Build your own NES Emulator... with Kotlin

Let Fools Dream

Artur Skowroński

VIRTUSLAB

Artur Skowronski

Head of Java / Kotlin Development

VIRTUSLAB

SCALA 3.7.0 RELEASED!

Wednesday 7 May 2025

Wojciech Mazur, VirtusLab

RELEASE

JVM Weekly

Home | Notes | Foojay.io Community Newsletter | Archive | About

"Optimizing the GC when Migrating Cloud Workloads to Arm" with Kieran Hejma...

Today, we have a guest post from Kieran Hejmadi, an Software and Academic Ecosystem Manager in Arm.

MAY 1 · ARTUR SKOWRONSKI AND KIERAN HEJMADI

Latest | Top | Discussions

April - The Rest of the Story - JVM Weekly vol. 127

I know there is a lot of April left, not my fault, calendar is working against my publishing schedule this month...

APR 24 · ARTUR SKOWRONSKI

Java in AWS with Wojtek Gawroński - JVM Weekly vol. 126

I'm experimental mood recently, so today, we have the interview!

APR 17 · ARTUR SKOWRONSKI AND WOJTEK GAWROŃSKI

Make Final Mean Final... and the other JEPs - JVM Weekly vol. 125

It's been a while since we've seen JEPs! But you know, not the fully baked ones, not the ones that have advanced far i...

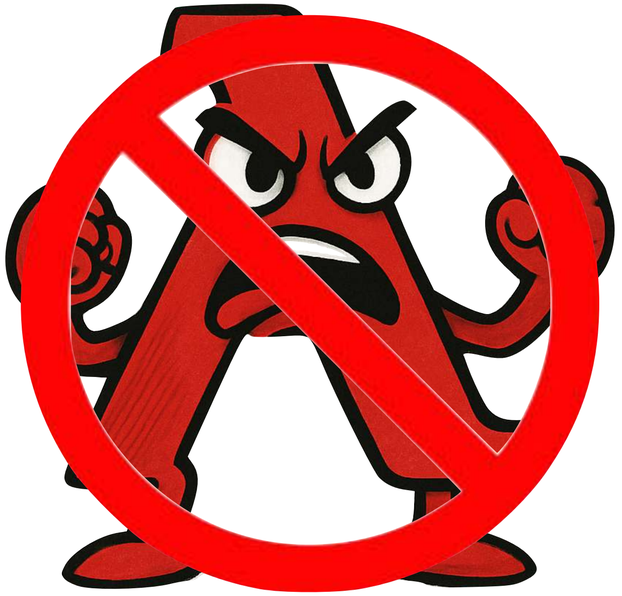
JVM Weekly

From the latest updates in JVM languages like Java, Kotlin, and Scala to emerging technologies like GraalVM and Quarkus, this newsletter covers a wide range of topics that are of interest to developers and tech enthusiasts.

✓ Subscribed

Recommendations MANAGE

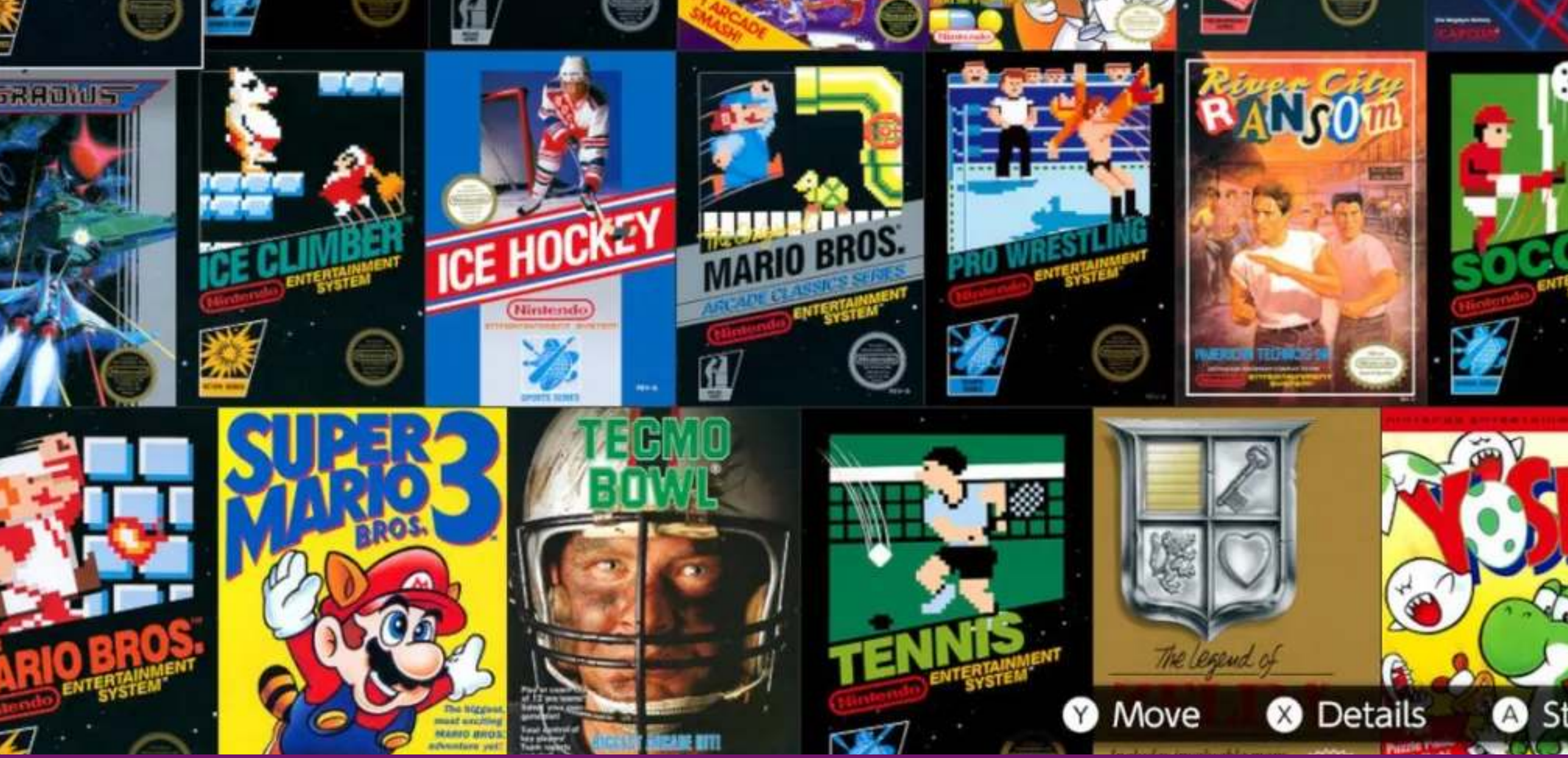
- **Tales from the jar side** — Ken Kousen
- **Architecture Weekly** — Oskar Dudycz
- **Engineering Primer** — Bartek Antoniak
- **Tech Talks Weekly**



It will be mostly about relieving memories...

...and especially about very old hardware

CGI
Scoreboar
CGI
Nintendo
ENTERTAINME
SYSTEM
Nintendo
ACTION SET
EUROPEAN VERSION



My first video console

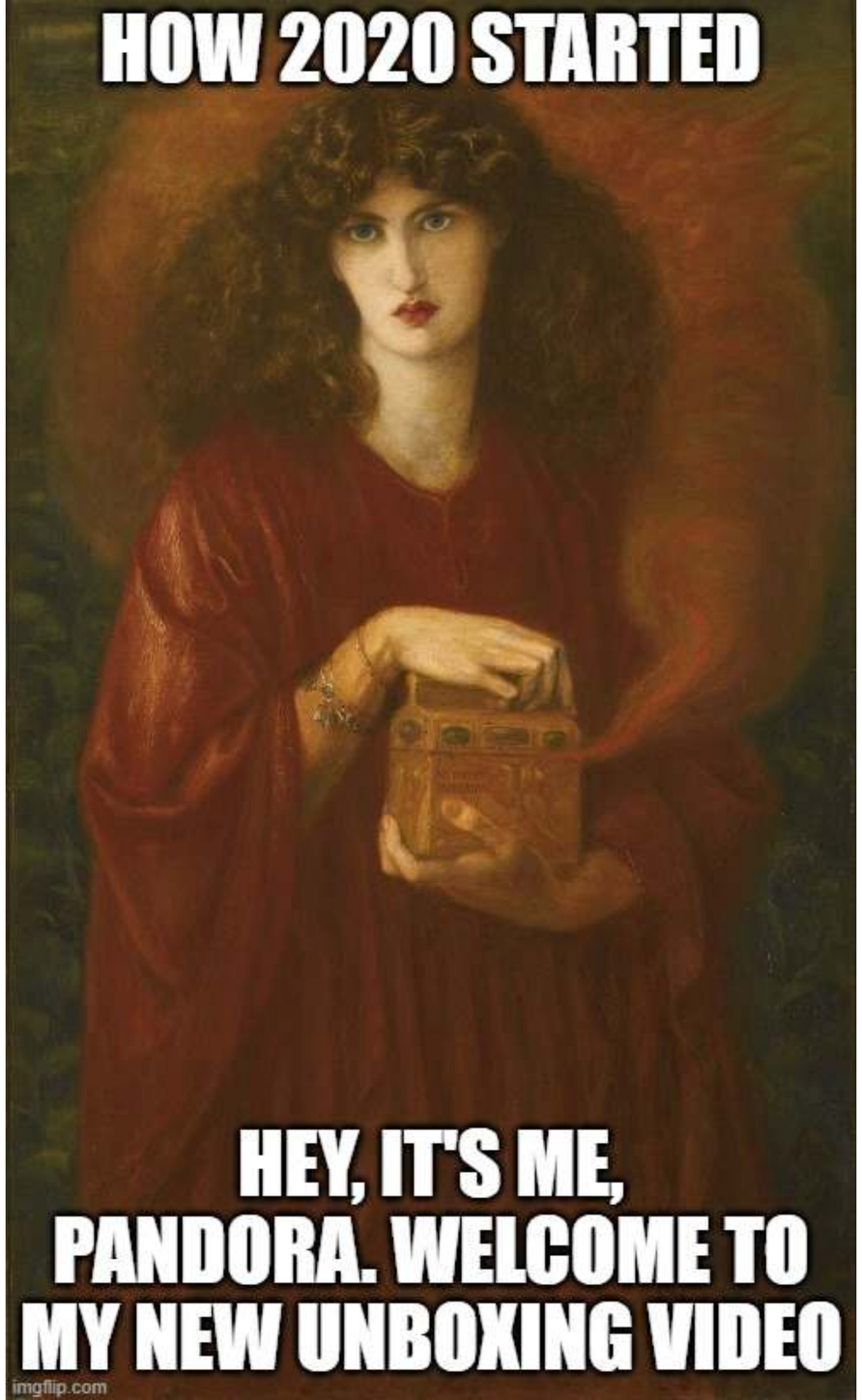
My first video console

So called famiclones

137 Slides

+ Demo 😃

The story starts in 2020


HOW 2020 STARTED
HEY, IT'S ME, PANDORA. WELCOME TO MY NEW UNBOXING VIDEO
imgflip.com

Babe Ruthless @unfitz.bsky.social

"Now Open 24/7" - Pandora's Box

Sun, 02 Feb 2025 09:59:55 GMT

View on Bluesky

Search articles

Blog / 2025 / March / 10 / Dolphin Progress Report: Release 2503

Dolphin Progress Report: Release 2503

Written by JosJuice, JMC47, MayImilae on March 10, 2025 / Last update on April 21, 2025 / Short link / Forum thread

It's time for us to again delve into the latest changes hitting the newest release of Dolphin Emulator. If you've been keeping tabs on development channels, you'll know there are some exciting bombshell features waiting for us just over the horizon. Unfortunately, given the close proximity to this release, we've decided to hold a lot of them off so they can get more widespread testing in the development builds before hitting a release. Think of the March 2025 release as an iterative improvement that focuses on user experience and stacking many small iterative improvements instead of swinging for every homerun.

Having said all of that? We do have a few major changes that made the cut.

Probably the most requested of the features in this progress report is the ability to *finally* set custom graphical settings on a per game basis. Our advanced users may be thinking, "*Wait, couldn't we already do that?*" and they'd be right. The major change here is that per-game graphical settings has been added directly to the game properties page, allowing for a very easy, visual way to change settings and see what settings are force enabled.

For our achievement hunters out there, RetroAchievements has seen its first major upgrade that will bring a huge amount of quality of life to many games. And if you're into weird old game behaviors? A developer dove into Summoner: A Goddess Reborn to determine why it crashes so much. The catch with this one? Some of these crashes have been confirmed to happen on **official hardware**!

Blog tags

3d 4.0 5.0 60FPS Accessory achievements adreno amd Analysis android announcement arm ARM audio ax-hle BBA bigfix bounding box bug bugfix cheats Cheats CI Commemoration community controller D3D D3D9 Datel Devdiary DevDiary discord Disney Trio of Destruction driver DS dsp-hle dvd timings Factor5 Feature Removal Foundation gamebug GameCube Gamehacks GBA gpu Graphics Hardware HD HDR hle homebrew input Intel iPhone JIT Legal Licensing linking Linux mac macOS mail memory card

I always loved video game mechanics

vNES Public

Watch 3 | Fork 13 | Starred 39

master | 1 Branch | 4 Tags | Go to file t | Add file | Code

bfirsh Add a readme 49703c2 · 8 years ago 5 Commits

src	vNES 2.16	8 years ago
LICENSE	vNES 2.15	8 years ago
README.md	Add a readme	8 years ago

README | GPL-3.0 license

vNES

A mirror of Jamie Sanders' Java NES emulator.

It has now disappeared from its original location, virtualnes.com, but I have pieced this together from files on my hard disk and from the web.

If you're looking for the actual README, version 2.13 has one.

JSNES is based on this emulator.

About

A mirror of Jamie Sanders' Java NES emulator, which once lived at virtualnes.com

emulator nes

Readme

GPL-3.0 license

Activity

39 stars

3 watching

13 forks

Report repository

Releases

4 tags

Packages

No packages published

Languages

Java 100.0%

vNES Public

Watch 3 | Fork 13 | Starred 39

master | 1 Branch | 4 Tags | Go to file t | Add file | Code

bfirsh Add a readme 49703c2 · 8 years ago 5 Commits

src	vNES 2.16	8 years ago
LICENSE	vNES 2.15	8 years ago
README.md	Add a readme	8 years ago

README | GPL-3.0 license

vNES

A mirror of Jami[redacted] Java NES emulator.

It has now disappeared from its original location, virtualnes.com, but I have pieced this together from files on my hard disk and from the web.

If you're looking for the actual README, version 2.13 has one.

JSNES is based on this emulator.

About

A mirror of Jamie Sanders' Java NES emulator, which once lived at virtualnes.com

emulator nes

Readme

GPL-3.0 license

Activity

39 stars

3 watching

13 forks

Report repository

Releases

4 tags

Packages

No packages published

Languages

Java 100.0%

Watch 3 | Fork 13 | Starred 39

Go to file t | Add file | Code

49703c2 · 8 years ago | 5 Commits

Emperor Demon Dragon God Java (天王鬼龍神 ジャバ *Tennō Ki Ryūjin Jaba*[7]), formerly known in western releases as **"Red Falcon"**, is the first main antagonist of the *Contra* series. He was introduced in 1987 as the final boss of the original *Contra*. He belongs to the Red Falcon army, being its highest-ranking member in that game and the leader in command of the first alien invasion attempt on Earth in 2633 A.D.

vNES

A mirror of Jami[illegible] Java NES emulator.

It has now disappeared from its original location, virtualnes.com, but I have pieced this together from files on my hard disk and from the web.

If you're looking for the actual README, version 2.13 has one.

JSNES is based on this emulator.

About

A mirror of Jamie Sanders' Java NES emulator, which once lived at

Releases

4 tags

Packages

No packages published

Languages

Java 100.0%

vNES Public

Watch 3 | Fork 13 | Starred 39

master | 1 Branch | 4 Tags | Go to file | Add file | Code

bfirsh Add a readme 49703c2 · 8 years ago 5 Commits

src	vNES 2.16	8 years ago
LICENSE	vNES 2.15	8 years ago
README.md	Add a readme	8 years ago

README | GPL-3.0 license

vNES

A mirror of Jamie Sanders' Java NES emulator.

It has now disappeared from its original location, virtualnes.com, but I have pieced this together from files on my hard disk and from the web.

If you're looking for the actual README, version 2.13 has one.

JSNES is based on this emulator.

About

A mirror of Jamie Sanders' Java NES emulator, which once lived at virtualnes.com

emulator nes

Readme

GPL-3.0 license

Activity

39 stars

3 watching

13 forks

Report repository

Releases

4 tags

Packages

No packages published

Languages

Java 100.0%

JDK 1.8

No build system

Applets

Tangled code...

README License

kNES - Kotlin NES Emulator

kNES is a Nintendo Entertainment System (NES) emulator written in Kotlin, forked from the vNES Java emulator. This project was created primarily for fun and educational purposes, allowing developers to learn about emulation techniques and NES hardware while enjoying classic games.

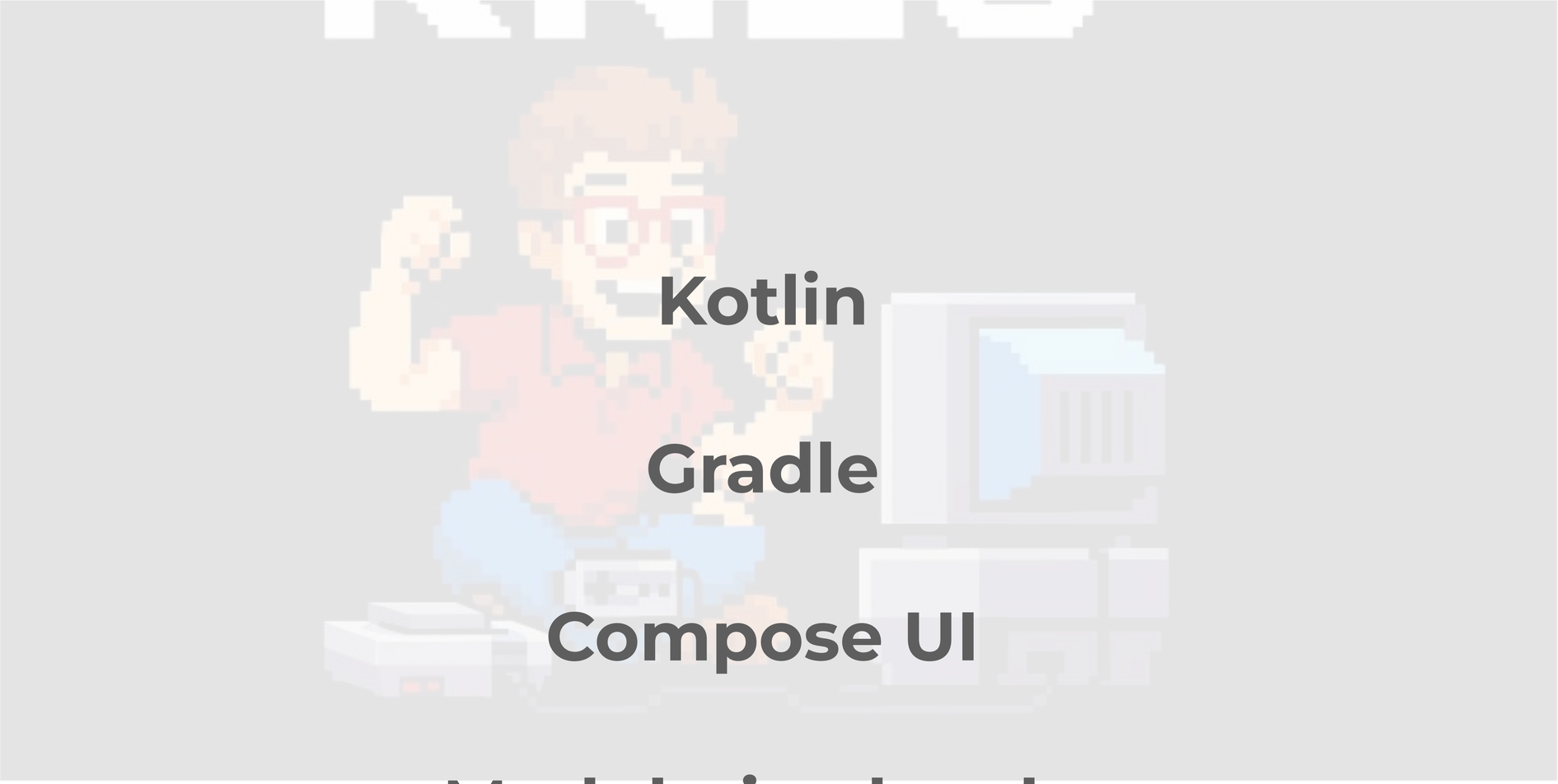
Java CI passing

About This Project

kNES is a reimplementation and extension of the vNES emulator (originally developed by Brian F. R.) in Kotlin. The project aims to:

- Provide a modern, Kotlin-based NES emulator
- Serve as an educational resource for those interested in emulation
- Demonstrate different UI implementation approaches in the JVM ecosystem
- Have fun with retro gaming and programming!

This project is distributed under the GNU General Public License v3.0 (GPL-3.0), ensuring it remains free and open source.



Kotlin

Gradle

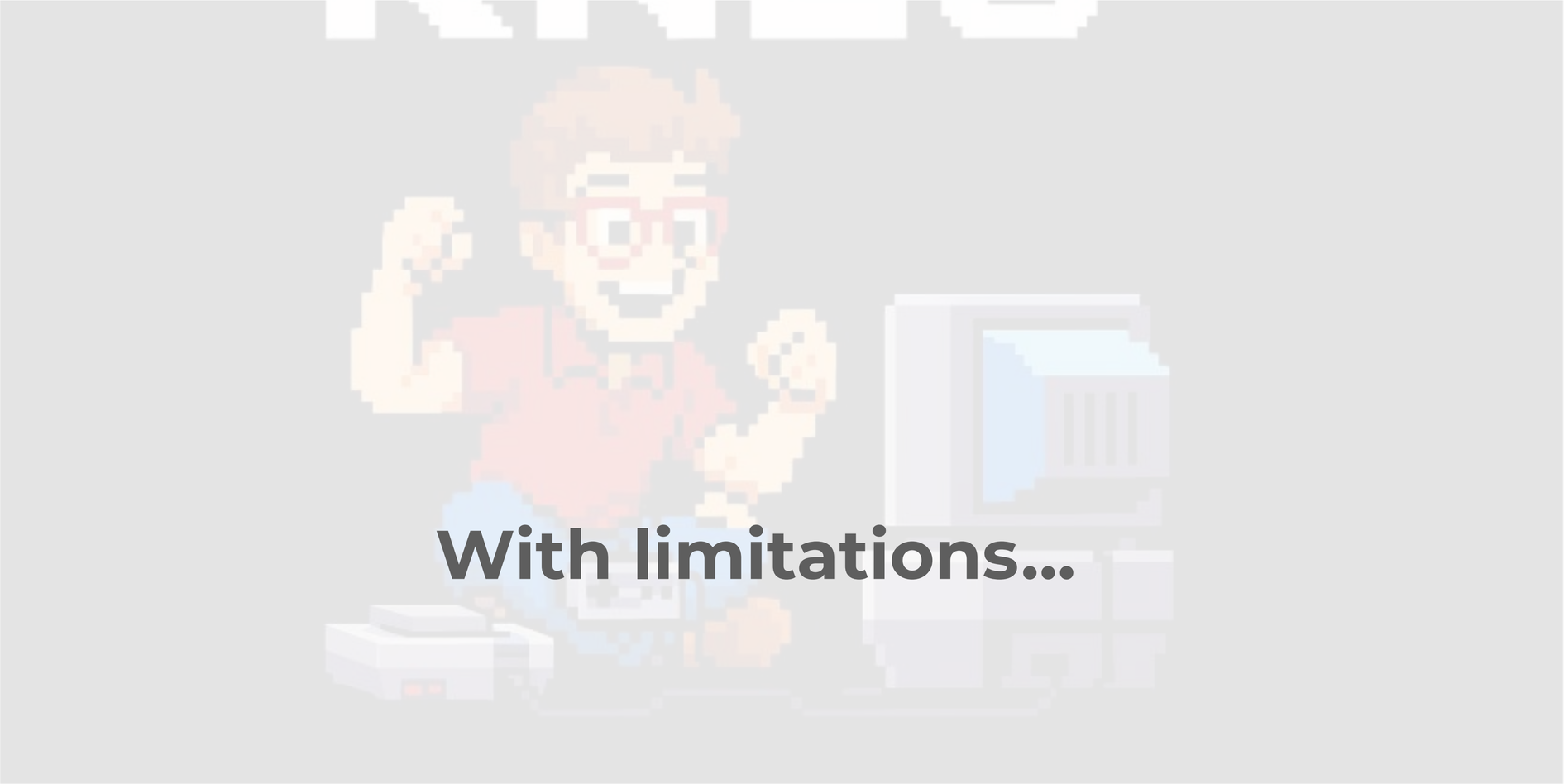
Compose UI

Modularized code

kNES is a Nintendo Entertainment System (NES) emulator written in Kotlin, forked from the vNES Java emulator. This project was created primarily for fun and educational purposes, allowing developers to learn about emulation techniques and NES hardware while enjoying classic games.

Java CI passing

About This Project



With limitations...

kNES is a Nintendo Entertainment System (NES) emulator written in Kotlin, forked from the vNES Java emulator. This project was created primarily for fun and educational purposes, allowing developers to learn about emulation techniques and NES hardware while enjoying classic games.

Java CI passing

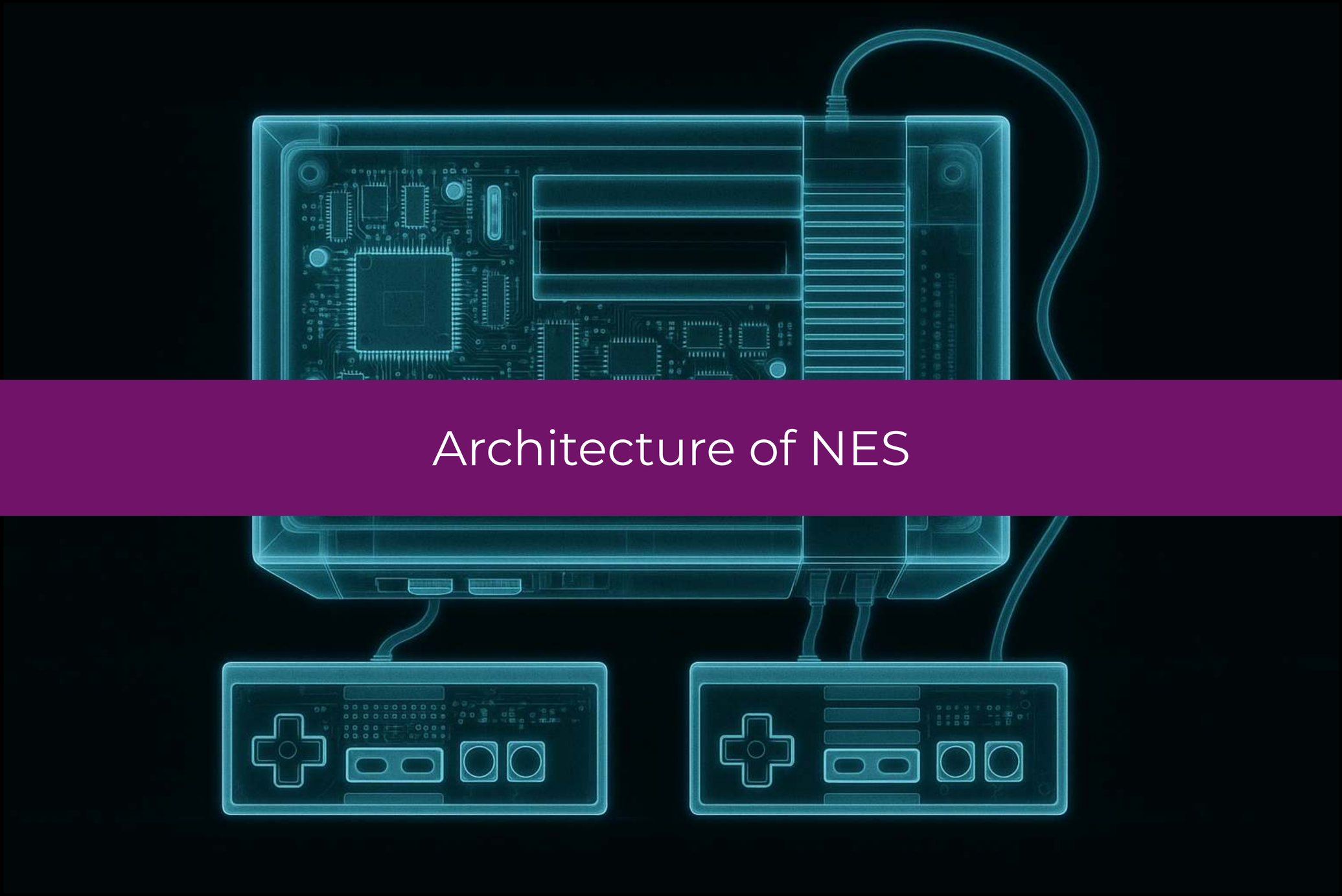
About This Project

NES Architecture

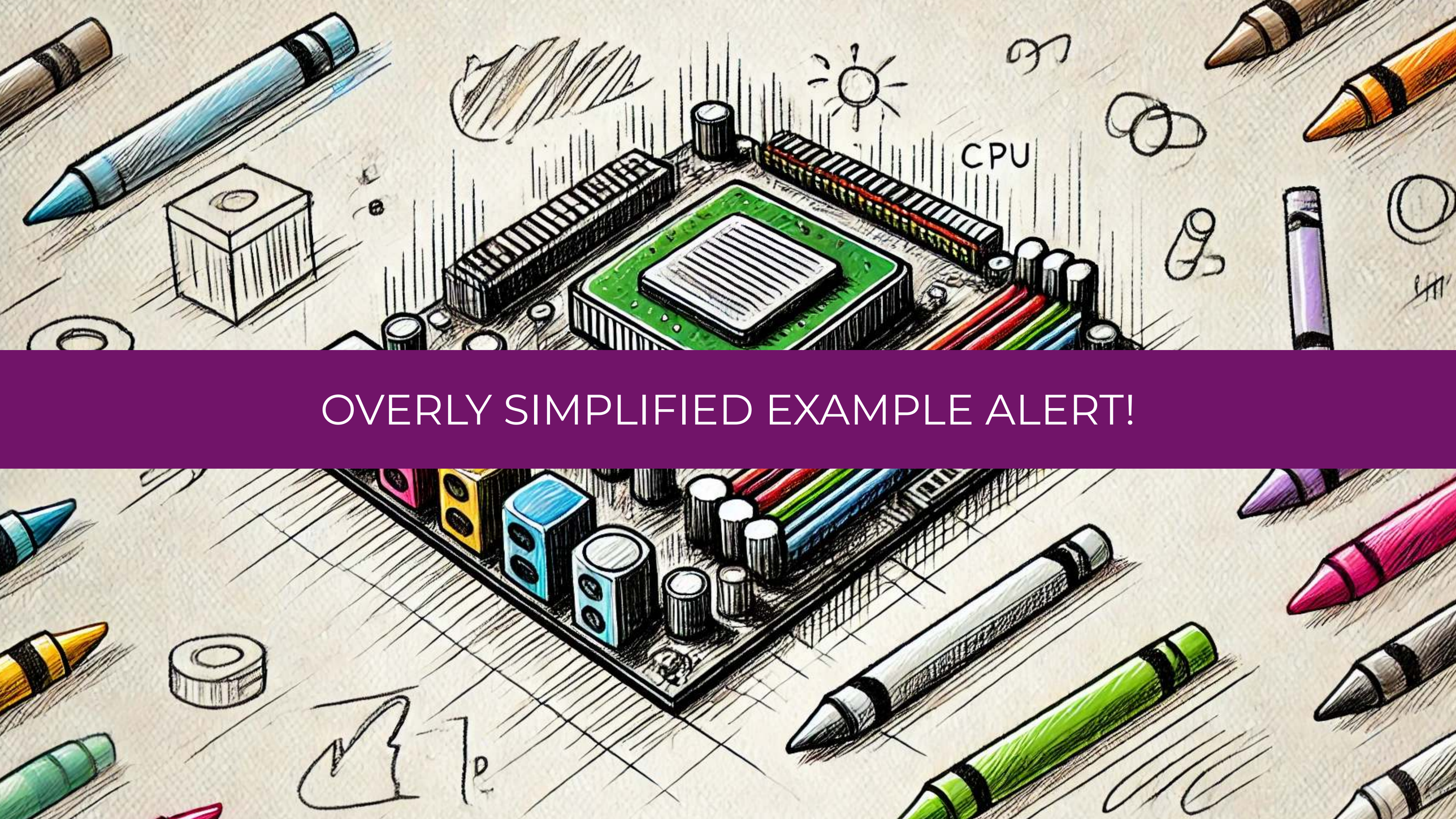
Why writing emulator is interesting (and hard as hell)

What are the interesting aspects of doing that in Kotlin

Why it is even worth to do that



Architecture of NES



OVERLY SIMPLIFIED EXAMPLE ALERT!

