular caution, not only because of the different operating voltages (5V vs. 3.3V), but also because of the incompatible logic standards (LVCMOS vs. TTL): even by rescaling the voltages, the two parts may not understand each other perfectly.

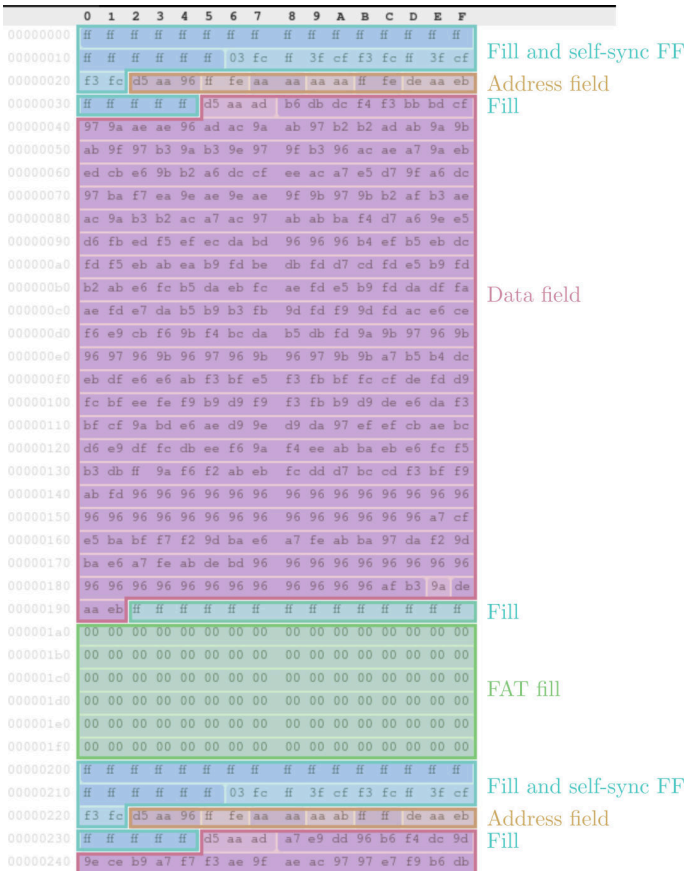
In the first experimental version, I settled for using what I had: inexpensive but slow level shifters to go down to the emulator and a few buffers to regenerate the emulator signal. Fortunately, there are integrated circuits suitable for this purpose, which I included in the PCB version.

I also took advantage of the LEDs on the development board to get visual feedback on the status of the player, for example, showing the position of the head.

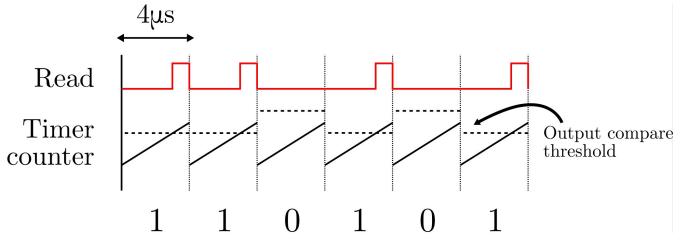
Reading

There are several formats that can be used to digitize Apple II disk images, but to maintain decent emulator performance,

I kept the .NIC format used on the SDISK II. This is a binary file that stores the raw bits physically read by the player's head. The file is structured in ascending sectors and tracks.
I have included a couple of extracts in the following pair of figures, highlighting the data structure and the aforementioned address field.
Including the gaps, the size of each sector in the .NIC is 416 bytes. To these are added 96 bytes of fill at 0, so as to keep each sector aligned to blocks of 512 bytes. This choice is dictated by the SD card read and write operating specification, which requires operating at multiples of a block, equal to 512 bytes.



Given the limited memory on the micro, it is not possible to download the entire .NIC image into RAM at the beginning of reading.
Instead, note the track and sector that must be supplied to the computer, simply position the SD card read address at the desired point and, every 4 microseconds, "pull out" a bit, generating a pulse or not.
This can be done entirely under program control, which is what was actually done in the original SDISK II project using some assembly routines.
In my implementation, I wanted to delegate the generation of the read signal to a hardware peripheral in the microcontroller, called a PWM timer. This peripheral allows you to independently generate a square wave signal, programming the period and the fraction of the period in which the signal must "remain low."
This peripheral is nothing more than a counter with a single associated output pin. Its maximum valueâ€"the periodâ€"and a threshold value can be programmed. As long as the counter value is less than the threshold value, the pin remains at 0. Once the intermediate value is exceeded, the output is set to 1.



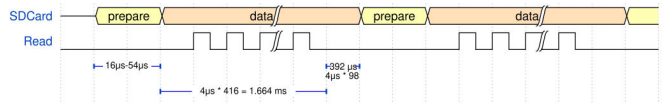
To generate a sequence of "1"s, simply program the peripheral with a period of 4 Âµs and a wait time of 3 Âµs. By setting the threshold value to a value higher than the period, the "match" will never occur and the output will remain zero, generating a "0".
At the end of each period, the software updates the threshold value according to the actual bit to be transmitted. It is no longer necessary to perform an active wait because the PWM peripheral is programmed to request an interrupt at the end of each period. At each interrupt, the program has just under 3 milliseconds to read a bit from the SD card and decide whether to enable the PWM output for the pulse.
The Drive II is not aware of the disk's sector division, so the current track is still read in its entirety and transferred to the interface card. Operations on a specific sector are performed in software, reading and interpreting each address field to decide whether to discard data from other sectors.
For the emulator, this translates into a very simple operating specification: as long as the drive enable signal is active, the emulator firmware cycles through reading all sectors of the same track.

Track control is performed via the four phase wires PHI0-PHI3, which in the original drive directly controlled the

stepper motor that positioned the head. In the emulator, track changes are managed by the main code; there is no need for interrupts because the signals are very slow, so much so that they can be observed with the naked eye via LEDs.

The "track control" routine monitors the four phase signals and appropriately increments the internal track counter according to the direction given. Track changes can occur at any time, even in the middle of reading a sector. In order not to overcomplicate the logic, I decided never to interrupt the reading of a sector and to apply the track change only when a new block is loaded from the SD card. In the real world, the data read during the track change would be invalid and discarded anyway.

Track control is incremental and without any feedback: in fact, each time the computer is turned on, it forcibly resets the position of the head by moving the motor blindly. This is the reason for the noise the disk drive makes when it is turned on. An initial reading experiment in which I had not implemented track management was interesting: the first track was read successfully, but when moving on to the next one, the software noticed the inconsistency through the address field. In this case too, an attempt was made to reset the head in the hope of realigning it with the correct track, unfortunately without success.



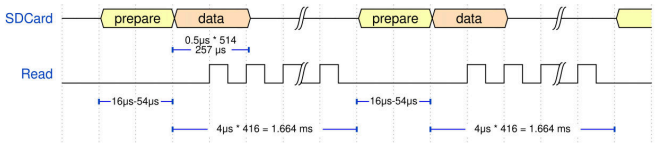
The most stringent time constraint in reading is imposed by the 4 Âµs bit period within the same sector. On the other hand, there is no rush to serve a new sector. To be clear, the Apple II software routines implement a timeout that ends reading if the requested sector is not readable within a certain time. But this time interval is sufficient to carry out a bit of bureaucracy. To serve a new sector, it is necessary to correctly address the correct block from the SD card, an operation performed by program control and requiring a few tens of Âµs.

As I said, in the first approach, I did not use sector buffering. The data is read bit by bit from the SD card every 4 Âµs and served immediately on the read line. The SD card requires that the entire 512-byte block be read, so at the end of the sector there are still 96 bytes to be ignored, which adds additional overhead before the next sector can be read. To reduce this downtime, I took advantage of the DMA peripheral.

DMA, or Direct Memory Access, is a component of the microcontroller that allows data to be exchanged between the memory and a peripheral without the direct intervention of the processor. In this case, the peripheral is the SPI that manages communication with the SD card.

With DMA, the overhead is reduced to the technical time required to set the address of the SD block to be read on the DMA, after which it will be loaded into RAM in about 300 Âµs. There is no need to wait for the DMA buffer to be fully loaded before starting to serve data on the read line, as the SD card is much faster.

I did not find it useful to further optimize the read phase; the choice to use DMA proved to be more useful for writing and formatting.



Writing

We need to differentiate between two distinct cases when writing to a floppy disk: overwriting a sector and formatting. Although both cases involve overwriting, in the first case the content of a single data field is altered, while in the second case the entire track is rewritten, including the address field. Formatting therefore operates on multiple sectors and has much more stringent time constraints, so much so that the first tinyDisk prototype was unable to support it.

Considering the simplest case, the overwriting of a sector, the following assumptions can be made:

-before starting to write a sector, the Apple routine always waits to receive the correct address field in order to align itself with the correct sector, then starts overwriting the data field that follows;

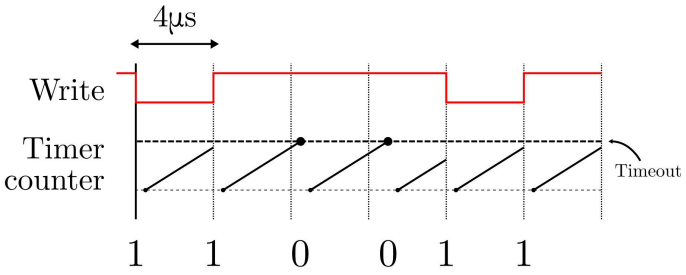
- the address field is never overwritten, it is used to align with the correct sector;

- 0xD5 is a unique and reserved byte, which can be used as an alignment marker in decoding the write signal;

- during the writing of a sector, the track will never be changed, as this would be rather counterproductive.

The strategy for converting the write signal into a binary stream is simple: to complete a byte, at least 8 symbols

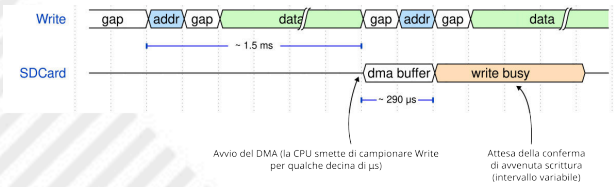
must be decoded. Each transition represents a "1." The "0," or the absence of a transition in a 4Âµs slot, requires a timer with a period slightly greater than 4us to account for the minimal fluctuations of the Apple II clock.



The Apple II software routine always starts by rewriting the self-synced FF sequence, which can be discarded. The first useful piece of information, which is essential for alignment, is the prologue of the data field, i.e., the byte 0xD5. Once this has been detected, all the following bytes can be saved in a buffer that will be loaded into the correct block of the SD card, overwriting the original .NIC file. At the end of the write operation, the Apple II returns to read mode to align itself, if necessary, with another sector to be overwritten.

If there are multiple sectors to overwrite, my firmware is unable to handle a new write until the "write back" to the SD card is complete. Fortunately, this time is not so long, and the Apple II has enough patience and does not go into error. Obviously, there is a timeout in case the address field to be overwritten is never detected.

Formatting is a different story. Since formatting redefines the geometry of the disk, in this case the track is completely overwritten without any wait between a data field and the address field of the next sector. Unfortunately, the microcontroller does not have enough RAM to store a track - a single sector already takes up 512 bytes of the 2K available.



The solution that seemed most reasonable to me was to try to implement a pipeline with the only sector buffer available. The principle is as follows: once the write request is registered, the buffer is populated by decoding the Write signal, as for sector overwriting. At the end, there is a short grace period represented by the writing of the gap between sectors, which the firmware uses to instruct the DMA peripheral to transfer the buffer to the SD card. At this point, writing to the SD card can proceed autonomously without the intervention of the CPU, which is free to decode the new sector arriving.

Even with a single buffer, it is rare for the reading of the new sector to overwrite a part not yet transferred to the mass memory, because there is an x8 factor between the output data rate of the SPI and the input data rate of the Drive II.

The critical factor, in fact, lies in the time it takes to finalize the writing to the SD card: after the buffer has been entered via SPI, these memories require additional time to perform physical erasure and overwriting, which varies depending on the model and age of the memory medium.

This additional delay has caused some headaches, but it can be mitigated by forcing the saving of sectors of the same track in physically contiguous areas of the SD card - effectively imposing a mild non-fragmentation of .NIC files - and allowing for optimization in the physical erasure and overwriting operations.

Final verdict and references

The goal of replacing the original drive with an emulator has been successfully achieved. I validated it in the field by testing a series of "exhibition" software disk images: games, applications, demos... I had no noteworthy problems with reading.

The same applies to writing, tested both by saving BASIC code from DOS and by arbitrarily modifying the contents of the sectors using CopyII+ software. Formatting still needs further investigation because it occasionally fails when dealing with particularly slow or worn SD cards.

Despite the simplifications, which limit the fidelity of emulation, the project proved to be a difficult challenge, and doing it at home was useful for learning something new. I have tried to summarize the most salient features of this project without going into excessive detail, hoping to arouse curiosity and interest in the subject. I leave it to the more willing to delve directly into the project code released under a free license, whether with the aim of creating their own version or proposing changes and improvements.

References

Project repository:
https://github.com/GLGPrograms/tinyDiskI

SDISK II, project home page:
https://tulip-house.ddo.jp/digital/SDISK2V1/english.html

Beneath Apple DOS:
https://mirrors.apple2.org.za/Apple%20II%20Documentation%20Project/Books/Beneath%20Apple%20DOS.pdf

The Amazing Disk II Controller Card, description of the disk controller by the designer of "Floppy Emu":
https://www.bigmessowires.com/2021/11/12/the-amazing-disk-ii-controller-card

Understanding the Apple II:
https://archive.org/details/Understanding_the_Apple_II_1983_Quality_Software/page/n19/mode/2up

Fig. 3 - Apple IIe computer (enhanced version)
(image: Bilby – source: Wikimedia Commons)

Beige is beautiful!
THE A400 Mini, a welcome return

by Carlo Nithaiah Del Mar Pirazzini

After a wave of micro-consoles including NES, SNES, Megadrive, PC Engine, and the return of micro-home computers such as C64 and Amiga, we now turn our attention to what must be the most interesting experiment to date: the Atari 400 Mini.

Based on Atari's range of 8-bit home computers, which hit the market in 1979, this small version is a "gem" created through a collaboration between Atari, Plaion, and Retro Games Ltd.

Atari is currently engaged in a sort of journey into the past, having released a collection of its games for modern consoles and the Atari 2600+, an (almost) full-size replica of its successful early 1980s console.

However, the Atari 400 is a less recognizable entity and could potentially be more difficult to sell.

Is it worth checking out? Let's find out.

A brief history

Released in 1979 alongside the Atari 800, the Atari 400 offered a more advanced audiovisual experience than its rivals thanks to its innovative use of coprocessor architecture, a first for a home computer.

While both machines used the same MOS 6502 CPU, the 400 was positioned in the market as the more affordable of the two and had a membrane keyboard and a single cartridge slot (the 800 had a mechanical keyboard and two cartridge slots, as well as expandable RAM and much more).

Although launched as personal computers, the Atari 400 and 800 were supported by a wide range of games, such as 1980's Star Raiders, a true killer application for the machine.

The fledgling Electronic Arts also actively supported the system; Danielle Bunten Berry's iconic MILE debuted on Atari's 8-bit range and took full advantage of the multiplayer controller ports.

As time went on, Atari revamped its range with the 1200XL, 600XL, and 800XL, and after Sam Tramiel took over the company and renamed it Atari Corporation, the XE series continued the lineage.

Compared to the 30 million units sold of the Atari VCS/2600, the 400/800 duo sold around four million units, not a commercial success in comparative terms, but still a relatively high success for the time.

ATARI 400 mini - Design and Controller

The 400 mini looks like a carbon copy of the real computer, with its angular Star Wars-inspired design, membrane keyboard, four yellow square keys, and red power LED.

Fig. 1 - The small "console" and the classic CX40 joystick

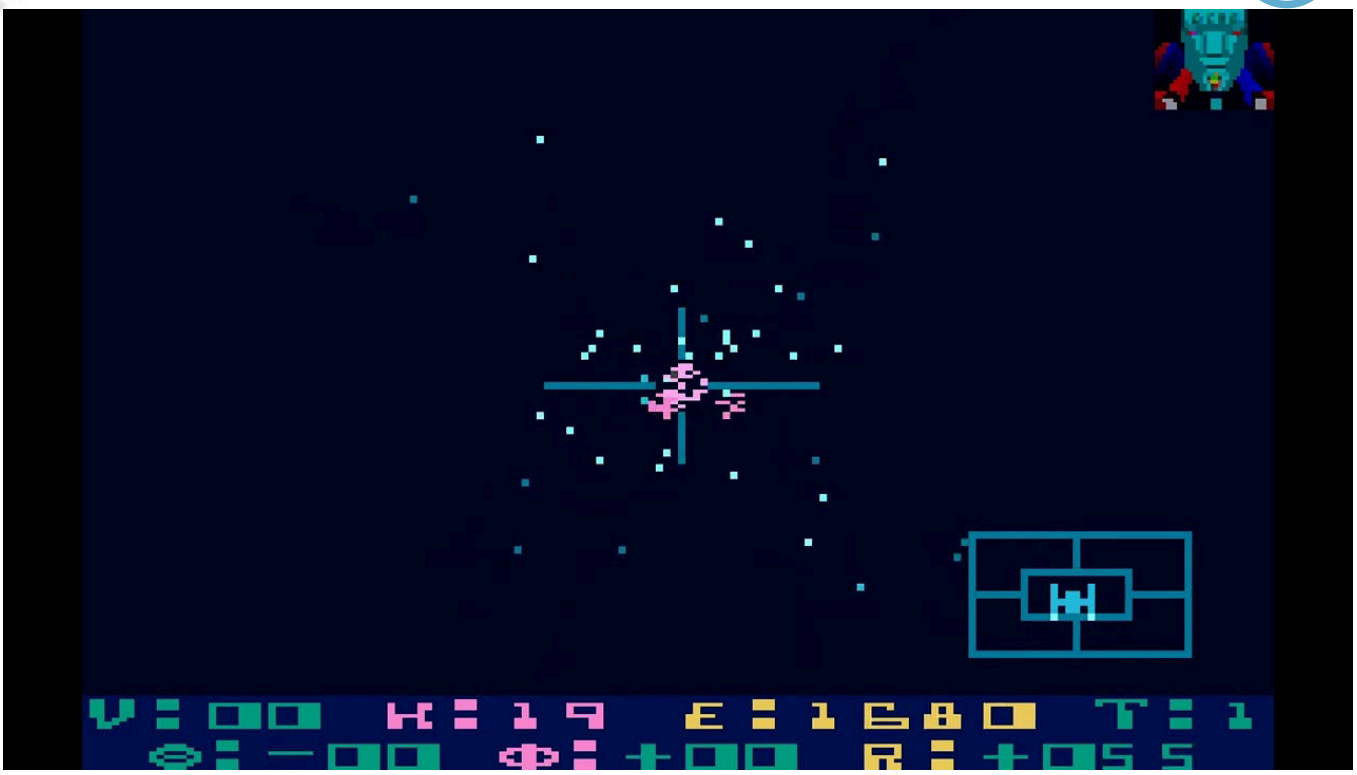


Fig. 2 - Star Raider: the killer application on ATARI 8Bit systems

The only visual difference is the word "THE400" replacing "Atari 400" on the front of the pseudo-cartridge slot.
The system measures 2.0x6.0x5.3 inches and weighs just over 200 grams. The keyboard is obviously non-functional, as is typical for miniature console emulations.
The front panel below the keyboard contains four USB ports with joystick functions, replicating the original computer (which means you can play MULE with four players).

On the back, there is a fifth USB port, an HDMI port, and a USB-C port for power. Of course, there is also a power button.

All 8-bit Atari systems have three custom coprocessor chips in addition to the 6502 CPU: ANTIC, GTIA, and POKEY. To show the power of these chips in emulation, the 400mini has an AllWinner H3 processor (basically a quad-core Cortex A7), 256 GB of RAM, and 128 GB of

Fig. 3 - Et voilà... The 400

Fig. 4 - The CXStick supplied in all its "plumpness"

storage, more than enough to save games and set up the system to its full potential.

The joystick is called TheCXStick and resembles the original CX40 that we all love so much. I must admit that it is an excellent stick, with precise movements and no unnecessary clicks. The fire button also feels good. Unlike the original, this one obviously has a USB connector at the end of the cable. It also has an additional set of buttons (7 in total), which are well integrated into the design and not too intrusive. Four are directional buttons hidden under the orange markings. The base has a button on the shoulder hidden in the upper left corner. The Black Menu and Home buttons protrude slightly from the rear panel.

The package also includes a beige HDMI cable, a USB-C power cable, and a quick start guide. The AC adapter for connecting the USB-C cable is missing.

Let's talk about games

The system comes with 25 titles. Many are timeless classics, including Bristles, Bruce Lee (renamed Lee), Capture the Flag, Fip and Flop, Miner 2049er, Star Raiders, and The Seven Cities of Golds.

The Atari 8-bit was a perfect machine for conversions of arcade classics, and here we are spoiled for choice: Asteroids, Battlezone, Berzerk Centipede, Crystal Castles, Millipede, and Missile Command are perfect on these machines!

Some classics are missing due to lack of rights (see Nintendo titles such as Mario Bros and Donkey Kong).

There are 25 carefully selected and highly playable titles, and you will find all the manuals on the page https://retrogames.biz/games/the400-mini/.

I tested the mini console on a 55-inch TCL TV in game mode and found that the controller has a very slight latency. Less than a quarter of a second and still within reasonable limits for a plug-and-play console.

None of the games tested had bugs, errors, or crashes.

The USB port on the back allows you to load additional titles via a FAT32-formatted USB stick.

This means you can access the entire game library available for the systems, as well as wonderful new productions.

The400 Mini accepts ATR, COM, and XEX file formats. It automatically detects the type of system the game runs on and whether it requires expansions and setup adjustments.

