



RetroMagazine

WORLD

future days are back

Issue 53



- Discovering the Salora Fellow
- Waffle CopyPRO Universal
- Super Turtle Pilot - The story of an adventurous software recovery
- PiStudio: a new IDE for Amstrad CPC
- All non-standard graphic modes of the VIC-II
- AmiGame Jam 2025

With retrocomputing, you can't go wrong...

Retrocomputing, as you well know, has never been merely a nostalgic look back. It's a journey that brings us unexpected discoveries every day, as if time, instead of flowing in a straight line, were coiling back on itself, leaving a new treasure around every corner. And just when we think we've seen it all—that computer, that chip, that hard-to-find peripheral—a manual forgotten in an attic pops up, a cassette with unreleased software, a programming book that rewrites the rules of what we thought we knew. With retrocomputing, you can never go wrong because the archaeological dig never ends: there's always something to uncover, and the surprise is part of the fun.

But the real turning point of recent years is something else. While the outside world debates whether today's games are “better” or “worse” than those of thirty years ago, something wonderful is happening within our niche: dozens of new titles, developed by enthusiasts for vintage platforms, often surpass the average quality of the original productions from the '80s and '90s. This isn't heresy; it's a fact. The new games for Commodore 64, Amiga, ZX Spectrum, or Amstrad CPC draw on decades of experience, rediscovered techniques, and tricks that no one had even imagined back then. They have the authentic flavor of the era, but the maturity of those who have studied every single flaw in the hardware and turned it into a strength.

The third pillar, perhaps the most fascinating, is the hardware. No longer just repairing, but reinventing. A flood of creatives, engineers by passion, hobbyists with a knack for soldering are building a solid bridge between yesterday and today: accelerator cards, alternative floppy drives, Wi-Fi devices for machines that didn't even have a modem, modern cartridges for vintage consoles. It is a bridge that does not look back, but propels retrocomputing into the future, making these machines not only alive, but up-to-date, connected, capable of interacting with the contemporary world without losing an ounce of their soul.

The issue you hold in your hands is the perfect example of this journey. Inside, you'll find precisely this explosive mix: the story of a recent discovery—software once thought lost and now brought back to light—reviews of new games that would have put the developers of the golden age to shame, and the presentation of hardware that transforms your old computer into a modern machine. And, as always, by the end you'll be left wanting more, wondering what's in store for the coming months and years.

Because with retrocomputing, you can never go wrong. And the next surprise is probably just around the corner.

Francesco Fiorentini

TABLE OF CONTENTS	PAGE
♦ PiStudio: a new IDE on CPC	Pag. 3
♦ Salora Fellow	Pag. 6
♦ Sinclair QL: Sir Clive's quantum leap	Pag. 10
♦ NES Zapper	Pag. 14
♦ WaffleCopyPro-Universal	Pag. 16
♦ Discovering Kyvos by George Sokianos	Pag. 18
♦ Kaleidoscopes in many flavours (C64)	Pag. 22
♦ A magic trick (C64 beginners)	Pag. 30
♦ Super Turtle Pilot (story of a weird recovery)	Pag. 34
♦ All non-standard graphic modes of the VIC-II	Pag. 38
♦ Building game maps (C64)	Pag. 46
♦ Digital preservation and code enhancing	Pag. 52
♦ AmigaOS 3.3 - The evolutions of the OS	Pag. 58
♦ Interview with Richard Farrell aka Vyper68	Pag. 60
♦ To collect or to accumulate?	Pag. 66
♦ Apple turns 50: half century of "Think Different"	Pag. 68
♦ Nintendo's (almost) forgotten mascots	Pag. 72
♦ AmiGame Jam 2025 - Return to arcade	Pag. 74
♦ Andoria: the return of the GDR on Apple II	Pag. 77
♦ Crystian and The Lost Crystals (C64)	Pag. 79
♦ Ghosts'N'Goblins (Amiga AGA)	Pag. 82
♦ Mappy (Amiga AGA)	Pag. 84
♦ Akalabeth 1000 (Sega SG-1000)	Pag. 86
♦ Wee Ninja (Atari 5200)	Pag. 88
♦ Space Bash: Contact!! (MSX 16KB)	Pag. 90
♦ Celeste (Amiga)	Pag. 92
♦ Randoom (Commodore Plus/4)	Pag. 94
♦ Chrono Runner (MSX)	Pag. 96
♦ Black Castle II (Gameboy)	Pag. 98
♦ Echodrome (Amiga)	Pag. 100
♦ Dragon View (SNES)	Pag. 103
♦ Escape From PETSCII Planet (C64)	Pag. 106
♦ Junk Runner 64 (N64)	Pag. 108
♦ Soul Blaze (Dreamcast)	Pag. 110
♦ Zezito and The Lost Stars (Acorn Electron 32K)	Pag. 112
♦ Burgertom (Atari Jaguar)	Pag. 114
♦ Tony Do It! (C64, Atari 8-bit)	Pag. 116
♦ Wings Of Fear (Amiga AGA)	Pag. 118
♦ Bubble Buddies (C64)	Pag. 121
♦ Miracle Boy in Dragon Land (Atari ST)	Pag. 122
♦ Curse of The Pharaoh (C64)	Pag. 125
♦ Axis (C64 - Plus/4)	Pag. 126
♦ Gravy Duck (C64)	Pag. 128

This issue's contributors:

- Francesco Fiorentini
- Carlo Del Mar Pirazzini
- Marco Bergomi
- Eugenio Rapella
- Jari Jalonen
- Takahiro Yoshihoka
- Roberto Lanciotti
- Ingrid Poggiali
- Chiara Carrara
- Alex Fridman
- Daniele Brahimi
- David La Monaca
- Stefano Tognon
- Roberto Del Mar Pirazzini
- Tommaso Pinturru
- Giampaolo Moraschi

- Cover Image:
Giuseppe Mangini
- Cover Layout:
Carlo Del Mar Pirazzini





PiStudio: the new development environment for the CPC

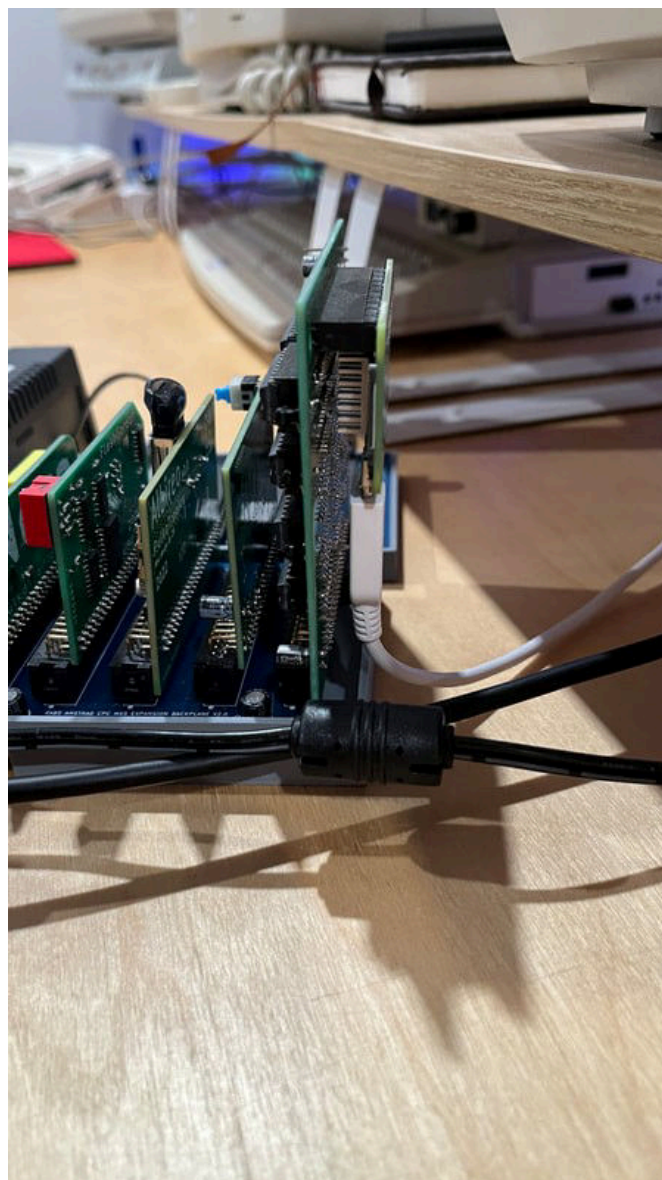
by Carlo Nithaiah Del Mar Pirazzini

PiStudio is a complete development environment that runs on Raspberry Pi and connects to an Amstrad CPC via Revaldinho's CPC-CPLink interface. From the user's perspective, however, "everything" happens on the CPC: keyboard, screen (Mode 2 overscan at 92×32 characters), input, and output are the original ones, while the Pi acts as a silent coprocessor handling compilation, debugging, advanced I/O, networking, and AI. The IDE is designed to work completely offline, with some optional features (chatbot, remote Git, networking) that take advantage of connectivity when available.

The architecture revolves around an integrated suite of tools that share the clipboard, interface, and even document



Example of a "hardware upgrade" with PiStudio



The board with the Raspberry

opening: you can switch from the editor to the debugger, the file manager, or the Git viewer without "leaving" the CPC context. The CPC-side client is a Z80 program (available as ROM or on disk) that sets the overscan, reads the keyboard, sends events to the Pi, receives only the "diffs" of modified characters to quickly update the screen, and manages breakpoints, RAM dumps, access to AMSDOS/UniDOS drives, and reconnection in case of link issues.

What's inside PiStudio?

The heart of PiStudio is a full-featured code editor, designed for development but suitable for any text. Among the main features are:





Multiple open documents with tab-style navigation
Horizontal/vertical split-view for comparison or dual editing.
Find & Replace in the current file or across the entire project, fuzzy file finder, bookmarks.

Compilation via shortcut: the binary is streamed directly into the CPC's RAM and executed.

Support for Locomotive BASIC, RASM assembler, and C via SDCC, with Code Inspector for functions/variables in ASM, BASIC, and C.

Unlimited undo/redo, clipboard history, calculator with hex/decimal conversion, ASCII/CPC glyph table.

Multilingual spell checking, ASM auto-formatting, configurable word wrap and tabs, print support, full session persistence (open files, cursors, splits).

Additionally, there is an AI-powered translation tool integrated directly into the editor, useful for comments, documentation, and text within projects.

Real-Hardware Debugger

The debugger allows real-time inspection and stepping of programs running on the actual CPC. When the code encounters a BRK, the CPC client captures the complete state of the Z80, part of the Gate Array state (ROM, RAM

banking), and a dump of the 64 KB of RAM, sending everything to PiStudio.

Hex editor and file manager

The hex editor is designed to work with binary files, disk images, or memory dumps, featuring a side-by-side Hex+ASCII view, insert/overwrite, selections with length and live checksum, cut/copy/paste, import/export blocks, text and hex search/replace, goto offset, split-view, and undo/redo, with printing. The file manager is a dual-pane browser that unifies the Pi's storage (PI:), the CPC's physical drives A:/B:, and any remote PiStudio instances on the LAN (via hostname). It offers full UniDOS support (long filenames, subdirectories, modern storage), folder creation, renaming, copy/move/delete, pattern matching, hidden file toggle, configurable columns and sorting by various criteria, bookmarks, fuzzy finder, and direct opening of files in text or hex editors. It also handles the creation and navigation of DSK images and ZIP archives, as well as reading AMSDOS headers, plus an internal shell.

Git, chatbot, and offline encyclopedia

PiStudio includes a full-featured Git frontend, designed

The screenshot displays the PiStudio debugger interface. The top menu bar includes File, Edit, Search, View, Tools, and Help. The main window is split into two panes. The left pane shows the disassembly of a program, with the current instruction at address 1029F: being a CALL instruction. The right pane shows the Z80 register state and a memory dump. The Z80 register state includes PC (1029), SP (8661), and various general-purpose registers. The memory dump shows the contents of memory starting from address FFD8.

```

File Edit Search View Tools Help
Disassembly (&1029)
0FFD: NOP
0FFE: NOP
0FFF: NOP
1000: JP MAIN
1003: PRINT_CHAR00000 DBG Printing char...
1029F: CALL TX1 OUTPUT
102C: RET
102D: _PRINT_STRING LD C,(HL)
102E: LD A,C
102F: OR A
1030: RET Z
1031: PUSH HL
1032: LD A,C
1033: CALL _PRINT_CHAR
1036: POP HL
1037: INC HL
1038: JP _PRINT_STRING
103B: _SET_CURSOR LD A,A
103C: CALL TXT_SET_CURSOR
103F: RET
1040: _SET_PEN CALL TXT_SET_PEN
1043: RET
1044: _MAIN LD A,&01
1046: CALL SCR_SET_MODE
1049: LD A,&01
104B: CALL _SET_PEN
104E: LD L,&05
1050: LD A,&0A

Z80
AF 480C ...P... I00 INT-OFF
BC 8148: 4B 6C 86 AF CD 32 BC KL...2.
DE B7FA: 00 00 00 00 81 61 0D .....a.
HL 1083: 48 65 6C 6C 6F 20 66 Hello f
IX BFFE: 6F DE 00 00 00 00 00 o.....
IY A6F8: FC A6 06 00 FC AB 07 .....
SP 8661: 18 C1 0B C2 47 C8 9E ....G..
PC 1029: CD 5A BB C9 4E 79 B7 .Z..Ny.
AF'8D8C: S..P..
BC'7F8D: 00 00 00 00 00 00 00 .....
DE'1170: 00 00 00 00 00 00 00 .....
HL'BB77: 91 CF 7C 91 CF 86 92 ..l....

Gate Array
Bank: &C0 Lower: OFF Upper: OFF

Memory Dump (&0000)
FFD8: 00 00 00 00 00 00 00 00 .....
FFE0: 00 00 00 00 00 00 00 00 .....
FFE8: 00 00 00 00 00 00 00 00 .....
FFF0: 00 00 00 00 00 00 00 00 .....
FFF8: 00 00 00 00 00 00 00 00 .....
0000: 31 89 7F ED 49 C3 91 05 ....I..
0008: C3 8A B9 C3 84 B9 C5 C9 .....
0010: C3 1D BA C3 17 BA D5 C9 .....
0018: C3 C7 B9 C3 B9 B9 E9 00 .....
0020: C3 C6 BA C3 C1 B9 00 00 .....I.
0028: C3 35 BA 00 ED 49 D9 FB .S...I.

C:\Programs\HELLO.c:21 I=StepInto O=StepOver R=Ret G=Go L=Source

```

The Debugger at work





for keyboard-based workflows. This allows for creating/ cloning repositories, opening recent ones, viewing changes in the working tree, commit history, side-by-side and line-by-line diffs, staging/unstaging, committing, reverting, resetting to specific revisions, and remote push/pull. The code chatbot/assistant integrates OpenAI (ChatGPT) and Anthropic (Claude) providers, with conversation management (saveable history), model/API key configuration, and split-view. It can analyze code, improve algorithms, generate comments, suggest label renames, translate documents, and generate new files—all without leaving the IDE; it can also be invoked as a dialog tool from the editor. The Developer Guide is a massive offline encyclopedia (~3 MB of text) with articles navigable via hyperlinks and full-text search, covering just about everything. There's truly a wealth of material: introductions to BASIC/ASM/ C, CPC history and culture, software and language manuals, a complete Locomotive BASIC reference, C with SDCC, hardware and firmware, Z80 programming, algorithms, file formats, graphics, sound, game development, demomaking, tips & tricks, and source code collections.

In short, a gem of preservation! From the global settings, you can configure Wi-Fi, network printer, API key for the

chatbot, Git identity, color theme, and display preferences, adapting the IDE for both pure offline use and a more connected environment.

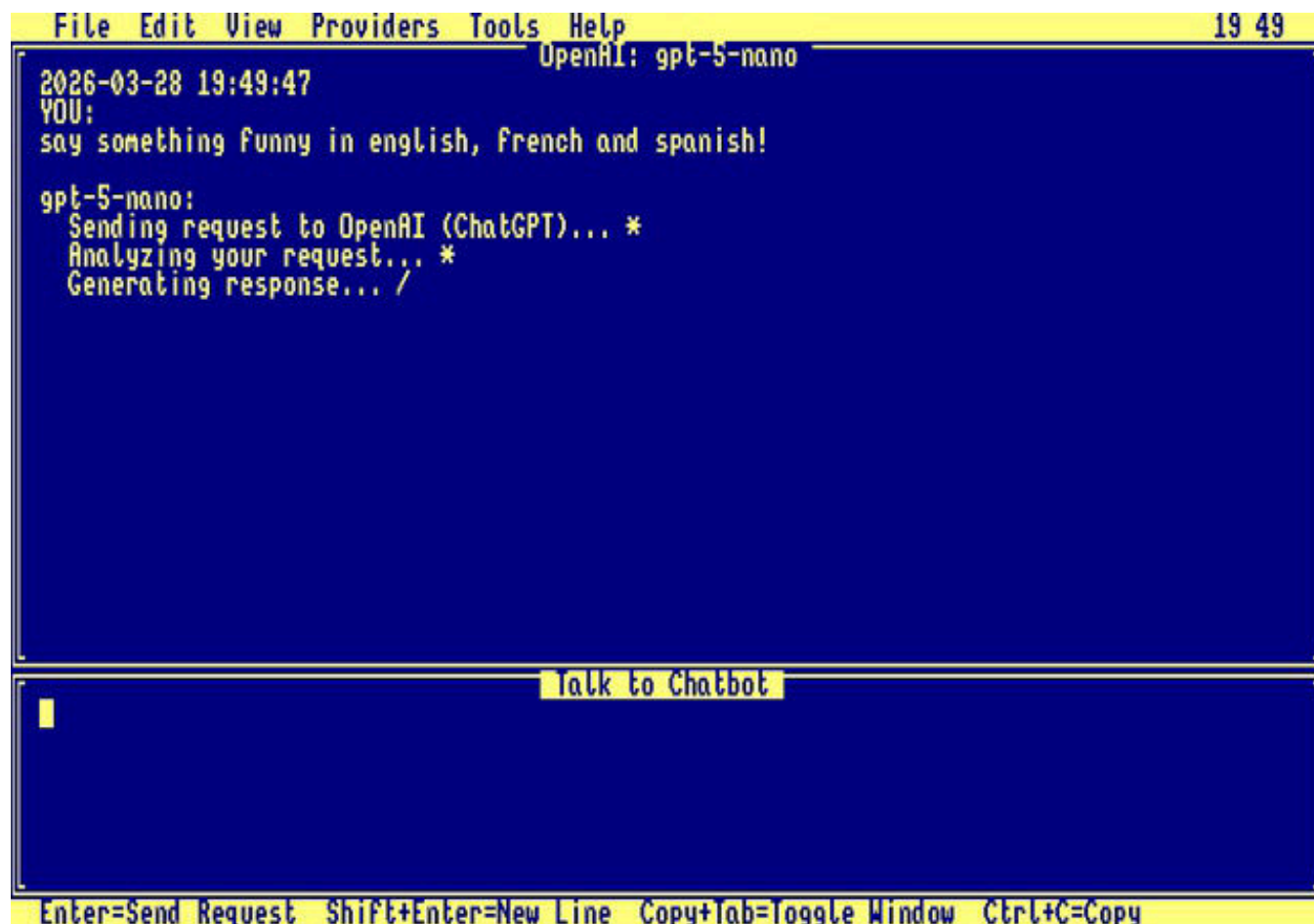
When will the project be released?

PiStudio has been announced but is not yet publicly available. The author (NoRecess) wants to first complete a real-world project of his own using the IDE, in order to refine the UX and stability. The release will be as a disk image for Raspberry Pi SD cards, to minimize installation complexity for the end user.

In summary, PiStudio takes a 1984 Amstrad CPC and, thanks to a Raspberry Pi, transforms it into a true development workstation with a modern editor, hardware debugging, file/Git management, AI, and extensive documentation—all while remaining “within” the original screen and keyboard.

Project website: <https://norecess464.weebly.com/news/pistudio-a-raspberry-pi-powered-ide-for-the-amstrad-cpc>

We thank the author of PiStudio for allowing us to use the pictures appearing in this article.



Chatbot





Salora Fellow

by Jari Jalonen

The Salora Context

There was a time when Salora was a major industrial player in Finland. For decades, the company manufactured various radios, then televisions, and a wide range of other electronic devices. Later, Nokia acquired Salora and began producing cell phones by leveraging Salora's technical expertise—but that's another story. Years before the cell phone era, Salora intended to launch a Salora-branded computer. The first of these computers was the Salora Fellow, based on VTech Laser 200 technology.

In practice, a simple "Salora Fellow" label was applied over the original brand name. The same applied to the software: a Salora Fellow sticker was affixed over the "VTech 200" text on the game packaging. Unfortunately, I have no information on why they chose the VTech 200 specifically, but that is how it was.

Technical Specifications

In Finland, the Salora Fellow, along with the Aquarius computer, was among the cheapest devices available, and its capabilities were minimal. It had 4 KB of RAM, of

which about 2 KB was usable for graphics. A 16 KB memory expansion was available, which was practically indispensable.

The device is equipped with a Zilog Z80A processor, a chip commonly found in many 8-bit computers of the 1980s, such as the Sinclair Spectrum or the Amstrad CPC.

Salora could have included those 16 KB of memory internally from the start, but that would likely have driven up the price. It was more strategic to sell the unit for 1,000 Finnish marks—which, adjusted for inflation, is equivalent to about 454 euros today. By comparison, a Commodore 64 at the time could easily cost 2,000 Finnish marks (about 900 euros today). Additionally, memory expansion had to be purchased separately if the user wanted to play most games. The display resolution is 32x16 characters in text mode with 8 colors, while graphics mode offers a resolution of 128x64 with 4 colors.

The audio is produced by a speaker inside the computer.

At first, I looked on the side of my device for RCA connectors for the audio output, expecting two or three ports. I eventually realized there is only one RCA port, and it is



Fig. 1 - The Salora Fellow





Fig. 2 - My Salora Fellow setup: the computer, the Salora Fellow Datassette, two joysticks, the 16k memory expansion, and that Y-cable (stereo-to-dual-mono cable). It's now part of my family. That blue cassette isn't part of the Salora Fellow; it's part of my MSX collection (SVI7x8). I think you all know what that means when you're a collector, right?

exclusively for the composite video signal (yellow). Since the audio comes from the internal speaker, it cannot be sent to the TV's speakers. For once, it's wonderfully simple.

A handful of games, a printer, a joystick, memory expansions, and a cassette recorder were released under the Salora Fellow brand. Additionally, all peripherals designed for the VTech 200 are fully compatible. I've seen videos where someone uses a Salora Fellow computer with a VTech 200 memory expansion.

My Salora Fellow

I was born in 1975, so I remember hearing about the Salora Fellow when I was a child. I recall seeing the device in a display window at a store called Musta Pörssi in Forssa. At the time, it was perceived as very primitive and "cheap" in a negative sense. I wasn't interested in it at all back then; the Commodore 64 was what captured my attention. My interest in the Salora Fellow only sparked in the 2020s, mainly because these devices have become quite rare.

I bought my unit around the turn of the millennium at a flea market in Forssa for 200 marks (about 52 euros

today). It came with a few games, a memory expansion, and a cassette recorder. I had almost completely forgotten about the device until 2023, when I became unemployed. I live in the countryside, in a place called Ypäjä, specifically in the village of Ypäjänkylä. I started exploring my attic and found several Commodore 64s, a couple of VIC-20s, a Spectravideo 328, and, to my great surprise, the Salora Fellow.

The device had withstood the Finnish climate—from winters at -30°C to summers at $+30^{\circ}\text{C}$ —in the attic for over 20 years. At first, it wouldn't start up. A VZ-200 was for sale on eBay, and I spotted a very familiar power supply there; so I had the idea to go back to the attic, and

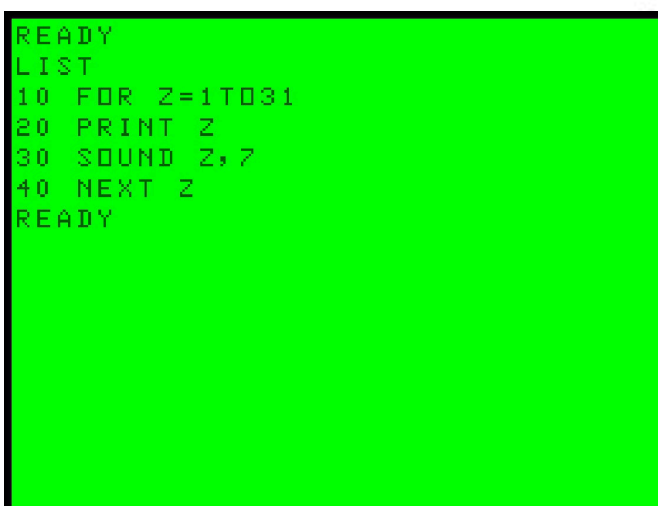


Fig. 3 - Example of sound generation in VZ Basic





after searching, I found a similar power supply. It was the right one: the light came on, and I had a working Salora Fellow! Eventually, I found a working cassette recorder online for 10 euros. I connected the recorder to the computer using a standard 3.5 mm TRS (minijack) cable.

The device itself works beautifully. Initially, the keys had some contact issues, since the machine hadn't been used for over 23 years. The keys are rubber, chiclet-style, similar to those on the Spectravideo 318 or the Sinclair ZX Spectrum. The feel is the same, but the technology seems superior: while the Spectrum is notorious for keyboard membrane failures, the Fellow's keys still work perfectly.

The audio is a very narrow, simple, mono sound coming from the internal speaker. There are 31 pitch levels, and programming follows the SOUND 15, 9 format, where 15 is the pitch and 9 is the duration. In terms of operation, the keyboard works like the Spectrum's; you can use key combinations (shortcuts) to speed up programming—for example, CTRL+P quickly produces the PRINT command.

I'm a programmer by nature. The local magazine Mikrobitti published program listings for the Fellow, and I had a strong desire to understand the unique VZ Basic language. One particular technical detail caused problems initially: the computer's port is a 3.5 mm stereo jack (TRS), but the cassette recorder has separate mono ports for EAR (In) and MIC (Out). One channel of the stereo cable carries load data, the other save data. Eventually, I found the original Y-cable, from stereo to dual mono, where the stereo end splits into two mono mini-jacks (red for Mic/Save and black for Ear/Load). Finally, I could save my code!

Experiments and Modern Solutions

I found a website (vz200.org) with many games for the VTech 200. I tried transferring the games from the PC to a physical cassette using the original cable, but it didn't work. The computer recognized the file name, but the games wouldn't start—probably due to signal interference or stereo crosstalk from modems or phones.

A second approach worked much better: I bypassed the cassette and connected the PC's headphone jack directly to the Salora Fellow's cassette port. I typed CLOAD on

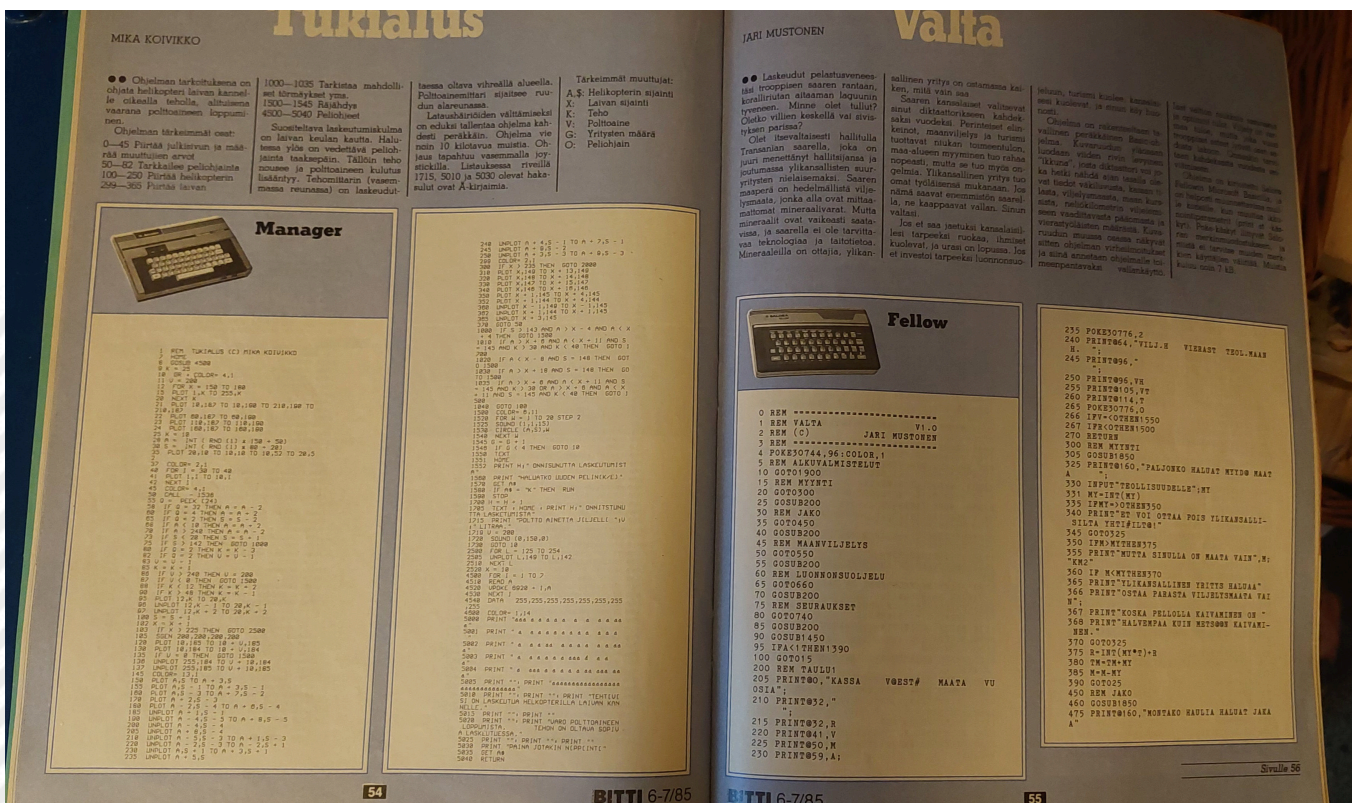


Fig. 4 - Example of listings in Mikrobitti magazine



Fig. 5 - Memory expansion to 16 KB

the Fellow and played a WAV file of the game Hamburger Sam from my PC. When the game's title appeared on my 14-inch Finlux portable TV at 3 a.m., I let out a loud cry of joy!

I hope you enjoyed my little adventure. I focused on the technical aspects and my personal experience. VZ Basic is a fascinating dialect to study, especially because of how its audio commands differ from GW-BASIC or SV-BASIC. It's a niche hobby, but for a retrocomputing enthusiast, it's a journey definitely worth taking.

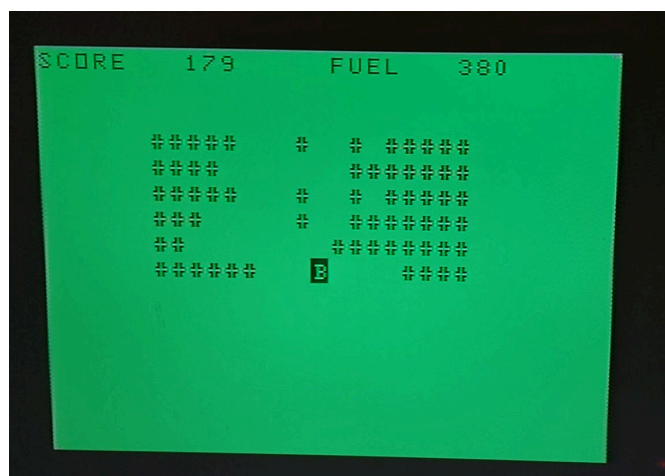


Fig. 6 - Example of a game

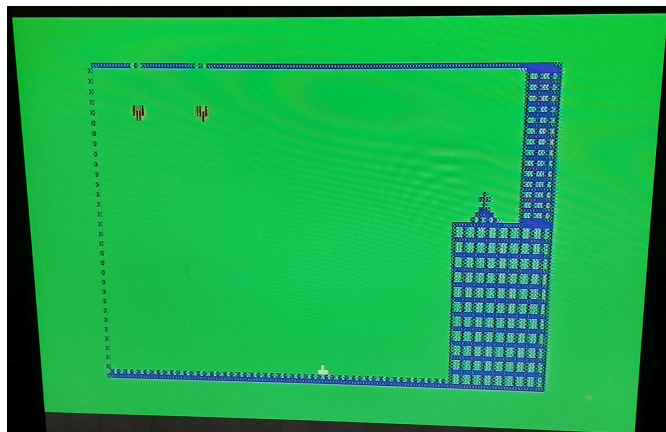


Fig. 7 - 'Ufoja' game: it could be called 'UFO'. There is a small white dot, and you have to shoot the UFOs falling from the sky. The code for 'Ufoja' comes from Mikrobitti magazine



Fig. 8 - Peli Loppu; we don't translate this into Italian - we just say "Game Over." :-)





Sinclair QL: Sir Clive's Quantum Leap

by Tommaso Pitturru

In 1984, while the ZX Spectrum was bravely battling the far more powerful Commodore 64 in a home computer market firmly anchored to 8-bit technology, Sir Clive Sinclair decided to drastically raise the stakes. The Sinclair QL (Quantum Leap) wasn't conceived as a simple successor—indeed, it abandoned all backward compatibility—but as a true “quantum leap”: a professional workstation offered at home computer prices, capable of bringing 32-bit power to every user's desk.

The QL was born under the guidance of a precise, almost stubborn, and even somewhat snobbish philosophical vision of Sir Clive, who considered the ZX Spectrum's success to have been achieved for the wrong reason: a system designed for computer literacy and basic research, equipped with a good processor—the Zilog Z80, on paper far more powerful than the MOS 6510 used by rival Commodore for the C64—and an excellent BASIC interpreter, which instead ended up in the hands of kids who used it as a versatile game console. Sir Clive felt a certain reluctance toward the video game industry, which

he considered intellectually inferior to the business, educational, and research sectors. The QL was therefore intended to be a tool that no one could consider an advanced toy: a professional and accessible computer, where entertainment was not contemplated except as a marginal byproduct.

Unfortunately, these ambitions clashed with design choices born of a philosophy focused on extreme cost-cutting. Characterized by an elegant and, in its own way, futuristic monoblock design by Rick Dickinson, the QL had a beating heart consisting of a 7.5 MHz Motorola 68008 CPU, a processor with 32-bit internal architecture but limited by an 8-bit data bus. This hybrid architecture, developed by chief designer David Karlin, combined with the use of two ULA chips (ZX8301 and ZX8302) designed in a rush and plagued by chronic overheating issues, nullified much of the system's theoretical power. Ironically, the main reason a “compromise” processor was chosen became obsolete shortly after the QL's rushed release, when the far more powerful Motorola 68000 underwent a drastic price cut,





eventually powering nearly all 16-bit home computers.

But back to the QL: even memory management was complex. With its 128 KB of base RAM, a significant portion of which was occupied by the video system, the multitasking capabilities of QDOS—the operating system written by Tony Tebbs—were theoretically possible but practically difficult to achieve, despite the excellent implementation of SuperBASIC by Jan Jones, considered by many to be one of the best BASIC interpreters ever created.

An aspect often underestimated, yet technically astonishing for the time, was the QL's native networking capability. Through two simple 3.5 mm jacks, the system implemented QLAN, a local network capable of connecting up to 64 computers. This innovation allowed for the sharing of peripherals and files across multiple workstations, anticipating modern office architectures by years. However, even in this case, the implementation suffered from the hardware's lack of robustness: the protocol was managed directly by the ZX8302 ULA using a "bit-banging" system that heavily loaded the CPU and made the signal extremely sensitive to electromagnetic interference, limiting the theoretical speed of 100 kbps to much more modest values in real-world scenarios.

The most controversial choice was the adoption of Microdrives as the standard storage medium. Conceptually fascinating and aesthetically futuristic, Microdrives were essentially rings of magnetic tape (derived from videotape) that rotated at high speed. Theoretically, the system could have achieved higher reliability standards through more refined control of the motor speed and the use of tapes with superior lubrication to reduce physical wear and stretching, the main cause of data loss. The adoption of more advanced firmware could also have implemented more robust error correction (ECC) systems and sector management that minimized latency during rotation.

Despite these potential improvements, the Microdrive could never have competed with the floppy disk: data access was sequential, whereas the floppy disk, thanks to its radial-access head, provided nearly instantaneous direct access. The inherent fragility of a tape running in contact with the heads made the medium susceptible to constant degradation, an unacceptable characteristic for a business machine.

Loki, the "SuperSpectrum"

In parallel with the QL's difficulties, the Loki project took shape in the Cambridge labs. Conceived as a





"SuperSpectrum," the original Loki was an extreme machine, designed by engineers of the caliber of John Mathieson and Martin Brennan to crush the competition with incredibly advanced hardware. Unlike the QL, the Loki was based on an enhanced version of the Z80 (the Z80H clocked at 7 or 8 MHz), but its true strength lay in its proprietary chipset:

The Loki Chip (Super ULA): This custom integrated circuit was designed to handle a resolution of 512 x 256 pixels with a palette of 256 simultaneous colors. The chip included memory management and graphics acceleration functions (a sort of early blitter) that would have enabled particularly smooth scrolling and animations, overcoming the "color clash" limitations of the original Spectrum. The design called for a full 128 KB of RAM dedicated exclusively to video, a configuration identical to that of the MSX2-based machines introduced shortly thereafter in Japan.

Wavetable Audio: The sound system would have been revolutionary for its time, featuring 8 8-bit audio channels managed via DMA, capable of reproducing samples and complex synthesis—features that would have placed the Loki on a higher level than the Amiga's Paula chip.

The idea was to maintain the compact, monoblock design, but with unprecedented graphics and sound processing power.

The Missed Evolution: The QL Plus and the Amstrad Legacy

Building on the work done for the Loki, a second, more pragmatic path could have been charted: a fusion of the QL's professional spirit and the Spectrum's playful legacy. This machine, a hypothetical QL Plus, would have retained the matte black monoblock chassis but would have corrected the shortcomings of 1984 through radical technical and software solutions:

Dual-CPU Architecture: The integration, alongside the 8 MHz Motorola 68000 with a 16-bit bus, of a dedicated Zilog Z80 coprocessor to ensure native hardware compatibility with the ZX Spectrum on one hand and the CP/M system on the other. The Z80 would operate in "bridge" mode, accessing a portion of the main RAM, making the QL Plus the definitive tool for the transition from home computing to professional personal computing.

The ICE Interface: The system software would finally abandon the austere command line to embrace the implementation of the ICE (Icons, Cursors, Events)





environment. This graphical framework, designed to give the QL a user experience similar to that of the Macintosh, would have transformed QDOS into a mature WIMP environment. Thanks to ICE, interaction with the powerful professional applications created by David Potter's Psion would become intuitive, using pointers and icons to manage the multitasking capabilities that were already in the QL's DNA.

Advanced Storage: The integration of a 3.5-inch drive directly into the monoblock chassis, on the right side of the keyboard, would have eliminated Microdrives in favor of a standard that was finally reliable.

The Amstrad Touch: Had Alan Sugar, following the April 1986 acquisition, decided not to stifle Sinclair's research but to adapt it to his business philosophy, we would likely have witnessed the birth of an "Amstrad Sinclair QL 16." A sturdy monoblock machine, with a front-mounted drive and standard ports, ready to compete on equal terms against the Atari ST.

The End of a Dream

As with other "What ifs" we've discussed, reality was far more prosaic. Amstrad's management chose the path of immediate profit and scrapped every experimental project. The original Loki ended up in a drawer, and the technology developed by its engineers took different paths: John Mathieson, Martin Brennan, and Ben Cheese—the masterminds behind the Loki project—founded Flare Technology, later contributing to the Flare One projects and the architecture of the Atari Jaguar.

For our part, we can only view this missed evolution as yet another lost opportunity. A machine that, with its elegant design and its ability to run both professional Psion software via the ICE interface and Spectrum classics, would have been the perfect third wheel between the Commodore Amiga and the Atari ST. What remains today is the wistful memory of a technological leap that Sinclair could have made, if only it had had the time to turn its ideas into reality.

Editor's Note: *The images accompanying this article were generated using an online AI service and are the result of the creativity and imagination of the article's author. This is an "uchronic" article, a piece of writing that describes historical events as if they had unfolded differently. It usually adopts the factual tone and formal structure of real-world journalism or academia to analyze an "alternate timeline".*





Accessories from the Retro World

NES Zapper

by Carlo Nithaiah Del Mar Pirazzini

In the past, consoles had various accessories to enhance the gaming experience. Racing games had special controllers designed to resemble a car's dashboard, such as the Arcade Racer Joystick for the Sega Saturn and the Hurricane Steering Wheel for the PlayStation 3; space shooters and aerial combat games had joysticks designed to feel more authentic, such as the NES Quick Shot and the Saturn Mission Stick. But for me, the most effective accessory is probably the light gun. It's one of the oldest toys, dating back to the earliest gaming systems and even earlier. On the NES, the most iconic accessory is undoubtedly the Zapper.



How It Works

The Zapper is essentially a single-button controller. When you pull the trigger, a signal is sent to the console to dim the screen for a brief moment, and all available targets on the screen turn into white squares. In that split second, a photodiode in the gun is activated, and if it detects white light, the system registers a hit.

When there are multiple targets, each white block is marked in sequence, and depending on the light detected, the system determines which target is hit. Some "legends" claim it's possible to trick the gun into registering a shot by aiming at a light bulb and pulling the trigger, but I've never known if that actually works. This is probably for the same reason the Zapper doesn't work on modern flat-screen TVs.



The Zapper has a design reminiscent of the old Flash Gordon serials. Simple and lightweight, yet surprisingly durable. There's only one button—the trigger—which makes a loud click when pressed. Without a start button, Zapper games require you to connect a standard controller as well (for selection and start). The original model was entirely gray, but Nintendo released the "orange" model in 1989.

Accuracy was the Zapper's strong suit. It required no calibration or the adjustments found in later, modern guns. From a distance of 2 meters, it was easy to hit targets with a reasonable degree of reliability.

The Essential Game

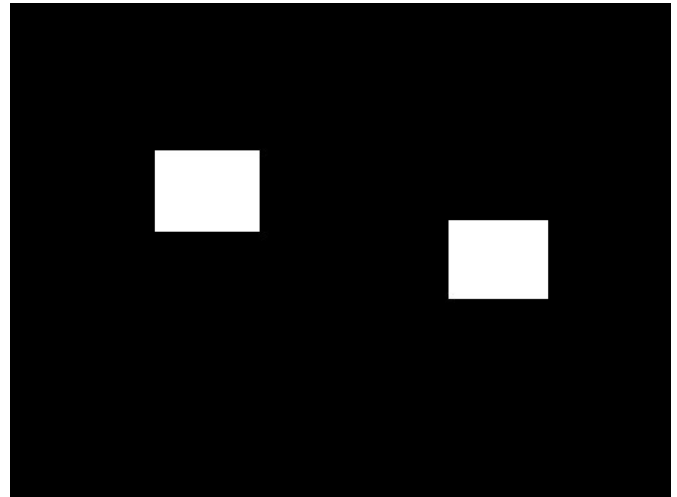
There were 17 games available for the Zapper. Some were particularly fun and original, others less appealing. But the undisputed king is undoubtedly Duck Hunt.





DUCK HUNT – Nintendo 1984

Released in 1984 in Japan, in 1985 in the U.S., and in 1987 worldwide, it was the launch title in the U.S. market, sold as a bundle with the gun. Inspired by a 1976 game of the same name featuring an optical gun and projector, Duck Hunt had three game modes. The first involved hunting a single duck, the second two ducks at a time, and the third was clay pigeon shooting. A wildly entertaining game that introduced the world to the player character's sidekick—the dog—who chuckles at us whenever we fail to hit our targets. We'll find him again as a playable character in the Super Smash Bros. series under the name "Duck Hunt Duo."





WaffleCopyPRO - Universal

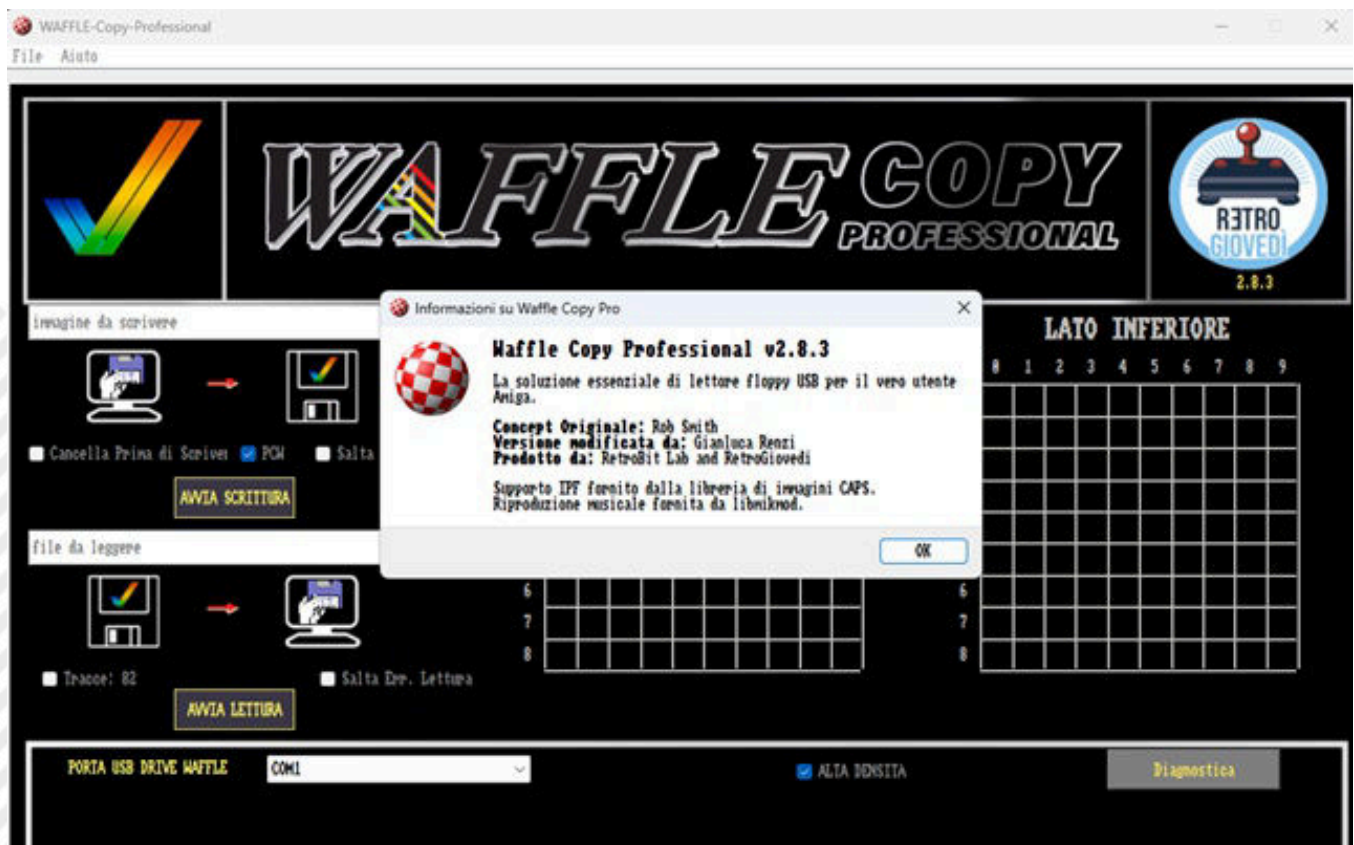
by Carlo Nithaiah Del Mar Pirazzini

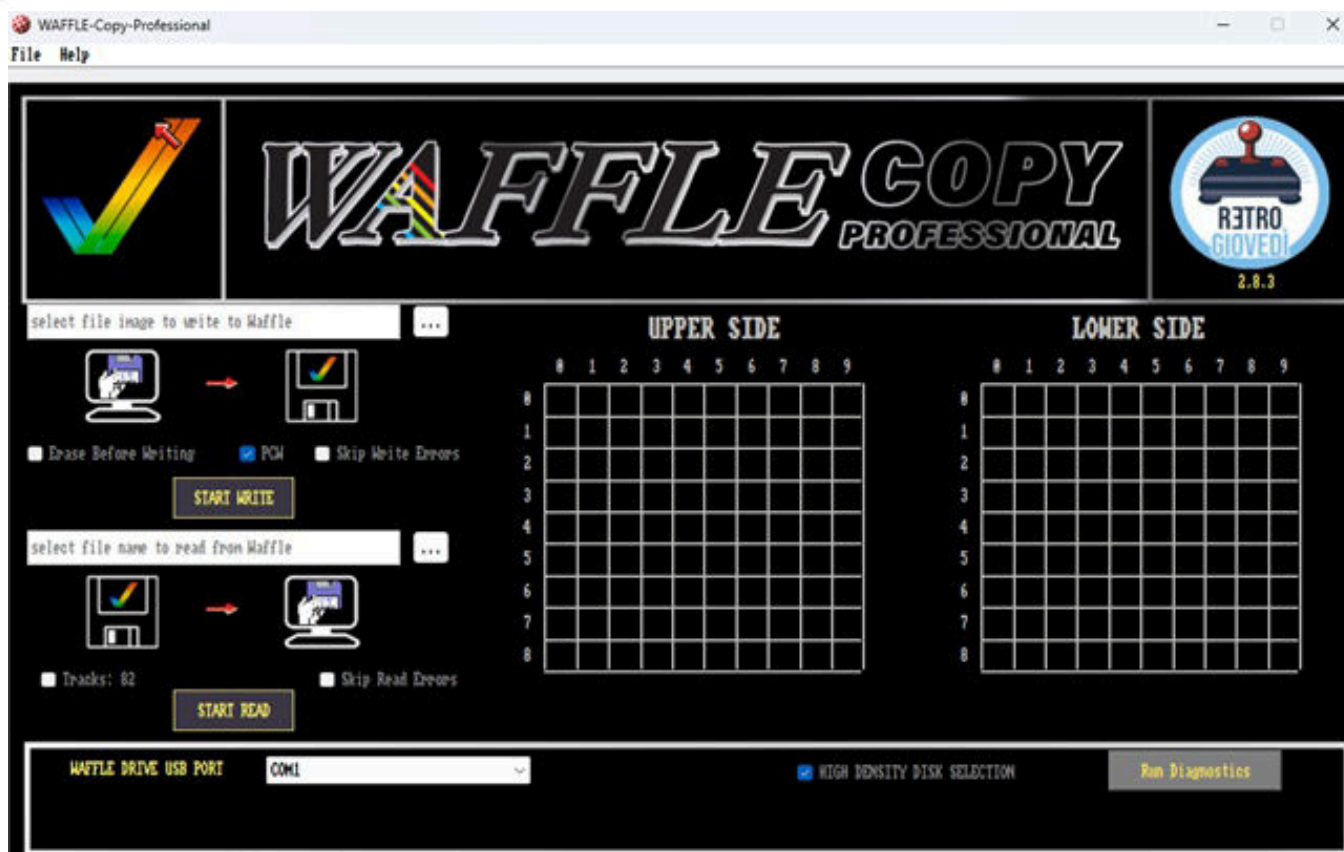
WaffleCopyPRO-Universal is the “definitive” front-end for those who want to use Waffle or DrawBridge as a universal Amiga (and similar) floppy drive on modern PCs, offering a successful balance between power and usability. It is a cross-platform application for Windows, macOS (both Intel and Apple Silicon), and Linux, designed to read and write floppy disks via interfaces such as the RetroGiovèdi Waffle or the Arduino DrawBridge, with a clear focus on Amiga formats like ADF and IPF. The target audience is the retro enthusiast who wants to preserve their collection of physical floppy disks and continue using them on both original hardware and in emulation, avoiding the need to navigate scattered tools, cumbersome CLIs, and DIY workflows.