NES / Famicom Architecture

A practical analysis by Rodrigo Copetti

If you use accessibility tools or old browsers, switch to the ‘classic’ edition.

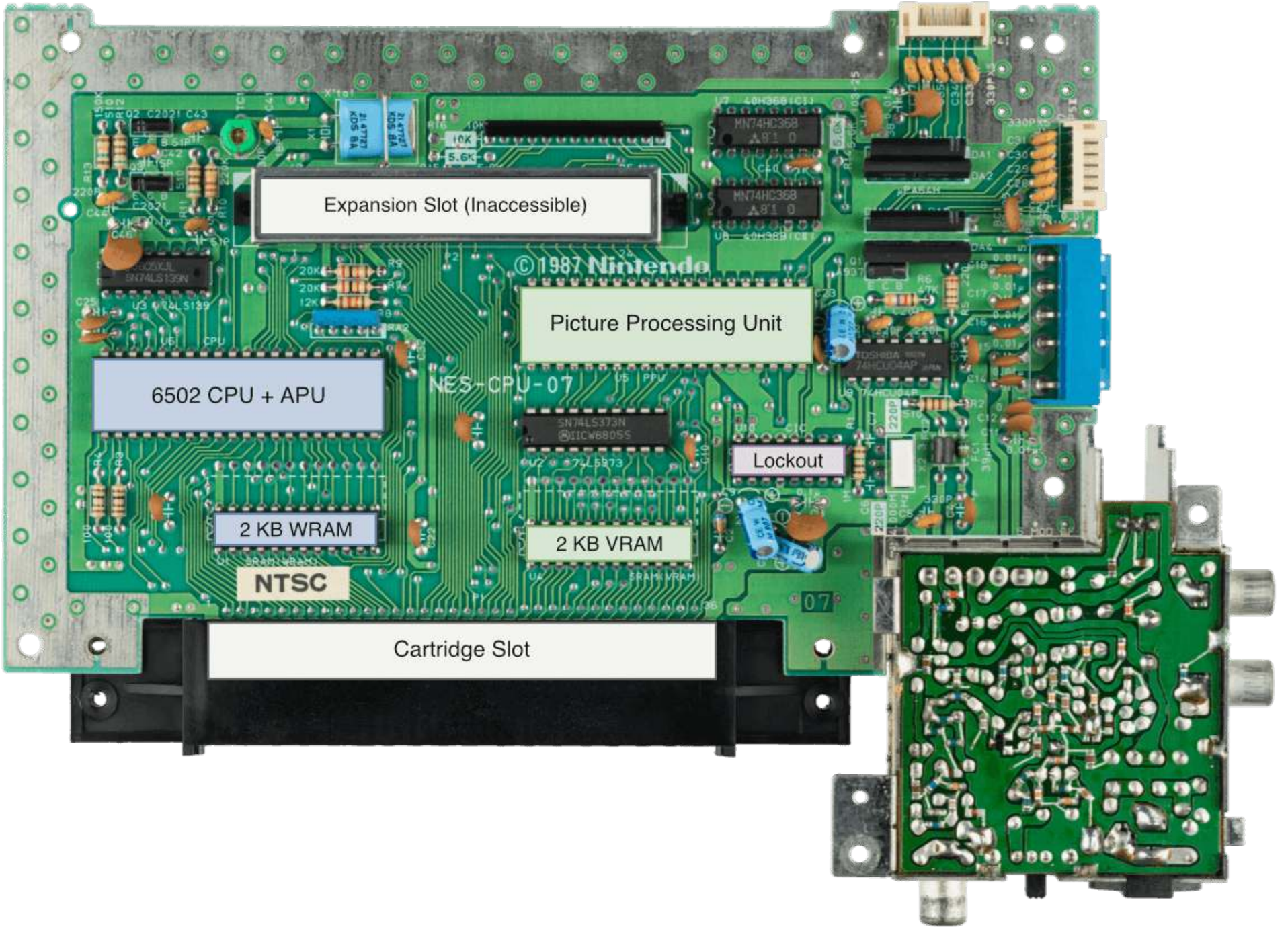
🇬🇧 - English

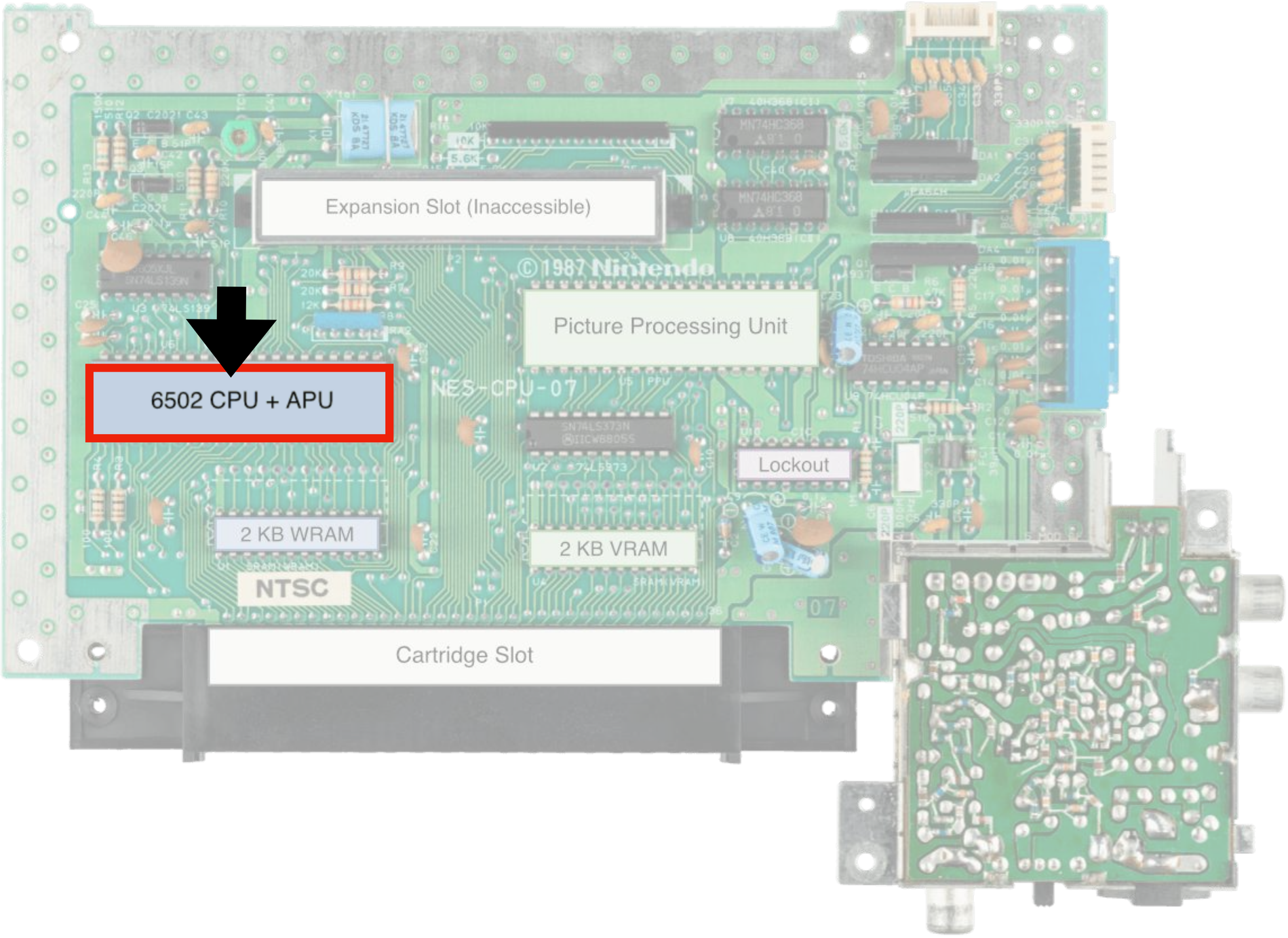
▼ Book edition

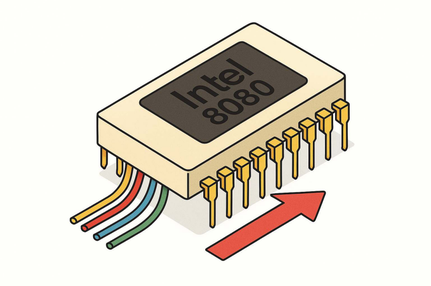
This article is also published on many digital book stores for the benefit of eBook readers. The new edition is DRM-free and can be read whilst offline. Furthermore, it's updated at the same pace as the website.

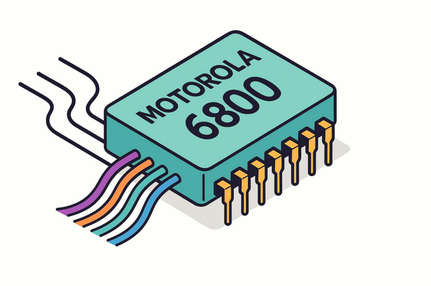
You can find the eBook at Amazon Kindle, Apple Books, Kobo and other stores. You can also find printed

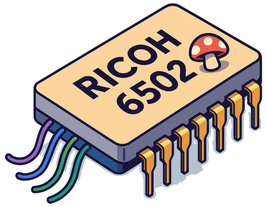
https://www.copetti.org/writings/consoles/nes/


Expansion Slot (Inaccessible)
Picture Processing Unit
6502 CPU + APU
Lockout
2 KB WRAM
2 KB VRAM
NTSC
Cartridge Slot
© 1987 Nintendo
NES-CPU-07


Expansion Slot (Inaccessible)
Picture Processing Unit
6502 CPU + APU
Lockout
2 KB WRAM
2 KB VRAM
NTSC
Cartridge Slot







Intel 8080

A popular CPU featured in the *Altair*, the first "personal'" computer.

It has an 8-bit data bus and a 16-bit address bus.

Motorola 6800

Another 8-bit CPU designed by Motorola, it contains a completely different instruction set.

Many do-it-yourself computer kits, synthesisers and all-in-one computers included the 6800.

MOS 6502

A redesigned version of 6800.

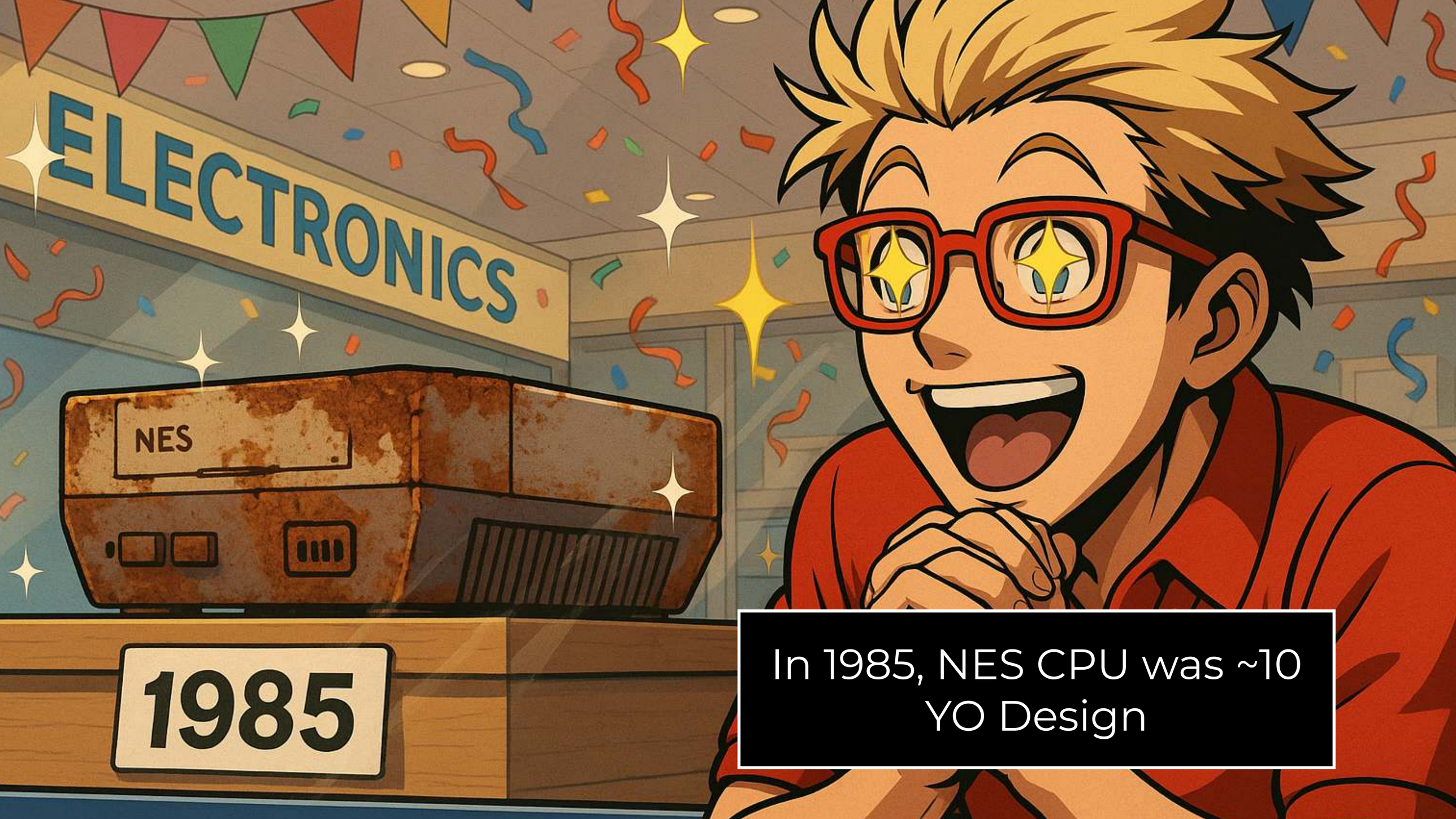
While incompatible, the new chip was much less expensive to produce and iconic computer makers (Commodore, Apple, Atari) chose the 6502.

Ricoh 6502

Nintendo needed something inexpensive but familiar to develop for, so they selected the 6502.

Ricoh, their CPU supplier, successfully produced a 6502-compatible CPU

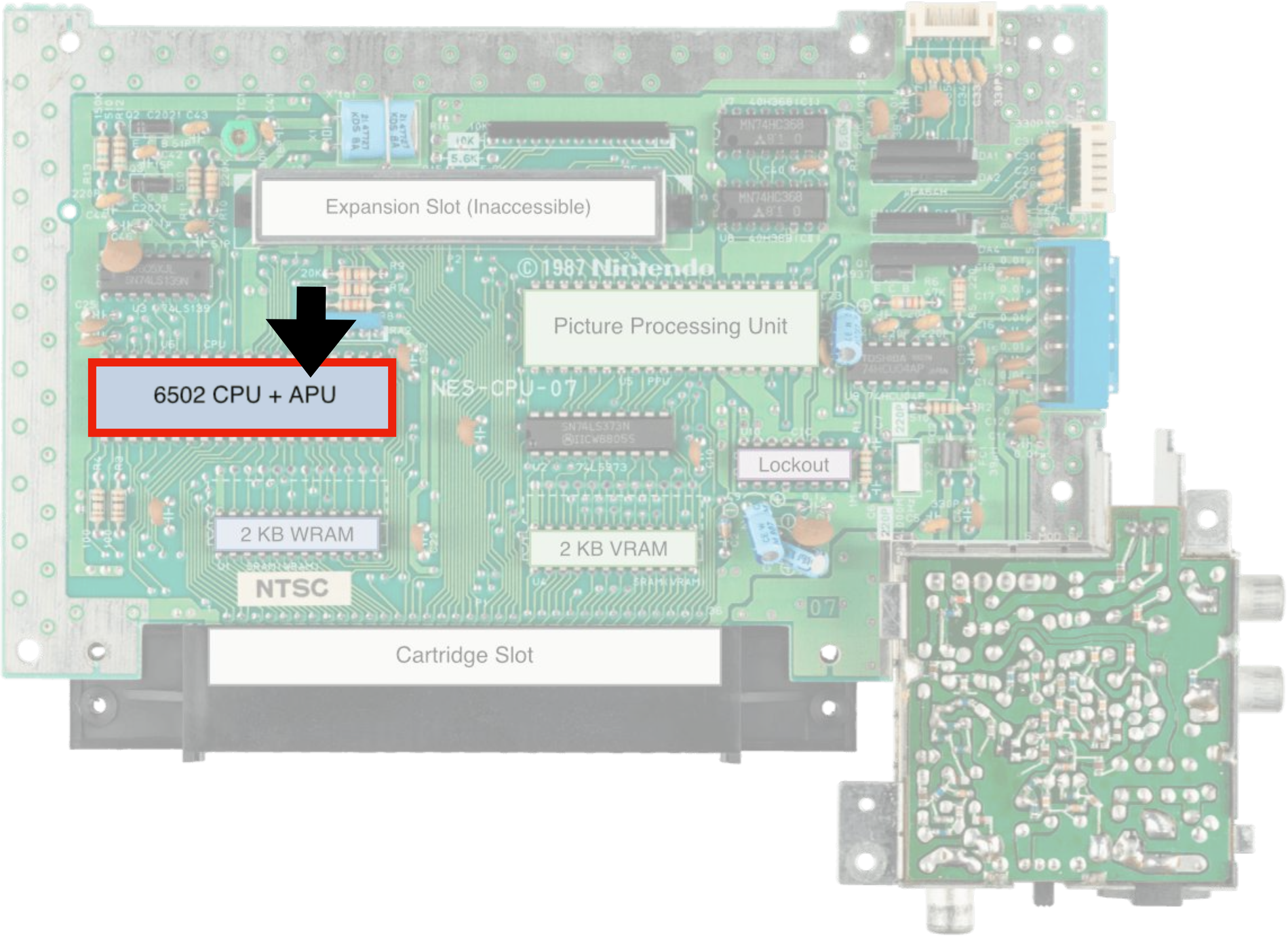
1974 **1974** **1975** **1983**

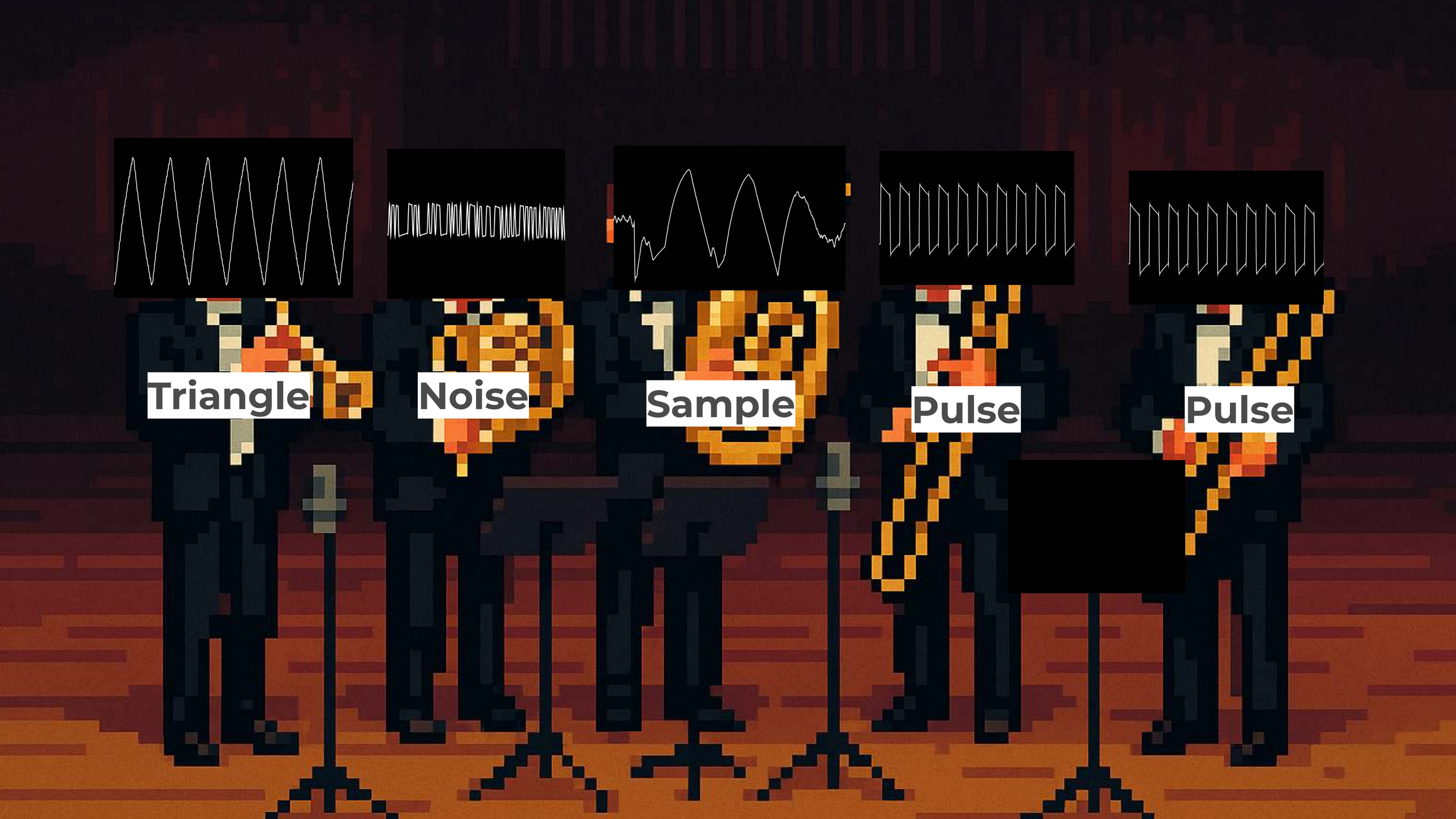

ELECTRONICS
NES
1985
In 1985, NES CPU was ~10
YO Design

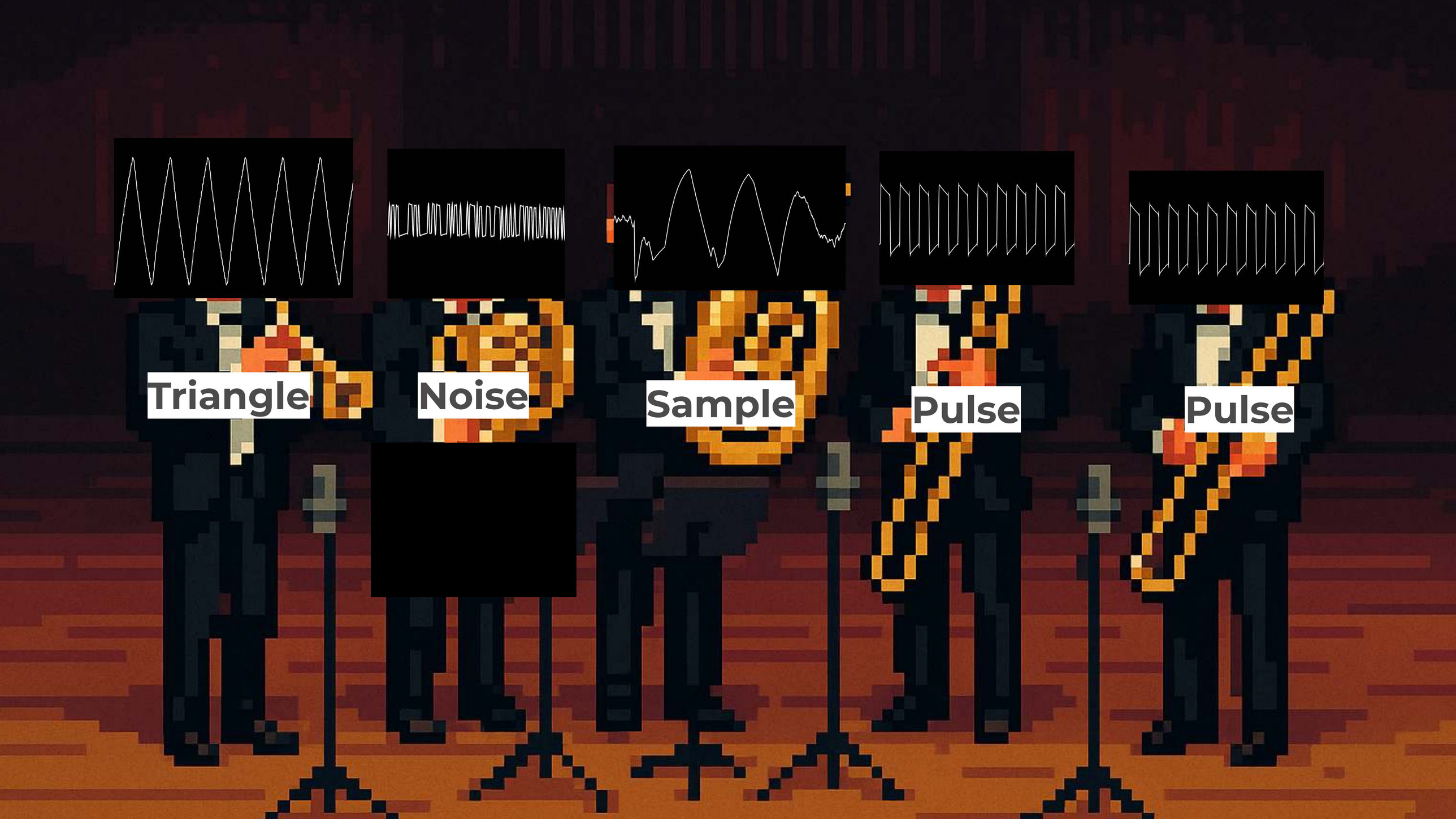
00 77 10

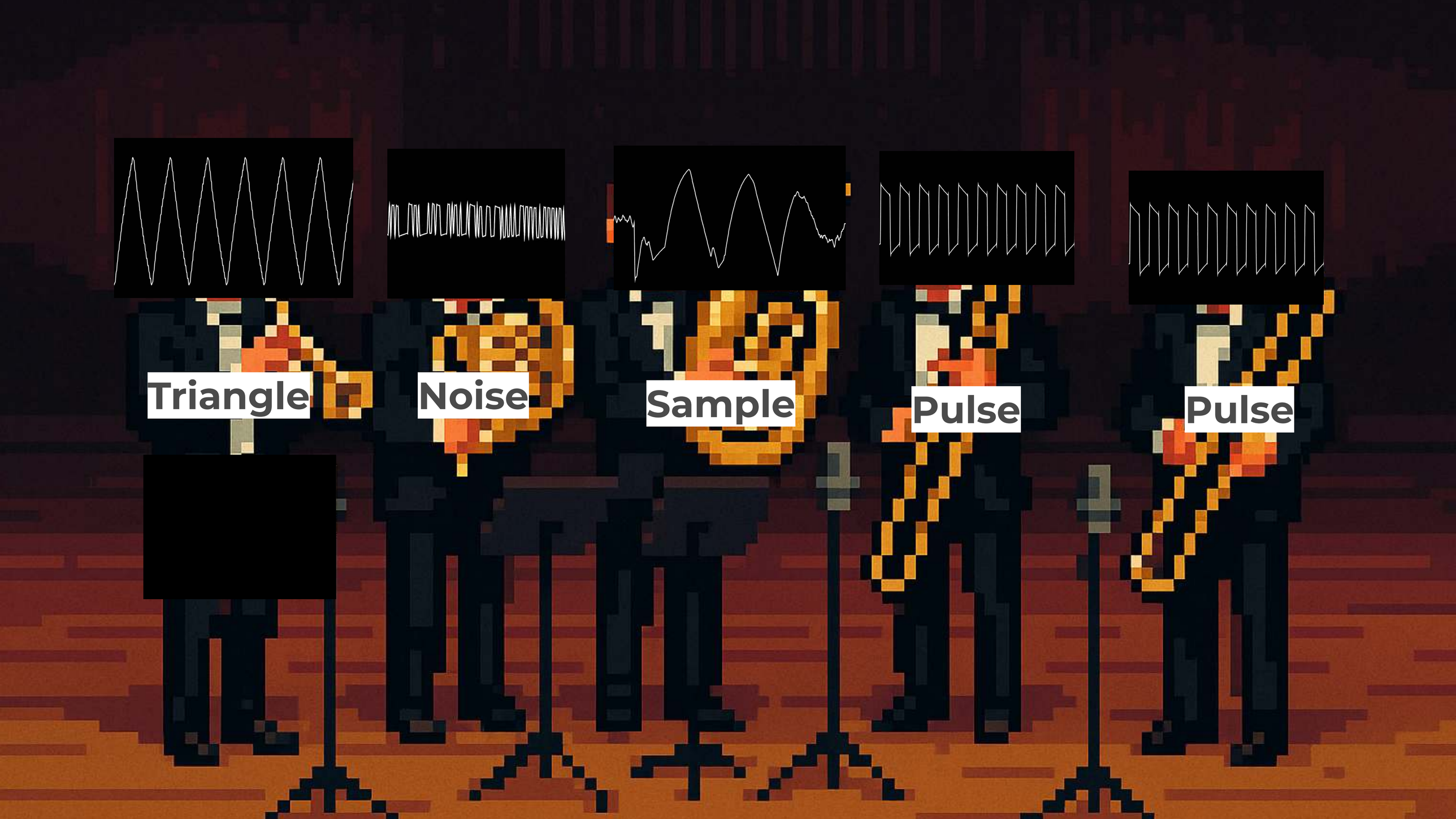
- “Bitness” historically refers to word width — that is, how many bits the processor fetches from RAM and processes in a single instruction. If registers are 8-bit (as in the 6502 / Ricoh 2A03), we call it an 8-bit machine - regardless of other specs.
- Beyond the technical meaning, “8-bit” today also refers to the video game era of the 1980s — NES, Commodore 64, Atari 800, and the like.
- The visual style from that era is often called “8-bit art.
- It has 8-bit registers, which means that the numbers it operates on range from 0 to 255 (2^8 = 256 values).
- It uses a 16-bit address bus, which allows it to address up to 65,536 bytes of memory (64 KB).