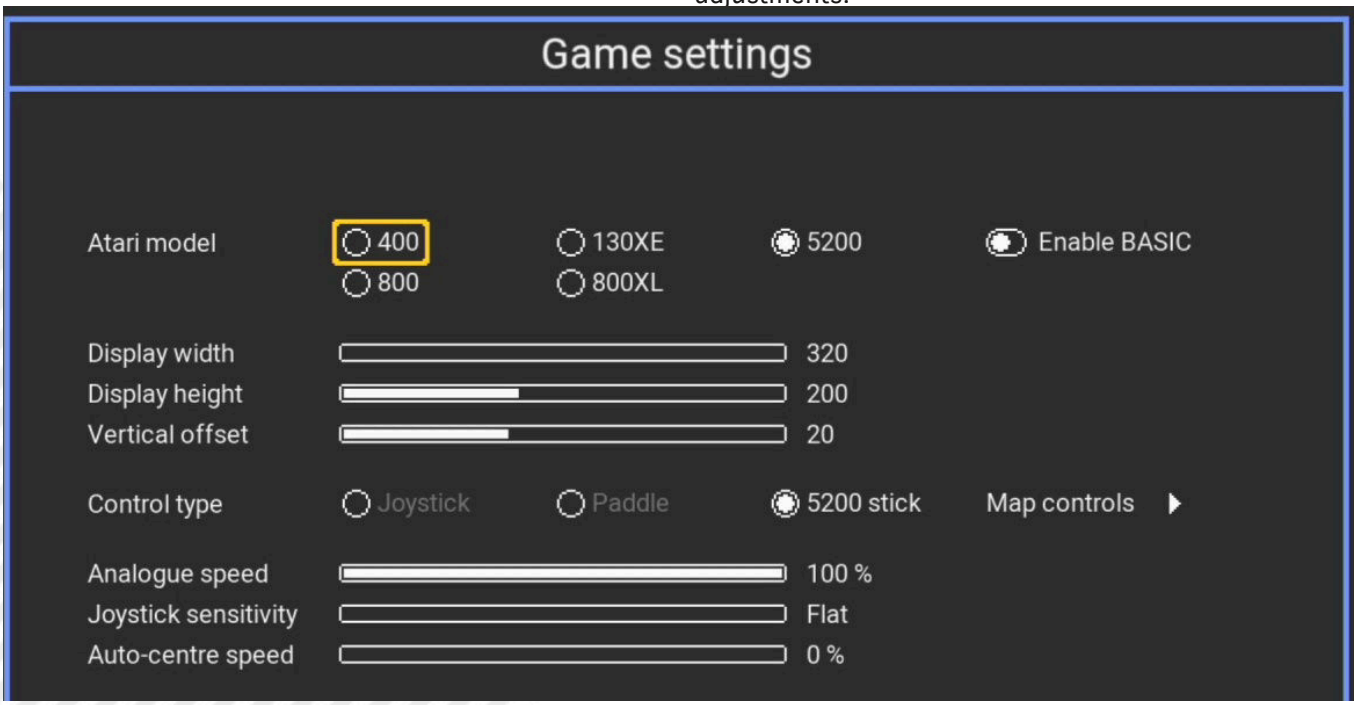


Fig. 5 - The settings menu for games

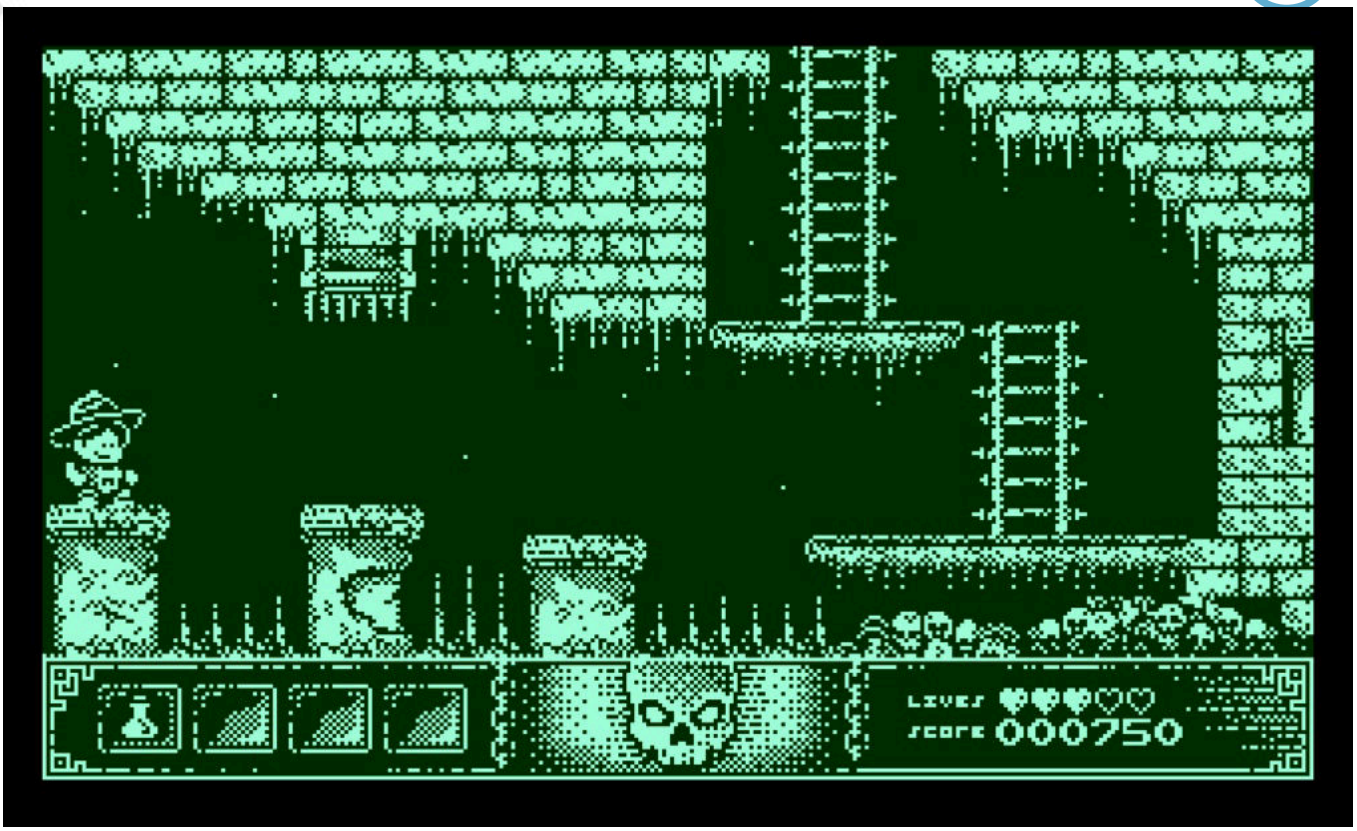


Fig. 6 - Tony Montezuma's Gold runs great on THE400 Mini

Pressing MENU during gameplay opens a screen to configure the ATARI model (400, 800, 800XL, 130 XE, or 5200, with or without Basic enabled), the height and width of the display, the vertical offset, and a way to remap the controls.

Programming, multi-disk games, and some bugs

Speaking of BASIC, you can program on THE400 mini by formatting a USB flash drive, enabling BASIC in the settings, connecting a USB keyboard to a free port, and loading THE400_BASIC.

This gives you access to the system and a disk to save programs.

Tests with modern keyboards have been mixed. The lack of a "real" keyboard is noticeable in terms of settings and comfort. The setup is obviously not real.

The lack of an included keyboard is also the reason why the system lacks some important titles, such as the original Star Raiders and Rescue on Fractalus! But with a little patience, you can make it work.

The system also supports multi-disc games with an ingenious solution: select the first with the fire button and the second with the S (back) button, then press HOME to load.

When it's time to insert the second or third disc, hold down the HOME and DOWN directional buttons. It sounds cumbersome, but it works perfectly.

The verdict

There's no denying that the Atari 400 mini fully achieves its goal: it presents a miniature replica of an 8-bit classic, packed with titles and enhanced with several modern features, the ability to add new titles, and a good supporting joystick.

The pre-installed titles are also a good choice: of course, it would have been nice to see the Nintendo or Pacman classics and even the aforementioned Star Raiders, but this problem can be overcome by adding them to a support USB stick.

Some doubts arise about the target audience; while the VCS or Nintendo consoles can rely on a large fan base, the Atari 8-bit family is less well known, even though its games are of incredible quality thanks to superior hardware.

If you grew up with this system, the 400 mini is the product of your dreams. Excellent emulation, the ability to load everything, and modern updates such as save systems and rewind function.

VERDICT: 8/10

Product website: https://retrogames.biz/

Compiling with Libdragon

by Takahiro Yoshioka – translated by Carlo Nithaiah Del Mar Pirazzini

Let's continue our exploration of Libdragon and this time we'll try the Compiling process.

We'll pass the ball to Takahiro.

Introduction

Compiling (or building) is the process of taking source code and converting it into an executable binary file.

There are a few intermediate steps such as preprocessing and linking, but for simplicity we will refer to the entire operation as "Compiling". On this page, we will look at how to start compiling Libdragon projects.

Preparing the Mobile Window

The first step is to get your Docker engine running. You can do this by opening Docker Desktop.

If you don't have Docker open or haven't created a Libdragon container, you may get an error like this when you run Make:

```
libdragon make
```

This guide assumes that you have followed the steps outlined in the Setting Up Lib section.

Running the build command

The build command is pretty straightforward. Simply open your terminal/CMD, navigate to the correct directory, and run this command:

This will take all your source code and produce a bunch of output files including your Nintendo 64 ROM.

If you get an error, *** No rule to make target '/include/n64.mk'. Stop.

It means you are using a normal make file instead of libdragon make.

Explanation of output files

Once the project is compiled, there will be several new files in the project folder.

The main file you will be interested in is the hello.z64 file, which will be your N64 ROM file. You can put it into your favorite emulator and run it to see how your program works.

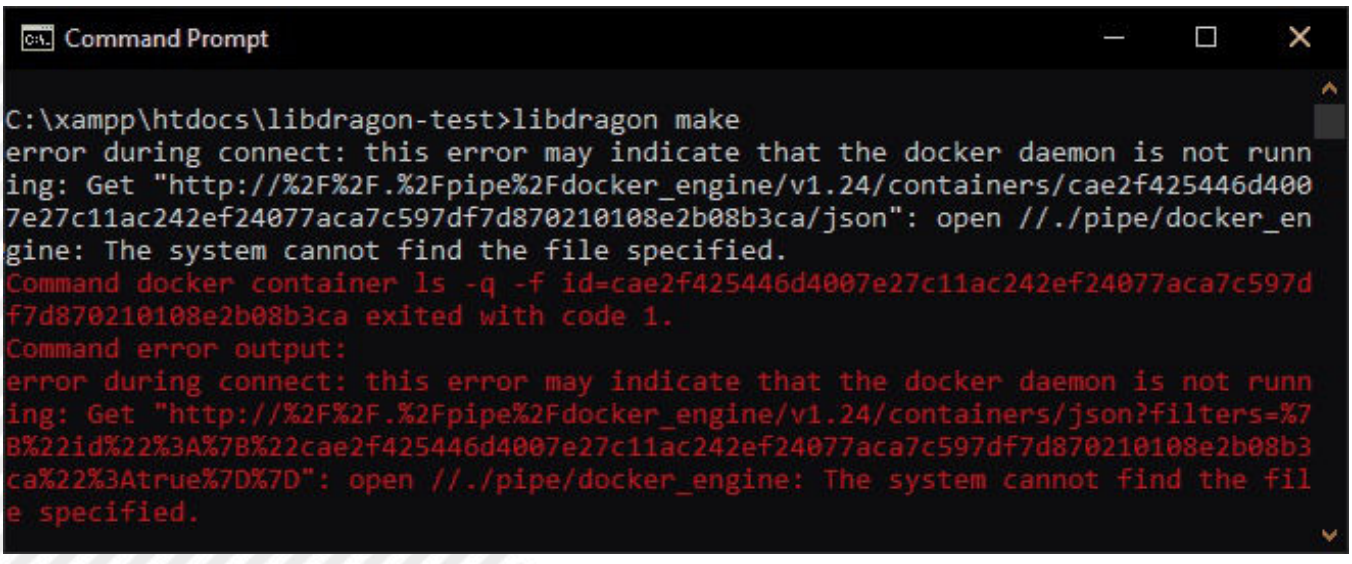


Fig. 1 - Example of command output: libdragon make

The rest of the files are placed in the ./build/ directory. Assuming you are working with the skeleton project, they are as follows (it will be different for more complex projects):

- hello.elf - This is the linkable executable file. It is a binary file that contains object code, shared libraries, and executable code.

- hello.elf.bin and hello.elf.sym - I'm not sure what these are, but they appear to be files that accompany the main .elf file containing binary machine code and symbol information.

- hello.map - Text file containing information about how memory is laid out and the various data/code sections of the executable.

- main.d - Dependency file. Shows which files the build depends on. In this simple program, it will only mention main.o, from which main.c was created.

- main.o - The object file for main.c. It contains the compiled code used for output in the compilation phase of the build (before linking).

Keep in mind that files named "hello._" refer to the build as a whole, while those named "main._" refer to individual source files.

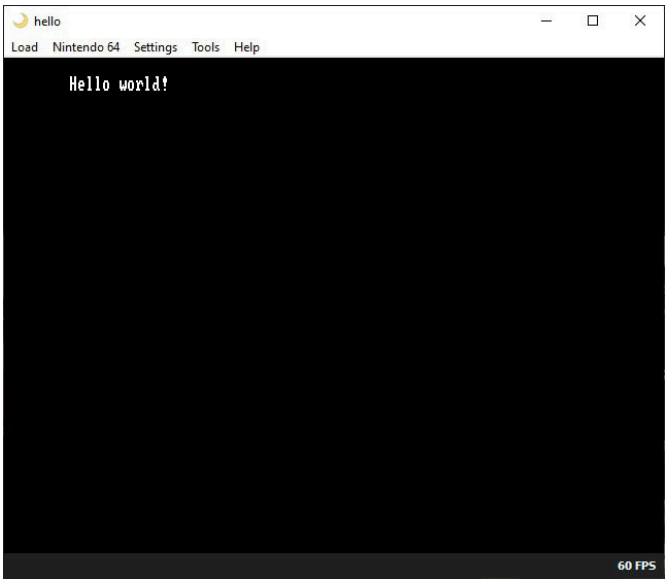
If you use Git for version tracking, it may be wise to add these files to your .gitignore exclusion list as they will change often, take up space, and are not very relevant to the development of your project. They can be useful for debugging if you know how to do so. Just add this:

```
build/
*.z64
```

Running the ROM

Run the ROM in an emulator of your choice. I tend to use Project 64, but in this case I will use Ares.

You should get something like this:



That's it, your first Libdragon N64 ROM!

PunyInform - Tutorial 3

by Fredrik Ramsberg - translated and adapted for RMW by Gianluca Girelli

Programming with PunyInform 1: The Basics

PunyInform is a tool for writing text adventures for both 8-bit computers and newer computers. If you have already installed PunyInform (see previous issues for all the information you need), it's time to start programming your first game. This tutorial will guide you through the first steps of writing a text adventure using PunyInform. The complete source code for the sample game can be downloaded from the link at the bottom of the page. This is the first in a series of tutorial articles. This first version of the game implements some environments and an object with which the player can interact, but there is no way to win or lose: this will be covered in the next tutorial.

Let's start writing a new game - Fig. 1

Compiler directives

The first two lines of code are compiler directives, which help make the game file as small as possible. You will generally want to have these lines at the top of the source code for every PunyInform game you write.

Comments

In lines 3, 4, 9, 16, and 20, we can see how to insert comments to make the code more readable: an exclamation mark means that the rest of the line is a free-form comment (unless the exclamation mark is followed by % and the line appears at the beginning of the file, in which case it is treated as a compiler directive).

Story and title

The second step is to declare the name of the game/story and the title, something similar to a slogan. We always have to define the "story name" and, in most cases, we

```
1  !% -~S
2  !% $OMIT_UNUSED_ROUTINES=1
3  !% $ZCODE_LESS_DICT_DATA=1
4
5  Constant Story      "The Mansion at the Edge of Town";
6  Constant Headline   "^An interactive nightmare.^";
7
8  ! Constants to control PunyInform are defined here
9  Constant DEBUG;
10 Constant OPTIONAL_FULL_DIRECTIONS;
11 Constant INITIAL_LOCATION_VALUE = Entrance;
12
13 Include "globals.h";
14
15 ! PunyInform extensions and entry point routines are placed here
16
17 Include "puny.h";
18
19 ! Verb grammar and actions go here
20
```

Fig. 1 - Let's start writing a new game

will also want to define a title. Keep in mind that ^ can be used in strings to indicate newline characters. (Also, ~ can be used to indicate double quotation marks, but it is not used in this code.)

Defining constants

The next section defines the constants that control PunyInform. More information about constants can be found in the manual included in the folder of each PunyInform release, under Customizing the Library. For some constants, such as INITIAL_LOCATION_VALUE, you need to provide a value. For other constants, however, it is sufficient to simply define the constant (effectively giving it the value 0). The PunyInform library checks whether the constant has been defined or not and modifies certain aspects of its behavior accordingly. Constant DEBUG; is used to enable debug mode, which means that a series of debug verbs will be available, so you can type things like "GONEAR TEDDY BEAR" to teleport yourself to where the teddy bear is, or "ACTIONS" to see what actions are triggered by what you type on the keyboard. It's very useful to have this during game development and testing, but remember to disable it before releasing the game to the public. Constant OPTIONAL_FULL_DIRECTIONS; enables the northwest, northeast, southwest, and southeast directions. If you want to use multiple directions (instead of the standard four: north, south, east, and west), you must define this constant. Keep in mind, however, that it makes the game a little bigger and a little slower.

Costant INITIAL_LOCATION_VALUE = Entrance; tells the game where the player will start their adventure. Also keep in mind that every instruction or statement must end with a semicolon.

Include globals.h

After defining all the constants we need, it's time to include the "globals.h" file. This sets up the global variables, constants, etc. needed by the PunyInform library.

Library extensions and entry point routines

It's time to include all the library extensions we want to use (see Extensions in the manual for more information) and write all the entry point routines we need (no need to read it now, but the supported ones are listed under Entry Point Routines in the PunyInform manual, and what they do is described in The Inform Designer's Manual under Entry Point Routines). [NDR: the game philosophy is based on the official Inform6 language manual, "The Inform Designer's Manual" written by Graham Nelson, now in its 4th edition. You can find the link at the bottom of the page.]. In this particular game, we have decided not to use any extensions or entry point routines. We can always change our minds later, if necessary.

Include puny.h

At this point, we include the second and last file in the library, called "puny.h." This file, in turn, will include all the other files in the library.

Verbs, grammar, and actions

Finally, we define new verbs, extend existing ones with new grammar, and define new actions. We'll leave this section empty for now.

Game objects - Fig.2

Now it's time to define the game objects. Typically, you create one object for each location in the game and one object for each thing the player can interact with. A thing can be something tangible like a key or a car, but it can also be something like air, water, the coastline seen from an airplane, or even a smell or a sound. Objects can also be used for abstract concepts and game concepts such as thoughts, conversations, menus, etc. Objects have a hierarchy, where one object can be "inside" another object. The meaning of "inside" depends on the properties of the container object (the so-called parent) and the contained object (the so-called child). Being in another object usually means that the child is in the parent, on the parent, held by the parent, worn by the parent, or part of the parent.

Rooms

A room (also called a location) is an object that has no

```
! The game objects begin here
Object Entrance "Entrance"
    with
        name 'front' 'door' 'cubicle' 'stairs',
        description "You are in the entrance area of the mansion. A wide set of
            stairs lead up. There is a small cubicle in the southeast corner,
            while the front door is to the south.",
        u_to UpstairsHallway,
        se_to Cubicle,
        in_to Cubicle,
        s_to "The front door has apparently been nailed shut.",
        cant_go "There's nothing of interest in that direction.",
    has light;

Object Cubicle "Cubicle"
    with
        initial "You squeeze in through the small opening, hitting your head
            against the doorway in the process.^",
        description "It's quite cosy in here. There are some marks on the wall
            where a desk probably used to stand. The exit is to the north.",
        out_to Entrance,
        n_to Entrance,
    has light;
```

Fig. 2 - Game objects

parent and in which the player can find themselves. Let's start by defining our first room. First, type Object followed by the internal name of the object. In this case, we choose to call it Entrance. Next comes the object name string. This is the name that will be printed on screen for this object in the game, enclosed in double quotes. For rooms, it is common practice to capitalize the printed name. You will notice that the internal name and printed name of an object are often identical or very similar. However, the internal name can only consist of the letters A-Z and a-z, digits, and underscores, with no spaces or other special characters. And while many locations may share the same printed name (for example, a series of locations might have the same printed name of "Forest" or "Moors"), the internal name of each location must be unique. We have chosen to put Object Entrance "Entrance" on the same line and then break before the with statement, but this is only a stylistic choice: you can have line breaks anywhere you want in an object definition.

Properties

Next, the object's properties are displayed, and the keyword with marks the beginning of this section. A property can contain one or more 16-bit values. Each value is often used to store a number, an object ID, the address of a dictionary word, the address of a string, or the address of a routine. If used as a number, it can contain any integer between -32768 and +32767. After the last value of a property, you must type a comma to indicate that this property ends here. Keep in mind that an object only gets the properties that are defined for it. This means that if you want to create a north exit in a room later in the game, but there shouldn't be one from the start, you must include n_to 0 in the object definition. This ensures that n_to exists and has a value, so that the value can be changed later. First, we give the Entrance object the name property and assign it four values, each (the address of) a word in the dictionary. When we give a room the name property, it means that if the player uses one of these words in this room, the game should print "You don't need

to use that word in this game." or something similar. This can be used to help the player focus on what is important in the game. Remember that dictionary words are always enclosed in single quotes. If a dictionary word consists of only one character, you must add // after that character to distinguish it from a character, which also uses single quotes. So: 'n' is a character, while 'n//' is a dictionary word. Now let's assign the description property, which is mandatory if the object is a room. The description can be a string or a routine that prints text, and this is printed or executed when the player enters the room or types LOOK. As you can see, a string can contain end-of-line characters, and therefore the next line can start with tabs or spaces: all this is contracted by the compiler into a single space character. Next, we choose to specify the exits from this room. For this we use the properties n_to, s_to, e_to, w_to, u_to, d_to, in_to, and out_to. If we have defined the OPTIONAL_FULL_DIRECTIONS constant, we can also use ne_to, nw_to, se_to, and sw_to. Each output can have a single value (not a list of values), and this value is usually (A) a room or (B) a string to print if the player tries to go in that direction. We then define cant_go, which is a message to print if the player tries to go in a direction for which we have not specified a value. If we do not define the cant_go property, the standard message "You can't go that way." will be displayed.