Functionally, WaffleCopyPRO-Universal acts as a single hub for various controllers: it directly supports Waffle, DrawBridge, and other Arduino/FTDI-compatible devices, centralizing in a single interface tasks that would otherwise

require multiple different programs. It handles ADF images and statically integrates IPF support via the CAPS library, eliminating the need for plugins or esoteric configurations to work with the most serious preservation formats. The basic workflow is intentionally straightforward: select the serial port, insert the floppy disk, then initiate a read operation to create an image or a write operation to transfer an ADF/IPF to a physical disk. This is complemented by diagnostic tools and advanced options that help determine whether the issue is due to the disk, the drive, or the controller—a very useful feature when dealing with magnetic media that is often on the brink of failure.

The user interface has an unapologetically old-school look, and we love it! It uses the Amiga Topaz font and incorporates visual elements from the demoscene, such as sine-scrollers and graphic themes that nod to the Amiga world, but without sacrificing readability and clarity. The UI is designed to adapt to both small screens and





larger monitors, with particular attention to the visibility of status indicators during copy operations, where progress and error status feedback is crucial. Also noteworthy is support for sixteen languages, including Italian—an unusual choice for such a niche tool and very convenient if you plan to use it in educational contexts or with less technical users.

On the technical front, the project is based on Qt 5.15.2, and the build process is documented for Linux, macOS, and Windows. On Unix-like systems, you simply need to set up the dependencies (Qt, serialport, libmikmod, libftdi, and similar) and proceed with the standard ``qmake && make``, while on Windows it relies on Visual Studio 2019, Qt, and vcpkg for library management. The repository is active and up-to-date. A release 2.8.3 is mentioned, which introduces, among other things, explicit firmware version checking (with a block for revisions that are too old, below 1.8) and improvements in debugging and logging—a sign that this is not a half-finished experiment but a living project. The fact that it is open source also makes it interesting as a basis for study or extension, both for adding support for other controllers or formats, and as a reference for integrating a consistent retro aesthetic

into a technical tool.

Overall, WaffleCopyPRO-Universal presents itself as a mature, well-crafted, and surprisingly accessible tool for the type of work it performs. If you're considering a workflow for "reading and writing real Amiga disks from a modern PC," it's certainly one of the most solid and "presentable" candidates because it combines a serious commitment to preservation with an openly Amiga-centric style that speaks directly to the retro audience without alienating those seeking a functional and reliable application.

Useful links

Repository:

- <https://github.com/gianlucaenzi/ArduinoFloppyDiskReader>





Discovering Kyvos by George Sokianos

by Takahiro Yoshihoka, translation by Carlo Nithaiah Del Mar Pirazzini

Kyvos is a cross-platform graphical front-end for QEMU, designed specifically to create and manage AmigaOS 4 and MorphOS virtual machines in a guided, “turnkey” manner, without having to manually type command lines or write complex scripts.

Objective and General Architecture

Kyvos is a desktop application for Linux, macOS x64, and Windows that acts as an intermediary layer between the user and QEMU. The technical concept involves the user interacting with a tab-based GUI (hardware, settings, VM management), while Kyvos generates and executes the appropriate QEMU commands on their behalf, handling paths, parameters, peripherals, disks, and drivers. This approach greatly reduces the learning curve for those who want to use AmigaOS 4 or MorphOS in emulation, especially with modernized configurations (e.g., virtio GPU).

The application is distributed as a separate archive for each desktop platform, accompanied by a manual of over 114 pages that details installation, VM configuration, hardware options, and known issues. From a software lifecycle perspective, Kyvos is updated regularly (the latest version listed is 1.8.2 from March 2026), with a changelog organized by version, reflecting the care of a “mature project” rather than a simple amateur utility.

Dependency Management and Configuration Wizard

One of the technical strengths is the onboarding system: Kyvos includes a wizard that helps the user locate or automatically download essential dependencies, namely the QEMU binaries for the operating system in use and 7-Zip (7zip) for managing and extracting the archives required by the VMs

The wizard checks for the presence of these components, guides the user through the download process if they are

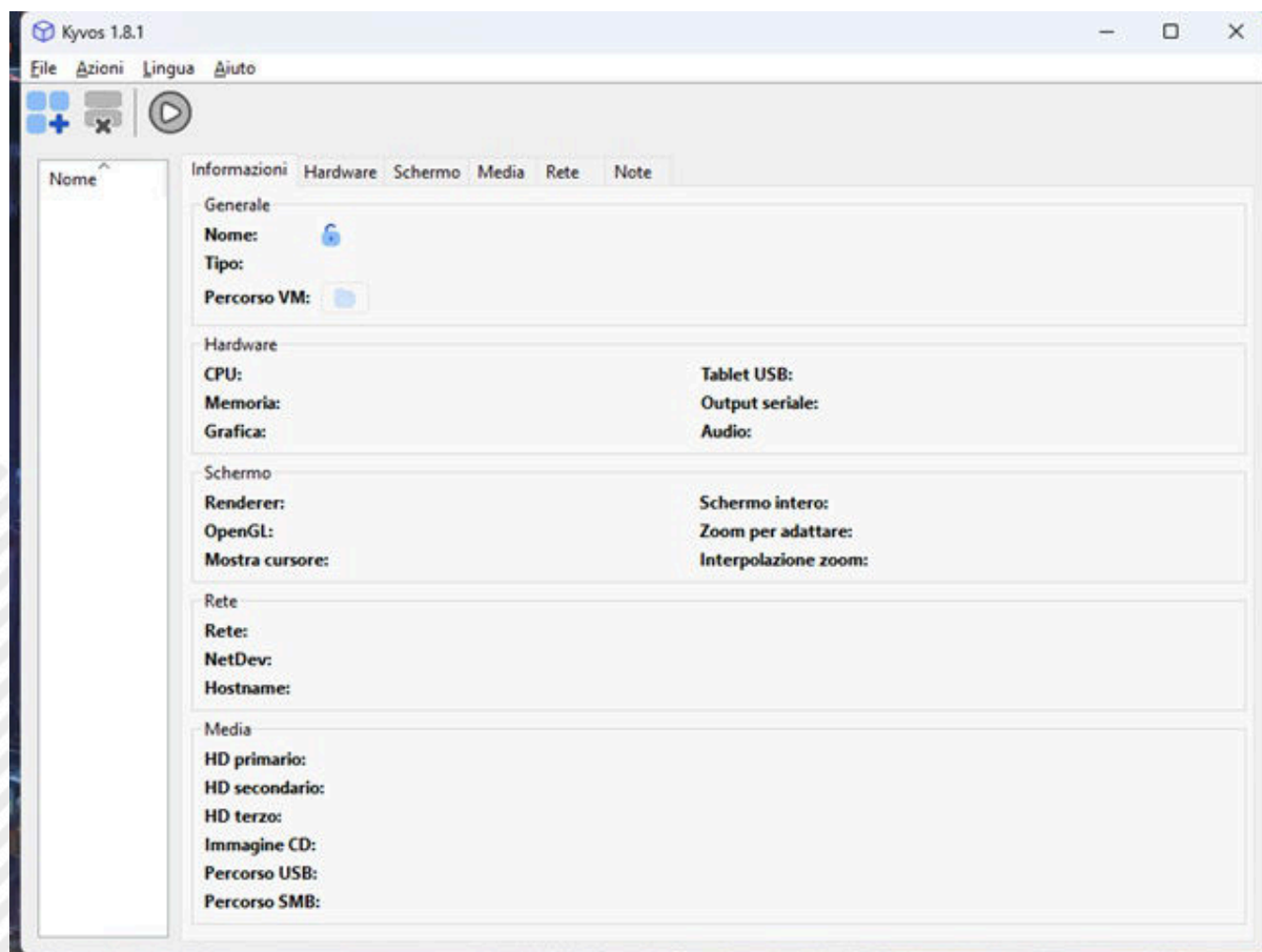


Fig. 1 - Right after installation (it's a simple .exe), it looks like this



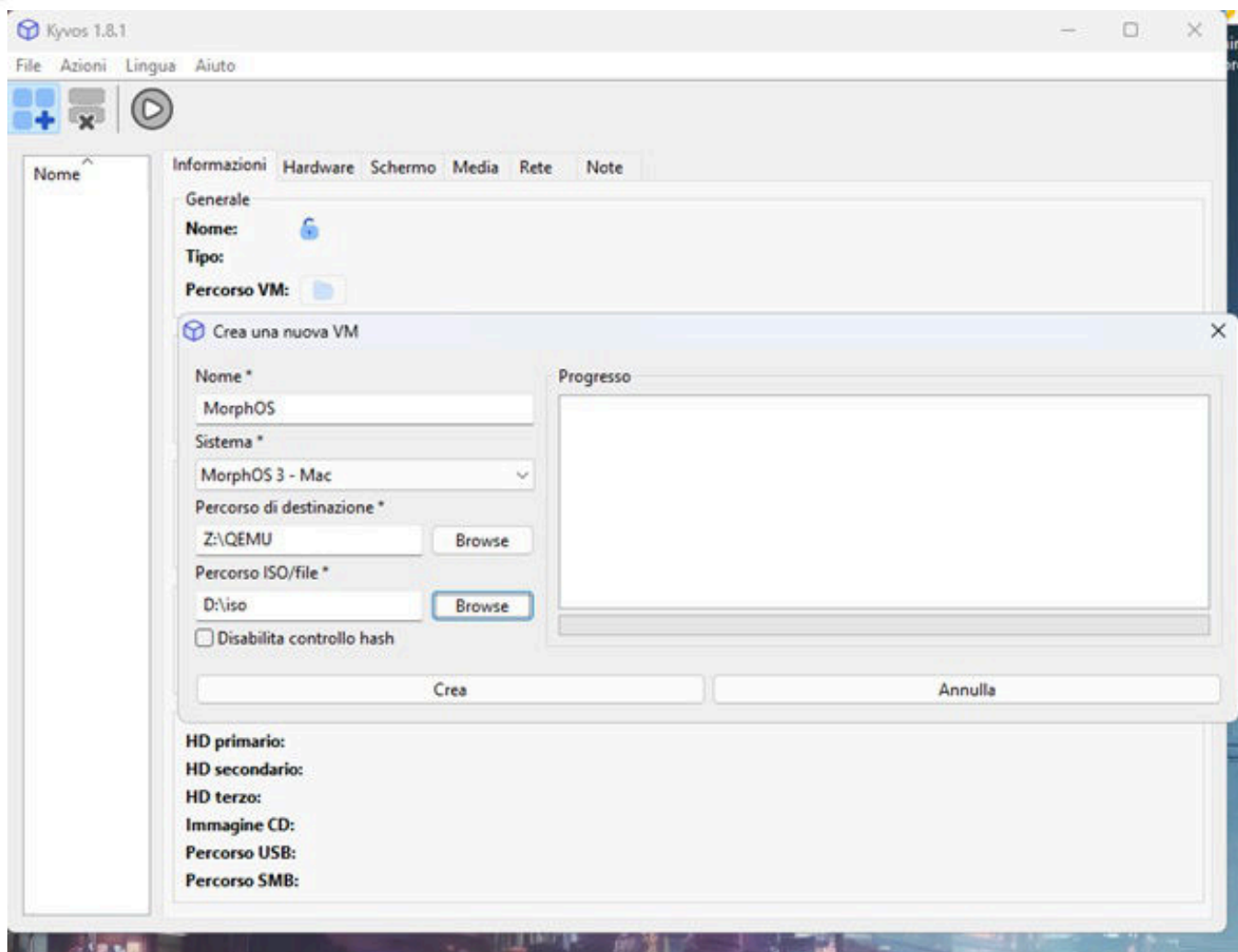


Fig. 2 - Creating a new “VM” is quite simple

missing, and sets the paths within the application. This spares the user from having to manually manage scattered QEMU installations, with the risk of incorrect parameters or conflicting versions.

Kyvos thus positions itself as an orchestrator: it does not provide QEMU or the AmigaOS 4 / MorphOS operating systems, but ensures that all external components (emulator, archiving tools) are linked and configured correctly, offering a seamless pipeline from installation to VM startup.

Virtual hardware modeling and advanced options

The GUI includes a dedicated Hardware tab, which abstracts QEMU settings into structured fields (CPU, RAM, disks, GPU, peripherals). The changelog highlights some interesting technical features:

- Management of multiple virtual disks, with specific fixes for conflicts between the third hard disk and the CD (indicating that Kyvos directly handles device/drive mapping in QEMU).

- “Debug level” field linked to serial output, useful for those who want to diagnose VM or guest system issues (e.g., detailed boot logs or kernel error traces).

- Monitoring of VM status upon Kyvos startup: if a VM is already running, the interface updates the icon in the list, so the user immediately has an overview of the machine fleet's status.

Robust file system error handling is also included: a check for errors when deleting VM files has been introduced, with an informational message box in case of failure. This is a “serious tool” feature that prevents ambiguous scenarios where the user thinks they have deleted a VM but the files remain on the disk.

Additionally, a tab dedicated to custom QEMU parameters has been added: advanced users can enter extra arguments, which are appended to the command line generated by Kyvos. This preserves QEMU's flexibility for power users while maintaining simplicity for those who don't want to touch anything advanced.



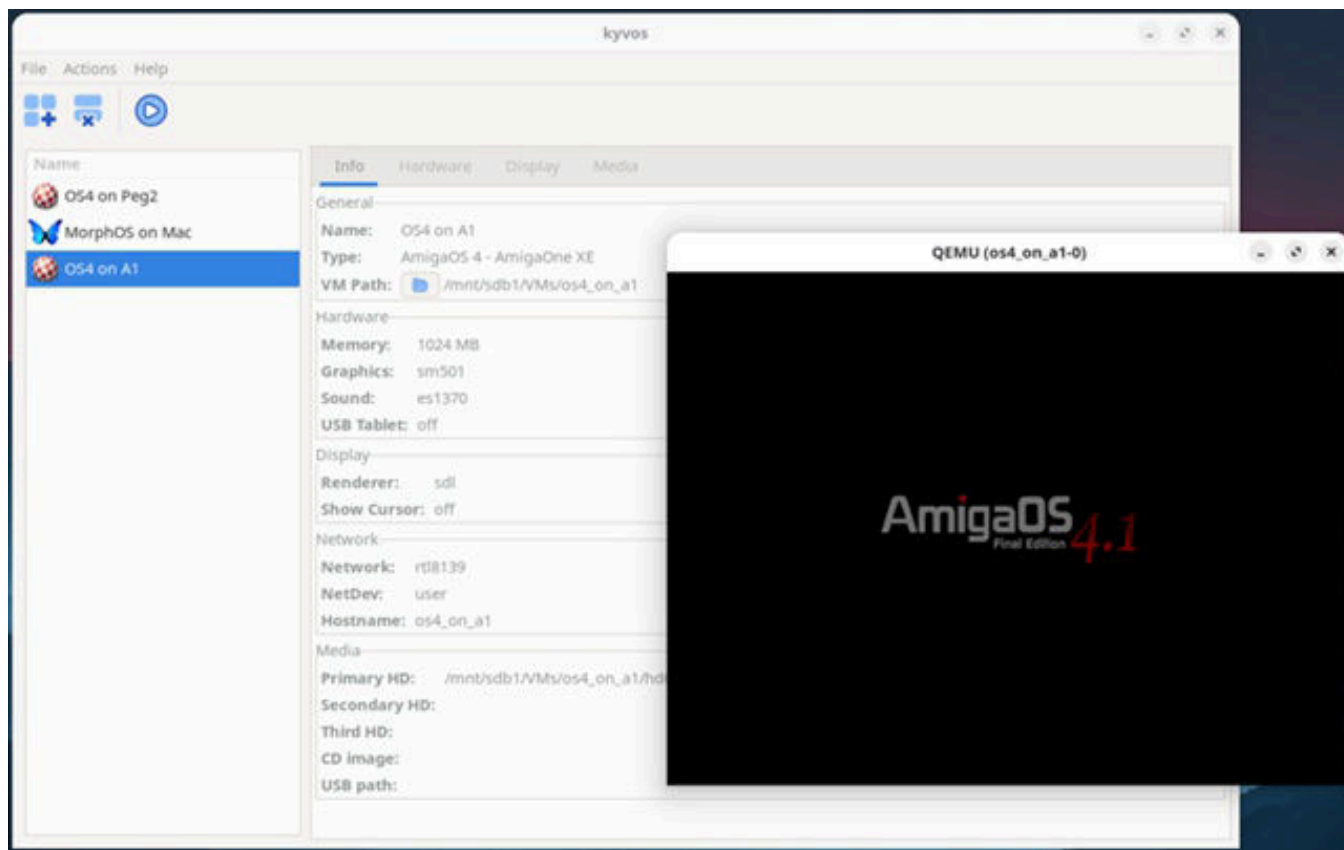


Fig. 3 - Installation of AmigaOS 4.1 FE on Kyvos

Integration with virtio GPU and AmigaOS 4

Kyvos supports the configuration of AmigaOS 4 environments with the virtio GPU driver, one of the most advanced approaches in modern AmigaOS emulation, but with an important limitation: the driver is still in private beta, distributed by A Eon, and is not included in Kyvos. Technically, this means that Kyvos provides:

- A QEMU configuration framework suitable for using virtio GPU
- Guided support for those who already have access to the drivers
- Documentation and FAQs on known incompatibilities and QEMU peculiarities in this scenario

In practice, Kyvos acts as a bridge between the user who has the correct files (drivers, ISOs, ROMs, etc.) and a non-trivial QEMU configuration that would otherwise need to be written by hand, such as those with paravirtualized GPUs and fine-tuned performance settings.

Manual, UX, and Project Maintenance

The 100+ page manual, complete with screenshots and step-by-step guides, is not a simple “readme,” but a comprehensive user guide covering: Installation on Linux, macOS x64, and Windows; creation of new AmigaOS 4 /

MorphOS VMs; known QEMU incompatibilities and their workarounds; and the use of the various interface tabs, including advanced options



Fig. 4 - The massive 126-page manual in PDF





The changelog shows continuous attention to actual errors encountered by users: bug fixes related to uninitialized fields ("Table field 'qemu386' was not initialized!" in the welcome window), missing objects (vmRunAutoInstall), and unwanted resets of custom paths during setup. This indicates an active feedback and maintenance cycle, with cross-testing across systems and configurations.

From a user experience perspective, "Check for updates..." and "Roadmap" items have been added to the Help menu, a sign that the project is designed to coexist with the user over time, not as a "disposable" tool.

Overall Assessment

From a technical standpoint, Kyvos is a specialized GUI front end, with an internal configuration model that translates user settings into robust QEMU commands. It is cross-platform and actively maintained, with clear versioning and a changelog focused on real bugs and useful features (serial debugging, custom arguments, control of running VMs). It is well-documented, with a

manual and FAQ covering the entire lifecycle of AmigaOS 4 and MorphOS VMs. It is integrated (on the configuration side) with advanced scenarios such as the use of virtio GPU, without distributing proprietary components.

For those who want to emulate AmigaOS 4 or MorphOS without becoming a QEMU command-line expert, Kyvos offers a structured and professional environment that drastically lowers the entry barrier while, at the same time, not limiting tuning options for more experienced users.

Links and references

Project GitHub:

- <https://github.com/walkero-gr/kyvos>

Developer's Ko-Fi page:

- <https://ko-fi.com/s/6476fdadd2>

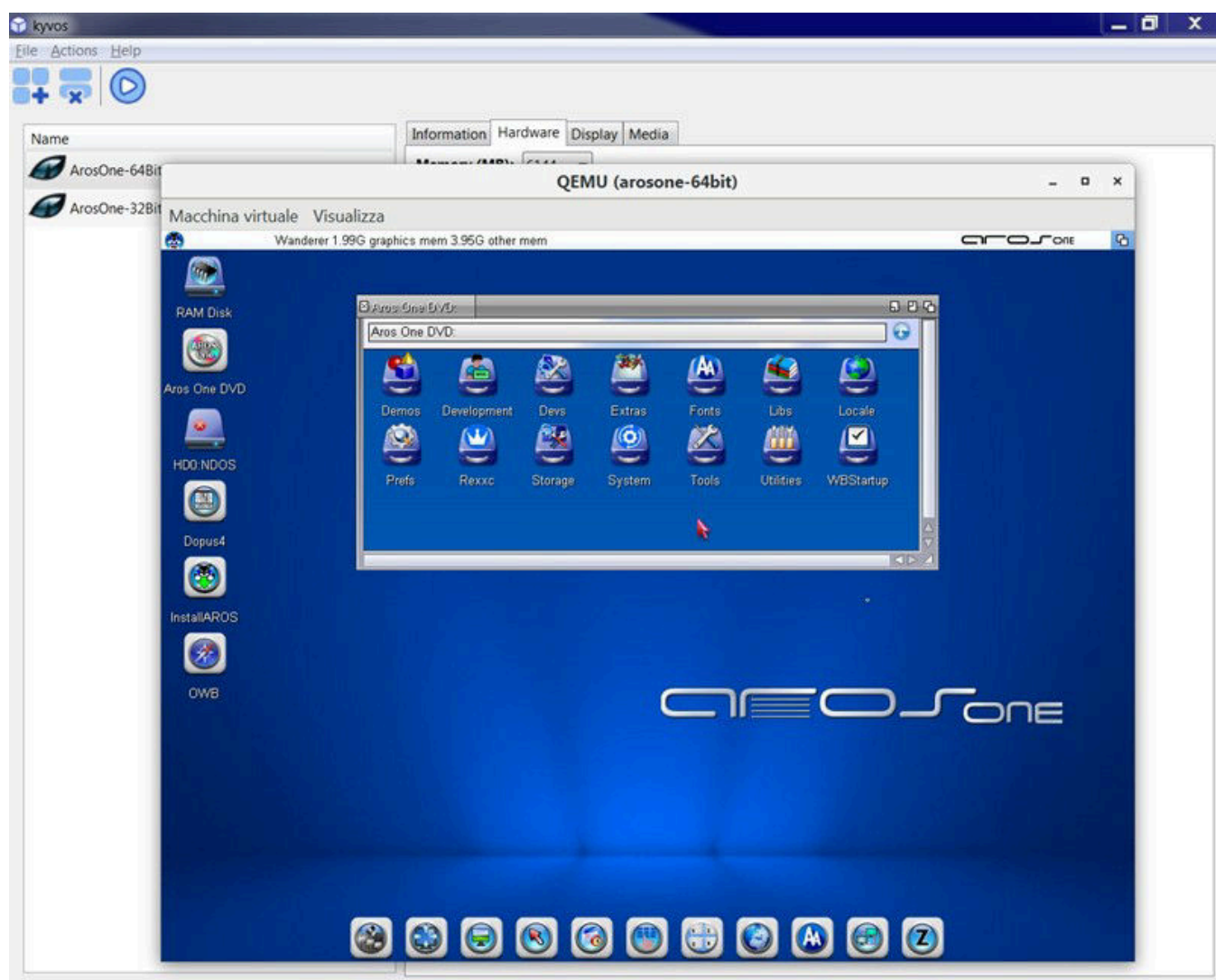


Fig. 5 - Aros emulated perfectly





Kaleidoscopes in many flavours (for C64)

by Eugenio Rapella

It is well known that the C64 screen is “mapped” to memory locations 1024 through 2023 for characters, and 55296 through 56295 for colors. This means that the POKE 1024,18 command will display the letter R in the top-left corner of the screen (the codes to use are listed in APPENDIX E of the “User Manual”), and the POKE 55296,1 command will color it white (APPENDIX G).

The following BASIC program (CALEIDO1.BAS) generates a “kaleidoscope effect” on the screen of our Commodore:

```
100 print chr$(147):cl=55296
110 poke 53280,0:poke 53281,0
120 for x=0 to 19:for y=0 to 19
130 poke 1024+x+40*y,81
140 poke cl+x+40*y,0
150 next:next
160 w=rnd(-ti)
170 x=int(rnd(1)*10)
180 y=int(rnd(1)*10)
190 c=int(rnd(1)*256)
200 poke cl+x+40*y,c
210 poke cl+y+40*x,c
220 poke cl+19-x+40*y,c
230 poke cl+y+40*(19-x),c
240 poke cl+x+40*(19-y),c
250 poke cl+19-y+40*x,c
260 poke cl+19-x+40*(19-y),c
270 poke cl+19-y+40*(19-x),c
280 goto 170
```

Instructions 100-110: Clear screen, black background and border.

Instructions 120-150: As a preliminary step, the upper-left portion is filled with a 20×20 “carpet” of “solid dots” (130 POKE ...,81), which are immediately colored black (instruction 140) so they blend into the background (but will remain there the entire time).

At instruction 160, the random number generator is initialized, after which... we begin: instructions 170

through 280 form an infinite loop that will animate our kaleidoscope.

At lines 170, 180, and 190, three random values are generated: X and Y are integers between 0 and 9, and C is an integer between 0 and 255.

At line 200, the POKE CL+X+40*Y,C command assigns color code C to the screen cell located at column X+1 and row Y+1. Since the “solid dots” are already present, the effect is to color the dot (initially black, and therefore invisible) with color C.

Subsequently, the same color is assigned to seven other dots located symmetrically to the first one with respect to the main diagonal (swap X with Y), with respect to the vertical axis separating the first 10 characters from the second 10 (keep the Y row and replace X with 19-X in POKE CL+19-X+40*Y,C), relative to the horizontal axis, etc. (see FIG. 1).

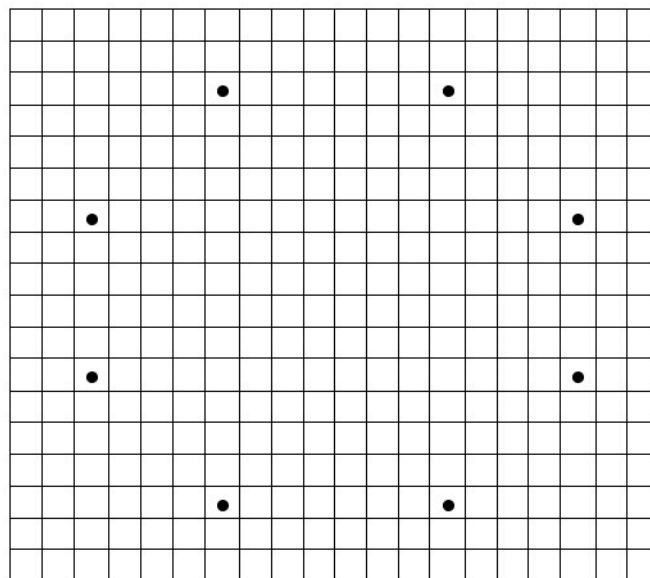


Fig. 1

280 refers back to 170, and the process starts over: another eight dots are colored with a new shade, then another eight, and so on.

This creates an “animated kaleidoscope,” a couple of frames of which you can see in Figures 2 and 3.



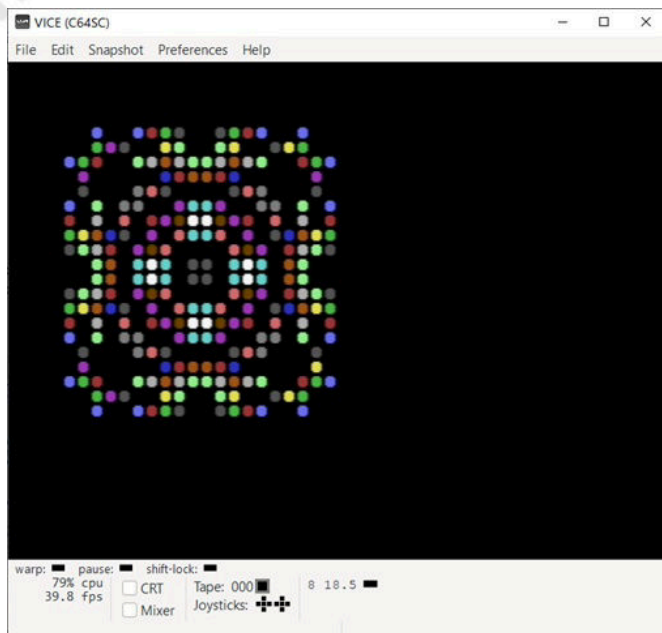


Fig. 2

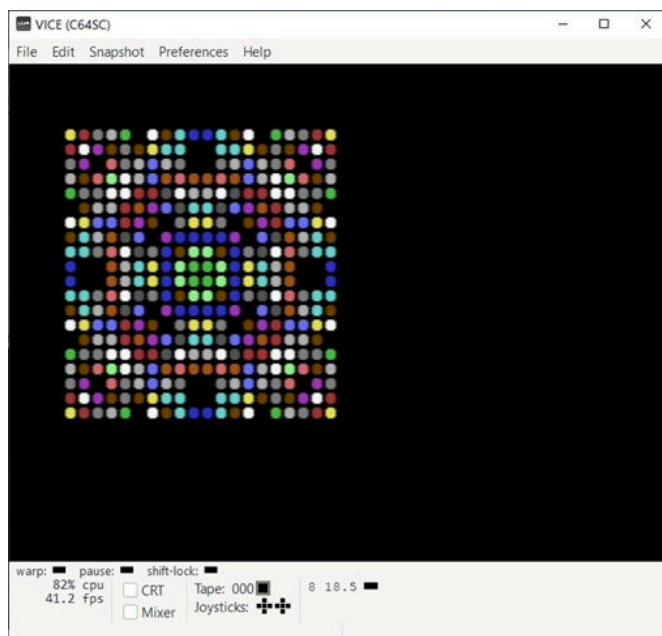


Fig. 3

In my opinion, the visual effect is pleasant even as it is, but we can try to see what happens if we manage to speed up the execution.

Inside the main LOOP (170-280), additions, subtractions, and multiplications are performed, which our beloved Commodore, alas, executes rather slowly. One possible improvement in terms of speed is to perform some of these calculations in advance outside the 170-280 LOOP and store the values in some vectors.

The CALEIDO2.BAS program is a variation of CALEIDO1.BAS:

```
10 dim dma(9), qa(9), qdm(9)
```

```
20 for a=0 to 9: dma(a)=19-a
30 qa(a)=40*a: qdm(a)=40*(19-a)
40 next
100 print chr$(147): cl=55296
110 poke 53280,0: poke 53281,0
120 for x=0 to 19: for y=0 to 19
130 poke 1024+x+40*y,81
140 poke cl+x+40*y,0
150 next: next
160 w=rnd(-ti)
170 x=int(rnd(1)*10)
180 y=int(rnd(1)*10)
190 c=int(rnd(1)*256)
200 poke cl+x+qa(y),c
210 poke cl+y+qa(x),c
220 poke cl+dma(x)+qa(y),c
230 poke cl+y+qdm(x),c
240 poke cl+x+qdm(y),c
250 poke cl+dma(y)+qa(x),c
260 poke cl+dma(x)+qdm(y),c
270 poke cl+dma(y)+qdm(x),c
280 goto 170
```

At instruction 10, three 10-element vectors (indexed from 0 to 9) are introduced: DMA(..), QA(..), and QDM(..). In the loop from 20 to 40, these vectors are loaded with the values 19-A, 40*A, and 40*(19-A) for A values from 0 to 9.

CALEIDO2 then matches CALEIDO1 except in instructions 200 through 270: instead of performing the calculation, the value to be associated with the POKE is taken from the vector. Thus, for example, instruction 270 of CALEIDO1:

POKE CL+19-Y+40*(19-X),C

becomes: POKE CL+DMA(Y)+QDM(X),C.

The time spent performing calculations is traded for the time required to access the vector elements. The gain is modest, but noticeable: when running CALEIDO2, the visual effect is slightly faster. I've conducted numerous experiments with small support programs (complete with a stopwatch), and the results were consistent: the time savings amount to 15% (I always use the VICE emulator without "warp mode").

If we want a further, significant increase in execution





speed, we could use a compiler (such as Blitz Compiler) or... switch to Assembly.

When it comes to 6502 Assembly, my skills are very limited. In this section, I'm therefore addressing a beginner like myself: let's try together to write some code using the most basic instructions.

The part we want to speed up is the one inside the main LOOP, so we'll keep the initialization section—which runs only once—in Basic.

Here is our “work plan” using CBM Prg Studio:

1. Create a Basic file (CAL1.BAS) with the initial section.
2. Create the file CAL2.ASM with the Assembly code for the main LOOP.
3. Using the “Tools > Generate BASIC loader” option, we'll get a Basic code where the instructions from CAL2.ASM are converted into a series of DATA automatically loaded starting from the location chosen for launching the Assembly program. This “loader” also contains the SYS instruction that will execute CAL2.ASM.
4. Now simply append this BASIC code to CAL1.BAS. We will then have a BASIC program that executes both the initial section and the loop written in Assembly: CAL3.BAS.

Here is the CAL1.BAS file:

```
100 print chr$(147):sc=55296
110 poke 53280,0:poke 53281,0
120 for x=0 to 19:for y=0 to 19
130 poke 1024+x+40*y,81
140 poke sc+x+40*y,0
150 next:next
160 for kk=0 to 9
170 v1=19-kk:v2=40*kk:v3=40*v1
180 poke 838+kk,v1
190 h=int(v2/256):l=v2-256*h
200 poke 848+kk,l:poke 858+kk,h
210 h=int(v3/256):l=v3-256*h
220 poke 868+kk,l:poke 878+kk,h
230 next kk
```

Instructions 100–150 are the same as those for CALEIDO1 and CALEIDO2 (background and border coloring, a “pattern” of black dots); Cycle 160–230 is responsible, for KK ranging from 0 to 9, for inserting the values 19–

KK into the 10 locations starting from number 838 (we will therefore have 19 in 838, 18 in 839, 17 in 840 ... 10 in 847).

The values $V2=40*KK$ and $V3=40*(19-KK)$ calculated at 170 require TWO bytes; for example, for $KK=9$, V2 and V3 take the values 360 and 400 respectively, both of which exceed 255. Instruction 190 is responsible for separating the “high byte” (H) and the “low byte” (L) of V2. For example, when $V2=360$, we will have $H=INT(360/256)=INT(1.40625)=1$ and $L=360-256*1=104$ (note that the decimal value 104, in hexadecimal notation, becomes \$68, and, overall, the value 360, in hexadecimal notation, becomes \$0168).

The “low byte” of $40*KK$ for KK from 0 to 9 is stored in consecutive locations starting at number 848; the “high byte” starting at 858.

The same applies to $V3=40*(19-KK)$: the high and low bytes are evaluated at address 210; the low bytes are stored in locations starting at 868, and the high bytes starting at 878.

In summary, for KK ranging from 0 to 9, we have:

19-KK in locations 838 through 847

Low byte of $40*KK$ in locations 848 through 857

Hi-byte of $40*KK$ in locations 858 through 867

Low byte of $40*(19-KK)$ at addresses 868 to 877

Hi-byte of $40*(19-KK)$ at addresses 878 to 887

Let's now move on to the Assembly section. To generate the random numbers, I used a code I had found on the Internet a long time ago. I had already used it in an article in this magazine (“The ‘Mini-Lotto’ Game” in issue 47 of RMW), where those interested can find a summary of the explanation Marco Pistorio had provided me regarding this code.

The generator requires “initialization”:

```
lda #$ff          ;*****
sta $d40e         ; initialization
sta $d40f         ; of the generator
lda #$80          ; of random numbers
sta $d412         ;*****
```

after that, every time the contents of location \$D41B are





read, a random value in the range 0-255 is obtained.

After the first instructions (the `*=$C000` indicates the starting location of the Assembly code), the main LOOP begins: these are the instructions between the LOOP label and the JMP LOOP instruction. Within this loop, a pair of subroutines is accessed multiple times, which we should analyze now:

SUBROUTINE TETA

```
; subroutine teta: a random digit 0 to9
; is generated and stored in A
teta
alfa lda $d41b      ; in A rnd [0,255]
      and #$0f      ; now in A [0,15]
      cmp #$0a      ; greater than or equal to 10?
      bcs alfa      ; if yes, return to alfa
      rts
```

The random number generated in \$D41B is loaded into the A register; the AND #\$0F performs a logical AND with the value \$0F, which is 00001111 in binary. The effect is to clear the four most significant bits and leave the four least significant bits unchanged. The accumulator will now contain a value between 0 and 15. The CMP #\$0A compares A with 10, and BCS ALFA jumps to the ALFA label if A is greater than or equal to 10. The return to the main program (RTS) occurs only if A is less than 10, i.e., if A is an integer between 0 and 9.

SUBROUTINE BETA

As we saw in CALEIDO1.BAS, in the main LOOP the same color must be loaded into eight locations: $XX+40*YY+55296$, $YY+40*XX+55296$, $(19-XX)+40*YY+55296$, etc. (Here we use XX and YY to avoid confusion with the X and Y registers that come into play in the Assembly code).

The first operand, XX or YY or 19-XX or 19-YY, occupies a single byte; the second ($40*(..)$) occupies two. The third operand, 55296, is \$D800 in hexadecimal, so its low byte is 00 and its high byte is \$D8.

The task of the BETA subroutine is to perform the addition and then insert the color code, located at address 890, into the correct location. Before the call, the first operand is loaded into address 891, the low byte of the second into 892, and the high byte into 893.

Therefore, regarding the required addition, we have:

LOW BYTE OF THE SUM: contents of 891 + contents of

892 (the low byte of the third operand is zero, so it is not added)

HIGH BYTE OF THE SUM: contents of 893 + \$D8 (+ any carry over from the previous addition).

The low byte of the sum is stored at location 894, the high byte at location 895.

To transfer the contents of the Accumulator to a 16-bit memory location, indirect addressing must be used. With the instruction STA(zero page),Y, the contents of A are stored in the memory location whose least significant byte (LSB) is located at the "zero page" address and whose most significant byte (MSB) is located at the address immediately following it. Once the 16-bit address is retrieved, the processor adds the contents of the Y register to obtain the destination address.

In the BETA routine, the "zero page" address is \$FB: after transferring the contents of 894 and 895 to \$FB and \$FC (LDA 894, STA \$FB, LDA 895, STA \$FC), simply load the relevant color code into A (LDA 890) and store it in the correct location (STA(\$FB),Y) after clearing the contents of the Y register (LDY #00).

Here is the code:

```
; routine that performs the sum and places
; color C at the location
beta
      clc                ; clear the carry flag
      lda 891
      adc 892
      sta 894            ; low byte of the sum
      lda #$d8
      adc 893
      sta 895            ; high byte of the sum
      lda 894
      sta $fb            ; zero page
      lda 895
      sta $fc            ; zero page +1
      lda 890            ; load color code
      ld y #00
      sta ($fb),y
      rts
```

Here is the complete code for the CAL2.ASM program, in which we still need to analyze the main loop:





```

*= $c000                ;sys 49152                jsr beta

    lda #$ff            ;*****
    sta $d40e           ; initialization            ; xx+40(19-yy) (4)
    sta $d40f           ; of the                    lda 888                ; xx in A
    lda #$80            ; random number            sta 891                ; xx in 891
    sta $d412           ; generator                ldx 889                ; value yy (offset)
                    ;*****                in register X
loop    lda $d41b        ; into A rnd [0,255]        lda 868,x            ; low byte of 40(19-yy)
    sta 890              ; in 890 color code        sta 892                ; goes to 892
    jsr teta            ; now in A rnd [0,9]        lda 878,x            ; high byte of 40(19-yy)
    sta 888              ; xx in [0,9] goes to 888    sta 893                ; goes to 893
    jsr teta
    sta 889              ; yy in [0,9] goes to 889
                    ; (19-xx)+40(19-yy) (5)
; xx+40yy (1)                ldx 888                ; xx in register X
    lda 888              ; xx in A                lda 838,x            ; 19-xx in A
    sta 891              ; xx in 891            sta 891                ; 19-xx in 891
    ldx 889              ; yy value (offset)        ldx 889                ; value yy (offset)
in register X                in register X
    lda 848,x            ; low byte of 40yy        lda 868,x            ; low byte of 40(19-yy)
    sta 892              ; goes to 892            sta 892                ; goes to 892
    lda 858,x            ; high byte of 40yy        lda 878,x            ; high byte of 40(19-yy)
    sta 893              ; goes to 893            sta 893                ; goes to 893
    jsr beta                jsr beta

; yy+40xx (2)                ; (19-yy)+40(19-xx) (6)
    lda 889              ; yy in A                ldx 889                ; yy in register X
    sta 891              ; yy in 891            lda 838,x            ; 19-yy in A
    ldx 888              ; value xx (offset)        sta 891                ; 19-yy in 891
in reg X                ldx 888                ; value xx (offset)
    lda 848,x            ; low byte of 40xx        in register X
    sta 892              ; goes to 892            lda 868,x            ; low byte of 40(19-xx)
    lda 858,x            ; high byte of 40xx        sta 892                ; goes to 892
    sta 893              ; goes to 893            lda 878,x            ; high byte of 40(19-xx)
    jsr beta                sta 893                ; goes to 893
                    jsr beta

; yy+40(19-xx) (3)                ; (19-yy)+40xx (7)
    lda 889              ; yy in A                ldx 889                ; yy in register X
    sta 891              ; yy in 891            lda 838,x            ; 19-yy in A
    ldx 888              ; value xx (offset)        sta 891                ; 19-yy in 891
in reg X                ldx 888                ; value xx (offset)
    lda 868,x            ; low byte of 40(19-xx)    into reg X
    sta 892              ; goes to 892            lda 848,x            ; low byte of 40xx
    lda 878,x            ; high byte of 40(19-xx)    sta 892                ; goes to 892
    sta 893              ; goes to 893

```





```

        lda 858,x      ; high byte of 40xx
        sta 893        ; goes to 893
        jsr beta

; (19-xx)+40yy (8)
        ldx 888        ; xx in reg X
        lda 838,x      ; 19-xx in A
        sta 891        ; 19-xx in 891
        ldx 889        ; value yy (offset)
into reg X
        lda 848,x      ; low byte of 40yy
        sta 892        ; goes to 892
        lda 858,x      ; high byte of 40yy
        sta 893        ; goes to 893
        jsr beta
        jmp loop

; subroutine teta: generates in A
; a random digit between 0 and 9
teta
alfa    lda $d41b      ; in A rnd [0,255]
        and #$0f       ; now in A [0,15]
        cmp #$0a       ; is it greater than
or equal to 10?
        bcs alfa       ; if yes, return to alfa
        rts

; routine that performs the sum and places
; color C at the location
beta
        clc            ; clear the carry flag
        lda 891
        adc 892
        sta 894        ; low byte of the sum
        lda #$d8
        adc 893
        sta 895        ; high of the sum
        lda 894
        sta $fb        ; zero page
        lda 895
        sta $fc        ; zero page +1
        lda 890        ; load color code
        ldy #00
        sta ($fb),y
        rts

```

Let's start with the instructions beginning at the LOOP label:
 * A random number is generated, representing the color code to be assigned to the eight cells; this value is stored at location 890

* Using the TETA subroutine, the two values XX and YY are generated—two random integers between 0 and 9. XX is stored at location 888, YY at 889.

At this point, eight times, we need to retrieve the correct elements from the vectors prepared in CAL1.BAS; these elements are passed to the BETA routine, which will set the color. The individual instructions are briefly commented on within the code, but let's look at a couple of examples in detail to better understand the mechanism:

Example 1: XX+40*YY

lda 888: the value of XX goes into A (the Accumulator)
 sta 891: now XX goes into 891, ready for the BETA routine
 ldx 889

the value YY goes into the X register

lda 848,x now A contains the element stored in 848+YY

that is, the low byte of 40*YY

sta 892

the low byte of 40*YY goes to location 892

ready for the BETA routine

lda 858,x now A contains the element in 858+YY

that is, the high byte of 40*YY

is 893

the high byte of 40*YY goes to location 893

ready for the BETA routine

The parameters required by the BETA routine are in place; we are ready for ...

jsr BETA

Example 2: (19-YY)+40*XX

ldx 889

the value of YY goes into the X register

lda 838,x the value contained in 838+YY goes into A

that is, the value 19-YY

sta 891

(19-YY) goes to 891





ready for the BETA routine

ldx 888

XX is now loaded into register X

lda 848,x

...

from here, proceed as in Example 1.

After the eight code segments, we reach the JMP LOOP for a new pair of coordinates and a new color.

We're at the end: use the "Generate BASIC loader" tool, close the "Assembly Dump" window, and choose a "Start Line Number" higher than the line numbers used in CAL1.BAS. I chose 1000 with an increment of 20. The result should be copied and pasted into the CAL1.BAS program.

Below is the final program, CAL3.BAS, that you obtain. You can copy it, paste it, and run it as a new Basic file in CBM Prg Studio without having to manually use "Generate BASIC loader."

CAL3.BAS:

```
100 print chr$(147):sc=55296
110 poke 53280,0:poke 53281,0
120 for x=0 to 19:for y=0 to 19
130 poke 1024+x+40*y,81
140 poke sc+x+40*y,0
150 next:next
160 for kk=0 to 9
170 v1=19-kk:v2=40*kk:v3=40*v1
180 poke 838+kk,v1
190 h=int(v2/256):l=v2-256*h
200 poke 848+kk,l:poke 858+kk,h
210 h=int(v3/256):l=v3-256*h
220 poke 868+kk,l:poke 878+kk,h
230 next kk
1000 REM GENERATED ML LOADER
1020 SA = 49152
1040 FOR N = 0 TO 283
1060 READ A% : POKE SA+N,A%: NEXT N
1080 SYS SA
1100 END
1120 DATA 169,255,141,14,212,141,15,212
1140 DATA 169,128,141,18,212,173,27,212
1160 DATA 141,122,3,32,238,192,141,120
1180 DATA 3,32,238,192,141,121,3,173
1200 DATA 120,3,141,123,3,174,121,3
1220 DATA 189,80,3,141,124,3,189,90
```

```
1240 DATA 3,141,125,3,32,248,192,173
1260 DATA 121,3,141,123,3,174,120,3
1280 DATA 189,80,3,141,124,3,189,90
1300 DATA 3,141,125,3,32,248,192,173
1320 DATA 121,3,141,123,3,174,120,3
1340 DATA 189,100,3,141,124,3,189,110
1360 DATA 3,141,125,3,32,248,192,173
1380 DATA 120,3,141,123,3,174,121,3
1400 DATA 189,100,3,141,124,3,189,110
1420 DATA 3,141,125,3,32,248,192,174
1440 DATA 120,3,189,70,3,141,123,3
1460 DATA 174,121,3,189,100,3,141,124
1480 DATA 3,189,110,3,141,125,3,32
1500 DATA 248,192,174,121,3,189,70,3
1520 DATA 141,123,3,174,120,3,189,100
1540 DATA 3,141,124,3,189,110,3,141
1560 DATA 125,3,32,248,192,174,121,3
1580 DATA 189,70,3,141,123,3,174,120
1600 DATA 3,189,80,3,141,124,3,189
1620 DATA 90,3,141,125,3,32,248,192
1640 DATA 174,120,3,189,70,3,141,123
1660 DATA 3,174,121,3,189,80,3,141
1680 DATA 124,3,189,90,3,141,125,3
1700 DATA 32,248,192,76,13,192,173,27
1720 DATA 212,41,15,201,10,176,247,96
1740 DATA 24,173,123,3,109,124,3,141
1760 DATA 126,3,169,216,109,125,3,141
1780 DATA 127,3,173,126,3,133,251,173
1800 DATA 127,3,133,252,173,122,3,160
1820 DATA 0,145,251,96
```

Now, to finish up, a couple more versions. Here's the one for Marco Spedaletti's splendid ugBASIC:

```
OPTION DEFAULT TYPE BYTE
RANDOMIZE TIMER
POKE 53272,21
CLS:w$=CHR$(209)
COLOR BACKGROUND 0
COLOR BORDER 0
x1=10:y1=10
DO
x=RND(x1):y=RND(y1)
c=RND(15):PEN c
LOCATE x,y:PRINT w$
```





```
LOCATE 19- x,y:PRINT w$
LOCATE y,x:PRINT w$
LOCATE y, 19 - x: PRINT w$
LOCATE x,19-y:PRINT w$
LOCATE 19-x,19-y:PRINT w$
LOCATE 19-y,x:PRINT w$
LOCATE 19-y,19-x:PRINT w$
LOOP
```

In this version, POKE is not used; the cursor is positioned using the LOCATE command before printing the “solid

dot” (which is CHR\$(209) if preceded by POKE 53272,21), and color C is assigned using the PEN C command (the KEYWORDS are described in detail in the “User Manual”).

Quite a few years ago, I discovered “Turbo Rascal Syntax Error,” which had recently appeared on the scene; naturally, I immediately put it to the test with my CALEIDOSCOPIO. Regarding my little program, I exchanged a few emails with Nicolaas Groeneboom, the author of TRSE, who suggested I include it among the tutorials.

