An adventure in writing an emulator in .NET Core.

$ whoami

Dale Nunns

Jira data capturer and occasional Software Developer

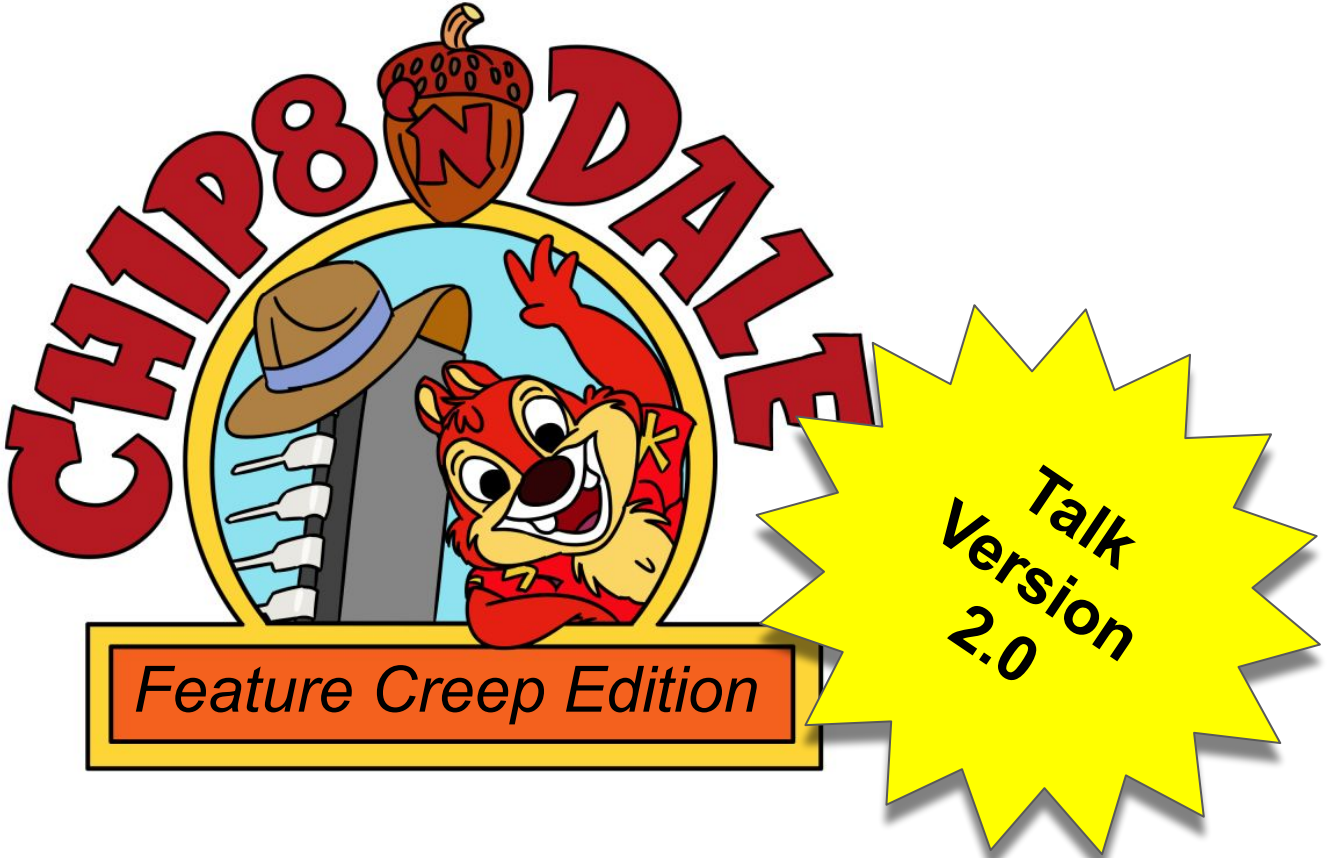
Father, Husband
Senior Software Developer

I dabble in hardware hacking, electronics, retro computing and making things.

Jack of all trades, serial skill collector and high-functioning hoarder.

@dale_nunns

https://xor.co.za



An adventure in writing an emulator in .NET Core.

(And a little Skia# and way too much GTK#)

In the beginning...