Attributes

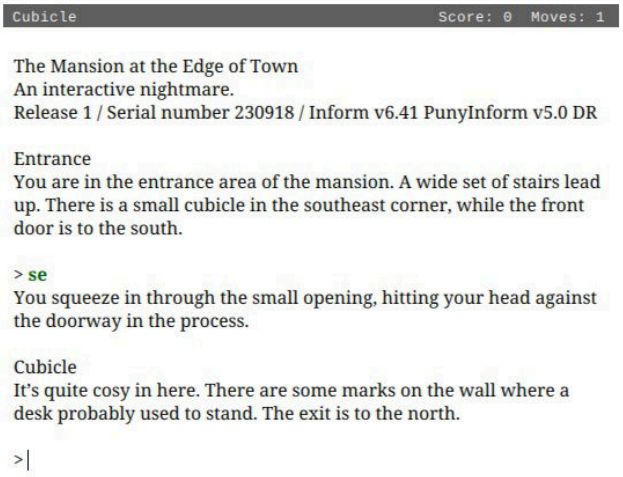
Now let's look at the keyword has, which means we're going to define the attributes of this object. An attribute is a single-bit flag, which can only be true or false. Unlike properties, all attributes exist for all objects, so even if you don't include an attribute in the object definition, you can still modify it later in the game. In this case, we give the room the attribute light, meaning that the room itself provides light. This means that the player can see in the room (meaning they can see what's in it and interact with any objects) without carrying their own lamp. Next, we insert a semicolon, which marks the end of the object definition.

```
45  Object UpstairsHallway "Upstairs Hallway"
46      with
47          description "Nothing much to see here really. There are some doors, but
48              they have all been nailed shut. The stairs lead down to the west.",
49          d_to Entrance,
50          w_to Entrance,
51      has light;
52
53  Object -> Painting "painting"
54      with
55          name 'painting',
56          initial "Hanging on the west east wall is a dark oil painting.",
57          description "It's a lovely old painting of a crying clown.",
58          after [;
59              Examine:
60                  give self general;
61          ],
62          short_name [;
63              if(self has general) {
64                  print "painting of a clown";
65                  rtrue;
66              }
67          ];
68
```

Fig. 3 - Object called UpstairsHallway

The Cubicle object

This object brings only one new thing to the table: the initial property. When used in a room, this property is printed immediately before the room name when the player enters the room. Here's what it looks like:



Upper floors and normal objects (not rooms)

Remember that we wrote in the Entrance object that going up (u_to) would lead to UpstairsHallway? This means that we have to define an object called UpstairsHallway somewhere in the code. See Fig. 3

This location does not illustrate any new new for us, but exhibition how sometimes times it convenient to have two exits different that lead to same location. With the text saying that the stairs go down to the west, it makes sense to have both d_to and w_to leading to the room at the bottom of the stairs. Now here comes our first object that is not a room. We're talking about a painting. By putting an arrow (->) after Object, we tell the language that this object is inside the last object that didn't have an arrow, so the painting is in the UpstairsHallway room. Next come the properties. Remember that the description property is mandatory for rooms? This is not the case for normal objects. In fact, there are no mandatory properties for regular objects. Generally, we want the player to be able to refer to the objects they encounter (not rooms, but pretty much all other objects). The easiest way to enable this is to give the object a name property and let it contain a list of dictionary words that are nouns and perhaps adjectives that can be used to refer to this object. In this case, we'll simply settle for 'painting'. And remember to always enclose dictionary words in single quotes. Next comes the initial property. For normal objects, this is used to describe the object until the player has picked it up, so we can use it to describe the painting in a way that only makes sense in the place where the painting is located at the beginning of the game. Next comes the description property. For normal objects, this is printed if the player examines the objects. It can be a string or a routine that prints text. Next, we define the after property. This is a routine (it starts with [; and ends with]) that is called after an action has succeeded. In this routine, you type the name of an action that the player can perform where this is the main object (also called noun), a colon, and the code you want to execute if this action has been taken. All possible actions are listed in the manual under actions, and while playing in DEBUG mode, all actions performed by the language can be viewed after typing the ACTIONS command. In this case, we want to execute some code after the player has examined the painting, and this is made possible by the instruction give self general; Let's analyze it. give is an instruction to set an attribute for an object. self is the object on which this code is executed (so in this case we could have typed Painting and achieved the same effect). general is a multipurpose attribute. We decide what it means for each object. In this case, we have decided that for the painting it means "Has the player examined the painting?". The attribute is reset at the beginning of the game, and as soon as the player examines the painting, it is set to "general." At the end of an after rule like this, we can usually choose to return true or false, where true means "We printed a suitable message, so the action routine should not print one." However, Examine is a special case. Since all it does is print something, it prints its message before the after routine is called, and it doesn't matter what we return. Note: routines in properties return false at the end, so if you don't explicitly return anything, they return false. Named routines (see below) return true at the end. Now let's examine short_name. This is an opportunity to override the name provided in double quotes on the first line of this object. It can be a string or a routine. If it is a

routine, the routine is called, and if the routine returns true, it is assumed that the complete short name of the object has been printed, and nothing else is printed. If it returns false, the normal name of the object is printed. Here we check if the painting has the general attribute set (which means that the player has examined the object), and if it does, we print the string "painting of a clown" and return true. Otherwise, we reach the end of the routine, which means that the routine returns false and the normal name of the object is printed instead. Note: true is just a synonym for 1 and false is a synonym for 0. You can return any value using the return statement, but there are shorthand forms for when you want to return true or false instead. return true; and return 1; and rtrue; are therefore equivalent. Just as return false; return 0; and rfalse; are equivalent. Note: An if statement can execute a single statement (if(x > 5) print "X is big!";) or multiple statements enclosed in curly brackets (if(x > 5) { print "X is big!"; y = x; }). Line breaks and tabs or extra spaces are added only for readability and do not affect the execution of the program. Let's see this code in action.

```
Upstairs Hallway                                Score: 0  Moves: 5

> gonear painting
Upstairs Hallway
Nothing much to see here really. There are some doors, but they have
all been nailed shut. The stairs lead down to the west.

Hanging on the west east wall is a dark oil painting.

> get painting
Taken.

> i
You're carrying a painting.

> examine the painting
It's a lovely old painting of a crying clown.

> i
You're carrying a painting of a clown.

>
```

These are all the objects in the game, at least for this initial version.

The Initialise routine

There is only one small mandatory routine left. Every PunyInform game must provide a routine called Initialise. See Fig. 4

Writing a "named routine" simply means starting with [then typing the name of the routine, any necessary local variables, and a semicolon. Then you write the code and end with];

Initialise is called by the library when the game starts. In this case, the only code we need here is the print statement to print the initial text when the game starts.

Below is the complete code for this "minimal" game, which will be refined and expanded in subsequent articles.

Have fun!

```
!% -~S
!% $OMIT_UNUSED_ROUTINES=1
!% $ZCODE_LESS_DICT_DATA=1

Constant Story      "The Mansion at the Edge
of Town";
Constant Headline   "^An interactive
nightmare.^";

! Constants to control PunyInform defined here
Constant DEBUG;
Constant OPTIONAL_FULL_DIRECTIONS;
Constant INITIAL_LOCATION_VALUE = Entrance;

Include "globals.h";

! PunyInform extensions and entry point
routines are placed here

Include "puny.h";

! Verb grammar and actions go here

! The game objects begin here
Object Entrance "Entrance"
  with
      name 'front' 'door' 'cubicle' 'stairs',
     description "You are in the entrance
area of the mansion. A wide set of
         stairs lead up. There is a small
```

```
69  ! And finally, the Initialise routine goes here
70  [Initialise;
71      print "^^You and your friends went out to the old abandoned mansion on the
72          outskirts of town. Once in the house, you remember getting separated
73          from the group, and then all went black. As you come to, several hours
74          have passed, you seem to be alone, and everything is quiet.^^";
75  ];
76
```

Fig. 4 - The Initialise routine

```
cubicle in the southeast corner,
        while the front door is to the south.",
    u_to UpstairsHallway,
    se_to Cubicle,
    in_to Cubicle,
    s_to "The front door has apparently
been nailed shut.",
    cant_go "There's nothing of interest
in that direction.",
  has light;

Object Cubicle "Cubicle"
  with
    initial "You squeeze in through the
small opening, hitting your head
        against the doorway in the process.^",
    description "It's quite cosy in here.
There are some marks on the wall
        where a desk probably used to stand.
The exit is to the north.",
    out_to Entrance,
    n_to Entrance,
  has light;

Object UpstairsHallway "Upstairs Hallway"
  with
    description "Nothing much to see here
really. There are some doors, but
        they have all been nailed shut. The
stairs lead down to the west.",
    d_to Entrance,
    w_to Entrance,
  has light;

Object -> Painting "painting"
  with
    name 'painting',
    initial "Hanging on the west east wall
is a dark oil painting.",
    description "It's a lovely old painting
of a crying clown.",
    after [;
        Examine:
        give self general;
    ],
    short_name [;
        if(self has general) {
            print "painting of a clown";
            rtrue;
        }
    ];

! And finally, the Initialise routine goes here
[Initialise;
  print "^^You and your friends went out to
the old abandoned mansion on the
    outskirts of town. Once in the house,
you remember getting separated
    from the group, and then all went black.
As you come to, several hours
    have passed, you seem to be alone, and
everything is quiet.^^";
];
```

LINK UTILI

- https://drive.google.com/drive/folders/1iYRoJYkPXx7RYnd-tl0gT74L2Sa04mSp
- https://www.inform-fiction.org/manual/about_dm4.html

From numbers to digits (for C64 - for beginners)

by Eugenio Rapella

Let's suppose we have a natural number X consisting of four digits (in base 10). For some reason (for example, to solve a "Quesito con la Susi" puzzle game (usually published in an Italian puzzle magazine), we need to obtain the four digits separately.

Ultimately, starting from X, we want to obtain the four variables U, D, C, M containing respectively the units digit, the tens digit, the hundreds digit, and the thousands digit (so, if X=7235, we will have U=5, D=3, C=2, and M=7).

Here is one possibility for our C64 (PROG1):

```
10 v1=int(x/10):u=x-10*v1
20 v2=int(v1/10):d=v1-10*v2
30 m=int(v2/10):c=v2-10*m
```

To understand how the program works, just follow an example. If X=7235, we have X/10 = 723.5 and V1=INT(723.5)=723.
Therefore, U=7235-10*723=5 as desired. Now V1/10=72.3, so V2=72 and D=723-10*72=3. Finally, V2/10=7.2, so M=7 and C=72-10*7=2.

Another possibility is to convert X into a string and use the MID$(..) function to extract a single letter from the string. Finally, we will obtain the numerical value with the VAL(..) function.

Here is PROG2:

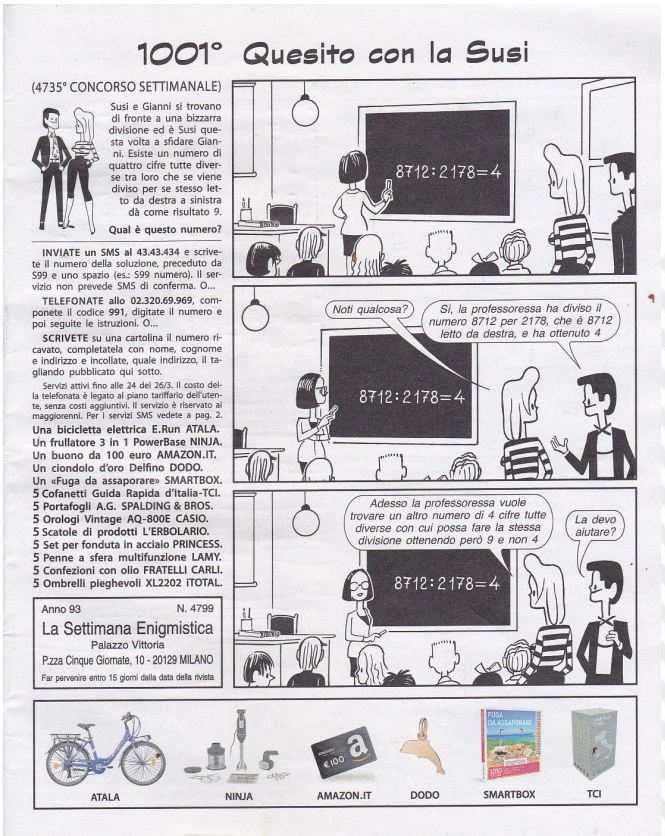
```
10 x$=str$(x)
20 u=val(right$(x$,1))
30 d=val(mid$(x$,4,1))
40 c=val(mid$(x$,3,1))
50 m=val(mid$(x$,2,1))
```

Note that the STR$(..) function converts the numeric value into a string by adding a space at the beginning. So, if X=7235, we will have a string of five characters: X$=" 7235".
RIGHT$(X$,1) returns the last character of the string ("5"), so U=VAL(RIGHT$(X$,1)) becomes the numeric value 5.
Since the MID$(A$,H,K) function returns a string consisting of K characters from string A$ starting at position H (so, if A$="MAGAZINE", we will have MID$(A$,3,4)="GAZI"), D will be 3, C will take the value 2, and M will take the value 7.

A comparison between PROG1 and PROG2 in terms of speed did not reveal any significant differences; the two programs can be easily modified if the number of digits is different from four.

And here we are with a new "Question with Susi," again from "La Settimana Enigmistica":

The teacher shows that if we divide 8712 by 2178, which is the first number written "backwards," we get exactly 4. The request is to find a four-digit number that, operating in a similar way, gives exactly 9 instead of 4.



Here's how our C64 (PROG3) proceeds:

```
10 for k=1000 to 9999
20 v1=int(k/10):u=k-10*v1
30 v2=int(v1/10):d=v1-10*v2
40 m=int(v2/10):c=v2-10*m
50 h=1000*u+100*d+10*c+m
60 if 9*h=k then print k,h
70 next
```

For each K ranging from 1000 to 9999, the individual digits are extracted from K and, at 50, the number H is constructed, which is K ... written backwards.
At 60, a check is made to see if K/H is exactly equal to nine, in which case K and H are printed.

An alternative to PROG3 is the following PROG4 where, starting from the individual digits, the two numbers K and H are constructed. Note that the thousands digit, M, starts from one so that K is, in effect, a four-digit number:

```
10 for m=1 to 9:for c=0 to 9
20 for d=0 to 9:for u=0 to 9
30 k=1000*m+100*c+10*d+u
40 h=1000*u+100*d+10*c+m
50 if 9*h=k then print k,h
60 next:next:next:next
```

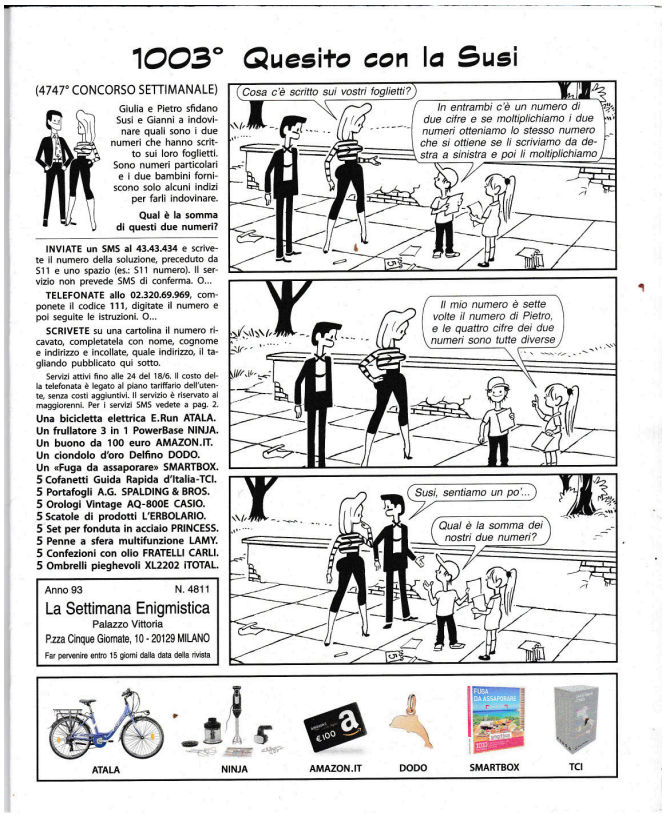
Both programs review many more candidates than necessary; obviously, readers of "La Settimana Enigmistica" are not expected to solve the problem with the help of the C64, and there must be some shortcuts (as usual, my Commodore forbids me from "spoiling" its programs by giving hints).

However, our PROG3 and PROG4 have the advantage of quickly providing other information:
- When a solution is found, processing continues anyway. In the end, there is only one printout, confirming that the solution is unique.
- The original text of the question states that the digits must be different from each other. The programs ignore this requirement: the solution is still unique even without this limitation.
- If the instruction IF 9*H=K THEN PRINT K,H is changed to IF 4*H=K THEN PRINT K,H, only the pair of numbers 8712, 2178 is found. Even in the example given at the beginning, the solution is unique.
- If, instead of 9 or 4, I run the program with one of the values 2, 3, 5, 6, 7, or 8, no solution is found.
- Of course, with IF H=K THEN PRINT K,H, all four-digit "palindromic" numbers are obtained: 1001, 1111, 1221, ... ,9889, 9999.

While I was writing this article, a new "Question with Susi" appeared in a new issue of "La Settimana Enigmistica":

Susi chats with Giulia and Pietro.
Susi: "What's written on your pieces of paper?"
Pietro: "Both have a two-digit number, and if we multiply the two numbers, we get the same number as if we wrote them from right to left and then multiplied them."
Giulia: "My number is seven times Pietro's number, and the four digits of the two numbers are all different."
Pietro: "What is the sum of our two numbers?"

Let's call the number on Pietro's piece of paper X1. If A



represents the tens digit and B the ones digit, we will have

X1=10*A+B. Similarly, if Y1 represents the number on Giulia's piece of paper, with C the tens digit and D the ones digit, we will have Y1=10*C+D.

PROG5 works by following the text very literally:

```
100 for a=1 to 9:for b=1 to 9
110 for c=1 to 9:for d=1 to 9
120 if a=b or a=c or a=d then 220
130 if b=c or b=d then 220
140 if c=d then 220
150 x1=10*a+b:y1=10*c+d
160 if y1<>7*x1 then 220
170 x2=10*b+a:y2=10*d+c
180 if x1*y1<>x2*y2 then 220
190 s=s+1:print"sol n. ";s
200 print"x1=";x1;"y1=";y1
210 print"prodotto:";x1*y1
220 next:next:next:next
```

At 100-110, the four FOR-NEXT cycles open: the digits vary from 1 to 9; zero is excluded so that all four numbers in play are two digits.

The four digits must all be different; when this is not the case, one of the three IFs (120-140) refers back to the choice of the next candidates.

At 150, the possible values of X and Y are calculated; if Y is not seven times X, the pair cannot be the solution (instructions 160).

At 170, the values of X and Y written "backwards" (X2,Y2) are calculated: if their product does not match the product of the original numbers, the pair (X1,Y1) is not the solution to the question (inst. 180).

Step 190 is only reached if all requirements are met; the program immediately finds a solution, but continues searching.

After about five minutes, the VICE emulator (no Warp Mode) ends processing without further printing: the solution is unique.

In fact, if the product of the two original numbers, 10*A+B and 10*C+D, must match the product of the two written "backwards," we must have:
(10*A+B)*(10*C+D)=(10*B+A)*(10*D+C).
By performing a few steps, we can deduce that 99*A*C=99*B*D, or A*C=B*D.

If you want, you can replace instruction 170 with 170 IF A*C<> B*D THEN 220 and delete instruction 180: the program will be a little shorter and a little faster.

Of course, this "Question with Susi" can also be solved without bothering our Commodore (the number on Giulia's piece of paper cannot be less than 12 - the smallest two-digit number with different digits - but it cannot exceed 14 - since 14*7=98 is the maximum possible value on Pietro's piece of paper, so ...), but our PROG5, suitably modified, can be useful for answering other questions.

For example, how many pairs of two-digit numbers, all different, have a product that is identical to that obtained by multiplying the numbers written backwards? There are quite a few, here are some: 12*63=21*36; 13*62=31*26; 14*82=41*28, 23*64=32*46, ... , 36*84=63*48.

Large factorials (for C64 - for beginners)

by Eugenio Rapella

Our C64 does not need a big program to calculate the factorial of a natural number "n" (indicated by "n!" and defined as the product of all integers between 1 and "n", with the convention of setting 0!=1; thus 4!=4*3*2*1=24; factorials come into play in various chapters of mathematics such as Combinatorial Calculus, series, etc.), in truth it can manage with a single instruction:

```
1 input" n = ";n:f=1:for k=1 to
n:f=f*k:next:print n;"! = ";f
```

If we ask RUN to calculate 10!, we immediately get 3628800 and, up to 12, we get the exact values: 12! =479001600. For 13!, the C64 switches to exponential notation: 13!=6.2270208E+09, or 6.227*10^9 is the approximate value of 13 factorial. This continues up to 33! = 8.68331762E+36. From 34 onwards, our beloved Commodore throws in the towel and reports an OVERFLOW ERROR IN 1.

There are numerous websites on the internet that provide the exact value of "n!" even for n>33. The C64 doesn't like this: "If others can do it, so can I!". Let's see how it fares.

As "n" increases, "n!" grows very quickly; first, we ask our Commodore to evaluate how many digits "n!" will consist of (it goes without saying that we are working in base 10).

We need to refresh some notions related to logarithms; we will use base 10 logarithms. The decimal logarithm of a natural number k>0 is closely related to the number of digits that form it. If, for example, k=7203, we will have 1000 â‰¤ k < 10000. Since the logarithm is an increasing function, we will have 3=log(1000)â‰¤log(k)<log(10000)=4, therefore log(k)=3,.... In general, if kâ‰¥1, the number of digits that make up "k" is int(log(k))+1 (in our example, log(k)= 3,â€¦. from which int(log(k))=3 and int(log(k))+1=4, which is precisely the number of digits in 7203).

Since "n!" is the product of integers from 1 to "n", log(n!) = log(1)+log(2)+ ... + log(n) (... remember? "The logarithm of the product is the sum of the logarithms..."). The LOG function of the C64 provides logarithms to the base "e" ("natural" logarithms), but it is easy to convert to logarithms to the base 10 ("decimal" logarithms) using the base change formula: LOG10(X)=LOG(X)/LOG(10).

Here is the mini-program that allows us to determine how many digits "n!" consists of:

```
10 input" n = ";n
20 for h=1 to n:s=s+log(h):next
30 print" il numero di cifre di n! e' ";int(s/
log(10))+1
```

We then discover that 100! is a number consisting of 158 digits, 500! consists of 1135 digits, and 1000! reaches 2568 digits!

To actually calculate these large numbers, let's go back to the initial program. While the counter K remains moderate, the variable F grows disproportionately. To maintain the exact value, one possibility is to store the digits in a vector C(..): the partial products K*F can be calculated by multiplying each digit of F by K and then reconstructing the digits of the product as in the following example:

Calculate 842*13

C(...)	0	0	8	4	2	
					*13	
C(...)	0	0	104	52	26	: 26=2*10+6
				+2		
C(...)	0	0	104	54	6	: 54=5*10+4
			+5			
C(...)	0	0	109	4	6	:109=10*10+9
		+10				
C(...)	0	10	9	4	6	
	+1					
C(...)	1	0	9	4	6	

The vector C(..) is updated in two steps. The value contained

in C(K) is separated into two parts: U, which contains the units digit of C(K), and D, which contains the part preceding U. If C(K)=725, we will have U=5 and D=72. For the calculation, D=INT(C(K)/10) and U=C(K)-D*10 (in fact, continuing with the previous example, D=INT(725/10)=INT(72.5)=72 and U=725-72*10=5), after which U becomes the new C(K) and C(K+1)=C(K+1)+D. Of course, the vector C(..) will be sized with the value calculated in the mini-program (for convenience, we will not use C(0)).

Here is the complete program:

```
100 poke 53280,6:print chr$(5);chr$(147)
110 input"n = ";n:if n<4 then 110
120 for h=1 to n:s=s+log(h):next
130 nc=int(s/log(10))+1:print:print"il
numero di cifre di n! e' ";nc
140 print:print"procedo con il calcolo, abbi
pazienza":print
150 dim c(nc):c(1)=1
160 for h=1 to n:print"..";h;
170 for j=1 to nc:c(j)=c(j)*h:next j
180 for k=1 to nc-1
190 d=int(c(k)/10):u=c(k)-d*10
200 c(k)=u:c(k+1)=c(k+1)+d
210 next k:next h
220 print:print:print n;"! = ";
230 for w=nc to 1 step -1
240 print right$(str$(c(w)),1);
250 next w
```

Instruction 100 takes care of the preliminaries: border color, font color, screen clearing, and 110 requests the number (greater than 3) for which the factorial is to be calculated.