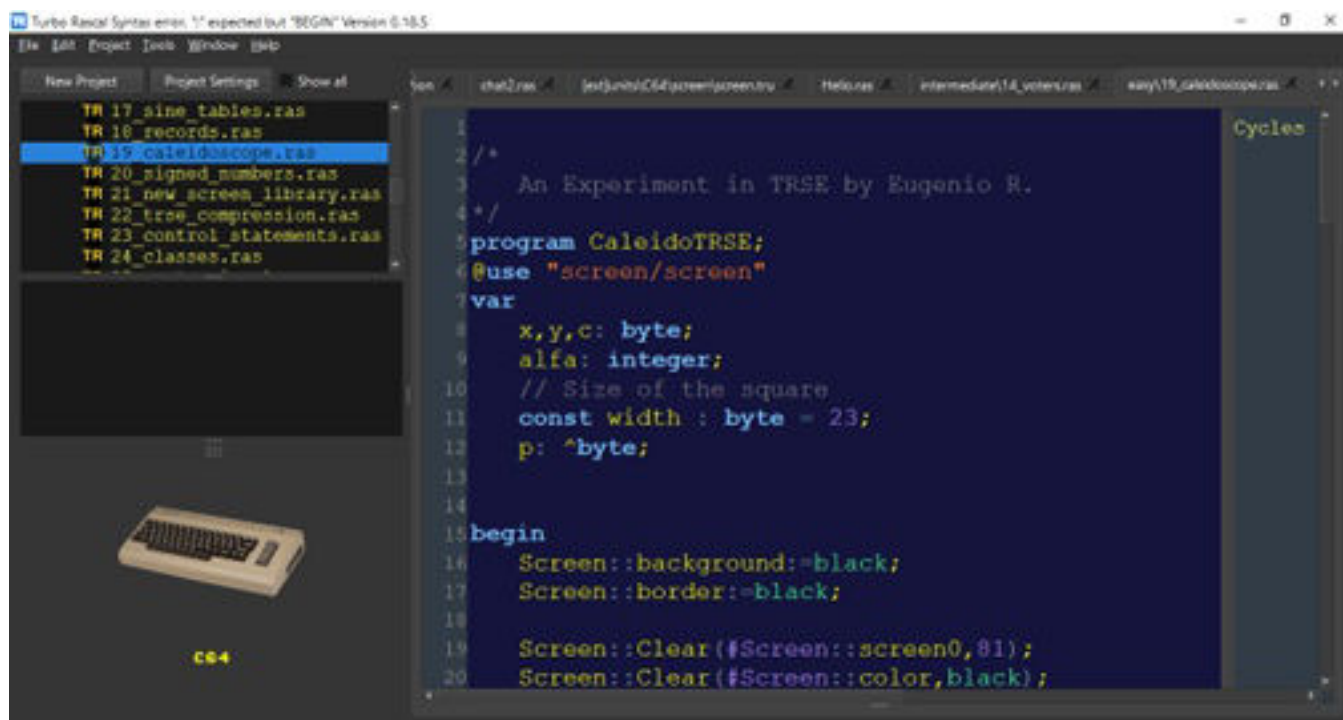


Fig. 4 - C64 Tutorials > easy > TR 19 caleidoscope.ras

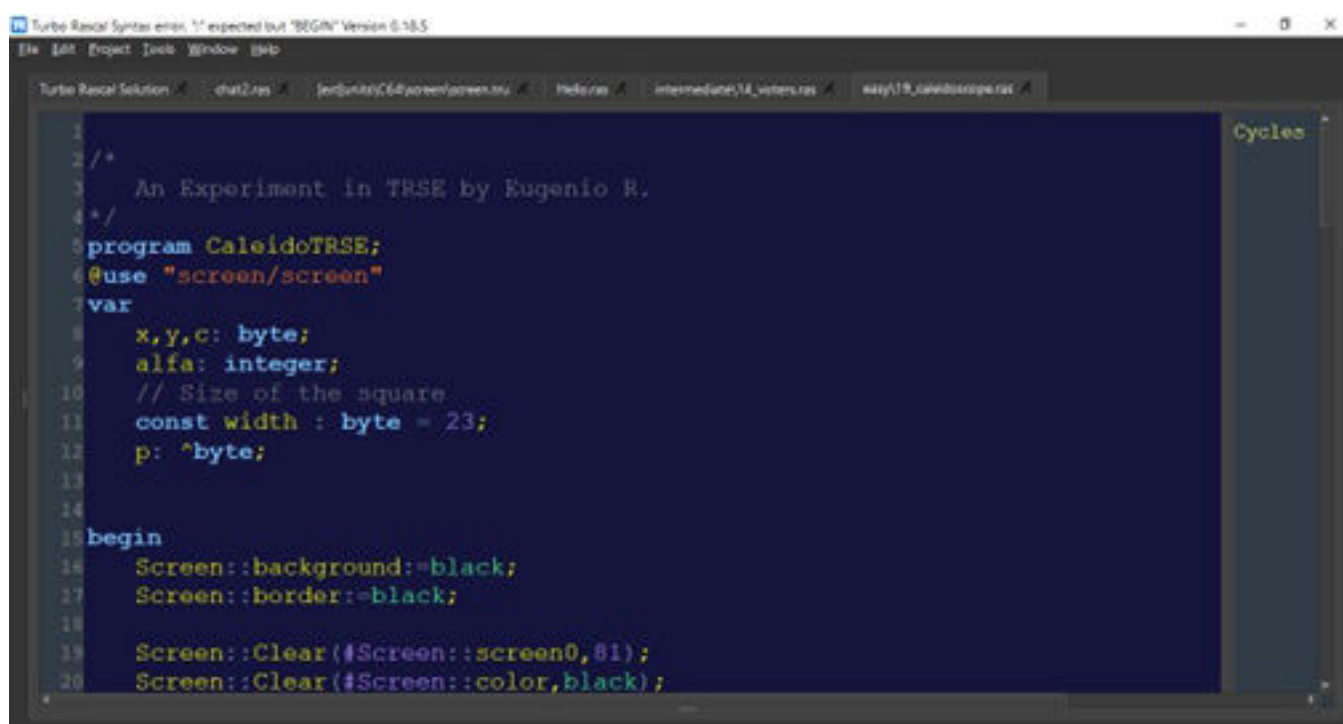


Fig. 5 - The caleidoscope.ras program loaded into the TRSE editor





A Magic Trick (for C64 beginners)

by Eugenio Rapella

On YouTube, you can find dozens and dozens of videos related to magic tricks with playing cards and beyond. Usually, the presentation of the trick is followed by a "tutorial" where the secret is revealed. There's a really nice guy, Matt McGurk, who presents many tricks. Here's one of them:

Form a deck by choosing 15 cards from a 52-card deck, take a look at the last card of the deck you've created, memorize it, and invite a spectator, let's call him Francesco, to participate in the trick. To start, say the word "MAGIC" letter by letter; with each letter of the spelling, move a card from the top of your deck and create a small stack of five cards in front of you, then drop the rest of the deck on top of the stack you've created (naturally, the cards are always face down). At this point, in great secrecy, write the memorized card on a piece of paper, and set the paper aside, making sure Francesco notices that what you've written can no longer be changed. Now ask Francesco to choose one of the 52 cards in the deck and, again by spelling, count the cards of your deck from the top, creating a new stack. In the end, drop the remaining cards onto the stack you've created. For example, if Francesco chooses the Queen of Hearts, you will spell "QUEEN OF HEARTS," thus moving 13 cards (for now, we keep the English language, in accordance with the video; we'll see later how to modify everything to perform the trick in Italian). During this phase, you can point out to Francesco that the number of cards moved depends on the chosen card, e.g., "TWO OF CLUBS" involves moving 10 cards and not 13 as in the previous example, and... the note has already been written!

We are almost at the end: now spell the word "TRICK" and, as before, create a new stack of five cards, on top of which you will drop the remaining block of cards.

Retrieve the note: your prediction will magically coincide with the card on top of the deck!

How is this possible?

To reveal the mystery, our C64 comes to our aid. Consider N face-down stacked cards, i.e., with the back facing the

spectator, and let $S1$ be the position of a card counting from the bottom. Now imagine taking K cards (with K between 1 and N) from the top, creating a new stack with them, and placing the remaining deck block on top. What is the new position $S2$ (still from the bottom) of the card that was in position $S1$?

Case 1: $K \leq N - S1$

If K is less than or equal to $N - S1$, the K cards moved end up at the bottom of the new deck, and placing the rest of the deck block on top causes the card in position $S1$ to move up by K spots, so $S2 = K + S1$. For example, if $N = 20$, $S1 = 3$ and $K = 5$, when the new deck is created, the five cards taken from the top end up at the bottom, and when the remaining block is placed on top, the card that was in the third position from the bottom ends up in the eighth position (still from the bottom, of course).

Case 2: $K > N - S1$

The situation is different if K is greater than $N - S1$ because, in this case, the movement of the K cards also involves the one in position $S1$. The first $N - S1$ cards from the top become the first $N - S1$ cards from the bottom in the new deck, and then comes the card that was in position $S1$. This is why we will have $S2 = N - S1 + 1$. For example, if $N = 20$, $S1 = 12$ and $K = 15$, after moving the first $20 - 12 = 8$ cards, it will be the turn of the card that was in position $S1 = 12$, which will occupy the ninth position from the bottom: $8 + 1 = 20 - 12 + 1 = N - S1 + 1$.

C64 BASIC Implementation

In C64 Basic, logical expressions like $(A < B)$ have the value -1 if true, and the value 0 if false. Therefore, we can define a function that, starting from the number X of cards intended to be moved to create the new stack, provides the new position $NP(X)$ of a card initially in position $S1$. In instruction 10 of the following program, if X is less than or equal to $N - S1$, the value of $(X \leq N - S1)$ is -1 while that of $(X > N - S1)$ is 0 , so the function $FN NP(X)$ takes the value $X + S1$. Otherwise, $(X \leq N - S1) = 0$, $(X > N - S1) = -1$, and $FN NP(X)$ becomes $N - S1 + 1$.

In the magic trick, as described, a deck formed by $N = 15$





cards is used; initially, the "secret card" to be noted on the paper is the last of the 15 cards, therefore in position $S1=1$ counting from the bottom. For convenience, let's call ALFA(Y) the procedure "slide a certain number Y of cards to create a stack and place the remainder on top"; ALFA(Y) is repeated three times: the first with $Y=5$ (with the spelling of MAGIC), the second with a variable number Y (e.g. $Y=10$ in the case of ACE OF CLUBS), and the third again with $Y=5$ (with the spelling of TRICK).

The MCGURK program manages the situation more generally. Its goal is to see how the final position of a certain card changes as the various parameters vary, especially the variation of the number of cards in the second repetition of the procedure we described:

Here is the MCGURK listing:

```
10 def fn np(x)=-(x<=n-s1)*(x+s1)-(x>n-s1)*(n-s1+1)
20 print chr$(5):input"totale carte ";n
30 input"posizione carta segreta";s1
40 input"n. carte mazzetto";m
50 sw=s1:print
60 for k=1 to n
70 a=fn np(m):s1=a:b=fn np(k):s1=b:c=fn np(m)
80 print k,a;b;c
90 s1=sw:next
```

At line 10, the function is defined which, given a card in position S1, provides its new position NP(X) when the ALFA(X) procedure is executed. At line 20, the program asks for the number of cards (variable N) used for the trick, and at line 30, the initial position of the card to be identified (variable S1). At line 40, it asks how many cards are involved the FIRST and THIRD time the ALFA procedure is executed (variable M). In the trick, as proposed, this value is 5, which is the number of letters in the word MAGIC and the word TRICK.

At this point, with N, S1, and M fixed, the C64 checks what happens as the number of cards in the SECOND instance of the ALFA procedure varies, because this is the value that depends on the card chosen by the spectator for the second spelling.

The position of the "secret card" is transferred to the variable SW, and... we start:

In the loop that begins at line 60, for K that goes from 1 to N:

- A becomes the position of the card originally in S1 after

the ALFA(M) procedure.

- The value of A becomes the new S1, because that is the new starting position before the second spelling.
- B becomes the position of the "secret card" after the second spelling; here, the number of cards moved is K.
- The value of B becomes the new S1, because that is the new starting position before the third spelling.
- C becomes the position of the "secret card" after the third spelling.

At line 80, the value of K (number of cards used for the second stack) and the position of the "secret card" after each of the three ALFA procedures are printed.

At line 90, the initial S1 value is restored for the next K. The mechanism becomes clearer if we run the program with the values from the magic trick as proposed by Matt McGurk:

TOTAL NUMBER OF CARDS is 15

SECRET CARD POSITION is 1 (the first from the bottom)

STACK CARD N. is 5 (number of cards used for the first – MAGIC – and third – TRICK – spelling).

Here is the result:

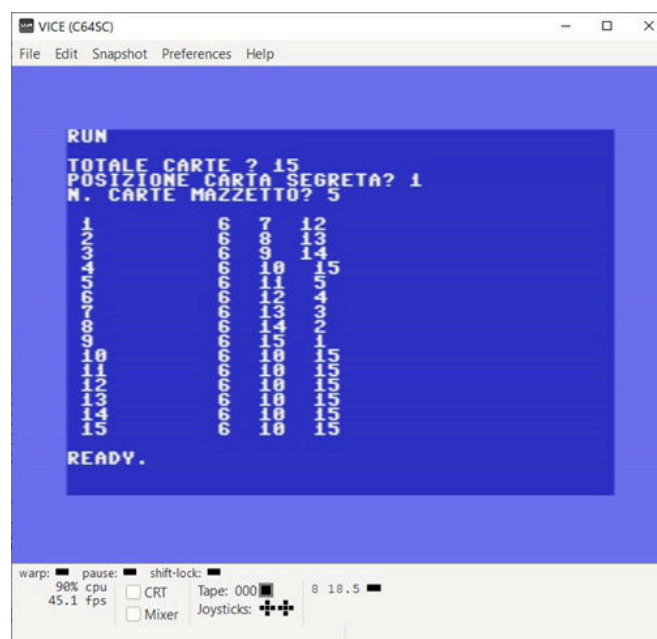


Fig. 1 - Screen 1

Let's look, for example, at the values in the seventh row: 6, 13, 3 which are obtained for $K=7$. The meaning is this: if the initial position of the "secret card" is the first from the bottom, after the first ALFA procedure (spelling of MAGIC), the "secret card" moves to position 6, after the ALFA procedure with a shift of $K=7$ cards it will go to position 13, and finally, after a new ALFA procedure (spelling of TRICK) with 5 cards, it will be in position 3,





meaning it will be the third card from the bottom.

You can try it: take a stack of 15 cards with the Joker at the bottom of the deck. Perform the ALFA procedure three times with 5, 7, 5 cards and you will see that the Joker will end up in the third position from the bottom. If instead you perform the ALFA procedure with the sequence 5, 2, 5 you will find the Joker in the thirteenth position from the bottom, as predicted by the second row of the screen. Now focus your attention on the last 5 rows of SCREEN 1: the final value is 15 regardless of the value of K from 10 to 15. This means that, if you execute the ALFA procedure with the sequence 5, K, 5, you will always get the secret card at the top of the deck (which corresponds to position 15 from the bottom) if K is 10 or 11 or 12 or 13 or 14 or 15.

Of course, you could ask the spectator to choose a number between 10 and 15 for the second ALFA procedure, but the trick proposed by Matt is more subtle: he asks the spectator to choose one of the 52 cards in the deck and then... performs the spelling.

Apparently, the spectator has many more choices, but... let's see what can happen.

Let's list the possible card values accompanied by the number of letters that express the value in English. We will have (A,3) because A=ACE is formed by 3 letters; (2,3) because 2=TWO is formed by 3 letters, and so on: (A, 3), (2, 3), (3, 5), (4, 4), (5, 4), (6, 3), (7, 5), (8, 5), (9, 4), (10, 3), (J, 4), (Q, 5), (K, 4)

The number of letters ranges from a minimum of 3 (ACE, TWO, SIX, TEN) to a maximum of 5 letters (THREE, SEVEN, EIGHT, QUEEN).

Regarding the suits, we have HEARTS with 6 letters, DIAMONDS with 8 letters, CLUBS with 5 letters, and SPADES with 6 letters: a minimum of 5 letters and a maximum of 8.

When spelling the cards, we also have two letters for the preposition OF. For example, with FIVE OF SPADES, a total of $4+2+6=12$ letters is obtained.

As stated earlier, whatever card the spectator chooses, the spelling for the second ALFA procedure will range from a minimum of 10 letters (e.g., SIX OF CLUBS) to a maximum of 15 (e.g., with QUEEN OF DIAMONDS), which are exactly the values that all lead to the same final position of the "secret card."

A Small "Interlude"

Looking at SCREEN 1, it may seem strange that the final position (15) does not depend on the ALFA(K) procedure for as many as six different values of K (from 10 to 15). After all, the card movement is different, and this should, in some way, modify the final position of our "secret card." Naturally, the magician relies precisely on this aspect: the maneuver has every appearance of shuffling the deck randomly.

In reality, in the ALFA procedure, when the "secret card" has already been placed on the new stack, it matters little whether the subsequent cards are placed on top due to spelling or dropped in a block: the new position of the secret card will not change (if you do a few experiments with cards, you will realize this easily).

The Trick in Italian?

But let's go back to the magic trick: will it also work if performed in Italian?

Regarding the values of the 13 cards, we have ASSO with 4 letters, DUE with 3 letters, etc. This time, the card value with the minimum number of letters is RE (King), which is formed by only 2 letters, while the maximum number of letters is obtained for QUATTRO (Four): a full 7 letters (the FANTE (Jack) has 5 letters or 4 letters if indicated as JACK; the DONNA (Queen) provides 5 letters or 6 letters if called REGINA (Queen). These choices are not important because they do not modify the minimum and maximum value of the number of letters).

Regarding the suits, we have FIORI (Clubs): 5 letters, QUADRI (Diamonds): 6 letters, PICCHE (Spades): 6 letters, and CUORI (Hearts): 5 letters.

In Italian, we also need to add exactly 2 letters for the preposition DI (of) so, whatever card Francesco chooses, the spelling will range from a minimum of 9 letters (RE DI FIORI or RE DI CUORI) to a maximum of 15 (QUATTRO DI QUADRI or QUATTRO DI PICCHE).

Uh oh, we're in trouble. SCREEN 1 tells us that for K=9, the final position will not be at the top of the deck.

What to do? The MCGURK program helps us find other solutions. Let's try to change the number of cards in the first and third stacks and increase this value to 6.

Here is the result:



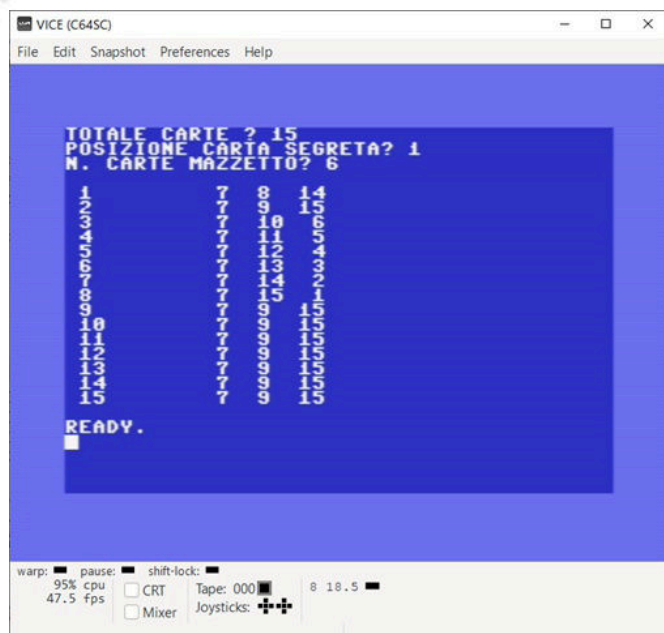


Fig. 2 - Screen 2

We made it! For all K from 9 to 15, the third value is 15. The trick will also work in Italian if, instead of using the pair (MAGIC, TRICK), we use one with 6-letter words, for example (PAROLA (Word), MAGICA (Magical)).

So far, we've used the MCGURK program to analyze the trick proposed in the video. However, we can run it with new values for N, S1 and M and thus invent new magic tricks! Let's run the program and proceed, for example, with:

TOTAL CARDS: 20 (N=20)
 SECRET CARD POSITION: 10 (S1=10)
 STACK CARD N.: 6 (M=6)

Our trusty Commodore informs us that the secret card will end up in position 11 for all K values between 5 and 20. Here is a brand-new magic trick (it really works, try it!):

The New 20-Card Trick

1. Tell Francesco, our spectator, that you need a deck of 20 cards. Take the first 20 from a well-shuffled deck and take the opportunity to peek at the card at the bottom of the deck. Memorize it: this will be your prediction. In his videos, Matt also shows how you can shuffle the cards and cut the deck without changing the position of the last card; these actions enhance the performance but are not essential.

2. Tell Francesco that you will reshuffle the deck in various ways and, to start, create a stack of 9 cards by taking three groups of 3 cards at a time from the top, and then drop the rest of the deck in a block onto the 9-card stack.

This will bring the card you memorized to position 10 (counting from the bottom).

3. Take a piece of paper and, in great secrecy, write down your prediction. The note is set aside and can no longer be modified.

4. Execute the ALFA(6) procedure: you will spell the term PAROLA (6 letters) creating a stack of 6 cards and place the remaining cards in a block on top.

5. At this point, ask Francesco for the "first and last name" of a person of his choice and execute the ALFA procedure. If, for example, Francesco says "IVY FIORENTINI," you will count 13 cards to form the new stack and drop the remaining cards in a block. Here you have ample choice since the value associated with the maneuver just needs to be between 5 and 20. For example, after forming the 13-card stack, you could say to Francesco: "If you want, I can add 0, 1, 2, or 3 cards to your stack. What do you want me to do?" This is to give the impression of further shuffling the deck.

6. As in step 4, execute the ALFA(6) procedure using the term MAGICA (6 letters).

7. At this point, the "secret card" is in position 11 from the bottom, or position 10 from the top. You can proceed as you deem best by discarding nine cards and leaving the card on top, or by spelling a nine-letter phrase, for example, "ecco qui" (literally "here it is", or you can use "there it is" to obtain a nine-letter phrase in English as well).

8. Before turning over the card, ask Francesco to read what is written on the note and... the trick is done.

Naturally, in step 5, you can replace the "first and last name" with an alternative; moreover, in the video, Matt suggests some misdirection maneuvers to make the trick more interesting (subscribe to his channel: Matt is a really nice guy!).

Be honest, you didn't expect the C64 to help you become a magician!



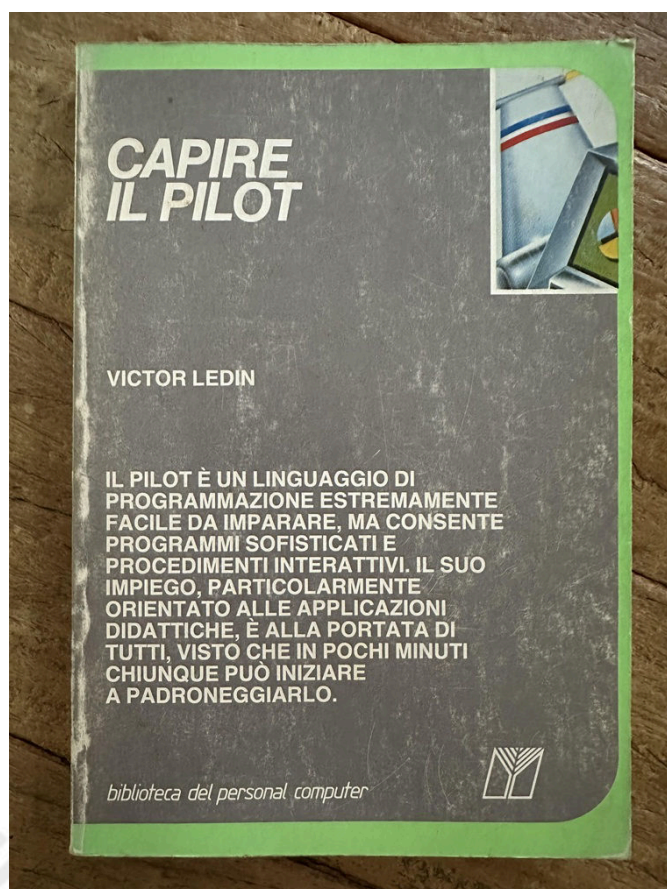


Super Turtle Pilot - story of an adventurous recovery

by Francesco Fiorentini

It all started with this book: Understanding the Pilot by Victor Ledin. I am an avid collector of books on computer science from before the year 2000, and this book was definitely one that deserved a place in my now well-stocked collection.

I already own other books from Franco Muzzio Editore, and I must admit that, before receiving it, I expected a book slightly larger than the almost “pocket-sized” format of this text.



PILOT

PILOT, an acronym for Programmed Inquiry, Learning, or Teaching, is a language developed in 1968 by John Starkweather at the University of California Medical Center in San Francisco. Being a non-algebraic language, it is easier to learn than many other languages of the time, such as Fortran, Basic, and Pascal. Its linear structure, although different from that of other languages, is ideal for a first approach to computers by programming novices. Of course, the Pilot language is available on many of our retro computers, and the Commodore 64 is certainly no

exception. Driven by curiosity, I therefore set out to find the versions available for Commodore's 8-bit machine.

Here is a list taken from the website: <https://telarity.com/~dan/cbm/languages.html#PILOT> - (filtered for C64)

PILOT [C64] (1983, CBM)

All the features of Common PILOT, plus interfaces for floppy disk drives and tape drives. Available for purchase at The Computer Bargain Store.

PILOT [C64,VIC,PET] (Michael Tinglof, COMPUTE! Dec. 1982, Vol.4,No.12,Iss.31)

A basic PILOT interpreter written in BASIC, so not very fast. It likely also works on the C128 and C264. It has also been ported to the KIM-1.

PILOT II [C64] (1983, Tamarack Software)

Includes 25 editing commands, 19 turtle graphics commands, and 23 programming commands with various subcommands and options. Supports high- and low-resolution graphics and sprites, and includes a sprite editor.

Vanilla PILOT [C64, PET] (1982, Tamarack Software, Brøderbund)

It supports “turtle graphics,” “find/replace,” and “trace and dump” commands, which are useful for system diagnostics and debugging.

As expected, the Commodore 64 never disappoints; think of any software, and our beloved 8-bit machine surely had an implementation of it. Incredible.

To leave no stone unturned, however, I decided to ask DeepSeek's artificial intelligence to point out other versions of Pilot for the C64, and sure enough, the AI indicated variants not present in the list above:

- **Commodore PILOT (1982/1983):** The official version sold by Commodore itself on floppy disk. It came with a manual and included utilities for creating sprites and sound synthesis, highlighting its capabilities that went far beyond simple text.

- **Super Turtle PILOT (1987):** A powerful version published in COMPUTE! magazine. It significantly improved the language by adding dedicated commands for turtle





graphics, sprite manipulation, music, and even disk file management. Perhaps its most notable feature was the ability to embed BASIC commands directly within a PILOT program, allowing them to share variables and combine the strengths of both languages.

- **Pilot-Plus/64:** Another commercial interpreter praised for its extensive and useful set of programming utilities. It included commands such as RENUMBER, FIND, AUTO (for automatic line numbering), and PLIST (to print a program's list), which were far more advanced than the standard BASIC editor of the time.

As you know, to get the most out of the Commodore 64, you need direct access to its memory (Basic's PEEK and POKE commands), so I asked Deepseek a follow-up question to find out which versions of Pilot offered this capability.

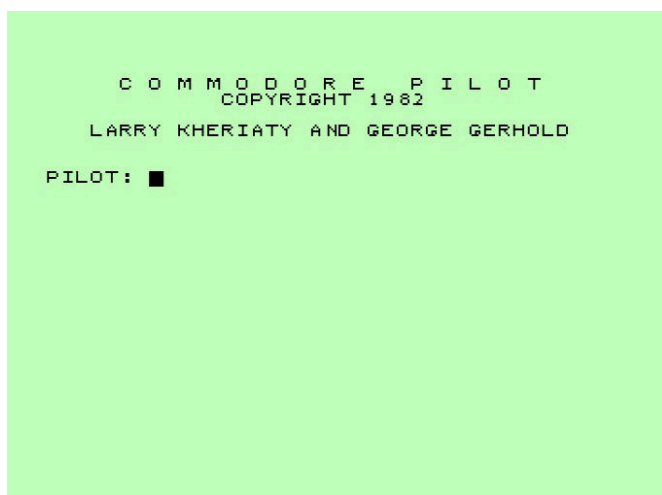


Fig. 2 - Commodore PILOT from 1982

Deepseek's response immediately pointed to Super Turtle Pilot as the only one capable of incorporating Basic commands like PEEK and POKE; DeepSeek adds fuel to the fire regarding the Commodore Pilot, arguing that the language was designed for dialogue and teaching, not for system programming. It did not have built-in commands like PEEK or POKE (it likely uses another mechanism to access memory, but I haven't investigated that yet; material for a future article... :-)).

At this point, the nerd inside me—who isn't exactly dormant—was immediately intrigued and decided to go look for this fateful Super Turtle Pilot.

Well, after a bit of searching, it quickly became clear that the Super Turtle Pilot wasn't available for download anywhere.

Sacrilege!

Compute!

DeepSeek's response, however, contained an important clue. The Super Turtle Pilot (I'll abbreviate it as STP) was software published in Compute!, an American magazine published from 1979 to 1994 that contained hundreds of listings of considerable interest to home computer users in the 1980s.

So all I had to do was find the issue containing the STP listing and, with the powerful tools available to us today, recover the ancient lost vase (oh wait, that was another story...).

Finding the magazine containing the listing was easy; after a quick Google search, the link to Archive.org for issue 89 from October 1987 popped up right away.

On page 74, the Super Turtle Pilot article takes center stage, and on page 77, the listing begins.

First hurdle: the listing is definitely in Assembly, but in hexadecimal format, ready to be fed into a utility distributed in a previous issue of Compute! called MLX—a machine-language code entry program that allows you to write the listing directly into the Commodore 64's memory locations while simultaneously checking for errors in the entered data.

Obviously, retrieving MLX and then manually copying the lines of code (there are 781 of them...) was out of the question.

I decide, however, to retrieve the code anyway, just to have it in electronic format, ready for any eventuality.

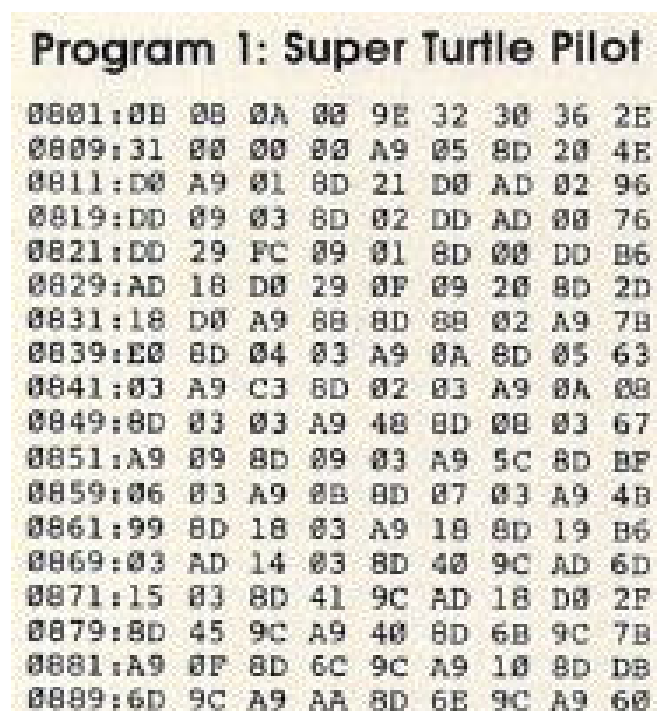


Fig. 3 - The STP listing on Compute!





I take a few screenshots and send them to DeepSeek to have him extract the listing, and within minutes, I have the code in Scite.

I then decide to call Marco Pistorio, a leading expert in C64 Assembly, to work out a strategy with him.

The DATA lines

After a brief chat, we decided we could try to create a small program in BASIC that, using DATA lines, would write the code to memory. We knew the starting and ending addresses, so we had the information to do it.

But of course, it's not that simple. After a careful examination, it becomes clear that the nine bytes that make up each line consist of 8 bytes of code and one control byte; furthermore, the Compute! listing must be converted from hexadecimal to decimal.

Removing the ninth byte is fairly simple; Scite allows selection not only horizontally but also vertically, so with just a few clicks I remove the control byte from each line. For the conversion from hexadecimal to decimal, I decide once again to enlist the help of AI. I therefore ask DeepSeek to convert all the bytes in the listing and insert them into a DATA line (starting at 1000), separating them with a comma. DeepSeek works its magic in just a few moments. We have the DATA lines to insert into a Basic program to be written to memory.

We create the little program, making sure to shift the Basic memory and leave enough space in memory for the code, and then we run it... And we get an error message. The little program crashes, messing up the emulator's Basic memory.

My technical knowledge of the Commodore 64 isn't helping, but Marco's expertise allows him to figure out the problem. The program must be executed in two steps: first the part that frees up memory, and then the part that writes the code. Marco manages to run the whole thing and... Another failure. Once the code is "injected" into memory, the program starts but does nothing except turn the background green.

Marco examines the Assembly code in the emulator's memory, and after a few lines, it becomes clear that some instructions are incorrect!

The fact that the Compute! code is incorrect doesn't even cross our minds; it's more likely that DeepSeek misread a few bytes while retrieving the code.

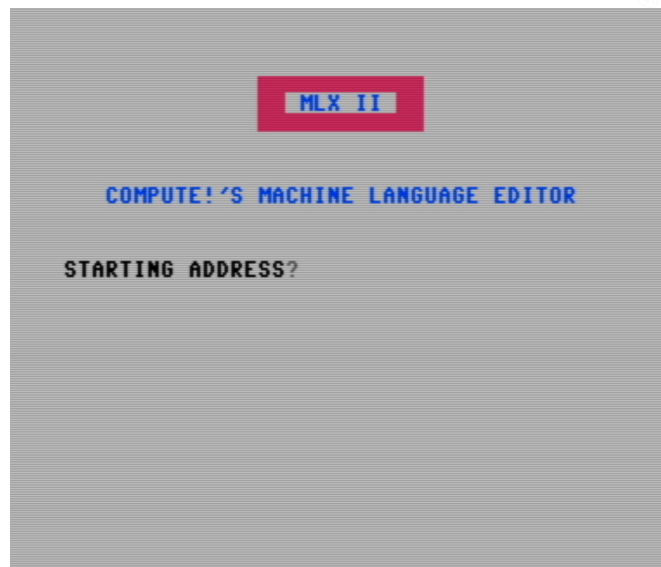


Fig. 4 - MLX II - Compute! Machine Language Editor

Confirmation comes as soon as we spot one of the first problematic lines: a byte in the line retrieved by DeepSeek is different from the one printed in the magazine...

So, do we give up? Never! Super Turtle Pilot seems to be an interesting program. Reading the article in Compute!, it's clear that it's an extension of Basic V2 and not a standalone language; despite this, however, STP isn't available anywhere, and it would be a shame to let it fade into oblivion.

It needs to be recovered, but at the same time, I can't check 6,248 bytes, one by one, by comparing the magazine and my listing... A drastic decision: I'll retrieve MLX and enter all the lines via copy-and-paste into the program, hoping that the check byte does its job.

MLX - Compute! Machine Language Editor

I was already ready to get to work on retrieving the MLX code when an idea struck me! Why not search for this program on the internet? Maybe someone, given the importance and popularity of Compute! magazine, has already done the legwork. A quick search and... Yes! MLX is already available! Let's go!

I click on the link provided by Google and find MLX available for download, ready to be used on the emulator. But, upon closer inspection, this is MLX version 1; issue 89 of Compute! refers to version II... I start to lose hope; Super Turtle Pilot is fighting tooth and nail. But then a thought occurs to me: anyone who retrieved version 1 might have done the same with the next version. Luckily, the Gaming Alexandria site has a handy search function: I try searching for "MLX II" and... Yes! The next version is there too.





I don't waste a minute downloading it and launching it on the emulator. It works. Now all I have to do is paste all my code and see where it crashes...

I copy the entire listing, paste it into MLX, and it crashes immediately. The time it takes for MLX to calculate the check byte and display the next line means that MLX can't accept more than one line at a time...

In for a penny, in for a pound!

At this point, I can taste the victory. I stoically decide to embark on the task of copying the code line by line into MLX, hoping it's worth the effort.

It seems like a long and tedious job, and it is, but after a couple of copy-and-pastes, I figure out how to speed things up a bit. With a double-click on the Scite line, I can copy the entire line, while with a single right-click on VICE, I can paste it into MLX. It's still a lot of work, but with a few hours of effort, I can manage it.

To make a long story short, it took me about three and a half hours, spread over two days, to paste all the code and correct the errors that MLX occasionally flagged. About 10% of the lines had at least one incorrect byte.

Thanks to MLX, which calculates the check byte not only based on the previous 8 bytes but also takes into account the memory location where the code must be written, I now have the Super Turtle Plot assembly code in memory. All I have to do is save the file and run it.

I save the file, exit MLX, restart VICE, and launch the file...

EUREKA! The Super Turtle Pilot screen informs me that the program is running and ready to accept my commands. The ancient vase had been saved... Oh no, Super Turtle Pilot had finally been recovered and was ready to be made available to everyone on our website!

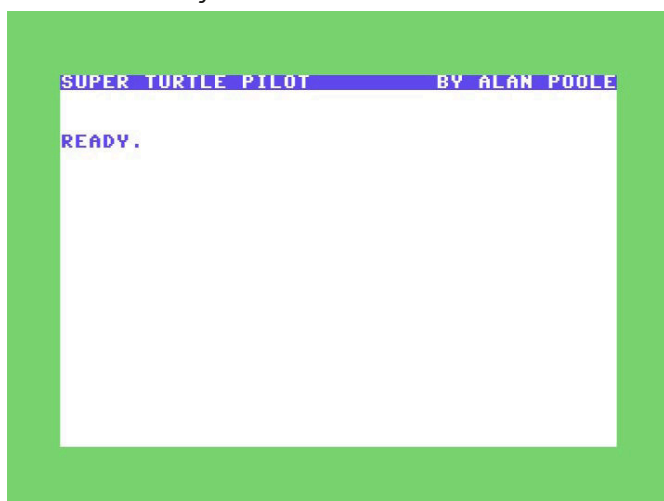


Fig. 5 - Super Turtle Pilot by Alan Poole

What is Super Turtle Pilot

STP is one of the hundreds of Basic extensions that flourished in the 1980s and 1990s. It is particularly interesting for its ability to add PILOT and Logo (Turtle) commands to Commodore 64 Basic. It is possible to combine Basic commands with the new commands added by STP, but with one limitation. You cannot enter more than one command per line because PILOT uses ":" (the colon) as part of its commands.

Please visit our website page:

<https://www.retromagazine.net/super-turtle-pilot-commodore-64-assembly/> f

or an in-depth analysis of STP commands and a programming example.

We'll surely discuss Super Turtle Pilot further in our next releases, but that's all for now. :-)

Useful links

Commodore PILOT (1982)

- <https://www.commodoreserver.com/PublicDiskDetails.asp?DID=EE24139BC8494912B5800597C17A36AD&SnapshotID=1> (richiede registrazione)

Manual of the Commodore PILOT

- https://www.retro-commodore.eu/c64-software/?rceu_page=2

Magazine Compute! - Issue: 89 - October 1987

- <https://archive.org/details/1987-10-compute-magazine>

MLX - Compute! Machine Language Editor

- <https://www.gamingalexandria.com/wp/2025/12/mlx-ii-v1/>

Super Turtle Pilot - download from the RetroMagazine website

- <https://www.retromagazine.net/super-turtle-pilot-commodore-64-assembly/>





All non-standard graphic modes of the VIC-II

by Stefano Tognon aka "Ice00"

Introduction

Although the VIC-II natively provides 8 graphics modes (3 of which are "invalid," and therefore unusable except to produce a completely black screen), the ingenuity of programmers—thanks to a deep understanding of the video chip acquired over time—has made it possible to create more than 35 new variants, turning what seemed impossible into reality. To understand how this is possible, one needs to know some basic principles of how the chip used in the Commodore 64 works.

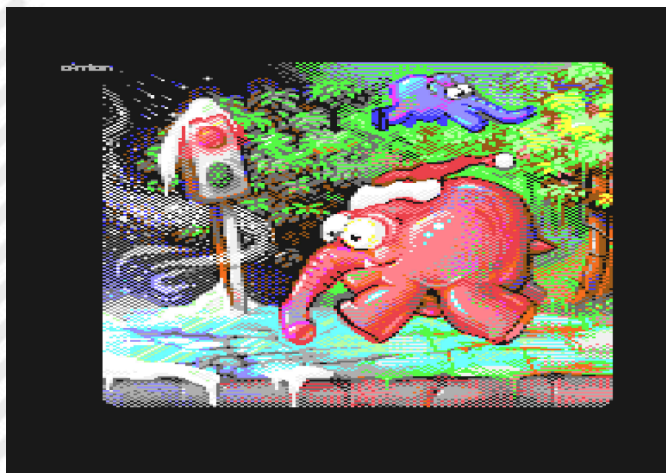
The VIC-II draws the screen pixel by pixel by controlling the raster signal (or electronic brush) of the TV/monitor from left to right and top to bottom, drawing a colored border around the screen's graphic area. When it needs to display the graphic image (which is always structured in 8x8-pixel blocks), the chip retrieves data from memory, thereby interrupting the 6510 processor's activity; the processor will then have fewer than the 63 clock cycles (normally provided by a PAL system) per line available to execute code.

The VIC-II allows an interrupt signal to be generated when it reaches a specific screen line, and—a crucial requirement for generating new graphics modes—the programmer must intervene by precisely filling the delay

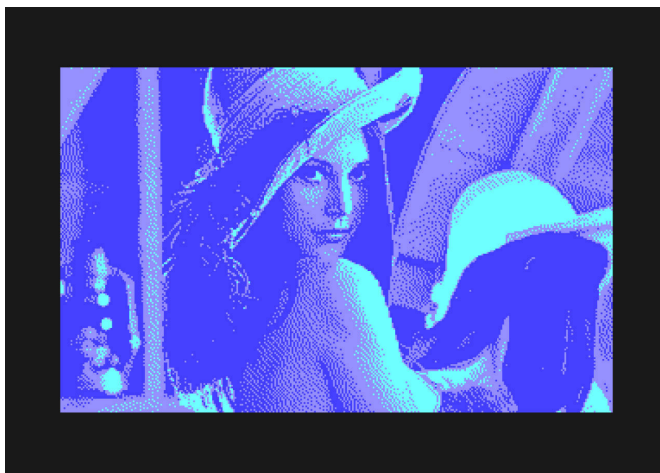
(which can range from 0 to 7 cycles) with which the processor receives the notification. At this point, various techniques can be used to achieve this result, such as the "double interrupt" or using the CIA chip's timer, to ensure the generated delay is precisely filled. But the key point is that we can know what the processor is executing at a given line in a specific cycle.

Furthermore, although the VIC-II sees (i.e., can access) only 16KB of memory (except for color RAM and character ROM, which are special cases), the central processor can instantly address it in one of the four different banks of the available 64KB, thus allowing for the implementation of different graphics modes using advanced methods that in some cases require as much as 40KB of RAM to function.

To proceed with the description of the VIC-II's standard and non-standard graphics modes, note that the VIC-II has a palette of 16 fixed colors and offers 8 independent hardware sprites with a resolution of 24x21 pixels, monochrome or multicolor (in the latter case, the number of "horizontal" pixels is halved but their size is doubled), whose horizontal and vertical dimensions can be doubled. The 8 sprites can also be reused multiple times on the screen (using the multiplexing technique) provided there are no more than 8 on the same raster line.



FLI (20KB): "Screw It Santa! I'm Gonna Be Rudolfant Anyway!" by Carrion (2013)



AH (17KB): "Example 3" by Algorithm, Algotech (2006)





NUFLI (26KB): "Forgotten" by Mufflon prg, Crest/Metalvotze (2010)

Standard Modes

Let's briefly review the standard video modes natively supported by the VIC-II.

SCM (Standard Character Mode): displays 40x25 characters in 2 colors (background and character color), chosen from 256 predefined characters. If the characters from the system ROM are used, the resulting image is called PETSCII.

MCM (Multicolor Character Mode): displays 40x25 characters, either monochrome (2 colors) or multicolor (with 2 additional common colors). In multicolor mode, the horizontal pixels are half as many and twice as wide, and only the first 8 colors of the system palette can be used.

HBM (High-resolution Bitmap Mode): bitmap graphics (i.e., composed of a bitmap representing pixels) at 320x200 pixels with 2 colors (background and foreground) per 8x8-pixel block.

MBM (Multicolor Bitmap Mode): 160x200-pixel multicolor bitmap graphics with 4 colors. The horizontal pixels are twice as large, so the graphic mode is equivalent to the 320x200 bitmap mode with a 2:1 aspect ratio. Two colors are common to all 8x8 blocks, one is independent, and the last is the background color.

ECM (Extended Background Color Mode): displays 40x25 characters in 2 colors. Characters are selected from just



IXFLI (29KB): "Digital Magic demo" by Dane, Crest (2002)

64 predefined characters. The difference from standard character mode is that the background color is chosen from 4 possible options.

Non-mode

Can the VIC-II generate images even with graphics disabled? Of course!

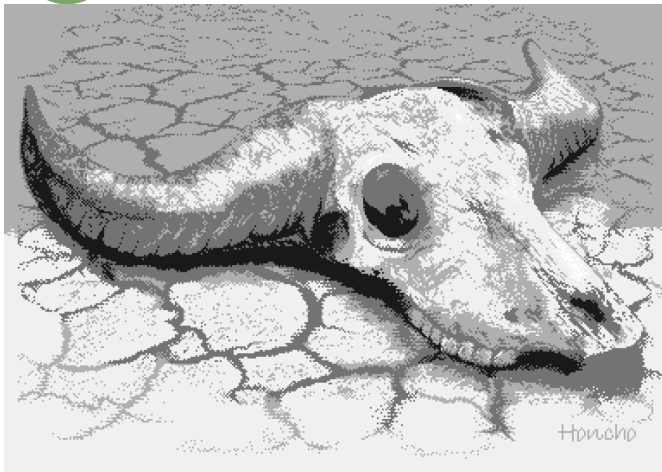
PRS (Permanent Raster Split): With this technique, the border color changes constantly, and because we use a routine synchronized to each individual clock cycle, the image remains stable. We thus obtain a 384x272-pixel image with 32x1-pixel cells, each displaying one of three different colors on the same row.

Edge Opening

The first alternative graphics modes emerged when it was discovered how to "open" the edges—that is, to trick the VIC-II so that it no longer draws the border but continues to display the sprites positioned on the border itself. In fact, the VIC-II can operate at 38x24 cells instead of 40x25, allowing the screen to scroll 8 pixels both horizontally (X-axis) and vertically (Y-axis)—a technique used by all scrolling games.

Opening the top and bottom borders is relatively simple: when the VIC-II checks whether it is time to draw the border, it switches from 25 to 24 lines (and then back again) to trick the video chip and prevent the border-drawing condition from triggering. Of course, this trick must be applied on two specific raster lines (one for the





Full screen tramite HBM+Sprite (14KB): "COP26" by Honcho, Atlantis (2021)



ESCOS (4KB/7KB): "Face" by OPM, Dutch USA-Team (1987)

top border and one for the bottom border).

The same principle can be used to open the side borders: you switch from 40 to 38 columns when the VIC-II is in a specific raster line cycle (and from 38 to 40 before it reaches the opposite side). This means the trick must be applied on every line and for all 200 lines, so a routine synchronized to the single cycle is a must!

The border-opening technique provides us with these alternative methods:

ESCOS (Expanded Screen Construction Set): the entire screen, including the edges, is completely covered (thanks to multiplexing) by horizontally expanded sprites, in multicolor mode, and, depending on the chosen implementation, may or may not be vertically expanded as well. This technique ensures that the graphic map data never appears on the screen, even though it is fully active, thanks to a routine perfectly synchronized to the cycle. The result is an image with a resolution of 384x272 pixels in 4 colors, composed solely of numerous appropriately drawn sprites and with an aspect ratio of 4:1 (or 4:2 if we also expand them vertically).

FLESCOS (Full Screen All Borders Unexpanded Graphics - Flexible Line Sprite COnstruction Set): the entire screen, including the borders, is filled with unexpanded monochrome sprites, at a resolution of 384x272 pixels with 2 colors. What? A maximum of 8 non-expanded sprites horizontally, each 24 pixels wide, provide a total of only 192 pixels—

certainly not the promised 384 pixels! Something doesn't add up... but as illusionists teach us, the trick is there but you can't see it!

In fact, the technique used for this graphic mode involves drawing a row of sprites on the left side of the screen to cover the first 192 pixels, then leaving a "gap" that extends to the next row, where drawing resumes with another 8 sprites after reaching the middle of the screen—so only on the right. The user's eye doesn't notice the empty space on the right and perceives the two rows of sprites as if they were joined together. Simply brilliant!

Full Screen: various graphics modes use the standard bitmap or multicolor mode for the screen, as well as sprites that cover open edges and/or added edges to provide greater detail in certain parts of the main image. Some modes opt for a full screen, while others leave only the side edges open or only the top and bottom edges.

Sprites to Add Colors

Sprites can also be cleverly used to increase the number of colors in an 8x8 block, using them as a background (or even in the foreground) and multiplexing them to cover the entire screen area.

AH (Advanced Hires): horizontally expanded single-color sprites are placed behind the high-definition bitmap graphics. The result is a 320x200-pixel image with 3 colors per 8x8 cell and an aspect ratio of 1:1 (even though the sprites have an aspect ratio of 2:1).

SH (Super Hires): The image size is reduced in high-





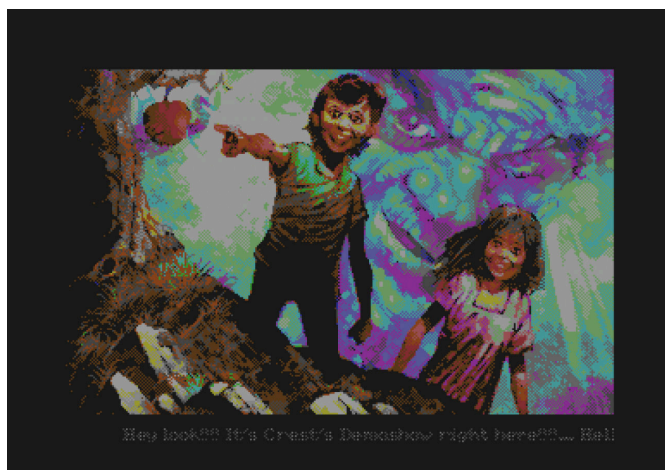
definition mode to just 96x200 pixels, but the 8 sprites are placed two by two, overlapping in 4 columns: this yields 4 colors per 8x8 block (with 2 colors from the sprite covering an area of 24x21 pixels). For ease of implementation, the image is often reduced to 96x167 pixels, thus utilizing only 7 sprite line changes.

MUCSU Hires (MUltiColor Sprite Underlay Hires): in this mode, several horizontally expanded multicolor sprites are used beneath a high-resolution image (in practice, this is the multicolor equivalent of the AH mode seen previously).

More Colors per Block

One of the most important innovations is the technique called Flexible Line Interpretation (FLI). The VIC-II reads graphic data one line every 8, but instead of inhibiting the read as in ESCOS mode, it is forced on every raster line (again via a routine that operates precisely on a single clock cycle). The result is the ability to vary colors independently line by line, since the VIC-II will have new data to retrieve because it reads it on every line.

In this mode, the CPU has little time left (because it is blocked by the VIC-II, which is busy reading data), and the timing is so tight that the first 24 pixels are not obtained from the graphic data read as usual, but from values that are always \$FF. This results in the display of, for example, a fixed color, but this can be remedied by hiding it, possibly by covering it with sprites that have the same color as the background.



ASSLACE (23KB): "Don't meet Crest"
by Crest (2004)

The FLI technique allows for the use of the following modes:

FLI (Flexible Line Interpretation): the basic technique is used with multicolor bitmap graphics and yields an area of 296x200 pixels with the ability to vary the two multicolor shades for each line; therefore, all 16 colors can potentially be used within the 8x8 block.