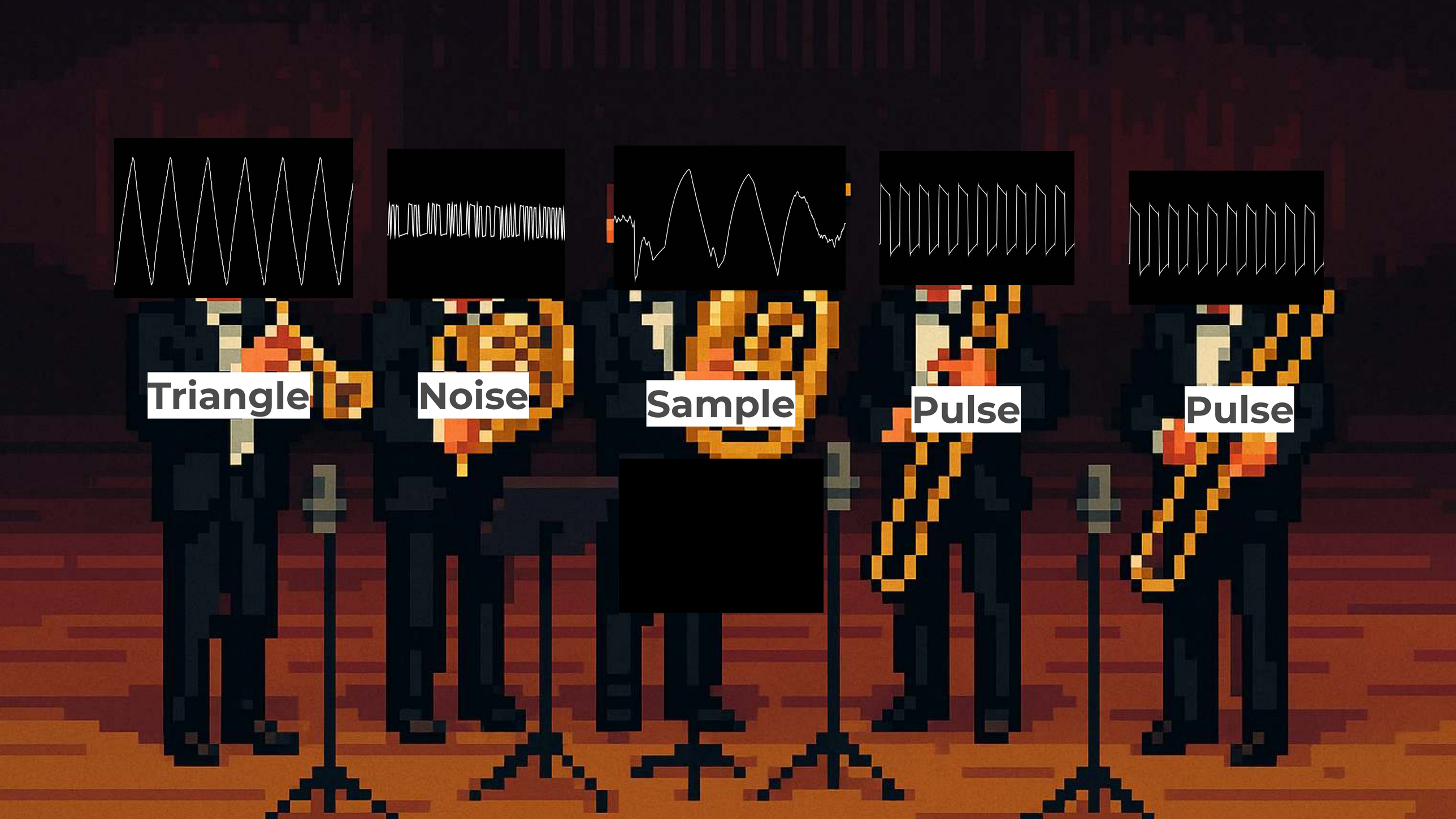
What 8-bit means?

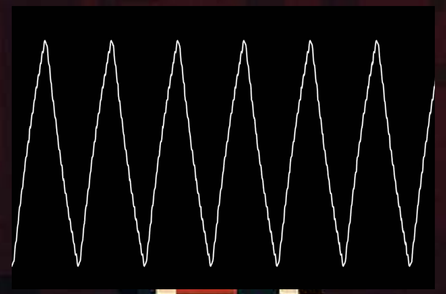

Expansion Slot (Inaccessible)
Picture Processing Unit
6502 CPU + APU
Lockout
2 KB WRAM
2 KB VRAM
NTSC
Cartridge Slot

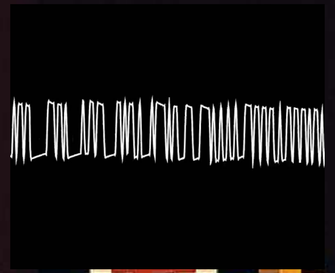

Triangle
Noise
Sample
Pulse
Pulse

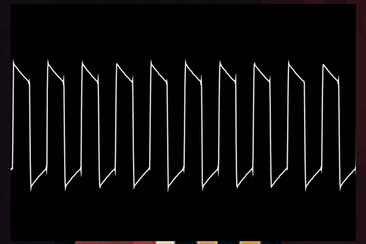

Triangle
Noise
Sample
Pulse
Pulse

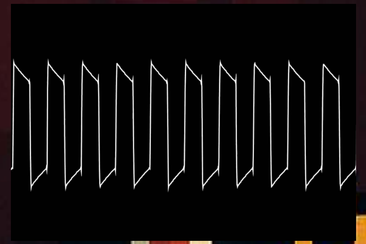

Triangle
Noise
Sample
Pulse
Pulse


Triangle
Noise
Sample
Pulse
Pulse







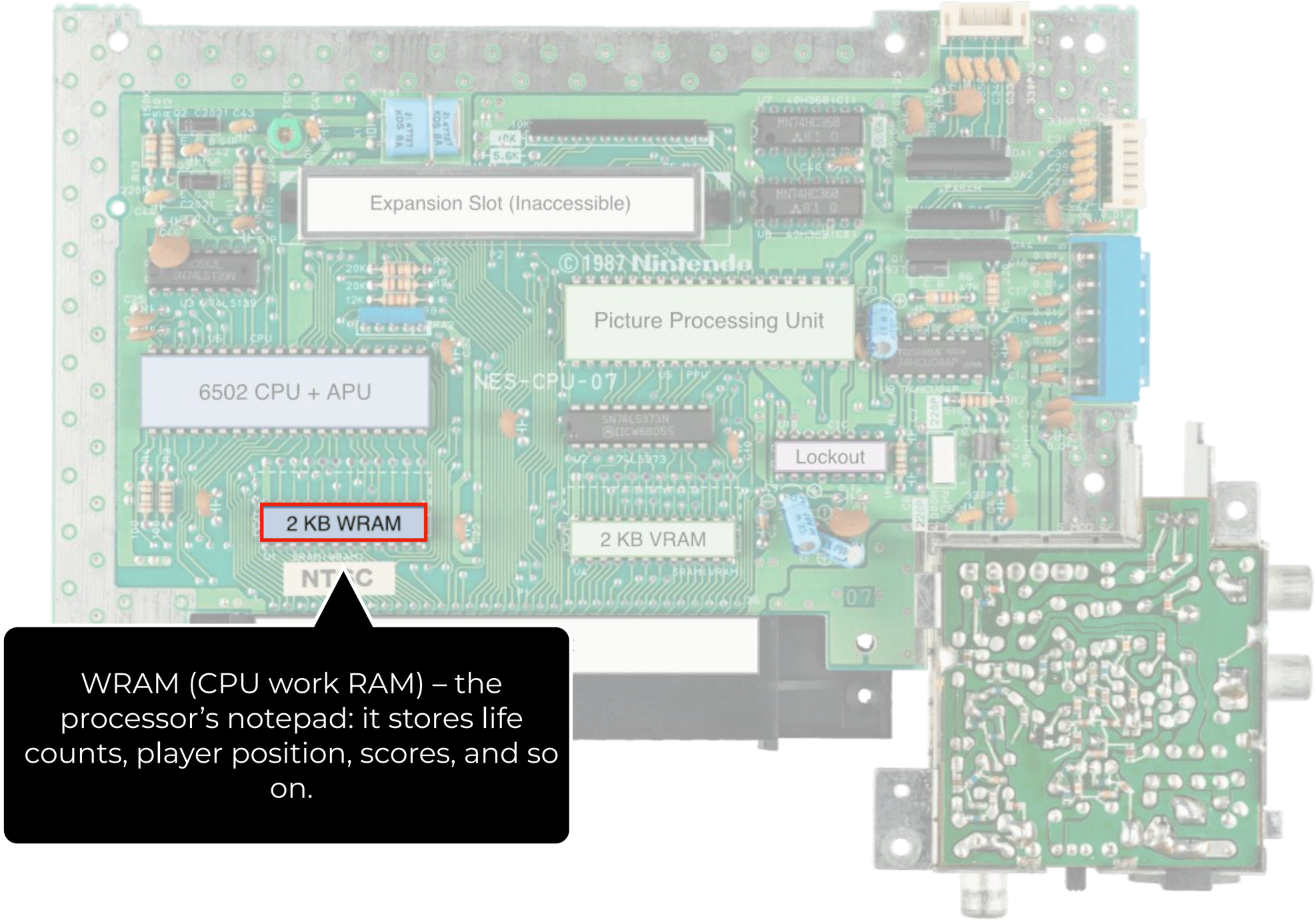


Triangle

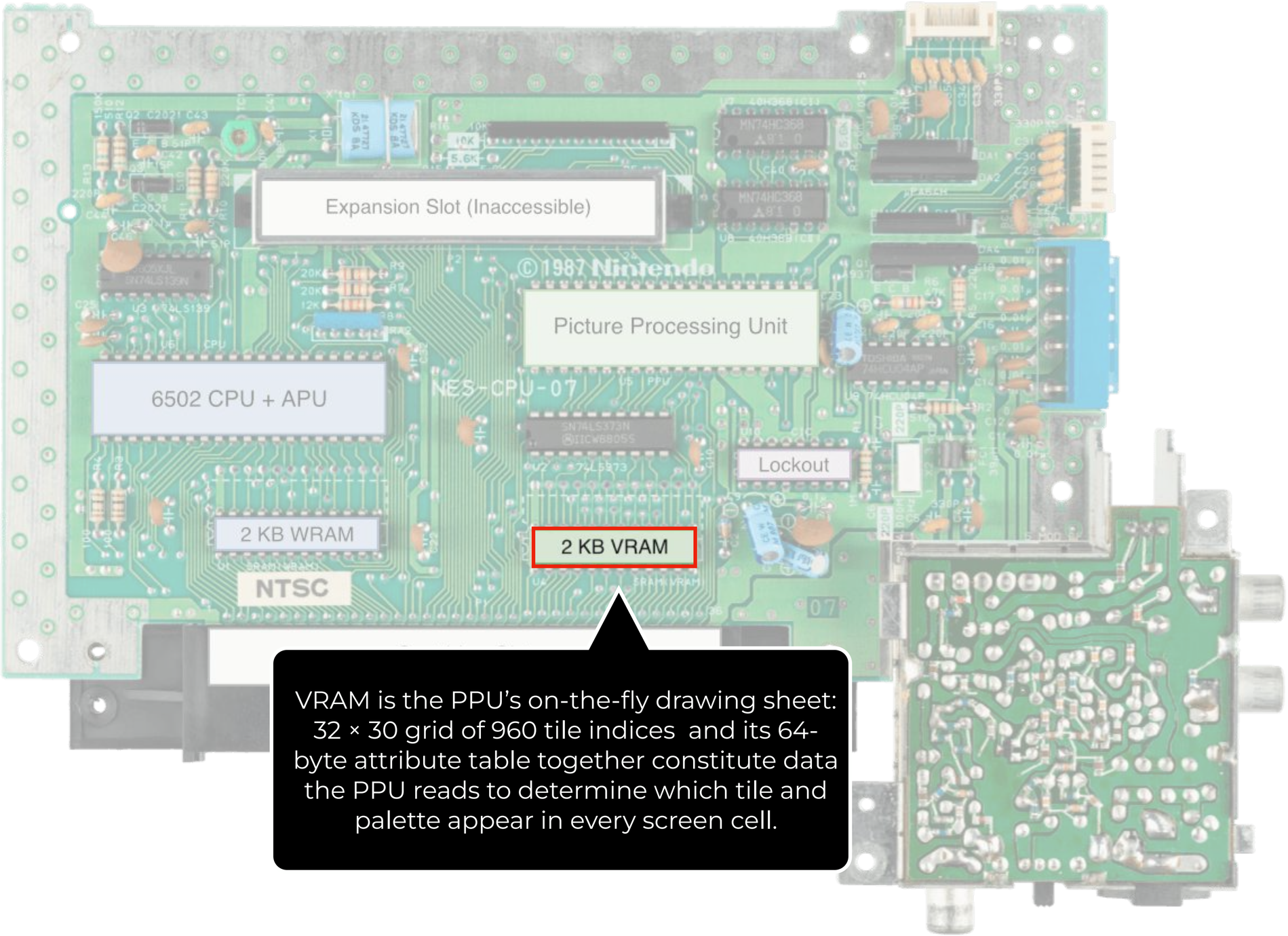
N

lse

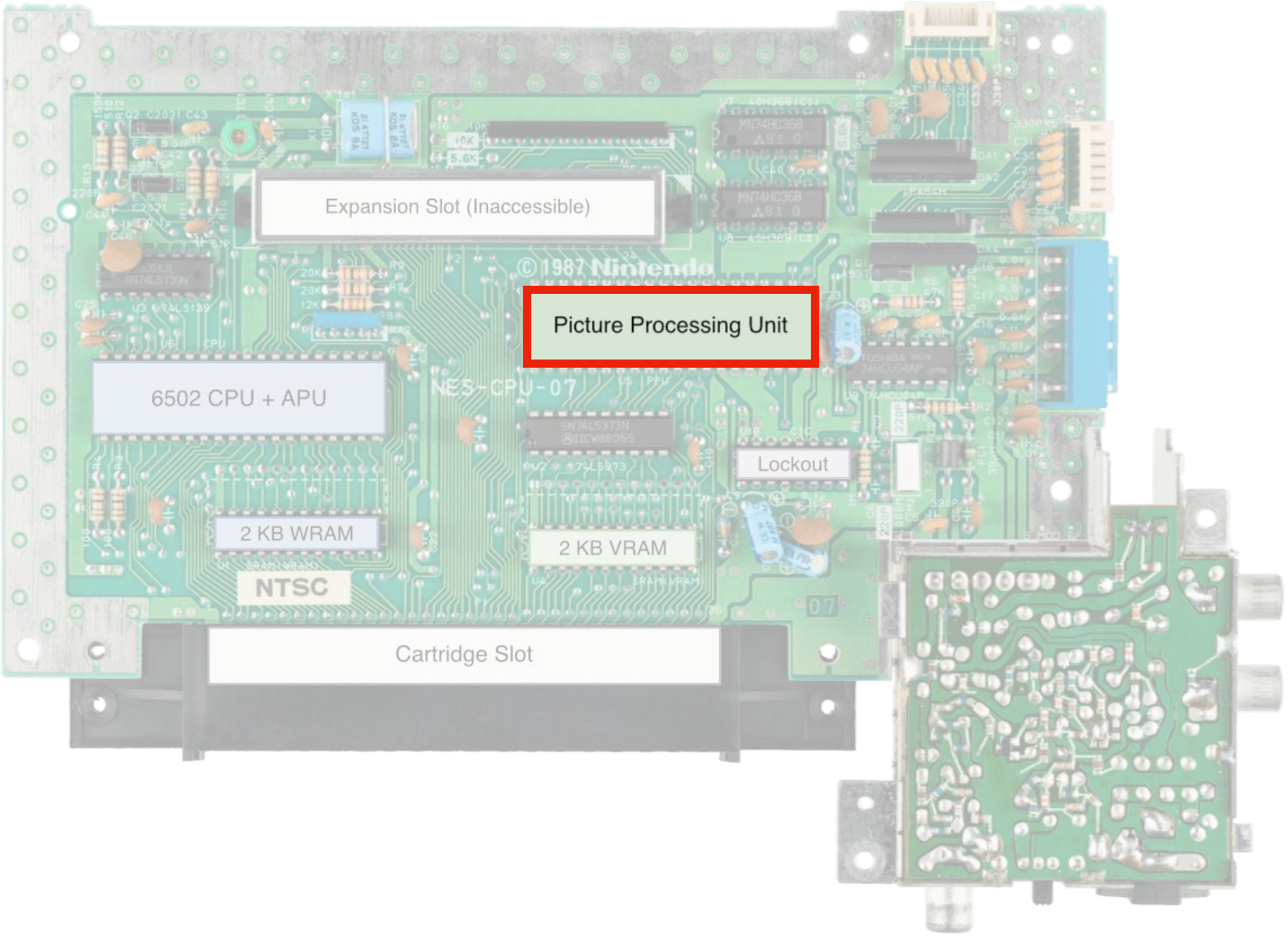
Pulse

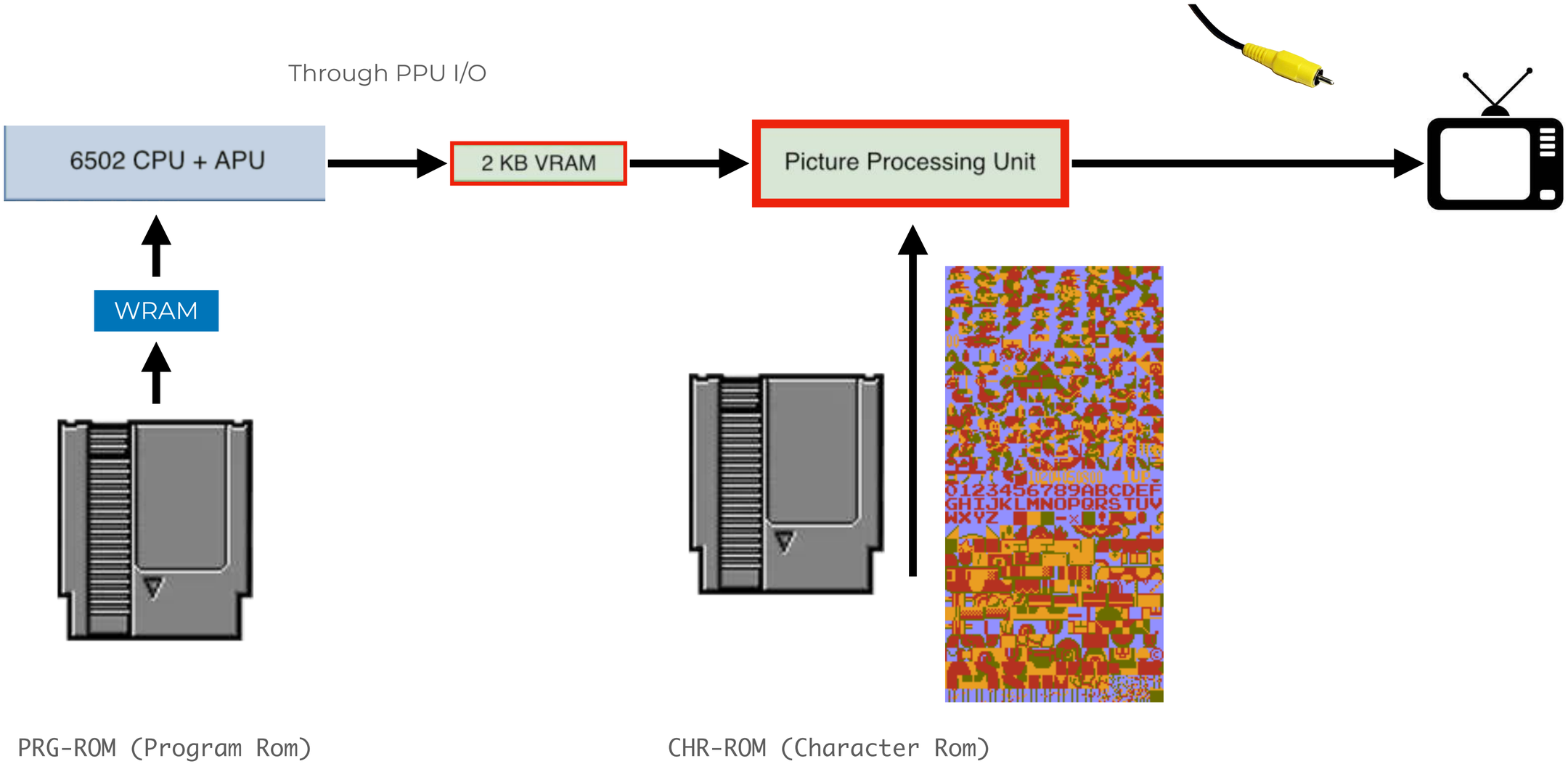


WRAM (CPU work RAM) – the processor's notepad: it stores life counts, player position, scores, and so on.



VRAM is the PPU's on-the-fly drawing sheet: 32 × 30 grid of 960 tile indices and its 64-byte attribute table together constitute data the PPU reads to determine which tile and palette appear in every screen cell.


Expansion Slot (Inaccessible)
Picture Processing Unit
6502 CPU + APU
Lockout
2 KB WRAM
2 KB VRAM
NTSC
Cartridge Slot



Picture Processing Unit

Now it’s time for true black magic - Cartidges

CD have only data

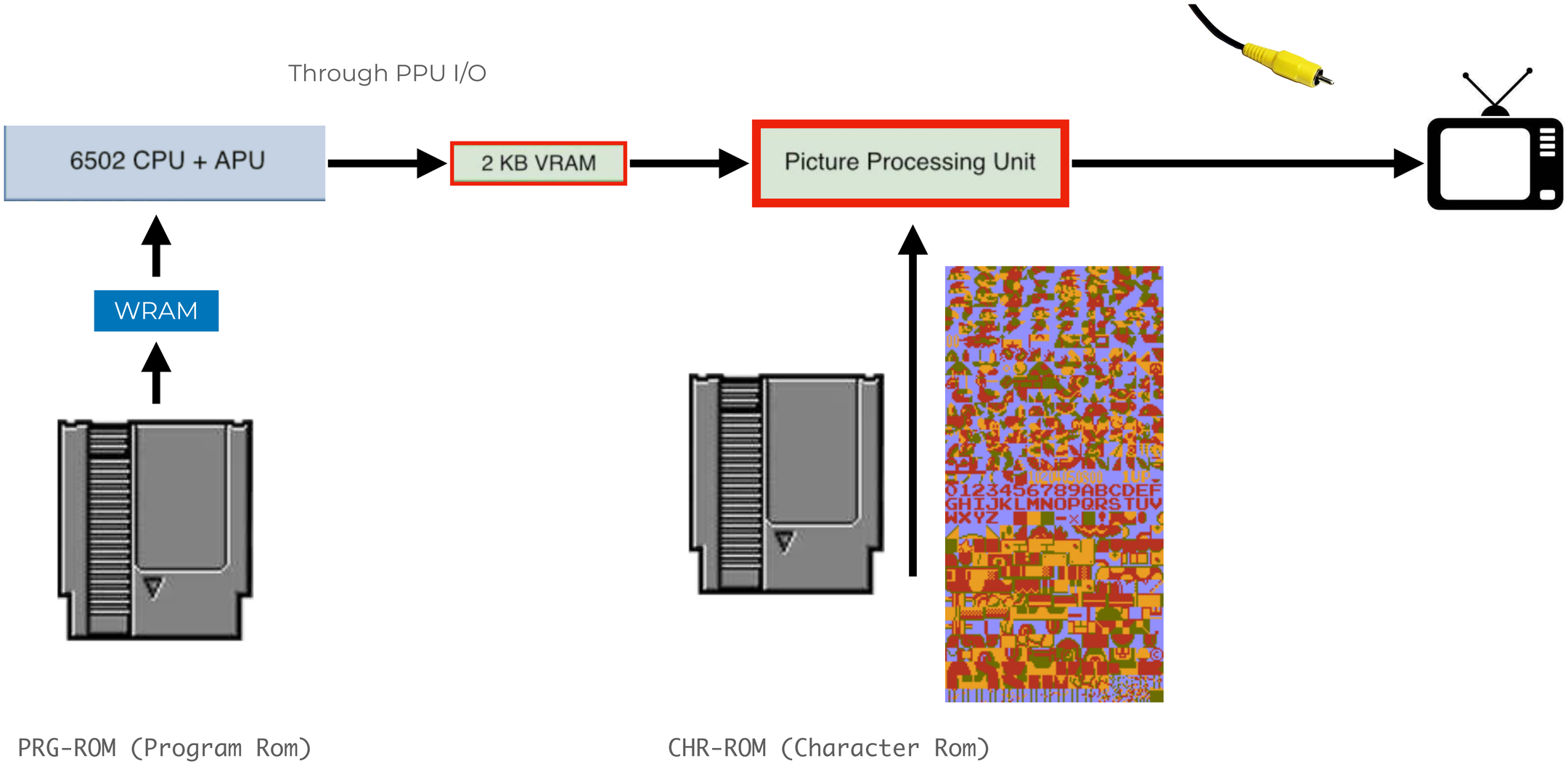
Most modern Cartidges are just memory cards

NES cartidge is a minicomputer

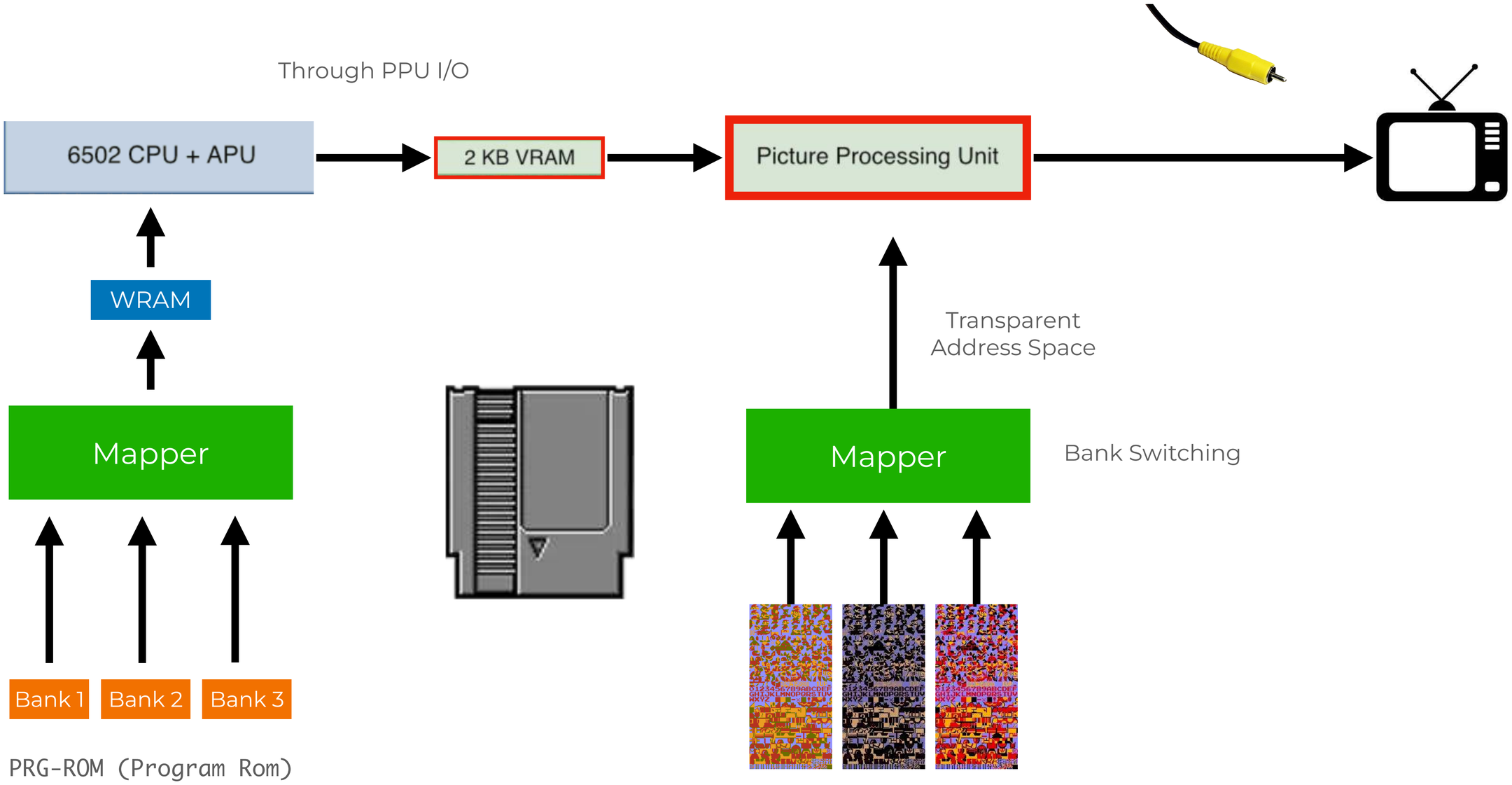
Cartidge is a minicomputer

Cartidge is a minicomputer

Cartidge



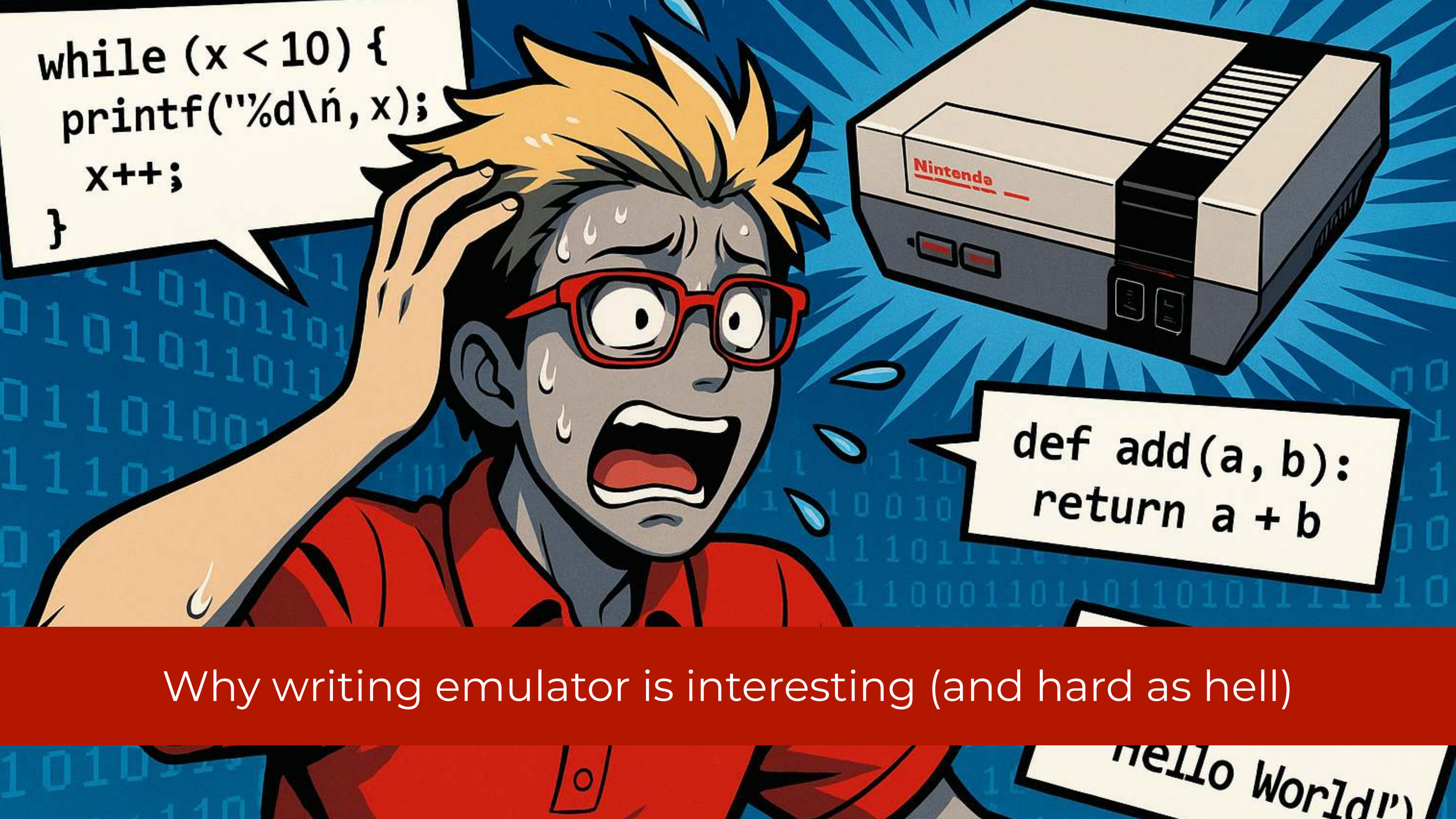
Picture Processing Unit


Through PPU I/O
6502 CPU + APU
2 KB VRAM
Picture Processing Unit
WRAM
Mapper
Bank 1
Bank 2
Bank 3
PRG-ROM (Program Rom)
Transparent
Address Space
Mapper
Bank Switching