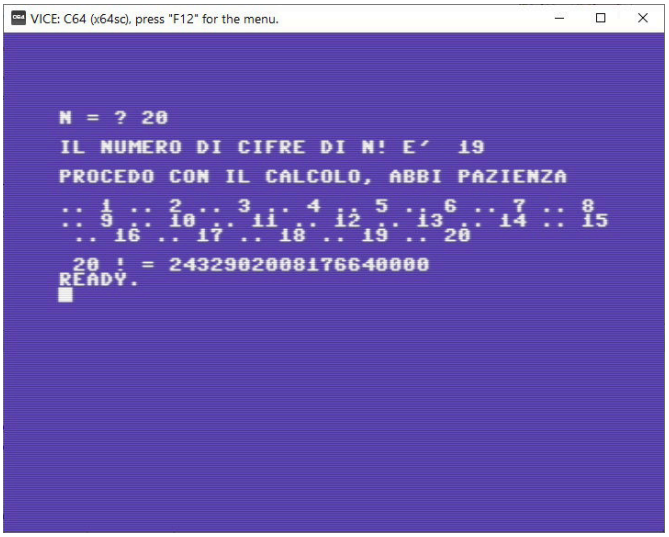
In 120 and 130, the number of digits (NC) of "n!" is calculated, and in 150, the vector C(..) is sized and initialized. In 160, the main cycle begins; the counter H, which tells us where we are in the calculation, is printed so that the user can get an idea of how long they will have to wait. Within the main cycle there are two other "for-next" cycles: the one at 170 multiplies each digit of C(..) by the current K, while the one from 180 to 210 performs the "transfer of digits" as explained above. As "n" increases, the number of iterations grows exponentially and waiting times can become prohibitive.

When we reach 220, the calculation is complete: C(1) contains the units digit, C(2) the tens digit, and so on up to C(NC), which contains the most significant digit.

The 230-250 cycle takes care of the final printing; the use of RIGH$(STR$(C(W)),1) serves to eliminate the initial space that the C64 automatically places before the digit. In this way, the various digits are printed "glued" to each other.

A tip: at the beginning, use the program with small values of "n", n=10, n=20, then... see for yourself. The C64's Basic interpreter, as we know, is a bit slow; calculating 100! through the VICE emulator took a good 24 minutes.

Here are a couple of outputs:



"Oh my God... my husband!"

by András Vajda

Summary

The famous line from "Tailleur pour dames" by the great Georges Feydeau, one of the most hilarious (and copied) pochades focusing on the theme of extramarital affairs, lends itself beautifully to introducing the no less famous combinatorial problem known as the "Stable Marriage Problem." In this article, we offer a brief description of the problem as originally posed by Gale & Shapley in 1962 and an elegant implementation in COMAL 80 for Commodore 64.

Introduction

Let's dispel a myth right away: scrolling through the now vast list of applications created around this problem and its variations ([Man13], pp. 30-31), we see that in reality the use of such algorithms in the field of marriage and similar areas (online dating, etc.) is sporadic, almost non-existent. As is very often the case, the reference to 'stable marriage' used in the name of the problem is deliberately metaphorical and serves only to create a witty mnemonic and situational cue for what is a very serious problem of resource allocation in operations research, which arose around an important issue (the allocation of students to universities and, independently, of trainee doctors to clinics) and then evolved into a series of generalizations with a vast field of application: from the allocation of institutional positions in public and parastatal bodies of all kinds to logistics, from production planning to the allocation of resources to cost centers, to the management of mentoring in the health or military sectors, etc.

Stable marriages: the historical origin of the problem

What exactly is the Stable Marriage (SM) problem? There are numerous variations, but in the original formulation, due to Gale and Shapley in 1962 ([GS62]), there are two distinct groups (men and women) of equal size. Each member of each group expresses their preferences for the opposite sex in a complete list, ordering themâ€"to clarify the conceptâ€"in descending order, from most to least desirable. In practice, each element of one of the two sets is associated with a particular permutation of the other set. Pairs are then formed according to the preference lists: the solution obtained is called a stable marriage if there are no two individuals, one from each set, who mutually prefer the other individual to their current partner. The algorithm published in 1962 (which we will call GS) is based on very simple steps called "proposals," which are accepted or rejected according to a simple positional comparison between the current partner and the proposer on the priority scale of the party receiving the proposal. If the recipient prefers their current partner to the proposer, Manzoni's "This marriage shall not take place" kicks in and we move on to the next preference. The typical idea, in Gale & Shapley's intentions, is that the set M of "men" (e.g., students) proposes and the set W of "women" (e.g., colleges, clinics) evaluates the proposals, but obviously nothing prevents the algorithm from being executed following the opposite convention. The main merit of Gale & Shapley is that they demonstrated that:

1. Any instance of the SM problem admits at least one stable solution;
2. This solution has the property of being proposer-optimal, i.e., it guarantees each member of the proposing set the best possible match compared to any other stable (and non-stable) solution. This property is known as weak Pareto optimality.

All this using an extremely simple algorithm which, given two non-empty sets M and W of cardinality n and the relative preference matrices, generates the male-optimal (or female-optimal) solution in O(n2) in the worst case. The analysis of the average case of execution would merit an entire article dedicated to it, but here we will limit ourselves to presenting the result involving the definition of harmonic number:

$$H_n = \sum_{k=1}^{n} k^{-1} = \gamma + \psi_0(n+1)$$

where Hn is precisely the nth harmonic number. In the analytical expression on the far right, γ is the Euler-Mascheroni constant and ψ0() is the digamma function, i.e., the logarithmic derivative of the gamma function Γ(), which in turn is an extension of the factorial to real numbers. See the classic [Leb72].

That said, the average number of proposals in the GS algorithm is given by n · Hn + O((logn)4). It follows that the average complexity is of the order of Θ(nlogn).

The solution obtained can obviously be expressed in various ways. Essentially, it consists of a list of n unique pairs (m,w) with m ϵ M and w ϵ W. W. Consider the following two sets: M = {Aldo, Bruno, Carlo, Dario} and W = {Elena, Flora, Grazia, Helga}. Each member of the sets expresses their list of preferences using the initials of their names:

Gentlemen				
A	H	E	F	G
B	F	G	E	H
C	F	H	G	E
D	G	E	H	F

Ladies				
E	D	A	C	B
F	A	C	B	D
G	A	B	C	D
H	D	A	C	B

Intuitively, the first row of the matrix on the right tells us that Ms. Elena prefers Mr. Dario, then Mr. Aldo as her second choice, and so on in descending order of preference. From a computational point of view, however, as in most of the literature, it has become customary to encode input sets using small integers for both men and women. Although the notation may initially cause some confusion, it is nevertheless an extremely effective, universal, and easy-to-manipulate encoding, especially for large sets. Thus, our tables take on a slightly different appearance, while retaining exactly the same meaning:

Gentlemen				
1	4	1	2	3
2	2	3	1	4
3	2	4	3	1
4	3	1	4	2

Ladies				
1	4	1	3	2
2	1	3	2	4
3	1	2	3	4
4	4	1	3	2

The stable suboptimal solution generated by GS is the following list of pairs that follows the convention (m,w): {(1,4),(2,3),(3,2),(4,1)}, which corresponds to {(A,H), (B,G),(C,F),(D,E)} using the respective initials. This list can obviously be expressed as a permutation in standard two-line notation.

Reading the notation from left to right and from top to bottom, we have the pairs (1,4),(2,3),(3,2),(4,1). In this notation, the identity permutation of the first row represents the proposing set in natural order. It follows that we can also use 1-line positional notation, as for any other permutation, without loss of information: the notation 4 3 2 1 represents exactly the same pairs, where men are implicitly assigned positions 1..n increasing to the right and their partners are specified explicitly.

However, the elementary combinatorial nature of the solution is not very helpful to us computationally. Finding the optimal solution among the possible permutations of the set W (or M, if applicable) by exhaustion, applying the "generate and test" paradigm, is completely impractical even for problems of very modest size. This solution can also be seen in its binary relation nature, i.e., a subset of the Cartesian product M Ã—W between the two given sets, which, in a nutshell, is nothing more than the list of all possible n2 ordered pairs formed by taking only one element at a time from each starting set. Ultimately, this is absolutely trivial from a combinatorial point of view, but as is often the case, the algorithms needed to obtain an exhaustive enumeration of such solutions, as well as those dedicated to finding particular types of solutions, were far from being developed at the time of Gale & Shapley. Almost all of them are characterized by very high computational complexity and/or require a completely unique approach, with the aid of non-elementary data structures.

The 1970s and 1980s

A few years after the publication of Gale & Shapley's article, McVitie and Wilson in 1970/71 [MW70, MW71b, MW71a] presented various algorithms in ALGOL60, the most important of which allows the exhaustive generation of all stable solutions for a given instance of SM (for the version of the problem generalized to the case of lists of unequal sizes, which as a special case solves the standard instance). Unfortunately, due to copyright restrictions still in force on these articles, it will not be possible here to explore the subject in the depth it deserves by presenting the complete original source code. Although this recursive algorithm has decidedly unimpressive performance, it is of considerable historical importance as it constitutes the first attempt to provide an exhaustive generation of solutions. In the same years, D. E. Knuth also proposed a substantially similar algorithm with similar performance.

It would take almost twenty years for this effort to be surpassed, with the publication of several articles that were later included in the seminal monograph by Gusfield and Irving [GI89], who, thanks to a careful study of the algebraic nature of the solution space as a lattice and as a ring of sets, and suggesting a judicious use of order-preserving data structurespreserving data structures capable of simultaneously guaranteeing both direct indexed access and deletions in O(1) - proposed the algorithm still in use today for exhaustive generation even on large instances, with performance in O(n2), practically the same order of magnitude as GS (which at that time, however, was only capable of generating a single solution, not all of them). We will explore elsewhere how such performance gains can be achieved.

Another characteristic (and amusing!) aspect of McVitie and Wilson's algorithm, which makes it even more worthy of study and interest, is its original implementation on a transistor-based KDF-9 from English Electric Computers (later English Electric LEO Marconi Computers), installed at the University of Newcastle: this is genuine retrocomputing at its best! There is an emulator written in Ada for this historic computer, equipped with magnetic core memory and accompanied by extensive documentation, which the author has enjoyed using for a long time.

As already mentioned, starting in the mid-1980s, a number of publications (mainly by Gusfield, Irving, and Leather: [IL86, ILG87, GILS87, Gus87]) had reawakened general interest in the problem, stimulating numerous popular articles in leading international application programming magazines and, as an immediate consequence, several study implementations in various high-level languages (including Ada, C, Clipper, and COMAL) by the author. In this article, we will focus on one of these implementations in COMAL 80 V. 2.01 for Commodore 64.

One last note. When the author first systematically addressed the problem in an academic context in the late 1980s, there were dozens of publications on the subject, SM was mentioned in at least half a dozen algorithmic texts dating back to the 1970s, and in addition to the aforementioned Gusfield-Irving (then hot off the press), there were two other dedicated monographs, one of which [RS90] focused primarily on game-theoretic aspects, and the other, by the venerable D.E. Knuth [Knu97], published only in French until 1997. Twenty-five years later, the number of publications and algorithmic texts mentioning the problem has almost doubled. However, it should be noted that many of the questions raised in the main historical monographs (to which a fourth [Man13] was added a few years ago) have not yet been resolved, and the problem remains rich in research and application ideas. Interested readers can download an in-depth article on the Gusfield & Irving algorithm, loosely related to retrocomputing (an implementation in C89 is proposed, in line with the tools available at the time of publication of the monograph, without resorting to Python and advanced data structures), whose size would make it difficult to publish here, even in several installments.

The Gale & Shapley algorithm

The most basic version, reported in algorithmic texts and numerous popular articles, is also the most widely used for implementations. It is an extremely simple iterative algorithm, but one that guarantees a result that is not at all intuitive when reading the problem. The non-determinism introduced by the lack of a particular order of proposers is an aspect that was taken for granted in scientific articles in the 1960s and 1970s: in reality, the order of the proposers is completely irrelevant, because the algorithm always generates the same suboptimal solution (or female-optimal, by simply inverting the preference tables), as Gusfield and Irving ([GI89], pp. 9-10) point out.

```
procedure STABLE_MARRIAGE
  assign each person to be free;
  while some man m is free do
  begin
     w:=next(m);
    if free(w) then
        assign m and w to be engaged {to each
other};
    else
          if w prefers m to her fiancé m' then
            assign m and w to be engaged;
            free m';
          else
             w rejects m {m remains free};
  end;
  output the n engaged pairs;
```

The algorithm's operation is very intuitive. The convenient function next(m) provides the woman w immediately following the current partner (or the first on the list, if this is the first execution) in the list of preferences of the man m, the proposer. The free(w) function, quite obviously, checks whether w already has a partner. If so, the positions of the proposer m and the current partner m' in w's list of preferences are compared and action is taken accordingly, creating the pair that assigns w the relatively best match according to her preferences and frees the least desirable, who can then propose to other potential partners in subsequent iterations. It is important to note that these comparisons, as well as the implementation of auxiliary functions, can and must be performed in O(1) without resorting to cumbersome linear searches in the main arrays, given the average size of the problems generally dealt with by this family of algorithms, which is in the order of thousands of proposers. It is equally important to note that bound checks on indices are essentially superfluous, as Gale & Shapley have shown that the algorithm always terminates without any proposer exceeding the end of their preference list, thus remaining unmarried.

The winning idea behind the algorithm can be explained very simply: any conflict of preferences between proposers is resolved deterministically, choosing between the two the one that is closest to the top of the contested woman's preference list. In this way, one or more suitors will have to settle for a partner they prefer less, as shown by the final outcome of the example we will see shortly, but who nevertheless (it can be proven) remains their best choice among all possible stable solutions.

Let's close this short section with a historical curiosity: in implementations up until the 1970s, the sign bit was used to indicate whether an individual m or w was married, without using an auxiliary array. This was a typical retro-programming trick that characterized an entire era of ALGOL and FORTRAN programs, even before the era of PETs and home computers, conditioned by the costs of physical memory and its proverbial scarcity.

Implementation in COMAL 80

Here is an example implementation in COMAL 80 (tested with version 2.01 on cartridge) for Commodore 64, a simplified and reduced version of one of the author's first implementations, dated 1984. To easily verify the correctness of the implementation, the preference tables of example size 8 from [GI89], p. 12 were chosen, which generate the following stable suboptimal solution, ordering the pairs as usual according to the convention (m,w):
{(1,5),(2,3),(3,8),(4,6),(5,7),(6,1),(7,2),(8,4)}

```
//*******************************
// save stable-marriage
//*******************************
DATA 5,7,1,2,6,8,4,3
DATA 2,3,7,5,4,1,8,6
DATA 8,5,1,4,6,2,3,7
DATA 3,2,7,4,1,6,8,5
DATA 7,2,5,1,3,6,8,4
DATA 1,6,7,5,8,4,2,3
DATA 2,5,7,6,3,4,8,1
DATA 3,8,4,5,7,2,6,1

DATA 5,3,7,6,1,2,8,4
DATA 8,6,3,5,7,2,1,4
DATA 1,5,6,2,4,8,7,3
DATA 8,7,3,2,4,1,5,6
DATA 6,4,7,3,8,1,2,5
DATA 2,8,5,3,4,6,7,1
DATA 7,5,2,1,8,6,4,3
DATA 7,4,1,5,2,3,6,8

DIM male'pref(8,8)
DIM female'pref(8,8)
DIM male'lut(8,8)
DIM m'coupled(8)
DIM w'coupled(8)
DIM stack(8)
stack'ptr:=0

PAGE
FOR i:=1 TO 8 DO
  FOR j:=1 TO 8 DO
    READ male'pref(i,j)
  ENDFOR j
  push(i)
ENDFOR i

FOR i:=1 TO 8 DO
```

```
  FOR j:=1 TO 8 DO
    READ female'pref(i,j)
    male'lut(female'pref(i,j),i):=j
  ENDFOR j
ENDFOR i

disp'data

WHILE stack'ptr>0 DO
  pop(man)
  m'coupled(man):+1
  woman:=male'pref(man,m'coupled(man))
  IF w'coupled(woman)<1 THEN
    w'coupled(woman):=man
  ELSE
                                        IF
male'lut(man,woman)<male'lut(w'coupled(woman)
,woman) THEN
      push(w'coupled(woman))
      w'coupled(woman):=man
    ELSE
      push(man)
    ENDIF
  ENDIF
ENDWHILE

disp'solution

END "Fine lavoro"

PROC disp'data
  PRINT "Uomini:              Donne:"
  FOR i:=1 TO 8 DO
    PRINT i,": ",
    FOR j:=1 TO 8 DO
      PRINT male'pref(i,j)," ",
    ENDFOR j
    PRINT AT i+1,21: i,": ",
    FOR j:=1 TO 8 DO
      PRINT female'pref(i,j)," ",
    ENDFOR j
    PRINT
  ENDFOR i
ENDPROC disp'data

PROC disp'solution
  PRINT AT 12,1: "Stable marriage (m,w):"
  FOR i:=1 TO 8 DO
    w:=male'pref(i,m'coupled(i))
    PRINT AT 12+i,1: "(",i,",",w,")"
    PRINT AT i+1,2+2*m'coupled(i): CHR$(158),w
          PRINT  AT  w+1,22+2*male'lut(i,w):
CHR$(159),i,CHR$(154)
  ENDFOR i
  PRINT AT 20,1: CHR$(154)
ENDPROC disp'solution

PROC push(d)
  stack'ptr:+1
  stack(stack'ptr):=d
ENDPROC push

PROC pop(REF d)
  d:=0
  IF stack'ptr>0 THEN
    d:=stack(stack'ptr)
    stack'ptr:-1
  ENDIF
ENDPROC pop
```

Although there are no comments, the listing is entirely self-explanatory. We chose to implement a stack in the most classic way, using the array named (with a bold stroke of imagination) stack() and the single variable stack'ptr. The value 0 is perfect as a guard value in this case, since in COMAL indexes always start from 1. Note that in COMAL, the apostrophe character is perfectly valid when used as a separator in procedure and variable names, just like the underscore in other languages.

The arrays male'pref() and female'pref() contain, unsurprisingly, the preference lists of the two sexes, while male'lut() is used for reverse lookup in constant time, since for every possible pair (m,w) it gives us the position of m in w's preference list, making it possible to immediately compare the position of the proposer with that of the current partner, as required by the algorithm.

One last trivial note: the array w'coupled() immediately tells us who w's current partner is for each w â^^ W, and its locations each contain a small integer 1 â‰¤ m â‰¤ 8 that identifies the partner, while the array m'coupled(),

as is evident from the code, is used in a slightly different way. The content of each cell is always a small non-negative integer, but this time it is used as an index relative to the position of w in m's preference list. In this way, for example, considering the pair (m,w) = (1,5), we will have m'coupled(m)=1 since male'pref(m,1)=5, and so on for every other pair.

Regarding arrays, just remember that they are implicitly initialized to zero, as are variables in general. Note once again how the language is a bit verbose (like ALGOL, COBOL, or FORTRAN) but extremely clear, readable, and orderly. The final result can be seen in the screenshot below, where the partners are listed explicitly and highlighted with different colors in the preference matrices from the disp'solution procedure, showing the relative placement of the partners in each preference list to stimulate the reader's reflection on the profound meaning of the Pareto optimality of the algorithm.

Conclusions

We have presented a very brief overview of one of the longest-standing and most famous combinatorial problems, with some fundamental references to the main stages in the development of solutions up to the 1980s. For a more complete overview of the vast existing literature, interested readers are referred to the monographs already mentioned, their bibliographies, and the most recent survey available: [IM08].

A complete implementation in COMAL 80 2.01 for Commodore 64 has therefore been provided, which allows us to exemplify the characteristics and great elegance of this advanced language, making its operation transparent thanks to the simplicity of the algorithm. The author's dual hope is that, on the one hand, readers will be intrigued by the problem and its numerous variations and, on the other, that they will increasingly appreciate the language and its great expressiveness.

Bibliographical references

[GI89] Dan Gusfield and Robert W. Irving, "The stable marriage problem: Structure and algorithms," MIT Press, Cambridge, MA, USA, 1989.

[GILS87] Dan Gusfield, Robert Irving, Paul Leather, Michael Saks, "Every finite distributive lattice is a set of stable matchings for a small stable marriage instance," Journal of Combinatorial Theory, Series A 44 (1987), no. 2, 304-309.

[GS62] D. Gale, L. S. Shapley, "College admissions and the stability of marriage", The American Mathematical Monthly 69 (1962), no. 1, 9-15.

[Gus87] Dan Gusfield, "Three fast algorithms for four problems in stable marriage", SIAM Journal on Computing 16 (1987), no. 1, 111-128.

[IL86] Robert W. Irving, Paul Leather, "The complexity of counting stable marriages" , SIAM Journal on Computing 15 (1986), no. 3, 655-667.

[ILG87] Robert Irving, Paul Leather, Dan Gusfield, "An efficient algorithm for the 'optimal' stable marriage" , J.ACM 34 (1987), 532-543.

[IM08] Kazuo Iwama, Shuichi Miyazaki, "A survey of the stable marriage problem and its variants", Proceedings of the International Conference on Informatics Education and Research for Knowledge-Circulating Society (Icks 2008) (USA), ICKS08, IEEE Computer Society, 2008, pp. 131-136.

[Knu97] Donald E. Knuth, "Stable marriage and its relation to other combinatorial problems," American Mathematical Society, Providence, R.I, 1997.

[Leb72] N. N. Lebedev, "Special functions and their applications," Dover Publications, Inc., New York, 1972.

[Man13] David Manlove, "Algorithmics of matching under preferences", World Scientific Publishing Company, 2013.

[MW70] D. G. McVitie, L. B. Wilson, "Stable marriage assignment for unequal sets", BIT Numerical Mathematics 10 (1970), no. 3, 295-309.

[MW71a] D. G. McVitie, L. B. Wilson, "The stable marriage problem" , Communications of the ACM 14 (1971), no. 7, 486-490.

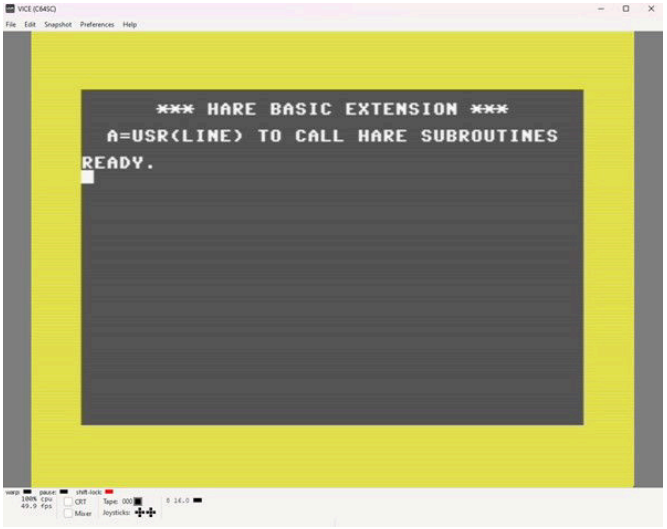
[MW71b] D. G. McVitie, L. B. Wilson, "Three procedures for the stable marriage problem", Communications of the ACM 14 (1971), no. 7, 491-492.

Hare Basic – 10 times faster

by Carlo Nithaiah Del Mar Pirazzini

Hare Basic was created by Aleksi Eeben, who also wrote Bunny Basic in 2019, with two goals in mind: to celebrate the 60th anniversary of BASIC and to create a fast and functional basic interpreter that could replace Basic 2.0 on the C64 and VIC20. Hare Basic is a highly optimized subset of Commodore Basic that can be enabled or disabled as needed. It is an interpreter, not a compiler, and has few commands and functions, all of which can be executed directly on the C64.

This is what Hare Basic looks like when it starts up:



The syntax is very similar, but there are some restrictions and differences.

There are no string variables, but there is simple support for NULL-terminated strings that can do this.

There are also no Arrays. Memory is accessed directly with POKE and PEEK commands to create larger data structures.