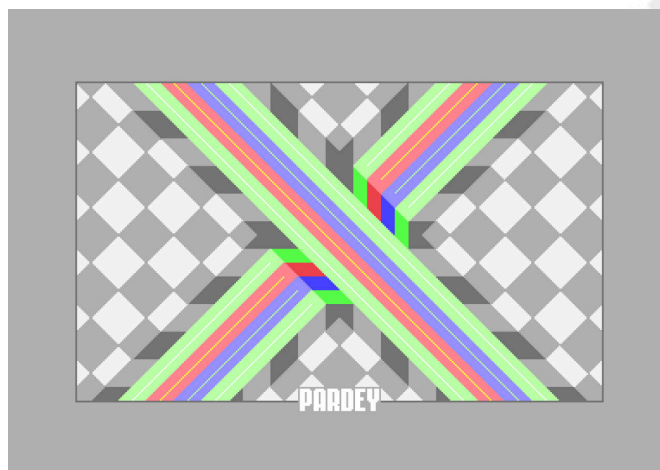
AFLI (Advanced Flexible Line Interpretation): This uses standard bitmap graphics, allowing both the background and foreground colors to be varied on each row of the block, again potentially displaying 16 colors simultaneously.

UFLI (Underlayed Flexible Line Interpretation): This technique uses a layer of sprites as a background to add even more colors to FLI mode (thus producing an image in AH mode but using the FLI technique).

NUFLI (New Underlayed Flexible Line Interpretation): the use of this technique further extends UFLI mode by employing sprites that compensate for the FLI bug area, bringing the effective resolution to 320x200.

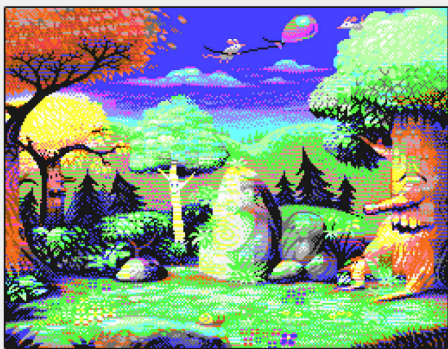
XFLI (eXtended Flexible Line Interpretation): uses a layer of 8 multiplexed sprites without targeted expansion, thus capable of generating images with a reduced resolution of 192x168 pixels.

MUFLI (Multicolor Underlayed Flexible Line Interpretation): in this mode, multicolor sprites are used as an underlying layer for an FLI image (essentially the multicolor sprite version of UFLI).



ECM (2,4KB): "Xpardey"
by 4gentE (2024)





AFLI (19KB): "Test Flight" by Mermaid, Genesis Project/Vision (2016)

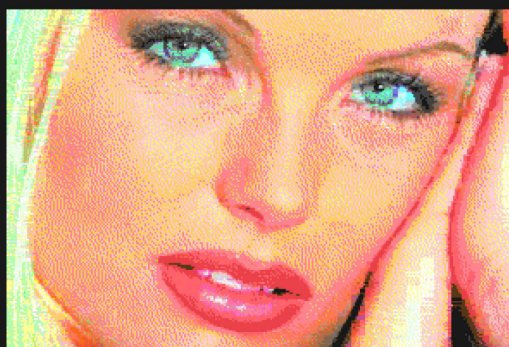
SHFLI (Super Hires Flexible Line Interpretation): the FLI technique is applied to a Super Hires image, thereby increasing the number of usable colors, since the primary color can be changed on every line (the resolution remains limited to 96x167).

SHFXL (Super Hires Flexible Line Interpretation eXtra Large, also known as SOH-FLI): extends the SH mode to 144x168 pixels by using a row of 6 multiplexed sprites underneath. The increased size is achieved at the expense of one color from the removed sprite row.

MUCSU-FLI (MULTiColor Sprite Underlay Flexible Line Interpretation): here, a layer of 5 multiplexed multicolor sprites with double the horizontal resolution (240x160) is used beneath a 272x160-pixel high-definition graphic (the peripheral area is less detailed, so the image must be carefully designed to account for this side effect).

Interlacing

Surely few will remember the interlaced graphics modes that old PC VGA monitors used to increase graphic resolution without increasing the number of pixels. You don't need special hardware to reproduce this technique because you can program the VIC-II to apply the same principle. When we display an image on the TV and, in the next frame, apply a second image that differs (in an appropriate way) from the first, the persistence of the image on the retina creates the perception of a greater number of colors or a higher resolution (although it would be better to refer to this effect as "reduced pixel aspect



IAFLI (27KB): "Hiflint2-diff example" by MixCol Hi-FlI Converter, Algothech (2006)

ratio").

The simplest and most straightforward way to achieve interlacing is to display an image in the first frame and then shift the image one pixel to the right in the second frame. What you get is the perception of as many as 128 theoretical colors compared to the 16 in the palette (though more realistically, you reach a maximum of 70, removing colors that are too similar). If we take a multicolored image, the horizontal pixels are twice as large as the vertical ones, so when shifted by one pixel, half of the multicolored pixels overlap the other half—that is, each pixel ends up on top of the one to its right. Thus, the 2:1 aspect ratio becomes 1:1, with an effective real resolution still of 320x200.

If we apply the same trick in the vertical direction, we get exactly the effect achieved by the interlaced modes of early monitors: if we take a 320x400-pixel image and display the even rows in one frame and the odd rows in the next, the visual blending seems to show us, within a 320x200 area, what was contained in the original image.

Among the modes that use interlacing, we include:

- MCI (Multicolor Interlace): two multicolor bitmap images alternate.
- IH (Interlaced Hires): two standard high-resolution bitmap images alternate.
- IFLI (Interlaced Flexible Line Interpretation): two FLI images alternate.
- UIFLI (Underlaid Interlaced Flexible Line Interpretation):





two UFLI images alternate.

- XIFLI (eXtended Interlaced Flexible Line Interpretation): two XFLI images alternate.

- MUIFLI (Multicolor Underlaid Interlaced Flexible Line Interpretation): two MUFLI images alternate.

- SHIFLI (SuperHires Interlace Flexible Line Interpretation): two SHFLI images alternate.

- SHI (SuperHires Interlace): two SH images alternate, always resulting in a resolution of 96x167 but ideally with a palette of 128 colors (with a maximum of 10 colors per 8x8 block).

- SHIFXL (Super HiRes Interlaced Flexible Line Interpretation eXtra Large): two SHFXL images alternate.

- IAFLI (Interlaced Advanced Flexible Line Interpretation): two AFLI images alternate.

- HMFLI (HiRes Mode Flexible Line Interpretation): one FLI image and one AFLI image alternate. This allows for the perception of high-definition details while maintaining a high color count.

- MEGATEXT: consists of 4 interlaced text-mode images. Here, flickering is much more noticeable than with other methods, both due to the use of text characters alone and because of the overlapping of 4 different images.

- TRIFLI (also known as TRILACE): three FLI images alternate. Although the flicker is potentially greater than in other methods using two images, it can be compensated for quite well with appropriate dithering. The number of colors potentially visible through overlapping reaches 2048!

- MRFLI (Multi-Resolution Flexible Line Interpretation): alternates two images where the even lines are FLI and the odd lines are AFLI, or vice versa.

- ECMI (Extended Background Color Mode Interlaced): used only in demos, alternates two ECM images.

Advanced Interlacing

The problem with interlaced mode is flickering: the human eye perceives the complete image at a frequency of only 25 Hz when using two images, so when there are colors with a significant chromatic difference, the image appears unstable and seems to wobble.

ASSLACE is a method that mitigates this problem: it uses an MCI image overlaid with a screen full of sprites featuring inverted hole patterns that alternate between frames; the overall tone becomes darker due to the color of the overlaid sprites prevailing, but the perceived flickering is significantly reduced!

Special Methods

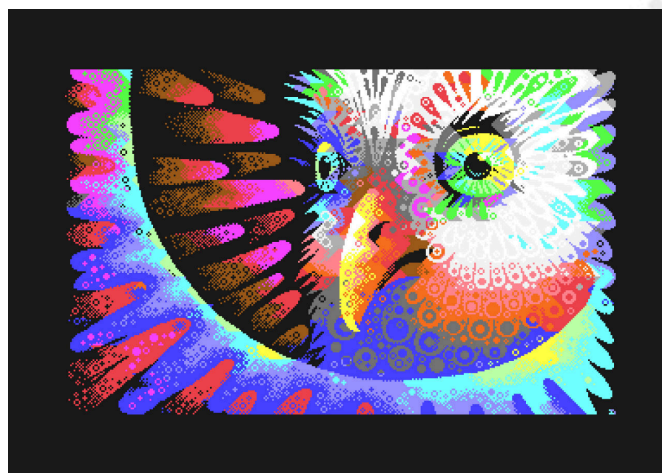
There are still a few special methods that have been used that remain to be classified:

X-SHBM (X-Scroll High Bitmap Mode): the image is a standard bitmap, but a shift is applied to each line in the horizontal (X) direction according to a specific cyclic pattern (thus requiring a routine synchronized to the single machine cycle). The effect is to virtually increase the colors available for each 8x8 block thanks to this phase shift.

QCH (Quarter Char Hires): this is an FLI mode for high-definition bitmaps in which "bad lines" are executed every

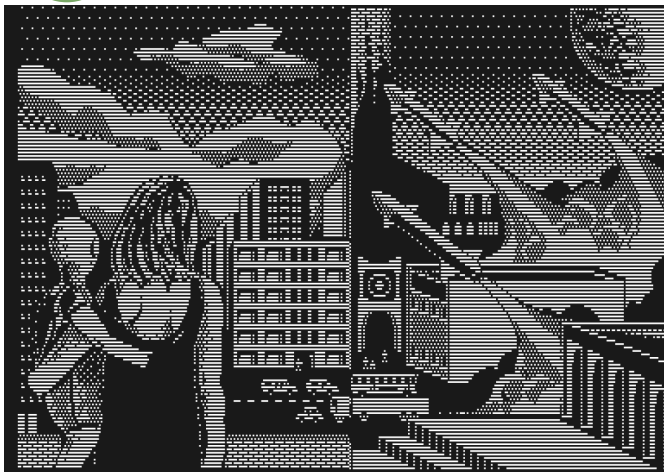


TRIFLI (40KB): "T3 scene 2"
by Algorithm, Algotech (2010)



X-SHBM (9KB): "Huhuu"
by Duce, Extend (2016)





FLESCOS (7KB): "13:37 Example 1" by The Sarge/Soya/Trident, Fairlight (2004)

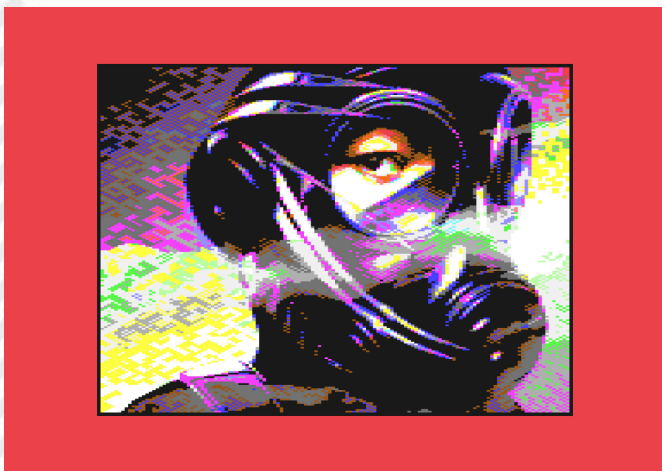
other line, effectively making the colors independent every 8x2 pixels.

HCB (Half Char Bitmap): This is an FLI mode for multicolor bitmaps, in which "bad lines" occur every 4 lines. With the machine cycles saved, it is also possible to vary the color memory, which is normally untouched.

Sprite + Rasterbar: With this technique, you can add more color to an image composed solely of sprites by changing the background color, as was done with the PRS. The routine is therefore a single-cycle operation, and the 6510 processor is halted only while reading the data related to the sprites.

Is that all?

Actually, no, because in practice there are other technical tricks that can result in different formats.



HCB (8KB): "Edge of Disgrace demo" by Booze Design (2008)



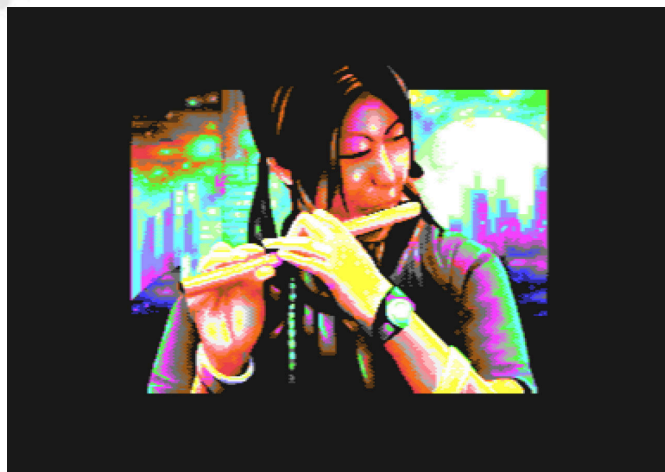
HBM (9KB): "Goblin Towers" by Mermaid, Genesis Project (2016)

For example, the so-called "ghost byte" (\$3FFF) can be used to input graphics and have the VIC-II display them in different areas of the screen, similar to how the value \$FF ends up covering pixels in the FLI bug area. Another example is "gray dots," which occur in certain revisions of the VIC-II chip and produce gray dots when the same color that the VIC-II is reading is written back.

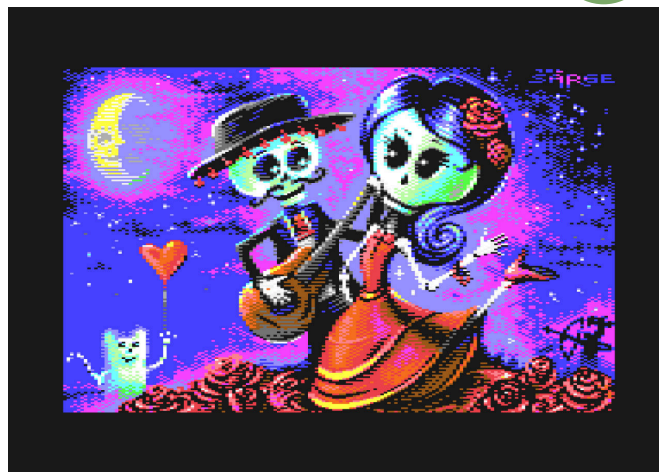
Undoubtedly, there are many other specific methods that have been used in certain demos; the coders' imagination is boundless in this regard, but since they are not (officially) documented, they likely still await discovery and study!

For those interested in exploring all these methods in greater detail, a book titled "Pixels & Cycles: The Art of VIC-II Graphics" is likely to be released in December. It will allow readers to understand and appreciate how the synergy between the CPU and the VIC-II works, enabling the practical implementation of all the mechanisms mentioned above. To date, there are no publications in the literature that focus so specifically on this topic; therefore, the book is designed to be accessible to everyone, so that readers can understand how "the magic" happens. However, it certainly includes the assembly listings of the key points, which are essential for understanding the logic of the cycles in the rasterline. Finally, the advanced user will be able to tinker with the routines for all the methods covered, as all the source code implementing each individual method will be made available.





IFLI (36KB): "Cosmic Fragility" by Jailbird, Booze Design (2006)



MBM (10KB): "Habanero" by The Sarge, Fairlight (2021)



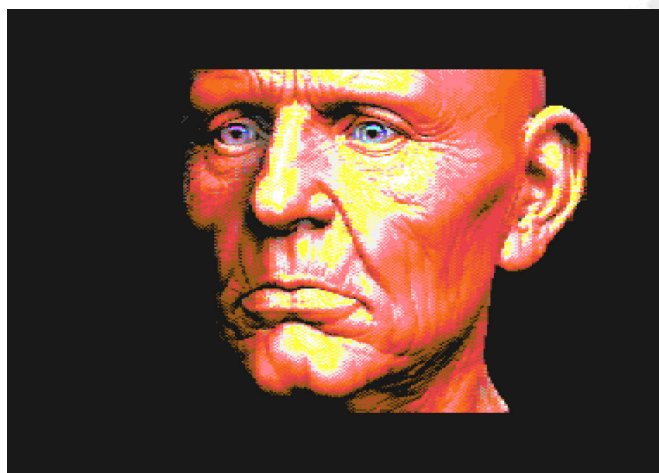
PETSCII (2KB): "Hermit" by redcrab, Genesis Project (2023)



SCM (4KB): "Oh, No! What Logo?!" by jmin (2024)



MCI (19KB): "Sweet Chili O'Mine" by Duce, Extend (2007)



HMFLI (28KB): "HMFLI" by Banshee/Peacemaker, Hitmen (2016)





Building game maps

by *Marco Bergomi*

Imagine you want to create a video game with fantasy settings, a rogue-like game, or an RPG—in all these cases, you need to create a game area.

One of the most resource-intensive parts is the construction of fixed screens or game maps; so I came up with a method to save memory and simplify the task, leaving it to our C64 to draw the screen, following simple and specific instructions.

Obviously, this isn't the only way to achieve the same result, but it's the one that best suits my purposes, namely creating a fixed-screen game.

Let's outline the steps for this project:

- 1) define the numeric codes that will be used to compose the screens/maps;
- 2) associate the codes with PETSCII characters;
- 3) define the dimensions of the game screen;
- 4) draw the map using the chosen codes;
- 5) write the code decoding and screen display routine.

Ready? Let's go!

Point 1)

Let's start by choosing the PETSCII codes to use in the map:

0 = space: the empty space where there is nothing; for this character, I've decided to use video code 32 (POKE table), with a black color (color table = 0);

1 = grass: you could think of a green square to represent the lawn, code 160, color green (13);

2 = water: same as for grass but in light blue or blue, code 102, color (14);

3 = walls: this could be a section of wall, made of stone or brick, code 104, color gray (15);

4 = a key, or a dagger—in short, an object: code 30, while for the color we can choose 10, light red;

We could continue like this for 256 elements (maximum), i.e., the entire character map, perhaps redrawing all the characters to achieve a better on-screen effect.

Point 2)

We have already decided which characters from the map will be used to draw the screen/map; let's summarize them:

```
000 space   code 32   color 0   Black
001 grass   code 160  color 13  light green
002 water   code 102  color 14  light blue
003 walls   code 104  color 15  light gray
```

```
004 object  code 30   color 10  light red
etc...
```

Once these identification codes are defined, we can start drawing the first screen using the codes defined above.

Step 3)

For simplicity, we will draw a 10x10 character screen/map to work quickly, but of course you can use the entire 40x25 character area or go beyond that by using as much memory as you deem necessary for the purpose.

	Column									
Row	0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0

As we mentioned, we will create a small 10x10 screen that we will store in memory starting at address 49152, where we have 4 KB of free RAM ready for this purpose.

Step 4)

Now let's proceed with the actual drawing of the screen:





	Column									
Row	0	1	2	3	4	5	6	7	8	9
	1	1	1	1	1	1	1	0	0	0
	1	1	1	1	0	0	1	1	1	1
	1	1	1	0	0	0	0	0	0	0
	1	0	0	3	0	3	0	0	0	0
	1	0	0	3	4	3	0	0	0	0
	0	0	3	3	3	3	3	0	0	0
	0	0	0	0	3	0	0	0	0	0
	0	0	0	0	0	2	2	2	2	2
	0	2	2	2	2	2	2	2	2	2
	2	2	2	2	2	2	2	2	2	2

This should be the final result:



A simple map to keep the goal in sight: the routines.

Here is a solution to save the map you just drew to memory:

```
1 CLR:PRINTCHR$(147)"READ MAPS"
5 M=49152:M1=99
10 FORI=MTOM+M1:READX:POKEI,X:NEXT
20 END
100 DATA 1,1,1,1,1,1,1,0,0,0
101 DATA 1,1,1,1,0,0,1,1,1,1
102 DATA 1,1,1,0,0,0,0,0,0,0
103 DATA 1,0,0,3,0,3,0,0,0,0
104 DATA 1,0,0,3,4,3,0,0,0,0
105 DATA 0,0,3,3,3,3,3,0,0,0
106 DATA 0,0,0,0,3,0,0,0,0,0
107 DATA 0,0,0,0,0,2,2,2,2,2
108 DATA 0,2,2,2,2,2,2,2,2,2
109 DATA 2,2,2,2,2,2,2,2,2,2
```

The only two lines worth noting are lines 5 and 10. The first sets two variables: M=49152 as the initial memory

address, and M1=99 (100-1=99, starting from zero); line 10 is a FOR loop to load the values in DATA and store them starting at address 49152.

Now that we have loaded our first map into memory, let's see how to draw it on the screen: the small program below reads the data, converts it into character and color code, then saves it to video and color memory for display:

```
1 clr:poke53280,0:poke53281,0:printchr$(147)
2 mv=1024:mc=55296:i=0:t=99:ri=0
3 sc=49152:jp=0:nc=5
5 dim og(4,1)
10 for i=0tonc-1
12 read og(i,0), og(i,1)
13 next
99 rem disegna mappa
100 sc=sc+jp*t
101 fori=0tot
102 pokemv+ri,og(peek(sc+i),0)
103 pokemc+ri,og(peek(sc+i),1)
104 ri=ri+1
105 if ri=10 then110
106 next:goto200
110 ri=0:mv=mv+40:mc=mc+40:goto106
200 geta$:ifa$=""then200
205 end
1000 data 32,0,160,13,102,14,104,15,30,10
```

Let's analyze the code:

The first line resets the variables, sets the screen, and sets the border to black. Lines 2, 3, and 5 are used to declare the variables and assign them the appropriate values: mv = video memory; this variable must be adjusted based on where we want to place our map;

mc = color memory; like the previous one, the correct position must be calculated for this as well.

In the example, mv=1024 and mc=55296 correspond to the top-left corner of the C64 screen.

Meanwhile, the variable i = 0 is used as a counter; t=99 is the map size (as mentioned earlier, 0-99 = 100); ri = row index; essentially, it holds the position in the current row—in other words, the column where to draw;





sc = 49152 is the data source, where the map is located; jp = 0 is used to calculate the memory position; that is, if we had more than one map, jp would hold the current position. Since we initially have only one map, the value of jp remains 0.

In line 5, a vector og (objects) is declared with dimensions 5x2 (remember that the first element of the vector is zero, so its actual dimensions are 4x1).

Lines 10–13 read the list of characters used in the map; this is a standard FOR loop from 0 to nc-1 (number of characters minus 1). The pairs of values (character code + color) will be stored in the og() array.

In line 100, the starting address of the current map is calculated, starting from the first map: jp = 0

sc = sc + jp*t, which is equivalent to: sc = 49152 + 0*99;

From lines 101 to 106, the loop for reading the data map located at address sc is executed; the value sc+i is used to read memory with the PEEK command. This newly extracted value is used as the row index of the og array in column 0, and this value is used as the argument to be sent to the color memory via POKE at address MV+RI.

The same process is performed on line 103 for the color map: the color code of the character to be displayed on screen is extracted from the og() matrix using the same index sc+i, but in column 1. This parameter is passed to the color memory via POKE.

Next come the increments of the position in the video and color maps; at position 105, a check is performed to ensure that the end of the map row has not been reached (in this example, 10 characters); if so, the program jumps to line 110, where the column is reset to zero and the row is incremented by 40.

The same loop is executed for all row numbers; once the end of the map is reached, the program terminates and waits for a key press.

Although the program is very simple, it works satisfactorily; however, since it is written in BASIC, it has its “issues” related to slowness.

Let's see how we could implement the same little program, but using assembly:

```

*=49152                                ;program start

lda #147                                ;clear the screen
jsr $f1ca                                ;

lda #0                                  ;set border and screen
sta 53280                                ;to black
sta 53281                                ;

jsr reset                                ;resets all variables
jsr setmap                               ;sets the map pointers

loop1

jsr readmap                             ;start reading the
first byte of the map
jsr print                                ;procedure to print
the read character

inc index                                ;increment the
index of the i-th character read

lda index                                ;compare with
the end of the table
cmp table                                ;
bne loop1                                ;If not equal,
return to loop1

jsr presskey                             ;Wait for a key press
rts                                     ;Exit the program

;=====
reset

lda #0                                  ;Resets the variables
sta $fa                                  ;
sta $fb                                  ;
sta $fc                                  ;
sta $fd                                  ;
sta $fe                                  ;
sta $ff                                  ;
sta pos                                  ;
sta index                                ;

lda #10                                  ;Loads 10 as the
maximum column width
sta colon                                ;

lda #2                                  ;Sets the game
level to zero - step 2
sta liv                                  ;

```





```

        rts                ;Returns to the caller        lda #0                ;set 0 in pos
;=====
                                (starting position)
                                sta pos                ;
presskey                                clc                ;Add to current
        jsr $ffe4                ;$GETIN kernel routine position
        cmp #$00                ;compare with 0        lda $fa                ;40 characters
        beq presskey            ;If equal to 0, then no (40 video columns)
key was pressed                                adc #40                ;to start a new line
        rts                ;Otherwise, jump        sta $fa                ;
to presskey                                lda $fb                ;
;=====
readmap                                adc #0                ;
        ldy index                ;Set the map read        sta $fb                ;
index position                                clc                ;Same procedure
                                for color memory
        lda ($fe),y                ;Read the first        lda $fc                ;
character                                adc #40                ;40 columns
        tax                ;Transfer it to X        sta $fc                ;
        lda defobj,x                ;Retrieve the Xth        lda $fd                ;
character from the DEFOBJ table            adc #0                ;
        sta objj                ;save it to OBJJ        sta $fd                ;
        lda defcol,x                ;fetch the Xth        rts
character                                ;=====
        sta objc                ;save it to OBJC setmap
        rts                                lda video                ;Set the pointers for
;=====
                                sta $fa                ;page zero, to manage
print                                lda video+1                ;video memory and
        lda objj                ;load the drawing code        sta $fb                ;color memory.
        ldy pos                ;read the position        lda color                ;
        sta ($fa),y                ;set in video memory        sta $fc                ;
        lda objc                ;load the color code        lda color+1                ;
        sta ($fc),y                ;set in color memory        sta $fd                ;
        iny                ;increment        ldx liv                ;Set the pointer
position index                                lda addmap,x                ;to the maps.
        sty pos                ;save the current        sta $fe                ;
position to the POS variable            inx                ;
        cpy colon                ;compare with        lda addmap,x                ;
COLON (screen limit)            sta $ff                ;
        beq nuriga                ;if limit reached,        rts
jump to new line                                ;=====
        rts
;=====
nuriga                                ;Variable definitions
                                video !word 1024

```





```

color    !word 55296
objj     !byte 0
objc     !byte 0
index    !byte 0
table    !byte 100
colon    !byte 10
pos      !byte 0
liv      !byte 0
defobj   !byte 32,160,102,104,30
defcol   !byte 0,13,14,15,10
addmap   !word map1,map2

map1     !byte 1,1,1,1,1,1,1,0,0,0
          !byte 1,1,1,1,0,0,1,1,1,1
          !byte 1,1,1,0,0,0,0,0,0,0
          !byte 1,0,0,3,0,3,0,0,0,0
          !byte 1,0,0,3,4,3,0,0,0,0
          !byte 0,0,3,3,3,3,3,0,0,0
          !byte 0,0,0,0,3,0,0,0,0,0
          !byte 0,0,0,0,0,2,2,2,2,2
          !byte 0,2,2,2,2,2,2,2,2,2
          !byte 2,2,2,2,2,2,2,2,2,2

map2     !byte 1,1,1,1,1,1,1,1,1,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,0,0,0,0,0,0,0,0,1
          !byte 1,1,1,1,1,1,1,1,1,1

```

I've commented out the lines of code to explain how the program works, which can be summarized as follows:

- read the first byte of the map;
- print the decoded character according to the predefined pattern;
- increment the index used to read the table;
- if we haven't reached the end of the table, repeat the

cycle from the beginning.

When run, this program prints the entire map in almost no time; excellent if we want to give our game a professional look. But we all know that using assembly isn't that simple—errors are always lurking; it's not a very flexible language, so we can compromise by using a compiled language, thereby having a development tool that's easier to use yet very powerful; I'm talking about Visual Basic version 1.1. Let's take a look at the code right away:

```

1 clr:cls:colors 1,0,0
5   mmv=1024:mmc=55296:scr=0:liv=0:oc=7:i=0:zz=0
6 x=0:xx=0:yy=0:oo=0:ogl=0:ocl=0:table=100
10 dim ogx(4),clx(4)
11 dim livscr(4)
35 (=====)
36 (decoder definition)
40 i=0
41 read ogx(i),clx(i):[inc i]:do41,5
43 (=====)
44 (set map addresses)
45 livscr(0)=49152
48 (=====)
51 (read first screen)
52 zz=livscr(liv)-1:x=0
55 add xx=x+mmv:add yy=x+mmc:add zz=zz+1
57 oo=peek(zz)
60 ogl=ogx(oo):ocl=clx(oo)
62 pokexx,ogl:pokeyy,ocl
64 [inc x]
65 if x=10 then x=0:add mmv=mmv+40:add mmc=mmc+40
66 do55,table
70 keypress:end
75 (=====)
1000 data32,0,160,13,102,14,104,15,30,10

```

The listing includes very few Vision Basic commands to maintain compatibility with Basic V2; let's look at the details:

- 1) Reset the variables, clear the screen, and set the border and screen color to black, and the text color to white.
- 5-6-10-11) Set the variables, the object and color vectors (unfortunately, Vision Basic does not support arrays),





and the number of table elements.

41) Loop to read character and color codes to display the map on screen.

45) I have only entered the starting address of the first map to be displayed; additional screens must be added to the LIVSCR vector.

52-55) Functions for calculating the video map (XX) and color (YY) addresses; the variable ZZ contains the read address of the map to be decoded on screen.

57) Read a byte from the map to be decoded

60) Extract the character and its color based on the code read from the map.

62) Write the decoded values to video and color memory

64) Increment the index that tracks how many bytes have been read from the map to be decoded.

65) Check that the end of the line has not been reached (which in this example is 10 characters); if so, update the variables (mmv) and (mmc) by jumping to the next line.

66) Repeat the loop a number of times equal to the map's size (10x10 = 100 elements)

70) Wait for a key press, then terminate the program

Comparing these three versions, what stands out most is the program's execution speed; the worst performer is classic Basic; next is the Visual Basic version, which is at least six times faster than its V2 counterpart; in first place is the assembly version: this runs almost instantly, but it is also the most complex and time-consuming to implement.

For informational purposes, there is also the option to use compiled Basic, achieving at least a 4-fold performance increase compared to the interpreted version.

It's up to us to decide how to implement this program, so the choice is yours—happy coding.

Allow me to spend a few words of thanks to Marco Bergomi for agreeing to my request to write an article about his work on game maps. Marco is the administrator of the MOS-Commando Facebook group, where he often posts interesting content, games, and reading recommendations. If you're not familiar with it, check it out, it's worth it.

Francesco F.

```
DATA 1,1,1,1,1,1,1,0,0,0
DATA 1,1,1,1,0,0,1,1,1,1
DATA 1,1,1,0,0,0,0,0,0,0
DATA 1,0,0,3,0,3,0,0,0,0
DATA 1,0,0,3,4,3,0,0,0,0
DATA 0,0,3,3,3,3,3,0,0,0
DATA 0,0,0,0,3,0,0,0,0,0
DATA 0,0,0,0,0,2,2,2,2,2
DATA 0,2,2,2,2,2,2,2,2,2
DATA 2,2,2,2,2,2,2,2,2,2
```

<u>Pos.0</u>	<u>Pos.1</u>	<u>Pos.2</u>	<u>Pos.3</u>	<u>Pos.4</u>	
32	160	102	104	30	Codice video
0	13	14	15	10	Codice colore

Fig. 1 - With the value 3 read from the map, the code 104 with color 15 is placed in video memory





Digital preservation & code enhancing

by David “Cercamon” La Monaca

One of the initiatives that RetroMagazine World has been pursuing for some time now is the project known as RetroLiPS. This project aims to recover, digitize, and preserve from oblivion software written for retro machines, particularly all those programs that, since the early 1980s, appeared in books and magazines in the form of listings written mainly in BASIC — the language that all computers of the era adopted as the default tool for interacting with technical resources.

Combining the desire to contribute to RetroLiPS with a personal sense of nostalgia — which, I’m sure, all retrocomputing fans have felt at least once — I recently picked up one of the few print magazines still in my retro library. It is *Commodore Computer Club No. 14* from October 1984, an Italian magazine that combined news, technical articles, and programming guides for the machines of the historic American brand, leaving little room for games and commercial reviews. Among Italian kids in the ’80s, CCC served as a valuable reference point and a concrete opportunity to learn about and study the hardware and operating systems of home computers, as well as to delve into methods and techniques for programming them effectively.

Issue 14 can in some ways be remembered as one of the most important in the magazine’s entire history, especially due to the full publication of Danilo Toma’s famous Graphics Routines II (starting on page 40), an extension comprising

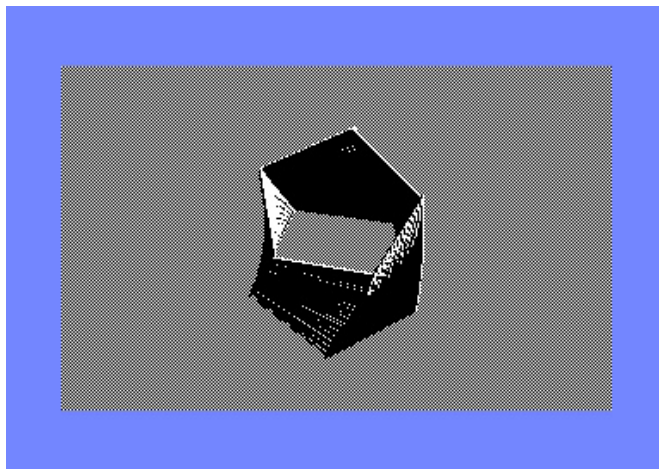


Fig. 2 - Solids - a twisted prism

no fewer than 9 new graphics commands for BASIC v2, which was notoriously lacking in instructions dedicated to managing high-resolution, multicolored bitmap graphics. At the time, Graphics Routines II represented a convenient and efficient solution for many Italian C64 enthusiasts to remedy the shortcomings of BASIC v2.

The first version of Toma’s graphics routines had been published a few months earlier in issue 10 of the same magazine in April 1984. This is a basic, essential version comprising 5 machine-language routines that can be called from BASIC programs using the SYS command. The routines allow you to activate and clear the high-resolution bitmap graphics (320x200 pixels), plot points and draw lines, choose the two available colors, and return to text mode. In just 717 bytes, without encroaching on the memory area reserved for BASIC programs, Graphics

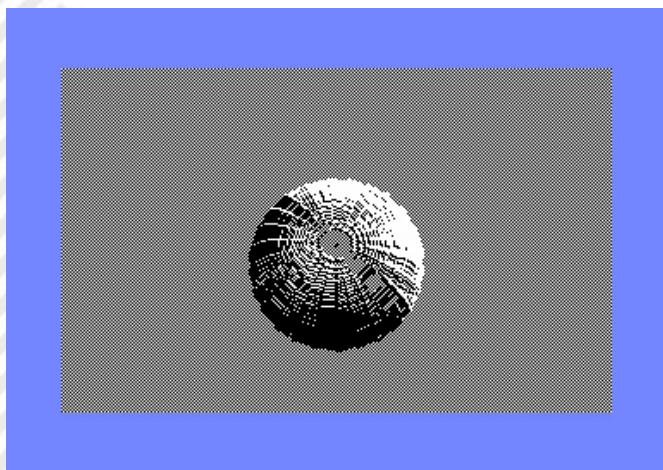


Fig. 1 - Solids - a half-sphere

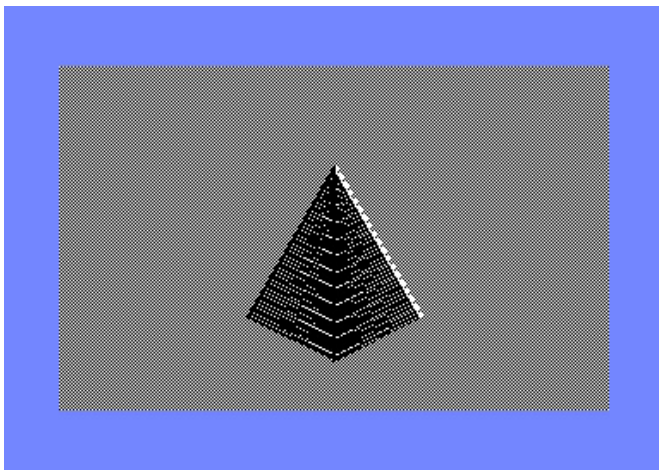


Fig. 3 - Solids - a pyramid





Original Solids Listing - CBM Prg Studio 64 encoding

```

0 rem "O{E*28}P"
1 rem "{cm g} {W*7}s o l i d i{W*7} {cm m}"
2 rem "{cm g} {cm m}"
3 rem "{cm g} version c 64 {cm m}"
4 rem "{cm g} {cm m}"
5 rem "{cm g} by Luca Galizzi {cm m}"
6 rem "{cm g} {cm m}"
7 rem "{cm g} Via Ninfea {cm m}"
8 rem "{cm g} {cm m}"
9 rem "{cm g} Cusano Milanino (MI) {cm m}"
10 rem "{cm g} {cm m}"
11 rem "{cm g} tel: 02/6187588 {cm m}"
12 rem "L{R*28}{sh @}"
13 rem "{cm +*30}"
14 rem "{cm +}{space*28}{cm +}"
15 rem "{cm +} load the gra- routines {cm +}"
16 rem "{cm +} Danilo Toma's chips ccc10{cm +}"
17 rem "{cm +}{space*28}{cm +}"
18 rem "{cm +*30}"
24 rem *** type-in/testing by dlm/rmw ***
29 clr:x%=0:y%=0:x1%=0:y1%=0
30 a=0:b=100:c=1:d=50:e=3:f=.25:g=.5:h=2:gr=atn(1)/45:w=200
40 pl=49693:dr=49728:cl=49763:co=49785:mo=49821
50 rem *** parameter input ***
60 poke248,0:sysmo:print"{clear}{down*3}"
70 print"{right*6}shape : "
75 print"{down}{right}{reverse on}1{reverse off} prism":print"{down}{right}{reverse on}
2{reverse off} pyramid"
77 print"{down}{right}{reverse on}3{reverse off} sphere"
78 print"{down}{right}{reverse on}4{reverse off} hemisphere":print"{down}{right}{reverse
on}5{reverse off} hyperboloid"
80 input"{down*3}which one do you choose ";tp
85 iftp<1orftp>5thenprint"{up*5}":goto80
90 input"{down}how many sides ";lf
100 iflf<1then90
110 input"{down}angle of incidence ";vr
111 if vr < 1 or vr > 90 then print "{up*3}": goto 110
120 input "{down} angle of rotation "; ar
130 input "{down} angle of torsion "; at
140 gosub 1000
150 rem *** drawing the solid ***
160 ar=gr*ar:al=100*sin(vr*gr):y0=100+al/2:x0=160
170 cs=cos(vr*gr):se=8*atn(1)/lf
180 forj=atob:ontpgosub250,260,270,280,290
190 ar=ar+at*gr/b
200 rem *** drawing of the plane figure ***
205 an=8*atn(1)+ar+.001:ry=di*cs
210 x%=c+x0+di*sin(ar):y%=w-(c+y0+ry*cos(ar))
215 fori=artoanstepse
220 y1%=w-(c+y0+ry*cos(i)):x1%=c+x0+di*sin(i)
225 poke2,c:sysdr:x%=x%-c:y%=y%-c:x1%=x1%-c:y1%=y1%-c
230 poke2,a:sysdr:x%=x1%:y%=y1%:nexti
240 y0=y0-al/b:nextj:goto300
250 di=d:return
260 di=d*(c-j/b):return

```





```
270 di=d*sqr(c-(c-j/d)^h):return
280 di=d*sqr(c-(j/d)^h):ifj=dthenj=110
285 return
290 di=d*sqr(f+e*((j/b)-g)^h):return
300 geta$: if a$ = "" then 300
310 goto 14
999 *** background grid ***
1000 poke 248, c: poke h, a: poke 251, c: poke 252, a
1002 sysco: syscl: sysmo
1005 p=57344:h=170:i9=85:i8=1
1010 forx=pto65343step2
1020 pokex,h:pokex+i8,i9:nextx
1035 h=2
1040 return
2003 rem"{E*30}"
2010 rem try entering these values
2020 rem respectively for the
2030 rem figure, sides, incidence,
2040 rem rotation, and torsion.
2050 rem      3      25      65      50      40
2060 rem      2       4      60       0       0
2070 rem      5       5      50       0       0
2080 rem      3      50      45      60      30
2090 rem      5       6      60       0       0
2100 rem"{cm p*30}"
2110 rem *** type/in/testing by dlm/rmw
```

Routines 1 perform their task admirably and with good speed, so well that the editorial staff of "Systems Editoriale" (the publisher of CCC magazine) adopted them from that point on, recommending them to all readers for writing programs capable of exploiting the graphics resources offered by the VIC-II. In issue 14, a surprisingly effective and efficient program appeared for drawing certain solid geometric figures in high resolution, explicitly based on the graphics routines I from issue 10. The article titled "High-Resolution Solids," authored by Luca Galluzzi, can be found on page 14 of the magazine and is accompanied by the complete listing capable of drawing prisms,

pyramids, spheres, hemispheres, and hyperboloids according to certain basic parameters using SYS calls to the aforementioned graphics routines.

One of the interesting aspects of this article is the method used to achieve the goal of creating a pseudo-3D space for visualizing the solid figures. The three-dimensional effect is in fact achieved through shading, which the program calculates for every single line or point of the solid drawn on the screen. Here a technical problem arises, since the C64's high-resolution graphics do not allow the use of more than two colors (one for the background and one for the drawing), but to plot solids effectively, at least three are needed: the background color, the foreground (or drawing) color, and the shading color. To overcome this limitation, the program uses the two base colors (black and white) for drawing and shading and a grid (made of black and white pixels arranged in an 8x8 grid) to create a pattern that is repeated across the entire screen, thus giving the impression of using a third color (gray) for the background. This solution works well: on the pattern of black and white pixels in the background, the two base colors trace the lines of the various geometric shapes provided by the program, resulting in a very

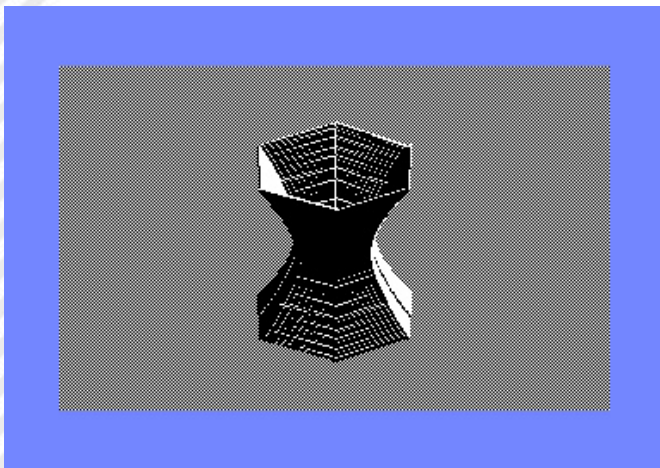


Fig. 4 - Solids - a hyperboloid





Listing of the Hires Bitmap Fill assembly routine

```

* = C2D0                ; start of routine - sys 49872

LDA #$00                ; $E000 start of hires
STA $FD                 ; in low and high bytes
LDA #$E0                ; at locations
STA $FE                 ; $FD and $FE
LDX #$1E                ; outer loop
loop1 LDY #$00           ; $1E=31 - 31*256 = first 7936 bytes
loop2 LDA #$55           ; $55 = 0101 0101 bit pattern
      STA ($FD),Y
      DEY
      ROL                ; from $55 to $AA = 1010 1010
      STA ($FD),Y
      DEY
      BNE loop2
      INC $FE             ; increment high byte
      DEX
      BPL loop1          ; continue the loop
      LDY #$40           ; last 64 bytes (8 blocks of 8x8)
loop3 LDA #$55
      STA ($FD),Y
      DEY
      ROL                ; double $55=85 -> 170=$AA
      STA ($FD),Y
      DEY
      BPL loop3
      RTS

```

pleasing final effect (see Figs. 1–4).

It is precisely on this “third color” and the method by which it fills the background area of the screen’s graphics page that the idea behind our humble contribution is based. Yes, because, as mentioned earlier, the “High-Resolution Solids” program does its job well, but obviously the drawing is relatively slow by today’s standards, even if it was absolutely normal for the time of its release. What is truly very slow is the background filling technique because it is implemented by a FOR/NEXT loop that fills the 8,000 locations of the high-resolution screen with the alternating values \$AA (decimal 170) and \$55 (decimal 85), thus creating a checkerboard of black and white dots. But why those two bytes specifically? Well, just consider the binary representation of the values \$55 and \$170 (Fig. 5). Lines 1005–1020 of the listing (Box 1) show how the FOR/NEXT loop starts at address \$E000 (57344) at the beginning of the high-resolution screen, an address that is modified by the graphics routines I from the default position \$2000 (8192).

After digitizing the listing for the “High-Resolution Solids” program, I decided to make a small modification to the BASIC listing to replace the background-filling loop with a small machine-language routine capable of performing the same task. This allows the screen to be “cleared” or, rather, initialized with the values that form the grid, in a fraction of the time compared to BASIC’s FOR/NEXT loop. The added routine (Box 2), which starts at \$C2D0, following the graphics routines ranging from \$C000 to \$C2CC, is nothing more than a sequence of two loops that traverse the locations from \$E000 to \$F3FF (57344–65343), storing the bytes that form the grid. The first loop covers the first 7936 bytes (31x256), while the second covers the last 64 bytes, always alternating between the values \$55 and \$AA. The high-res screen fill routine uses the page zero locations \$FD and \$FE to point to the start of the memory area \$E000, reserved for the high-res graphics mode screen. It then uses indirect indexed addressing to scan the locations starting from \$E000. As mentioned, the second loop fills the gap of the missing 64 bytes to





Modified BASIC listing - code for CBM Prg Studio

```
10 rem ** solids (routine 1)
12 rem **
14 forzz=49872to49872+41:readvv:pokezz,vv:next
18 data 169,0,133,253,169,224,133,254,162,30,160,0,169,170,145
22 data 253,136,42,145,253,136,208,245,230,254,202,16,238
26 data 160,64,169,170,145,253,136,42,145,253,136,16,245,96
28 rem **
29 clr:x%=0:y%=0:x1%=0:y1%=0
30 a=0:b=100:c=1:d=50:e=3:f=.25:g=.5:h=2:gr=atn(1)/45:w=200
40 pl=49693:dr=49728:cl=49763:co=49785:mo=49821
50 rem *** parameter input ***
60 poke248,0:sysmo:print"{clear}{down*3}"
70 print"{right*6}shape : "
75 print"{down}{right}{reverse on}1{reverse off} prism":print"{down}{right}{reverse on}2{reverse off}
pyramid"
77 print"{down}{right}{reverse on}3{reverse off} sphere"
78 print"{down}{right}{reverse on}4{reverse off} hemisphere":print"{down}{right}{reverse on}5{reverse
off} hyperboloid"
80 input"{down*3}which one do you choose ";tp
85 iftp<1or tp>5thenprint"{up*5}":goto80
90 input"{down}how many sides ";lf
100 iflf<1then90
110 input"{down}angle of incidence ";vr
111 if vr < 1 or vr > 90 then print"{up*3}":goto 110
120 input"{down}angle of rotation ";ar
130 input"{down}angle of torsion ";at
140 gosub 1000
150 rem *** solid drawing ***
160 ar=gr*ar:al=100*sin(vr*gr):y0=100+al/2:x0=160
170 cs=cos(vr*gr):se=8*atn(1)/lf
180 for j = atob: ontp gosub 250, 260, 270, 280, 290
190 ar = ar + at * gr / b
200 rem *** drawing the flat figure ***
205 an = 8 * atn(1) + ar + 0.001: ry = di * cs
210 x%=c+x0+di*sin(ar):y%=w-(c+y0+ry*cos(ar))
215 for i=1 to 250
220 y1%=w-(c+y0+ry*cos(i)):x1%=c+x0+di*sin(i)
225 poke2,c:sysdr:x%=x%-c:y%=y%-c:x1%=x1%-c:y1%=y1%-c
230 poke2,a:sysdr:x%=x1%:y%=y1%:nexti
240 y0=y0-al/b:nextj:goto300
250 di=d:return
260 di=d*(c-j/b):return
270 di=d*sqr(c-(c-j/d)^h):return
280 di=d*sqr(c-(j/d)^h):ifj=dthenj=110
285 return
290 di=d*sqr(f+e*((j/b)-g)^h):return
300 geta$:ifa$=""then300
310 goto28
999 *** background grid ***
1000 poke248,c:pokeh,a:poke251,c:poke252,a
1010 sysco:syscl:sysmo
1020 sys49872:rem ** fill high-res screen
1030 h=2
1040 return
```





reach the end of the high-res area at address 65343.

The final challenge is to convert this into DATA lines to be inserted into the BASIC program to replace the slow FOR/NEXT loop. This problem seems trivial but actually hides, in this specific case, a few pitfalls. It is not efficient to replace the original fill loop with another loop in the same location to read and store the values of the DATA lines: the routine would be loaded into memory every time the program restarts. Therefore, it is better to place the DATA read/write loop for the fill routine into RAM before the start of the program at line 29. However, due to the structure of Toma's I graphics routines, it is not possible to use any permanent variables before the integer variables (such as X%, Y%, etc.) that are used for the coordinates passed to the PLOT and DRAW routines. Therefore, the routine is loaded only once before line 29, so that the routine is stored in memory and remains available for the entire duration of the program's execution. An interesting note: in the code for the background fill routine, we used the ROL instruction (Fig. 6) to change the value of the CPU's accumulator register from \$55 to \$AA, that is, instead of LDA #\$AA. By shifting the accumulator bits to the left (ROL stands for ROTate Left) (\$55 = 0101 0101), we obtain exactly the value \$AA = 1010 1010 (bit 0 is filled with a 0, coming from the carry bit). It's not a huge byte savings (only 2), especially in a routine that is already short and compact, but those who develop in assembly are accustomed to always optimizing the code wherever possible.

Among the references, you can find a link to download a floppy disk image, solidi-rgt-final.d64, which contains everything needed to run the program in both the original version and the modified version with the bitmap-filling

```
01010101
10101010
01010101
10101010
01010101
10101010
01010101
10101010
```

Fig. 5 - 8x8 bit grid block

routine in machine language. We recommend comparing the two versions and enjoying the solid graphics on the original C64, C64U (even by increasing the CPU frequency), or on an emulator (e.g., VICE, using Warp mode, ALT+W). As this article demonstrates—and this is the true purpose of our contribution—even the simple act of typing and preserving BASIC listings originally published in books and magazines can lead to something interesting and, above all, can provide food for thought and concrete material to study and, if possible, improve.