Mappers

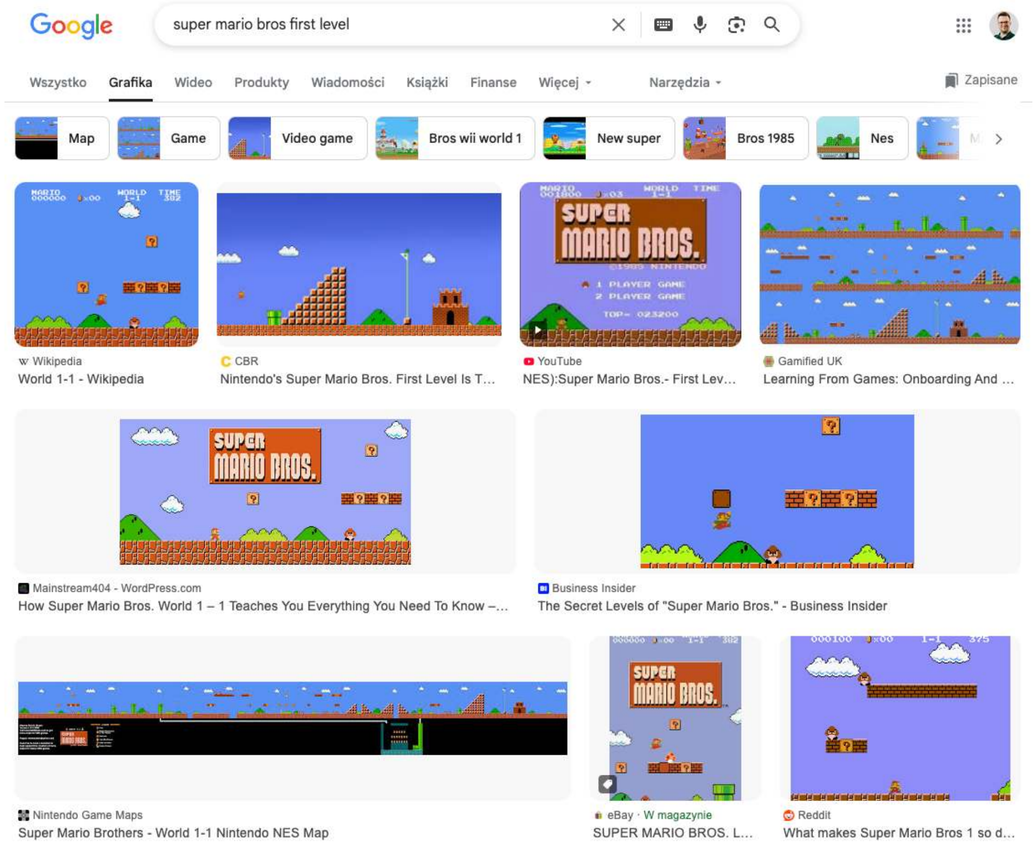
- Rather than the cartridge containing only the game program, it has a separate set of address and data lines that connect directly to the PPU, containing its graphics and tiles.
- Game cartridges can provide the RAM, if that's needed.
- They were really expensive to make
- There were also special Mappers for Sound

Cartidge



Why writing emulator is interesting (and hard as hell)

Fun Fact #1 - There is no color palette in NES



5990F5

5990F4

9DA8F8

6888F8

8E8CF8

949FCF

“Super Mario Bros” first level Google Search

5990F5

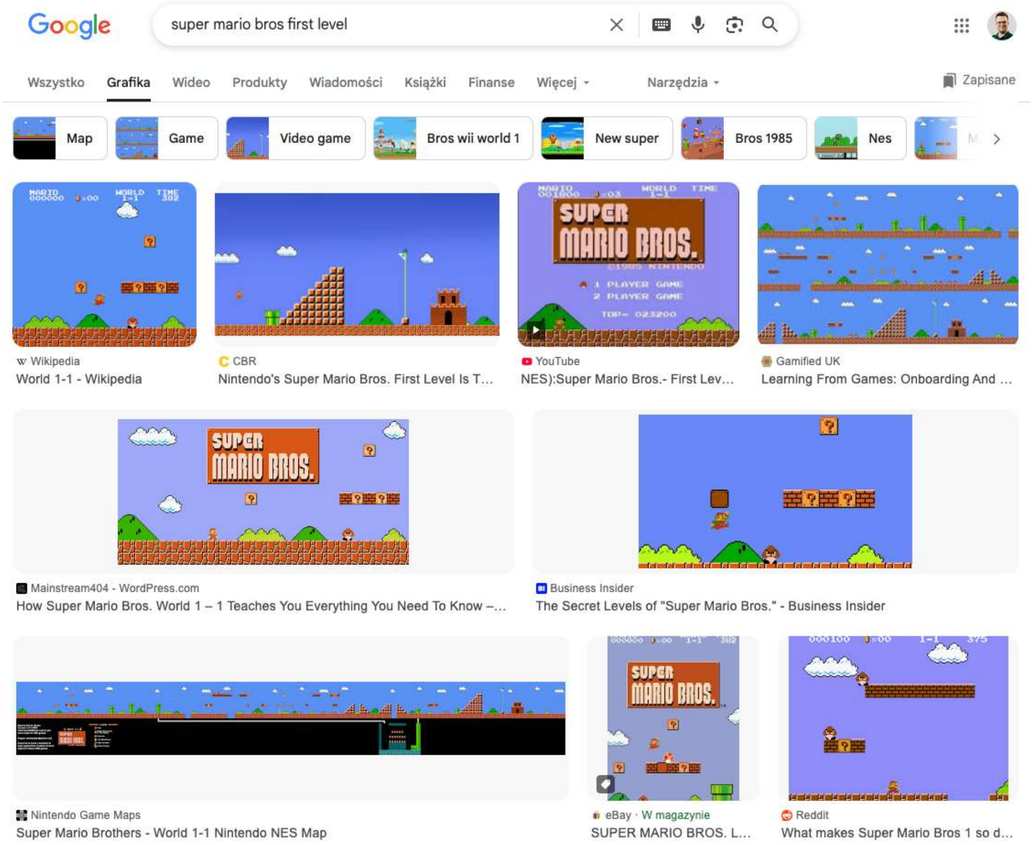
5990F4

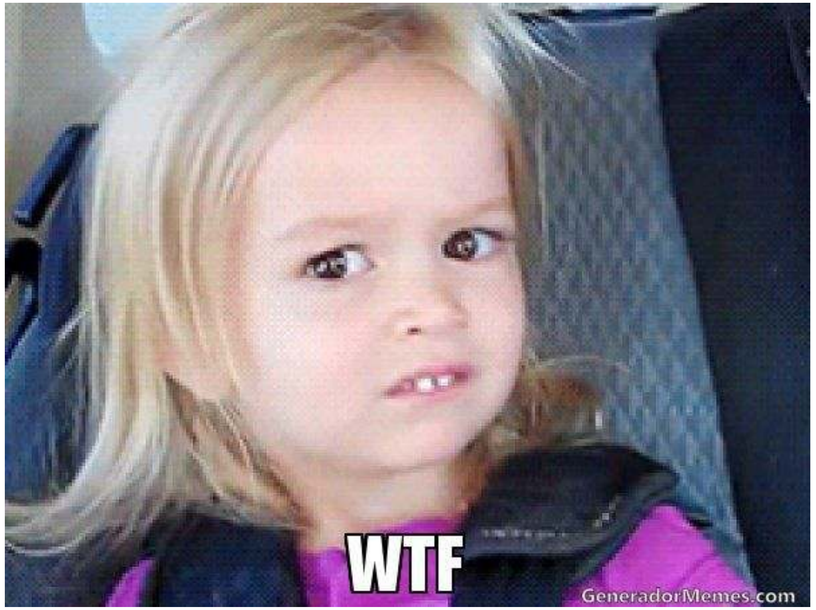
9DA8F8

6888F8

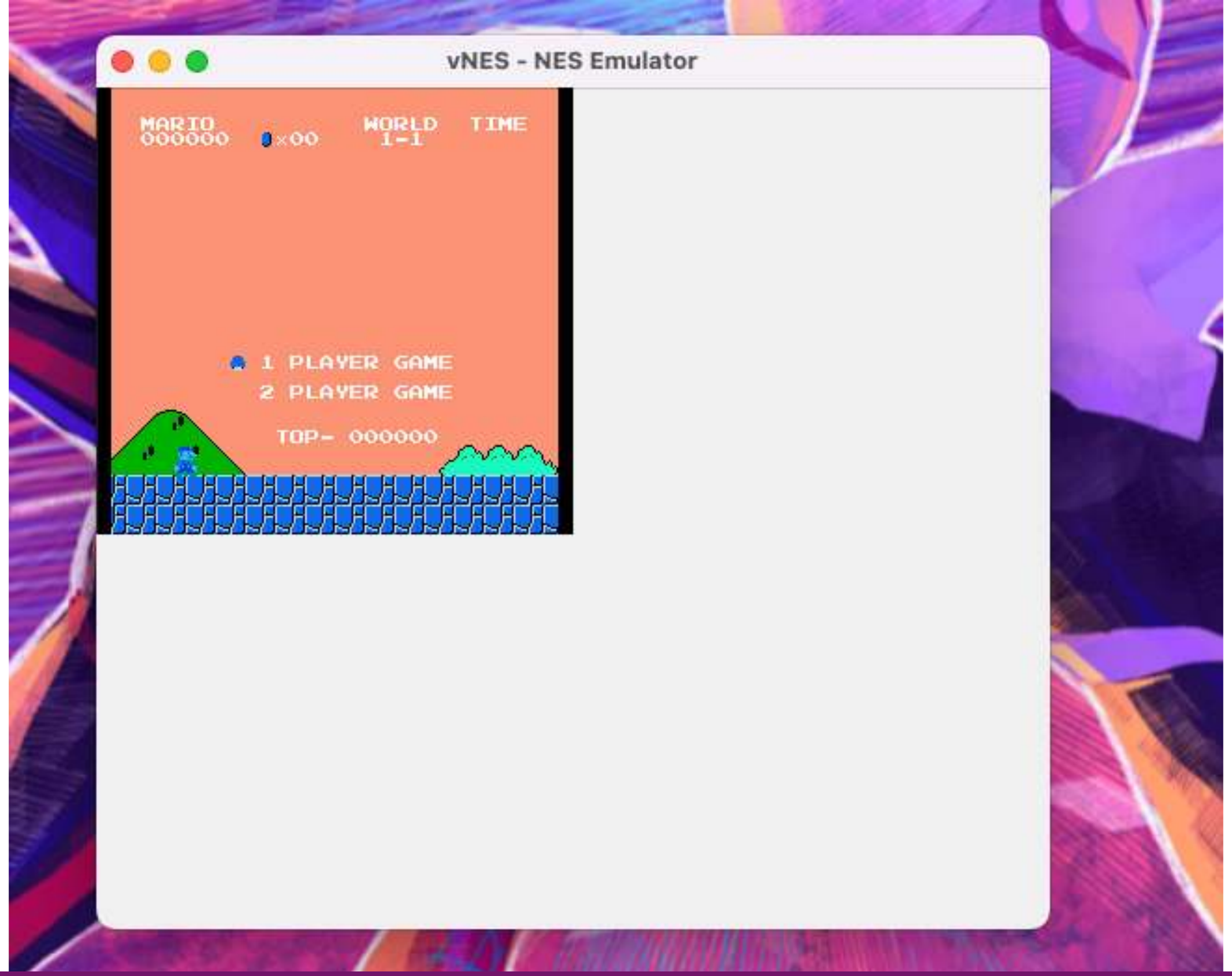
8E8CF8

949FCF





“Super Mario Bros” first level Google Search



There is no color palette in NES 🤦

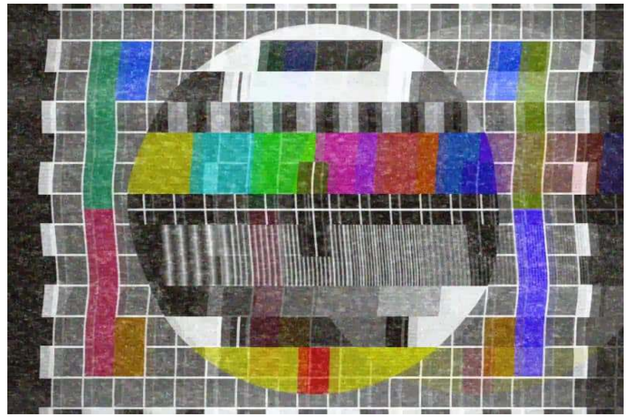
RGBA

Digital

There is no color palette in NES 🤷

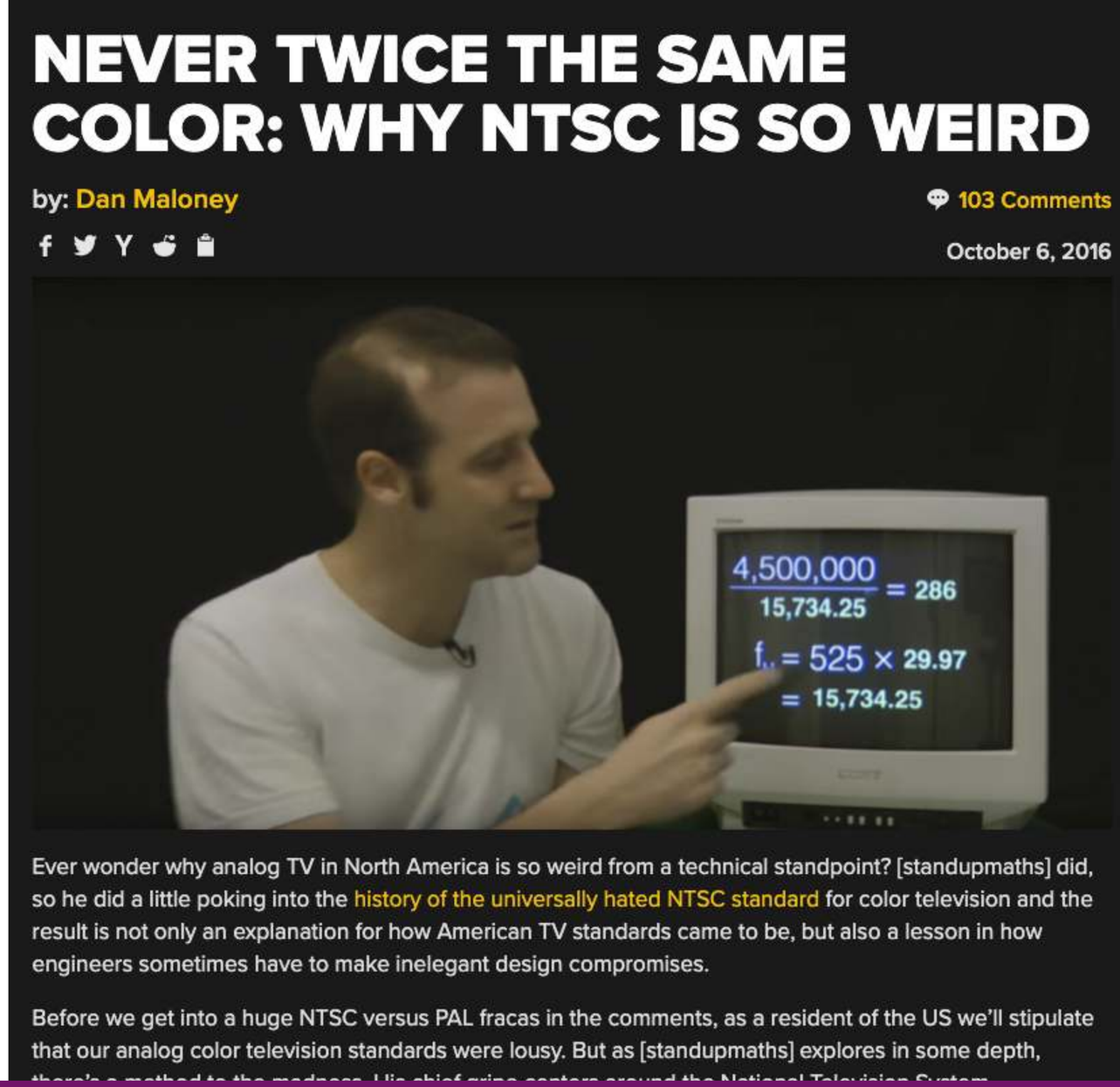
Radio Frequency!

Analog



There is no color palette in NES 🤷

One cable (yellow RCA), where the luminance (Y) and color (C) signals were mixed together.



NEVER TWICE THE SAME COLOR: WHY NTSC IS SO WEIRD

by: Dan Maloney

103 Comments

October 6, 2016

Ever wonder why analog TV in North America is so weird from a technical standpoint? [standupmaths] did, so he did a little poking into the history of the universally hated NTSC standard for color television and the result is not only an explanation for how American TV standards came to be, but also a lesson in how engineers sometimes have to make inelegant design compromises.

Before we get into a huge NTSC versus PAL fracas in the comments, as a resident of the US we'll stipulate that our analog color television standards were lousy. But as [standupmaths] explores in some depth,

Frames per second NTSC/PAL

The NES didn’t have a true combined RGB palette like modern devices. Its PPU generated colors from **16 brightness levels × 12 hues**, giving a theoretical total of **192 combinations**.

However, NES features a fixed palette of 64 colors, each assigned a unique index ranging from 0x00 to 0x3F. These indices are stored in the PPU palette memory and are used by games to reference specific colors when rendering backgrounds and sprites.

There is no color palette in NES 🤷

NesDev
Info, Programs, and More!

NES WIKI | SNES WIKI | WS WIKI | FORUMS | DISCORD

NesDev.org

A community of homebrew game developers and hardware researchers for the Nintendo Entertainment System (NES) and other retro consoles.

Wikis

- NES Wiki
- SNES Wiki
- WonderSwan Wiki
- Obscure Platforms Wiki

Community

- Forums
- NESdev Discord
- NESdev Compo 2023

Resources

- NES Cart Database

NES Dev

Palettes

The 2C02 (NTSC) and 2C07 (PAL) PPU is used to generate an analog composite video signal. These were used in most home consoles.

The 2C03, 2C04, and 2C05, on the other hand, all output analog red, green, blue, and sync (RGBS) signals. The sync signal contains horizontal and vertical sync pulses in the same format as an all-black composite signal. Each of the three video channels uses a 3-bit DAC driven by a look-up table in a 64x9-bit ROM inside the PPU. The look-up tables (one digit for each of red, green, and blue, in order) are given below.

RGB PPUs were used mostly in arcade machines (e.g. Vs. System, Playchoice 10), as well as the Sharp Famicom Titler.

2C02

The RF Famicom, AV Famicom, NES (both front- and top-loading), and the North American version of the Sharp Nintendo TV use the 2C02 PPU. Unlike some other consoles' video circuits, the 2C02 does not generate RGB video and then encode that to composite. Instead it generates NTSC video directly in the composite domain, decoded by the television receiver into RGB to drive its picture tube.

> **Do not use color $0D. It results in a "blacker than black" signal that may cause problems for some TVs.**
> Further details on $0D and its effects can be found here.

Most emulators can use a predefined palette, such as one commonly stored in common .pal format, in which each triplet represents the sRGB color that results from decoding a large flat area with a given palette value.

See this page for more details on a general algorithm to decode a PPU composite signal to color RGB.

The following palette was generated using Persune's palette generator v0.17.0 with the following arguments:

```
palgen_persune.py --skip-plot -cld -phs -5.0 -e -o docs/NESDev/2C02G_wiki
```

Download: File:2C02G wiki.pal

$00	$01	$02	$03	$04	$05	$06	$07	$08	$09	$0A	$0B	$0C	~~$0D~~	$0E	$0F
$10	$11	$12	$13	$14	$15	$16	$17	$18	$19	$1A	$1B	$1C	$1D	$1E	$1F
$20	$21	$22	$23	$24	$25	$26	$27	$28	$29	$2A	$2B	$2C	$2D	$2E	$2F
$30	$31	$32	$33	$34	$35	$36	$37	$38	$39	$3A	$3B	$3C	$3D	$3E	$3F

Other tools for generating a palette include one by Bisqwit and one by Drag. These simulate generating a large area of one flat color and then decoding that with the adjustment knobs set to various settings.

2C07

The PAL PPU (2C07) generates a composite PAL video signal, which has a -15 degree hue shift relative to the 2C02 due to a different colorburst reference phase generated by the PPU ($x7 rather than $x8), in addition to the PAL colorburst phase being defined as -U ± 45 degrees.

The following palette was generated using Persune's palette generator v0.17.0 with the following arguments:

```
palgen_persune.py --skip-plot -cld -ppu "2C07" -phs -5.0 --delay-line-filter -e -o docs/NESDev/2C07_wiki
```

Download: File:2C07 wiki.pal

$00	$01	$02	$03	$04	$05	$06	$07	$08	$09	$0A	$0B	$0C	~~$0D~~	$0E	$0F
$10	$11	$12	$13	$14	$15	$16	$17	$18	$19	$1A	$1B	$1C	$1D	$1E	$1F
$20	$21	$22	$23	$24	$25	$26	$27	$28	$29	$2A	$2B	$2C	$2D	$2E	$2F
$30	$31	$32	$33	$34	$35	$36	$37	$38	$39	$3A	$3B	$3C	$3D	$3E	$3F

2C03 and 2C05

This palette is intentionally similar to the NES's standard palette, but notably is missing the grayscale entries. The 2C03 is used in Vs. Duck Hunt, 2C04 games, and PlayChoice games, and the Sharp C1 (Famicom TV). The 2C05 is used in some later Vs. games as a copy protection measure. A variant of the 2C05 without copy protection measures is present in the Sharp Famicom Titler, albeit with adjustments to the output (see below). Both have historically been used in RGB mods for the NES, as a circuit implementing `A0' = A0 xor (A1 nor A2)` can swap PPUCTRL and PPUMASK to make a 2C05 behave as a 2C03.

https://www.nesdev.org/wiki/PPU_palettes

There are particular set of pallets “agreed”

ntsc.txt

```
#525252
#000080
#08008A
#2C007E
#4A004E
#500006
#440000
#260800
#0A2000
#002E00
#003200
#00260A
#001C48
#000000
#000000
#000000

#A4A4A4
#0038CE
#3416EC
#5E04DC
#8C00B0
#9A004C
#901800
#703600
#4C5400
#0E6C00
#007400
#006C2C
#005E84
#000000
#000000
#000000
```

```
fun loadPalette(file: String): Boolean {
    try {
        if (file.lowercase().endsWith("pal")) {
            val fStr = javaClass.classLoader.getResourceAsStream(file)
            val tmp = ByteArray(64 * 3)

            var n = 0
            while (n < 64) {
                n += fStr!!.read(tmp, n, tmp.size - n)
            }

            val tmpi = IntArray(64 * 3)
            for (i in tmp.indices) {
                tmpi[i] = tmp[i].toInt() and 0xFF
            }

            for (i in 0 until 64) {
                val r = tmpi[i * 3 + 0]
                val g = tmpi[i * 3 + 1]
                val b = tmpi[i * 3 + 2]
                origTable[i] = r or (g shl 8) or (b shl 16)
            }
        } else {
```

There are particular set of pallets “agreed”

ntsc.txt

```
#525252
#000080
#08008A
#2C007E
#4A004E
#500006
#440000
#260800
#0A2000
#002E00
#003200
#00260A
#001C48
#000000
#000000
#000000

#A4A4A4
#0038CE
#3416EC
#5E04DC
#8C00B0
#9A004C
#901800
#703600
#4C5400
#0E6C00
#007400
#006C2C
#005E84
#000000
#000000
#000000
```

```
val hsl = RGBtoHSL(emphTable[currentEmph][i])
var h = getHue(hsl) + hueAdd
var s = (getSaturation(hsl) * (1.0 + saturationAdd / 256f)).toInt()
var l = getLightness(hsl)

r = 128 + lightnessAdd + ((r - 128) * (1.0 + contrastAddValue / 256f)).toInt()
g = 128 + lightnessAdd + ((g - 128) * (1.0 + contrastAddValue / 256f)).toInt()
b = 128 + lightnessAdd + ((b - 128) * (1.0 + contrastAddValue / 256f)).toInt()

colorTable[0] = getRgb(124, 124, 124)
colorTable[1] = getRgb(0, 0, 252)
colorTable[2] = getRgb(0, 0, 188)
colorTable[3] = getRgb(68, 40, 188)
colorTable[4] = getRgb(148, 0, 132)
colorTable[5] = getRgb(168, 0, 32)
colorTable[6] = getRgb(168, 16, 0)
colorTable[7] = getRgb(136, 20, 0)
colorTable[8] = getRgb(80, 48, 0)
colorTable[9] = getRgb(0, 120, 0)
(…) /// All 64 (or 52…) needs to be mapped.
```

There are particular set of pallets “agreed”

Fun Fact #2 - Emulating “mappers” is the worst

nicole express

All Posts Games @ itch.io

What made the NES so interesting?

Mar 26, 2022

I often like to cover oddities here; details of computers and arcade systems people may not have heard of, that didn't sell well but had unique or interesting characteristics. But focusing on oddities like that can disguise the fact that sometimes, even systems that were very popular can stand out for unique design. Such be it with the Nintendo Entertainment System: had Nintendo's console flopped, it'd definitely have a post already. But it doesn't. Let's fix that! And answer a simple question I was asked: what's the deal with special chip games?

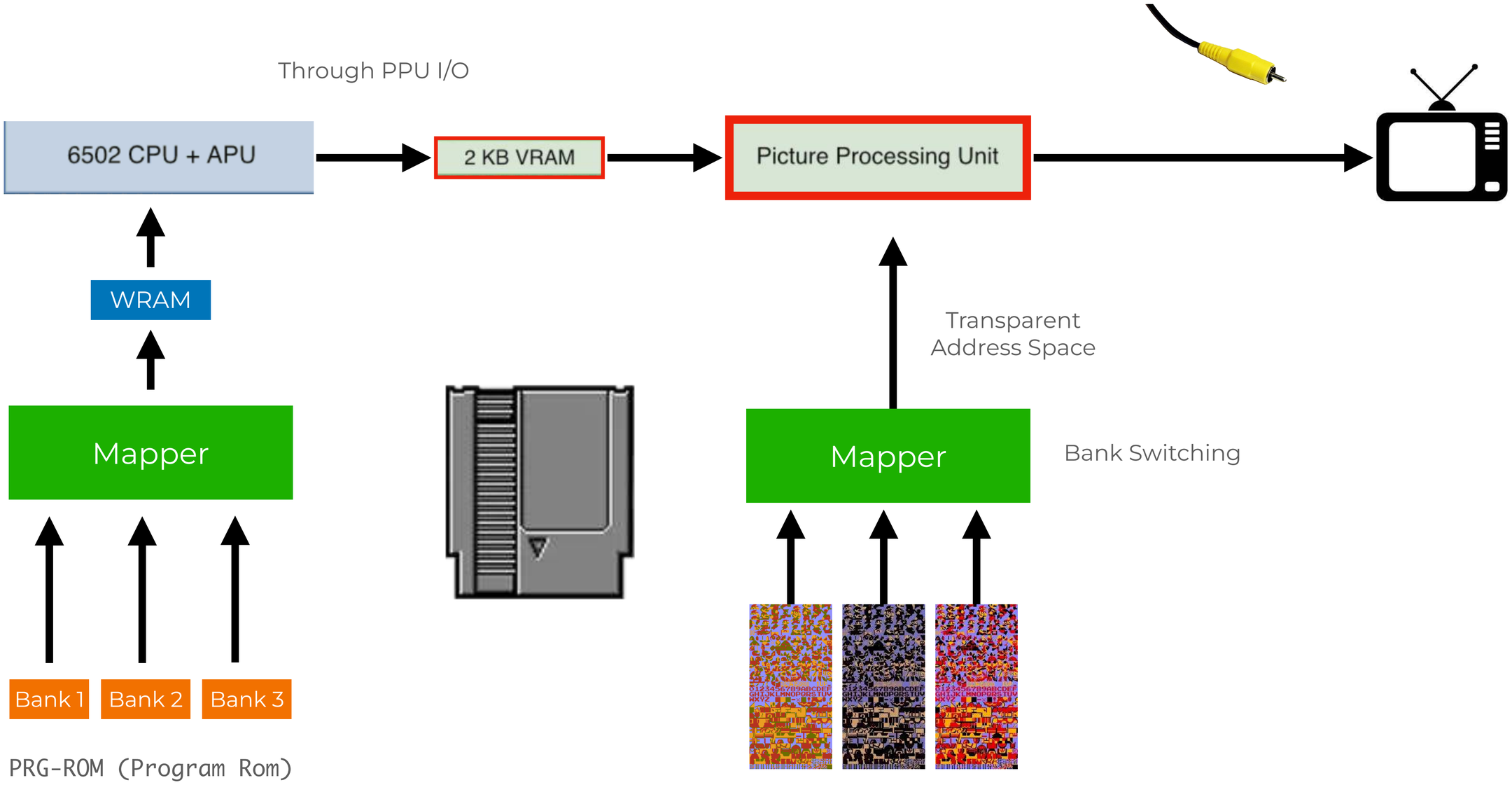
Nintendo MMC3

Another thing came along in the end of 1988: the MMC3 enhancement chip. This chip powered a lot of the console's mega-hits, including *Super Mario Bros. 3*, the American *Super Mario Bros. 2*, *Adventure Island IV*, *Shadowgate*, *Kirby's Adventure*, and the last licensed NES game in North America, *Wario's Woods*. NESdev claims a list of 300 titles. Not as many as MMC1, but not too shabby.