Only one arithmetic operation is possible at a time.

White space cannot be used. You will receive a ?HARE ERROR READY message if you do.

GOTO and GOSUB can take a variable as their destination. The command THEN does not accept only a number as a variable. However, the construct THENGOTO works with variables.

The STR$ function is provided to refer to NULL-terminated strings in memory. You can use variables as pointers or use a direct memory address.

RND is implemented as a command that inserts a random number in the range 0-65534.

The LOAD command does not restart the program but continues with the next instruction.

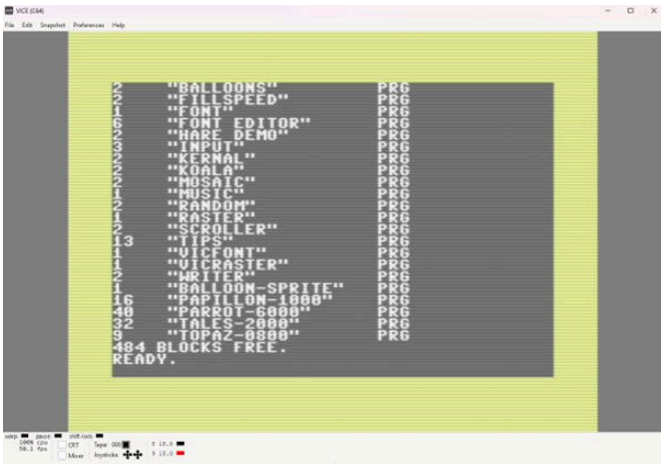
An example listing can be found in the HARE BASIC documentation folder:

There are also many other differences, all explained in detail

with numerous examples in the manual on the website https://csdb.dk/release/?id=242880, which we strongly recommend you to read.

On the author's Dropbox page (https://www.dropbox.com/scl/fo/qkn7ehgoctgcq4ldrlx43/AHwpvxdq06t3rVN9dkCrY4g?rlkey=t44bux2c3qtsa31t5eupank8t&e=2&st=0oe22jyb&dl=0) you can also find a series of useful examples to help you understand the speed of execution and the differences.

There are many examples included in the software package.

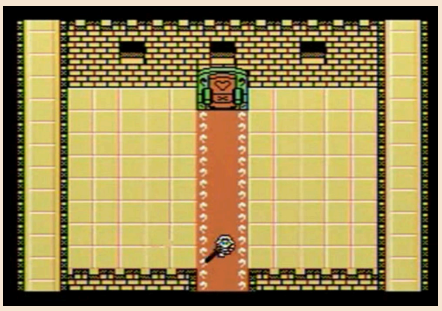


In conclusion, I enjoyed "playing" with this streamlined and fast version of Basic, a discovery and certainly a breath of fresh air, a sign of a retro development scene that is alive and well.

In the next issues, we will analyze some listings developed with HARE BASIC together with some collaborators, so don't miss it!

QUEST FOR THE GOLDEN CHALICE

Year: 2012
Editor/Developer: Team Pixelboy
Genre: Avventura
Platform: Colecovision
Web site: https://www.teampixelboy.com/quest_chalice.php



Dragons, castles, bats, and a cowardly prince... my God!
All this is Quest for the Golden Chalice, a 2012 project for Colecovision.
A game inspired by Adventure for Atari 2600; a title much loved by developers, as can be seen in the game's layout.
But this Quest for the Golden Chalice is one of those titles we felt was needed on the Coleco console.
Thanks to the console's graphics capabilities compared to the Atari 2600, this "tribute" to Adventure takes on a graphic style and modern feel reminiscent of the "Zelda" saga.
A fun game with a bizarre plot that sees the prince of Larosia, known for his cowardice, face monsters, evil creatures, and powerful dragons.
The golden chalice has been stolen by an evil wizard named Mardok and hidden away.
Only his sword, a good dose of luck, and several keys scattered here and there will allow the Prince of Larosia to find the chalice and return it to the Yellow Castle.
The gameplay is as simple as it was in Adventure. Wander around the labyrinthine maps, collect the keys, and avoid being killed by terrifying monsters.
The visuals are well done and very reminiscent of Zelda. The graphics are sharp and everything moves smoothly.
In short, this is a title that is definitely worth checking out, even though it is only 12 years old.

by **Giampaolo Moraschi**

OUR FINAL SCORE

» Gameplay 85%
A great tribute to Adventure for Atari 2600 but with a Zelda style. Simple and fun.

» Longevity 80%
There are also several Easter Eggs here, as in the original title. Fun.

NEW GAME

TIMO'S CASTLE

Year: 2022
Editor/Developer: Roman Warner
Genre: Platform
Platform: Commodore 64
Web site: https://romwer.itch.io/hc

Timo's Castle is a well-made product. Behind the game is a seasoned team and excellent programming.

In the game, we play Timo McClane, a young Scottish nobleman. A boy who wants nothing to do with nobility and traditions, but who finds himself investigating the mysterious death of his parents.

From this tragic event, we have inherited a small manor lost in the Scottish Highlands.

Together with his girlfriend Amber, Timo finds himself inside with the aim of tidying it up and, perhaps, discovering the fate of his family.

But something is immediately wrong. The castle is cursed and a wicked witch kidnaps Amber.

And so our story begins.

Timo's Castle is a tribute to the famous Henry's House. Like its illustrious predecessor, it is a fixed-screen platform game.

The game levels are the rooms of the castle (8 in total), each with its own unique characteristics and full of possessed objects that want to tear the protagonist to pieces.

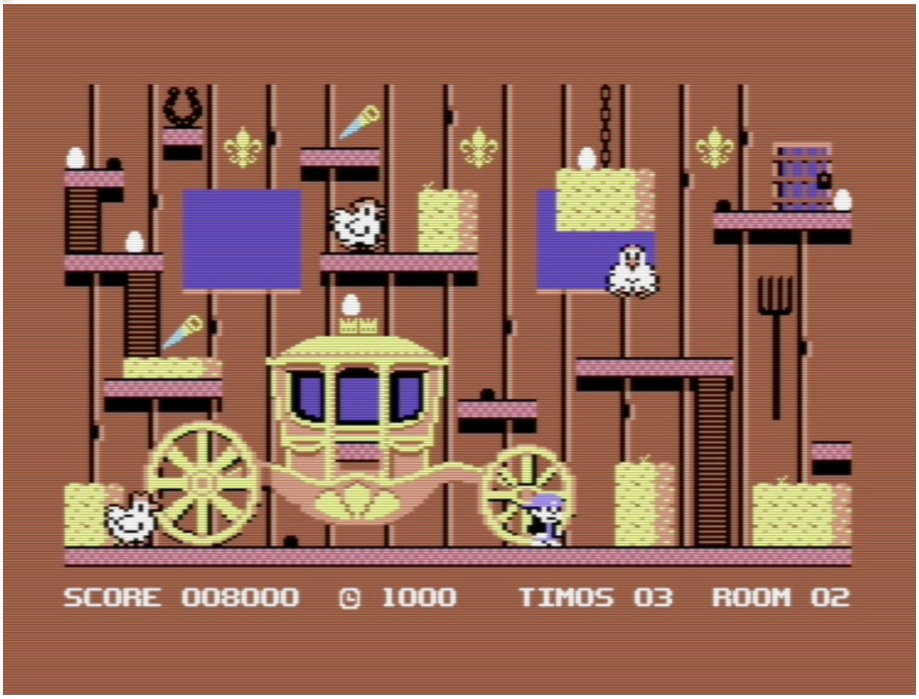
All this is immersed in countless familiar references and an atmosphere reminiscent of the legendary 70s, the acrobatic 80s, and the "grunge" 90s.

Timo's Castle retains all the elements that made Henry House special. The rooms are well designed and offer a fair amount of difficulty. The enemies are always ready to hinder our hero, and there is that desire to see what lies beyond each level.

This connection has obviously been

improved: we have a more modern look, faster animations, and greater attention to detail.

The graphics are very pleasing, and we appreciate the protagonist's sprite, which reminded us of the hero of Magic Pockets, with that weird '80s



OUR FINAL SCORE

» Gameplay 90%

It maintains the style of Henry's House in terms of levels, but is easier to handle. The collisions and level design are beautiful.

» Longevity 85%

It suffers from the "old" style of the past and requires practice, but completing it is very satisfying.

'tamarro' look.

But it's the sound that we loved the most. All the rooms feature a beautiful soundtrack, and the intro features a perfect SID version of 'Scotland the Brave'.

The style of Henry's House is evident. It's not an easy game, especially if you take it lightly. It requires precision and attention to complete the levels.

Reaching the final screen is no easy feat, but the desire to see it will keep you pressing the fire button until the end.

The game is available for download at a cost of $4.99, and it's worth every penny. It may not be an absolute masterpiece, but it's a damn fun title.

by **Carlo Nithaiah Del Mar Pirazzini**

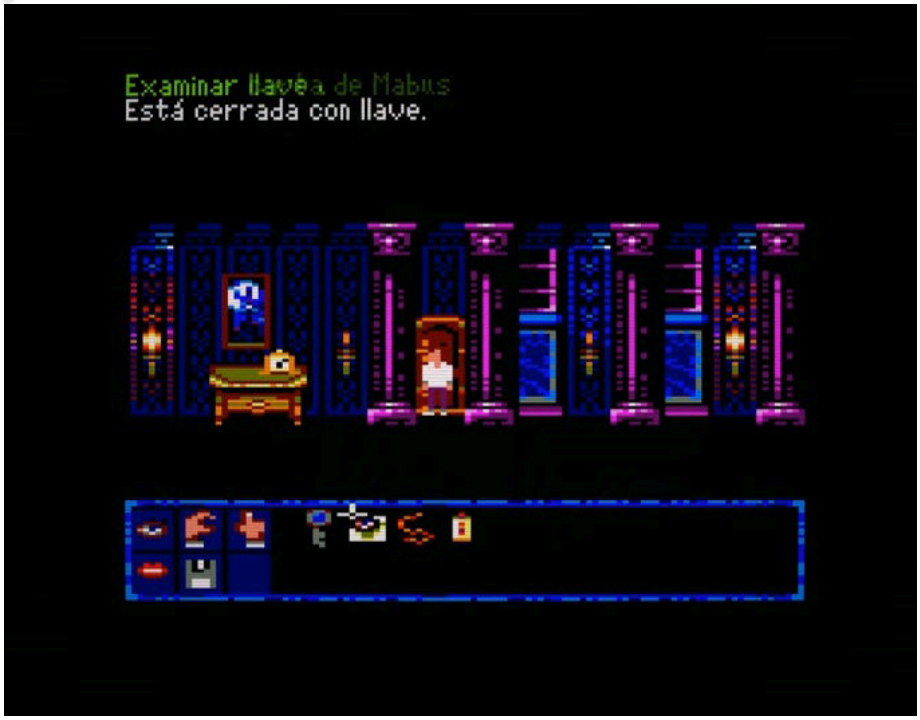
NEW GAME

THE KEY

Year: 2024
Editor/Developer: Pakete Soft
Genre: Avventura punta e clicca
Platform: Amstrad CPC
Web site: https://www.paketesoft.com/es/thekey

A terrible event that took place forty years ago, some tapes bearing witness to the atrocities that occurred, a note and a key affixed to the front door, and a disturbing note about the mysteries of Carfax Mansion.

These are the opening elements of the first episode of The Key, a third-person point-and-click adventure for Amstrad CPC.



OUR FINAL SCORE

» Gameplay 85%
The LucasArts style is a plus for a point-and-click adventure with a solid plot.

» Longevity 80%
Very entertaining puzzles.

The development team, Pakete soft, already released a well-made point-and-click title in 2019 called Escape the ROM, created to showcase the Paket engine, but with The Key, they present us with a true homage to the spirit of the "LucasArts" adventures of yesteryear.

The plot flows well and is full of puzzles (some of which are truly Machiavellian!), allowing us to play without ever reaching a dead end.

The interface is vaguely reminiscent of Scumm and works wonderfully, complemented by well-made technical aspects with detailed and colorful graphics and a good musical accompaniment.

The beauty of this title starts with the way it is downloaded... via a puzzle!

On the official Pakete website, there is a crazy clickable puzzle that, once solved, will allow you to download the title. Brilliant!

The plot is interesting and well developed, with some touches of genius in the dialogues.

The game is available in Spanish, but two versions are planned for release soon: one in French and one in English.

All in all, I can only recommend this title and suggest that you follow the Pakete Soft team.

by **Giampaolo Moraschi**

NEW GAME

LYLE IN CUBE SECTOR

Year: 2024
Editor/Developer: Michael Moffit
Genre: Platform/Metroidvania
Platform: Sega Megadrive
Web site: https://mikejmoffitt.itch.io/lics-md

"My name is Lyle. Unfortunately, I can't tell you my last name; video game heroes usually have one-word names, and I don't want to be the exception.

Also, like most video game heroes, I can jump higher than normal, my head is bigger than my torso (hell, even my eyes are bigger!), and my health isn't determined by pressure, stress, food, or my surroundings, but by a simple bar at the top of the screen you're looking at.

I live in a small shack in the middle of the Cube sector, a realm populated by all sorts of nasty monsters. And yes! There are lots of cubes around here: durable green ones, explosive red ones, double-sized orange cubes, and so on. In fact, I have a green cube right in my backyard.

My only companion in life is a kitten named Keddums. Seriously! But enough about me... if you download the ROM, you can read all about the game in the txt file. Ugh!

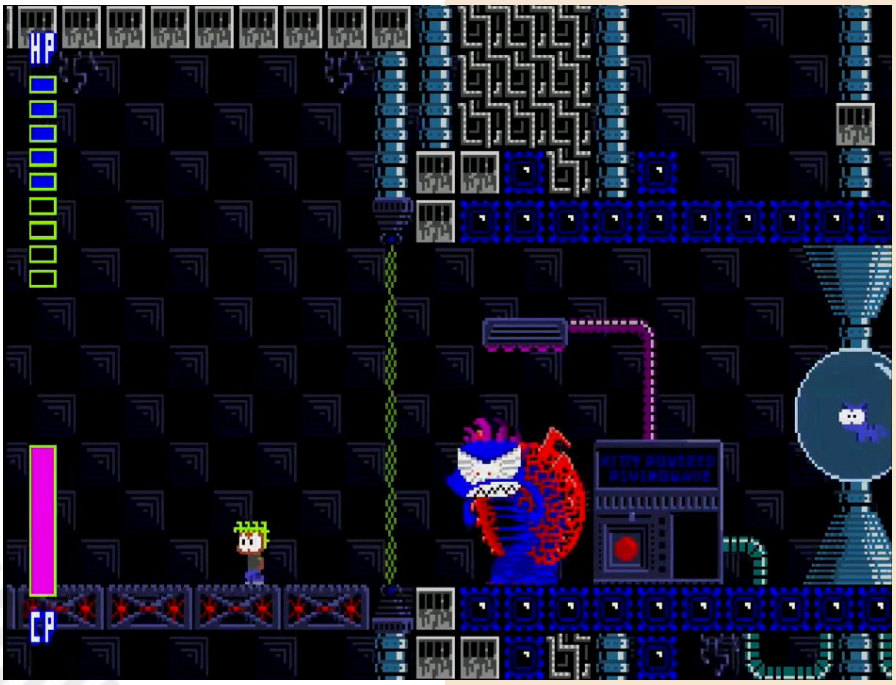
Last night, while I was sleeping, a hooded figure entered the yard and kidnapped my cat. Needless to say, I find this intolerable: Keddums is my only friend! So, as I write these lines, I have decided to set out to reclaim my furry, cubic friend.

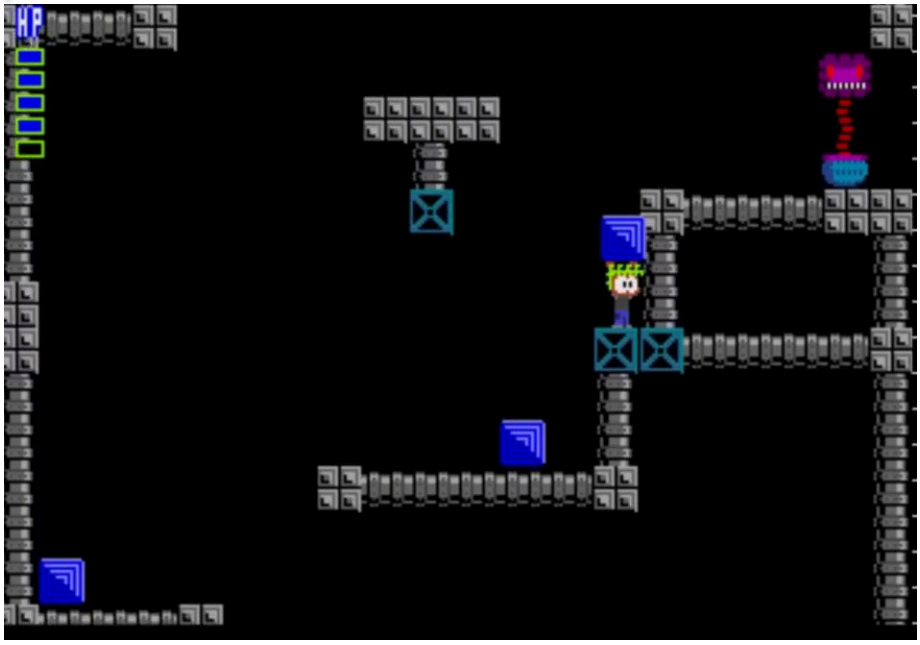
But this adventure is not easy... damn, as soon as I leave the house, I'm as defenseless as a worm. I can't even jump on enemies' heads like that mustachioed Italian-Japanese plumber does. I have to find power-ups to improve myself. Obviously, they're all focused on the cubicity of the game.

But it's fun, come on. I can throw cubes, lift cubes, kick cubes, and attack enemies with cubes.

In the end, it's a cube's life, but it's fun."

Lyle in Cube Sector is a game that appeared some time ago on PC and only now sees a version for Sega

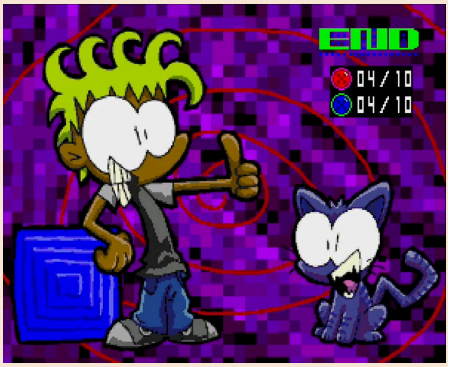




OUR FINAL SCORE

» Gameplay 90%
Extensive but enjoyable in places. Lacks simplicity at the beginning.

» Longevity 75%
Sometimes you find yourself wandering aimlessly.
Initially, it is not easy to stay alive.



Megadrive that is practically identical but definitely more "enjoyable" in terms of speed.

It's a Metroid-style game. The classic Metroidvania that's so popular these days. It shares the exploration system of the famous Nintendo game but fortunately is less 'vast' and allows for more focused exploration.

It also has a strong dose of humor and a nice, polished, rather minimalist graphic style that looks great on the Sega console.

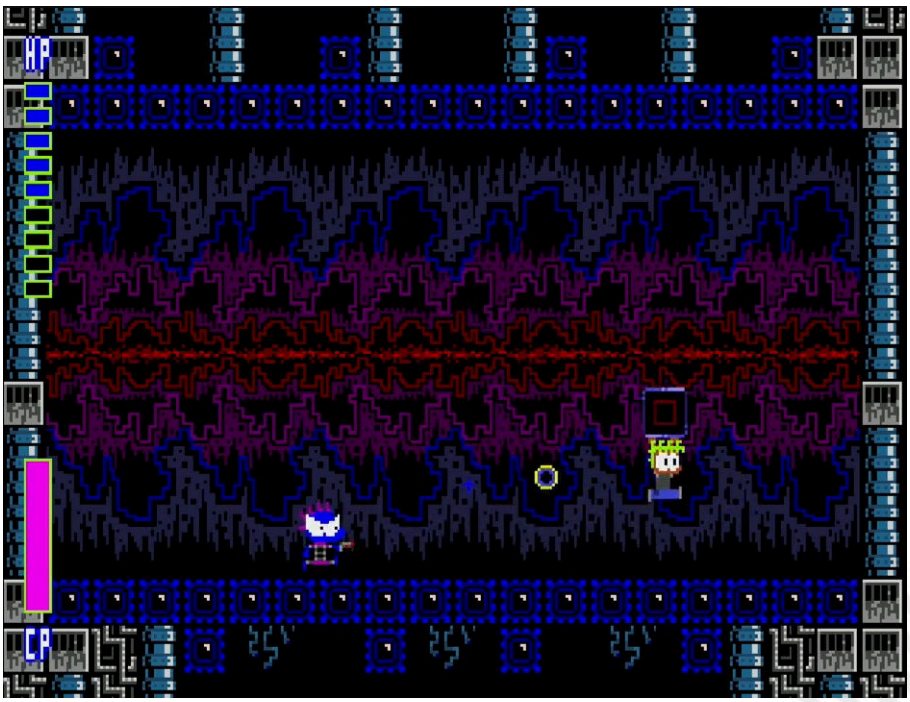
It's not a perfect title, of course. In some places, the plot is scattered and offers little guidance on how to move through the adventure. This will lead us to explore the same places over and over again and, in the long run, could bore us.

Another point is the initial difficulty, which is really 'tenacious'. Starting without upgrades and very weak will often lead us to game over. I would have balanced this better with something more accessible.

But overall, this game is worth downloading and enjoying.

There is a lot to explore and it will keep you busy for a while. The final sequence is fun.

by **Carlo Nithaiah Del Mar Pirazzini**



ATLANTEAN

Year: 2024

Editor/Developer: Aetherbyte

Genre: Shoot em Up

Platform: Pc Engine

Nowadays, the PC Engine is anything but a footnote in the history of video games for many people.

True video game lovers can't help but appreciate it as one of the most fascinating classic platforms, and having a new shooter released 20 years after its demise is nothing short of exceptional.

Developed by a small independent studio in 2014, Atlantean is effectively the result of a labor of love aimed at a hardcore fan base, like many other examples that have preceded it in recent years.

Atlantean doesn't try anything new or revolutionary, and at first glance could be mistaken for a pseudo-sequel to the universally acclaimed Deep Blue. Fortunately, the only thing the two games have in common is the underwater theme, with the actual gameplay being very different from each other. The homegrown homage to Defender is clearly evident, and since there is no game with an equivalent style in the PC Engine library, we can all hail it as the first of its kind for NEC's video game console. Sure, it took more than 20 years to get a new title, but now that it's here, I imagine some people will get a taste of the old Defender gameplay rush.

Anyway, the player's mission is to protect the underwater inhabitants of the planet Atlantis from being kidnapped by the robotic Aquanoid

invaders for unspeakable purposes. To accomplish this heroic task, you can shoot (button II), activate bombs that clear the screen (button I), and move horizontally at hyper speed (START). The playing field is the same as in Defender and its thousand clones,

OUR FINAL SCORE

» Gameplay 80%
Technically not the best, but definitely playable even if it suffers from some slowdowns.

» Longevity 80%
A good level of difficulty.

the scrolling speed is dictated by the player, and a radar at the top tells you the position of all enemies and also of the Atlanteans you should protect from abduction. Turning left and right must take into account the continuous combination of speed and inertia.