*Special thanks go to our **Marco Pistorio/RMW** and **Phaze101** for their usual helpfulness.*

References

RetroLiPS – article on the official website:

- <https://www.retromagazine.net/solidi-in-alta-risoluzione-c64-basic-rgt/>

Collection of CCC magazine scans:

- <https://ready64.org/download/categorie.php?cat=51>

solidi-rgt-final.d64 image disk:

- <https://www.retromagazine.net/download/rlips/solidi-rgt-final.d64>

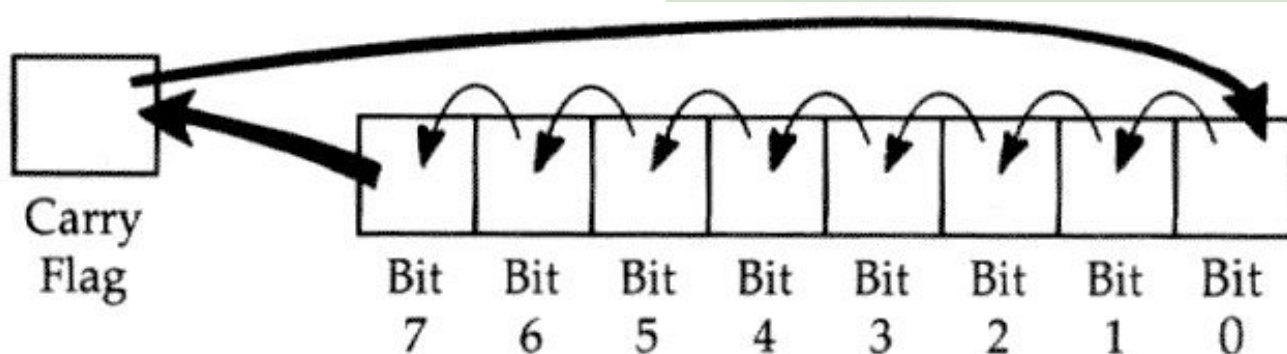


Fig. 6 - ROL instruction





AmigaOS 3.3 – The Evolution of the Operating System

by Giampaolo Moraschi

AmigaOS Classic continues to evolve, and AmigaOS 3.2 marked the first major update in many years, modernizing the system for machines such as the A1200, A3000, and A4000. This version revamped the Workbench, many system tools, preference programs, and command-line utilities, effectively serving as a “rebirth” of the classic system.

A Look at the New Operating System

AmigaOS 3.3 is not just an architectural revolution; it also refines the system by introducing new tools, improving usability, and polishing various internal components. Among the main new features is PartitionEdit, a new ReAction-based disk management tool that replaces HDToolBox, designed to simplify the configuration of CF, SD, and modern disks. The “Moonlight” icon set is also introduced, offering a cleaner and more consistent look,

designed for high resolutions and RTG, though it remains possible to revert to the classic 4-color icons.

The menu system has been updated with “floating” and “sticky” menus, in the style of MagicMenu: it is no longer necessary to hold down the mouse button, and menus can remain open until an option is selected, all of which is configurable in the preferences. The Amiga shell receives improvements for scripting and command-line use, and development leaks reveal a new console called WCON, featuring vertical scrollbars and tabs to support multiple sessions within a single window. Some preference panels, such as Locale and Serial, are rewritten in ReAction to unify the interface and improve usability.

Behind the scenes, much of the work focuses on Intuition, the AmigaOS GUI engine. Features are being imported from newer systems (including AmigaOS 4), such as

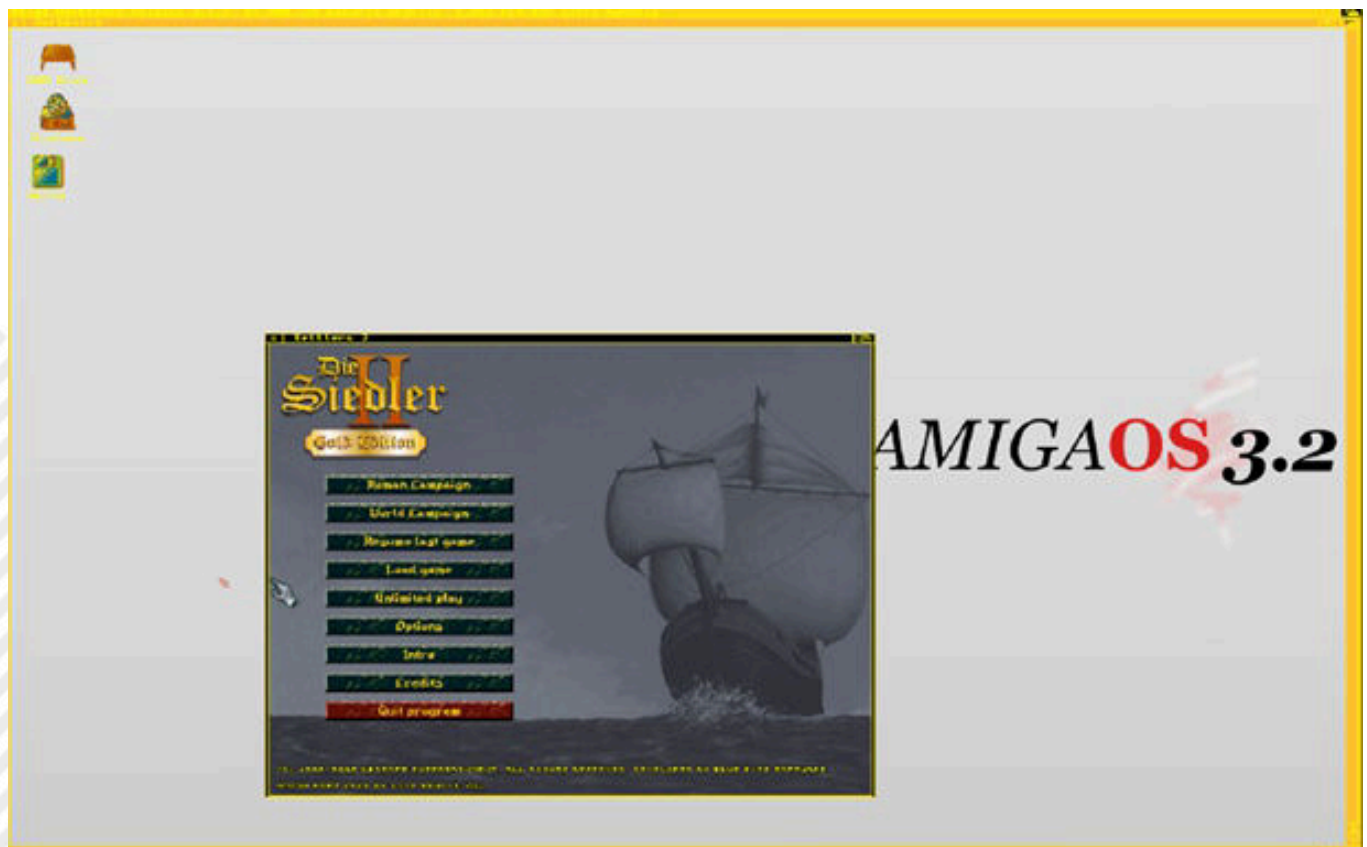


Fig. 1 - AmigaOS 3.2





Fig. 2 - The new AmigaOS 3.3

WA_ToolBox, the GN_EXTENT drawing method, native support for floating/sticky menus, and an infrastructure for context menus. Stability improvements are also being made: for example, the behavior of MakeClass() in calculating object sizes is being corrected to prevent potential overflows, and memory allocations are being improved with flags such as MEMF_NO_EXPUNGE to prevent unwanted expunging of libraries.

There are also minor code optimizations (removal of duplicates, memory savings) and new switches for various CLI commands (AVAIL, UNITS, DACONTROL GET, DISKDOCTOR VERIFY, MAKEDEV ICON), which expand diagnostic and scripting capabilities.

In summary, AmigaOS 3.2 rebuilt and modernized the foundation of the classic system, while AmigaOS 3.3 refines it with new tools, visual updates, improved menus and shell, and a more robust and modern Intuition, all while maintaining full compatibility with legacy hardware and software.

All that remains is to wait for the official release scheduled for 2026, while Hyperion released the digital download version of AmigaOS 3.2 and its upgrades a few weeks ago.

Link - https://secure.2checkout.com/order/product.php?PRODS=53302996&QTY=1&AFFILIATE=252771&__c=1





Interview with Richard Farrell aka Vyper68

20 questions to an electronics technician: an important figure in retrocomputing, the kind of expert we turn to when our beloved vintage systems need repairing

by David “Cercamon” La Monaca

Introduction

Meeting electronics technicians, repair specialists and experts in retro hardware means entering the operational fabric of computing history. Their presence on social media and in specialised communities is steadily growing, mirroring the expanding interest in retrocomputing. These individuals possess a wide range of skills, yet they share a technical heritage that has become increasingly rare for those who study, preserve or simply use vintage systems. The core question is: how can we transmit and document their extensive technical and practical knowledge? Home computers and consoles produced between the 1970s and 1990s continue to function thanks to a fragile ecosystem supported by the experience of those who understand their architectures, components and operational limits. Yet time affects even historically robust hardware: ageing materials, capacitors that lose efficiency, batteries corroding traces, memory chips and TTL logic prone to unpredictable failures. When a malfunction occurs, the question immediately arises: is the repair viable? Spare parts are increasingly difficult to find, costs are rising, and it is not easy to identify professionals or true experts with proper training in technologies that have long ceased production.

In this scenario, specialised repairers play a crucial role. Often working far from the spotlight, they intervene with their wealth of experience and rigorous methodologies: instrumental diagnostics, board analysis, restoration of damaged traces, replacement of now-obsolete components with compatible equivalents. They have become key figures in preserving our computing heritage—custodians of technical and practical knowledge that might otherwise be lost.

For this reason, RMW has introduced a regular feature dedicated to interviews with exactly these experts: professionals and enthusiasts who work behind the scenes of retrocomputing. We have selected 20 questions to highlight the experiences, procedures, challenges and solutions adopted by those who, with competence and method, restore full functionality to the machines that shaped the early digital era.



Fig. 1 - Richard Farrell aka Vyper68

In this interview we meet Vyper68 from UK. For many years now, he has transferred his passion for early home-computing and entertainment systems—together with his knowledge of consumer electronics—into forums and other Internet sites.

1. DLM – Let’s greet and thank Vyper68, who has kindly agreed to this interview. First of all, could you briefly introduce yourself and share a few notes about your personal background with regard to electronics and computers/consoles?

Vyper68 – Hi David, So I was born in Scotland but have spent my adult life in the Northeast England in a small sleepy village called Hurworth. As far as work goes I’m retired now but in the places I did work although I didn’t actually work as an I.T. technician I was often asked to take on that role as I used to build PC’s for people at work and repair older computers and PC’s. So I was asked to look after the networks and the servers at work and





troubleshoot/repair company PC's as an extra duty. I used to enjoy doing that so much I never asked for any extra money! LOL. I had a fairly standard education and I left school with several "O" levels and CSE's including Computer Studies and Physics. I went to college to study Computer science at a higher level as well, but I never went to University as there was pressure on me to start earning money quite early on. So, I went into the workplace instead.

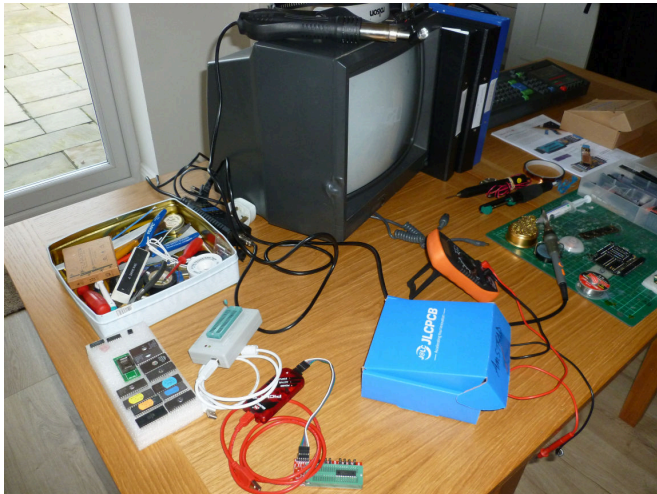


Fig. 2 - Richard's workbench

2. DLM – How did you first become interested in electronics and computers? Which machine or vintage brand do you most enjoy using/expanding/repairing, and why?

Vyper68 – I was always interested in technology as my father was extremely interested in Science and we used to watch a television show called "Tomorrows World" on the BBC together. I was fascinated on how things were made and the internals that made up devices. I can remember my first memory of taking my Atari 2600 apart to see inside and looking at the main PCB and trying to make sense of it. I was only ten at the time so I didn't really understand what I was looking at but I wanted to. Skipping forward a few years technology had made big steps, and it was then that I saw my first microcomputer at school when my teacher was playing "The Hobbit" on a ZX Spectrum and setting up a computer club. I was later able to get a ZX Spectrum for my birthday and it promptly failed as soon as I plugged it in. In hindsight that RAM had failed straight away. So, we had to go back to the store and get another one and that was when I started thinking about needing to be able to repair any computer I owned myself. We started building electronic projects in the computer club and learning to program. The good thing about the club was there were different computers owned by different students. Oric's, C64's, Atari's and Dragons. Luckily, there was a large amount

of magazines and books available by then and I was able to educate myself to a standard I could repair my Spectrum myself which I had to a couple of times and then other people's computers as word spread.

I have several favourites 8-Bit computers but if I really had to choose it would be the Atari range, particularly the 800XL and secondly the Oric-1/Atmos range. The 800XL is such an elegant design and the custom chipset makes for a really nice machine (except for the BASIC) there are several expansions available for the 800XL/65XE including the Ultimate 1MB which massively expands on the memory and features especially if you use FlashJazzCats firmware. The other computer I have a soft spot the Oric is down to the people and the AY sound chip. I was a member of Oric User Monthly and was able to get a Cumana disk interface, so I was able to use the SEDORIC disk operating system which was my go-to machine for several years even though I also had an Atari STFm back then. Some of the modern era Oric games are of a really excellent quality, they put commercial games from the 80's to shame. I am currently beta testing a new ULA for the Oric which gives the option to have a crisp DVI output so there are also additions to the Oric too, though to be fair that is true for most computers from that era. There are modern disk drive replacements and along with tape emulation devices which make it much easier to use now. The other thing I do like is the Dragon/Tandy computer systems, mainly because of the 6809 CPU and I am trying to teach myself 6809 assembler on my PAL CoCo2 just as a brain exercise.

3. DLM – What are the most common challenges you face when restoring and repairing retro hardware? In your experience, what types of faults tend to occur most frequently?

Vyper68 – The most common challenge is sourcing custom ICs for these computers or their peripherals, like the Atari 800XL's ANTIC or POKEY or the Oric's ULA. Don't get me started on Commodore chips! These are long out of production now and the only way to source them is to cannibalise others, which I hate doing. This applies to most computers of that era as most had custom IC's so that is THE biggest challenge. The most common failure I come across these days on the 8-Bit generation is RAM failures. This is not so bad as you can still get NOS 4164's or the other memory IC's you need on the market. The other frequent problem these days are dry solder joints which cause a failure and can be hard to find. I am getting





a lot of Oric-1's to repair with faulty keyboards and they have all been dry solder joints on the keyboard PCB.

4. DLM – Have you ever completed a so-called “mission impossible”? Is there one success story you are particularly proud of? Could you tell us about it?

Vyper68 – That is a difficult one to answer as they always feel a little bit like that to start with. There was an Oric Atmos that springs to mind which had been shorted out. The expansion port is an IDC connector and has bare pins exposed and is right next to the 9V DC input you cannot see these when you have the computer on a desk as they face away from you so if you are not careful you can send 9V through the board by touching the DV jack on the pins of the expansion and with parts being 5V only 9V is disastrous. So I had to work my way through it desoldering the logic IC's and testing them, then the main ICs needed removing and testing. I had to rebuild it from scratch. The PAL UHF circuit has a PROM which is impossible to get now but thankfully you do not need the UHF circuit as most people use the RGB output with a SCART lead. So, by the end I had a working Oric minus the UHF so I was quite pleased with that outcome overall.

5. DLM – How do you source spare parts or original pieces? How difficult is it to find them 30 or 40 years after production?

Vyper68 – Most RAM comes from some trusted suppliers I have found over the years. There are a few small electronic stores in the UK that sell Logic, RAM and CPUs. There is also eBay as well if you are totally stuck, although I do have a couple of sellers on there I trust as well. I tend to avoid Chinese sites as there is a problem with fake or rebadged chips these days. Things like Disk controller IC's, CPU's or PIA's and AY sound chips are examples I have had issues with. As I mentioned earlier the hard part is getting custom ICs, someone has found new old stock, they have broken down a machine and are selling parts individually or you buy a donor machine yourself. You can in some cases get modern replacement parts like the POKEYMAX or the ORICULA but sadly the situation is only going to get worse as time moves on. The progress of making replacement custom ICs with modern FPGA equivalents is a godsend really and it's something that was not really available a few years ago.

6. DLM – What role does the retrocomputing community play in your work? Does it help to keep your passion



Fig. 3 - Fixing an Amstrad CPC

alive? Are you able to attend and comment personally and frequently to posts and questions on forums or social media?

Vyper68 – Retrocomputing and repairs is a large part of my life now I no longer work and the interaction I get with other people is especially important to me. I have made some great friends over the years through this hobby and. I have often been asked for help on forums and try to help as much as I can, conversely, I can asked for help myself as I don't pretend I have all the answers for all machines. So it's a two-way process. I tend to avoid social media these days so it mainly forums and machine specific websites I use but I enjoy doing that a lot.

7. DLM – In your opinion, how important is it to preserve not only the devices themselves, but also the software and documentation associated with the hardware (i.e. technical schematics, circuit designs, component tables and specifications)?

Vyper68 – I think it's vital, the early development of microcomputers in the late 70's early 80's was a massively important point in technology and culture. People were able to have in their home something that had been up to that point out of their reach. The boom of game development, particularly in the UK and I suspect most of Europe can be linked to the home computer boom in the early 80's. I know technology moves on and these devices became outdated but they are part of history and need preserving, especially working examples. The same applies to documentation like manuals, schematics as well as magazines with articles and user made software. We have already lost a lot of this to time which is a shame. We do have a lot of people out there trying to keep these machines working and designing new software for them so that's good.





8. DLM – In recent years, have you noticed a renewed interest in retrocomputing/retrogaming among younger generations and beyond? Who typically approaches you for technical advice or repairs?

Vyper68 – I think there is a lot of interest in retro gaming these days. There was always a hard core group that were invested in their particular formats but people have more disposable income and have started to buy the old computers they used to own partly down to nostalgia I suspect. The sad side effect is that supply and demand has made prices increase steeply over the last few years. With the mini consoles and emulation, it has allowed people to see what all the fuss was about as well. You only have to look on the internet and YouTube to see this. As I said I think retro computing tends to be people getting back into the machines they had as children or they never stopped and have used them all the way through, however, I do think the gaming side is more prevalent though as most people who had a home computer back then spent a lot of time playing games on it despite what you told your parents.

One of the things I get these days is I bought this computer on eBay and it doesn't work help! That tends to be new users who have bought an "untested" machine off eBay which equates to plugged in and didn't work so listed as "untested". Regulars already know what they are in for but new users can feel a bit annoyed about it but regulars still also come to me for repairs.

9. DLM – What is your opinion on software emulation and FPGA-based reproductions of old systems? Do they support or hinder the preservation of original hardware?

Vyper68 – Aside from the piracy argument of emulation I still think it is in some ways a good thing. Some of these old systems are quite scarce and therefore very expensive now, putting them out of the reach of most people. So emulation enables people to experience these rarer systems and see what all the fuss was about. I think for an historical perspective it is important to preserve old software as well. There has already a lot of old software as listed as missing in action on archive websites which is sad. Emulation and modern FPGA remakes also enables people to save wear and tear on their original hardware as well so that's a plus as well, although running software on real hardware has a definite charm which these modern systems lack in my opinion. FPGA reproductions of old systems like the C64 mini or Atari mini fall into that category I guess, they are not the same as using a real

64 or 800XL but they are not supposed to be I guess. So I suppose the short answer is I support the idea.

10. DLM – If you could restore one legendary item—perhaps for a retrocomputing museum or even for yourself—which would you choose and why?

Vyper68 – I would like to work on a Computers Lynx or a SAM Coupe computer or an Atari Falcon030. I owned a Computers Lynx and a SAM back in the 90's but sadly we were burgled and a lot of my old computer kit was stolen and those were among the items taken. I would like to have worked on the Falcon030 as it was Atari's last computer and it sounded like a fantastic machine. I have used Hatari to emulate one and it seems like a real beast. The prices for all three machines these days are stupidly expensive so I don't think I will ever own one again or in the case of the Falcon at all but as I said there is always emulation, Simcoupe is a great SAM emulator and Jynx is a Java based emulator of the Computers Lynx.

11. DLM – Which basic hardware and software tools do you normally use to diagnose faults in vintage systems? What is your primary method of intervention?

Vyper68 – So I have the usual kit you would expect, Soldering Iron, Desolder pump/braid, Desolder gun and I am hopefully getting a Hot Air station so I can work on more modern systems and consoles. I have a multimeter, logic probe and a small oscilloscope.

My main thing is multimeter to start. I always do the same thing my mantra is ASAP (Always Start At the Power) so that's my main starting point. Normally at a lower voltage with a modern power supply. Once passed the power regulator I may bypass that altogether and inject 5V directly onto the board. Once you have 5V on all the chips then you can get a look at the display which also gives you an idea as to what might be wrong and I also give the RAM a quick touch as well. If a RAM chip has failed it will probably be incredibly hot. So they are how I start off.

12. DLM – When working on a historically significant or valuable motherboard or PCB, do you prefer to preserve it as it is, or do you perform modifications/adaptations in order to fix it?

Vyper68 – I always try to preserve them as original as much as possible when repairing them but there are service modifications that were issued by the manufacturer too. I will always fit them unless asked not to. For example,





the Oric-1 has a fault where the ROM will eventually destroy the AY sound chip (according to Tangerine Computers) and they released a service bulletin to fix this. I always fit this if I find an Oric-1 without it. So some of these are important modifications released by the manufacturer and to my eyes still keep the machine original and also better. Homemade modifications people designed and put online I don't bother with for customer machines.

I might modify one of my old computers like my VIC-20 to support S-Video for a better picture. Personally, I like to have one stock machine and one that I can modify so I can enjoy both worlds. It's not always possible to do this so I do have some that are modified but these are my own and tend to involve improving the video output to help connect them to modern TV's, I would not do that for someone else unless they specifically asked for it as some people want upgrades fitted as well as repairs and I will basically do what the customer wants (within reason)

13. DLM – How do you deal with oxidation and contact degradation after decades of inactivity? What other faults are the most frequent in your experience for old computers and consoles? What maintenance techniques do you typically recommend to those who receive a machine from you after repair?

Vyper68 – If you have some IC's that have oxidisation on the legs from heat or time, I will use a very mild abrasive to clean the oxidisation off the legs and if the socket is in poor condition I will replace it. I would use contact cleaner for switches and edge connectors. I also use a rubber to clean edge connectors on computers and cartridges; it works surprisingly well.

As far as computer cases I don't tackle discolouration of the plastic as I am not convinced Retrobrighting is good for the plastics in the long run. So if a case is yellowed it stays as it is. My VIC-20 is a case in point. It is very yellowed but I wouldn't Retrobright it.

As I said previously RAM is a very common failure on 8-Bit computers along with dry solder joints, with consoles it can be all sorts of things. They have a lot of custom IC's and ASIC's which can fail. I suppose if you have just bought a machine of eBay start with the PSU and check the voltages are good. If possible use a PSU at a lower (5V) supply to test the voltages on the PCB or inject 5V onto the board and bypass the PSU completely, give it a good clean as well and check for continuity and dry solder joints. Check the RAM to see if any are insanely hot and

if they are remove them and test them replacing as needed.

14. DLM – Have you ever had to recreate or replicate components that are no longer manufactured, or improve original ones with modern designs? If so, how did you approach it? What sources did you use for documentation?

Vyper68 – That is something I have never really gotten involved in, the creation of a modern FPGA replacement for an old IC is not something that interests me and there are smarter people than me out there doing just that. I prefer repairing rather than designing.

15. DLM – Do you ever use FPGA-based hardware emulators or reverse-engineering tools to test obsolete parts? In your opinion, what are the fastest diagnostic tools?

Vyper68 – I don't have anything like FPGA hardware but I do use a RAM tester and a Logic chip tester for some of the diagnostic tools I use and they are very quick.

16. DLM – How do you handle data preservation from fragile magnetic media (floppy disks, cassettes, hard drives, tapes)? Do you select the most suitable method each time depending on the machine or system you are working on?

Vyper68 – I have a few tools to use when trying to dump a disk or tape. It depends on the media. For floppy disks there is hardware you can use (Greaseweasel, SIO2PC, XUM1541) and tape is trying to copy a file using software like Audacity and then cleaning the audio input. I don't do it that much anymore and a lot of people use modern devices that emulate these devices like Fujinet or Pi1541 or the TZXDduino.

17. DLM – What is your opinion on using modern capacitors and electronic components to replace original ones during repairs? More generally, do you think common repairs today are easier/faster than in the era when these systems were originally sold?

Vyper68 – I think replacing capacitors with modern high quality ones is a good idea. These machines were built to a budget to maximise profit. So not all components were the best quality, replacing these with modern quality versions is a given. It helps keep them running for another 40 years.

I think repairs back then were easier in some ways as you could call the Atari or Commodore service centre and buy a POKEY or SID whereas now you can't. On the flip side





there is a lot more information at a persons fingertips with the internet. Modern tools are also better and more readily more available. So it's swings and roundabouts.

18. DLM – Which programming languages or software tools do you use to interface with legacy hardware, especially for firmware updates or ROM/EPROM programming and integrated-circuit handling?

Vyper68 – I don't really use programming languages aside from the normal BASIC and a bit of FORTH as most computers had a BASIC interpreter installed. So you can write a small program to check various things like memory size and system variables. I do dabble in writing 6809 assembler to do the same thing but a lot faster but that is for Dragons and Tand CoCo's only. I have an EPROM programmer for dumping a ROM or burning a new replacement is the old one has failed or there is an updated ROM available and the customer wants one installed.

19. DLM – How do you document your work? Do you take notes, draw schematics, write guides or share code in public repositories or multimedia content? Are you somehow jealous of your techniques, or are you willing to share your knowledge and experience with others?

Vyper68 – I take photos and jot down what I am doing and the results so I have a reference for when I have a similar issue with a machine in the future. I will post my results on forums with pictures and write articles to go with them. I don't really see myself as anyone special just someone who had a interest in computers and hardware so I'm not precious about my processes as I'm sure a lot of people do the same thing. So I quite willingly share what I know. The more the merrier for me.

20. DLM – How much do you genuinely enjoy repairing systems, diagnosing faults, and how satisfying is it when you finally solve an issue that initially seemed insurmountable? And have you ever had to give up because a system had too many faults or the cost simply wasn't worth the effort?

Vyper68 – I really do enjoy doing this now I'm retired and have more time to spend on it. I don't do it as a business, I just like the process. To be able to repair a device is something I get a great deal of satisfaction from and to start out with a totally dead system and end with a working one is very satisfying for me. I only have one that I had real problems with. There was an Oric-1 I managed to get working but it would not load cassettes and no matter

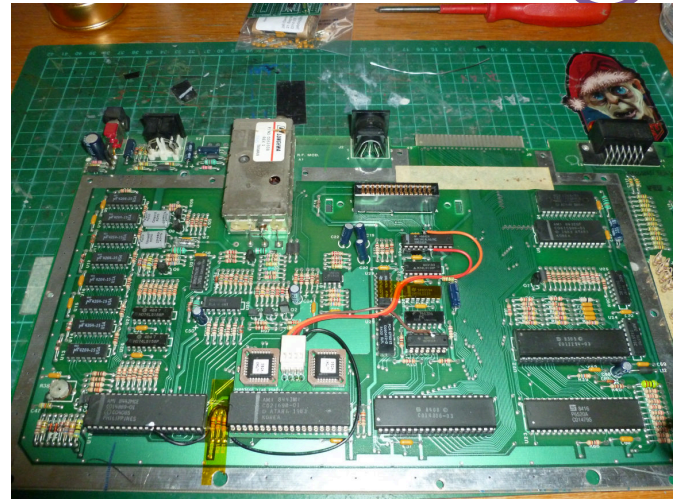


Fig. 4 - Motherboard Atari with 512KB expansion

what I did it would not load a tape and it got to the point where it was not financially worth spending more money on it so it became a Erebus or disc only system. Given that Oric tape loading was so unreliable it was no great loss.

Conclusions

We thank Richard for sharing his point of view and for telling us something about his adventures in retrocomputer repair. In the world of electronic technicians, it is not always easy to find someone willing to share stories, time and expertise — all essential elements for showing the "behind the scenes" of repairs, the everyday challenges and, above all, the value of this craft. It is precisely the passion for our retro computers that drives hobbyists and enthusiasts like Richard to carry on the mission of preserving and bringing back to life those vintage systems that, over the years, have given us so much enjoyment and knowledge. If you too have computers or peripherals from the past at home, drop by the forums and social media channels frequented by Vyper68. You will find tips, documentation, repair stories and an immense wealth of shared experience. But above all, you will become part of a welcoming community where you can exchange messages, deepen your knowledge and engage with Vyper68 himself and many other retrocomputing and electronics enthusiasts.

Forums & Links

SpectrumComputing.co.uk, AtariAge.com, Atari-Owner.com/club, Forum.DefenceForce.org, Archive.WorldofTheDragon.com, StarDot.org.uk, ColorComputerArchive.com, Lemon64, CPCWiki.eu/Forum

FB Groups: [OricOwners](#), [SpectrumForEveryone](#)





To collect or to accumulate?

by Roberto Lanciotti

I've written a personal reflection on collecting—on what I've experienced as a passion for years and now see in a different light: that fine line between preserving and hoarding, between memory and possession. It's a very personal piece, born from a long journey and some realizations that have only recently come to me. If anyone recognizes these dynamics or has experienced something similar, perhaps they'll find food for thought here.



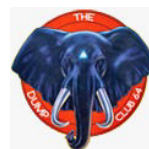
For many years, I believed that collecting was one of the highest forms of passion: a way of caring for memory, a means of preserving the stories of an era that would otherwise be lost.

Only now, looking back, do I realize how thin the line is between the desire to preserve and the need to possess.

What I long called preservation has, at times, become hoarding. And hoarding, when it grows silently over the years, doesn't just take up physical space: it ends up taking time, energy, relationships, and, in some way, a part of ourselves. Collectors don't simply gather things: they gather memories, the material evidence of personal and collective passions.

In this sense, collecting has always had a very profound cultural dimension. It is an act that speaks of taste, of exploration, and of knowledge.

Yet, precisely within this apparent nobility, an ambiguity lies hidden. Personal experience has taught me how fragile the line is between the care of memory and the logic of



Dump Club 64
Preservare e Condividere

hoarding. At first, every new piece in the collection gave me a certain satisfaction: the feeling of having saved something, of having recognized it, chosen it, and brought it into my space.

It was a pleasure that concerned not only the object itself, but what it represented. Over time, however, that satisfaction began to prove fleeting.

The object soon ceased to be a discovery and became a habit. So the focus immediately shifted to the next search, to the next purchase, to the desire to complete, surpass, and possess even more. It was only in recent days, however, that all of this became clear to me.

In the garage where I keep most of my stuff, I noticed some stains on the wall. At first I thought it was mold, but upon closer inspection, it was a whitish substance, typical of moisture. In reality, I had a water leak somewhere, which had seeped down to the level of my shelves. That wall forced me to stop. I had to take down the shelves and clear them of the stuff I'd accumulated over the years: stacked boxes of games, consoles, books, tons of cassettes, accessories, and duplicates of every kind.

And it was precisely at that moment, as I was removing box after box, that I finally put two and two together. I realized I had spent a lifetime collecting. Too much!!

I saw myself in a daily routine that had become almost automatic: every morning, the first thing to do was look for the bargain, the deal, the once-in-a-lifetime discount. I imagine many of my fellow collectors (sorry, I don't mean to offend anyone) will recognize this dynamic. The day often began with the search for the right piece, the bargain lot, the rarity that suddenly appeared at some flea market or in an ad. It had become a reflex. Only now do I realize how much time, energy, and attention this habit has absorbed. Time taken away from family, from those close to me, from the things that really matter. It wasn't wrong to have a passion. What was wrong was the imbalance. I could have done it in a more measured, more frugal, more mindful way.





This is accompanied by an even more painful thought: at times, letting myself be carried away by an ego that ended up taking up too much space, I ended up losing friends as well—people who eventually drifted away. It was never intentional, and perhaps that is precisely why it weighs even heavier on me today.

Without realizing it, I had focused too much on my research and my priorities, ending up devoting less attention to those around me. Many encounters were genuine, while others developed within more ambiguous dynamics: a sort of mutual complacency, fueled by the desire to accumulate and possess.

Sometimes I approached people with a self-serving attitude, but often it was a mutual dynamic because a form of interest emerged from the other side as well. The goal was, in a way, the hope that sooner or later that person might sell, trade, or even give away the missing piece for the collection.

Yet, the original impulse, at least for me, was never merely about possession; what I have always sought to convey through my work was a sense of simple, almost childlike happiness: a form of quiet serenity, made up of inner calm and the little things that make everything feel lighter. I wanted to convey that kind of happiness I remembered having as a child, when it didn't take much to feel good, when simply sharing something was enough to feel at peace with everything else.

The problem is that, at a certain point, preservation and collecting begin to clash. It starts with a noble idea: to preserve, catalog, archive, and transfer content—that is, to prevent technical and cultural memory from being lost. But often, behind this justification, selfishness creeps in and, consequently, hoarding.

That is where the cultural act risks turning into a compulsive habit. Fortunately, however, as time goes by, we grow and come to understand that there is a difference between what is truly useful for preservation and what, on the other hand, should simply be let go. It's fine to have a few spare parts—what's really needed for data migration, data recovery, or media preservation. But everything else, often, isn't memory. It's a burden. Perhaps this is precisely

the most interesting point: collecting, when it degenerates, produces not only material excess but also relational impoverishment. The object, from a symbol of memory, becomes a screen. Today I realized that the real distinction is not between collecting and not collecting, but between giving meaning and seeking compensation. No collection, however precious, can replace the value of time spent with those close to us. And this is precisely what, at some point, I had forgotten: as kids, we shared everything and were happy with very little. We were simply together. If today I feel the need to put all this down on paper, it is also to offer a sincere thought to all those people who, over the years, perhaps did not find in me the person they expected. To those who, for this reason, chose to distance themselves. Today, with greater clarity, I carry with me the regret for what I could have done but didn't, for the attention I failed to give, for the time I didn't spend, for the people who perhaps deserved a better version of me. I could have done more, that's for sure. But life is also made up of this: of choices made at a certain moment, of misplaced priorities, of enthusiasms that carry you far away without you even realizing it. And yet, in the end, I believe I left something good behind. A few pieces saved, a few fragments of memory preserved, a few passions lived to the fullest, even if sometimes in an unbalanced way. But will all of this matter in the end? As that song said, "We'll only find out by living."

The human spirit is strange: it easily gets lost in its own affairs, but fortunately, over time, we also manage to look back and make sense—at least partially—of what has been. And perhaps, if this article has any meaning, it will also be to make someone else reflect on their own journey, on what accumulates, and on what truly matters.

DumpClub64:

- <https://www.dumpclub64.it/>

Link to the #67 Dumping:

- <https://www.dumpclub64.it/dumping/67-dumping/>





Apple turns 50: half a century of “Think Different”

by Carlo Nithaiah Del Mar Pirazzini

On April 1, 1976, in a garage in Los Altos, one of the most influential stories in modern computing began. Fifty years later, Apple celebrates its anniversary not only with staggering figures—nearly \$4 trillion in market capitalization, billions of devices in circulation—but with an uncomfortable question: what does “Think Different” really mean in 2026?

From the “bare” circuit board to the future

In 1976, the Apple I was, literally, a bare circuit board. No case, no keyboard, no monitor—just a circuit board to slip into a box and connect to a TV. Wozniak designed it, with his minimalist and ingenious approach; but it was Jobs who realized that this board could be sold, packaged, and transformed from a curiosity for hackers into a product. The Apple II marked the turning point! A plastic case, built-in keyboard, colors, floppy drive. It was one of the first computers you could imagine on a non-corporate desk, and with VisiCalc, it suddenly became an office tool as coveted as a high-end calculator. Even here, we can glimpse the traits Apple would never abandon: care for the object, obsessive attention to the user experience, and the conviction that there must be a story behind the numbers.

In 1984, the Macintosh raised the bar: a graphical interface,



The logo celebrating the “first” 50 years

a mouse, and typography worthy of a “real” typesetter. The iconic “1984” commercial didn’t just sell a product, but an idea of rebellion against the IT standardization dominated by IBM. Even when profits began to falter and projects failed to take off, that identity—distinct from the rest of the industry—remained etched in the public consciousness.

Setbacks, comebacks, and the miracle of a second chance

After Jobs’ departure in the mid-1980s, Apple entered what many call “the lost decade.” Too many Macintosh lines, confusing strategies, and the rise of Windows destabilized the company and its loyal customers. Financials worsened, market share plummeted, and by the mid-1990s, the company was teetering on the brink of



Apple I: boards assembled in various homemade cases





The Success of the Apple II

bankruptcy (and came dangerously close to it).

Paradoxically, it was precisely that crisis that paved the way for a rebirth. The acquisition of NeXT in 1996 brought Jobs back home, along with a technological foundation (NeXTSTEP) that would become the heart of Mac OS X. Jobs's cure was as brutal as his approach to staff. Product cuts, the end of the clone program, and a focus on a few clear models. And, above all, a return to an obsession with design and image through the "Think Different" campaign.

The colorful iMac of 1998 is not just a computer but a visual manifesto declaring "Apple is back." The iPod, three years later, proves that the company can break out of the personal computer niche and reshape seemingly mature markets like the music industry. With the iPhone in 2007, the concept of a "phone" was fused with that of a pocket computer and application platform. From then on, the company entered a growth trajectory that led it, according to the data reported in the article, to annual revenues in the hundreds of billions and profits in the hundreds of billions of dollars.

"Think Different" as a guiding principle

The famous 1997 campaign is not just a nostalgic ad

featuring the faces of Einstein and Gandhi, but a statement of intent... Apple wants to position itself where technology and the liberal arts meet, where value lies not in benchmarks



The stunning iMac

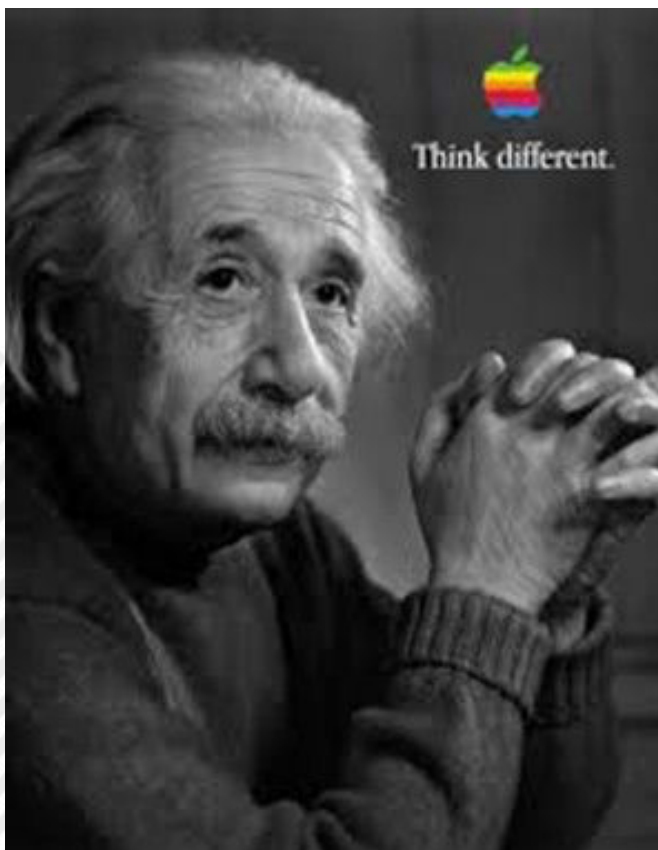




Jobs introduces the NeXT and the NeXTSTEP operating system

but in how a product changes creative work and daily life. This approach is evident in the details of its products: from the consistent interfaces across Mac, iPhone, and iPad, to APIs designed to give developers deep access to the hardware without forcing them to reinvent the wheel,

to accessibility features that allow blind, visually impaired, or physically disabled people to use the same devices as everyone else. It is also why the App Store has become, for better or worse, the gateway to an economy of millions of apps: strict control, but also a stable platform on which to build entire companies.



The “Think different” campaign

The Cook Era... Beyond Hardware

When Steve Jobs passed away in 2011, the shadow of the “aftermath” loomed large. Tim Cook, former head of Operations, lacks the founder’s charisma, but brings with him an organizational skill that allows Apple to do what was once unthinkable: put complex devices into the hands of over two billion people and keep them updated for years. Under his leadership, the scope expands. We see the arrival of the Apple Watch and AirPods, which transform the body and ears into natural extensions of the ecosystem; services (App Store, iCloud, Apple Music, TV+, Pay, Fitness+, Arcade) become a second, highly profitable pillar of the business. The introduction of Apple Silicon brings proprietary CPUs and GPUs to Macs, with leaps in performance and battery life that redefine expectations for “serious” laptops.

At the same time, the company is taking a strong stance on issues such as privacy, accessibility, and sustainability.





Tim Cook shows the future

Data centers powered by renewables, carbon neutrality goals across the supply chain, tracking systems that are more limited than those of competitors, and health and safety features ranging from fall detection to atrial fibrillation monitoring.

Fifty Years: Between Celebration and the Test Bed

As it reaches its 50th anniversary, Apple has chosen a celebration that blends spectacle and storytelling. Live events at iconic stores like Grand Central in New York, performances by artists such as Alicia Keys, a likely grand finale at Apple Park, “Today at Apple” initiatives dedicated to creativity, and a social media archive dedicated to the stories of users and developers. In a public letter, Tim Cook openly thanks everyone who has contributed, from the first employee who soldered circuit boards in a garage to the indie developers who publish on the App Store today, reiterating that every product is just the beginning of a story written by those who use it.

“Think Different”... in 2026

The beauty (and the challenge) of turning 50 is that Apple can no longer limit itself to “thinking different” from others; it must prove it can think differently even from itself. This means, first and foremost, avoiding becoming a cyclist who always rides in the same gear (the classic annual iPhone with incremental updates) while waiting for the road to change its gradient. It means continuing to use vertical integration as a lever to truly simplify the user’s life, and not just to lock down the ecosystem. And, above all, it means putting AI at the service of the individual (accessibility, creativity, privacy) rather than using it solely as a driver to optimize revenue. Is the best yet to come? Only time will tell!





The (almost) forgotten mascots of the Nintendo world

by Carlo Nithaiah Del Mar Pirazzini and Giampaolo Moraschi

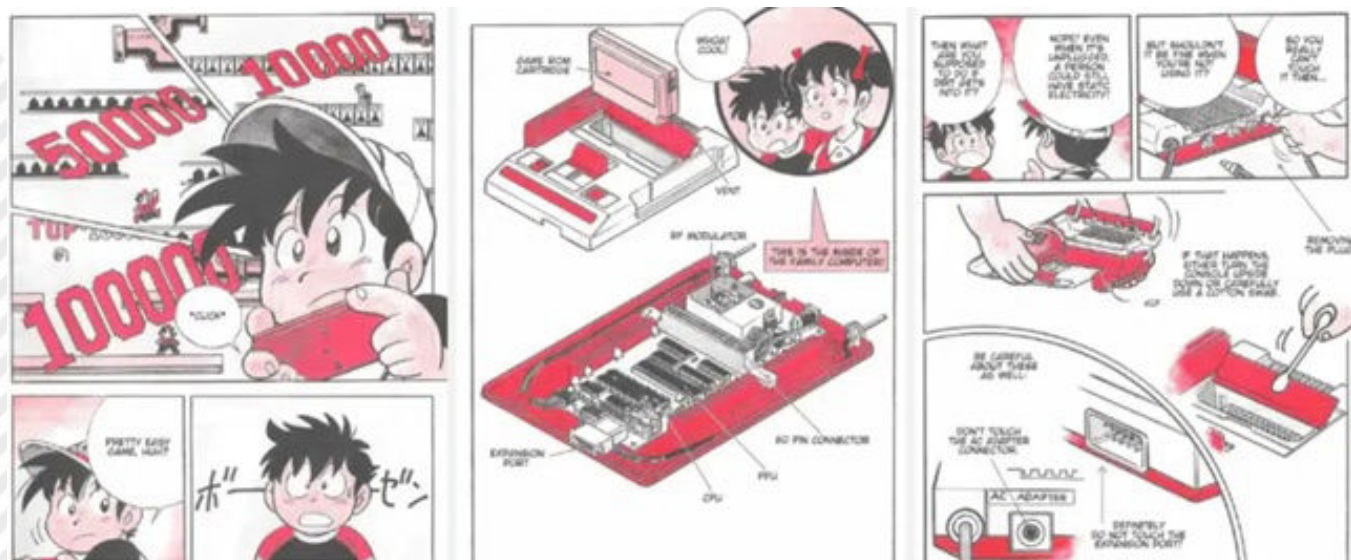
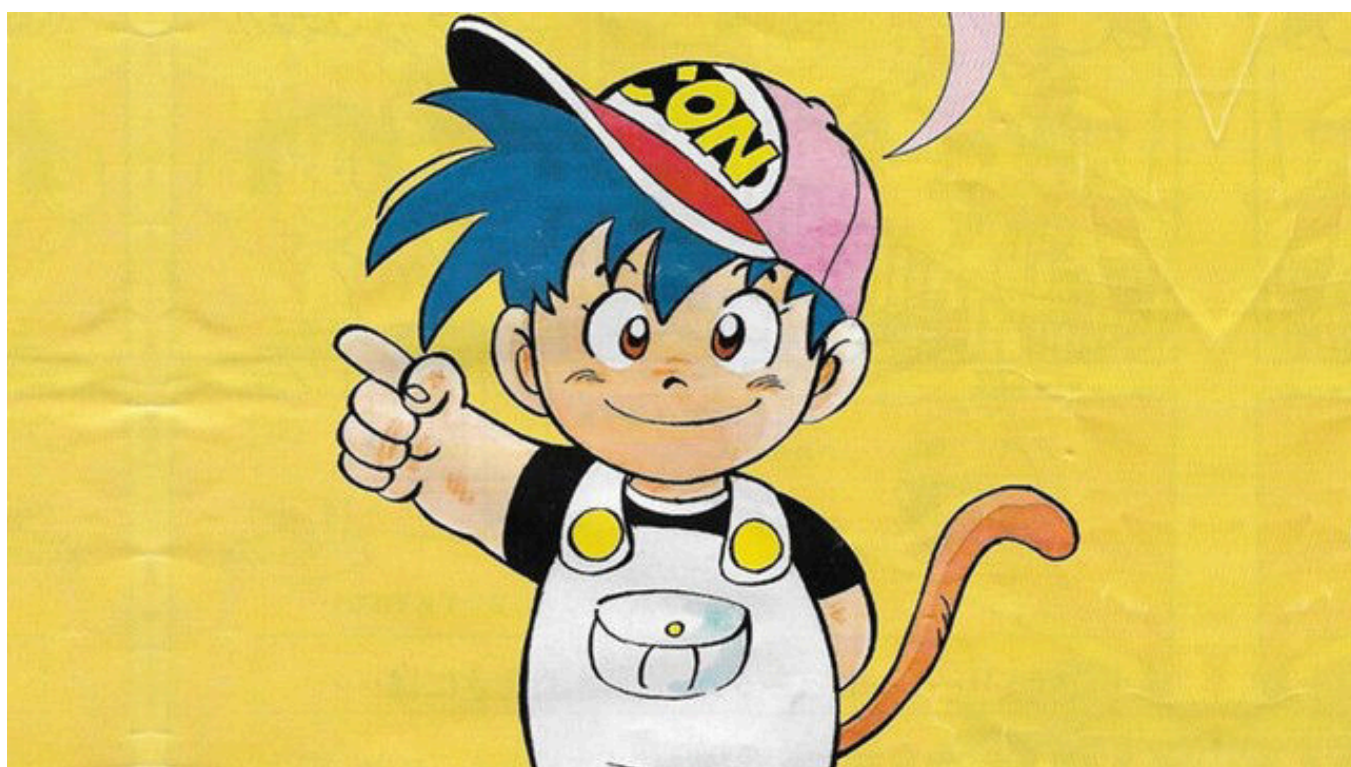
Konkichi, Stanley the Bugman, Excite-kun, Mach Rider... Names certainly not as familiar as Mario and Link, but ones that have left their mark on the world of Nintendo video games. Let's discover them together.

Before Mario, there was... Konkichi!

That's right! Before Super Mario, the official mascot was named Konkichi and bore a strong resemblance to Akira

Toriyama's characters. In the early '80s, Mario made his debut in the world of video games under the simple name of Jumpman. He was beloved, but the Japanese company needed something truly iconic.

Konkichi appeared in the Famicom manuals in 1983. In two booklets, to be precise. The young Konkichi explained, through illustrations and dialogue, how to best use the console. A lively little boy with short hair and a funny hat.



Famicom Manga Instruction Manual... the charming booklet featuring Konkichi





Stanley the Bugman

A mascot designed for children that paid homage to the characters of two great manga artists... Akira Toriyama and Osamu Tezuka.

Despite his funny appearance and great character design, Konkichi disappeared almost immediately... the reason? The Super Mario boom.

Stanley the Bugman

The protagonist of the game "Donkey Kong 3," his name is often confused with other characters, and his impact on gaming culture was limited, despite the game being a commercial success. He was meant to replace Mario in the series with DK, but was soon forgotten despite his funny appearance. He would reappear decades later as a trophy in Smash Bros.

Excite-Kun

The mascot of the Excitebike series, he is a stylized character representing the motorcyclist. Although he appeared in many commercials and game covers, he never had character development comparable to that of other

Nintendo icons.

Mach Rider

The character Mach Rider, protagonist of the futuristic racing game of the same name, is known for its post-apocalyptic setting and unique style. Although the game enjoyed some success, the character did not become an icon like other Nintendo protagonists.



These characters, though less famous, help define the richness and variety of the Nintendo world, proving that even lesser-known characters can be part of a broader gaming history.





AmiGame Jam 2025 - Return to the Arcade

by Carlo Nithaiah Del Mar Pirazzini and Giampaolo Moraschi

This new edition has proven to be truly interesting, with 20 high-quality titles. But before we dive into the top five ranked titles, let's talk a bit about the Jam.