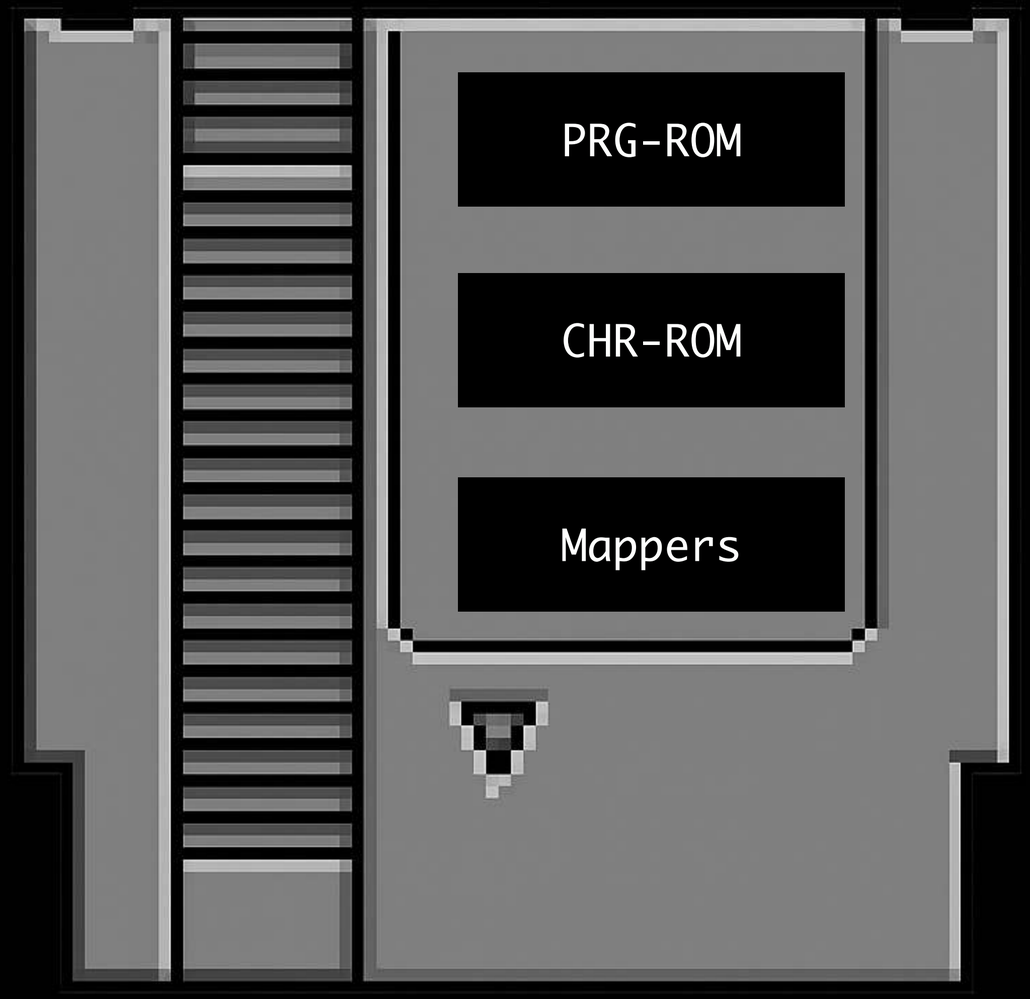
The MMC3 pictured above comes from the game *Legacy of the Wizard*, whose Japanese version, *Dragon Slayer 4: Drasle Family* used a Namco 109 chip just like *Gauntlet*, though it didn't have the additional nametable RAM. As it turns out, the MMC3's PRG and CHR banking match the Namco 108 family. From its release in the fourth quarter of 1988, MMC3 began to be used for games ported from the Namco 108 chips.

Fun Fact #2 - Emulating "mappers" is the worst

May you live in interesting times



Remember when we talk about mappers



There was A LOT OF different mappers 🥶

Cartidge

ASIC mappers with simple banking

Also see Comparison of Nintendo mappers

		PRG		CHR					
iNES Mapper	Short name	bank style	max (KiB)	bank style	max (KiB)	Mirroring	IRQ	WRAM	Comments
206	N108	8+8+16F	128	2×2+4×1	64	fixed	none	none	none
4	MMC3, MMC6	N108; 8F+8+8+8F	512	N108; 4×1+2×2	256	H/V	scanline	ext (MMC6 1024B int)	
(none)	Famicom Disk System	32RAM+8F	arbitrary	unbanked	8	H/V	cycle	none	Expansion audio
1/155	MMC1	16+16F; 16F+16; 32	256 (512)	8; 2×4	128	1/H/V	none	sometimes bankable ext	
5	MMC5	32; 2×16; 16+8+8; 4×8	1024	8; 2×4; 4×2; 8×1	1024	arbitrary	scanline	bankable ext + 1024B int	Expansion audio, vertical split, hardware multiplier, 8x8 attribute zones
9	MMC2	8+24F	128	2×4	128	H/V	none	none	tile triggered CHR banking
10	MMC4	16+16F	256	2×4	128	H/V	none	ext	tile triggered CHR banking
16/159	LZ93D50 (I²C)	16+16F	256	8×1	256	1/H/V	cycle	serial EEPROM	
18	Jaleco SS88006	3×8+8F	256?	8×1	256	1/H/V	cycle	ext	ADPCM expansion audio
19	Namco 163	3×8+8F	512	12×1	256	arbitrary	cycle	ext + 128B int	Expansion audio, ROM nametables
21/23/25	VRC4	MMC3	256	8×1	512	1/H/V	scanline	ext	
22/23/25	VRC2	N108	256	8×1	256	1/H/V	none	serial EEPROM	
24/26	VRC6	16+8+8F	256	8×1; 6×1+2×3; 4×2+4×1; 4×1+4×2	256 (512)	arbitrary	scanline	ext	Expansion audio, ROM nametables
27	"Pirate" VRC4	3×8+8F	?256?	8×1	512	1/H/V	?	?	
31	NSF subset	8×4	1024	unbanked	8	fixed	none	none	
32	Irem G101	MMC3	256	8×1	256	H/V	none	none	
33/48	TC0190	N108	512	N108	512	H/V	scanline(48)	ext	
64	RAMBO-1	3×8+8F	256	MMC3; 8×1	256	H/V	scanline/cycle	none	
65	Irem H3001	3×8+8F	512	8×1	256	H/V	cycle	ext	
67	Sunsoft 3	16+16F	128?	4×2	128?	1/H/V	cycle	ext	
68	Sunsoft 4	16+16F	256	4×2	256	1/H/V	none	ext	ROM nametables
69	Sunsoft FME-7	4×8+8F	256	8×1	256	1/H/V	cycle	ext	Expansion audio
73	VRC3	16+16F	128	unbanked	8	fixed	cycle	ext	
74/194	TQROM-like (2kiB)	MMC3	1024	MMC3	254	1/H/V?	scanline?	ext?	CHR RAM and ROM, ?like TQROM?
75	VRC1	3×8+8F	128	2×4	128	H/V	none	none	
76	N108 (inflated CHR)	N108	128	4×2	128	fixed	none	none	
80	X1-005	3×8+8F	256	N108	256	H/V	none	128B int	
82	X1-017	3×8+8F	512?	MMC3	256	H/V	cycle	5120B int	
85	VRC7	3×8+8F	512	8×1	256	1/H/V	scanline	ext	Expansion audio
90	for: Tekken 2 HKO	8+32F; 8+16+16F; 4×8+8F; 8+32; 8+2×16; 4×8	2048?	8; 2×4; 4×2; 8×1	2048?	1/H/V	scanline/cycle/pixel	?	hardware multiplier
209/211	for: Tekken 2 HKO	8+32F; 8+16+16F; 4×8+8F; 8+32; 8+2×16; 4×8	2048?	8+4×1; 2×4+4×1; 4×2+4×1; 12×1	2048?	arbitrary	scanline/cycle/pixel	?	ROM nametables, hardware multiplier
91	for: Street Fighter 3 HKO	N108	128	4×2	512	?	?	?	
95	N108 (nametable)	N108	128	N108	32	1/H	none	none	
97	Irem TAM-S1	16F+16	256	unbanked	8	1/H/V	none	none	
105	NES-EVENT	16+16F; 16F+16; 32	256	1×8; 2×4	8	1/H/V	other	ext	
112	"chinese" N108	N108	256	N108	256	H/V	?	?	
118	TLSROM, TKSROM	MMC3	512	MMC3	128	arbitrary	scanline	ext	
119	TQROM (8kiB)	MMC3	512	MMC3	64	H/V	scanline	ext	CHR RAM and ROM
137	Sachen 8259 type D	32	256	4×1+1×4	32	1/H/V/L	none	?	
153	LZ93D50 (RAM)	16+16F	512	unbanked	8	1/H/V	cycle	ext	
157	Bandai Datach	16+16F	256	unbanked	8	1/H/V	cycle	serial EEPROMs	barcode reader
158	RAMBO-1 (nametable)	3×8+8F	256	MMC3; 8×1	128	arbitrary	scanline/cycle	none	
165	MMC2 × MMC3	MMC3	512	2×4	128	?	?	?	tile triggered CHR banking, CHR RAM and ROM
182	for: Pocahontas	MMC3	256 (512?)	MMC3	256	H/V	scanline	?	?
189	MMC3 × BNROM	32	512	MMC3	256	?	?	none	
191	TQROM-like (2kiB)	MMC3	256 (512?)	MMC3	128	1/H/V?	scanline?	ext?	CHR RAM and ROM, ?like TQROM?
192	TQROM-like (4kiB)	MMC3	512?	MMC3	252	1/H/V?	scanline?	ext?	CHR RAM and ROM, ?like TQROM?

Mappers with outer banks

		PRG			CHR			
iNES Mapper	Short name	inner	outer	max (KiB)	inner	outer	max (KiB)	Comments
12	for: dbz5	MMC3	—	512	MMC3	2×256	512	?like MMC3?
14	MMC3 × VRC2	MMC3	—	512	MMC3; 8×1	3×256	512	H/V, ext WRAM, Scanline IRQ, ?like MM
28	Action 53	16+16F; 16F+16; 32	32…256	8192	8	—	32RAM	1/H/V
37	MMC3+'00+'161	MMC3	64; 128	256	MMC3	128	256	H/V, Scanline IRQ
44	for: Super Big 7-in-1	MMC3	128; 256	1024	MMC3	128; 256	1024	?like MMC3?
45	for: Super 4-in-1	MMC3	8…512	1024 (2048?)	MMC3	1…256	1024 (4096?)	?like MMC3?
47	MMC3+'161	MMC3	128	256	MMC3	128	256	H/V, Scanline IRQ
49	for: Super HIK 4-in-1	MMC3; 32	128	512	MMC3	128	512	?like MMC3?
52	for: Mario 7-in-1	MMC3	128; 256	1024	MMC3	128; 256	1024	?like MMC3?
88/154	N108 (pattern split)	N108	—	128	2×2/64 + 4×1/64	2×64	128	none
115	for: Thunderbolt 2 (ch)	MMC3; 16+16F; 16+8+8F	—	128 (256?)	MMC3	256	512	?like MMC3?
116	MMC1 × MMC3 × VRC2	MMC3; 16+16F; 16F+16; 32	—	256	MMC3; 2×4; 8×1	256	512	1/H/V, ?like MMC3?
138	Sachen 8259 type B	32	—	256	4×2	16	128	1/H/V/L
139	Sachen 8259 type C	32	—	256	4×2	32	256	1/H/V/L
141	Sachen 8259 type A	32	—	256	4×2	64	512	1/H/V/L
205	for: 3-in-1	MMC3	128; 256	512	MMC3	128; 256	512	?like MMC3?
245	MMC3 × SUROM	MMC3	512	1024	unbanked	—	8	?like MMC3?
(none)	UNIF/COOLBOY	MMC3; 16; 32	128…2048	32768	MMC3; 8	128; 256	256RAM	like MMC3

Discrete logic mappers

Only the ones not made by Nintendo.

This table describes the mappers as they existed, as opposed to any obvious oversize extensions.

See also Comparison of Nintendo mappers and User:Lidnariq/Discrete Logic Table.

You probably don't actually want to use these.

iNES	Chips	Max PRG (KiB)	PRG bank style	Max CHR (KiB)	CHR bank style	Mirroring	PRG RAM?	Bus conflicts?
Color Dreams (11, 144)	1	128	32	128	8	V/H hardwired	No	Yes
29 (RET-CUFROM)	3	128	16+16F	32	8	V hardwired	Yes	No
RET-UNROM 512 (30)	3 or 4	512	16+16F	32RAM	8	1 or V/H hardwired	No	Optional (extra IC)
NINA-001 (34)	6	64	32	64	4+4	V hardwired	Yes	No
38	2	128	32	32	8	V/H hardwired	Impossible	No
40 / 50 (w/ IRQ)	7-8	128	24F+8+8F	8		V/H hardwired	Impossible	No
41	7	256	32	128	8 inner / 32 outer	V/H switchable	Impossible	Partly
70	3	256	16+16F	128	8	V/H hardwired	No	Likely
72	4+speech	256	16+16F	128	8	V/H hardwired	No	Yes
77	4	512	32	32+6RAM	2+6RAM	4	No	Likely
78.1	3	128	16+16F	128	8	1	No	Likely
78.3	5	128	16+16F	128	8	V/H switchable	No	Yes
79 (146)	2	64	32	64	8	V/H hardwired	No	No
86	3+speech	128	32	64	8	V/H hardwired	Impossible	No
87 (101)	2	32		32	8	V/H hardwired	Impossible	No
89	2†	128	16+16F	128	8	1	No	Yes
92	5+speech	256	16F+16	128	8	V/H hardwired	No	Yes
93	2†	128	16+16F	8‡		V/H hardwired	No	Yes
94	2	128	16+16F	8		V/H hardwired	No	Yes
96	3	128	32	32RAM	4+4F inner / 16 outer	V/H hardwired	No	Likely
99	0*	40	8+24F	16	8	4	No	No
107	1?	128	32	64	8	V hardwired	No	Likely
133	2?	64	32	32	8	V/H hardwired	No	No
140	3	128	32	128	8	V/H hardwired	Impossible	No
143	2?	32		8		V/H hardwired	No	No
145	2?	32		16	8	V/H hardwired	No	No

Around 100

ASIC mappers with simple banking

Also see Comparison of Nintendo mappers

		PRG		CHR					
iNES Mapper	Short name	bank style	max (KiB)	bank style	max (KiB)	Mirroring	IRQ	WRAM	Comments
206	N108	8+8+16F	128	2×2+4×1	64	fixed	none	none	none
4	MMC3, MMC6	N108; 8F+8+8+8F	512	N108; 4×1+2×2	256	H/V	scanline	ext (MMC6 1024B int)	
(none)	Famicom Disk System	32RAM+8F	arbitrary	unbanked	8	H/V	cycle	none	Expansion audio
1/155	MMC1	16+16F; 16F+16; 32	256 (512)	8; 2×4	128	1/H/V	none	sometimes bankable ext	
5	MMC5	32; 2×16; 16+8+8; 4×8	1024	8; 2×4; 4×2; 8×1	1024	arbitrary	scanline	bankable ext + 1024B int	Expansion audio, vertical split, hardware multiplier, 8x8 attribute zones
9	MMC2	8+24F	128	2×4	128	H/V	none	none	tile triggered CHR banking
10	MMC4	16+16F	256	2×4	128	H/V	none	ext	tile triggered CHR banking
16/159	LZ93D50 (I²C)	16+16F	256	8×1	256	1/H/V	cycle	serial EEPROM	
18	Jaleco SS88006	3×8+8F	256?	8×1	256	1/H/V	cycle	ext	ADPCM expansion audio
19	Namco 163	3×8+8F	512	12×1	256	arbitrary	cycle	ext + 128B int	Expansion audio, ROM nametables
21/23/25	VRC4	MMC3	256	8×1	512	1/H/V	scanline	ext	
22/23/25	VRC2	N108	256	8×1	256	1/H/V	none	serial EEPROM	
24/26	VRC6	16+8+8F	256	8×1; 6×1+2×3; 4×2+4×1; 4×1+4×2	256 (512)	arbitrary	scanline	ext	Expansion audio, ROM nametables
27	"Pirate" VRC4	3×8+8F	?256?	8×1	512	1/H/V	?	?	
31	NSF subset	8×4	1024	unbanked	8	fixed	none	none	
32	Irem G101	MMC3	256	8×1	256	H/V	none	none	
33/48	TC0190	N108	512	N108	512	H/V	scanline(48)	ext	
64	RAMBO-1	3×8+8F	256	MMC3; 8×1	256	H/V	scanline/cycle	none	
65	Irem H3001	3×8+8F	512	8×1	256	H/V	cycle	ext	
67	Sunsoft 3	16+16F	128?	4×2	128?	1/H/V	cycle	ext	
68	Sunsoft 4	16+16F	256	4×2	256	1/H/V	none	ext	ROM nametables
69	Sunsoft FME-7	4×8+8F	256	8×1	256	1/H/V	cycle	ext	Expansion audio
73	VRC3	16+16F	128	unbanked	8	fixed	cycle	ext	
74/194	TQROM-like (2kiB)	MMC3	1024	MMC3	254	1/H/V?	scanline?	ext?	CHR RAM and ROM, ?like TQROM?
75	VRC1	3×8+8F	128	2×4	128	H/V	none	none	
76	N108 (inflated CHR)	N108	128	4×2	128	fixed	none	none	
80	X1-005	3×8+8F	256	N108	256	H/V	none	128B int	
82	X1-017	3×8+8F	512?	MMC3	256	H/V	cycle	5120B int	
85	VRC7	3×8+8F	512	8×1	256	1/H/V	scanline	ext	Expansion audio
90	for: Tekken 2 HKO	8+32F; 8+16+16F; 4×8+8F; 8+32; 8+2×16; 4×8	2048?	8; 2×4; 4×2; 8×1	2048?	1/H/V	scanline/cycle/pixel	?	hardware multiplier
209/211	for: Tekken 2 HKO	8+32F; 8+16+16F; 4×8+8F; 8+32; 8+2×16; 4×8	2048?	8+4×1; 2×4+4×1; 4×2+4×1; 12×1	2048?	arbitrary	scanline/cycle/pixel	?	ROM nametables, hardware multiplier
91	for: Street Fighter 3 HKO	N108	128	4×2	512	?	?	?	
95	N108 (nametable)	N108	128	N108	32	1/H	none	none	
97	Irem TAM-S1	16F+16	256	unbanked	8	1/H/V	none	none	
105	NES-EVENT	16+16F; 16F+16; 32	256	1×8; 2×4	8	1/H/V	other	ext	
112	"chinese" N108	N108	256	N108	256	H/V	?	?	
118	TLSROM, TKSROM	MMC3	512	MMC3	128	arbitrary	scanline	ext	
119	TQROM (8kiB)	MMC3	512	MMC3	64	H/V	scanline	ext	CHR RAM and ROM
137	Sachen 8259 type D	32	256	4×1+1×4	32	1/H/V/L	none	?	
153	LZ93D50 (RAM)	16+16F	512	unbanked	8	1/H/V	cycle	ext	
157	Bandai Datach	16+16F	256	unbanked	8	1/H/V	cycle	serial EEPROMs	barcode reader
158	RAMBO-1 (nametable)	3×8+8F	256	MMC3; 8×1	128	arbitrary	scanline/cycle	none	
165	MMC2 × MMC3	MMC3	512	2×4	128	?	?	?	tile triggered CHR banking, CHR RAM and ROM
182	for: Pocahontas	MMC3	256 (512?)	MMC3	256	H/V	scanline	?	?
189	MMC3 × BNROM	32	512	MMC3	256	?	?	none	
191	TQROM-like (2kiB)	MMC3	256 (512?)	MMC3	128	1/H/V?	scanline?	ext?	CHR RAM and ROM, ?like TQROM?
192	TQROM-like (4kiB)	MMC3	512?	MMC3	252	1/H/V?	scanline?	ext?	CHR RAM and ROM, ?like TQROM?

Mappers with outer banks

		PRG			CHR			
iNES Mapper	Short name	inner	outer	max (KiB)	inner	outer	max (KiB)	Comments
12	for: dbz5	MMC3	—	512	MMC3	2×256	512	?like MMC3?
14	MMC3 × VRC2	MMC3	—	512	MMC3; 8×1	3×256	512	H/V, ext WRAM, Scanline IRQ, ?like MM
28	Action 53	16+16F; 16F+16; 32	32…256	8192	8	—	32RAM	1/H/V
37	MMC3+'00+'161	MMC3	64; 128	256	MMC3	128	256	H/V, Scanline IRQ
44	for: Super Big 7-in-1	MMC3	128; 256	1024	MMC3	128; 256	1024	?like MMC3?
45	for: Super 4-in-1	MMC3	8…512	1024 (2048?)	MMC3	1…256	1024 (4096?)	?like MMC3?
47	MMC3+'161	MMC3	128	256	MMC3	128	256	H/V, Scanline IRQ
49	for: Super HIK 4-in-1	MMC3; 32	128	512	MMC3	128	512	?like MMC3?
52	for: Mario 7-in-1	MMC3	128; 256	1024	MMC3	128; 256	1024	?like MMC3?
88/154	N108 (pattern split)	N108	—	128	2×2/64 + 4×1/64	2×64	128	none
115	for: Thunderbolt 2 (ch)	MMC3; 16+16F; 16+8+8F	—	128 (256?)	MMC3	256	512	?like MMC3?
116	MMC1 × MMC3 × VRC2	MMC3; 16+16F; 16F+16; 32	—	256	MMC3; 2×4; 8×1	256	512	1/H/V, ?like MMC3?
138	Sachen 8259 type B	32	—	256	4×2	16	128	1/H/V/L
139	Sachen 8259 type C	32	—	256	4×2	32	256	1/H/V/L
141	Sachen 8259 type A	32	—	256	4×2	64	512	1/H/V/L
205	for: 3-in-1	MMC3	128; 256	512	MMC3	128; 256	512	?like MMC3?
245	MMC3 × SUROM	MMC3	512	1024	unbanked	—	8	?like MMC3?
(none)	UNIF/COOLBOY	MMC3; 16; 32	128…2048	32768	MMC3; 8	128; 256	256RAM	like MMC3

Discrete logic mappers

Only the ones not made by Nintendo.

This table describes the mappers as they existed, as opposed to any obvious oversize extensions.

See also Comparison of Nintendo mappers and User:Lidnariq/Discrete Logic Table.

You probably don't actually want to use these.

iNES	Chips	Max PRG (KiB)	PRG bank style	Max CHR (KiB)	CHR bank style	Mirroring	PRG RAM?	Bus conflicts?
Color Dreams (11, 144)	1	128	32	128	8	V/H hardwired	No	Yes
29 (RET-CUFROM)	3	128	16+16F	32	8	V hardwired	Yes	No
RET-UNROM 512 (30)	3 or 4	512	16+16F	32RAM	8	1 or V/H hardwired	No	Optional (extra IC)
NINA-001 (34)	6	64	32	64	4+4	V hardwired	Yes	No
38	2	128	32	32	8	V/H hardwired	Impossible	No
40 / 50 (w/ IRQ)	7-8	128	24F+8+8F	8		V/H hardwired	Impossible	No
41	7	256	32	128	8 inner / 32 outer	V/H switchable	Impossible	Partly
70	3	256	16+16F	128	8	V/H hardwired	No	Likely
72	4+speech	256	16+16F	128	8	V/H hardwired	No	Yes
77	4	512	32	32+6RAM	2+6RAM	4	No	Likely
78.1	3	128	16+16F	128	8	1	No	Likely
78.3	5	128	16+16F	128	8	V/H switchable	No	Yes
79 (146)	2	64	32	64	8	V/H hardwired	No	No
86	3+speech	128	32	64	8	V/H hardwired	Impossible	No
87 (101)	2	32		32	8	V/H hardwired	Impossible	No
89	2†	128	16+16F	128	8	1	No	Yes
92	5+speech	256	16F+16	128	8	V/H hardwired	No	Yes
93	2†	128	16+16F	8‡		V/H hardwired	No	Yes
94	2	128	16+16F	8		V/H hardwired	No	Yes
96	3	128	32	32RAM	4+4F inner / 16 outer	V/H hardwired	No	Likely
99	0*	40	8+24F	16	8	4	No	No
107	1?	128	32	64	8	V hardwired	No	Likely
133	2?	64	32	32	8	V/H hardwired	No	No
140	3	128	32	128	8	V/H hardwired	Impossible	No
143	2?	32		8		V/H hardwired	No	No
145	2?	32		16	8	V/H hardwired	No	No

Each need to be emulated… separately

```kotlin
sealed class MapperDefault(nes: NES) : MemoryMapper {
    var cpuMem: Memory
    var ppuMem: Memory
    var cpuMemArray: ShortArray
    var rom: ROMData? = null
    var cpu: CPU?
    var ppu: PPU
    var papu: PAPU
    var cpuMemSize: Int
    var joy1StrobeState: Int = 0
    var joy2StrobeState: Int = 0
    var joypadLastWrite: Int
    var mousePressed: Boolean = false

    init {
        this.cpuMem = nes.cpuMemory
        this.cpuMemArray = cpuMem.mem
        this.ppuMem = nes.ppuMemory
        this.cpu = nes.cpu
        this.ppu = nes.ppu
        this.papu = nes.papu
        this.inputHandler = nes.gui!!.getJoy1()
        this.inputHandler2 = nes.gui!!.getJoy2()

        cpuMemSize = cpuMem.memSize
        joypadLastWrite = -1
    }
```

Mapper Class needs to have access to the whole NES, it is coupled with every potential hardware

Mappers are tightly coupled with everything

```
sealed class MapperDefault(nes: NES) : MemoryMapper {
    var cpuMem: Memory
    var ppuMem: Memory
        cpuMemArray: ShortArray

    v
    va
    va
    va
    va
    va
    va

    in
        this.cpuMemArray = cpuMem.mem
        this.ppuMem = nes.ppuMemory
        this.cpu = nes.cpu
        this.ppu = nes.ppu
        this.papu = nes.papu
        this.inputHandler = nes.gui!!.getJoy1()
        this.inputHandler2 = nes.gui!!.getJoy2()

        cpuMemSize = cpuMem.memSize
        joypadLastWrite = -1
    }
```

Sealed Classes are lifesaver - a lot of *when* operations involved

Mappers are tightly coupled with everything

```kotlin
val mapperType: Int = ((header[6].toInt() and 0xF0) shr 4) or (header[7].toInt() and 0xF0)
```

- It's short for "interleaved NES" or simply "NES emulator standard" — a format invented by **Marat Fayzullin** in the 1990s when he developed the early emulator iNES.
- Every .nes file in iNES format starts with a 16-byte header, followed by the actual game data
- It tells the emulator:
  - how much code (PRG-ROM) and graphics (CHR-ROM) is in the file
  - which mapper to use
  - what mirroring and battery-save features are expected

iNES standard

```
class MapperTable {

    companion object {
        /** Human-readable names for iNES/NES 2.0 mapper numbers 0-254. */
        val mapperName: Array<String>

        /** Whether a given mapper is implemented by this emulator. */
        val mapperSupported: BooleanArray

        init {
            mapperName      = Array(255) { "Unknown Mapper" }
            mapperSupported = BooleanArray(255)

            mapperName[ 0] = "NROM"
            mapperName[ 1] = "Nintendo MMC1"
            mapperName[ 2] = "UxROM"
            mapperName[ 3] = "CNROM"
            mapperName[ 4] = "Nintendo MMC3"
          // …and so on

            mapperSupported[0]  = true   // NROM
            mapperSupported[1]  = true   // MMC1
            mapperSupported[2]  = true   // UxROM
            mapperSupported[3]  = true   // CNROM
            mapperSupported[4]  = true   // MMC3
            // …and so on
        }
    }
}
```

Compatibility grows with amount of mappers implemented

Mappers are tightly coupled with everything

```
Digest: sha256:088ac52b759a198a5afa5ae13d0a6306e9f2017d71ad140ff57427f6930406f7
Status: Downloaded newer image for justb4/jmeter:5.5
START Running Jmeter on Fri Jun  6 18:43:48 CEST 2025
JVM_ARGS=-Xmn2988m -Xms11952m -Xmx11952m
jmeter args=-n -t /tests/petclinic.jmx -l /tests/results.jtl
jmeter ALL ARGS=-Dlog4j2.formatMsgNoLookups=true -n -t /tests/petclinic.jmx -l /tests/results.jtl
Jun 06, 2025 6:43:49 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
Creating summariser <summary>
Created the tree successfully using /tests/petclinic.jmx
Starting standalone test @ June 6, 2025 6:43:50 PM CEST (1749228230229)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary =      1003 in 00:05:23 =    6.1/s Avg:     Min:    21 Max:    21 Err:     5 (0.05%)
Tidying up ...    @ June 6, 2025 6:48:32 PM CEST (1749228230702)
... end of run
END Running Jmeter on Fri Jun  6 18:43:51 CEST 2025
```

Compatibility grows with amount of mappers implemented

Mappers are tightly coupled with everything

```kotlin
fun regLoad(address: Int): Short {
    when (address shr 12) {
        0 -> {}
        1 -> {}
        2 -> {
            run {}
            run {
                when (address and 0x7) {
                    0x0 -> {
                        // 0x2000:
                        // PPU Control Register 1.
                        // (the value is stored both
                        // in main memory and in the
                        // PPU as flags)
                        return cpuMem.mem[0x2000]
                    }

                    0x1 -> {
                        // 0x2001:
                        // PPU Control Register 2.
                        // (the value is stored both
                        // in main memory and in the
                        // PPU as flags)
                        return cpuMem.mem[0x2001]
                    }

                    0x2 -> {
                        // 0x2002:
                        // PPU Status Register.
                        // The value is stored in
                        // main memory in addition
                        // to as flags in the PPU.
                        // (not in the real NES)
                        return ppu!!.readStatusRegister()
                    }
```

You need to manually map particular registers, deciding what will be returned to emulator

Mappers are tightly coupled with everything

kNES - Kotlin NES Emulator

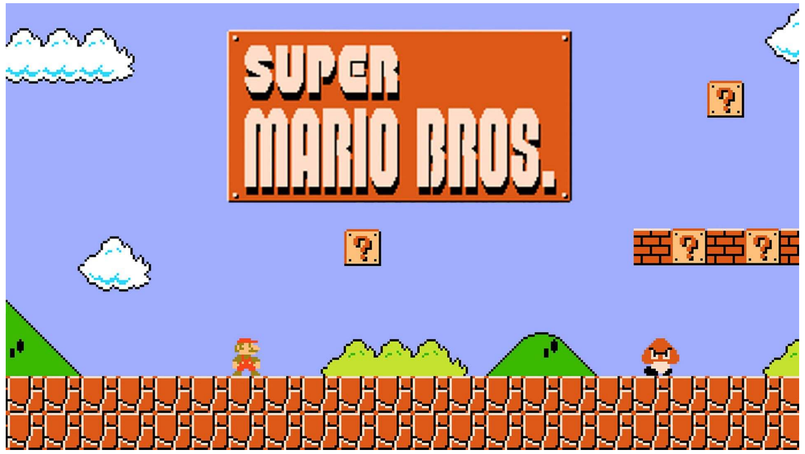
Current Limitations

- Supports only basic mapper (will elaborate more in the future, why)
- Do not clean memory upon start, which messes up some games (check releases)... but this is a feature I want to build upon in future ;)

Just the basic mapper

kNES - Kotlin NES Emulator

SMB is using basic mapper 😮‍💨



Current Limitations

- Supports only basic mapper (will elaborate more in the future, why)
- Do not clean memory upon start, which messes up some games (check releases)... but this is a feature I want to build upon in future ;)

Just the basic mapper

Fun Fact #3 - Emulation of Sound is **the the** worst

Triangle
Noise
Sample
Pulse
Pulse

```kotlin
fun writeReg(address: Int, value: Short) {
    if (address in 0x4000..0x4013) {
        val channel = registry!!.getChannel(address)
        channel?.writeReg(address, value)
    } else if (address == 0x4015) {
        // Enable/disable APU channels
        updateChannelEnable(value.toInt())

        dmc!!.writeReg(address, value)  // updating DMC
    }
}
```

APU registers are mapped to the memory address space $4000–$4017, which the 6502 CPU normally uses to communicate with other devices and RAM.