Each stage consists of five sections. In the first, you will face three consecutive waves of invaders, in the second you will be in the center of the screen to cross the cracks in the mine walls as they approach, and in the third, another wave of invaders. Then you reach the "Aqualord" boss, followed by a bonus section where you must capture falling Atlanteans to have the opportunity to repopulate the kidnapped or dead inhabitants. This bonus round is only absent in the 4th and final level, as the game ends when the 4th Aqualord is defeated.

The sounds and graphics are fairly average by PC Engine standards. The total lack of power-ups is a bit disappointing, but at least the basic gameplay is enough to keep things interesting. Parallax layers are used throughout and can cause some stuttering (particularly in the mine section), while the colorful design feels monotonous and doesn't allow any stage to stand out from the others. There is a destroyed Statue of Liberty somewhere in the game, but what exactly is it doing on an alien planet? Joking aside, the game fails to go beyond the basic level of implementation.

Overall, the only real downside to the actual gameplay is the noticeable slowdown when you encounter too many enemies at once, which briefly makes the controls unresponsive and affects the fire/bombing functions.

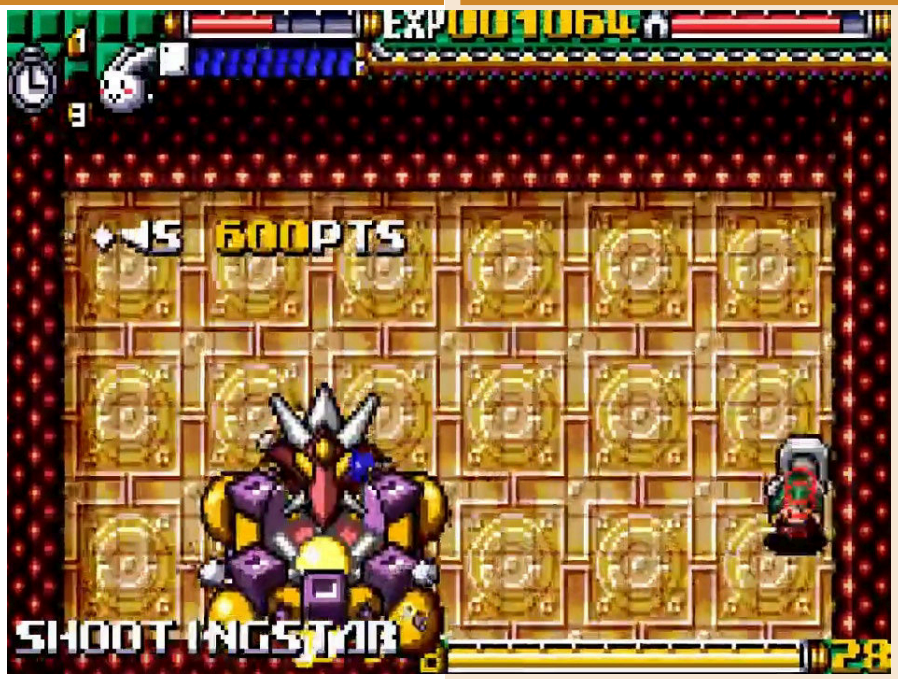
All in all, it's a title that may not be technically outstanding but is still fun. It's not among the best shooters for the console, but it's worth a play every now and then.

by **Carlo Nithaiah Del Mar Pirazzini**

DICING KNIGHT PERIOD

Year: 2004
Editor/Developer: Qute/Platine Dispositif
Genre: ARPG
Platform: WonderSwan Color

Dicing Knight Period is a very clever game. Now, I don't mean that in the sense of a deep plot, intricate game mechanics, and clever in-game puzzles. In reality, it's nothing more than a humble action game released towards the end of the WonderSwan handheld console's life cycle (we talked about this unfortunate console in issues 37 ita and 15 uk of RetroMagazine, go check them out, Ed.).



But it's the way Dicing Knight encapsulates that action that makes it so clever.

The WonderSwan was a handheld system with a very distinctive design, and this game reflects that.
From the small rooms to the action, it's amazing how much this title demonstrates the unity between the platform and its design. It's a much stronger experience than it initially suggests.

But what exactly is Dicing Knight? Calling it an action game doesn't say much about what it is and how it works. Instead, it's better to describe it as a mix of gameplay: the action of Final Fantasy Adventure and a deep exploration of roguelike dungeons. You swing your sword at various monsters and explore a dungeon, room by room, floor by floor.
Obviously, there's not much to say about a game like this. It doesn't require any special skills to play, and with only two buttons to interact with, it's a simple game. But this is where DK's strengths lie (let's abbreviate it for convenience!).

Its simplicity allows the game to strike a perfect balance between relaxing and engaging! It's as if the game distills fun into a pure, unrestricted

OUR FINAL SCORE

» Gameplay 95%
Everything is done with the directional pad and two buttons, and it's satisfying.

» Longevity 90%
Balanced and structured. It leads to absolute Nirvana for those looking for something that gives complete satisfaction.

form. It's amazing how easily I found myself rummaging through the game's dungeons, wandering around looking for activities to occupy my time.

Much of the fun comes from the focus of the design; every element serves to provide the player with quick enjoyment. Hitting the enemy, for example, gives you a tangible sense of power. The immediate feedback is evident. The levels are balanced. Rooms are never larger than a single screen (reminiscent of the simplicity of Zelda for NES), and enemies usually die with a few well-placed hits. Each of these design choices creates a game built around very small chunks of in-game time, leaving little between you and your enjoyment.

The game does this. It uses time to structure the experience; to give a player something to work with so that it's fun. A commendable intention, which seems obvious but often isn't. Playing this title is fun. Damn fun.

There's a perfect action aesthetic that balances with a gradual and satisfying level of difficulty.

It's a title that brings joy to those looking for something simple and clever to discover amid brain-twisting modern roguelikes or monstrous tutorial-heavy adventures.

Dicing Knight is perfectly aware of what kind of game it wants to be: a fast-paced, action-oriented dungeon crawler that satisfies players. It's hard to think of a better game on the WonderSwan, or even on other systems.
Rediscover it!

by **Roberto Del Mar Pirazzini**

NEW GAME

GENESIS
DAWN OF A NEW DAY

Year: 2010/2024
Editor/Developer: Retroworks
Genre: Shoot em up
Platform: ZX Spectrum
Web site: https://retroworks.itch.io/genesis-dawn-of-a-new-day

No one on Earth had ever taken the threat of the terrible Dork aliens seriously (well, with a name like that... ndNith). No one thought they could attack... but humanity never learns.

The earth is invaded by these bizarre creatures and only one pilot can free it and save everyone.

It goes without saying that we will be piloting this pilot, just as it goes without saying that freeing ourselves from the Dork will not be easy: first we will have to recover all the pieces of the Genesis spaceship and then face them.

However, it is not a given that we will be able to recover the pieces, which are scattered across five different planets. Once assembled, we will head back to Earth, ready to fight and free it from the Dork.

A horizontal scrolling shoot 'em up with all the elements of the classics of the past.

Power-ups, double shots, smart bombs, and lots and lots of enemies to fill our day with healthy space blasting.

You can play it with a Kempston or Sinclair joystick or with keyboard controls.

It's not an easy game, far from it! It has a high mortality rate, and often the best way to tackle a level is to eliminate only the enemies directly in front of us or to prepare some alternative routes.

Persist, persevere, and fight... get a game over and start again... this is

OUR FINAL SCORE

» Gameplay 85%
Power-ups, double shot types, smart bombs... rich waves of enemies and nice level design.

» Longevity 70%
I would have made the game more "human." It seems to have been born from the mind of the Marquis De Sade.

the imperative of Genesis.

The game runs smoothly and the graphics are quite varied and entertaining.

It underwent a very long gestation period but is finally available in both TAP format and physical edition.

A good product for lovers of the genre and for space masochists looking for a title with a strong impact on their nerves.

by **Giampaolo Moraschi**

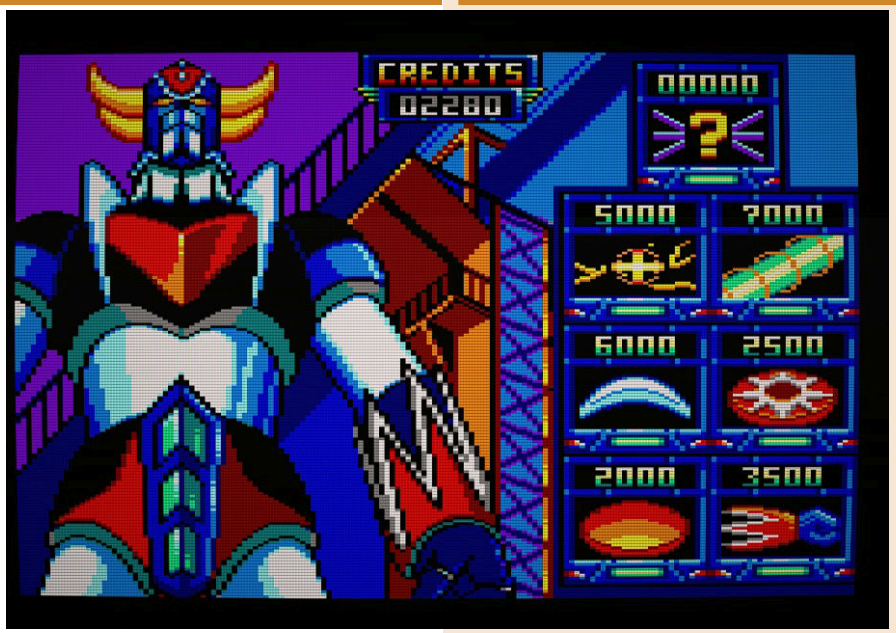
NEW GAME

GOLDORAK

Year: 2024
Genre: Shoot em up
Platform: Amstrad GX4000/ Plus
Web site: https://amstradmuseum.emu-france.info/Fichiers/Projets/Goldorak_CPC/Goldorak_CPC.html

"Go, destroy evil, go! (Goldrake)!
You have a thousand weapons, never give up,because you are good, you are with us."
Italian cartoon theme song

Tell the truth, you were singing it, weren't you?
The theme song written by Luigi Albertelli, Vince Tempera, and Massimo Luca Bandini and sung by Actarus, aka Michael Tadini.
An almost mythical piece from 1978 and the closing theme song of the cartoon Atlas Ufo Robot.
How many memories Goldrake brings back. The first mainstream cartoon launched in 1978 and a masterpiece created by the genius Go Nagai.
The third in the Mazinger saga (Z, Great Mazinger, and Goldrake) and the result of a series of typically Italian atomic errors in the adaptation.
Created in Japan under the name UFO Robot Grendizer, it arrived in Europe through Jacques Canestrier's Pictural Film, which had purchased the rights and changed the name to Goldorak (French edition).
The title Atlas UFO Robot comes from the brochure that Nicoletta Artom, RAI manager for children's products, had seen.
Atlas was not a high-sounding, exotic name, but only the guide/brochure, and UFO ROBOT was the presentation title and was misunderstood as the title for the series.
Over time, attempts were made to claim that 'Atlas' in the title was not a mistake, but a deliberate choice due to the pleasant (but coincidental) reference to Atlas.
Italian mysteries, myths, perhaps



ignorance of the time... the fact is that all this became myth.
Readers are part of the 'Goldrake' generation and are always happy to see games or new productions about the giant robot piloted by Duke Fleed. While waiting for the new Japanese-Arabic series and after seeing the terrifying "Goldrake il banchetto dei lupi" for the new consoles, here I am on Amstrad to talk about this brand

new version.
It is a vertical scrolling shoot 'em up that vaguely reminds us of Banpresto's classic arcade game "Mazinger Z Arcade."
The aim is very simple. You throw yourself into the fray with Goldrake to recover the pieces of Alcor's (Koji Kabuto) flying saucer, which have been destroyed and hidden by the Vega fleet (these vegans are too stressed... lack of B12).
Obviously, recovering the pieces of the flying saucer is no easy task, given that the inhabitants of Vega are rather aggressive and ready to kill the horned robot.
To defend ourselves, we have basic weaponry called Gamma Missiles, but we can upgrade our robot by upgrading its defenses every time we return to base.
The available weapons are all well characterized and have more or less decisive functions in certain battles.
Our robot starts with three lives and an energy level that is consumed with each collision or hit received.
When you reach zero lives and zero energy, it's game over.
What can we say, it's a classic of its genre with the classic waves of enemies falling from bottom to top.
The graphics were created entirely in Multipaint, a really well-made program that we recommend checking out at this address:
http://multipaint.kameli.net/.
The game is very colorful and well characterized and, even if it doesn't have a lot of stuff on the screen, it's pretty fast.
The enemies are easily distinguishable and there are a variety of bosses.
The intro by Eric Cubizolle (TITAN), who was responsible for all the graphics and the script, is beautiful.
The musical accompaniment by Pulsophonic is not bad. The original theme songs are reproduced and the sound effects are interesting.
The game runs on the unfortunate GX4000 or on any CPC Plus and in emulation.
For fans of Nagai's mega robot, it's a nice product, but perhaps a little boring. Personally, I would have gone a little further in the number of enemies and their variety on screen.
The game is certainly very interesting and worth trying.
From what we have seen, versions for Atari 8-bit and other platforms will also be available in the future.
It is a well-designed product, complete with a manual and website with all the documentation on development, music, graphics, and much more.
If you love this genre, giant robots, Go Nagai, and are from the Goldrake generation, don't miss out.

by **Giampaolo Moraschi**

OUR FINAL SCORE

» Gameplay 80%
Easy to play and with good level design. The waves and types of enemies are not very varied, but it is interesting. Every aspect (graphics, sound, and presentation) is well done.

» Longevity 70%
It's not difficult and unfortunately not very varied.

NEW GAME

CECCONOID

Year: 2024
Editor/Developer: Thalamus/ Triple Eh?
Genre: Shoot em up
Platform: Amiga
Web site: https://thalamusdigital.itch.io/cecconoid-amiga

Cecconoid is a typical arcade game. Competitive, rigorous, difficult... A modern title that could very well come from a parallel dimension where we are in the midst of the Golden Age of video games.

It is reminiscent of Cybernoid, but with the appeal of a modern roguelike and a touch of Metroid.

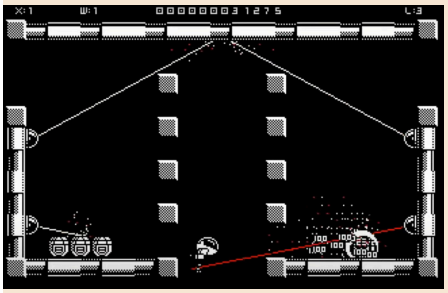
The story is as fascinating as its black, white, and red graphics. It exudes 80s vibes.

The Equinox spaceship is under attack. Stormlord and his henchmen are destroying everything inside, and humanity's only hope is the Samurai-1 attack capsule, i.e., us.

The mission involves recovering Captain Solomon's key and opening Stormlord's core, i.e., destroying him. Sounds fun, right? Yes, maybe the story is a bit banal, but the result is definitely entertaining.

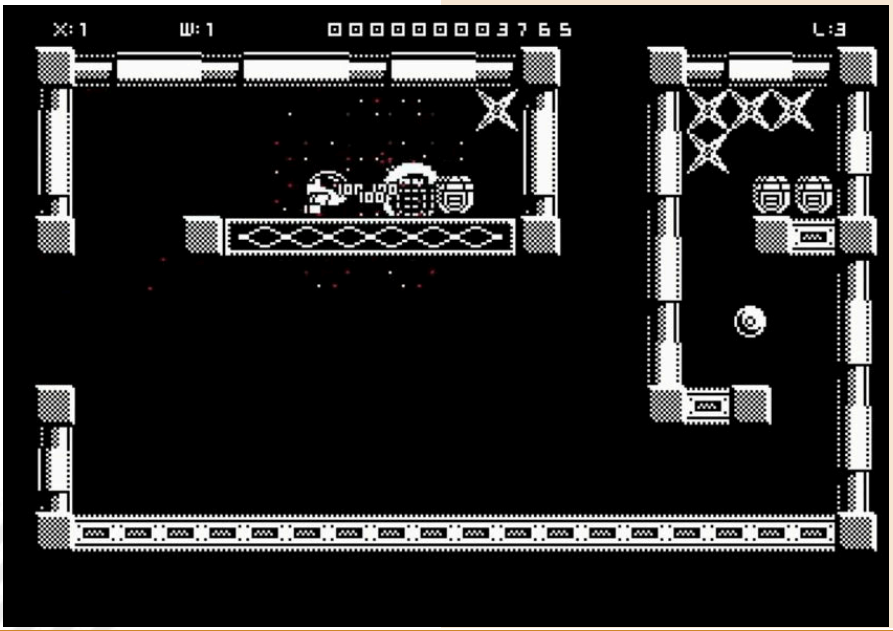
The game is a multi-directional side-scrolling shooter that is not fast-paced. It requires attention in combat and positioning. You will focus on precise tactics to face enemies with your pixelated spaceship. Enemies are definitely fierce and ready to kill you. Not only other spaceships, but also missiles, mines, traps, laser beams... in short, anything that moves on the screen wants to destroy you.

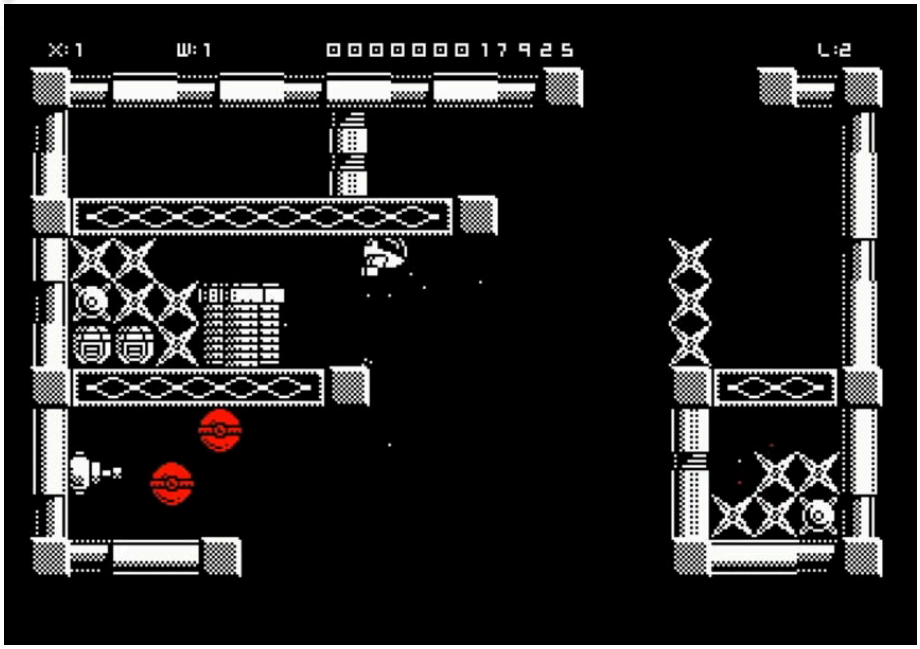
The game uses two joysticks: the first joystick is used to move, the second to aim your shots. In the initial menu, you can configure it however you want and according to your needs.



The game mechanics are very unique and also present a very high level of difficulty, suitable for experienced players and not casual gamers.

Aesthetically, it is a great product. The black and white (and red) graphics are a gem. A combination of old and new with remarkable animated effects.





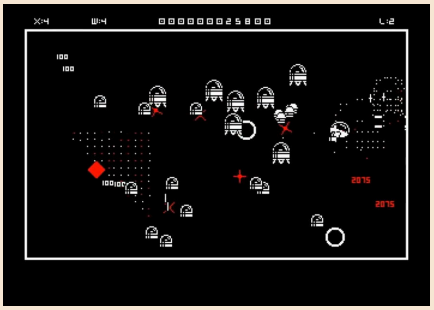
OUR FINAL SCORE

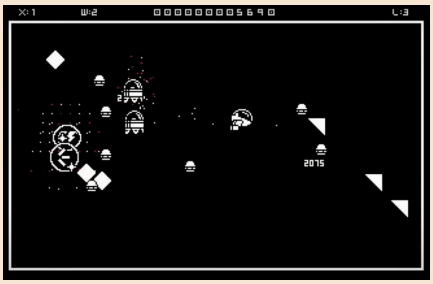
» Gameplay 90%

The mechanics of the two levers are interesting and stimulating.
Eugatron is a great added value.

» Longevity 85%

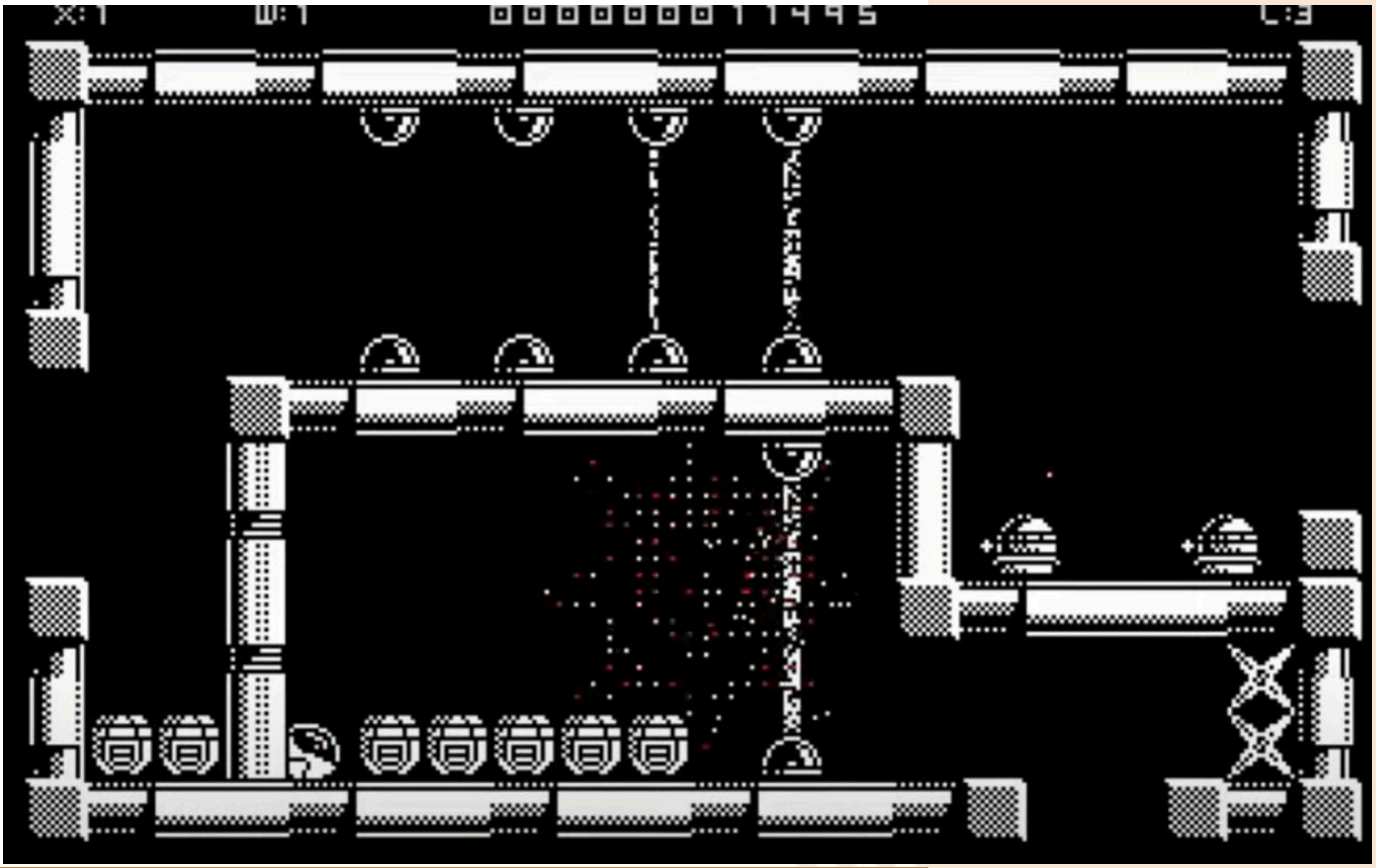
It is not an easy game and requires a lot of manual dexterity.