The Origins

The AmiGameJam is an international game development competition dedicated to new Amiga games, hosted on itch.io and organized by Amiga Cammy (Moya Jackie McGeough), which in just a few years has become a sort of "annual health check" for the Commodore homebrew scene. The 2025 edition, for example, collected 20 titles submitted between May 31, 2025, and February 28, 2026, all strictly playable on real hardware or an emulator, and generated a whopping 809 votes from the public, with an average of over forty ratings per game: numbers that show a community that is not only nostalgic but still active and curious.



**La simpatica Moya Jackie McGeough
aka Amiga Cammy**

The process is simple but highly effective. Developers register their project on the jam's page by the deadline, uploading the full game or a substantial version that effectively represents the concept. Once registration closes, the ball is in the community's court: for a few days, participants and players can download, try out, and vote on all the competing titles directly from itch.io, leaving scores and, often, comments and suggestions. At the end of the voting period, the platform calculates an average of the ratings and compiles the final rankings: in 2025, first place went to Echodrome by PixelPlop, closely followed by Double Baboon Ninja by DanLabG, while Wings of Fear II and Creepy Crawlers Goliath: Reloaded tied for third place, followed by a slew of projects ranging from platformers to shooters, from puzzles to racers, all the

way to cross-platform experiments like the Cogs N Logs demo for Amiga and Megadrive.

The purpose of the AmiGameJam, however, goes beyond a simple ranking. It was created to keep creativity around the Amiga alive, offering a shared stage where coders, graphic designers, and musicians can showcase their work knowing it will be tested, discussed, and compared with that of others. Each edition thus becomes a snapshot of the scene: on one hand, there are well-known names returning with new projects or more polished works in progress; on the other, small teams or solo developers emerge, finding in the jam the most direct way to get noticed. The result is a diverse catalog, where completely original games, reinterpretations of classic genres, and demos that serve as a springboard for future commercial or physical releases coexist.



Il logo con il tema della competition di quest'anno

In this sense, the AmiGameJam is not just a scoring competition, but a sort of collective workshop. The constraint of a deadline pushes creators to finalize a working prototype rather than get lost in endless development, while public comparison with other titles helps gauge their own work in terms of graphics, sound, gameplay, and overall presentation. For players, however, the jam serves as a concentrated showcase of new releases: in just a few days, you can discover twenty different games, follow the buzz, compare impressions, and see how the Amiga continues to inspire new ideas despite its age. A must-attend event for Amiga enthusiasts: a celebration of the machine's past, yet with an eye toward the present and future, transforming each edition into a small 16-bit festival of creativity where, beyond the podium, the true reward is seeing how many new games continue to emerge for a platform that has officially ceased to exist for decades,





yet still demonstrates surprising numbers and vitality on itch.io.

The top five finishers... and what the RMW editorial team thinks!!!

Echodrome - Pixelpop - <https://pixelplop.itch.io/echodrome-amigajam2025>



Winner of the Jam, Echodrome is a loud and fun arena shooter with a minimalist yet “trippy” aesthetic. It runs on Amiga OCS/ECS with just 1 MB of memory required. Waves upon waves of enemies to blast and a final boss to pummel. The level design is interesting and the difficulty is well-balanced. It’s a well-deserved spot on the podium.
Gameplay: 90% - Longevity: 90%

Double Baboon Ninja - DanLabG - <https://danlabg.itch.io/double-baboon>



Baboons, platforms, '80s neon lights, crazy aliens... A fantastic game and our “winner.” For now, it’s just a preview with 4 levels, but it’s already a top-notch title showcasing excellent level design and top-notch gameplay. Aesthetically, it’s a masterpiece!

Gameplay: 92% - Replay Value: 80%

Wings of Fear II - Roartj - <https://roartj.itch.io/wings-of-fear-ii>



In this issue, we reviewed the first chapter; here we see a demo of the second chapter, which features new scrolling (horizontal instead of vertical) and a significant improvement in color usage and gameplay speed.

Gameplay: 85% - Replay Value: 80%

Creepy Crawlers Goliath: Reloaded - Digilicious Nibbles - <https://digiliciousnibbles.itch.io/creepy-crawlers-goliath-reloaded>



Another top-tier title and a solid, well-crafted shoot 'em up. Extremely fun and terrifying! It’s a true “arcade” game that punishes even the slightest lapse in attention. The graphics are absolutely delightful.

Gameplay: 90% - Replay Value: 85%





Cake Rush - Hambo3 - <https://hambo3.itch.io/cake-rush>
A competitive and highly playable party game. Perfect for multiplayer and packed with the typical sadism of this genre. Being a pastry chef in this game is deadly; surviving is nearly impossible.

Awesome!

Gameplay: 90% - Replay Value: 90%

The other top finishers at the Jam are:

- 6th Blue And Red by Kaiju Wasp
- 7th The Osiris Infection by Hoyster Games
- 8th Microlyte Tankzone by SteveB
- 9th Pauldron by Zener
- 10th Glow Point by Retro Nanny
- 11th Strike Force Omega by Reborn80
- 12th Kaboomania by Axel
- 13th Santas Present Drop 5 by RobSmithDev
- 14th Arpeggio by Tedzogh
- 15th Submarine Attack by Camden Town Games
- 16th Backflip Endless by GameSoulStudio
- 17th Micro Karting by Alimede
- 18th Burn by Ripper253
- 19th Movimiento Constante by DJcaye
- 20th Cogs N Logs by RetroGamma

You can find them at this link:

- <https://itch.io/jam/amigamejam/results>



Blue And Red by Kaiju Wasp



The Osiris Infection by Hoyster Games



Microlyte Tankzone by SteveB





The Return of the RPG on Apple II with Andoria

by Roberto Del Mar Pirazzini

By now you've figured it out. I have a visceral love for any form of role-playing game. Classics, JRPGs, action games, dungeon crawlers (Nith prefers these), top-down... in short, I love them all, and the Ultima saga is to blame for all of this.

I've always adored this legendary saga created by Richard Garriot, and I'm happy to say that this new project by Emmanuel Castello has the feel of Britannia/Sosaria. Andoria is an Apple II RPG that strikes a chord with anyone who grew up with the Origin saga, but with the stated goal of avoiding the tediousness typical of the great '80s classics.

Concept and Setting

The author presents Andoria as an "Ultima-style" game written in 6502 assembly, designed to run on an Apple II Plus with a language card—making it perfectly plausible for 1979. The protagonist is Shamino, leader of the village of Thalisar, summoned by Lord Ricard (here too we see a clear affinity with Lord British) to Andoria Castle to confront an ancient evil; traditional magic has all but vanished, replaced by druids and potions, in a deliberately simple and deeply nostalgic plot.

Gameplay Pace and Design Philosophy

The designer states his goal was to create an RPG that could be completed in weeks, not months, while maintaining atmosphere and depth without slipping into repetitiveness. Every step on the map represents hours of travel, making route planning and the management of food, water, and rest essential—failure to do so results in death by starvation, thirst, or exhaustion.

Exploration, Survival, and Transportation

Outside the cities, there are no allies: every creature is hostile, and every excursion into the outside world is a calculated risk. Unnecessary detours and aimless wandering are punished by the rapid depletion of supplies and the risk of nighttime ambushes while camping. Progression includes horses, rafts, boats, and, in the full version, a large ocean-going vessel used as a mobile base with increased cargo capacity, destined to radically change the scale of exploration.

Gameplay and Equipment Management

Andoria adopts a single-key command system in true Apple II style: arrow keys for movement and a letter for each action (attack, enter, speak, search, etc.). A defining feature is the use of instruction manuals: weapons, tools, and combat items become usable only after finding and



The game intro and the prompt to insert the second disk containing the game world





Exploration moment

reading the corresponding manual, often hidden in bookcases and shelves to be examined with the “Search” command. An option I found truly innovative!

The arsenal is surprisingly diverse:

- Melee and ranged weapons, including crossbows and bows for tactical combat.
- Progressive armor, helmets, shields, and gloves for those who prefer a more head-on approach.
- Jewelry and gems as high-value loot and “luxury currency.”
- Numerous potions color-coded by type, ranging from classic healing potions to advanced effects (invisibility, night vision, attack/defense bonuses, summons), many of which are not yet available in the demo.

Combat and Tactics

Combat prioritizes speed and positioning. The protagonist is faster than any enemy, favoring intensive use of ranged weapons. The player can switch between weapons and consumables on the fly, using potions and offensive items such as fireballs, traps, bombs, and future advanced abilities related, for example, to environmental control.



More unwelcome encounters...
but this time we're ready!



A run-in with a villager that went decidedly wrong!

It is always possible to flee from a battle at the cost of gold or items, a choice that reduces the frustration of a game over and encourages prudent resource management.

NPCs, Dialogue, and Game World

Cities, villages, and castles are populated by NPCs with whom you can interact via a keyword-based dialogue system, in true Ultima style. The inhabitants introduce the main topics, and the player can type in keywords to delve deeper, making careful reading and note-taking an integral part of the experience.

Demo, Current Limitations, and Ambitions

The available demo offers only a taste: a reduced map, no playable dungeons, no ship, and many advanced features still locked. Despite this, the intended structure of the full game, the centrality of the sea voyage, and the role of dungeons and special areas in progression are clearly evident.

Our preliminary verdict

Even in its incomplete form, Andoria appears to be a coherent and ambitious project that reinterprets the legacy of Ultima in a more modern context, both in terms of timing and player demands. The wealth of equipment, transportation options, potions, and tools hints at a world brimming with emerging possibilities, while the “authentic Apple II” technical constraint adds to the overall charm. If development delivers on its promises regarding dungeons, ocean voyages, and quests, Andoria has all the potential to establish itself as a new cult classic in the Apple II homebrew RPG scene.

Link to try the demo:

- <https://andoria.neocities.org/>





CRYSTIAN AND THE LOST CRYSTALS

Year: 2026

Editor/Developer: Ntpr

Genre: Platform

Platform: Commodore 64

Website: <https://ntpr.itch.io/crystian-and-lost-crystals>

“Crystian and the Lost Crystals” is one of those C64 titles that arrive without much fanfare and, within a few days, immediately become a “sensation” in the homebrew scene. Technically ambitious, aesthetically polished, and with a narrative structure more emotional than the usual maze-like platformer.

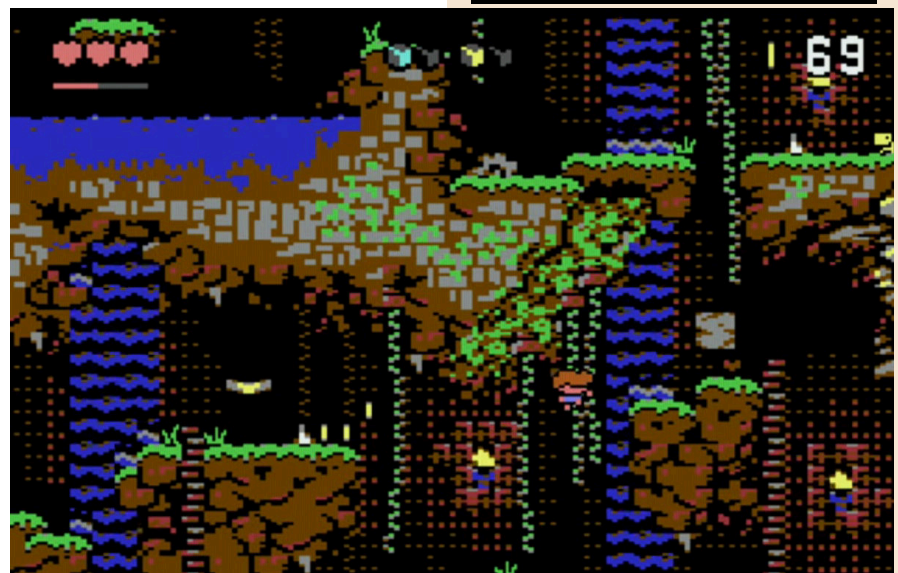
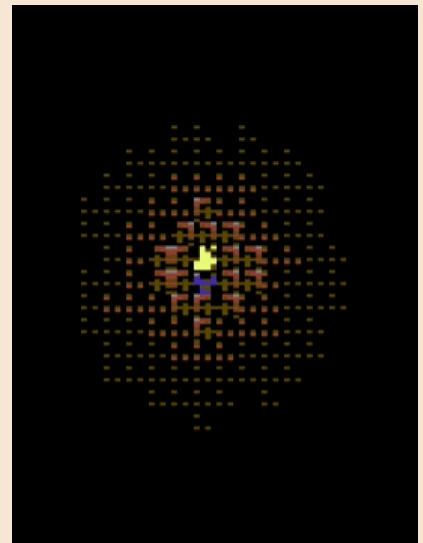
The game, developed by Marcel Sásik (ntpr), who handled the code, graphics, sound effects, and concept, tells the story of Crystian, an ordinary boy who discovers the sentimental value of the “quasi-crystals” he found as a child in a mine alongside his geologist uncle, George. Those stones are worthless to the world, but to him they symbolize his bond with his uncle; when an apocalypse of landslides, water, and lava devastates the tunnels, scattering the collection and leaving his uncle injured and immobilized, the game becomes a recovery mission driven more by sentiment than by greed. It’s a simple premise, yet unusual for a C64 platformer. At its core isn’t “save the world,” but the weight of personal memory and the emotional value of objects.

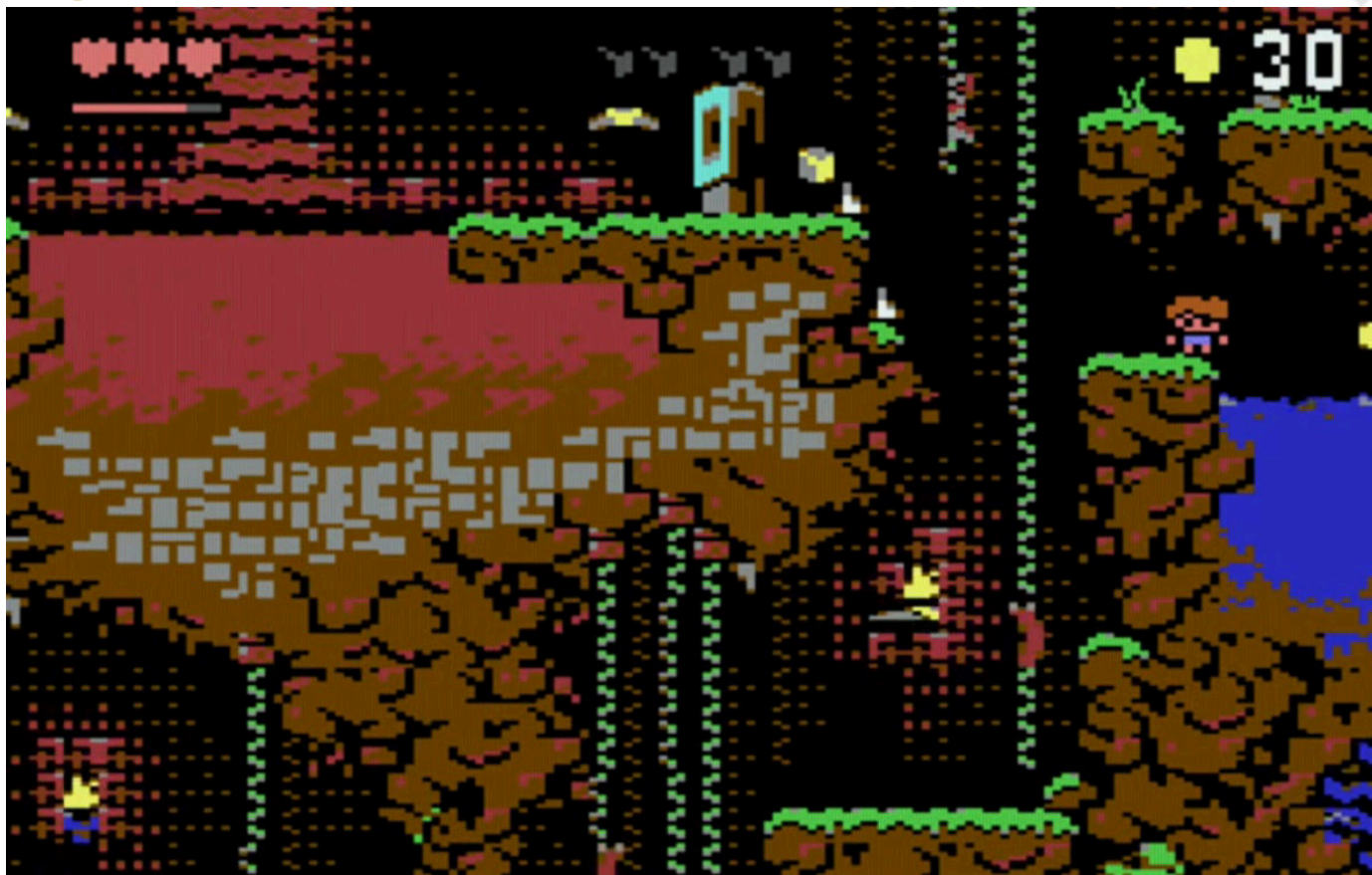
In terms of gameplay, Crystian is a maze-exploration platform-adventure, featuring three graphically rich macro-environments filled with traps, moving platforms, and enemies. Each location features over 60 enemies, with different behaviors and varying damage, and progression is based on a gradual exploration of the “maze” rather than



simple linear levels from left to right. The level design places a strong emphasis on density. There are many sprites on screen, fluid animations, detailed backgrounds, and a clear desire to push the C64 to its limits in terms of animations and the number of concurrent objects.

The gameplay mixes platforming, coin collecting, and a light puzzle element (but very light indeed). In addition to classic coin collecting (100 coins =



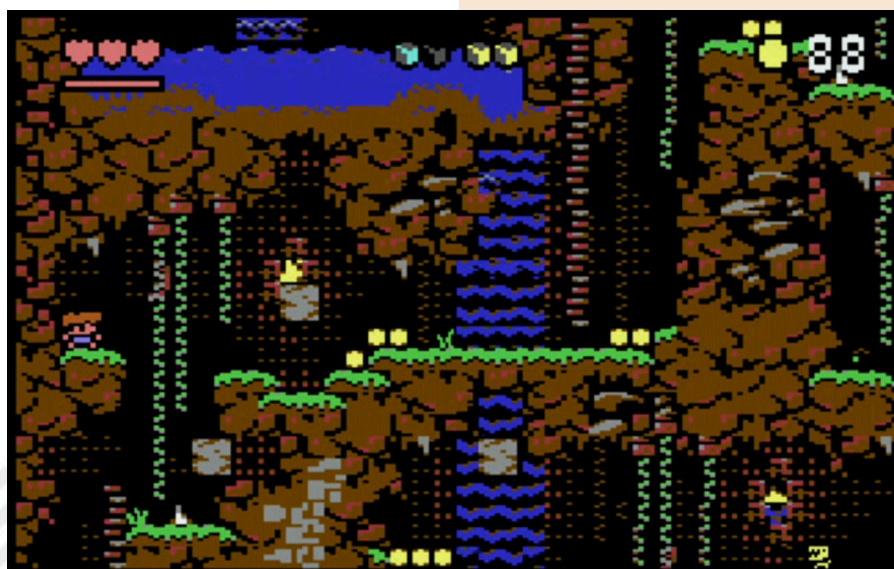
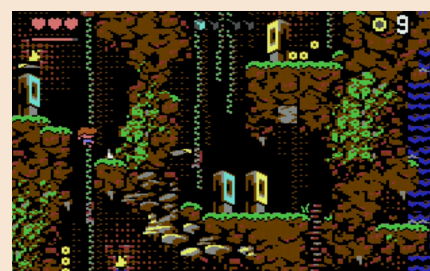


one extra life) and energy and life pickups, progression through the areas is tied to a system of magic cubes that act as keys to open doors and gates. Crystian can carry up to four cubes in total—two per pocket—and deciding which cubes to bring along and when to use them adds a bit of planning to your movements. It's a design touch reminiscent of "soft" puzzles integrated into platforming, without ever turning the game into a pure puzzle game.

The controls are probably the most divisive and defining aspect. The game is based on a fully "grid-based" engine. Every movement, by the player and enemies alike, is aligned with the level's grid cells. This is directly reflected in jumping: if you try to jump "mid-step" by pressing fire in the middle of a movement, the jump won't trigger. To perform a proper jump, you must hold down fire and then choose the direction. The jump is registered and executed at the end of the current movement, which requires a bit of timing planning rather than improvising the input. At first,

you'll be taken aback and find it somewhat difficult, especially on real hardware, but once you get the hang of it (hold fire, then direction), the game becomes much more manageable. The author himself explains that this choice is not a bug, but an intentional component of the game's difficulty and the engine's identity.

Visually, Crystian is a personal showreel. The animations, fluidity, and richness of detail and scenery demonstrate just how much a single developer can still





get out of the hardware in 2026. The sprites are numerous and well-defined, with extensive use of movement and environmental animations. The color palettes are a bit chaotic to distinguish at first, but you get used to them quickly. The soundtrack, composed by Hans Svala with contributions from Sásik himself, is described as “atmospheric” and “cosy music” but set against rather intense gameplay. This is an intentional contrast between the almost contemplative aesthetic of certain sequences and the difficulty of the challenges. Exquisite Bytes’ request for a standalone SID file and the musician’s direct response confirm that the tracks have also impressed those who follow the scene primarily for the audio aspect. In terms of content, the three worlds offer a compact yet intense campaign, with “per-level” saves. When you run out of lives, you restart from the beginning of the level you’ve reached, not from the start of the game. A “rough” difficulty level suited for hardcore players.

It runs on VICE and on original hardware, is compatible with both PAL and NTSC, and is distributed as a .d64 image available for purchase on itch.io for \$3.99 or more.

In summary, “Crystian and the Lost Crystals” is a platform-maze game for the Commodore 64 that places a strong emphasis on atmosphere, visual density, and deliberately unique controls, all woven into an unexpectedly emotional personal story. It’s less intuitive than other arcade-oriented titles, mainly due to the grid-based jumping system, but for those willing to “get into the rhythm,” it offers exactly the kind of experience that, in 2026, distinguishes a simple, charming homebrew from a benchmark game for the scene.

by **Carlo Nithaiah Del Mar Pirazzini**



OUR FINAL SCORE

» Gameplay 80%

Rough around the edges, with some mechanics that require careful and attentive learning. The level design is well-crafted and visually stunning.

» Longevity 80%

The control scheme will test your patience a bit, but once you get the hang of it, it’s a blast!





GHOST'N'GOBLINS AGA

Year: 2026

Editor/Developer: JOTD

Genre: Action/Platform

Platform: Amiga AGA

Website: <https://jotd666.itch.io/ghostsnoblins>

There's a generation that doesn't need 4K trailers to get a thrill: all it takes is seeing a knight in his underwear running through a pixelated graveyard. For us over-40s, Ghosts'n Goblins isn't just an arcade game: it's the smell of smoke and plastic in the bars, the sweaty coins clenched in our fists, the metallic clink of the coin sliding into the cabinet, the light from the monitor piercing the darkness of the arcade. It's the game that forgave you nothing, the one that humiliated you at the first zombie, but forced you to try again and again, because every meter conquered tasted like an epic feat. It was coming home with the ghost of its jingles in your head, drawing Arthur in your school notebooks, telling your friends "I made it ALL THE WAY TO THE CASTLE" as if you'd climbed Everest. And today, when we see that purple sky again, those tombs, those winged demons, we're not just looking at a game: we're reopening a door to a time when all it took was a joystick, two buttons, and a lot of determination to feel like heroes—even if only for three lives. "Ghosts'N'Goblins" by jotd is the Amiga AGA port that Capcom fans have been waiting for thirty years. All seven levels of the original arcade game, 64 on-screen colors, and a fidelity to the gameplay that borders on philology, including the legendary difficulty of the level design. Arthur runs, jumps, and throws his weapons exactly as expected, with collisions, enemy patterns, and a pace of action that closely mirror the coin-op—



without a shadow of a doubt, a "brilliant conversion." In terms of options, the port doesn't just reproduce the game but frames it with a series of very welcome extras. We find difficulty and life count adjustments, a full-fledged arcade-style "service mode," and a suite of built-in cheats (invincibility,





level skip, weapon swap, vertical warp, score boost) that make the experience adaptable from hardcore masochists to nostalgic casuals. The distribution in LHA format with WHDLoad installation also makes it perfect for real A1200s, upgraded setups, and even machines like the THEA500 Mini, as the author himself confirms in the comments on the itch.io page. It's not all gold without a few pixels out of place. The early releases aren't free of scrolling issues (especially in PAL) and peculiar compatibility with certain pad/CD32 configurations, with the author explicitly suggesting to force NTSC or 50 Hz mode and candidly admitting he's still working on the compromises imposed by the AGA's 64-pixel horizontal scrolling. That said, we're talking about an evolving version that's updated frequently (releases 0.92 and 0.93 have already improved NTSC support and fixed "nasty" bugs that surfaced in WHDLoad), a sign of a vibrant, active project.

Overall, this "Ghosts'N'Goblins" is one of the most compelling Amiga reinterpretations of the Capcom arcade classic. It is faithful to the original, technically ambitious, and supported

by an enthusiastic community that is helping to polish it to perfection. If you've always dreamed of suffering through the game just like in the arcade but with the AGA pumping out graphics and audio, this port is already a must-have, destined to become the definitive benchmark as development continues.

by **Carlo Nithaiah Del Mar Pirazzini**

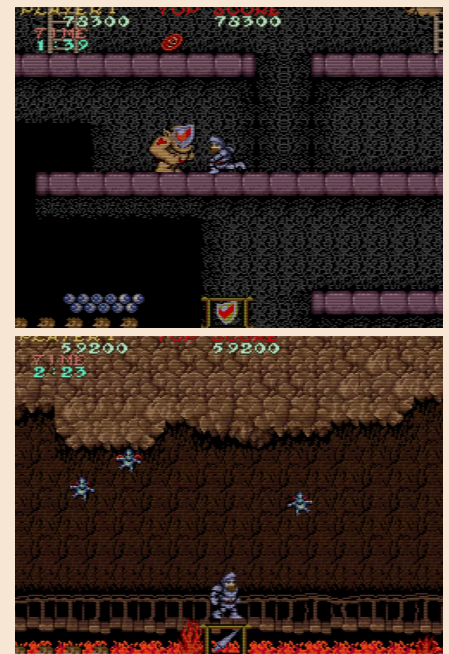
OUR FINAL SCORE

» Gameplay 90%

Responsive, punishing, and rigorous—for better or worse. Arthur responds well to controls, and the feel of jumping, throwing, and collisions closely mirrors the arcade version, but precisely because of this, the difficulty curve remains brutal and unforgiving.

» Longevity 99%

Seven full levels, the fierce difficulty, and the inclusion of options specific to this version and cheats make the game something you'll return to often, both to try to improve your skills and to explore the various settings.





MAPPY

Year: 2026

Editor/Developer: JOTD

Genre: Platform/Puzzle

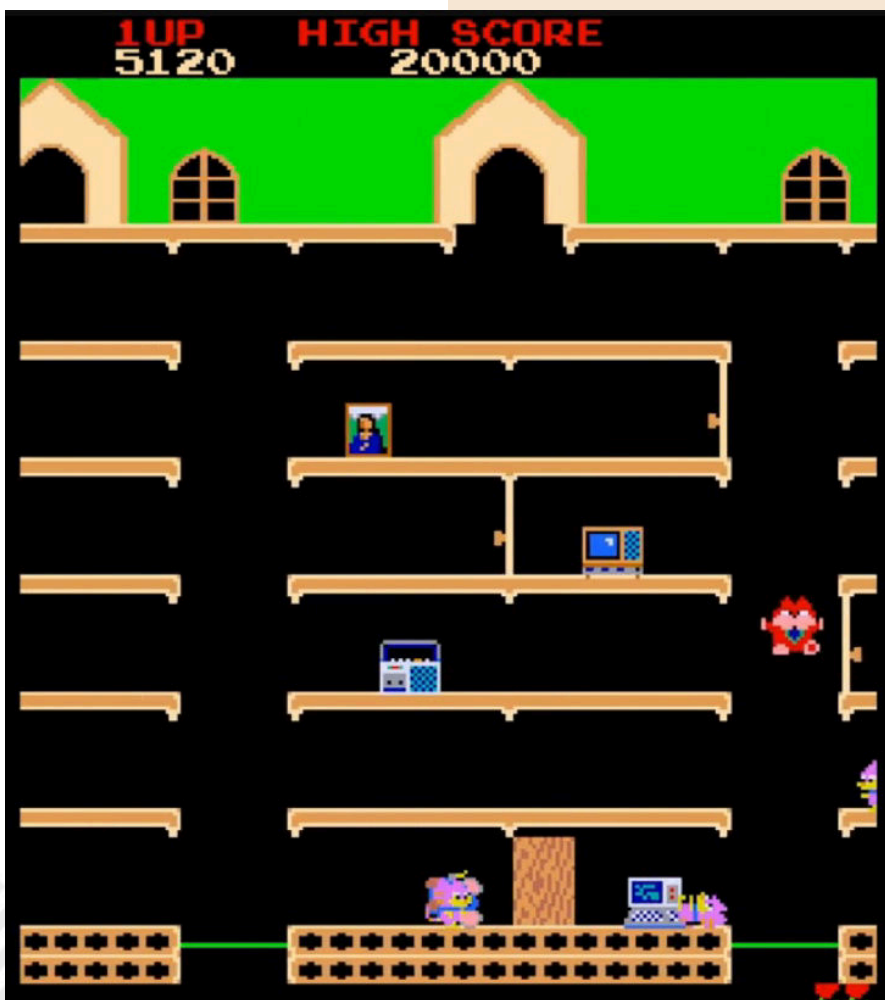
Platform: Amiga AGA

Website: <https://jotd666.itch.io/mappy>

JOTD has a stated mission: to bring the great arcade classics back to the Amiga, recreating that arcade feel. In the original arcade game, Mappy is a police mouse tasked with retrieving a series of stolen items inside a multi-story maze-like mansion, while being chased by a gang of thieving cats led by Goro. The game takes place across a set of six-level houses connected by trampolines that replace jumping. The character cannot jump freely, but bounces on trampolines to move vertically, with mechanics that make timing crucial and transform each level into a chess match of trajectories and chases. Doors serve as the primary tactical weapon; slamming them in the Meowkies' faces stuns them, while blue doors release waves that temporarily clear entire corridors, adding a strategic layer to a platformer that thrives on patterns, memory, and spatial management. As the stages progress, bells appear that can be dropped on enemies and rainbow floors that collapse after you pass over them, while the game speeds up with the infamous "Hurry!" and the arrival of the punishing Gosenzo Coin, which forces the player to quickly complete the pattern.

This Amiga AGA port of JOTD was created with a very specific philosophy: to perfectly recreate the arcade experience on the Amiga. With the same colors, the same gameplay, and the same "arcade feel." The developer

used reverse engineering to decode the original code, translate it to 68000, and adapt it to the Commodore machine, also handling the graphics and sound conversion. The use of the AGA palette allows for a very close replication of the arcade cabinet's vibrant look, which at the time ran on Namco hardware derived from Super Pac-Man. Sprites, backgrounds, and the interface capture the color palette and crispness of the original, preserving that mix of cartoonish aesthetics and tension that defines the game. We're sure a way will be





found to release it in an ECS version as well, but for now, there's no word on that possibility.

In terms of gameplay, the Amiga version fully reproduces the level structure, enemy behaviors, and stage progression. Controls remain basic, with movement via joystick or keyboard, a single key to open doors, and the usual arcade credit simulation (5 to insert "coins," 1 or 2 to start the game). The feel remains that of a puzzle-platformer disguised as an action game: without free jumping, the player must manage bounces on trampolines and predict the cats' trajectories, using doors and shockwaves to clear safe paths. The physics of the trampolines and their "wear and tear" if used too often are preserved, maintaining the tension typical of the later levels. On itch.io, anyone can download the floppy version (zip) or the HD installation (LHA) for free and, if they wish, leave a donation to support development.

The game reflects the style and action of the original title, which, as we recall, was from 1983. A different era, where arcade cabinets were calibrated to make us "lose coins" due to difficulty that wasn't always balanced. Mappy belongs to that world and is perfect for those nostalgic for the past... less

so for those seeking a more balanced challenge.

Overall, it stands as a faithful tribute to a Namco classic, more concerned with preserving the structure and character of the 1983 game than with reinterpreting it. The combination of carefully recreated gameplay, a faithful color palette, and small modern conveniences makes this port a significant milestone for those who love seeing the Amiga take on the great arcade games of the era, and an ideal gateway for those who want to discover why a police mouse besieged by thieving cats continues to be remembered as one of the most intelligent puzzle-platformers of the 8-bit era.

by **Giampaolo Moraschi**

OUR FINAL SCORE

» Gameplay 85%

The arcade experience is here in all its glory!

» Longevity 80%

...but it suffers from the gameplay style of the era and a difficulty curve that isn't quite balanced!





AKALABETH 1000

Year: 2026

Editor/Developer: Dark

Genre: Dungeon Crawler

Platform: Sega SG1000

Website: [https://](https://www.smspower.org/Homebrew/Akalabeth1000-SG)

www.smspower.org/Homebrew/Akalabeth1000-SG

In our beloved retro world, everything revolves around nostalgia and experimentation, and this Akalabeth 1000 almost seems like a joke.

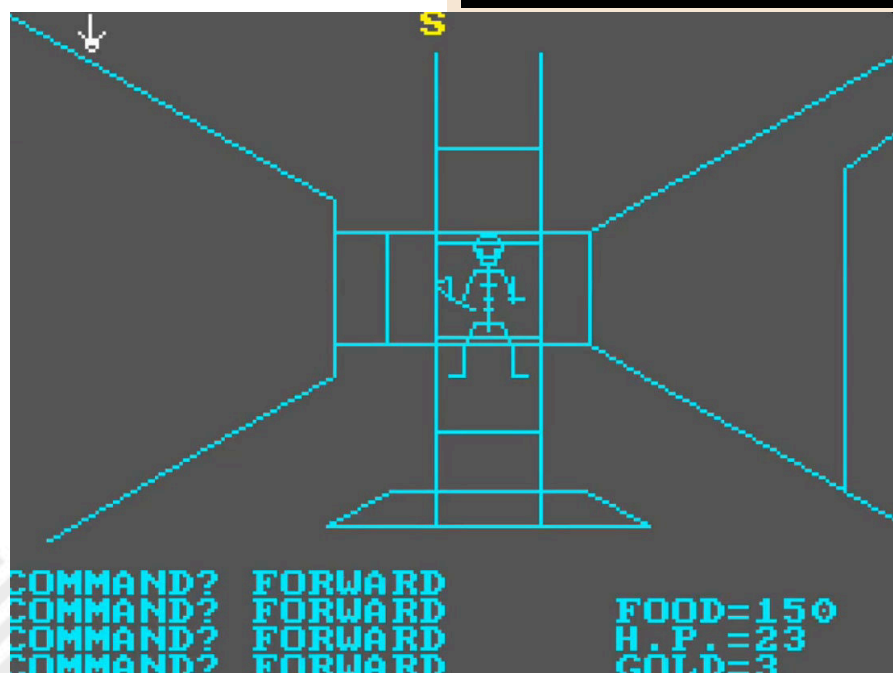
Taking the prototype of Ultima—born as a semi-amateur experiment by Richard Garriott in 1979—and cramming it into Sega's first home console, a machine designed for shooters and platformers and certainly not for first-person dungeon crawlers. The result, however, is no laughing matter—it works, and surprisingly well. The heart of the project is a port of the GW-BASIC version of Akalabeth, entirely rewritten in C and optimized to the point of running on an SG-1000 with just 1 KB of available RAM, with the bulk of the work handled by the ROM, which comes in at nearly 48 KB, including aggressive compression of text, graphics, and audio. The author took advantage of this structure to add several improvements over the source material, introducing new features and content that transform the game into something more than a mere historical curiosity disguised as a stylistic exercise.

Played on an SG-1000 or SC-3000, Akalabeth 1000 is still clearly a late-'70s dungeon crawler featuring alternating top-down maps and pseudo-3D wireframe dungeons, text-based combat, and a rough progression that preserves the spirit of the original, with all its brutality and pioneering charm. Visually, the presentation is clean but spartan, with colors appearing more faded on actual hardware than in emulation, and some



areas less polished, particularly in the combat layout and text output, as the author himself admits when discussing the “poor” handling of certain texts and possible residual bugs. From a historical and conceptual standpoint, it is a fascinating endeavor for at least two reasons. On the one

```
Choose thy lucky number:
42      1980      6174      30000      65535
Level of Play:
01 02 03 04 05 06 07 08 09 10
HIT POINTS...09
STRENGTH...199
DEXTERITY...19
STAMINA...19
WISDOM...10
GOLD.....07
Shall it thou play with these
qualities?      ACCEPT      REROLL
```





OUR FINAL SCORE

» Gameplay 85%

Brutal but beautiful and pioneering in terms of interface and game structure.

» Longevity 80%

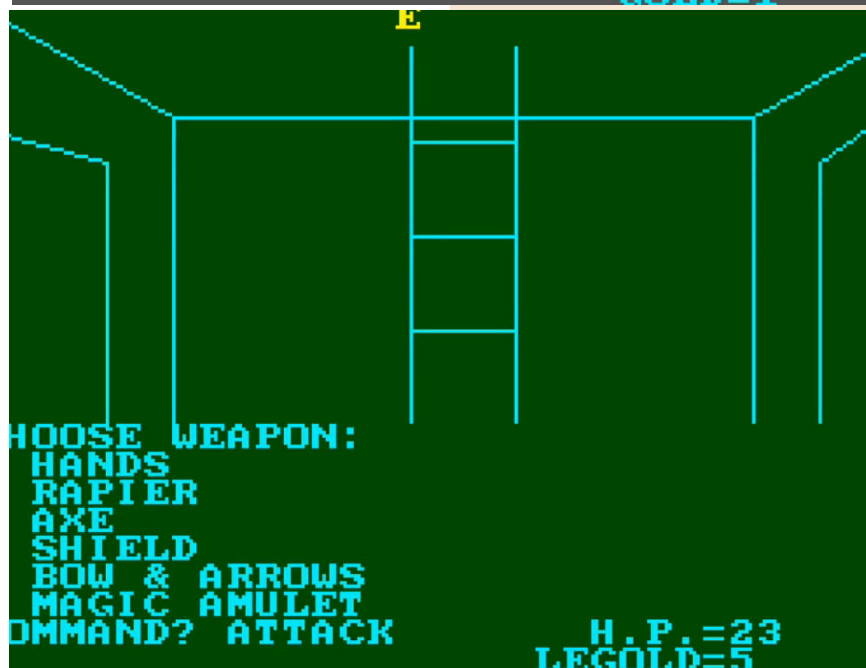
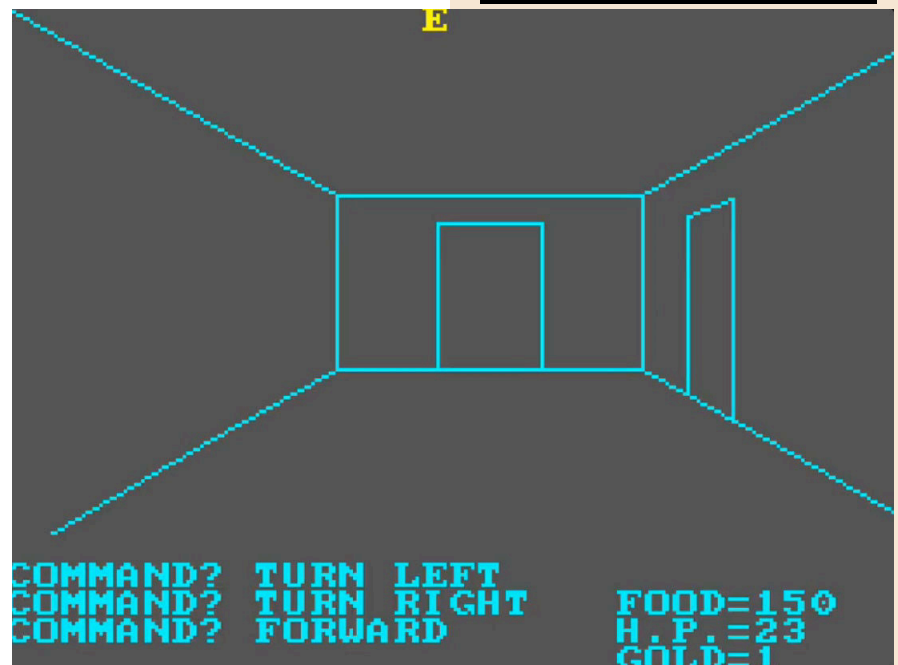
Not very long but challenging.



hand, it brings one of the founding fathers of role-playing games to a console that, at the time, had practically nothing comparable, ideally anticipating the RPG experiments that Sega would see only years later; on the other, it demonstrates just how much can still be squeezed out of the SG-1000's modest hardware thanks to modern tools and even AI-assisted optimizations, used to compress and speed up the C code to almost absurd limits.

Akalabeth 1000 is neither an "accessible" RPG in the modern sense nor a glossy remake, but it is a deeply respectful, technically ambitious, and openly work-in-progress tribute that uses the homebrew scene to bridge three different eras: that of late-'70s BASIC, an early-'80s console, and the development tools of 2026. For those familiar with the history of role-playing games, or for collectors of SG-1000 oddities, it's a gem worth exploring, if only to see just how far a machine designed for other purposes can be bent to tell the story of a dungeon nearly half a century old.

by **Roberto Del Mar Pirazzini**





WEE NINJA

Year: 2026

Editor/Developer: Playsoft

Genre: Platform game

Platform: Atari 5200

Website: <https://forums.atariage.com/topic/382115-wee-ninja-release/>

Wee Ninja is, quite simply, the “Mario-style” platformer that the Atari 5200 never had back in the day (in fact, it never had many games to speak of), but which now arrives in a surprisingly polished form that is well aware of the hardware’s limitations (and strengths). The game offers 32 levels divided into 8 worlds of 4 stages each, with linear progression and world-ending bosses that crown the advance, including a second variation of the boss to break up the monotony of repeated battles. The level design fully embraces the grammar of Super Mario. We have everything a platformer requires: platforms, holes, accessible green pipes hiding secret areas, and enemies that lend themselves to both “head-on” jumps and strategic use of available weapons. The difficulty curve was refined during development until it reached a more than pleasant balance, with a pace that avoids frustration and feels immediately intuitive from the very first playthrough.

The controls are classic “Nintendo 8-bit” style, with the stick for movement, the bottom button for jumping (with height determined by how long you hold it), and the top button dedicated to attacks in the powered-up forms, featuring solid jump physics that deliver a modern feel despite the 5200 controller.

The heart of the system is the ninja’s triple transformation, activated by hitting blocks. The red ninja grants temporary invincibility, the black allows you to throw “square shurikens”



from a distance, and the blue enables close-range attacks with the katana/staff; in all cases, taking a hit reverts you to the base state rather than immediately deducting a life. The final version adds an “easy” mode with extra hit points represented by a heart





next to the life counter, distributes hidden bonus hearts obtainable only by meeting specific conditions, and introduces limited continues (five) plus world codes after bosses, allowing for a less “brutal” learning curve compared to a classic run.

Under the hood, Wee Ninja utilizes a 64K ROM with Bryan’s M.U.L.E. bank-switching scheme—a historic solution for bringing large software titles to the 5200—and runs smoothly on real hardware via Atarimax Ultimate SD or BackBit Pro as well as via Altirra (preferably with the .car file for autodetect). The use of bank-switching allows for a wealth of graphics, game logic, and music that far exceeds the standard of the system’s original commercial titles (again, this comes as no surprise, given the scarcity of titles and their generally low quality), so much so that several users point out how a product like this would have made a difference had it arrived in the early ’80s. From an audiovisual standpoint, the backgrounds and tiles immediately evoke a Mario-like aesthetic—perhaps a bit too much—but they are tastefully adapted to the 5200’s color palette and capabilities, while the soundtrack is a reworking of tracks from Super Boy 3 on MSX, another

Mario-style clone, with a few audio glitches that the author plans to fix in a future release. Although it owes everything to the Super Mario model, Wee Ninja is not a simple graphical hack. The management of the ninja’s three forms, the difficulty tuning, the presence of world codes, hidden extra lives, and boss variations give the feeling of a complete platformer, designed for the 5200. At the same time, the aesthetic similarity to the original Nintendo game is so striking that the author prefers distribution solely in ROM form to avoid potential legal issues regarding a physical release, deliberately leaving it up to individuals to create their own homemade cartridges. In the Atari homebrew scene, Wee Ninja thus carves out a role as both a technical showcase and a manifesto, demonstrating just how much can still be squeezed out of the maligned Atari console in terms of “Japanese-style” platforming and, at the same time, how thin the line is between homage and clone when playing with icons of the past. Excellent product!

by **Carlo Nithaiah Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 90%

Excellent responsiveness in controls and a nice variety in gameplay style. A Mario clone, but with a nice extra variety.

» Longevity 90%

The 32 game levels are plentiful, and the challenges are enjoyable. It will keep you glued to the controller.





SPACE BASH: CONTACT!!

Year: 2026

Editor/Developer: Dave's Retro Forge, The Geps, Hassink

Genre: Puzzle/Platform

Platform: MSX, 16 KB di RAM

Website: <https://daves-retro-forge.itch.io/space-bash-contact-msx>

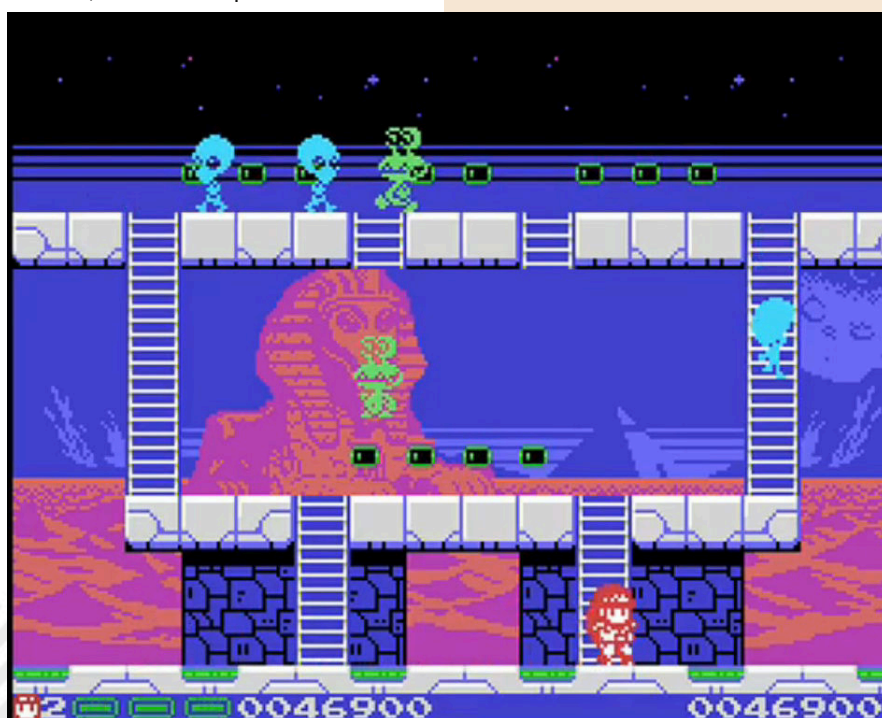
My name is Ingrid Poggiali, and I'll admit it: I jumped into *Space Bash: Contact!!* thinking I was going on a space tour, but I ended up in an intergalactic move armed with a "laser beam," a Kamehameha, and a portable ladder.

On paper, it looks like a fixed-screen platformer in the style of Rodland, but in practice, it's a mini-lesson in game design and a tribute to Jaleco's classic title.

The mission is crystal clear from the start! Clear twenty levels—both indoor and outdoor environments—of every kind of bizarre, otherworldly alien imaginable. All this in a puzzle-action mix of platforms, walls, tight spaces, and that surgical precision in jump management that really makes you realize just how much obsessive love went into this game. The fact that all this "BEAUTY" runs on any MSX with at least 16 KB of RAM makes it even more satisfying to see how much variety can be squeezed out of such limited resources. Controller in hand—or keyboard, since we're used to that—Space Bash: Contact!! is a mini-gym for your reflexes. Move the heroine, build folding ladders to reach places where simple physics won't cut it, launch into glides to cover absurd distances, and all the while try not to get too enthusiastically "hugged" by the aliens. The coup de grâce is the attack system! Stun an enemy and "smash" them into the



others, triggering a chain reaction of stuns that turns half the screen into a perfectly satisfying, delightfully mind-blowing cosmic bowling alley! To this, the development team has





added the secondary energy shot, a chargeable energy ball that saves the day in situations too complex even for a space ninja. Using it clears the field and feels great right away, but it's a precious resource—not an infinite “No Panic” button—and forces you to plan ahead.

Technically, it's top-notch! A Konami SCC-style Roma at about 544 KB, music and sound effects handled by the SCC chip, and graphics that nod to a more “advanced” color palette and choices while remaining perfectly believable on the basic hardware. The work on the animations is superb. I love the aliens with eyes on top of their heads that spit to take us out, or the little “farting” cubes that flutter across the screen. The music by John Hassink pumps up the action in a way worthy of a miniature arcade. There's so much classic arcade vibe that you'll want to crank the volume all the way up!

Behind the scenes is a tight-knit team: concept, game design, and graphics by Davide Bottino (the creator of Toki Remastered, Bubble Bobble Remastered, and Bubble Bobble: The Lost Cave on the C64, ed.), the aforementioned

Hassink for the music, and TheGeps (Giuseppe Ettore Pintus) on coding and programming. These three talented craftsmen put it all together for MSXdev 2025 with a free release in line with the contest rules. As a self-proclaimed “failed engineer” (meaning I still have a ton of exams left to pass), I can't help but appreciate how Space Bash: Contact!! manages to layer the experience. You can play it “casually,” limiting yourself to surviving and collecting batteries, or you can study the layout, exploit every inch of the slopes and glides, orchestrate stun combos, and squeeze the scoring system for extra lives and personal prestige. I can only admit that, if this is how new chapters for the MSX1 will continue to be written in 2026, then I reckon my father's old home computer and I still have plenty of galactic evenings to spend together.

by **Ingrid Poggiali**

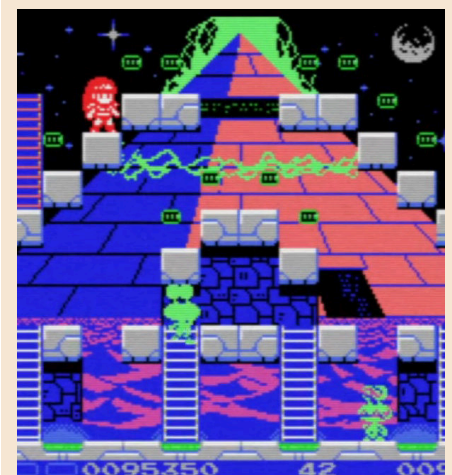
OUR FINAL SCORE

» Gameplay 95%

Wonderful hit box, incredible technical presentation, meticulous level design, and extreme attention to detail in every aspect. It's the MSX title of 2026.