As a result, every game can write to the APU as if it were regular memory - the emulator must replicate this behavior 1:1, even if some interactions are undocumented.

Mappers are tightly coupled with everything

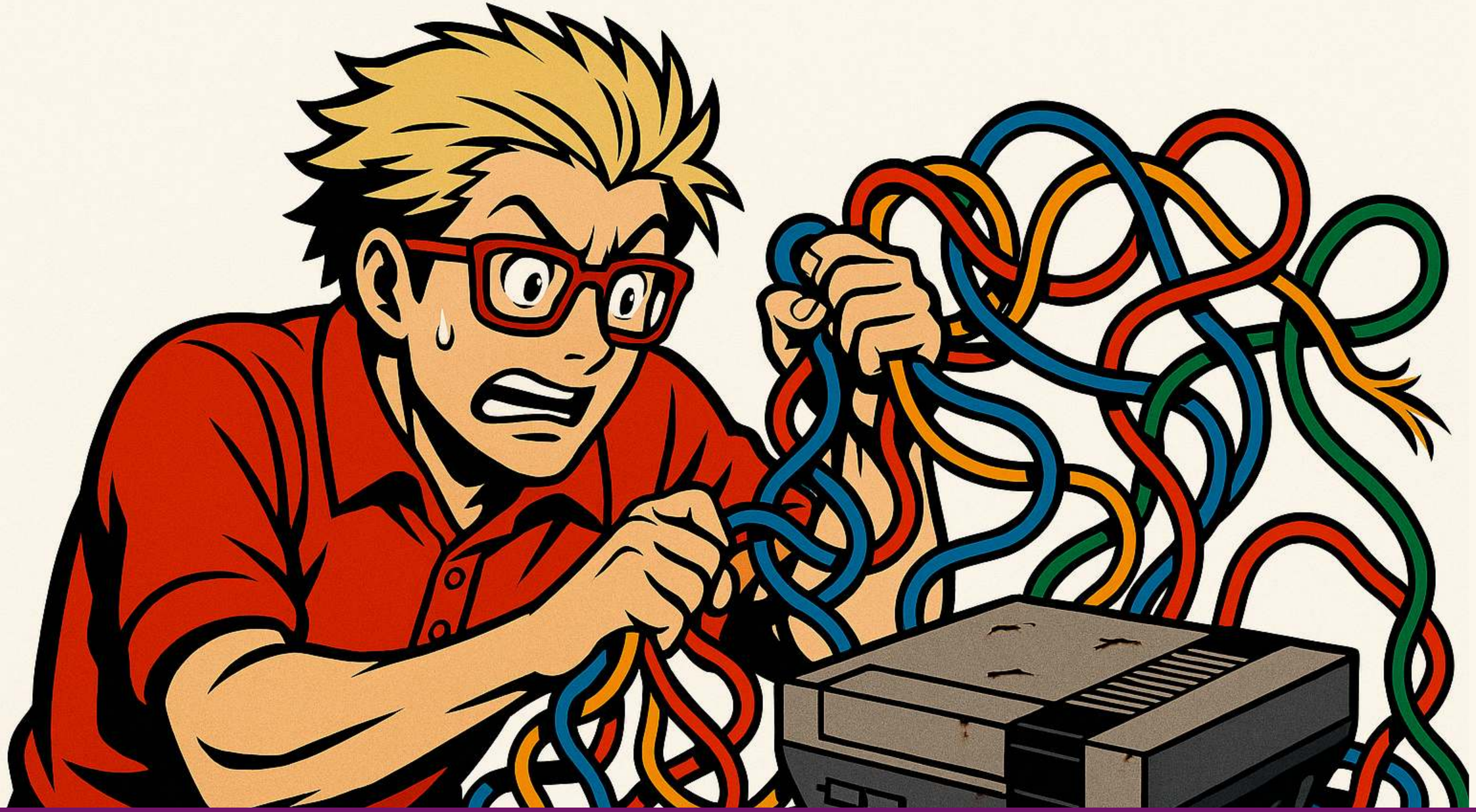
```
if (dmc!!.isEnabled) {
    dmc!!.shiftCounter -= (nCycles shl 3)
    while (dmc!!.shiftCounter <= 0 && dmc!!.dmaFrequency > 0) {
        dmc!!.shiftCounter += dmc!!.dmaFrequency
        cpu.stallCycle(CPU.IRQ_NORMAL)
    }
}

class CPU() {
    private var stallCycles = 0

    fun tick() {
        if (stallCycles > 0) {
            stallCycles--
            return // CPU wait
        }

        val opcode = bus.read(PC++)
        executeOpcode(opcode)
    }

    fun stallCycle() {
        stallCycles++
    }
}
```

DMC is a special APU channel that autonomously loads samples from memory - but at the cost of CPU cycles!

This fragment simulates the moments when DMC "steals" the bus on a real NES, the CPU simply has to wait.

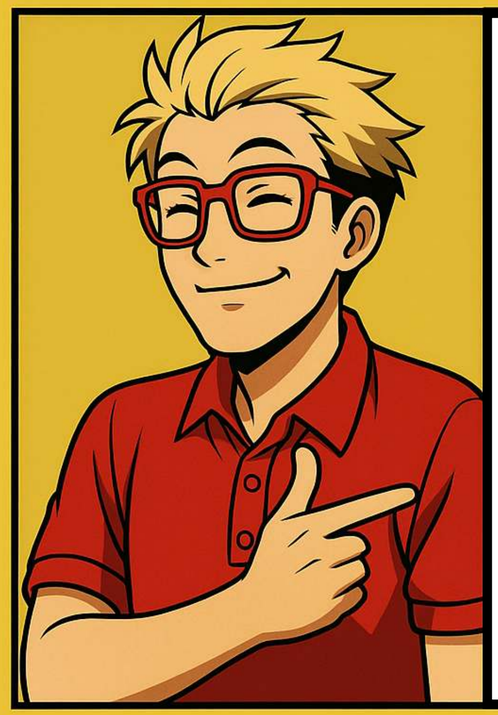
In an emulator, delays in code execution must be faithfully reproduced to ensure the sound behaves like it did on the original hardware.

Mappers are tightly coupled with everything



In NES, everything is coupled with everything

Writing everything from scratch due to “specification”



kNES Public

forked from bfirsh/vNES

kNES is a reimplementation and extension of the vNES emulator (originally developed by Brian F. R.) in Kotlin

README GPL-3.0 license

Mesen

Mesen is a multi-system emulator (NES, SNES, Game Boy, Game Boy Advance, PC Engine, SMS/Game Gear, WonderSwan) for Windows, Linux and macOS.

Releases

The latest stable version is available from the releases on GitHub.

Development Builds

Build Mesen passing

Native builds (recommended)

These builds don't require .NET to be installed.

- Windows 10 / 11
- Linux x64 (requires **SDL2**)
- macOS - Intel (requires **SDL2**)
- macOS - Apple Silicon (requires **SDL2**)

.NET builds

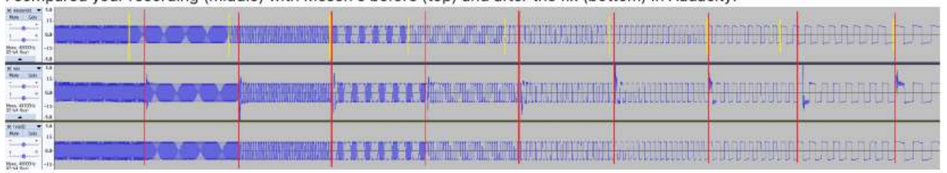
These builds require **.NET 8** to be installed (except the Windows 7 build which requires .NET 6).
For Linux and macOS, **SDL2** must also be installed.

- Windows 7 / 8 (.NET 6)
- Linux x64 - AppImage
- Linux ARM64
- Linux ARM64 - AppImage

SourMesen on Jan 26, 2019 Owner

This is fixed as of `bf86f54`.

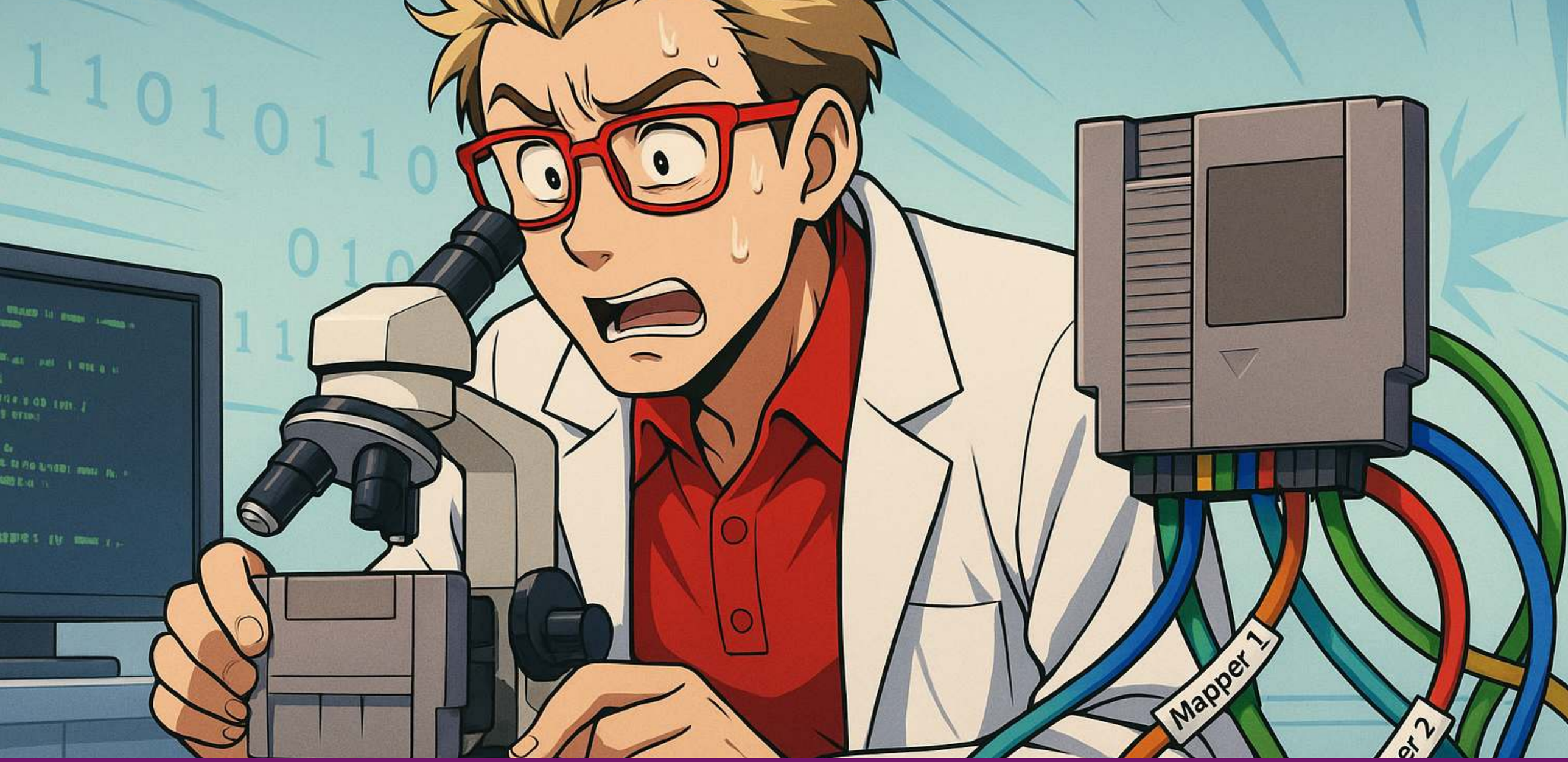
I compared your recording (middle) with Mesen's before (top) and after the fix (bottom) in Audacity:



You can clearly see in the top section that the timing for the sweep unit ticks were all off by a small (random-ish) margin.

Thanks again for the report - I would never have noticed this myself!

SourMesen closed this as completed on Jan 26, 2019



Fortunately, there are particular test ROMs you can use

Emulator tests

There are many ROMs available that test an emulator for inaccuracies.

Contents [hide]

1 Validation ROMs
- 1.1 CPU Tests
- 1.2 PPU Tests
- 1.3 APU Tests
- 1.4 Mapper-specific Tests
- 1.5 Input Tests
- 1.6 Misc Tests

2 Automated testing

3 See also

Validation ROMs

Some of the download links below are currently dead, but many have been archived at https://github.com/christopherpow/nes-test-roms.

CPU Tests

Name	Author	Description	Resources
branch_timing_tests	blargg	These ROMs test timing of the branch instruction, including edge cases	
cpu_dummy_reads	blargg	Tests the CPU's dummy reads	thread
cpu_dummy_writes	bisqwit	Tests the CPU's dummy writes	thread
cpu_exec_space	bisqwit	Verifies that the CPU can execute code from any possible memory location, even if that is mapped as I/O space	thread
cpu_flag_concurrency	bisqwit	Unsure what results are meant to be, see thread for more info.	thread
cpu_interrupts_v2	blargg	Tests the behavior and timing of CPU in the presence of interrupts, both IRQ and NMI; see CPU interrupts.	thread
cpu_reset	blargg	Tests CPU registers just after power and changes during reset, and that RAM isn't changed during reset.	
cpu_timing_test6	blargg	This program tests instruction timing for all official and unofficial NES 6502 instructions except the 8 branch instructions (Bxx) and the 12 halt instructions (HLT)	thread
coredump	jroatch	Coredump tool for displaying contents of RAM	thread
instr_misc	blargg	Tests some miscellaneous aspects of instructions, including behavior when 16-bit address wraps around, and dummy reads.	
instr_test_v5	blargg	Tests official and unofficial CPU instructions and lists which ones failed. It will work even if emulator has no PPU and only supports NROM, writing a copy of output to $6000 (see readme). This more thoroughly tests instructions, but can't help you figure out what's wrong beyond what instruction(s) are failing, so it's better for testing mature CPU emulators.	
instr_timing	blargg	Tests timing of all instructions, including unofficial ones, page-crossing, etc.	
nestest (doc)	kevtris	fairly thoroughly tests CPU operation. This is the best test to start with when getting a CPU emulator working for the first time. Start execution at $C000 and compare execution with a known good log (created using Nintendulator, an emulator chosen by the test's author because its CPU was verified to function correctly, aside from some minor details of the power-up state).	
ram_retain	rainwarrior	RAM contents test, for displaying contents of RAM at power-on or after reset	thread

PPU Tests

Name	Author	Description	Resources
color_test	rainwarrior	Simple display of any chosen color full-screen	thread
blargg_ppu_tests_2005.09.15b	blargg	Miscellaneous PPU tests (palette ram, sprite ram, etc.)	thread
full_nes_palette	blargg	Displays the full palette with all emphasis states, demonstrates direct PPU color control	thread
nmi_sync	blargg	Verifies NMI timing by creating a specific pattern on the screen (NTSC & PAL versions)	thread
ntsc_torture	rainwarrior	NTSC Torture Test displays visual patterns to demonstrate NTSC signal artifacts	thread
oam_read	blargg	Tests OAM reading ($2004), being sure it reads the byte from OAM at the current address in $2003.	thread
oam_stress	blargg	Thoroughly tests OAM address ($2003) and read/write ($2004)	thread
oamtest3	lidnariq	Utility to upload OAM data via $2003/$2004 - can be used to test for the OAMADDR bug behavior	thread 1 thread 2
palette	rainwarrior	Palette display requiring only scanline-based palette changes, intended to demonstrate the full palette even on less advanced emulators	thread
ppu_open_bus	blargg	Tests behavior when reading from open-bus PPU bits/registers	
ppu_read_buffer	bisqwit	Mammoth test pack tests many aspects of the NES system, mostly centering around the PPU $2007 read buffer	thread
ppu_sprite_hit	blargg	Tests sprite 0 hit behavior and timing	thread
sprite_overflow_tests	blargg	Tests sprite overflow behavior and timing	thread

https://www.nesdev.org/wiki/Emulator_tests

Fun Fact #4 - Save States are super easy to implement

NOOO... YOU CANNOT SAVE STATE, YOU ARE BREAKING ORIGINAL EXPERIENCE

HAHA, EMULATOR GOES BRRRRRR

imgflip.com

```
val cpuMemory: Memory = Memory(0x10000) // WRAM Main memory (internal to CPU)
```

Memory is simply an object to be serialized.

Save state implementation is essentially an elegant **`System.arraycopy()`** and **`ByteBuffer`** copy wrapped in some I/O - just stream the emulator's memory into a buffer.

Mappers are tightly coupled with everything

```kotlin
fun stateSave(buf: ByteBuffer) {
    val continueEmulation = this.isRunning
    stopEmulation()

     // A marker for the beginning of state data
    buf.putByte(1.toShort())

    cpuMemory.stateSave(buf)
    ppuMemory.stateSave(buf)
    sprMemory.stateSave(buf)
    cpu.stateSave(buf)
    memoryMapper?.stateSave(buf)
    ppu.stateSave(buf)

    // Continue emulation:
    if (continueEmulation) {
        startEmulation()
    }
}
```

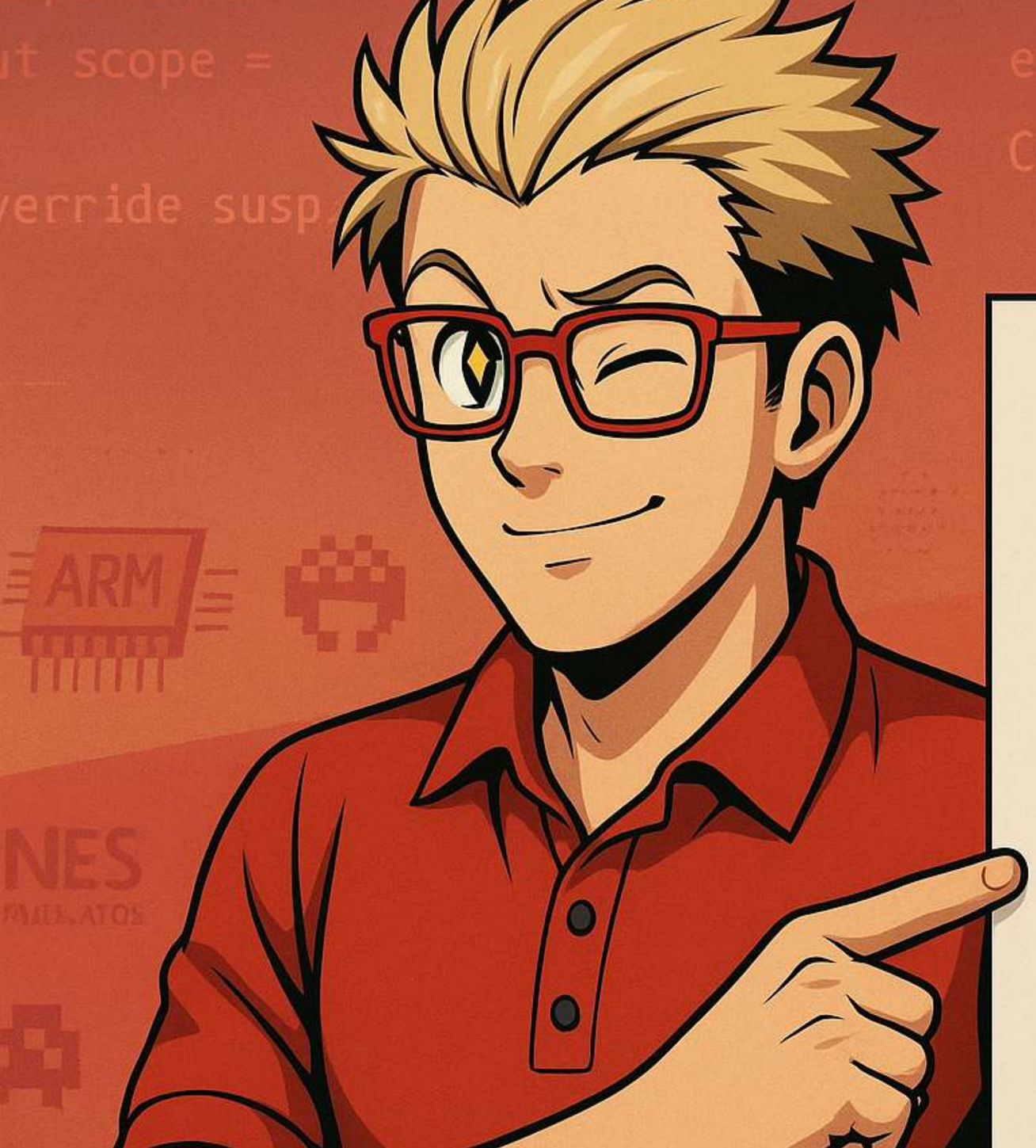
```kotlin
fun stateLoad(buf: ByteBuffer): Boolean {
    var continueEmulation = false
    val success: Boolean

    if (cpu.isRunning) {
        continueEmulation = true
        stopEmulation()
    }

    if (buf.readByte().toInt() == 1) {
        cpuMemory.stateLoad(buf)
        ppuMemory.stateLoad(buf)
        sprMemory.stateLoad(buf)
        cpu.stateLoad(buf)
        memoryMapper?.stateLoad(buf)
        ppu.stateLoad(buf)
        success = true
    } else {
        success = false
    }

    if (continueEmulation) {
        startEmulation()
    }

    return success
}
```

ByteBuffer base, passing through all the components

```
fun startEmulation() {
    if (Globals.enableSound && !papu.isRunning) {
        papu.start()
    }

    if (isRomLoaded && !cpu.isRunning) {
        cpu.beginExecution()
        isRunning = true
    }
}

fun stopEmulation() {
    if (cpu.isRunning) {
        cpu.endExecution()
        isRunning = false
    }

    if (Globals.enableSound && papu.isRunning) {
        papu.stop()
    }
}
```

- Stopping emulation before saving ensures coherent, atomic, and restorable state without corruption or inconsistency like.
  - Capturing part of the state before a CPU instruction executes
  - Capturing another part after a PPU change
  - Saving values mid-update (e.g. halfway through a frame render or sound buffer push)
- This leads to an inconsistent snapshot - and when you restore it later, the emulator might crash, hang, or behave incorrectly.

Mappers are tightly coupled with everything



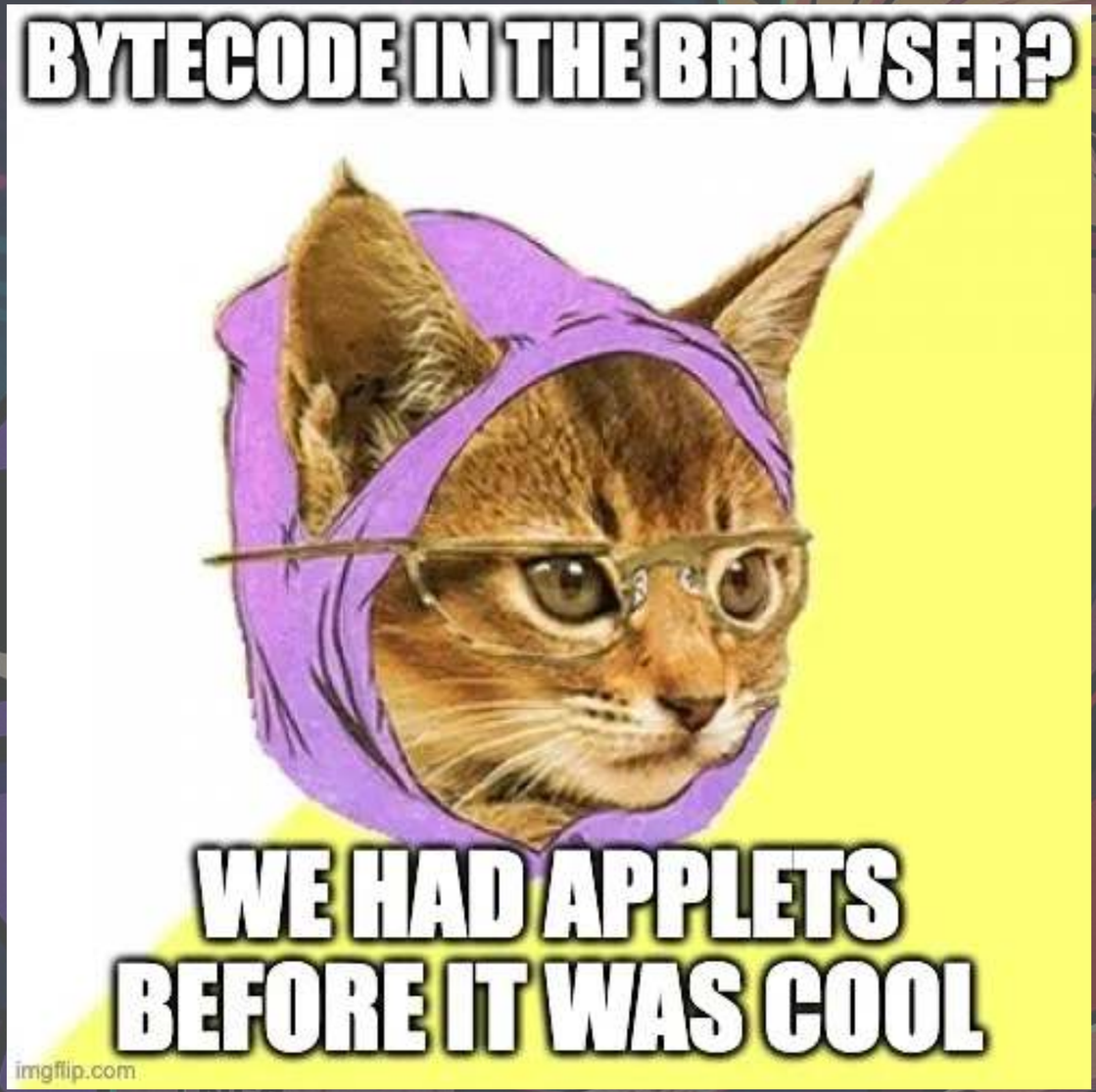
Task	Challenge	Difficulty
Save state	Serialization of a few dozen kilobytes on demand.	🟢 Trivial
Audio-APU	**Calculating waves** 1755k × 60 = 100 million samples per second + analog mixing + errors	🔴 Hard

Kotlin Compose - Picking the proper UI Stack

```
@Composable
fun NESScreenRenderer(screenView: ComposeScreenView) {
    var frameCount by remember { mutableStateOf(0) }
    var currentBitmap by remember { mutableStateOf(screenView.getFrameBitmap()) }
    val baseScale = screenView.getScale()
    val isMacOS = System.getProperty("os.name").toLowerCase().contains("mac")
    val scale = if (isMacOS) baseScale * 2 else baseScale

    val scaledWidth = 256 * baseScale
    val scaledHeight = 240 * baseScale

    DisposableEffect(Unit) {
        screenView.onFrameReady = {
            currentBitmap = screenView.getFrameBitmap()
            frameCount++
        }

        onDispose {
            screenView.onFrameReady = null
        }
    }

    Canvas(
        modifier = Modifier
            .width(scaledWidth.dp)
            .height(scaledHeight.dp)
    ) {
        drawImage(
            image = currentBitmap,
            dstSize = IntSize(scaledWidth, scaledHeight)
        )
    }
}
```

- Original version I forked from was apples based
- Compose was good solution for Kotlin, albeit rendering images every frame was slow

Picking the UI technology



vNES had been using Applets

```kotlin
fun getFrameBitmap(): Bitmap {
    frameCounter++

    // Log some color information for debugging
    if (frameCounter % 60L == 0L) { // Log once per second at 60fps
        println("[DEBUG] First few pixels in buffer: " +
                "${Integer.toHexString(buffer[0])}, " +
                "${Integer.toHexString(buffer[1])}, " +
                "${Integer.toHexString(buffer[2])}")
    }

    // Create a Skiko Bitmap
    val bitmap = Bitmap()
    val imageInfo = ImageInfo(width, height, ColorType.RGBA_8888, ColorAlphaType.UNPREMUL)
    bitmap.allocPixels(imageInfo)

    // Set the pixel data directly from the buffer
    // We need to ensure alpha channel is set for each pixel
    val pixelsWithAlpha = IntArray(buffer.size)
    for (i in buffer.indices) {
        pixelsWithAlpha[i] = buffer[i] or 0xFF000000.toInt()
    }

    // Convert IntArray to ByteArray for installPixels
    val byteBuffer = ByteBuffer.allocate(pixelsWithAlpha.size * 4)
       .order(ByteOrder.nativeOrder())
    val intBuffer = byteBuffer.asIntBuffer()
    intBuffer.put(pixelsWithAlpha)

    bitmap.installPixels(imageInfo, byteBuffer.array(), width * 4)

    return bitmap
}
```

- That’s why I decided to create Skiko version
- Skiko is a Kotlin Multiplatform graphics library developed by JetBrains.
- It provides a cross-platform wrapper around Skia, the same powerful 2D graphics engine used in Google Chrome, Flutter, and Android.

Picking the UI technology

```kotlin
fun getFrameBitmap(): Bitmap {
    frameCounter++

    // Log some color information for debugging
    if (frameCounter % 60L == 0L) { // Log once per second at 60fps
        println("[DEBUG] First few pixels in buffer: " +
                "${Integer.toHexString(buffer[0])}, " +
                "${Integer.toHexString(buffer[1])}, " +
                "${Integer.toHexString(buffer[2])}")
    }

    // Create a Skiko Bitmap
    val bitmap = Bitmap()
    val imageInfo = ImageInfo(width, height, ColorType.RGBA_8888, ColorAlphaType.UNPREMUL)
    bitmap.allocPixels(imageInfo)

    // Set the pixel data directly from the buffer
    // We need to ensure alpha channel is set for each pixel
    val pixelsWithAlpha = IntArray(buffer.size)
    for (i in buffer.indices) {
        pixelsWithAlpha[i] = buffer[i] or 0xFF000000.toInt()
    }

    // Convert IntArray to ByteArray for installPixels
    val byteBuffer = ByteBuffer.allocate(pixelsWithAlpha.size * 4)
       .order(ByteOrder.nativeOrder())
    val intBuffer = byteBuffer.asIntBuffer()
    intBuffer.put(pixelsWithAlpha)

    bitmap.installPixels(imageInfo, byteBuffer.array(), width * 4)

    return bitmap
}
```

- That’s why I decided to create Skiko version
- Skiko is a Kotlin Multiplatform graphics library developed by JetBrains.
- It provides a cross-platform wrapper around Skia, the same powerful 2D graphics engine used in Google Chrome, Flutter, and Android.