The audio is also incredible. The soundtrack is spot on.
The title also features the game Eugatron. An even more "extreme" version of the main game.
A deathmatch game where you have to face hordes upon hordes of opponents in the style of classics such as Robotron 2084 or Smash TV.
Beautiful, thunderous, destructive!
For fans of this type of game, Cecconoid is a phenomenal experience. High difficulty, beautiful graphics, great sound, and good level design. Perhaps suitable for more "experienced" players, but definitely a great title.

by **Carlo Nithaiah Del Mar Pirazzini**



NEW GAME

MIKIE

Year: 2024
Developer: Krzysztof "Vega" Góra, Michał "Miker" Szpilowski e Krzysztof "Kaz" Ziembik
Genre: Action game
Platform: Atari 8bit

Mikie was a 1984 arcade video game developed by Konami with a style that was irreverent for its time.
It took us into the stereotypical world of American high schools in the shoes of a funny kid with a pompadour hairstyle and "rough" manners, with little desire to study and a great desire to be with his girlfriend who was outside the school.
The game unfolded in five scenarios: The first was during class, where the blond "tamarro" knocked his classmates off their chairs and collected little hearts from under them, while the teacher tried to stop him by throwing his dentures at him. The second saw us in the locker room grappling with the teacher, a rough janitor, and the cook. Here, too, we had to escape.
The third was the cafeteria, where we had to avoid the cooks and the increasingly furious teacher.
The fourth was the gym, and finally we arrived at the courtyard where, avoiding large American football players, we could reach our beloved. Only three lives and a lot of manual dexterity were needed to complete the story.
The game was initially converted for Sega SG 1000 and later for Amstrad CPC, Commodore 64, ZX Spectrum, Acorn Electron, and BBC Micro... so for all the 8-bit systems of the time... All that was missing was an Atari 8-bit version, and finally it has arrived.

The title was created by a group consisting of Krzysztof "Vega" GÃ³ra, MichaÅ, "Miker" Szpilowski, and Krzysztof "Kaz" Ziembik. It runs on Atari with 128k in PAL format, which means machines such as the 130XE and machines equipped with memory expansion, or on Altirra in emulation

OUR FINAL SCORE

» Gameplay 90%

It retains the characteristics of the coin-op and is impeccably made.
Perfect controls.

» Longevity 80%

One game leads to another, even if it will drive you crazy many times.

(properly configured).

And we have to say that it runs very well and is as fun as it used to be.

It uses Antic 4 graphics mode, i.e., 40x24 characters in five colors. It moves fast and looks great.

The sound is beautiful. It features the famous songs that characterized the coin-op, namely A Hard Day's Night and Twist and Shout by The Beatles.

The gameplay is a product of its time. It's a game from 1984, a coin-eater, and it will often drive you crazy. It won't be easy to get past the professors, janitors, dancers, teachers, and various thugs present, but as with the coin-op, the desire to start again even after "dying" is overwhelming. One game leads to another.

Another great title for this new era of our beloved retro systems.

by **Carlo Nithaiah Del Mar Pirazzini**

Website:

https://atarionline.pl/v01/index.php?ct=kazip&sub=M&title=+Mikie#kaz_c559e7d8dc286f05c79af72fa972060f

NEW GAME

ROGUECRAFT

Year: 2024
Editor/Developer: Thalamus, BadgerPunch Games
Genre: Dungeon Crawler/ Roguelike
Platform: Amiga
Web site: https://thalamusdigital.itch.io/roguecraft

I have always enjoyed Dungeon Crawlers. This genre has some features derived directly from role-playing games, but simplifies them by placing them in labyrinths where spoken dialogue is less important and action is more necessary.
One of the first video games of this genre was Pedit5, created in 1975 by Rusty Rutheford. It was a basic title that included several features from D&D and the presence of many treasures within the adventure.
Among the most famous and beloved (especially on Amiga) are the ever-praised Dungeon Master, the Eye of the Beholder series, and the various Ultima, Diablo, and company.
And it is on Amiga that we find this genre again with Roguecraft, a title developed by BadgerPunch Games and distributed by Thallion, which takes us into the dangerous world of Mordecom.
We will be guiding Zendar, one of the last true heroes of his era, as he sets out in search of a powerful amulet hidden in the cursed dungeons.
Before doing so, however, we must choose Zendar's class.
We can choose between the Warrior (easy mode), who is resilient and equipped with a sword; the Thief (medium difficulty), who is lighter, armed with dual knives and the ability to teleport quickly; and finally the Mage (maximum difficulty), who is physically weak but has the ability to cast powerful magic spells.
Each game class has its own maximum health indicator shown at the top with small red hearts and a strength indicator just below with a blue bar.
It is possible to increase both health and strength by killing a certain number of enemies or by collecting certain items or potions. Killing a large number of opponents will release a red gem (increases hearts) or a blue gem (increases strength) in the room.
Zendar moves through rooms that are revealed on a side map as they are explored.
The goal is to find the staircase that

leads to the next level. This is no easy task, as the key is usually hidden somewhere in the dungeon itself and well protected.

Once retrieved, the door to the new level opens and is marked on the map with a blue dot.

Roguecraft's dungeons are procedurally generated and feature the classic menu of this genre of games: treasures, secrets, traps, and a ton of enemies (BEWARE OF MIMICS DISGUISED AS TREASURE CHESTS!).

As the adventure progresses, not only will we be able to upgrade ourselves, but we will also encounter increasingly tougher enemies, so we recommend balancing your strength wisely before tackling the lower levels.

The game is turn-based, with Zendar moving on his turn and the monsters in the room moving in response to his action. You engage in combat by moving towards the monster's turn and facing it in the adjacent square (or by throwing a magic projectile from a distance if you choose the Wizard).

The battle will take place automatically. The fire button is used for the inventory. Pressing it takes you to a panel at the bottom where you can select and read what you have collected during your wanderings.

These are usually potions, and there are several types: the green potion is a classic healing potion, while the blue and red potions change effect in each game.

Possible effects include Freeze, Explosion,

OUR FINAL SCORE

» Gameplay 95%

Beautiful procedurally generated levels and an excellent gameplay system. The dungeons are always varied and well-designed.

» Longevity 91%

The three game classes offer different approaches to dungeons. Playing as a wizard is hell, but the game never gets old.

Revelation, and Teleportation. There is also a crazy bottle containing Alcohol that will test the nerves of even the most experienced players with its curious effects.

If you die before reaching the final mega boss, a Game Over screen will appear summarizing your score, the level you reached, how many chests you recovered, how many secrets you revealed, and... which monster killed you!

The development of Roguecraft has been quite long and eagerly awaited. Having got our hands on this press version, we can say with certainty that it was worth the wait because Roguecraft is a real gem. It's beautiful to look at and listen to. A small pixelated work of art that amazes with its details and wonderful color choices.

The detail for every single thing is obsessive. An example? The monster animations. Wonderful.

There is a lot of variety in tackling the levels with the three game classes, there is variety in the dungeons and monsters.

And what about the soundtrack... it's a blast that engages and creates a unique and seductive atmosphere.

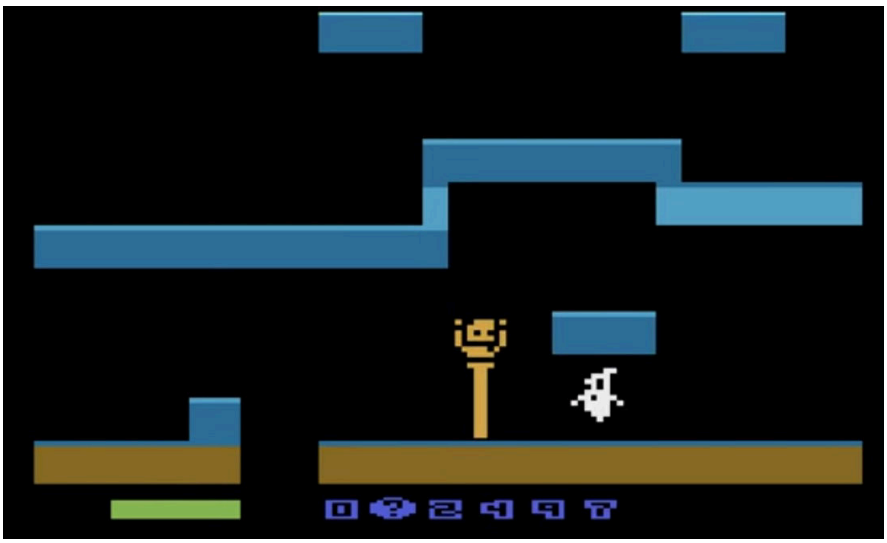
The strong point, however, is in the level design and the way you get involved in the exploration. If at first it seems like a simple exploration game, as the games progress it becomes addictive. The only tiny criticism is the difficulty in 'Wizard' mode. In fact, the level of challenge with this class is almost impossible. Like true heroes. It's a minor issue, but it is recommended that you practice extensively with the other classes before embarking on the adventure with the wizard. Roguecraft can be played on Amiga with ECS/AGA chipset, 1 Mb of Chip RAM and 1 Mb of other RAM, and a 68000 processor. The guys at BadgerPunch recommend an A1200 configuration for the best gaming experience and fewer slowdown issues. Obviously, it works without any problems on any emulator and on the A500 mini.

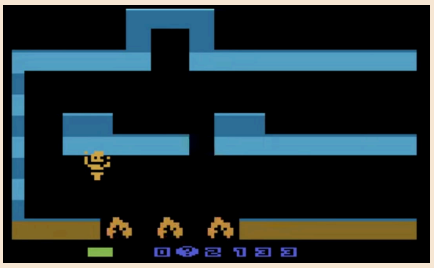
by **Carlo Nithaiah Del Mar Pirazzini**

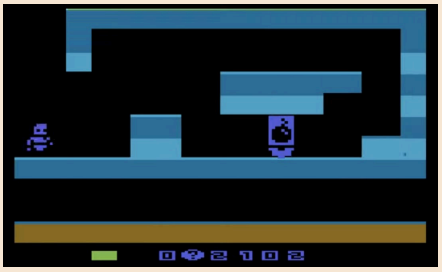
NEW GAME

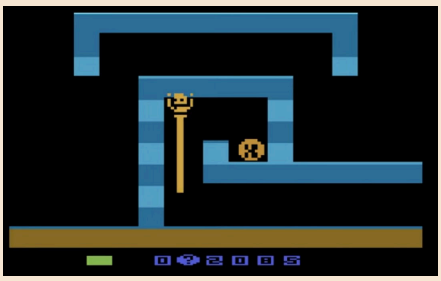
ROBO TITO

Year: 2024
Editor/Developer: VHZC Games
Genre: Platform/Action
Platform: Atari 2600
Web site: https://vhzc.itch.io/robotito









The little robot Tito has only one flaw. His programming didn't go well and he suffers from a terrible "allergy" to monstrous things, ghosts, and monsters.
Unfortunately for him, he finds himself trapped in a terrifying place infested with all the most horrible creatures in creation.
However, the place is on our Atari VCS and the game is really fun in its simplicity.

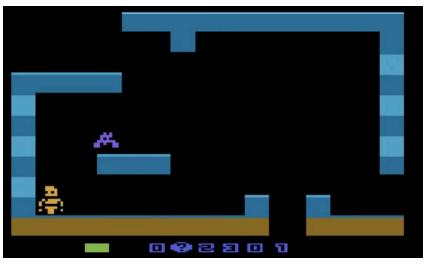
Tito is not armed with any laser beams or super strength, but he can stretch and cling to the walls above to escape traps and monsters.
It's easy for us! We have to help him get out of there.

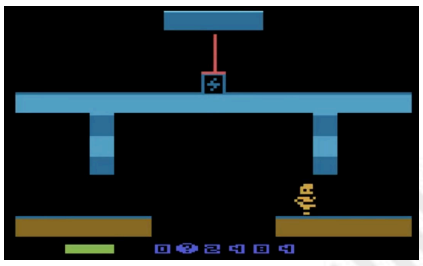
The game is simple and fun, and I must admit that I enjoyed playing it on the VCS that was lent to me.
It's quite a challenge to get Tito to safety, and it doesn't require any kind of "brain-twisting" system. You just have to run away and understand the movements of your enemies as best you can.

A simple game, carefully programmed and very entertaining.
I really liked the sprites of Tito and some of the monsters (I love the ghost).

The game is fast-paced and easy to love. Is there anything else I can add? Not really, except to support the developer and upload the game to VCS or emulation.

by **Marta Rossmann**





OUR FINAL SCORE

» Gameplay 90%
I really liked the mechanics and level design. I love the sprite.

» Longevity 80%
A balanced challenge. Not too difficult or too easy. Very enjoyable.

NEW GAME

THE HEART OF SALAMANDERLAND

Year: 2024
Editor/Developer: Juan J. Martinez
Genre: Platform
Platform: Amstrad CPC/ GX4000
Web site: https://www.usebox.net/jjm/heart-of-salamanderland/

Patton is the greatest adventurer of his time. Always ready to embark on reckless adventures; between Cursed Temples, Lost Arks, or mysterious Templar relics. Always ready to unroll his whip... Hey! Am I wrong, or does he remind us of Indiana Jones?

The Heart of Salamanderland is a platform game for Amstrad CPC (64k RAM and above) and the protagonist is clearly inspired by Lucas' famous hero.

The goal is to recover a relic hidden throughout 55 rooms full of monsters and traps. Patton's whip will be our only weapon of defense/offense, and to get to the sacred relic, we'll have to explore the dungeons and recover the guardian's nine tears. At that point, we'll have to kill the final mega boss and claim our well-deserved prize.

The structure is reminiscent of the most well-established platformers of the golden age. The levels are well designed and generally not too challenging or convoluted, but structured with an increasing difficulty curve.

The title was developed by Juan J. Martines, who was responsible for the graphics, sound, and code. It's a job well done, as everything is very colorful, well animated, and accompanied by good music.

Eric Cubizolle, aka TITAN, also had a hand in the game, creating the (very beautiful) introductory screen inspired by classics such as Castlevania, Rick Dungerous, and Montezuma's Revenge.

The game supports two buttons: one for the whip and one for jumping.

And jumping is the key. The Heart of Salamanderland belongs to that genre of games where precision from platform to platform is the key to seeing the credits roll.

Jumps must be "perfect."

The title is available for free download at usebox.net, which I have linked to in the box below. The authors are also working on a physical edition scheduled for release in late 2024 on the Poly

OUR FINAL SCORE

» Gameplay 80%

The two game buttons are essential. Jumping with just the joystick can lead to exhaustion. The game levels are well structured.

» Longevity 75%

55 game screens may seem like a lot, but in reality, with a little practice, the task is easily accomplished.

Play website.

But is it a good game in the end? Yes, it's a nice platformer that's fun to play and can be replayed as many times as we want without pretending to be an absolute masterpiece.

Passed with flying colours.

by **Marta Rossmann**

NEW GAME

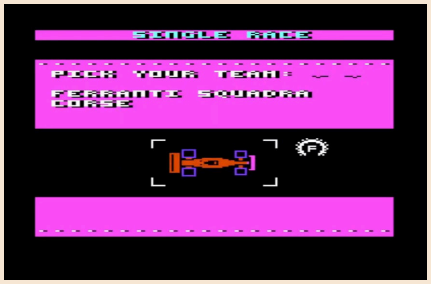
FORMULA V20 1985

Year: 2024

Editor/Developer: Huffeduff

Genre: Driving

Platform: VIC 20 (+24kb di espansione)



Formula V20 1985 is a racing game for VIC 20 that pays homage to classics such as Pit Stop II and Pole Position. It was designed to be played on a 24k expanded VIC.

The game was developed in Kick Assembler, a versatile tool also used for other titles by the developer.

You can select from seven different teams and compete on four different circuits. You can choose championship mode or race in a single race, as well as choose the difficulty level.

The game is very fast and moves well. The graphics are convincing and fairly detailed, and the sound effects are appropriate for the genre.

It plays quite well with the help of the pad/stick, and the difficulty level is balanced on the tracks, starting from a low level up to the maximum possible challenge.

This is a good product for VIC, suitable for fans of the genre and lovers of Pit Stop 2 and similar games.

It can be run on Vice in emulation by selecting the memory expansion options in the settings.

by Giampaolo Moraschi

OUR FINAL SCORE

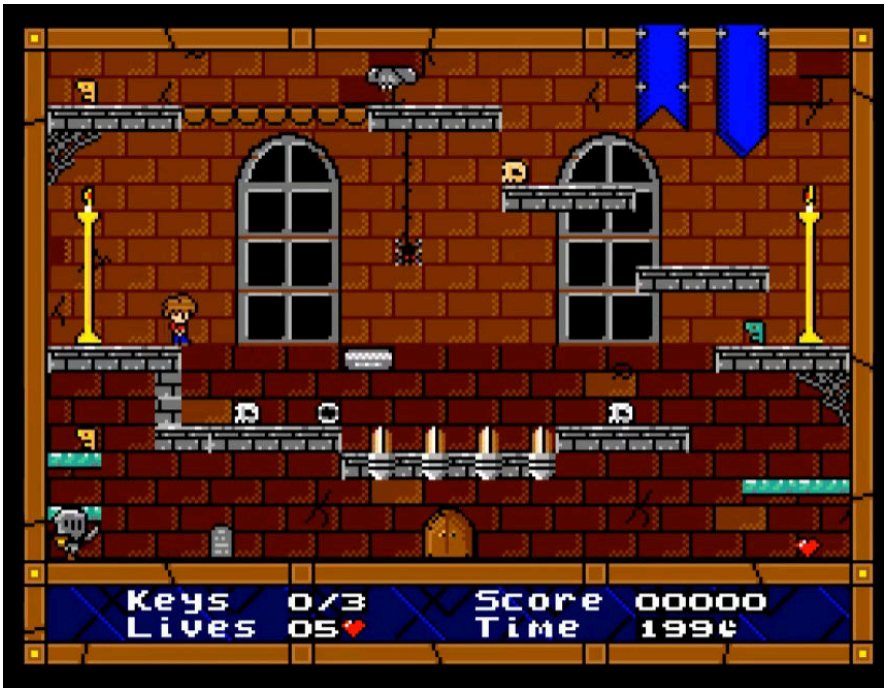
» Gameplay 75%
A good racing game for expanded VIC, along the lines of historical titles.

» Longevity 70%
I would have liked more tracks.

NEW GAME

PRINCESS PALOMA'S RESCUE

Year: 2024
Editor/Developer: InfiniteMSX
Genre: Platform/Puzzle
Platform: MSX2 (64kb/128kbvm, MSX-Music)
Web site: https://www.msxdev.org/2024/05/03/msxdev24-01-princess-palomas-rescue/



The first title in the wonderful MSXdev24 competition is Princess Paloma's Rescue, an action game for MSX2 in which an intrepid hero embarks on a quest to rescue a kidnapped princess.

I love MSXdev; I think it's one of the best competitions for new titles ever, and every year it churns out and showcases little gems.

This solid game from InfiniteMSX is in pole position to attract the attention of the judges and us industry insiders.

Princess Paloma has been kidnapped by an evil wizard named Morgath, and our hero has the task of crossing five distinct kingdoms, recovering keys, and trying to reach the beautiful Paloma unharmed.

As always in this genre of games, it is no easy feat. It is a title full of traps and obstacles, and it takes ingenuity, a little cunning, and gray matter to get through the levels.

The title is native MSX2 and requires MSX-Music functionality, the 1987 audio standard.

Aesthetically, it is pleasing in every way. It has cute, well-animated graphics and beautiful music accompanied by good sound effects.

Be warned! It is not an easy game and tests your reasoning skills in certain situations.

Some levels are definitely difficult to complete.

However, it is a great little game and we just have to wait and see what else this 2024 edition of MSXDev has in store.

by **Giampaolo Moraschi**

OUR FINAL SCORE

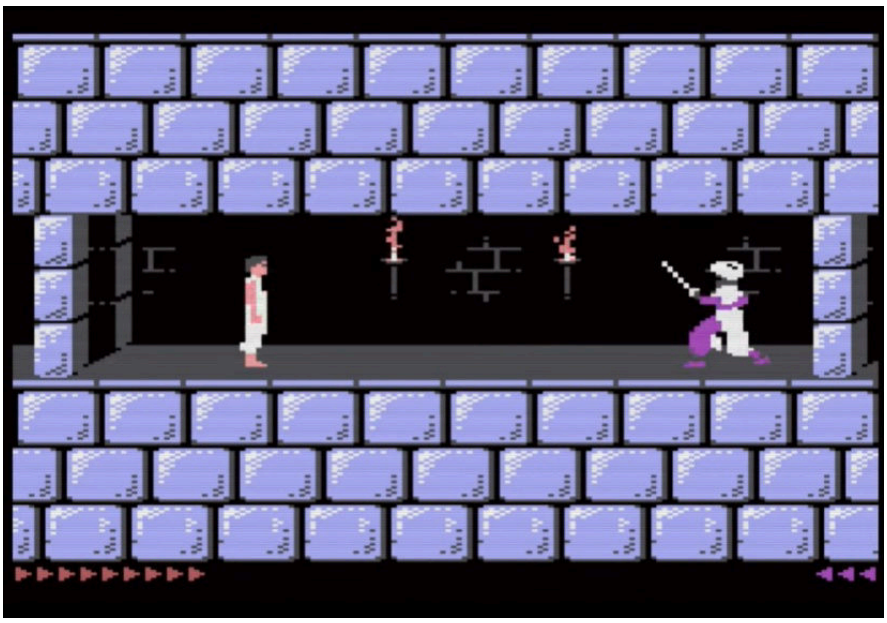
» Gameplay 85%
Simple mechanics and beautiful game levels.

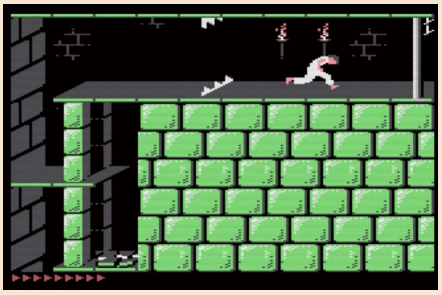
» Longevity 80%
Difficult in some places and requires attention.