» Longevity 95%

One game leads to another.





CELESTE

Year: 2026

Editor/Developer: Paweł Juen Nowak

Genre: Action/Platform

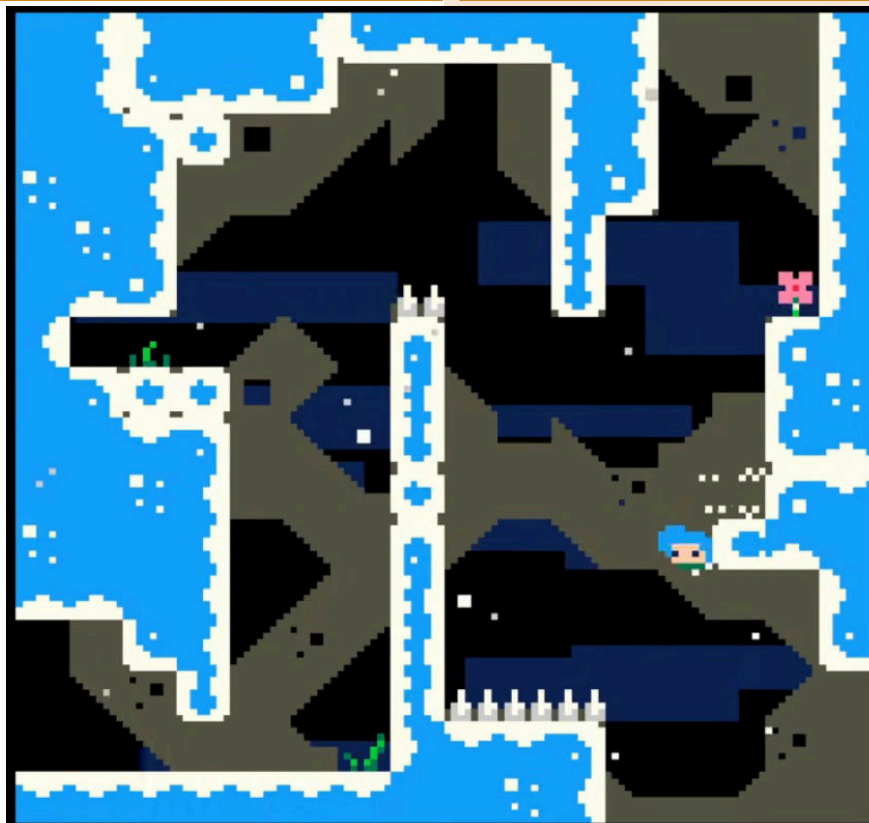
Platform: Amiga

Website: <https://twoj.szajb.us:442/index.php/s/2500eKmECbyw82g?opendetails=>

The cult platformer Celeste continues to take the retrogaming scene by storm!

Celeste is an indie platformer developed by Maddy Thorson and Noel Berry (Maddy Makes Games / Extremely OK Games), which began as a Pico-8 prototype created in four days during a game jam. The full version was released on January 25, 2018, on PC, consoles, and Linux/macOS, and puts us in the shoes of Madeline, a girl who decides to climb Mount Celeste to confront anxiety and depression, with gameplay based on pixel-perfect jumps, timed climbs, and a rechargeable mid-air dash. Behind the speedrunner-level reflexes lies a deeply personal narrative, a soundtrack by Lena Raine, and a challenging yet never punishing level design, featuring instant respawns and death used as a metric for improvement. Paweł Juen Nowak's Amiga port starts with the open-source Pico-8 version—that is, "Celeste Classic"—which features around thirty levels designed for rapid, precise movements. From there, everything is adapted to the Amiga's planar world using C2P (Chunky to Planar), a technique that converts the linear framebuffer typical of modern/virtual machines into graphic planes compatible with OCS/ECS/AGA, at a CPU cost that is far from negligible.

From a gameplay perspective, the Amiga port captures the essence of





the original Celeste: movement based on jumping, time-limited climbing, and a mid-air dash in any direction; compact level design, full of deadly obstacles but with instant respawn; collectibles like strawberries in tricky spots that push you to master movement.

Obviously, we won't get the full 2018 release package (all chapters, B-side/C-side variants, the 2019 free Farewell DLC), but the portion ported to Amiga is the "purest": the technical and skill-based challenge, without too many frills. The big difference lies in the rendering. The Amiga has richer audio and graphics than the NES and Pico-8, but its planar architecture makes managing sprites and layers designed for a linear framebuffer more complex, so every smooth frame is a minor miracle of optimization. The game is designed for "powerhouse" Amigas (A1200/A4000 with various accelerators) or well-configured emulators, with the community—particularly the PPA.pl forum—providing configurations and feedback to squeeze every last CPU cycle. This is a port that doesn't aim

for the universal accessibility of the modern version (which even offers an Assisted mode to adjust the difficulty), but is clearly aimed at those who know the hardware, accept the compromises, and want to see just how far the good old Amiga can go with a 2018 design.

by **Giampaolo Moraschi**

OUR FINAL SCORE

» Gameplay 85%

Celeste is fun and enjoyable. The level design is well-defined, and it's a pleasure to play.

» Longevity 80%

The game definitely has longevity, but it struggles a lot and is only suitable for "high" settings.





RANDOOM

Year: 2026

Editor/Developer: Epy, Csabo, Unreal, SukkoPera

Genre: Puzzle/Platform

Platform: Plus/4

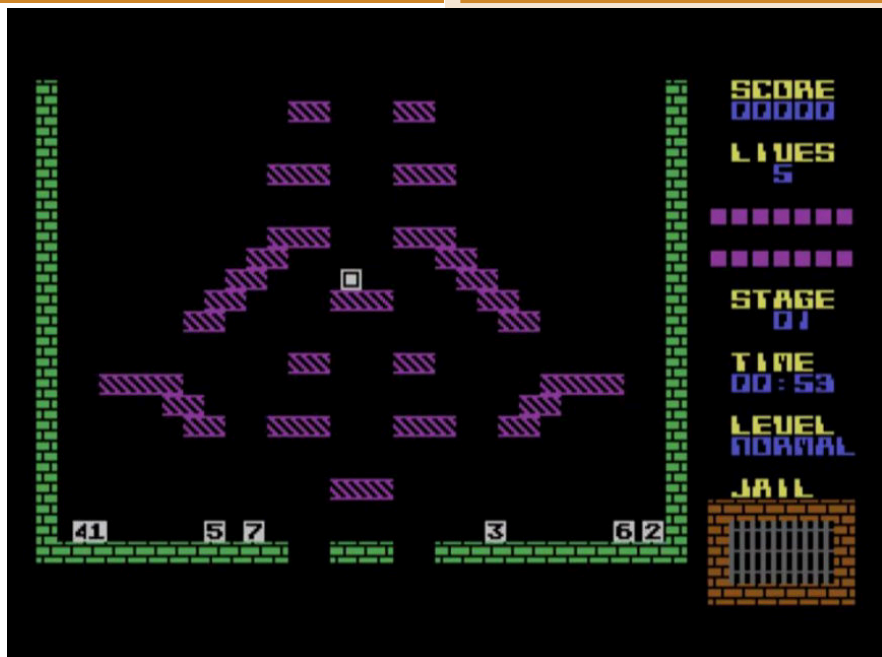
Website: <https://plus4world.powweb.com/software/Randoom>

Randoom tricks you! A game that seems “tiny” on paper but, in just two minutes of gameplay, pulls you into a loop of “just one more game and I swear I’ll stop.”

A platformer born from a minimalist concept, it puts us in the shoes of a “little guy” tasked with collecting squares in the correct order to complete the level and move on to the next. Simple to understand, but far from trivial to master when the timer is ticking and the layout changes from level to level.

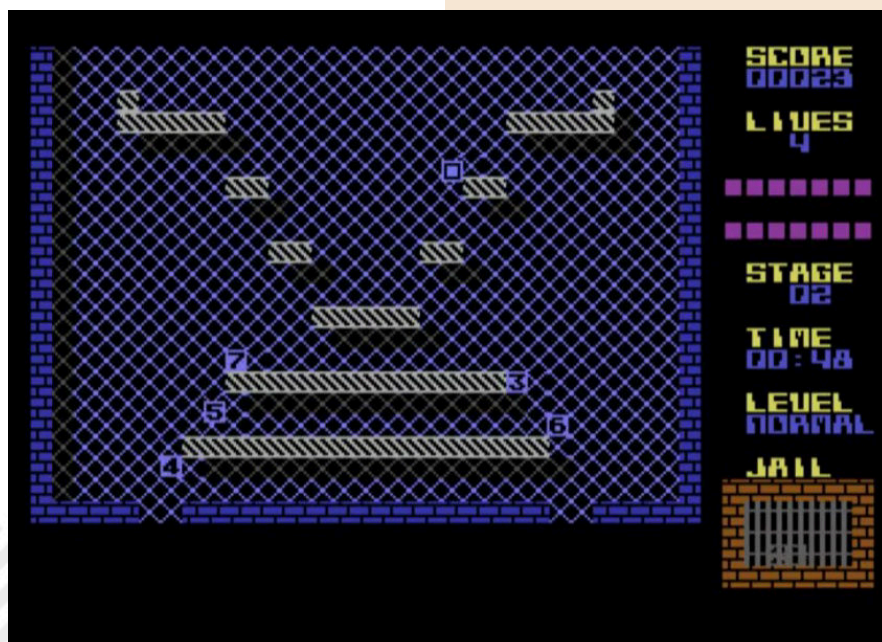
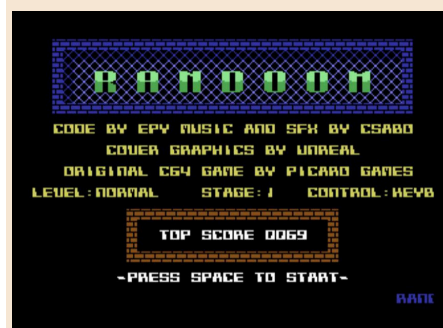
The structure is based on 31 selectable stages, accompanied right from the start by an unusually generous difficulty system that lets you adjust the level from five selectable modes: too easy, easy, normal, hard, and insane. Each of these modifies the pace, the timer’s strictness, and the precision, allowing the player to adapt. Perfect, therefore, for the novice as well as for the master gamer in search of the “Perfect Score.”

On the mechanics front, in addition to standard movement, the mid-air jump stands out—a small leap that doesn’t distort physics but saves our skin, adding that extra margin of control needed to correct trajectories at the last moment. Time management is central: every second saved translates into Bonus Time, incentivizing aggressive play—almost



bordering on speedrunning—rather than a cautious but inefficient approach.

The game’s technical quality is well above the average for our beloved Plus/4. It makes the most of the machine’s 64K, featuring excellent





OUR FINAL SCORE

» Gameplay 92%

A joy to play, with a wealth of options to create the “perfect” session and unexpected but welcome support for the SNES controller.

» Longevity 90%

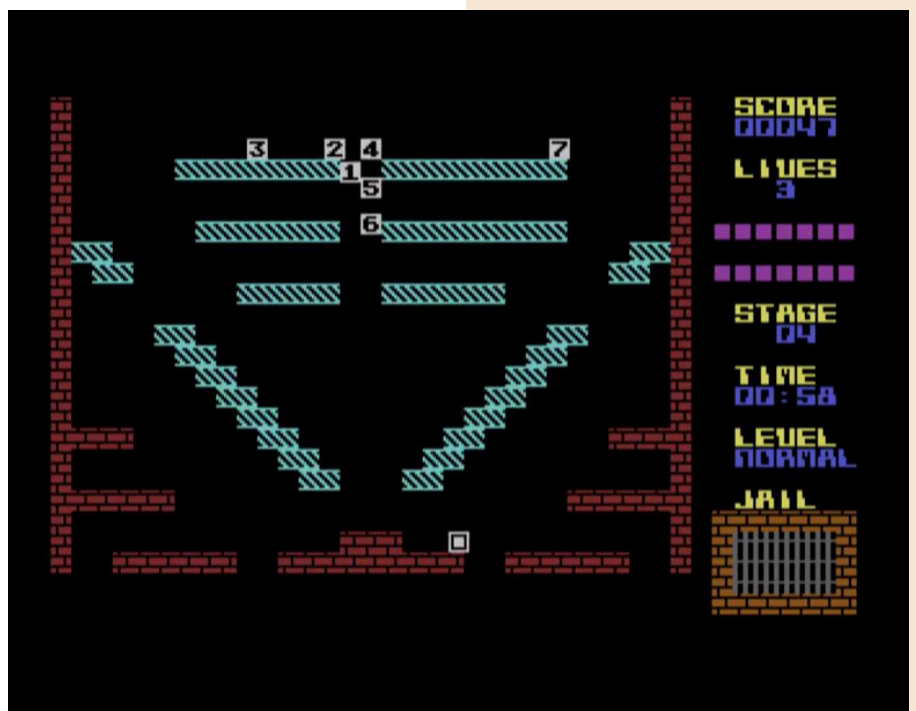
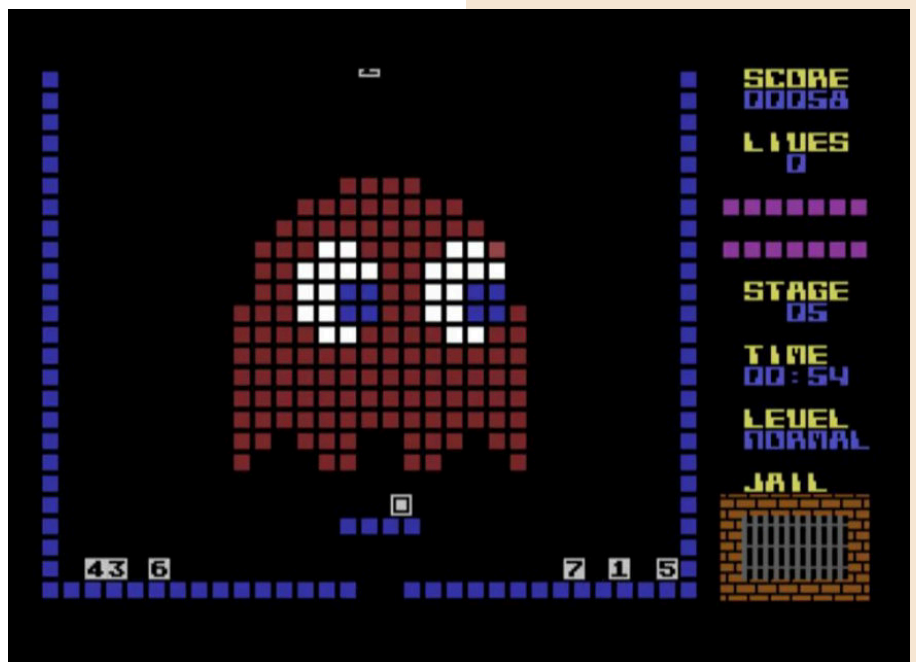
The difficulty system helps any type of player learn the mechanics. It’s without a doubt the kind of game where “one match leads to another.”

programming and a range of options worthy of modern indie games. Support for a wide range of controllers is a standout feature, from the keyboard to the Super NES controller. This last “gem” makes the game super-comfortable even during very long sessions—a detail not to be taken for granted on the original hardware.

The team behind all this is seasoned and well-known in the Plus/4 scene. The code features Epy, graphics by Unreal, a soundtrack by Csabo, and additional support code by SukkoPera, all packaged as freeware in PAL-only mode.

Randoom is an action platformer with stripped-down mechanics, whose real appeal lies in precise controls, scalable difficulty, and an obsession with scoring, proving once again how the Plus/4, in the right hands, can host surprisingly contemporary designs. Having enjoyed it on other platforms (MSX and C64), it’s a pleasure to see it in such fine form here as well.

by Carlo Nithaiah Del Mar Pirazzini





CHRONO RUNNER

Year: 2026

Editor/Developer: Associazione MSX Italia

Genre: Platform/Puzzle

Platform: MSX

Website: <https://www.msxdev.org/2026/03/14/msxdev25-38-chrono-runner/>

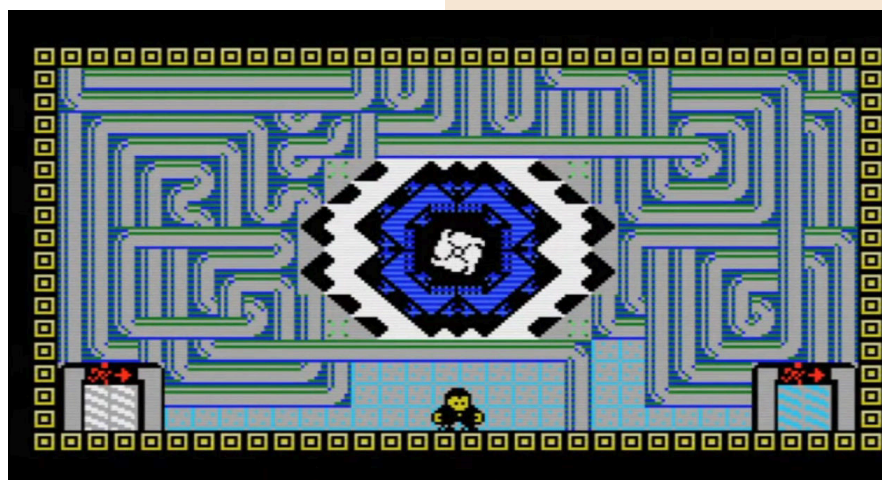
Only 60 minutes (real time) to find four Tachyon Singularities in a massive maze full of rooms located inside a high-security research center in Berlin. A race against time and against surveillance drones, traps, and claustrophobic mazes to be tackled with speed, quick reflexes, and patience... lots of patience!

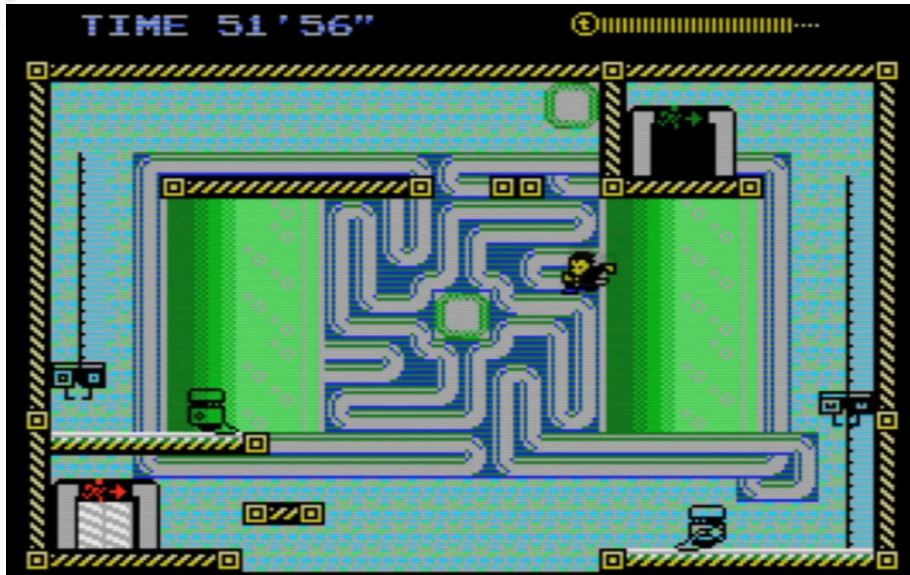
A game where every extra second is a weapon in your arsenal... only those crazy folks at the MSX Italia Association could have come up with this! And it turned out really well!!!

A platformer/puzzle game with a labyrinthine structure and many rooms to explore, yet with the spirit and classic adrenaline rush of games suited for speedrunners.

Under the hood of Chrono Runner lies Aoineko's MSXgl, a well-tested modern C/ASM engine, featuring intriguing, fast-paced graphics and rock-solid level design. All of this is designed to run on an MSX without crazy expansions or SCC... just what we had in our bedrooms back in the glorious '80s.

The game features responsive controls that demand lightning-fast reflexes. Calibrated jumps, reaction times against enemies, and level analysis in general require split-second attention... there's no room for





OUR FINAL SCORE

» Gameplay 95%

Precise and lightning-fast controls that require practice but are incredibly satisfying. The level design is insane!

» Longevity 95%

It's a challenging game that keeps you glued to the controller. It's currently my most-played retro title since the start of the year.



This year's MSXDev is undoubtedly the best edition yet, with a high level of competitiveness! Well done!

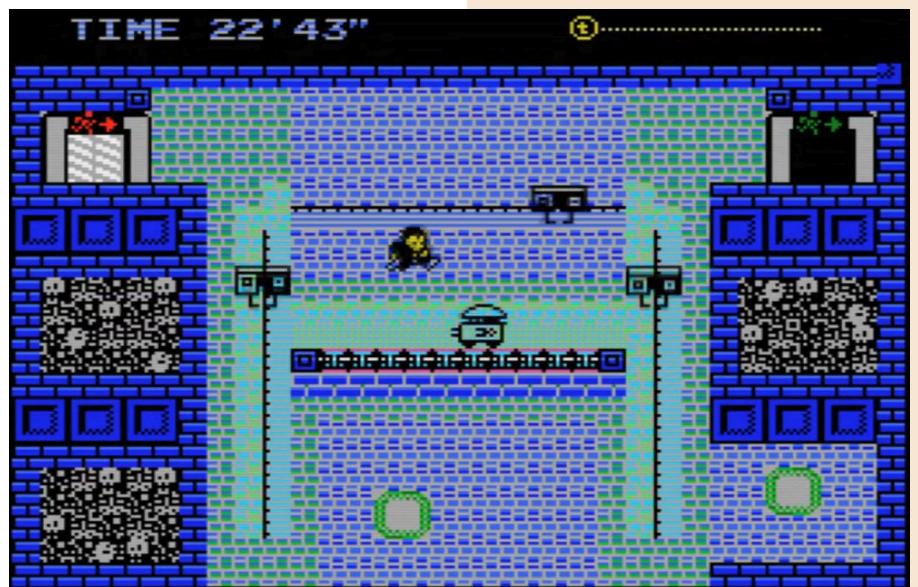
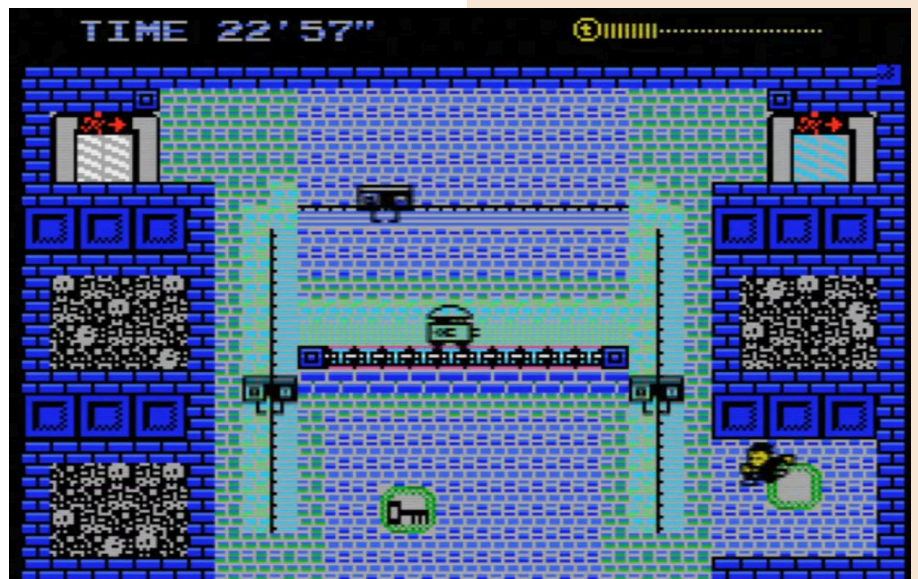
by **Giampaolo Moraschi**

hesitation. Each room speaks to us and clearly tells us the order to regain access, avoid the drones, and reach the exit without wasting precious seconds... you just need to read it carefully in a few seconds.

Warning: this game isn't for everyone, but only for lovers of challenges and those with the spirit of a hardcore gamer... everyone else should look elsewhere for other styles. This is a tough game and requires incredible dedication to complete.



All that's left is to refer you to the final reviews and thank the guys at the MSX Italia Association for Chrono Runner.





BLACK CASTLE II

Year: 2026

Editor/Developer: User0x7f

Genre: Action/Platform

Platform: Game Boy

Website: <https://user0x7f.itch.io/black-castle-2>

This game makes me want to dig out my Game Boy, make myself some tea, and cancel any plans with the excuse, “Sorry, the evil wizard Malakar takes priority... you know how these video game villains are.”

A direct sequel to Black Castle, also by User0x7f, still an action-platformer for Game Boy, but with that nice “same outfit, better cut” vibe. In short, more levels, more weapons, more secrets—more of everything.

The plot immediately feels like a medieval soap opera, just the way I like it. Years ago, the knight Aric had already beaten the sinister sorcerer Malakar in Black Castle. All good? Not a chance. The vile sorcerer survives, licks his wounds in the shadows, and returns more determined than ever, resolved to plunge the world into an eternal eclipse using a ritual in which—a minor detail—he has chosen Lyra, Aric’s sister, as the sacrifice. And so she, as a good, anxious sister sitting in front of the screen, finds herself thinking, “Okay, Aric, put your armor back on, grab your sword, and go fix this family mess yet again.”

Compared to the first episode, this one is definitely a structural upgrade. Gone is the four-stage “snack” adventure; instead, there are five full, meaty levels just the way I like them, each with its own final boss, hidden passages, and items to discover. The game remains strictly “old school” in its controls—A to jump, B to attack, cross to move, Start to begin/pause, Select is just for show—but the feel





is more polished. The level design is “clean” and much more intuitive, with platforms and enemies placed where your gamer instinct would say “it makes sense to put them here,” not where a sadistic designer decided to make you suffer.

And so you find yourself fighting your way through the castle’s corridors, room by room, at a pace that keeps you on edge but rarely feels unfair. The beauty of it is that along the way, you don’t just find hearts and points, but also new weapons, secret passages, and bonus items that slightly change how you tackle the levels. One player mentions a secret area in the second level that seems like a direct homage to Super Mario Bros., and the developer confirms that yes, the inspirations are indeed Mario and Castlevania, complete with breakable bricks scattered about. Basically, if you love jumping at every suspicious piece of wall, this game rewards you constantly. Visually, it’s the “I put on lipstick and my best dress” version of the first game. Super-clean graphics and smooth scrolling with a look that seems straight out of a late-

’90s Game Boy commercial lineup—in the best possible way. Every screen is easy to read, enemies stand out clearly from the background, and the black castle is a place where you feel right at home after just a few minutes. Horrid, gloomy corridors, unholy chapels, dungeons filled with things that want to rip out your last remaining heart. Musically, we’re on the same wavelength, and it’s no coincidence that the soundtrack is by Retro Asobi, who crafts chiptunes that make you nod your head while dodging bullets.

The funny thing is that, despite being a bigger sequel, it doesn’t turn into one of those endless marathons that make you age in front of the screen. The difficulty curve is well-balanced and punishes mistakes while always leaving the feeling that, with another run, you can get further. In short, a real gem for the GB and a saga we sincerely hope continues... maybe with a nice Black Castle 3!

by **Ingrid Poggiali**

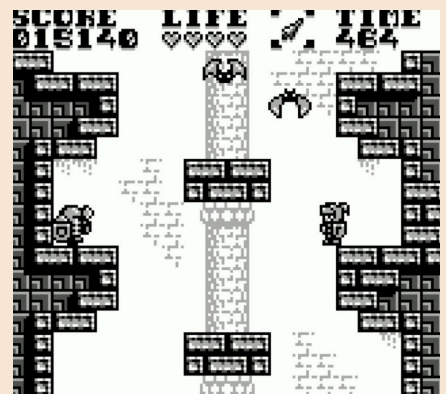
OUR FINAL SCORE

» Gameplay 90%

Solid, classic, clean, and well-structured. Definitely beautiful!

» Longevity 82%

Difficult and punishing, but with that satisfying feeling of... I’ll try again!



DEVELOPED BY YFH
2026

Black Castle

PUSH START

HIGH SCORE
000000





ECHODROME

Year: 2026

Editor/Developer: PixelPlop

Genre: Shoot'em up

Platform: Amiga

Website: <https://pixelplop.itch.io/echodrome-amigamejam2025>

Echodrome was created with a very clear goal: to bring the Amiga back to arcades, without nostalgic filters or technical excuses. Developed entirely in C++ in about six months for AmiGameJam 2025—whose theme was “Return of the Arcade” and which it won!—it is a distillation of minimalist design and an obsession with performance on real hardware, not just a simple “emulator game.”

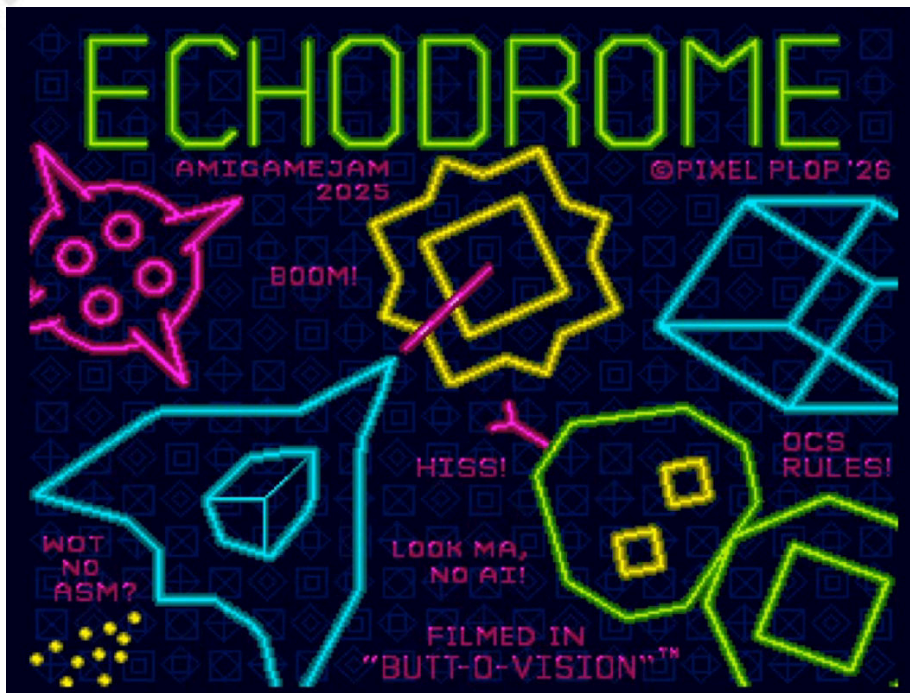
And you know what it does? It runs on basic hardware like a dream. In fact, it requires an Amiga OCS/ECS or higher with 1 MB of memory—specifically 512 KB of Chip RAM and 512 KB of Slow RAM—plus a joystick or keyboard. On this extremely minimalist foundation, Echodrome builds an arena shooter that, visually and mechanically, evokes Geometry Wars and Tron, yet possesses its own distinct identity. The action takes place entirely within “The ECHODROME,” defined by the author himself as “the ultimate digital arena of death and destruction,” a neon grid where each room is a closed arena to be cleared. The objective of each room is simple: eliminate all waves of enemies, avoid direct contact, don't step on the mines scattered across the grid, locate and defeat the arena boss to earn a substantial bonus and access the next area.

In terms of controls, the system is surprisingly robust for an OCS title. In the standard configuration, the game can be played with a single



joystick or keyboard: movement is handled via the joystick or arrow keys, while firing is controlled by the fire button or the left Shift key. By holding down the fire button, the firing direction locks, and the character continues to fire automatically, allowing you to partially separate movement from aiming—a very effective solution for





OUR FINAL SCORE

» Gameplay 92%

Visually flawless and incredible in terms of level and game design. Perfect gameplay combinations with the pad, dual joysticks, and CD32 pad.

» Longevity 92%

A game that will likely never be removed from our Amiga's "drive" and that demonstrates how to make great games even on minimal hardware configurations.

simulating a twin-stick feel on single-button hardware. For those seeking the full experience, Echodrome supports two advanced setups: Twin Stick mode, using two joysticks (one for movement, the other for shooting in eight directions), and a Four Button mode that utilizes the CD32's four-button joypads to emulate a "button-based" twin-stick setup, assigning different firing directions to the various buttons. These options are enabled from the options menu by changing the input style and, in the case of the CD32, by appropriately setting the joystick type in emulators such as UAE, MiSTer, or A500 Mini. For a jam project, the level of attention to peripherals is unusual and clearly aimed at those who want a truly arcade-like experience, even extending to the use of two Monster Joysticks in parallel, as reported by those who've tested it on the A500+.

The arena layout is organized like a sort of maze of interconnected rooms, with multiple paths that the player must "choose wisely," because some rooms are significantly more deadly than others. Each room features different patterns of enemies, traps, and mines, and the choice of path effectively becomes a





strategic element: it's not just about surviving the current wave, but about planning the order in which to tackle the toughest battles to maximize the chances of reaching the boss in decent shape. The presence of the arena boss—which must be located and defeated to earn a “big bonus” and advance to the next block—completes this structure that feels almost “roguelike,” while remaining strictly arcade-style and devoid of meta-narrative progression elements.

Technically, Echodrome makes such efficient use of the hardware that it has impressed the gaming community. In just 880 KB of ADF (version 0.11 distributed on itch.io), the game incorporates an engine capable of handling numerous sprites, projectiles, explosions, and grid distortions without sacrificing readability or fluidity

Visually, the TRON-like aesthetic is Echodrome's trademark. The arena is a luminous grid on a dark background; enemies are geometric and vector-based figures reminiscent of early '80s digital imagery, while explosions and trails leave visual “echoes” that justify the title itself. The use of color is restrained yet meticulously crafted, with a sharp contrast between active elements (enemies, mines, bullets) and the arena's background, ensuring high readability even during moments of maximum chaos—a significant challenge in any arena shooter, especially on a graphics chipset developed long before this type of design emerged.

On the audio side, the soundtrack and sound design take a consistent approach. Tense electronic tracks, crafted to sustain a consistently fast-paced gameplay rhythm, and a selection of effects (shots, explosions, hit feedback) that prioritize impact and clarity over timbral variety for its own



sake. With version 0.11, the developer has further improved the music, adding a game over screen and a high-score entry screen “embellished” with fireworks and dedicated tracks—a sign that the polishing work continues beyond the jam's deadline.

The combination of technical and design choices has produced an “arcade-style shooter” with an authentic arcade feel, rich in extras and meticulously crafted details. Echodrome thus succeeds in the goal it set for itself from the start: not merely to be a tribute to the arcade era, but to act as if the Amiga were still a living platform, capable of hosting a

modern twin-stick shooter on a machine with 1 MB of RAM and a single-button joystick. For those viewing the OCS/ECS through a technical lens, it's a small lesson in efficiency; for those viewing it through a gamer's eyes, it's simply one of those titles that keeps you glued to the screen.

by **Roberto Del Mar Pirazzini**





DRAGON VIEW

Year: 1994

Editor/Developer: Kemco/
Kotobuki System

Genre: RPG

Platform: Super Nintendo

Welcome, role-playing and rarity enthusiasts: today we're going to explore a truly unique sequel released exclusively on the SNES.

Dragon View is the sequel to Drakkhen that, on paper, seems like the answer to all the problems of the first game, yet ends up fixing many of them while losing a good deal of that hypnotic strangeness that made the original unique. Technically, it represents a clear leap forward, and in terms of gameplay, it's much more playable, but as a 1994 Super Nintendo action-RPG, it should be viewed today with a bit of distance. Its predecessor was a European role-playing game published by Infogrames on Amiga and Atari, later ported first to MS-DOS and, in 1994, to the SNES—a fertile ground for this genre but one facing stiff competition from more classic ARPGs and JRPGs. Drakkhen featured unique mechanics, several issues stemming from a rigid gameplay system, and a Western vision ill-suited for the console audience, more aligned with the sensibilities of PC gamers. Fortunately, this sequel, released exclusively for Nintendo's 16-bit console, turns the tables, making profound changes to much of the original style.

The first major difference is the gameplay system. No more four-character parties, semi-automatic controls, or rigid menus. In this installment, the player controls only Alex, a blond warrior from the island of Drakkhen, and the combat system is fully action-based. You move in



eight directions, striking with your sword based on the angle of attack—forward, upward, and downward while jumping—and you have a special attack at your disposal that consumes a portion of your HP. There is also a secondary ranged weapon, the hauza, a sort of boomerang that, on paper, expands your options, but in practice often goes unused because it is less intuitive than the sword. Added to this are quick-use items, primarily the elemental rings of fire, ice, and lightning, which consume magic points but allow you to both clear the screen with area-of-effect attacks and solve small environmental puzzles by breaking rocks, freezing surfaces, or reaching otherwise inaccessible areas. Above all, there is a classic progression system: every defeated enemy and every boss grants experience, the level-up bar below the HP rises in a very clear manner, stats increase

predictably, and enemies drop hearts, stars, and Jade, the game's currency. All of this makes Dragon View infinitely more accessible than Drakkhen, without resorting to pure, shallow action.

The world's structure builds on the basic concept of its predecessor, with the overworld rendered in pseudo-3D using Mode 7, while everything else is on 2D maps. The island the player explores is enormous, divided into easily recognizable regions—plains, mountains, swamps, snowy areas, and desert—and dotted with cities, temples, and dungeons. As the adventure progresses, you acquire maps that reveal increasingly larger areas and clearly indicate where key points are located and how far away they are. In theory, this is a choice that encourages exploration; in practice, two limitations immediately become apparent: the number of





physical obstacles that force you to take long detours and a movement pace that isn't exactly brisk. The island is full of mountain ranges, rocks, and barriers that can only be crossed through specific passages; often you can see a destination just a few steps away, but to reach it you have to take a significant detour. Alex, for his part, isn't exactly lightning-fast, and the game has a penchant for backtracking: it sends you back and forth between cities, temples, and NPCs, requiring you to revisit places you've already been to trigger new events or retrieve key items. Constellation-based teleports mitigate the problem, but they don't come into play until late in the game, by which point you've already covered a lot of ground on foot. The result is an overworld that's conceptually interesting but weighed down by slow pacing and frequent backtracking, which eventually become more of a burden than they should be.

When *Dragon View* confines the player to its interior spaces, however, the tone shifts. The dungeons are expertly crafted. Layouts that are clear but not predictable, a touch of verticality, switches and keys used sensibly, a few detours to collect extras, and, above

all, a good progression in the intensity of the battles. The bosses, in particular, are one of the game's strengths: large sprites that take up a good portion of the screen, clear attack patterns, and a successful blend of observation and reflexes. The towns also work well as hubs for restocking, dialogue, and plot progression, though they suffer from a tonal issue that runs through much of the game: meta-jokes, pop culture references, and deliberately out-of-context gags coexist with moments that aim to be very serious or dramatic, creating a disconnect that doesn't always work.

The save system is one of the most questionable choices. While in *Drakkhen* it was possible to save practically anywhere in the overworld, here progress is tied to healing shrines or wandering healers, which are scattered and not always within easy reach. This decision, combined with slow movement and backtracking, means that an unfortunate death can wipe out not just the last section played, but entire sequences of leveling, exploration, and dialogue. The tension builds, certainly, but the risk is turning a session that went wrong into a frustrating experience, especially in a

OUR FINAL SCORE

» Gameplay 80%

Streamlined, "console-style" gameplay, but terrible save system.

» Longevity 80%

It lasts just the right amount of time, and what you play is enjoyable. Worth revisiting.

game that isn't very long but isn't very short either.

On the technical front, *Dragon View* is significantly more polished than *Drakkhen*. The sprites are more detailed, Alex boasts fluid animation—with his hair fluttering almost as a character in its own right—the NPCs aren't mere mannequins but move their heads to follow the character, and the bestiary offers a good variety of enemies, from snake-men to imps, from giants to scorpions, all the way to reapers and slimy blobs. The bosses are spectacular in size and design, often composed of multiple animated parts. The overworld utilizes Mode 7 with a gradient sky, landscape elements that scale in and out—rocks, trees, ice, walls—and an overall rendering that is much cleaner than the old *Drakkhen*. The interiors of the temples, the forests, and the waterfalls demonstrate a flair for atmosphere that, for a mid-range SNES title, is more than respectable. On the audio front, the soundtrack abandons the jarring new-age sounds of the first installment in favor of a style more classic for the era, halfway between orchestral and rock. Some overworld tracks, several temple themes, and certain pieces from the final phase work well and stick in your head, while others—like the city theme or the music in the final dungeon—come across as flat and forgettable. At least the battles finally have a proper musical accompaniment, and the sound effects, from the Imps' laughter to the bosses'





growls, are polished enough.

The game struggles more on the narrative and atmospheric fronts. The plot stems from a very conventional premise: Alex, a young warrior, rescues his master trapped in a cave, but upon returning, he discovers that his beloved Katarina has been kidnapped by the wizard Argos; behind him lurks Giza, the true antagonist who aims to plunge the world into darkness using a powerful artifact. It serves as the driving force behind the action, but rarely surprises. Giza, in particular, is portrayed as a sarcastic, over-the-top villain who is meant to be charismatic but ends up being more irritating than threatening. The connection to Drakkhen, meanwhile, exists but is treated almost as an afterthought: a text in the temple of Keire clearly mentions a hero and three companions who in the past restored balance to the world, a direct reference to the events of the previous game, but aside from a few mentions of dragons and a handful of other details, the Drakkhen universe is revisited in a very superficial manner. The bizarre mythology, the surreal bestiary, the talking constellations, and that sense of an alien world—which, despite all the flaws of the first game, contributed

to a unique atmosphere—have almost entirely disappeared. Here, we find ourselves in decidedly more traditional action-RPG territory. The ending also deserves a mention, as it represents a rather extreme technical and directorial choice for a SNES cartridge game. After the final battle, Dragon View treats itself to about twenty-six minutes of epilogue consisting of unskippable dialogue, closing sequences, and end credits. It's almost a show of strength regarding how much narrative can be crammed into a ROM, but with the controller in hand, it can become exhausting, especially when compared to the more concise yet emotionally effective endings of other RPGs from that era.

All things considered, Dragon View is a competent action-RPG worth rediscovering, with a solid combat system, generally well-designed dungeons, and technical production that exceeds the average for second-tier titles on the Super Nintendo. At the same time, it's a game that tends to get in its own way: the massive but sluggish overworld, frequent backtracking, the limitations of the save system, a lackluster plot, and the near-total loss of Drakkhen's quirky



atmosphere keep it from reaching the level of the genre's true must-plays. Those who appreciate Drakkhen primarily for its madness will likely find Dragon View "too normal"; those, on the other hand, who were looking for a more playable evolution of the first game will find it here, but must accept certain design choices that, even today, raise more than a few eyebrows.

by **Roberto Del Mar Pirazzini**





ESCAPE FROM PETSCII PLANET

Year: 2026

Editor/Developer: Jimbo

Genre: FPS

Platform: Commodore 64

Website: <https://tinyurl.com/ch356rdm>

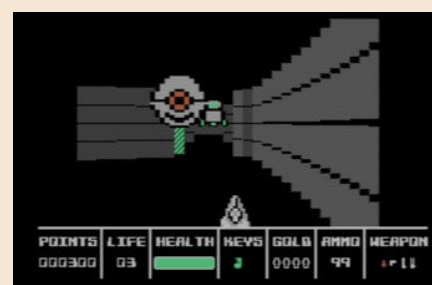
A true ray-casting FPS inspired by Wolfenstein 3D, built on the PETSCII aesthetic and designed to be played and “squeezed” in 2026. Here’s what happens when we load Escape From PETSCII Planet onto our old breadbin. The developer, Jimbo, starts with an idea that’s crazy in and of itself: bringing a first-person engine originally created for the Commodore PET to the C64 (crazy, right?), and then beefing it up until it becomes a full-fledged dungeon shooter, complete with multiple floors, keys, secrets, and a classic “find the exit” loop. The result is a game where we find ourselves stranded on a remote planet, trapped in an underground dungeon, and willing to do anything to survive. We’ll have to explore corridors, locked doors, secret passages, collect weapons, ammo, keys, gold, and find the elevator that takes you higher and higher, all the way to escape. The antagonists are stubborn alien robots that guard the corridors and populate the various levels; each floor is designed as a small labyrinthine arena with its own dose of tension and “mayhem.”

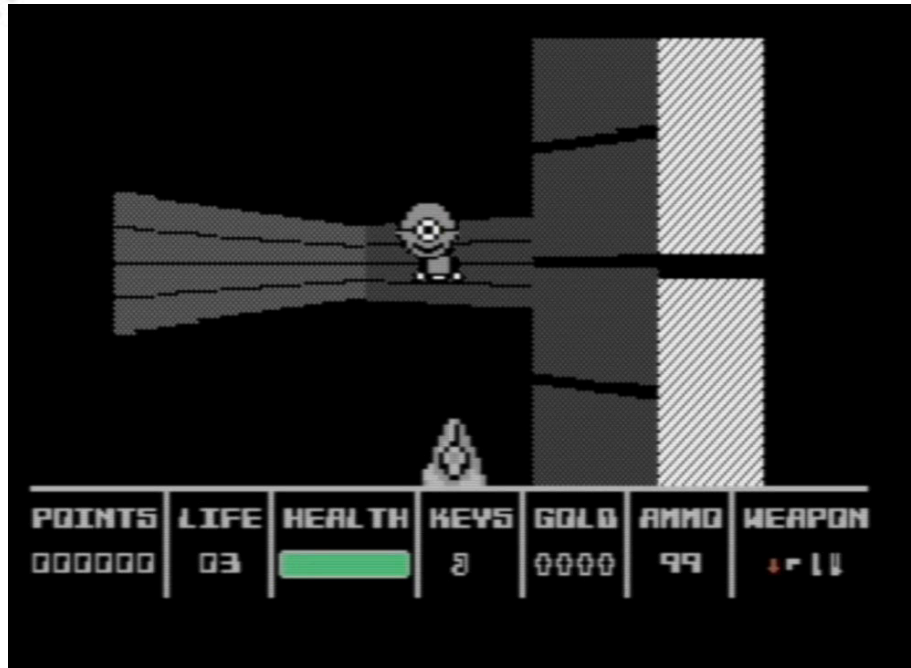
Structurally, the package includes the full game with Campaign A: five floors with over 100 enemies in total, spread across multiple selectable difficulty levels. In addition to the normal mode, there’s an extra-hard mode for those who really want to suffer, a timed Gold Rush mode where the focus shifts to quickly collecting



gold, and a Zeus Mode where you essentially become omnipotent. Progress is automatically saved at the end of each level, a very practical choice that makes the level-based structure more manageable on real hardware, without forcing you to run marathons every time.

The heart of the project is the 8-bit ray-casting engine, explicitly inspired by Wolfenstein 3D and adapted to the limitations of the C64. The style doesn’t aim for faux-realism but embraces the PETSCII syntax, so we’ll find walls, elements, and UI built with graphic characters, creating an ultra-stylized yet legible look that sits halfway between an old roguelike and a primitive raycaster, with a speed that, for our C64, is remarkable. It’s worth noting how the engine manages to deliver a sense of fluid movement while staying within the machine’s limits. On the design front, the developer doesn’t limit themselves





to a sequence of generic corridors. Each floor features doors locked by colored keys, secret walls hiding important items, and, of course, classic dungeon crawler elements. Rounding it all out are scattered ammunition, weapons to pick up, danger zones, and layouts designed to make you run in circles until you start to truly “read” the map.

A significant bonus is the inclusion of a separate Level Designer, provided as a dedicated tool for creating and playing your own custom levels. The author explicitly invites players to design maps and share them with him, transforming the game into a potential platform for community-generated content. In an ecosystem like the C64's, where editors are often external and cumbersome, having a designer specifically tailored for this engine opens up interesting possibilities for remixes, jams, and future releases.

In terms of hardware accessibility, *Escape From PETSCII Planet* is surprisingly flexible. It can be played with a keyboard, mouse, or joystick, as confirmed in promotional posts on C64/Ultimate groups. The distributed file is a classic .d64, compatible with real

hardware and solutions like C64 Ultimate, and the game is available on itch.io on a “pay what you want” basis... technically free, but with the option to support the developer if you're convinced (which I always recommend doing). Overall, *Escape From PETSCII Planet* is a successful experiment in bringing the language of '90s FPS games to an 8-bit platform, without trying to imitate *Doom*, but going straight to the Wolfensteinian heart of the genre. The PETSCII aesthetic gives it a strong visual personality, the campaign structure and extra modes increase its longevity, and the level editor pushes it beyond a simple “one-and-done” game to try once and put away. For those already familiar with raycasters on limited hardware, it's a must-try; for those coming from the more traditional C64 scene, it's a practical demonstration of just how much room for maneuver there still is when you really decide to push 8-bit hardware beyond its comfort zone.

by **Carlo Nithaiah Del Mar Pirazzini**

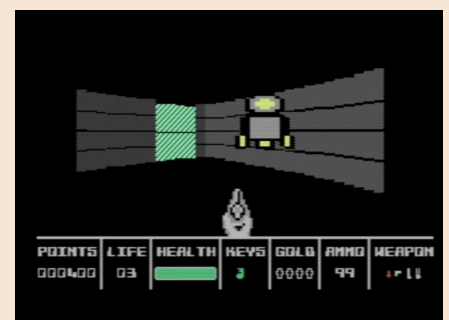
OUR FINAL SCORE

» Gameplay 90%

Fast-paced with good game design.

» Longevity 90%

Well-balanced and enjoyable. The Level Editor is a nice surprise, effectively increasing the product's replay value thanks to the development of new maps and potential Jam sessions.





JUNK RUNNER 64

Year: 2026

Editor/Developer: Team Ultra Rare

Genre: Adventure

Platform: Nintendo 64

Website: <https://drive.google.com/file/d/1MMoweVrkuQkwsFSR0XqIownXE-CicYTV/view>

Junk Runner 64 is a surprisingly ambitious Nintendo 64 homebrew. An open-world adventure/running game that makes excellent and sensible use of the hardware, both in terms of map scale and game design. In the game, you play as Jax, determined to save his sick brother Samson by repairing the village's machines with scrap parts scattered throughout the world. Each machine requires at least one component. The first is nearby, while the others are scattered across distant areas; the robot in the village center provides maps and directions, making exploration guided yet not mundane. After repairing the hoverbike, the game really begins. We can explore a vast open world divided into three macro-zones—the river to the southeast, sand dunes to the north, and a deserted city to the southeast—each containing parts for a specific vehicle, reducing repetitive backtracking. Only the final item has parts scattered throughout the world, encouraging us to use the map more systematically.

Circuits with checkpoints are scattered across the map. Some tracks require one to three full laps, and only one of these hides a crucial part for the final repair, while the others reward you with 20 “nuts.” Nuts are the game's currency and are used exclusively to upgrade the bike:

- On-demand speed boost
- Scanner to locate nearby nuts

- Quick teleportation to key points on the map

You only need 50 to buy everything, so “grinding” is limited! The focus remains on exploration and discovery rather than compulsive collection. The map reveals itself as we explore, but we can also find hidden maps that uncover entire sections at once, speeding up navigation. A first “blind” run takes about an hour, but once you learn the layout of objects and paths, the game becomes much faster to complete.