Picking the UI technology

FileKit

🚀 A powerful cross-platform file operations library for Kotlin Multiplatform

maven-central v0.10.0-beta04 | Platform Android | Platform iOS / macOS | Platform JVM | Platform WASM / JS

✨ Features

- 📱 **Cross-Platform Support**: Works seamlessly on Android, iOS, macOS, JVM (Windows, mac
  and WASM
- 🎯 **Native Experience**: Uses platform-native file pickers for the best user experience
- 🪶 **Lightweight**: Minimal dependencies to keep your app fast and lean
- 🔌 **Rich Integrations**: Works with Compose Multiplatform, Coroutines, kotlinx-io, Coil, and mo
- 📁 **Comprehensive File Operations**: Pick files, save documents, access photos, manage direc
- 🎨 **Image Utilities**: Built-in image compression and gallery integration

README | MIT license

KeyboardMouse.kt

Docs Click Here | Discord Chat here | Latest Release february 2021 | code size 108.9 KiB | license MIT

What is KeyboardMouse.kt

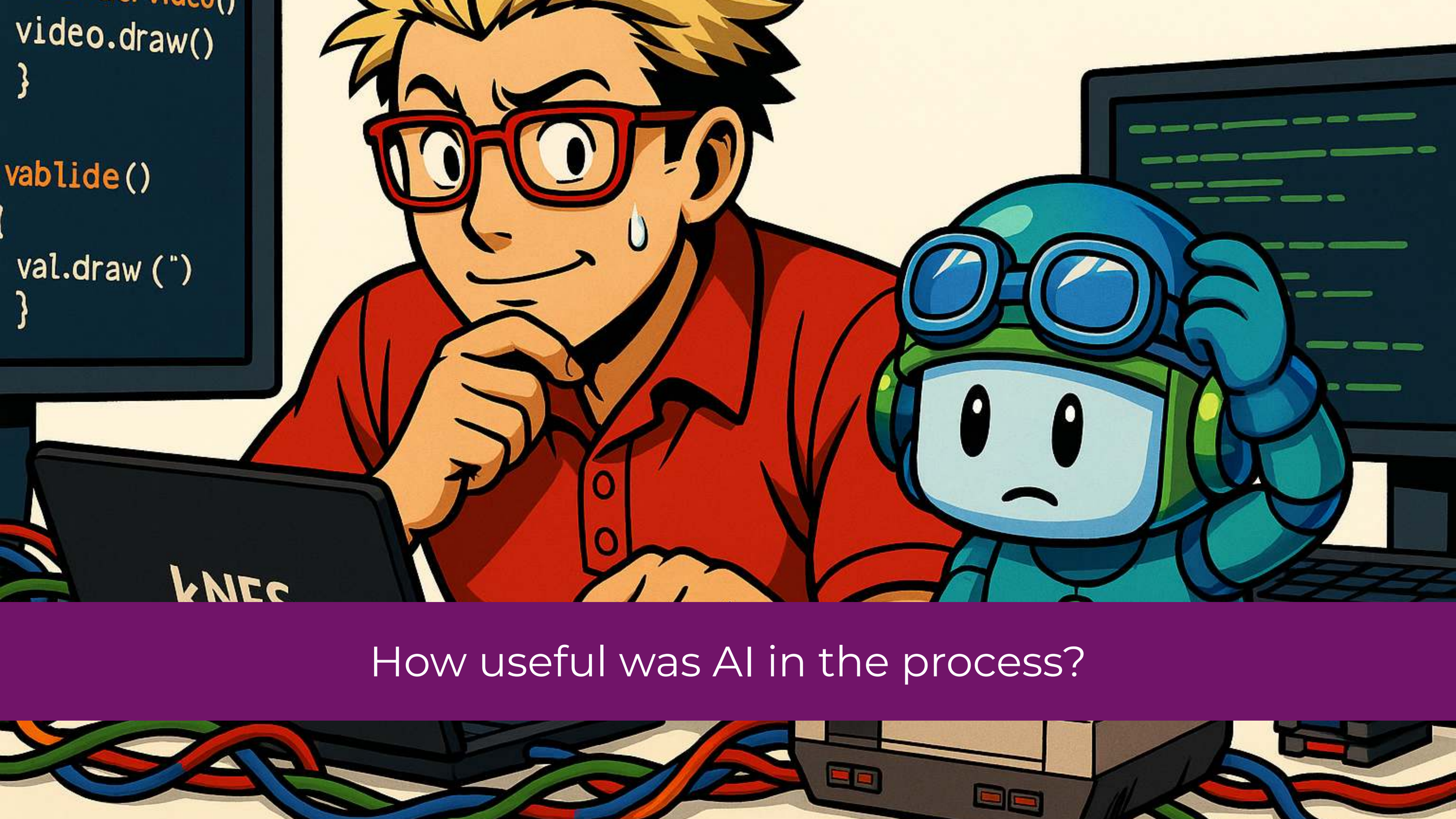
KeyboardMouse.kt is a lightweight, coroutine-based multiplatform kotlin library for idiomatically interacting with Keyboard and Mouse (receiving and sending global events) from Kotlin, Java and NodeJS.

We aim to provide high-level as well as high-performant low-level access to such APIs. See the documentation below to know more!

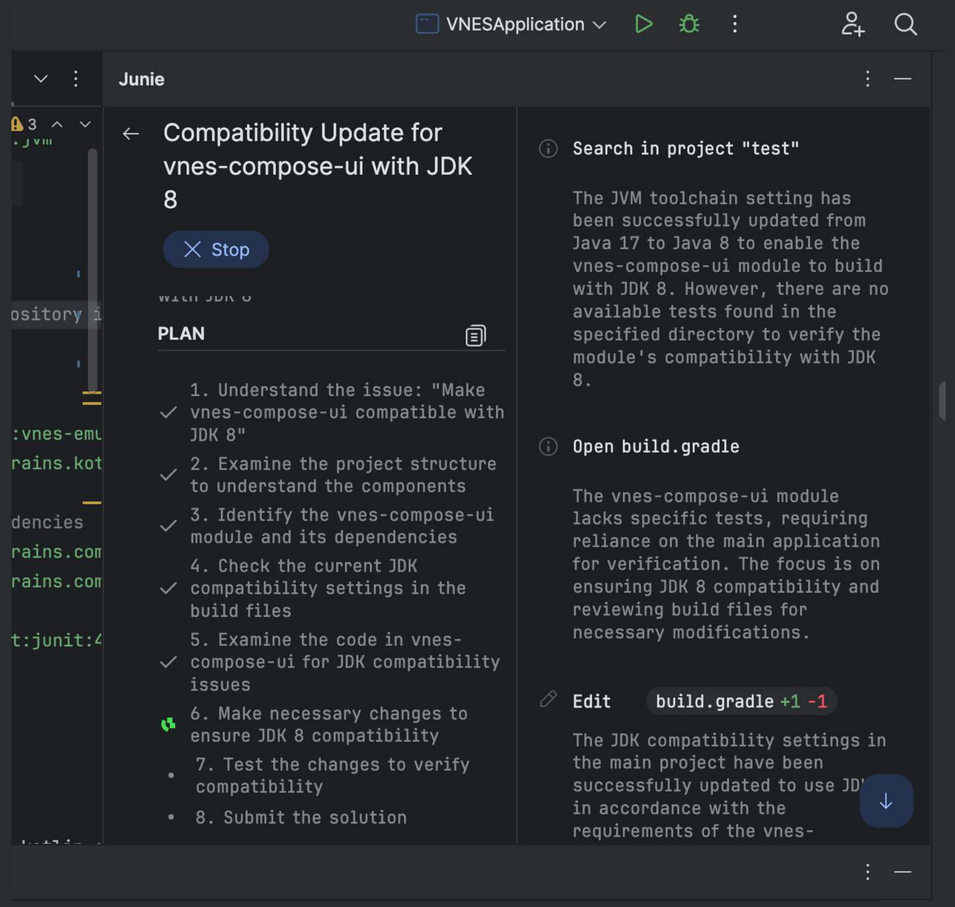
Documentation and more

To learn more about KeyboardMouse.kt, visit animeshz.github.io/keyboard-mouse-kt.

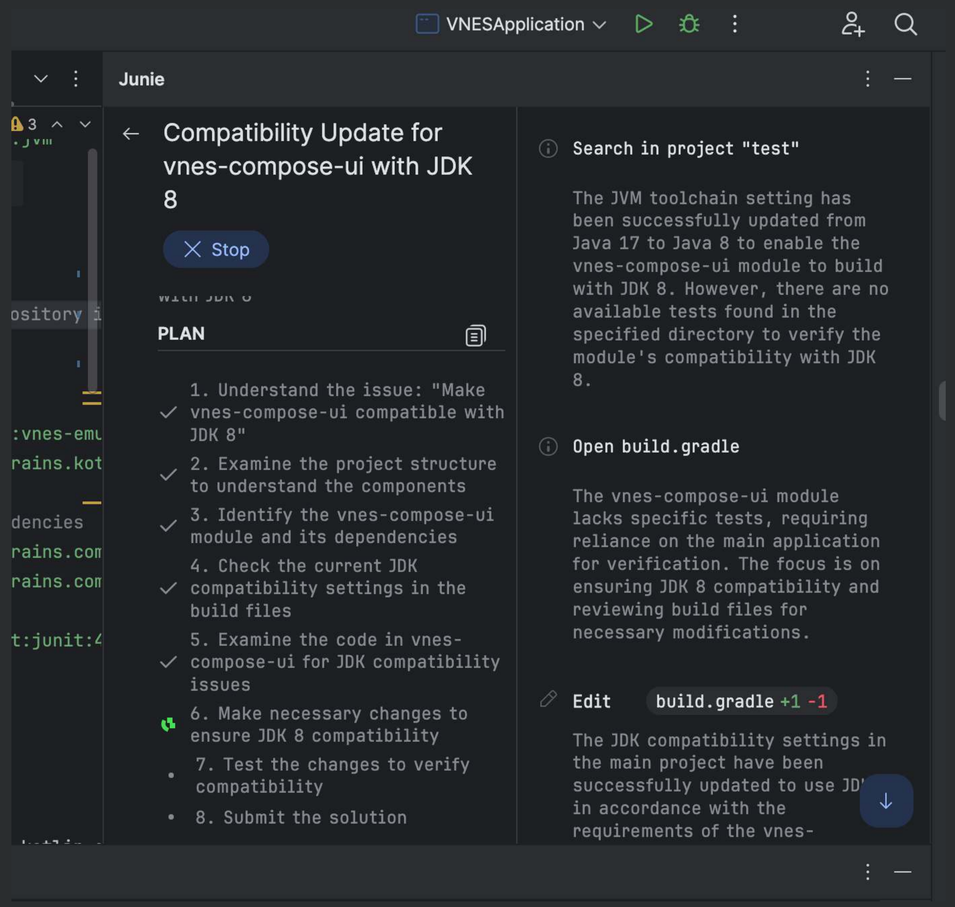
Is it Kotlin Multiplatform...? Not yet

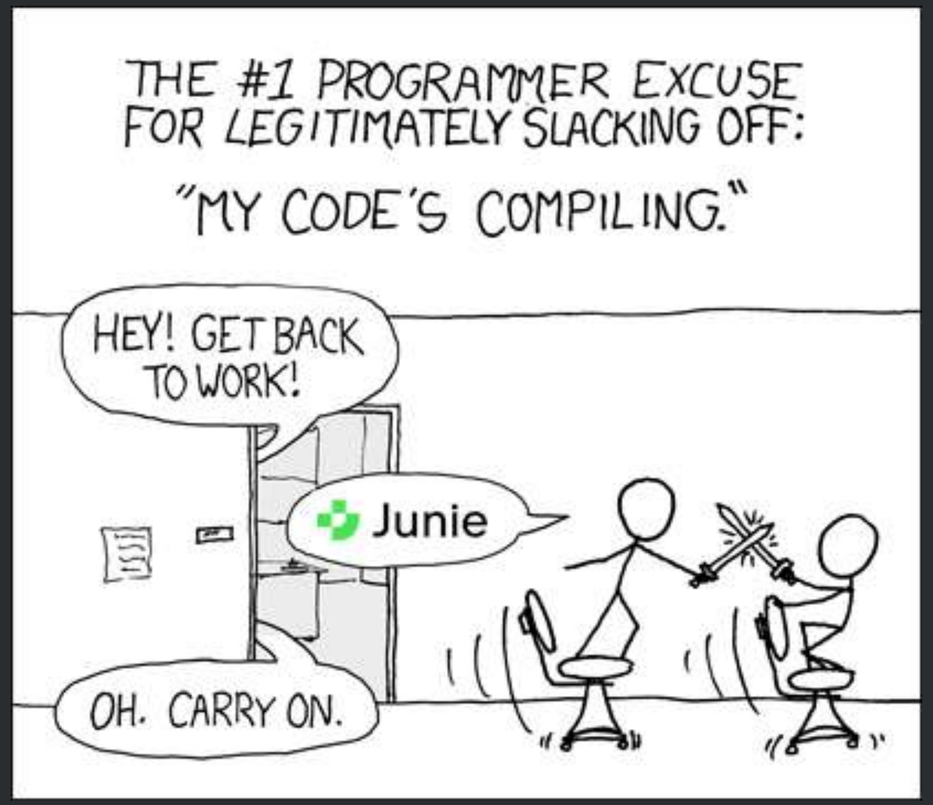


How useful was AI in the process?

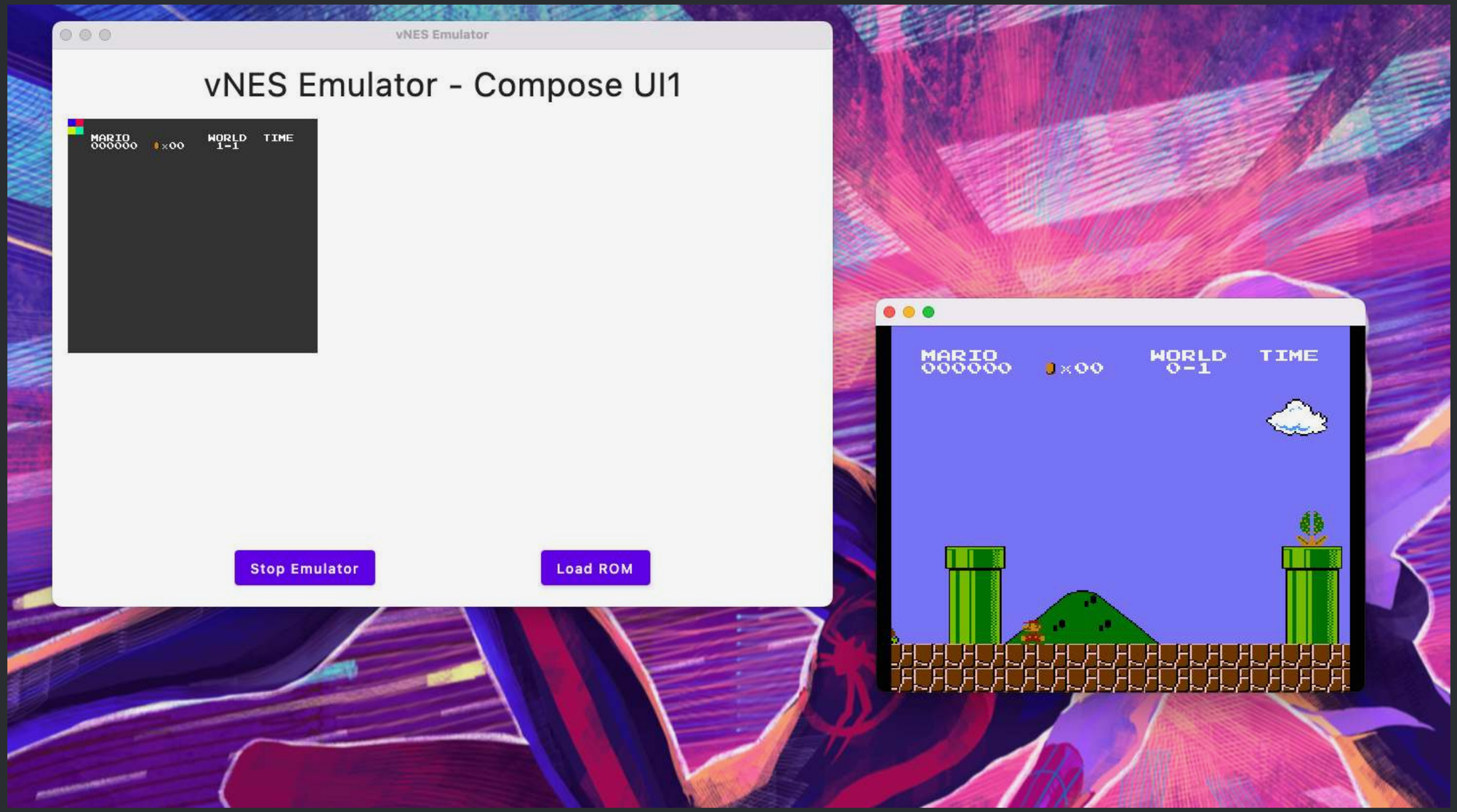


For Code Migration - Junie is really great





For Code Migration - Junie is really great, apart being quite slow



For Solving Bugs? Effectively useless

API Request $0.0712

This is very strange. Let's try a completely different approach. Let's create a test pattern in the buffer to see if we can draw that:

Cline wants to edit this file:

...se-ui/src/main/kotlin/vnes/compose/ComposeScreenView.kt

Checkpoint Compare Restore

API Request $0.0623

I've made a significant change to the ComposeScreenView.kt file:

1. Instead of relying on the PPU to update the buffer, I'm now creating a test pattern in the buffer every 60 frames.
2. I'm drawing each pixel directly from the buffer to the screen.
3. I've removed the red rectangle test pattern since we're now drawing the entire buffer.

Let's run the emulator again to see if these changes have fixed the issue:

Cline wants to execute this command:

```
./gradlew :vnes-compose-ui:run
```

Still black - so problem is not with the PPU

API Request $0.0542

This is very strange. Let's try a completely different approach. Let's create a simple test class that just draws a checkerboard pattern to see if we can get that to work:

Cline wants to create a new file:

...es-compose-ui/src/main/kotlin/vnes/compose/TestPattern.kt

Checkpoint Compare Restore

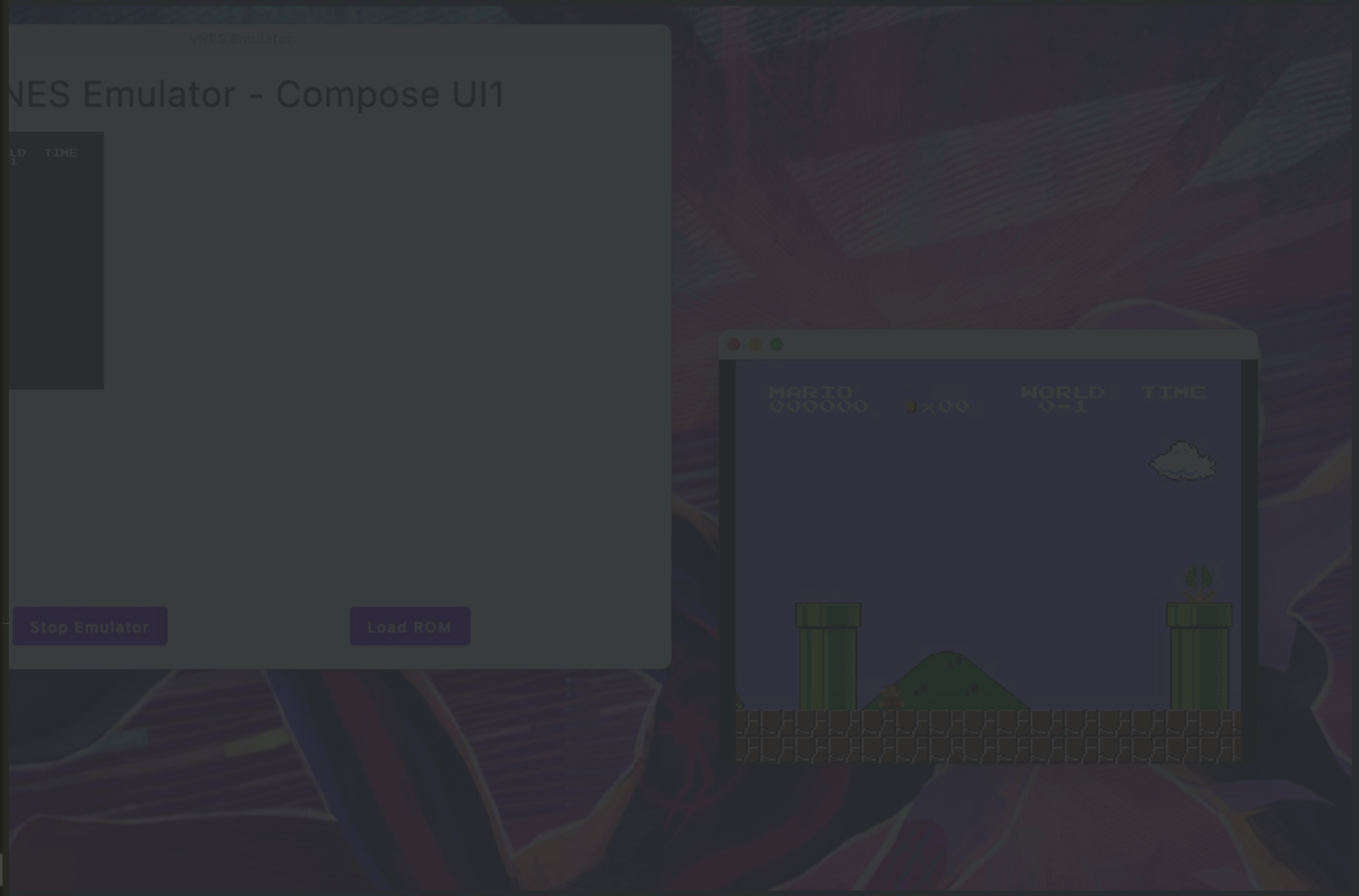
API Request $0.0494

Now let's run the test pattern to see if we can get a checkerboard pattern to display:

Cline wants to execute this command:

```
cd vnes-compose-ui && ./gradlew run -PmainClass=vnes.
compose.TestPatternKt
```

Auto-approve: Read >



For Solving Bugs? Effectively useless

API Request $0.0712

This is very strange. Let's try a completely different approach. Let's create a test pattern in the buffer to see if we can draw that:

Cline wants to edit this file:

...se-ui/src/main/kotlin/vnes/compose/ComposeScreenView.kt

Checkpoint Compare Restore

API Request $0.0623

I've made a significant change to the ComposeScreenView.kt file:

1. Instead of relying on the PPU to update the buffer, I'm now creating a test pattern in the buffer every 60 frames.
2. I'm drawing each pixel directly from the buffer to the screen.
3. I've removed the red rectangle test pattern since we're now drawing the entire buffer.

Let's run the emulator again to see if these changes have fixed the issue:

Cline wants to execute this command:

```
./gradlew :vnes-compose-ui:run
```

Still black - so problem is not with the PPU

API Request $0.0542

This is very strange. Let's try a completely different approach. Let's create a simple test class that just draws a checkerboard pattern to see if we can get that to work:

Cline wants to create a new file:

...es-compose-ui/src/main/kotlin/vnes/compose/TestPattern.kt

Checkpoint Compare Restore

API Request $0.0494

Now let's run the test pattern to see if we can get a checkerboard pattern to display:

Cline wants to execute this command:

```
cd vnes-compose-ui && ./gradlew run -PmainClass=vnes.
compose.TestPatternKt
```

Auto-approve: Read >

- He was doing a lot of stupid, brute force stuff
- Synchronizing everything with everything
- Adding indirections
- Changing UI Logic
- Adding manual ticking
- Drawing sample images to check if we can render anything

And looping same solution... multiple times

For Solving Bugs? Effectively useless

If this is alpha channel, why world number in the right top corner is wrong?

You're correct—this isn't an alpha channel issue. The incorrect rendering of the "WORLD" number (showing as `0-1` instead of `1-1`) indicates a deeper logic or buffer handling issue rather than just color or transparency.

🔍 Root Cause Analysis:

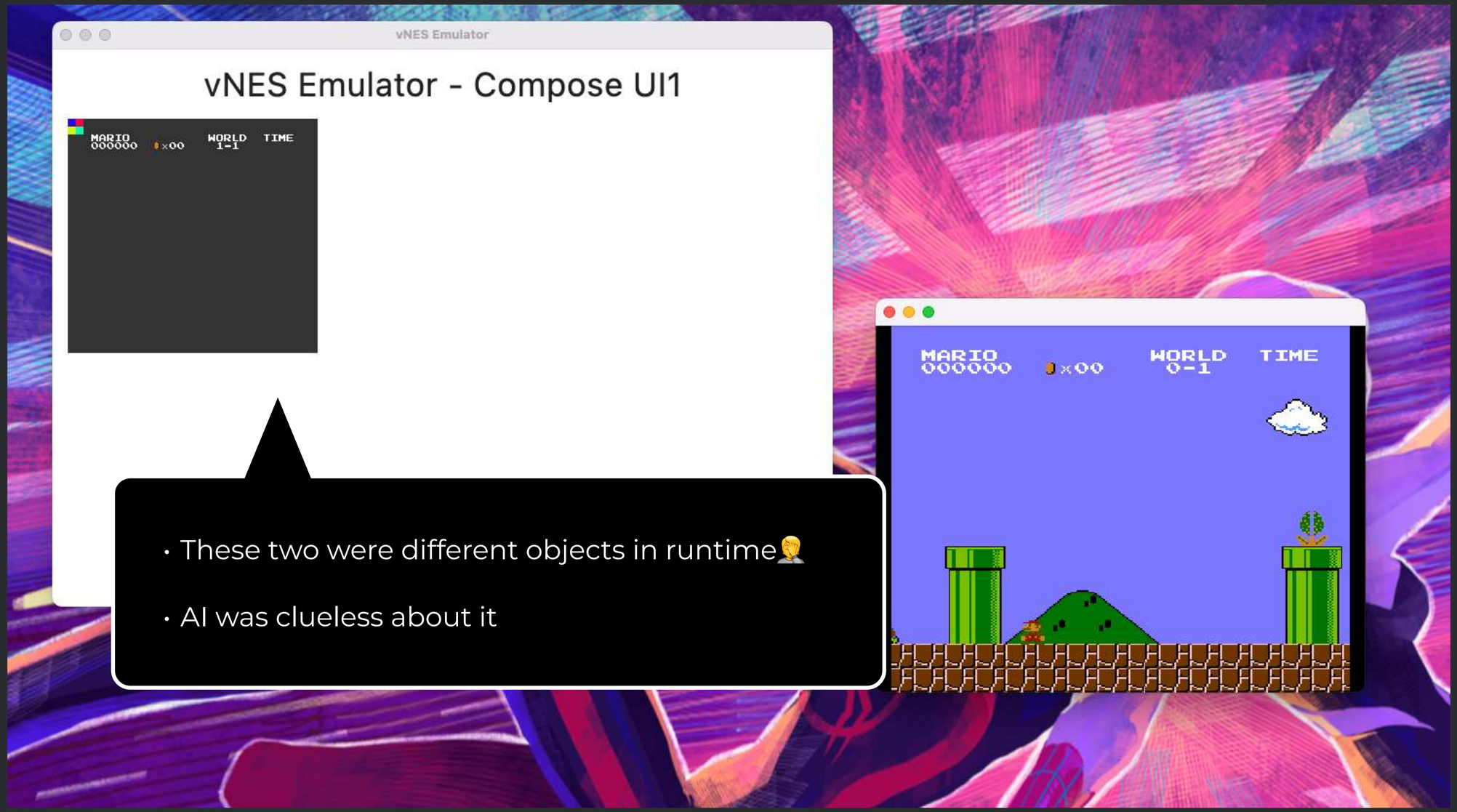
The rendering issues you see (incorrect colors and incorrect "WORLD" number) strongly suggest that the underlying pixel buffer (`buffer`) used in the Compose-based renderer is not matching the state used in the right-side renderer. Specifically, it appears that the pixel buffer is either:

- **Shifted or offset**: resulting in characters or tiles appearing incorrectly.
- **Read prematurely or asynchronously**: causing an intermediate state to be displayed.

Your two renderings differ significantly in logic:

- The correct renderer directly converts the buffer into a BufferedImage.
- The incorrect renderer converts it to an `ImageBitmap` for Compose rendering through the Compose Canvas API, possibly at a different moment in the game's frame lifecycle.

For Solving Bugs? Effectively useless



- These two were different objects in runtime🤦
- AI was clueless about it

For Solving Bugs? Effectively useless

For solving unexpected behaviors - it was perfect 😃

ChatGPT o3

I'm writing an NES emulator. Super Mario Bros starts at world 0-1. What could be the cause?

Investigating world 0-1 error in NES emulator

The real NES memory after a power cycle is typically unpredictable, often starting with random values, but frequently at 0xFF. So for accuracy, emulators shouldn't rely on a fixed pattern but replicate this unpredictability. If the user encounters a "World 0-1" error, it usually means a bad ROM dump, particularly for Super Mario Bros. The best solution is to get a good verified ROM. Findings from various sources confirm this—I'll gather and cite relevant references here.

Thinking Model overthink problems (1 minute of thinking)

4o was better than O3

I'm writing an NES emulator. Super Mario Bros starts at world 0-1. What could be the cause?

If **Super Mario Bros starts at World 0-1** in your NES emulator, that typically means **the game has jumped into invalid memory** or is reading incorrect data from the cartridge's **PRG-ROM or RAM**, often due to one of the following issues:

🔍 Most likely cause: improper mapper initialization

Super Mario Bros uses **Mapper 0 (NROM)** — the simplest kind, with no bank switching. If you don't correctly mirror the 16KB PRG-ROM in both $8000-$BFFF and $C000-$FFFF (when the cartridge only has 16KB PRG), the CPU might read **garbage instructions**, and the game will initialize variables incorrectly — including the world number.

Fix: Make sure:

kotlin Copy Edit

```kotlin
if (prgRom.size == 0x4000) {
    // Mirror the 16KB PRG-ROM in the upper bank as well
    memory.map(0x8000, prgRom)
    memory.map(0xC000, prgRom)
}
```

🧠 Other possible causes

1. **Incorrect reset vector**
   - At power-on, the NES CPU starts execution from the address located at `$FFFC–$FFFD` (little endian).
   - If your emulator reads the wrong value there or doesn't map PRG-ROM properly, the CPU may jump into the wrong code.

   Fix: Confirm that your memory mapper provides correct data at `0xFFFC`.
2. **Stack pointer / RAM issues**
   - RAM not zeroed out or stack misbehaving? SMB uses stack variables early. If your RAM contains garbage, the game logic might see the world number as zero.

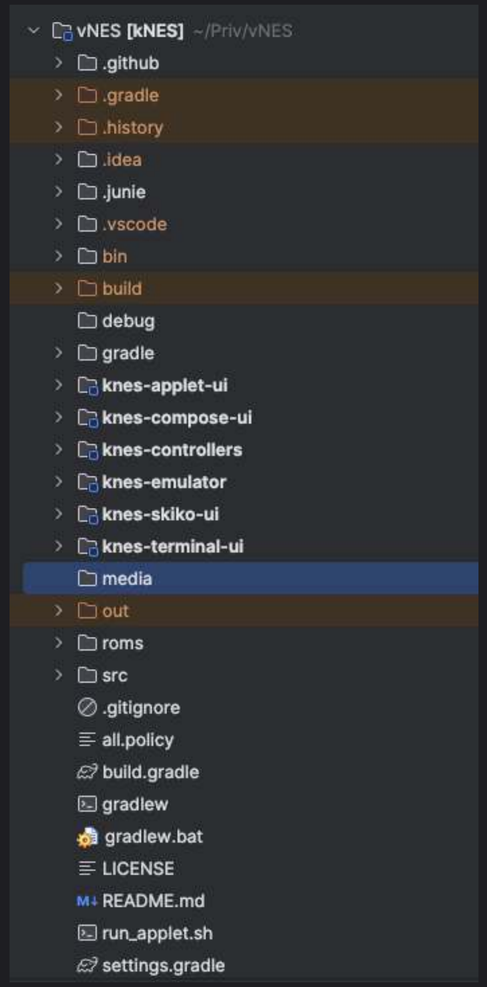
   Fix: Initialize internal RAM ($0000-$07FF) to 0x00 at startup.

"Standard" model simply one-shot correct answer.