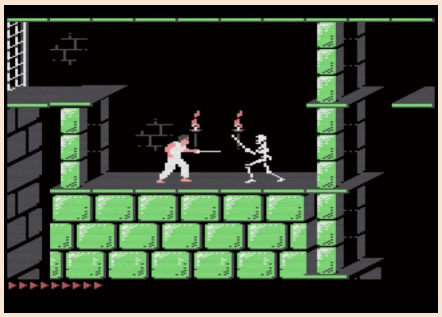
NEW GAME

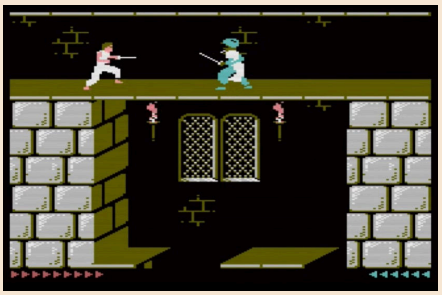
PRINCE OF PERSIA

Year: 2024
Editor/Developer: TCFS
Genre: Platform
Platform: Plus 4
Web site: https://plus4world.powweb.com/software/Prince_of_Persia









Long live the Plus 4!
A machine that, in my opinion, "had little time."
Little time to be known by the public, little time to be known by developers and programmers, and, above all, little time to be taken seriously by Commodore itself.

Yet, in 2024, it flexes its muscles.
The team composed of TCFS, Unreal, and Csabo have shown us over the years how excellent knowledge always leads to impressive results.
Prince of Persia for PLUS 4 is a marvel!
Colorful and animated with great care.
Detailed, fast, with all the bright and vivid colors of TED.
The only flaw is the sound, which slows down a bit.

The Plus 4 is a home computer that needs to be rediscovered and appreciated.
The TCFS team is doing just that.
It has given new life to a machine that unfortunately had very little time to be loved.

by **Giampaolo Moraschi**

OUR FINAL SCORE

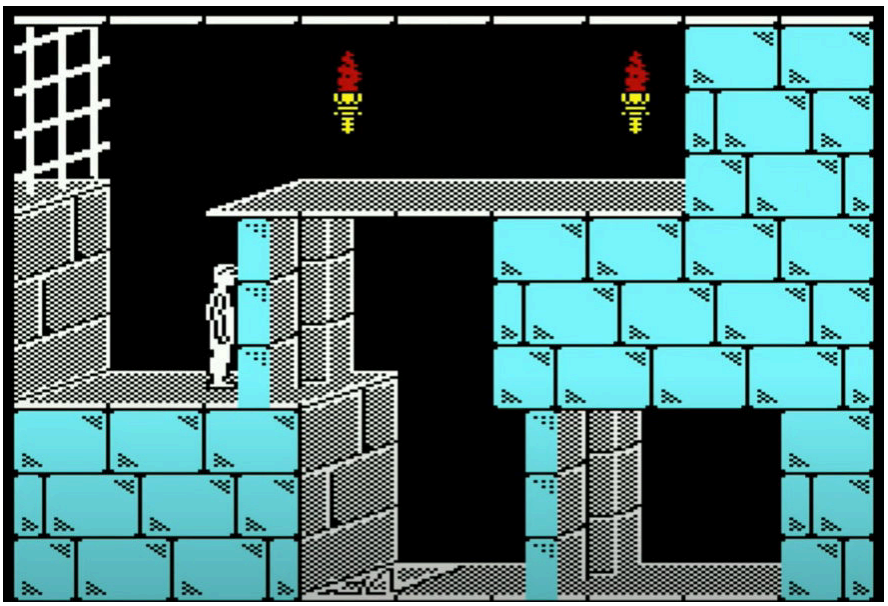
» Gameplay 90%
A feast for the eyes with all the levels and gameplay of the original title.

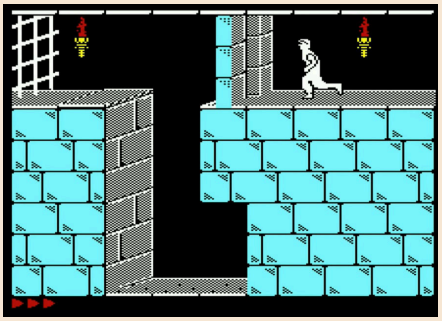
» Longevity 90%
It's Prince of Persia... need I say more?

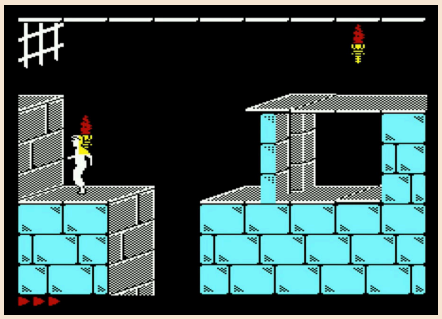
NEW GAME

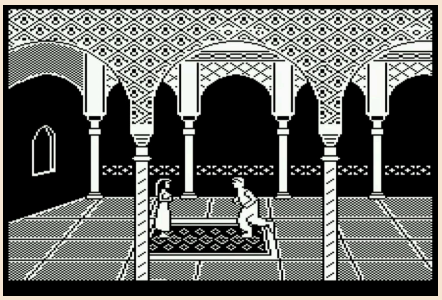
PRINCE OF PERSIA

Year: 2024
Editor/Developer: LeChuck
Genre: Platform
Platform: VIC 20 (+35kb)
Web site: https://sleepingelephant.com/ipw-web/bulletin/bb/viewtopic.php?p=120870#p120870



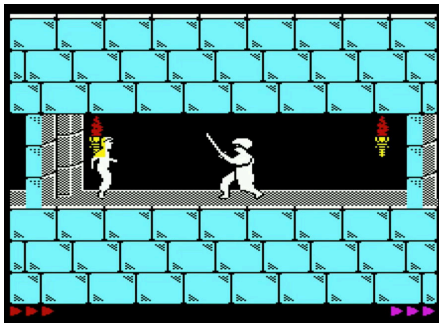


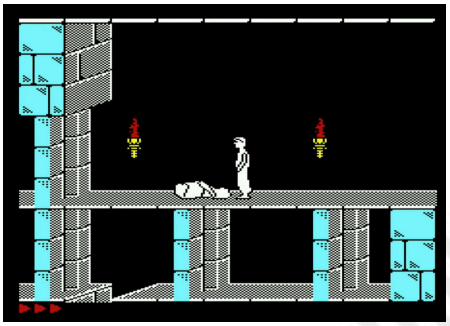




It's been 35 years since the Prince first appeared on our gaming screens. An iconic saga with numerous sequels and a large number of fans.
The charm of the first chapter, however, is undeniable. We've seen it in all its forms, but it seemed unthinkable on a VIC 20.
The memory limitations of Commodore's small home computer have always been an obstacle to the creation of certain titles... Until now... It can be done! And developer LeChuck has done it, inspired by Nicodim's (beautiful) version for the ZX Spectrum. The VIC 20 version of Prince of Persia has only 9 levels (plus the final one) and several shortcomings in terms of animation, but it's... beautiful!
To play it, you need a 35Kb memory expansion and a good joystick, but otherwise, all the magic of Prince of Persia is there.
The game uses 27 columns x 29 rows and therefore runs on PAL formats.
The Prince on VIC moves well and is easy to play, presenting all the peculiarities of this beloved title.
Now we just have to wait for the Plus 4 version, which is currently in development.

by **Giampaolo Moraschi**





OUR FINAL SCORE

» Gameplay 85%
Technical limitations and fewer levels, but it's still a great game.

» Longevity 90%
It's Prince of Persia, a title that wants to be played to the end.

POCKET BOMBERMAN

Year: 1997 versione Game Boy, 1998 versione Game Boy Color
Editor/Developer: Nintendo, Hudson Soft
Genre: Platform
Platform: Game Boy, Game Boy Color

Today I'm going to talk about this little gem, which is a real treasure for Game Boy.

It's a completely different version of the classic Bomberman game because, unlike the usual maze screen (in fact, it belongs to the "Maze" genre like Pac-Man), this is a platform game where our protagonist has to cross five worlds (Forest, Ocean, Wind, Cloud, and Evil) and face a boss at the end of each level. The last one will be Evil (which, strangely enough, they didn't censor). Here's a quick summary of the plot: "A long time ago, the Sun was enveloped by a dark and sinister cloud. An old legend claims that this cloud was the result of a powerful monster that had cast a curse on the "Sword of the Sun," sealing its power. The only hope left to bring light back to the earth is to collect the five Stones of Power, guarded by the evil monsters of Evil Mountain. Only the Stones of Power have the strength to break the sword's curse once and for all and return things to normal."

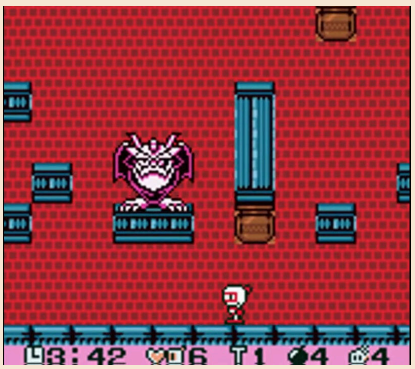
With power-ups that allow us to place more bombs, others that allow us to detonate bombs at will, and many others such as wings that allow us to jump high, we collect our stones to advance through the levels and defeat the various monsters and enemies that stand in our way in a unique adventure for Game Boy!

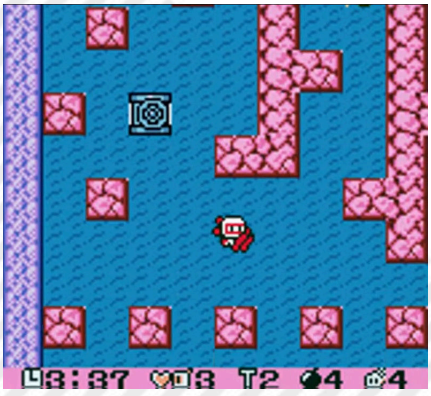
In addition, for those who dislike the gray/green screen, there is the Color version released in 1998. Not content with that, there is also a mini-game called "Jump" mode, where Bomberman must be guided up a slope and explode the various blocks that appear in front of him to continue the path.

A cute and unusual platform game that mixes the classic elements of Bomberman with those of Super Mario. The graphics are simple and basic, as is the sound.

It's a title worth rediscovering, perhaps not an absolute masterpiece but definitely a good game.

by **Barbara "Morgana" Murgida**





OUR FINAL SCORE

» Gameplay 80%
It's a classic platformer with some touches of genius. Easy to play and learn.

» Longevity 60%
Unfortunately, it ends too soon... too few levels!

NEW GAME

THE AMIGA ARCADE CLASSIC CD32

Year: 2024
Editor/Developer: JOTD666
Genre: Compilation di giochi arcade
Platform: Amiga CD32
Web site: https://jotd666.itch.io/arcade-classics-for-cd32

A beautiful compilation like in the glorious Golden Era!

I always got excited when these types of compilations or Budget-style ones came out. It was like a big lottery.

The Amiga Arcade Classic CD32 is an ISO (so it can be booted from CD32) containing all the titles developed by JOTD for Amiga, starting with some arcade classics. Direct ports from Z80 to 68000.

Inside we find Bagman and Super Bagman, Karate Champ 2 player, Moon Patrol, Ms. Pacman and Pacman, Amidar, Donkey Kong, Galaxian, Scramble, Pengo, Pheonix, Tetris, and Xevious.

We reviewed some of these titles in the magazine a few issues ago. They are all excellently ported to Amiga and deserve to be discovered and rediscovered.

Tetris and Xevious in particular are practically perfect.

This compilation is completely free and can be downloaded from the developer's Itch.io page.

It works on real machines and in emulation.

Personally, I would have paid more attention to the title selection menu, perhaps creating something more "graphically" appealing.

Did we need it? No, but it remains a good package of games to try or introduce to our children/grandchildren.

by **Giampaolo Moraschi**

OUR FINAL SCORE

» Gameplay 75%
There are many titles from the past with the gameplay limitations of their era. Some titles are wonderful, others... less so.

» Longevity 80%
There is a lot of "stuff" to play and try out.

MARATHON

Year: 1994
Genre: Sparatutto in prima persona
Music/Soundtrack: Bungie
Editor: Bungie (originale) Team Aleph One (realizzazione di porting moderni)
Platform: Windows/Mac/Linux

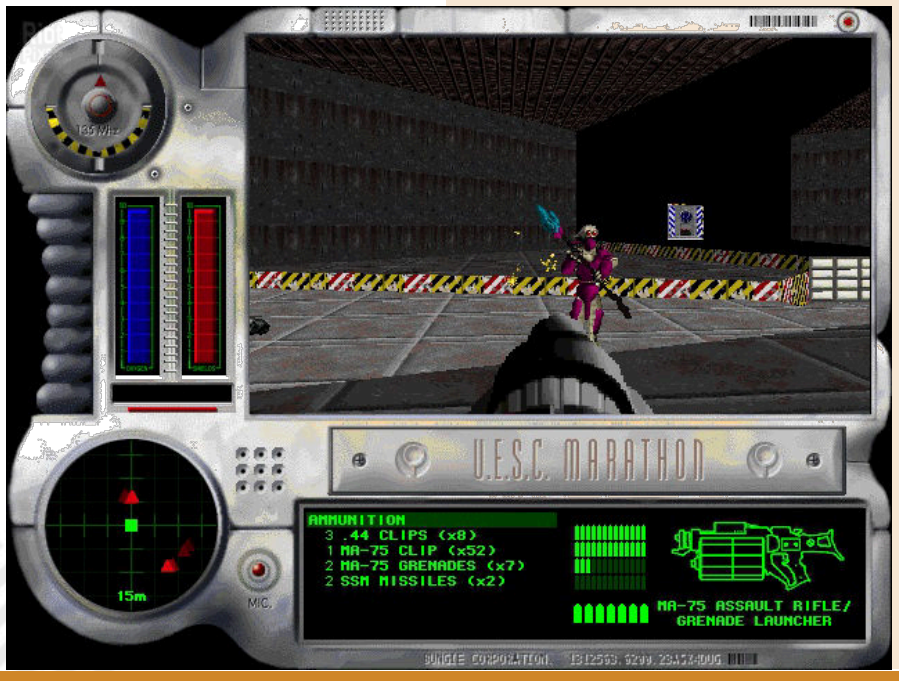
The 1990s were the golden age of first-person shooters, seeing the birth of important titles such as Wolfenstein 3D and Doom, the latter indirectly coining the term "Doom-like," a label that was given to any FPS released from then on (it would be a few years before the term First Person Shooter or FPS became the official term for this genre of video games).

Among the few developers at the time who tried to ride the wave of Doom's success was the now famous Bungie, which attempted to revamp the shooter formula of the era, first with Pathway into Darkness and then, the following year, with Marathon.

Marathon is set in the distant future, entirely inside the colony ship of the same name, which has been boarded by a faction composed of various alien races. The player takes on the role of a security officer whose mission is to save the ship. Throughout the various levels, you will encounter various terminals, each of which will put you in contact with three artificial intelligences called Leela, Durandal, and Tycho. These terminals will reveal important details about the plot, as well as about the aliens and the AIs themselves, who they are and what their intentions or tasks are.

The first notable difference from Doom is the way the plot is presented, which this time is not just a simple excuse to fight your way through enemies, but is an active and important part of the game. The same goes for the characters, both allies and enemies, who are not mere extras.

Another unique feature is the ability to look up and down. At the time (we're talking about 1994), the view in Doom-style shooters only moved left or right.



OUR FINAL SCORE

» Gameplay 80%

Marathon features levels that may be claustrophobic or sometimes confusing, but you will quickly realize that the game environments make it enjoyable and less monotonous. You will find yourself doing everything from simply reading terminals and searching for keys to open doors, to helping crew members in distress or solving puzzles to continue in a certain area.

» Longevity 80%

The game is quite long, with large maps and levels that can often be completed in a non-linear fashion.

There is also the possibility of reloading weapons, a feature that has now become the norm in shooters, although it is only possible to do so automatically when the magazine is empty. Ammunition is not abundant, and you have to be careful not to run out, as it can only be found around the map and in relatively small quantities. There is also a motion sensor, another new feature compared to Doom, which allows you to detect the position of enemies.

In classic Bungie style, the plot revolves around philosophical themes that are central to the story, namely AI becoming sentient and beginning to experience various feelings, a classic science fiction theme, but this time presented in a truly interesting and engaging way.
Since The Source Code was released for free, you can download the game and its sequels for free from Steam or from the website of the team that worked on the port, Aleph One.

by **Maurizio Diamanti**

NEW GAME

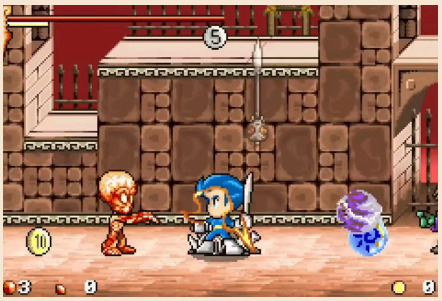
KIEN

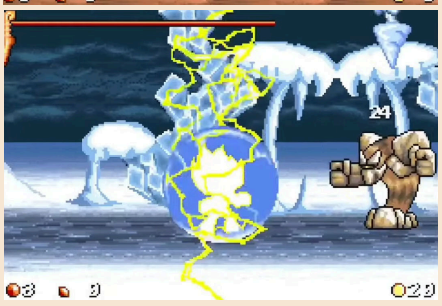
Year: 2024
Editor/Developer: Incube8
Genre: Platform/Action
Platform: Game Boy Advance
Web site: https://incube8games.com/en-eu/products/kien-gba

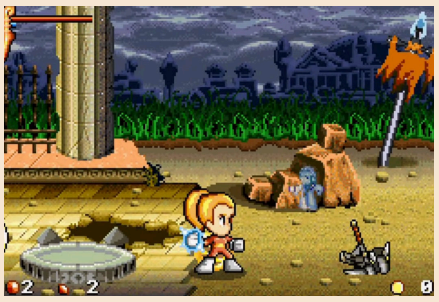
The most delayed game ever has finally arrived on our GBA.
I'm talking about Kien, an action game that could be described as a "Pre-Souls" game, which was supposed to be released in the early 2000s on the Game Boy Advance.

Unfortunately, due to circumstances beyond our control and significant delays, it has been postponed until now.
But what is it?
It's an action platformer with several elements borrowed from role-playing games, which takes the player through 23 rather challenging and complicated fantasy levels.
The gameplay is reminiscent of Ghouls 'n Ghosts in its relentless pace and the sheer number of enemies ready to kill the protagonists (at the beginning of the adventure, we can decide whether to play as the warrior or the sorceress).
The non-linear feature allows the player to explore the vast world in their preferred way and to upgrade their character to the fullest.
Available in two editions: the first is digital and contains the game ROM and manual in PDF format, while the second is in cartridge format, complete with box, booklet, and sheet full of stickers. Let's just say that the cost of the cartridge is quite high.
But after all these years of waiting, can we consider ourselves satisfied? I would say not completely.
Kien is a nice game, but it has some problems.
The graphics are certainly bright but childish, with our heroes and monsters looking like they've been taken from a Cartoon Network series. The static images that tell the story are lacking in detail and seem out of context.
The animations are cute but few and far between and repetitive.
The sound is terrible, bordering on unbearable, and I recommend turning it down as soon as possible.
The GNG gameplay is interesting but becomes tedious due to the repetitiveness of the levels and enemies.
The PASSWORD save system is also inconvenient, requiring you to enter a password at the end of each level to access it. But a save slot? In 2024, it could have been done.
In short, we expected something more... much more.

by **Marta Rossmann**







OUR FINAL SCORE

» Gameplay 40%
Repetitive levels, monsters, and gameplay. Enemy respawning is poorly managed and definitely exaggerated.
Terrible sound.

» Longevity 50%
Not impossible to finish, but definitely boring.

3D POOL

Year: 1989

Editor/Developer: Firebird/ Microprose Software

Genre: Sport

Platform: Commodore 64

Summer has arrived and, slowly but surely, so have the holidays. For many of us who lived through the 80s and 90s, going to tourist resorts meant spending whole days in arcades among dozens of arcade machines, playing with tokens and watching others play, perhaps those who were better than us and finished the game with just one token.

In the numerous arcades we visited, in addition to arcade cabinets, there were also ping pong tables, foosball tables, and, especially for the older ones, pool tables. Not the classic ones we found in village bars, but American-style ones, a little different from the classic ones we were used to.

It wasn't long before pool simulations arrived on home computers, so even us kids could play and learn the rules right away. Firebird's 3D Pool is a good 3D simulator that faithfully reproduces the rules and style of American pool. The game begins with the balls arranged in a triangle and the cue ball ready to make the first shot, which will be the one we have to use to pot all our balls.

The first ball to be pocketed will determine which of the two types we will have to play if we are the first to pocket a ball. In play are the solid balls, i.e., the solid-colored ones, and the striped ones, all numbered plus a black one (number 8) that must be pocketed last. If we accidentally pocket it first, we will suffer immediate defeat.

Like all 3D games on 8-bit computers, we may notice a slowness that, in my opinion, I did not find as tedious as in others. The available options are designed to give the game a good longevity, from a single game, practice (very useful), to a tournament that starts in the quarterfinals and then faces Joe Maltese, a well-known billiards champion of the time, in the final.

The controls are simple. With just the fire button, you only have to set the power of the shot with a power bar located at the top of the screen and the effect of the ball by calibrating a cross on the white ball next to the power indicator.

If the cue had also been included in the game, it would have been more realistic, but it is still a good pool game that deserves to be experienced by enthusiasts and others. There is no sound except for the balls when they touch and go into the pocket. After all, concentration is essential in a game like this, and I have never particularly liked music in sports games during competitions, even if they were my favorite hits.

I hope you are getting ready for your holidays by taking the new issue of this magazine with you under your beach umbrella, along with some summer and holiday-themed games and, above all, lots of fond memories of those golden years, given that, alas, arcades have now completely disappeared, except for the two or three that remain. I wish all Italian and foreign readers a pleasant stay in the places they have chosen to relax and have fun... See you in the next issue with another rich season of retro!

by **Daniele Brahimi**

OUR FINAL SCORE

» Gameplay 85%
Not difficult to get the hang of. A little more difficult to win.

» Longevity 70%
Several options will make the game dustproof, but then again, pool is pool...

Sorry for the delay

Dear readers, first of all, we want to thank you for your patience and loyalty to our magazine.
As many of you know, creating high-quality content requires considerable effort, and during this period, the demands of daily life have had a significant impact on our ability to produce content. Managing personal and professional commitments, along with the unexpected challenges that arise every day, has taken more time and energy than anticipated, inevitably leading to a delay in the publication of this new issue of RetroMagazine World.

Despite the difficulties, our commitment to bringing you in-depth articles and compelling content remains unchanged. We have worked hard to ensure that every page of the magazine continues to reflect the passion and dedication that sets us apart, and we have taken this extra time to ensure that quality is not compromised.

At the same time, I want to reassure everyone who was beginning to worry about the future of RetroMagazine World that the magazine and everything surrounding it are alive and well. I assure you that we are doing everything possible to keep our commitment to continue offering you the content you love and eagerly await.
As usual, the invitation to participate in the life of the magazine with content, questions, curiosities, and anything else is open to everyone. We look forward to hearing from you!

Once again, we thank you for your understanding and support. Without you, none of this would be possible.
We are confident that you will find this issue of the magazine as interesting and stimulating as the previous ones, perhaps even more so, thanks to the large number of game reviews.
Thank you for your patience and continued support.

Francesco Fiorentini

RetroMagazine World
Year 4 - Issue 26 - WINTER ED. 2025

Editor in Chief
Francesco Fiorentini
Editing/Deputy Manager
David La Monaca
Publishing Manager
Marco Pistorio
Website/Social Media Managers
Giorgio Balestrieri/Carlo Nithaiah Del Mar Pirazzini

AI Disclaimer - No content included in this digital magazine (editorials, articles, reviews, any texts or images) has been produced or generated by AI tools