Technically, Junk Runner 64 is remarkable! An open world of such scale that it could be considered something even many retail titles of the era hadn't dared to attempt. The ROM weighs in at about 16.2 MB, a sign of significant use of graphical and logical resources on hardware notoriously limited in terms of memory





and bandwidth.

The hoverbike's handling is one of the game's most successful elements. It bounces wonderfully over the terrain, takes advantage of the natural contours of the ground, and the whole experience is very enjoyable (even just cruising around). From a level design perspective, the three main zones are diverse enough to avoid the feeling of a flat wasteland: between the river, dunes, and city, the scenery changes and keeps the player engaged. Graphically, the game is top-notch for a homebrew title. The character models evoke a "cleaned-up" version of the 3D action style of the era, with characters resembling an improved version of certain well-known 3D platformers. Although the map is technically vast and partly empty, the layout of the elements ensures the environment doesn't feel repetitive: there's always something to look at, and the transition between areas is distinct enough to keep curiosity alive. However, there are a few technical "issues." The village area suffers from noticeable slowdowns that also affect the music, making the experience somewhat sluggish at that point. On some occasions, you risk falling through

the geometry; however, there is a failsafe that immediately brings the character back to the surface, so the bug remains annoying but doesn't ruin the game. There is also a crash, with the potential to lose all game progress.

These issues are typical of a very ambitious project for the hardware, but mitigated by the presence of smart fallbacks (such as automatic recovery when falling off the edge of the world). Junk Runner 64 was developed by Team Ultra Rare (James Lambert, Pyroxene, SapphireOnze, Beta_dynast, Terzdesign, Kaelin) for the N64brew Game Jam 2025, released on February 2, 2026, and won first place in the competition. The ROM and source code are publicly available (downloadable ROM and code on GitHub), making the project an excellent learning resource for homebrew developers interested in open-world games and vehicles on the N64.

For those interested in the homebrew scene or N64 development, this is a title worth studying as well as playing.

by **Roberto Del Mar Pirazzini**

OUR FINAL SCORE

» Gameplay 88%

A vast world to explore, even if it has few distinct areas. The control scheme is engaging and never boring, encouraging you to play and explore, but also to wander around just to relax!

» Longevity 80%

There are several bugs that undermine the product's "durability." Nothing extreme, but they are annoying.





SOUL BLAZE

Year: 2026

Editor/Developer: Rogério Perdiz, Subin Karkani

Genre: Adventure/Action

Platform: Sega Dreamcast

Website: <https://darkpath.itch.io/soul-blaze>

Soul Blaze is a new project/minigame for the Dreamcast that showcases, with surprising technical ambition, what the Spiral Game Engine is capable of today on actual SEGA hardware.

It is designed as a fast-paced minigame, not as a full-fledged game, and was created explicitly to demonstrate the “power and flexibility” of Spiral on the Dreamcast. It was created for the Dream Disc '25 jam and distributed as a CDI image and a ZIP package, so it can be used on both real consoles and in emulation.

In the game, we play as Rhyo, a young martial artist searching for a rare “flownder” for a personal mission, but this narrative framework primarily serves to provide context for an explorable village and a series of battles against mechanical enemies. Everything is designed to showcase fluid movement, collision handling, and the engine’s graphical rendering.

Technically, the most important aspect of Soul Blaze is that it runs on the Spiral Game Engine, a engine currently under development specifically for the Dreamcast. The developers confirm that Spiral already boasts “quite a lot of features,” but it is not yet ready for public release; the goal is a free release, but only once it is deemed sufficiently stable. This is because there are several issues and some glitches

that still make it “unreliable” (especially in emulation).

These clues point to an engine that works quite closely with the hardware, featuring sensitive timing and video pipelines—a hallmark of projects aiming to maximize performance on a specific machine like the Dreamcast.

In terms of gameplay, Soul Blaze offers a mix of village exploration—with NPCs to talk to for clues about the Flownder—fist-and-kick-based combat with combos against waves of mechanical crabs infesting the area, and fairly complex yet enjoyable boss battles.





The pace is admittedly fun but suited for more skilled players. The focus is on the responsiveness of the controls and the engine's handling of multiple entities on screen rather than on a lengthy progression. The goal is to allow the player to quickly experience fluid animations, graphical effects, transitions, and combat mechanics. The art and visual design are by Rogério Perdiz, and it's truly a work of high quality and technical care, with a strong visual identity in the 3D environments, colors, and village designs. The characters all share a consistent style, particularly the mechanical crabs and the boss, which combine arcade aesthetics with modern details.

The fact that some emulators have issues with shadow volumes and FMVs indicates that non-trivial graphical features are being used—features designed primarily for the actual hardware—and that Soul Blaze is designed to “push” the Dreamcast a bit, not just as a visual tech demo.

The soundtrack is composed by Subin Karkani, which fits the game's setting perfectly. Together with the graphics and animations, it helps give the product an above-average

audiovisual profile for a homebrew jam project.

For those interested in Dreamcast homebrew and the technology behind new engines for retro consoles, Soul Blaze is less “just a game” and more a well-crafted technical showcase, offering a glimpse of what the mature version of Spiral could achieve in future indie projects for the Dreamcast. We hope to see greater compatibility with the emulation world and perhaps an even further expansion of the game world.

For now, we can only say “well done” to the team and give the title our initial rating.

by **Roberto Del Mar Pirazzini**



OUR FINAL SCORE

» Gameplay 80%

Precise controls and good game design, with great technical attention to detail.

» Longevity 70%

There is still very little content, and there are several bugs, especially in emulation.





ZEZITO AND THE LOST STARS

Year: 2025

Editor/Developer: Iapetus

Genre: Platform

Platform: Acorn Electron 32K

Website: <https://iapetus2019.itch.io/zezito-and-the-lost-stars>

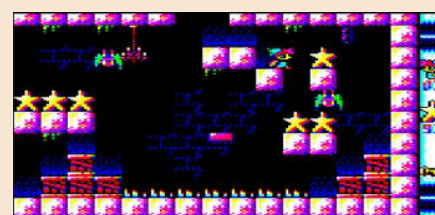
“Zezito and the Lost Stars” is a platformer for the Acorn Electron 32K that explicitly aims to showcase the hardware’s capabilities, combining a vibrant aesthetic with accessible gameplay that allows us to reevaluate the Acorn Electron, historically treated as the poor cousin of the BBC Micro.

The premise is simple but spot-on: the witch Malandra steals 99 stars and locks them in her castle; if no one retrieves them, the night sky turns into a blank CRT screen. Zezito takes advantage of the fact that the witch is at the “annual witches’ convention” (crazy, right?) and sneaks into the castle like a little 8-bit Indiana Jones, determined to bring light back to the firmament one little star at a time. The result is a labyrinthine platformer with interconnected single screens, where each room is a Manic Miner-style micro-puzzle room, but stripped of Miner Willy’s gratuitous nastiness and closer to the “challenging but not mean” vibe of a good ’80s computer game.

One of the most surprising things is how much the title “unleashes” the Electron’s potential. It features some of the best graphics ever seen on the machine, with fully masked sprites (goodbye to those hideous EOR artifacts when two sprites overlap), fluid movement, and a bold use of the color palette. To put it simply, it almost feels like

watching an Electron trying to be the 8-bit Sonic of its catalog—not in terms of speed, of course, but in terms of visual impact compared to the system’s historical standard. On the technical side, Iapetus plays dirty—in a good way. To manage to fit Mode 2, masked sprites, and a full game into 32K, it resorts to a very “demoscener”-style trick. It uses about 8 KB in hidden screen space—100 display lines above the playable area—to house code and data. No backbuffer, no luxury, just a sprite engine refined over years of experimentation, which he himself calls “the best I’ve written so far.” It’s a bit like watching those C64 games that exploit the overscan border or raster splits to squeeze more stuff where it shouldn’t fit: here, the Electron receives the same treatment of technical respect.

The controls, in true British 8-bit tradition, have their own little quirks. You’ll notice that if you try to jump with a block right above





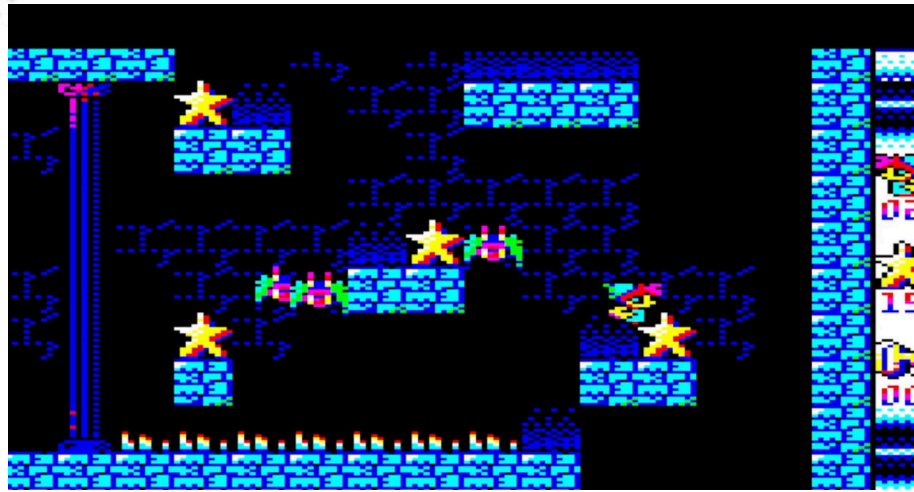
OUR FINAL SCORE

» Gameplay 80%

It takes a bit of getting used to, but it's quite well-made and offers a good variety of levels. The graphics are very appealing!

» Longevity 80%

A good middle ground. It's not extremely difficult, but it's not a walk in the park either.



your head, the horizontal movement behaves “strangely,” and that jumps between rooms—especially when moving from the top of one screen to the bottom of the next—require a bit more careful timing. The developer responds very honestly. The first behavior is a deliberate compromise, a small price to pay to make the rest of the engine work; the second is on his list of potential future improvements. It's the same kind of love-hate relationship many of us had with the jumps in Jet Set Willy or Monty on the Run: a bit “quirky,” but part of the game's character.

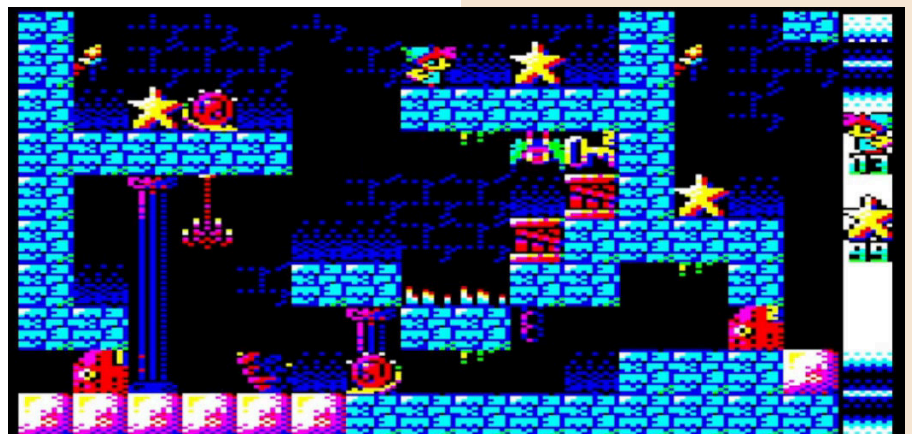
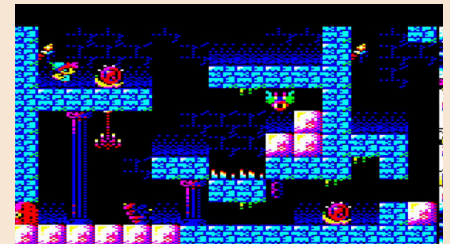
The overall atmosphere is that of a “cute but mean” platformer. It's not “joystick-killer and pixel-perfect,” but it's no walk in the park either. The spiky snails and bats do their dirty work, but the tone remains lighthearted, almost like an afternoon cartoon.

On the distribution front, Zezito released it in 2025 on itch.io with an .ssd disk version and two .uef tape variants (including a fast-load option so you don't go crazy waiting for it to load), all perfectly compatible with real hardware or emulation.

Ultimately, “Zezito and the Lost Stars” is a small indictment of the idea that the Electron was destined only for school-based BBC clones. Here, it feels like seeing what would

have happened if, back then, someone had decided to treat it as a mini-C64 rather than as the poor man's version. It's a game that sits halfway between a magazine-style platformer and a tech demo with soul, and it works precisely because it manages to make you forget the “minor machine” label and leaves you with just one very simple question: “ok, what else can you pull out of this thing?”.

by **Giampaolo Moraschi**





BURGERTOM

Year: 2026

Editor/Developer: Swap d0

Genre: Platform

Platform: Atari Jaguar

Website: [https://](https://swapd0.itch.io/burgertom)

swapd0.itch.io/burgertom

BurgerTom is a new homebrew port of BurgerTime for the Atari Jaguar, created by Spanish coder swap d0 (already known in the scene for other ports and remakes such as Jumping Jack), designed to let players rediscover a great Data East classic on Atari's 64-bit system with technical features suited to the hardware and a feel as faithful as possible to the original arcade game.

The story remains the same as in the classic arcade game: playing as the chef (here renamed Tom Tomatoes), you run across platforms and stairs across multiple levels, attempting to assemble giant hamburgers by stepping on ingredients to make them fall onto the plate below, all while avoiding a small army of "hostile foods" patrolling the screen.

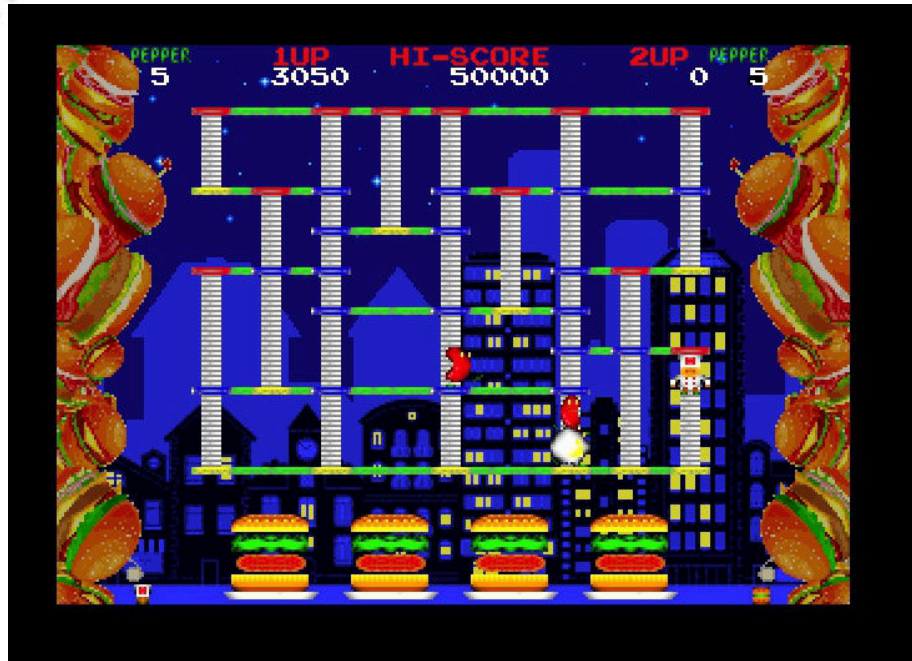
Each completed level contributes to the construction of the burger, with the weight of the ingredients capable of crushing or knocking down opponents below, recreating the classic dynamics of BurgerTime. The pace is typical of arcade games from the golden age of the 1980s, featuring patterns to memorize, precise trajectory management, and a difficulty level that rises rapidly after the first



few levels.

On the Atari Jaguar, BurgerTom is presented as a standalone ROM (approximately 265 kB) to be loaded onto Skunkboard or JagGD, so it is clearly designed for the actual hardware and for those familiar with the tools of the Jag scene.





OUR FINAL SCORE

» Gameplay 85%

Perfect in its ruthlessly arcade-style nature.

» Longevity 85%

It's a pleasure to play a round every now and then.

The author's description promises "enhanced performance" and smooth controls, emphasizing input responsiveness and vibrant graphics that evoke the original arcade cabinet, with saturated palettes and sprites that remain clearly visible even on the RGB outputs typical of modern Jaguar setups.

A promise kept 100%. We are looking at a solid title that stays true to the original game.

The developer's goal is to fill a gap in the official catalog by offering a home version of a classic, with all the convenience of a ROM loadable onto a flash-cart, plus polished presentation and flawless real-hardware controls. If you love BurgerTime and are hungry for new games on the Jaguar, this is one of the conversions worth having. but vibrant library of "new" titles for the Jaguar that draw on the icons of the '80s.

by Alex "LEX" Fridman





TONY DO IT!

Year: 2026

Editor/Developer: Monochrome Production

Genre: Puzzle game

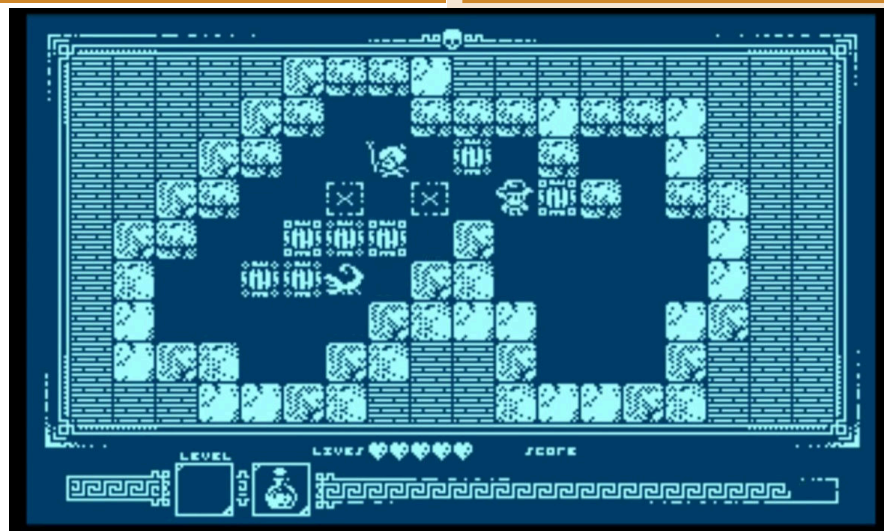
Platform: Commodore 64, Atari 8bt

Website: <https://monochrome-productions.itch.io/>

Tony Do It! is the third installment in the series developed by Monochrome Production (following Tony Montezuma's Gold and Go! Tony Go!), inspired by the mechanics of Sokoman. The story picks up where Montezuma's Gold left off, with the good old explorer finding himself trapped in a room where the exit is blocked by crates that must be strategically placed on designated crosses.

The mechanics are the same as Sokoban's, but enhanced with innovations that set it apart. Tony can not only push the crates but also pull them using the joystick's Fire button, opening up more fluid and creative movement possibilities to solve the levels. This "push-pull" duality allows access to otherwise unreachable areas and enables the creation of complex configurations, requiring careful planning from the very start. Complicating matters are scorpions and wandering natives patrolling the screen. If Tony touches them, they quickly drain the available time, introducing an element of urgency and risk that turns every move into a critical decision.

One of the most appreciated features is enemy management, which adds tactical depth without weighing down the experience. Instead of passively



avoiding them, players can trap them in enclosed spaces by pushing crates around them. Once trapped, they die after a few seconds, clearing the space and allowing you to progress.

This strategic approach rewards observation and patience, making the puzzles not only logical but also dynamic. The levels are compact and progressive, with challenges that gradually increase in difficulty, yet always remain solvable without





OUR FINAL SCORE

» Gameplay 85%

Refreshed Sokoban mechanics with solid strategy to employ.

» Longevity 75%

It shines in short games... in longer sessions, it can feel repetitive.

excessive frustration thanks to built-in cheats like F3 for infinite time or F5 to skip a level—useful for both beginners and those who want to experiment with alternative solutions.

The pixel art graphics, crafted by Rafał Dudek, make the most of the console's color palette with vibrant sprites and fluid animations, creating an adventurous atmosphere that evokes the glory days of the '80s. Poptimo's code ensures a stable frame rate even during the most chaotic moments, while Manganoid's soundtrack, featuring catchy jingles and crisp sound effects, rounds out an audio package that won't disappoint fans. Released in March 2026 and distributed for free on itch.io in .prg format, it is compatible with both original hardware and emulators such as VICE on the C64 or Altirra on the Atari 8-bit, making it accessible to a wide audience without financial barriers.

Perhaps the mechanics might become tedious over time, but it remains a solid title suitable for puzzle game enthusiasts on both platforms.

In any case, this installment stands out for its immediacy. It's a perfect title for short but satisfying sessions, ideal for fans of retro puzzle-solving who appreciate innovation without drastic changes.

by **Giampaolo Moraschi**





WINGS OF FEAR

Year: 2025

Editor/Developer: Roartj

Genre: Shoot'em up

Platform: Amiga AGA

Website: <https://roartj.itch.io/wings-of-fear>

Wings of Fear is a personal dream come true, crafted with artisanal patience and refined together with the community until it became a vertical shoot 'em up capable of standing shoulder to shoulder with the classics that inspired it. Born from the passion of Roar "Roartj" Tjørhom and developed with the Scorpion Engine for AGA machines, the game puts us at the controls of a white and gold Sopwith Camel in the midst of World War I, launched on a secret mission to find a hidden German base amidst leaden skies, bombed-out cities, and landscapes that alternate between historical accuracy and an almost "weird war" atmosphere.

From the very first minutes, it's clear that this isn't just a nostalgic exercise. Wings of Fear plays out like an '80s/'90s arcade game, with a structure punctuated by levels, bosses, special missions, and bonus stages, but it does so with a modern sensibility in pacing. The six main missions, accompanied by two challenging bombing runs and a series of bonus stages, carefully balance intense dogfights, ground-bombing phases, and moments where the game lets you catch your breath, giving you the chance to replenish your lives and power-ups before the next wave of anti-aircraft fury. The bonus stages aren't just extras, but an integral part of the game's balance: if you don't make good use of these "safe zones," the bosses—especially the multi-phase ones with guided missile patterns—quickly become





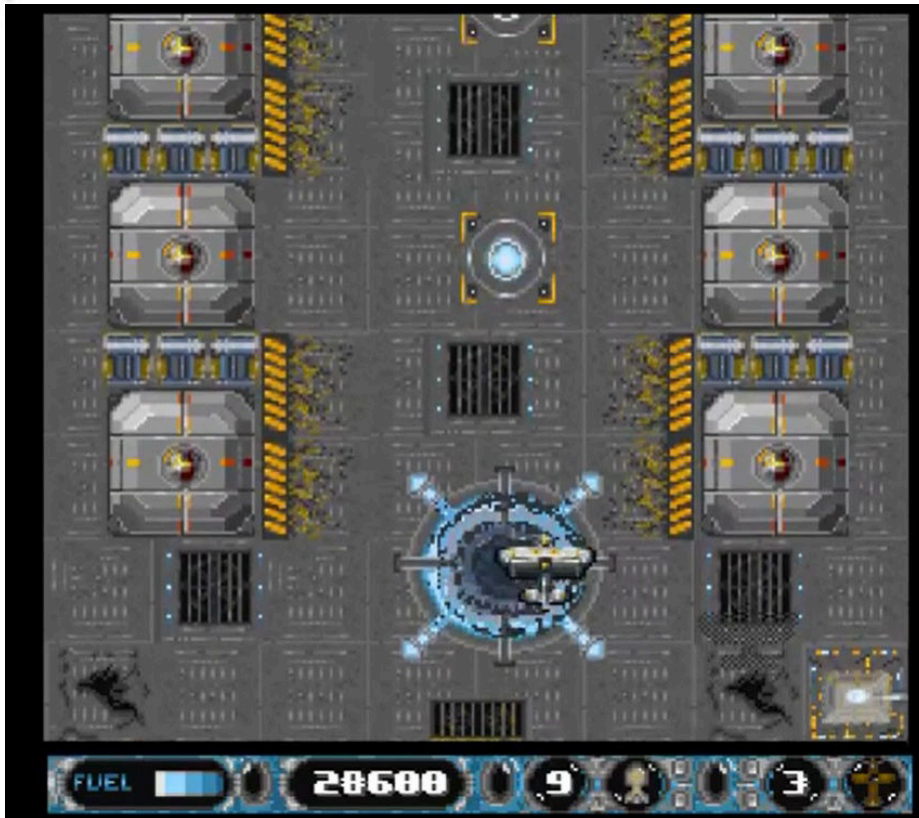
OUR FINAL SCORE

» Gameplay 80%

A very '80s vibe and solid gameplay accompany you throughout the adventure.

» Longevity 75%

A title that doesn't give anything away in terms of gameplay. It requires attention and knowledge of the opponents' patterns.



insurmountable obstacles.

Managing pickups is one of the keys to gameplay. Red barrels serve as a lifeline for fuel, blue ones heal damage and allow you to upgrade your main weapon, and every decision to dive for a power-up—perhaps in the middle of a barrage of shots—is a constant game of risk and reward. Unlike many of today's “comfort” shooters, Wings of Fear isn't afraid to make you suffer. You can feel this from the very first boss, capable of throwing even the least-trained players into a crisis. At the same time, however, the difficulty is never gratuitous; there are precise strategies, such as using the lower part of the screen to “knock out” homing missiles, and those who take the time to read the trajectories and study the layout will find, in return, the satisfaction typical of well-designed arcade titles.

Visually, Wings of Fear features 78 on-screen colors that are used not to show off saturation, but to create clear, atmospheric environments such as stormy seas, ravaged countryside, and military installations, leading to increasingly strange and unsettling scenarios as you venture deeper into

the heart of the enemy base. Sprites and backgrounds, created with modern tools like Aseprite, Resprite, and Procreate, are all hand-drawn; the developer has consciously chosen not to use AI-generated graphics, and this stylistic consistency is evident. The game has the feel of a “real,” almost commercial production, closer to a mid-'90s store-shelf title than an improvised





hobby project, thanks in part to the animation work on the enemies and the varied explosions for different ground targets.

The soundtrack is the other major pillar of the experience. The main theme by Márton Imre Gáspár introduces the missions with an epic tone reminiscent of a pre-takeoff briefing, while in-game tracks by Frank "Curt_Cool" Ewil Larsen alternate between fast-paced rhythms and darker tones, mirroring the rising tension of the bombing levels and boss battles. It's not mere background music, but a soundscape that captures the quintessential vibe of the Amiga scene: more than one review highlights it as one of the game's strengths, so much so that it's often cited in videos as an element contributing to that sense of "modern revival" perfectly in line with the spirit of the Commodore machine. The Scorpion Engine is utilized in fetchmode 3, with the play area shifted to the right side of the screen to squeeze extra performance out of the AGA chipset: a choice that may seem like a minor aesthetic quirk to some, but which results in smoother scrolling and greater frame rate stability on machines equipped with sufficient Fast RAM. To ensure those who prefer a more traditional layout aren't left behind, version 4.8 also offers a build in fetchmode 1, with a centered screen and a focus on compatibility, albeit with some compromises in terms of speed in more crowded situations. The game requires an Amiga AGA with at least 2 MB of Chip RAM and 2 MB of Fast RAM, but performs best on configurations with 4 MB of Fast RAM and a 68040 CPU or higher, whether on real hardware, FPGA, or emulation; not surprisingly, the packages curated by AmigaVault and associates include WHDLoad versions, hard disk installations, CD32 support, and even dedicated configurations for the A500 Mini, the result of extensive testing to avoid slowdowns and memory issues.

Of course, it's not a game for everyone. The high difficulty (especially if tackled without making full use of the bonus stages and without studying the boss patterns) can discourage players less accustomed to the genre, who are used to more forgiving shooters or ones loaded with continues and saves. Similarly, the need for decent AGA hardware or a well-configured emulator makes it less "plug-and-play" compared to other lighter homebrew titles, and the off-center screen layout of the early versions may turn off purists who prefer a centered layout. But these are compromises born of deliberate design and target audience choices. Wings of Fear is clearly aimed at those who want a solid arcade experience, in line with the great names of the past, and willingly accepts that the entry barrier is a bit higher in order not to compromise its identity.

by **Giampaolo Moraschi**





NEW GAME

BUBBLE BUDDIES

Year: 2026

Editor/Developer: geon3

Genre: Platform

Platform: Commodore 64

Website: <https://geon3.itch.io/buddies>

You never get tired of talking about a game like Bubble Bobble, even if you risk being repetitive—and perhaps you shouldn't, given that it continues to thrill us with its classic version, which is still played today after nearly forty years.

Produced in every possible form for almost every computer and console, as if that weren't enough, new titles are now being released that aren't just new chapters, but expansions (if we want to call them that) or reimagined versions to create new levels, new challenges, and add more gameplay!

Following the Remastered version and The Lost Cave, here comes Bubble Buddies, another title ready to thrill us with another hundred new levels and curiosities from the Bubble Bobble universe.

The game offers the challenge of a full hundred new levels to explore, packed with countless enemies, plus various gadgets and fruits to collect—all bonuses waiting for us in every level.

To make it even more interesting and full of challenges, enemies don't appear in the later levels or in order of increasing threat level, but almost immediately. So you'll encounter those cute bottle-shooting goblins right from the very first screens.

The levels are almost all structured

on simple platforms, fortunately not inaccessible—in fact, perfectly reachable in single-player mode—but personally, I find this a bit of a shame, because I enjoyed seeing and playing the levels with the text, even if some were of unprecedented difficulty.

The graphics quality is exactly the same as the Remastered version, no ifs or buts, and so is the soundtrack (which perhaps could have been changed).

When I played it for the first time, I told myself, "Yeah, I'll just do about ten levels, then I'll go to bed," but instead I found myself facing the final boss of the hundredth level! And please, don't start wondering who it is—just take your time and enjoy this new masterpiece. The boss is always there waiting for you, even though he knows he'll be defeated for the millionth time.

You'll have the usual nine lives at your disposal, more than enough, I'd say, to make it all the way through, not to mention the various umbrellas, which seem to be quite plentiful in this version... And those aren't the only useful shortcuts.

As for the gameplay, there's probably no need to say more because it's simply the best of all the versions!

by **Daniele Brahimi**



OUR FINAL SCORE

» Gameplay 95%

The best, hands off.

» Longevity 90%

Even after the 100th level, an endless desire that lasts a lifetime.





MIRACLE BOY IN DRAGON LAND

Year: 2026

Editor/Developer: SamSoft

Genre: Avventura/Rpg

Platform: Atari ST

Website: <https://samsoft.itch.io/miracle-boy-in-dragon-land>

Miracle Boy In Dragon Land is pure magic and excitement. It is an action-adventure/platformer that began as a grand love letter to Wonder Boy in Monster Land, but has established itself as one of the most ambitious and technically remarkable projects in the modern scene on this platform. The game is set in Miracle Land, a fantasy kingdom under threat from the evil Dragon Count. The protagonist is a young hero on a quest to liberate the kingdom by traversing territories filled with monsters, puzzles, shops, NPCs, and mini-games. The structure echoes that of the action-RPGs from the original series, featuring interconnected maps, recurring bosses, upgradeable equipment, and a world that feels more like a small-scale adventure than a simple sequence of linear levels.

Technically, Miracle Boy is developed entirely in 68000 assembler for the Atari 1040STf/STE, with smooth



scrolling and performance that make it a true “technical showcase” for the machine. The graphics make good use of the ST’s color palette and resolution, and the DMA-SC audio on the STE delivers music and sound effects considered top-notch by the platform’s standards. Personally, I find the use of color palettes, the animations, the variety of characters included, and the incredible musical accompaniment exceptional.





OUR FINAL SCORE

» Gameplay 95%

Beautiful, responsive, with great aesthetics and excellent level design. It's the Wonder Boy chapter that was missing on ST.

» Longevity 95%

The platformer/adventure/RPG mix is perfect! There's so much to explore, discover, fight, and see.

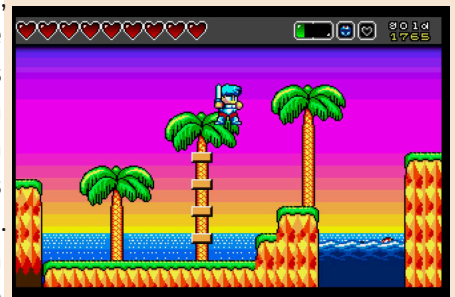
Development took nearly four years and was completed on April 5, 2026; the final version is scheduled for mid-May 2026, with a limited physical edition already available for pre-order.

Wonder Boy in Monster Land: The Arcade-Adventure Blueprint

To understand the essence of Miracle Boy In Dragon Land, it is essential to look at its primary inspiration, Wonder Boy in Monster Land. The 1987 Sega game (System 2 arcade, later ported to numerous home platforms) represents a crucial transition for the Wonder Boy series. It shifted from the purely arcade-style platformer of the first installment to a hybrid formula combining action, platforming, RPG elements, and a pseudo-open-world structure featuring towns, dungeons, and multiple bosses. This was an incredible success that was reaffirmed in the splendid Wonder Boy: Dragon's Trap for the Master System. In Wonder Boy in Monster Land, the protagonist traverses a kingdom divided into interconnected areas, featuring villages where talking to NPCs yields information, shops where you can buy armor, boots, swords, and spells, and dungeons where you face increasingly challenging enemies, collecting coins and items that unlock better equipment. All of this is filtered through an arcade-style framework: limited time, finite lives, end-of-level bosses with specific attack patterns, and a strong incentive to learn routes, secrets, and coin-farming spots.

Miracle Boy In Dragon Land adopts this very hybrid structure. The idea of moving through a shared realm, encountering "colorful" NPCs who provide clues or services, entering buildings that serve as shops, inns, or key progression points, and facing bosses at specific locations on the map is directly inspired by Wonder Boy in Monster Land. The fact that the adventure is structured into 19 levels with 5 bosses, includes mini-games and puzzles, and is based on equipment upgrades places Miracle Boy in the same family of "arcade-style action-adventures" of which Monster Land is a progenitor.

The choice of a colorful, fantasy-inspired, and relatively bright aesthetic—more fairy-tale-like than dark—also falls within the legacy of Wonder Boy in Monster Land, which even back then stood out from many other action-RPGs due to its bright, cartoonish visual style. Miracle Boy carries on this legacy on the Atari ST, a platform that at the time never received a conversion as polished





as this one.

Wonder Boy: The Dragon's Trap and the Modern Reimagining
The other major influence cited for Miracle Boy In Dragon Land is Wonder Boy: The Dragon's Trap, the modern remake of the classic Wonder Boy III: The Dragon's Trap (Master System / PC Engine), reimaged by Lizardcube and DotEmu in 2017.

The Dragon's Trap takes a game that was already a "pioneering Metroidvania"—featuring protagonist transformations, extensive backtracking, abilities that unlock new areas, and a structure far more open than a linear platformer—and reinterprets it for a contemporary audience with meticulous attention to presentation, animation, and smooth controls, without, however, distorting the original design. It retains the interconnected map, the iconic bosses, and the transformations (dragon, mouse, piranha, lion, hawk), but modernizes the interface, refines the combat feel, and adds quality-of-life features in line with modern standards. Miracle Boy In Dragon Land draws on this approach rather than simply replicating the original arcade model. It doesn't just "copy" Wonder Boy in Monster Land, but adopts the same philosophy as The Dragon's Trap: taking an '80s formula and reimagining it for today, preserving the core mechanics and progression structure while refining fluidity, clarity, content density, and narrative coherence.

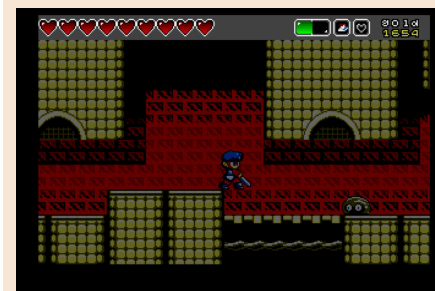
The decision to develop the game entirely in 68000 assembler to achieve scrolling and responsiveness on par with console productions, the focus on the DMA soundtrack, the integration of puzzles, mini-games, and "slow" moments between action sections, and the care taken in the worldbuilding of Miracle Land all point in this direction: using the Atari ST hardware to build an experience that, while running on a mid-1980s computer, approaches—in philosophy—the modern reinterpretations of classics, of which

The Dragon's Trap is a prime example. In this sense, Wonder Boy in Monster Land provides Miracle Boy with the basic structure (kingdom, villages, party, bosses, arcade-RPG progression), while Wonder Boy: The Dragon's Trap suggests how to make it feel "alive" and cohesive in 2026: tight controls, a well-balanced rhythm between exploration and combat, attention to aesthetic details and music, and a world-building approach that invites the player to view Miracle Land as a place to explore, not just a sequence of stages to overcome.

Conclusion

In this context, Miracle Boy In Dragon Land stands not only as a great tribute to Wonder Boy in Monster Land, but also as a successful attempt to apply, on the Atari ST, the same sensibility that made Wonder Boy: The Dragon's Trap a benchmark remake. It inherits the blend of platforming, action, and RPG progression, the attention to atmosphere and worldbuilding, and the desire to make a structure born in the 1980s feel "new," while remaining faithful to its arcade-adventure spirit.

by Carlo Nithaiah Del Mar Pirazzini





NEW GAME

CURSE OF THE PHARAOH

Year: 2026

Editor/Developer: Odin Strabo Jensen

Genre: Platform

Platform: Commodore 64

Website: <https://furidk.itch.io/curse-of-the-pharao>

Following the monthly releases that have become a fixture even forty years after the Commodore 64's debut, here we are again exploring Egyptian tombs... And more.

We won't be talking here about yet another Indiana Jones or Lara Croft clone, or Rick Dangerous, if we want to stay on the Commodore theme, but rather about a fun new platformer set in an Egyptian tomb full of dangers and curses.

The Curse of the Pharaoh is a platformer set in an Egyptian tomb with ten levels packed with suspense, mummies, and pinpoint precision.

The goal of the game is to collect a set number of golden stones scattered throughout the level and make your way to the exit—no easy task given the multitude of enemies, such as mummies, snakes, and you name it, all ready to stand in your way.

On top of these, there are other hazards like spikes and spears popping up from the floor. Just to make sure nothing's missing...

Ten levels are few for my taste, and the difficulty isn't very high—so much so that you'll finish it by the fourth or fifth playthrough. However, the concept and setting are really nice, and they bring to mind the exploits of archaeologists on the big screen and archaeologists on consoles.

On the tenth level, the Pharaoh—or

what's left of him—awaits us, guarding the final room that will lead us to the exit and victory.

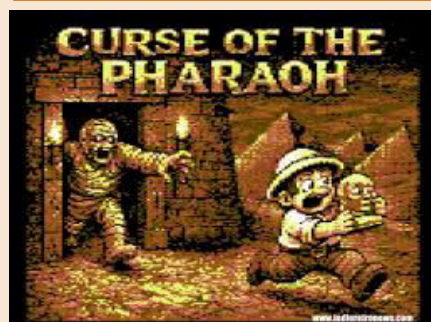
The only action you can perform is jumping to reach platforms and avoid enemies. Strangely, you don't die if you fall from great heights, and this is a plus, as is the overall fluidity, which pleasantly surprised me.

I think this is one of those titles that, even after you've finished it, leaves you wanting to play it again, even just for a quick round. In short, a timeless classic!

I hope that sooner or later they'll release a new chapter (or even more than one), perhaps set in different locations elsewhere in the world, because, though short, I found it charming, playable, cute, and full of elements that stir up memories.

Are you ready to visit and escape from Egyptian tombs right from the comfort of your own home?

by **Daniele Brahimi**



OUR FINAL SCORE

» Gameplay 80%

Millimeter-precise jumps; everything else is solid.

» Longevity 60%

Cute, but ten levels aren't enough, and you're done in no time.





AXIS

Year: 2026

Editor/Developer: Anystone Games

Genre: Action/Puzzle

Platform: Commodore 64/
Commodore Plus/4

Website: <https://anystone.games/>

Forget Doom, Duke Nukem, or any game where you can actually defend yourself. In Axis, your protagonist is an astronaut who clearly left his survival kit at home and decided that the only way to handle an alien invasion is... to run!

The concept is so simple it borders on psychological torture. You have no weapons. You have no powers. You have only your legs and a life expectancy comparable to that of an ice cube in a volcano.

You must find a key and reach the exit without being touched by the local residents (the aliens), who are very territorial.

All this across 10 sectors packed with enemies that seem programmed to read your mind. If you touch them, you don't lose energy... you explode! Or die of fright! Either way, you start over.

The game is unforgiving, unforgetting, and probably laughing at you under its breath (if chips had breath...). You'll spend 90% of your time staring at an alien blocking a narrow corridor, waiting for the right moment to pass, only to realize you'd calculated everything except your long-dead sloth-like reflexes.

The environments are so clean and





geometric that they look like they were designed by a Swedish architect obsessed with order. The resolution is so high that you'll be able to enjoy every single pixel of your astronaut as he gets vaporized. It's the triumph of "less is more": the fewer frills there are, the more space remains.

The SID chip spits out a soundtrack that is essentially pure anxiety translated into sound waves. It has that relentless rhythm that makes you feel constantly chased, turning your room into a claustrophobic space station.

Axis is proof that in 2026, we still need games to remind us just how bad we are. It's an honest, fast-paced title—frustrating enough to make you feel like a genius if you manage to finish it without throwing your computer out the window.

The perfect game for those with little time and a great desire to suffer. It's a clean, ruthless, and so stripped-down port that you almost feel guilty for not being able to finish it on the first try.

There are no weapons, there are no

excuses... just you and your embarrassing ability to run into the only alien within a three-kilometer radius.

by Chiara "Kika" Carrara



OUR FINAL SCORE

» Gameplay 82%

The controls are so precise that you can't blame the joystick for your failures. The astronaut moves with surgical precision, which is fantastic until you realize that it was you, with your own hands, who launched him into that alien. It's a millimeter-perfect dance between glory and spontaneous explosion.

» Longevity 65%

Ten levels that last about as long as your patience after the fifth consecutive Game Over. Once you've memorized the paths, you'll finish it in the time it takes to drink a cup of coffee. It's perfect for those short on time or for anyone who wants to feel like a gaming prodigy before reality (or dinner) comes knocking.





GRAVITY DUCK

Year: 2026

Editor/Developer: WPQREK

Genre: Platform

Platform: Commodore 64

Website: <https://wpqrek.itch.io/gravity-duck-64>

If you thought the only thing capable of defying the laws of physics on the Commodore 64 was the power supply's ability not to explode after two hours of gameplay, get ready to think again. Gravity Duck 64 reminds us that, in 2026, the best way to have fun is to pretend you're in 1984—but with a duck that has serious balance issues. Gravity Duck 64 is the brainchild of a trio of developers from the Polish homebrew scene, known as wpqrek. The protagonist is a yellow duck who clearly skipped LEGDAY at the gym... because he can't jump! On the other hand, he possesses the divine power to reverse gravity... but really, why jump over obstacles when you can literally headbutt the ceiling? With the press of a button, you flip the world upside down. It's like trying to furnish an apartment while someone shakes the room like a cocktail.

You must retrieve a golden egg through 40 levels of pure architectural madness.

Why? Well, the protagonist probably suffers from a rare form of heavy metal fetishism that drives him to ignore the laws of physics just to fill his nest with egg-shaped ingots. In any case, don't ask questions: run, fall upward, and retrieve that lovely little egg even though Easter is long gone.

The Sprites are so crisp they look laser-cut. It's all so fluid that you risk forgetting your computer has less





OUR FINAL SCORE

» Gameplay 85%

The controls are so responsive that the duck carries out your commands before your brain even formulates them. It's smooth, precise, and no-frills. The only real limit is the coordination between your thumb and your sense of balance, which after ten minutes will start asking for a divorce.

» Longevity 70%

With 40 levels, you'll finish it in the time it takes your neighbor to park their car. It's a lightning-fast experience! Once you've collected all the eggs, the only reason to play it again is to prove to yourself it wasn't just luck. Great for a quick challenge, less so if you were hoping to kill the whole weekend.



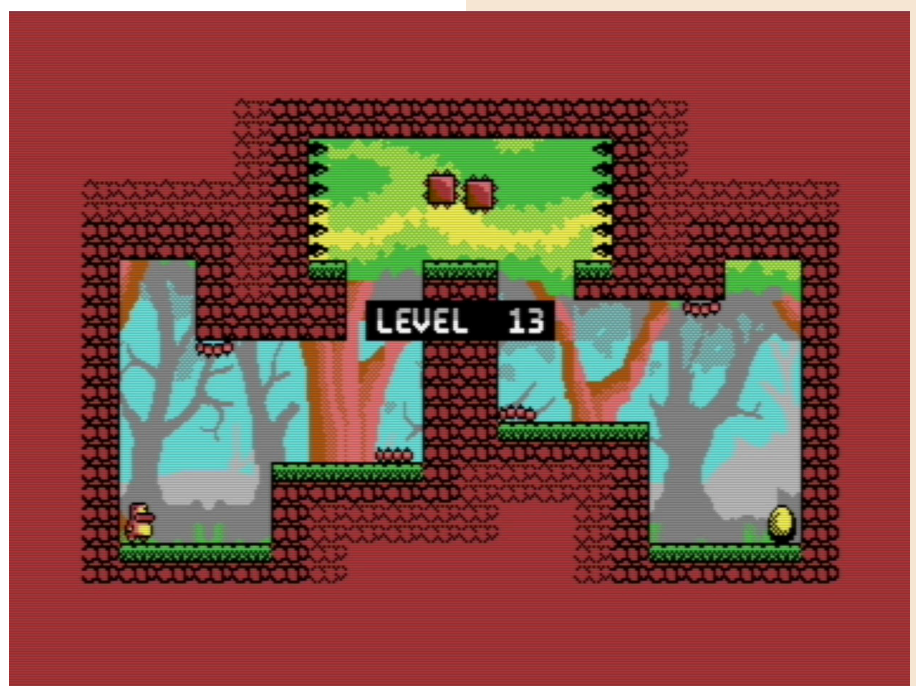
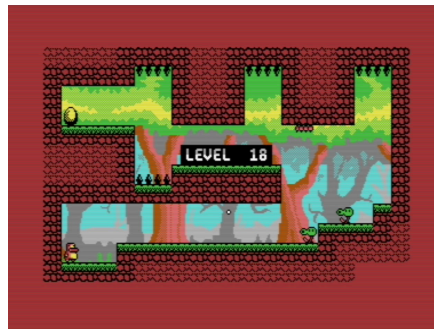
memory than what your electric toothbrush needs just to turn on.

The SID chip delivers a rhythmic and super-catchy soundtrack, the kind that gets stuck in your head and never leaves. It's a hypnotic tune that turns every level into a gravitational dance, capable of making you whistle for hours even after you've turned off the monitor. A small masterpiece of sound synthesis that gives the game that magical touch typical of great C64 productions.

The control system and precision of the commands deserve praise, though I should note that it takes a bit of practice to master the game's gravity system.

In any case, Gravity Duck is a little gem of charm and precision. It's the ideal game to pamper your Commodore and remind you that, every now and then, it's nice to look at the world from a different perspective, even if only for the time it takes a wing to flap. And as I always say, it's worth giving it a try!

by Chiara "Kika" Carrara



Refurbished software & revisited games

Retrocomputing in 2026 is a true cultural and technical platform, a reality that connects the past to the future. In an era of rapid obsolescence and dematerialization of software, recovering systems and programs from 30 or 40 years ago has a value that goes beyond mere collecting. It is an act of historical preservation and, at the same time, a laboratory for innovation. Preserving the software of the past means saving a heritage of ideas. In the early days of "home computing", developers and programmers worked with just a few kilobytes of memory, limited processors, and slow, fragile tools. Yet, from these limitations emerged elegant solutions, ingenious algorithms, and refined programming techniques. Studying that code is like reading literary classics: one can discover fundamental principles that remain valid even today. However, preserving digital artifacts is no simple process. The original programs are often found only on deteriorated media and in paper form. Converting them to digital form means making them accessible and available, allowing anyone to run them on emulators or actual hardware. It is painstaking work—one might say akin to archaeological work—and requires technical expertise and historical sensitivity.

But retrocomputing is not limited to preservation. In recent years, a surprising phenomenon has emerged: the creation of new games and software for obsolete platforms. Programmers now use modern tools to push those systems beyond their original limits. The result is a body of work that not only matches, but sometimes surpasses that of the "golden age." This dialogue between past and present opens up interesting possibilities. The goal is to respect the aesthetics and logic of the era while introducing innovative elements that demonstrate just how much untapped potential those machines still held. Retrocomputing thus becomes a space for experimentation, where constraints represent a stimulating challenge. A fascinating aspect of these processes is "reverse engineering." When the original source code is unavailable, working on a program means analyzing its functionality starting from the binary code. It is a complex process that requires advanced skills and patience, but it can also lead to extraordinary results. In this sense, preservation is no longer a passive act, it becomes dynamic. It is not just about saving what has been published, but also about reinterpreting it, modifying it, and—why not?—improving it. A form of technological continuity that breaks the linearity of time and shows how the past can still engage with the present.

Of course, there are ethical and legal issues to address, such as software ownership and copyright. However, it is difficult to deny the cultural value of these activities, which help keep a fundamental part of the history of computing alive. Digital preservation (as our RetroLiPS project clearly shows) serves as a bridge between generations of technologies and programmers. It is both memory and reinvention. In this age that tends to forget quickly, this practice reminds us that even the most outdated bits can hide an innovation capable of surprising us.

David "Cercamon" La Monaca

Disclaimer

RetroMagazine World as an aperiodic magazine entirely ad-free is a non-profit project and falls off any commercial circuit. All the published material is produced by the respective authors and published thanks to their authorization.

RetroMagazine World is licensed under the terms of: Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) <https://creativecommons.org/licenses/by-nc-sa/4.0/>

This is a human-readable summary of (and not a substitute for) the license. You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms. Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.



AI Disclaimer - Unless explicitly stated by the authors, contributors and the publisher RMW, no content in this digital magazine (editorials, articles, reviews, text or images) has been produced or generated by AI tools.



ISSN: 3103-697X - RetroMagazine World [Online]
DOI: 10.5281/zenodo.20088377

© 2017-2026 RetroMagazine World - All rights are reserved.
Publisher: Associazione Culturale RetroMagazine World
ID: IT-93031480580 - Registered office and HQ: Rome, Italy
retromagazineworld@gmail.com - <https://www.retromagazine.net>

RetroMagazine World
Year 8 - Issue 53 - MAY 2025

Editor in Chief **Francesco Fiorentini**
Editing/Deputy Manager **David La Monaca**
Publishing Manager **Marco Pistorio**
Webmaster **Giorgio Balestrieri**
Social Media Manager **Carlo Nith Del Mar Pirazzini**