4o was better than O3

This is one of the most well-known errors in gaming

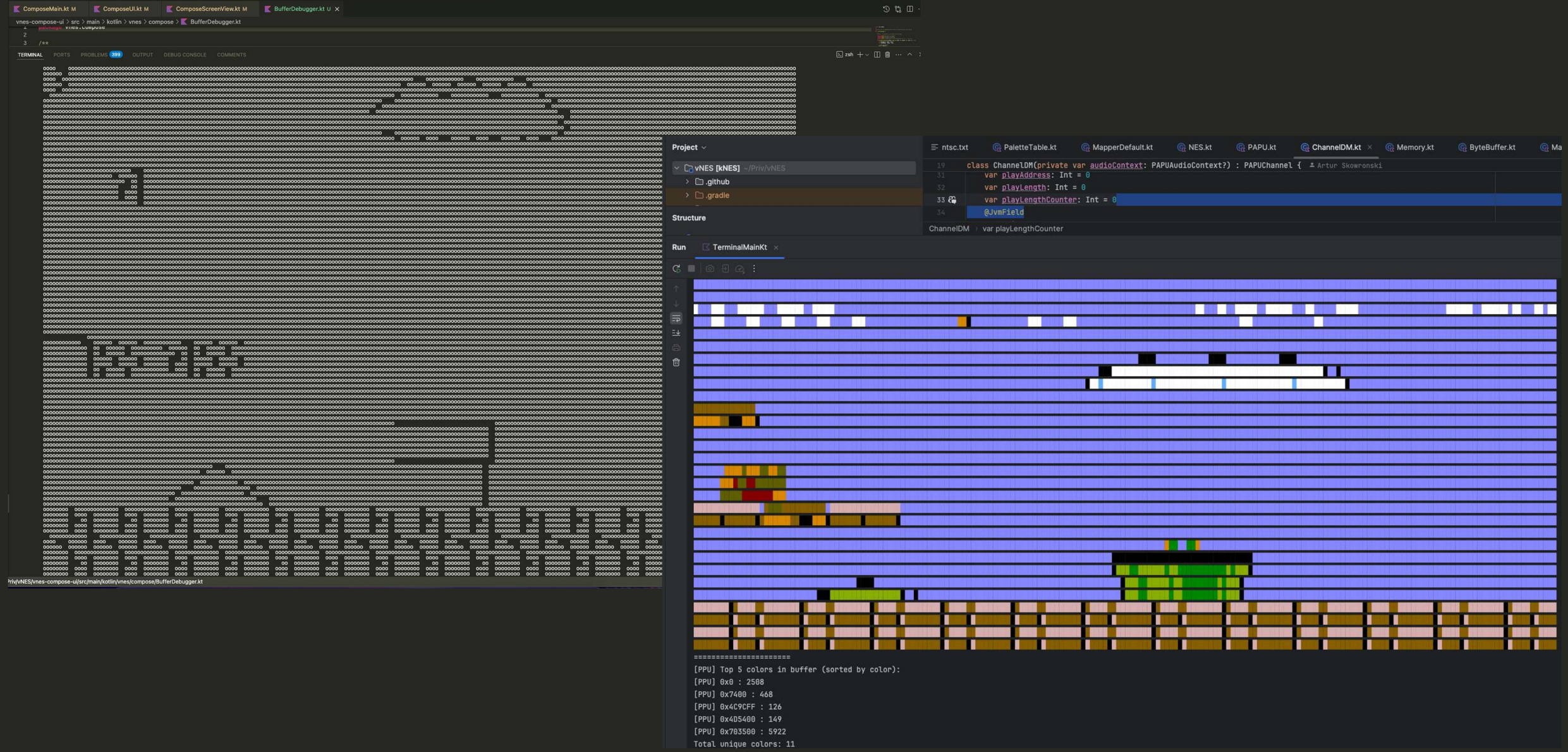
Why is it worth to do that? Ability to experiment ❤️



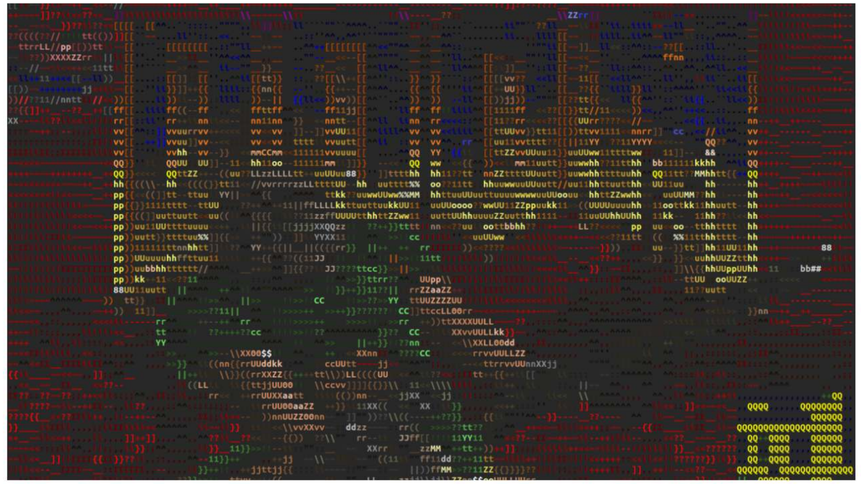

```kotlin
fun main() = application {
    val windowState = rememberWindowState(width = 800.dp, height = 600.dp)
    var isEmulatorRunning by remember { mutableStateOf(false) }

    // Create the UI factory and components
    val uiFactory = remember { ComposeUIFactory() }
    val screenView = remember { uiFactory.createScreenView(2) as ComposeScreenView }
    val controller = remember { KeyboardController() }
    val nes = remember { NES(null, uiFactory, screenView, controller) }
    val composeUI = remember { uiFactory.getComposeUI() }

    // Get the input handler from the UI factory using the new controller system
    val inputHandler = remember { uiFactory.createInputHandler(controller) as ComposeInputHandler }
```

We can compose **UIFactory** and **Controller** Implementation as we want

Swappable Components - Hardware and Control



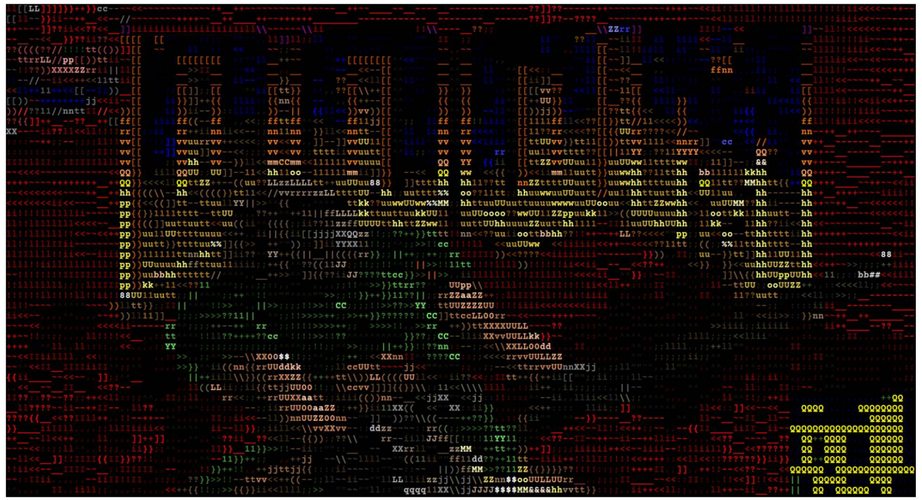
Rendering in Terminal



JediTermFX and DOOM – Running the Legendary Game in JavaFX

Techsenger
3 followers

DOOM-ASCII

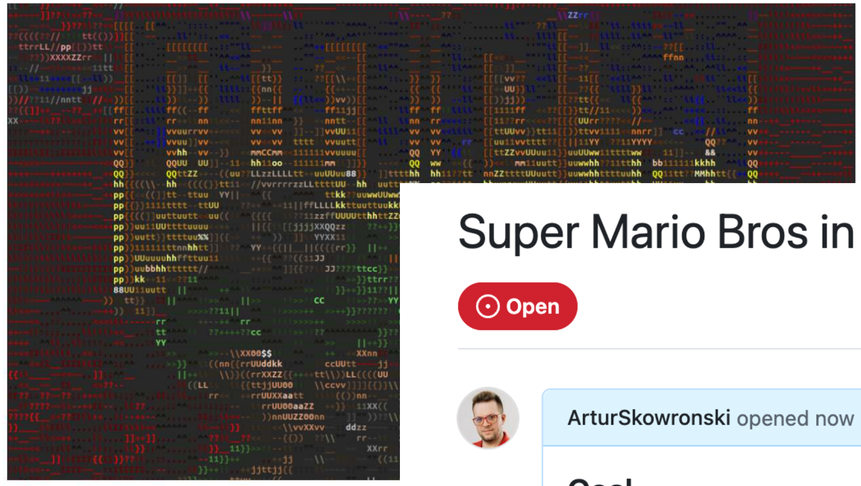


Text-based DOOM in your terminal!

Source-port of doomgeneric. Does not have sound.

You will need a WAD file (game data). If you don't own the game, the shareware version is freely available.

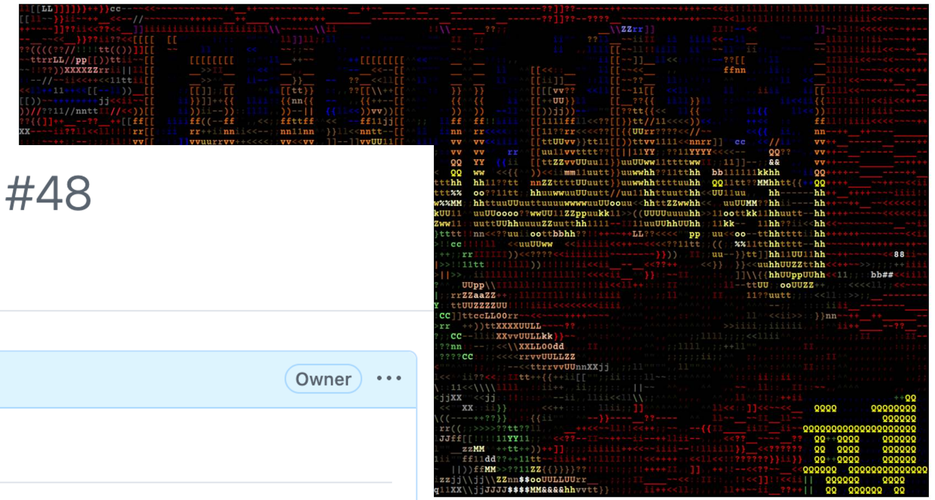
Rendering in Terminal



JediTermFX and D
Legendary Game i

Techsenger
3 followers

DOOM-ASCII



n the game, the shareware version is freely available.

Super Mario Bros in ASCII (JediTermFX Mode) #48

Open

ArturSkowronski opened now — Owner

Goal

Render **Super Mario Bros.** sprites in a terminal-style **ASCII mode** using @JediTermFx, as a switchable frontend to kNES (useful for demos, low-GPU environments, and debugging).

Scope

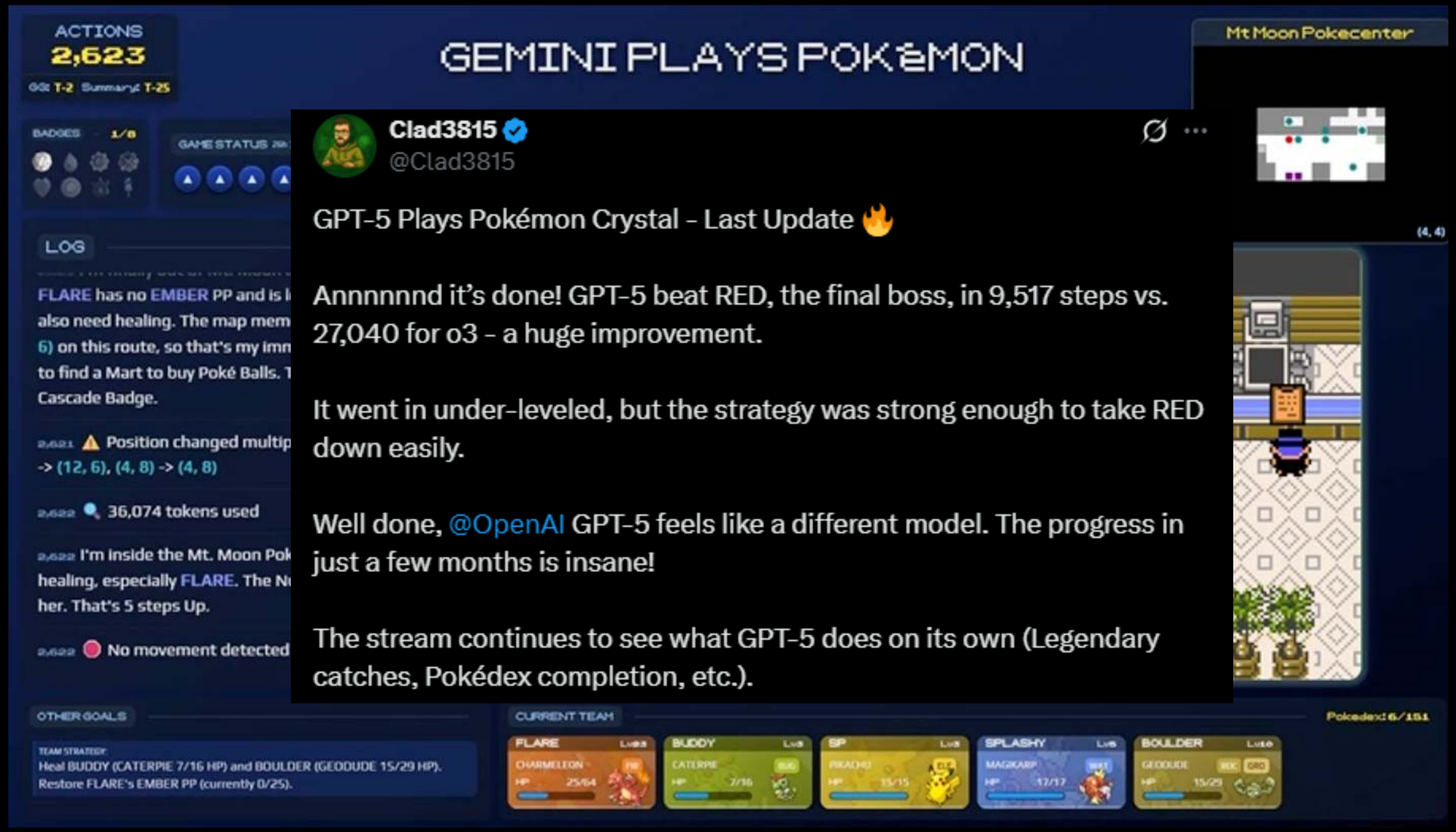
- New renderer that maps the 256×240 NES framebuffer → coarse grid (e.g., 80×75) with ASCII glyphs + basic color approximation.
- Toggle between **JavaFX Canvas** and **JediTermFX ASCII** (`--ascii` flag + runtime hotkey).
- Frame skip/turbo hooks to keep ASCII mode responsive.

Tasks

- [ ] Add `AsciiRenderer` implementing existing kNES rendering interface.
- [ ] Integrate **JediTermFX** surface and text pipeline.

Rendering in Terminal

Mobile Phone as a 🎮 through WebSockets (also multiplayer)



Gemini Plays... anything but not Super Mario Bros ❤️

Gemini Plays Final Fantasy (on kNES)

Final Fantasy is using custom mapper

ArturSkowronski / kNES

Type / to search

Code | Issues 4 | Pull requests 2 | Actions | Projects 1 | Models | Wiki | Security | Insights | Settings

Implement Gemini Plays Final Fantasy (on kNES) #47

Edit | New issue

Open

ArturSkowronski opened on Aug 3 · edited by ArturSkowronski Edits Owner

Enable a reproducible "Gemini plays Pokemon" loop on kNES with deterministic stepping, lightweight observations, and robust save/rewind - suitable for automated evaluation and demos.

Background

Final Fantasy (NES) uses **MMC1 (Mapper 1)** with 5-bit serial configuration writes, PRG/CHR banking, and mirroring changes. The game is **menu/text-heavy**, so pixel-only perception is brittle; we'll expose a minimal, structured HUD from RAM/PPU alongside cropped frames. LLM latency prevents 60 FPS control, requiring frame-skip/turbo and batched observations.

Requirements

Emulator fidelity & determinism

- [ ] Implement/verify **MMC1** accuracy (5-bit shift register, PRG/CHR banking, mirroring).
- [ ] Serialize **all** emulator state in savestates: CPU/PPU/APU, **MMC1 internal state**, SRAM, mapper latches.
- [ ] Savestates round-trip: save → load → frame-advance yields bit-identical outputs

Assignees
No one - Assign yourself

Labels
agentic-experiments

Projects
kNES
Status No status

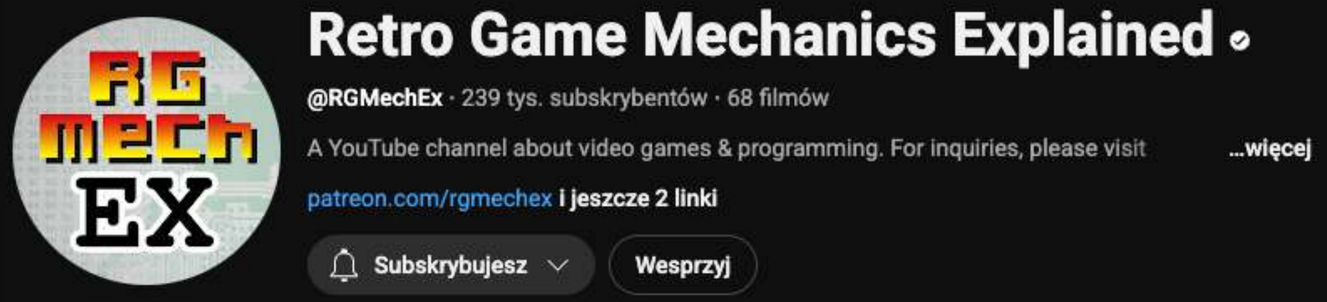
Milestone
No milestone

Relationships
None yet

Development

Final Fantasy is using custom mapper

But also we have direct access to memory

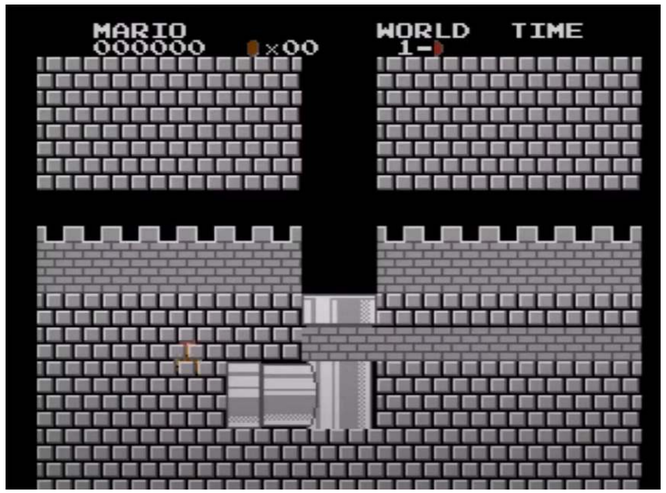


youtube.com/**@RGMechEx**

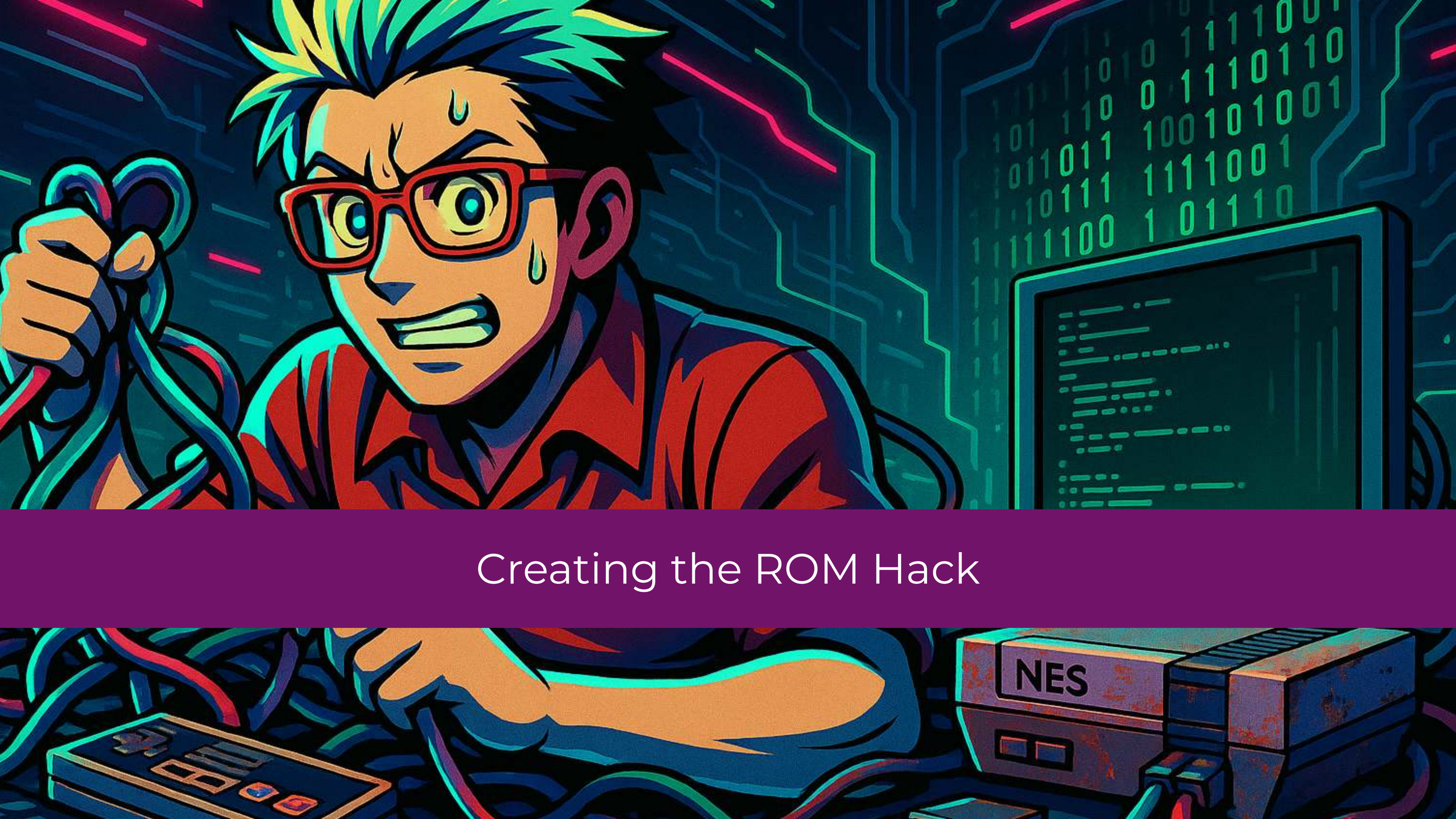
Strange Mario Worlds Generator

```
val cpuMemory: Memory = Memory(0x10000) // Main memory (internal to CPU)
val ppuMemory: Memory = Memory(0x8000) // VRAM memory (internal to PPU)
val sprMemory: Memory = Memory(0x100) // Sprite RAM  (internal to PPU)

class Memory(var memSize: Int) {
  var mem: ShortArray

    init {
        mem = ShortArray(memSize)
    }

    (…)
}
```

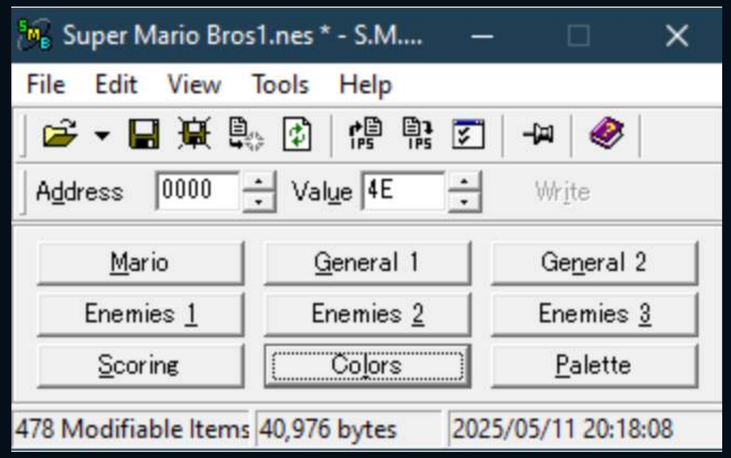
Memory is simple an array of registers 🤷

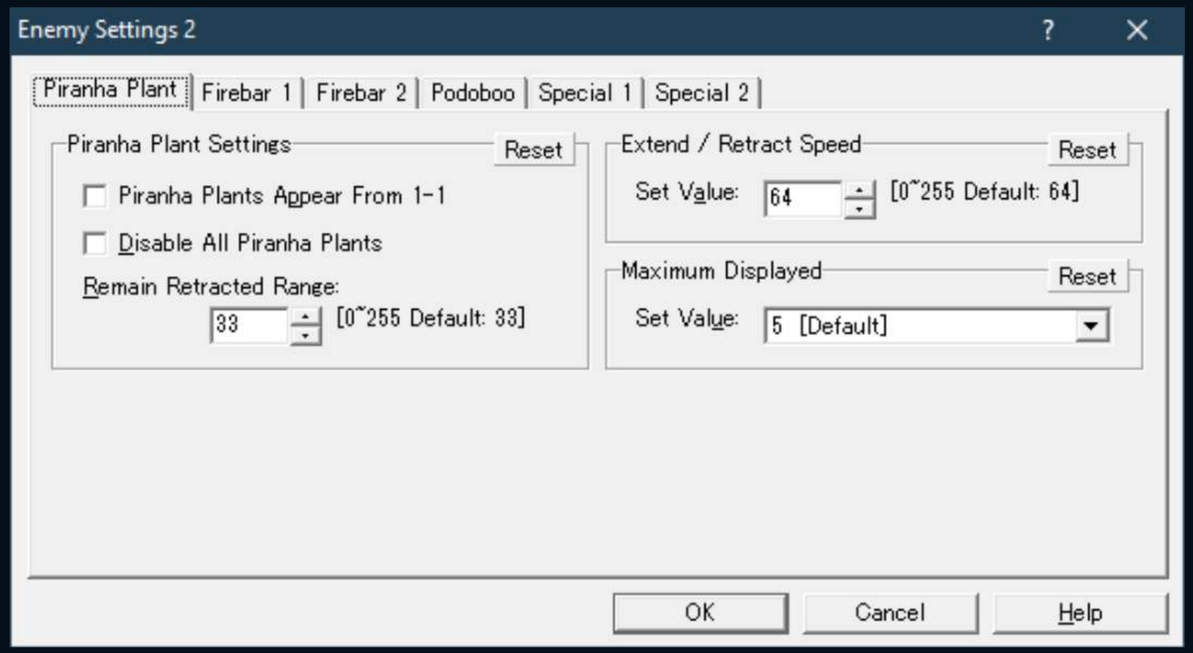


Strange Mario Worlds Generator



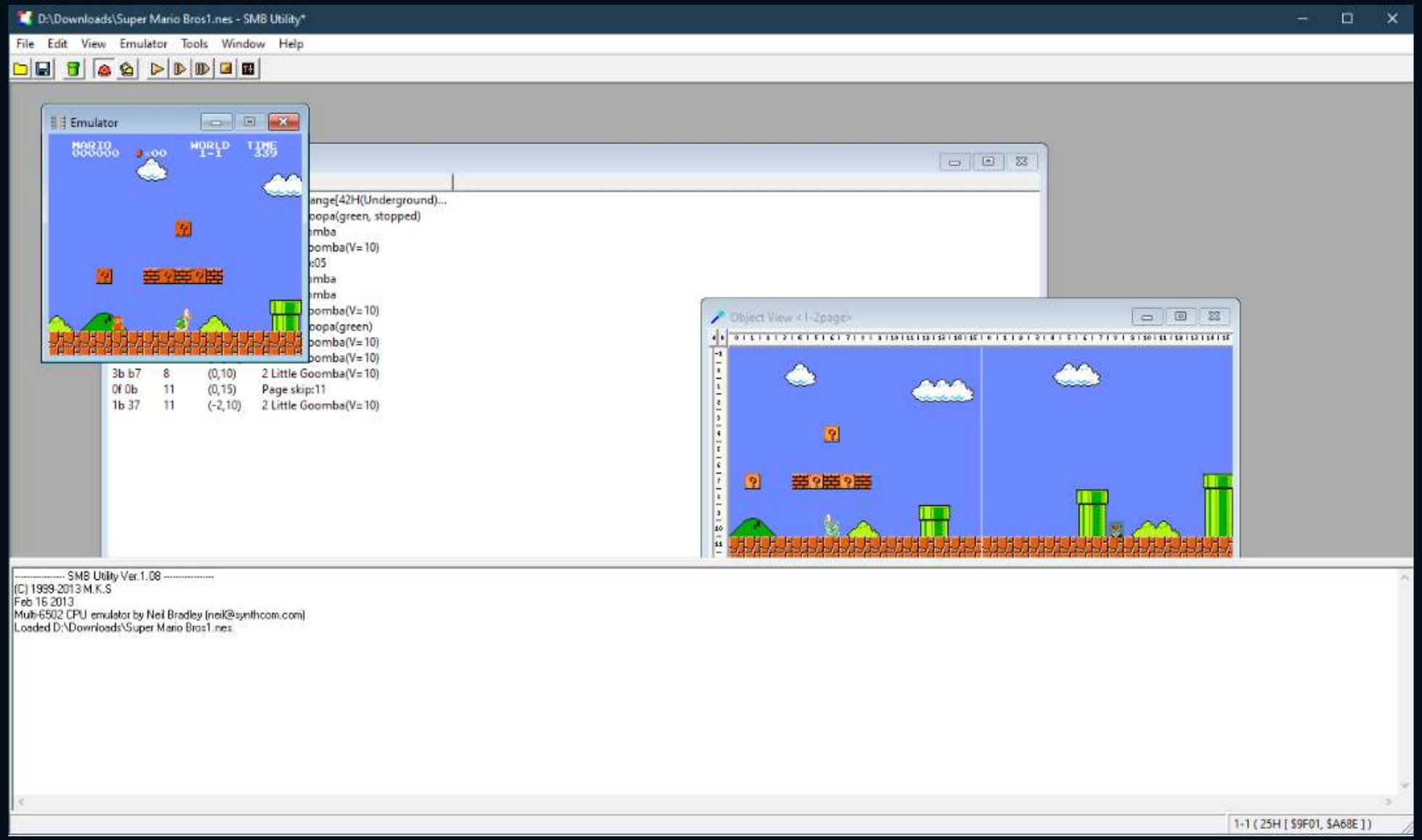
Creating the ROM Hack





Creating the ROM Hack

Creating the ROM Hack



Creating the ROM Hack

Nintendo Lawyer Lifts the Lid on Approach to Piracy and Emulation

"This is a point often debated."

BY SAYEM AHMED | UPDATED: JAN 15, 2025 2:20 PM | POSTED: JAN 15, 2025 2:12 PM

Nintendo is famously litigious when it comes to emulators. In March 202 developers of Nintendo Switch emulator Yuzu were ordered to pay $2.4 million in damages after a court settlement with Nintendo. In October 2024, Switch emulator Ryujinx ceased development following “contact from Nintendo.” In 2023, the developers behind Dolphin, an emulator for Gamecube and Wii, were advised against a full Steam release by lawyers at Valve, who were contacted by Nintendo’s lawyers alongside “strong legal wording.”

Demo... and questions

Thank you 🙇

@ArturSkowronski

VIRTUSLAB

jvm weekly