A friend and I were talking about emulators. (Hi Ross 😄)

(As one does)

Ross: How do emulators work?

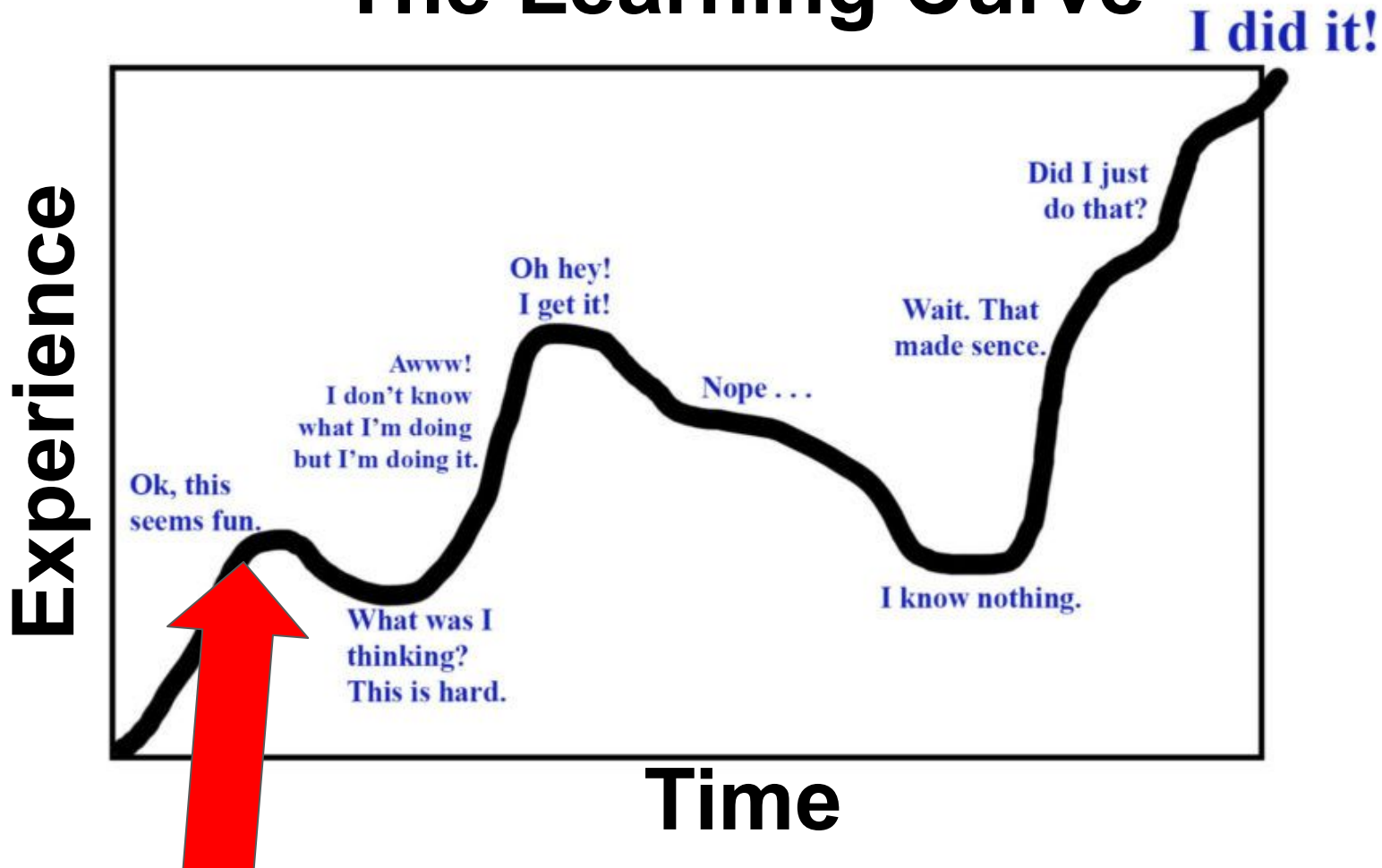
Me: I have a vague idea.

...

Me: Maybe I should learn more by writing one.

Me: *How hard could it be?*

The Learning Curve



How hard could it be?


IMPERIAL-BEDROOMS

What is an emulator?

What is an emulator?

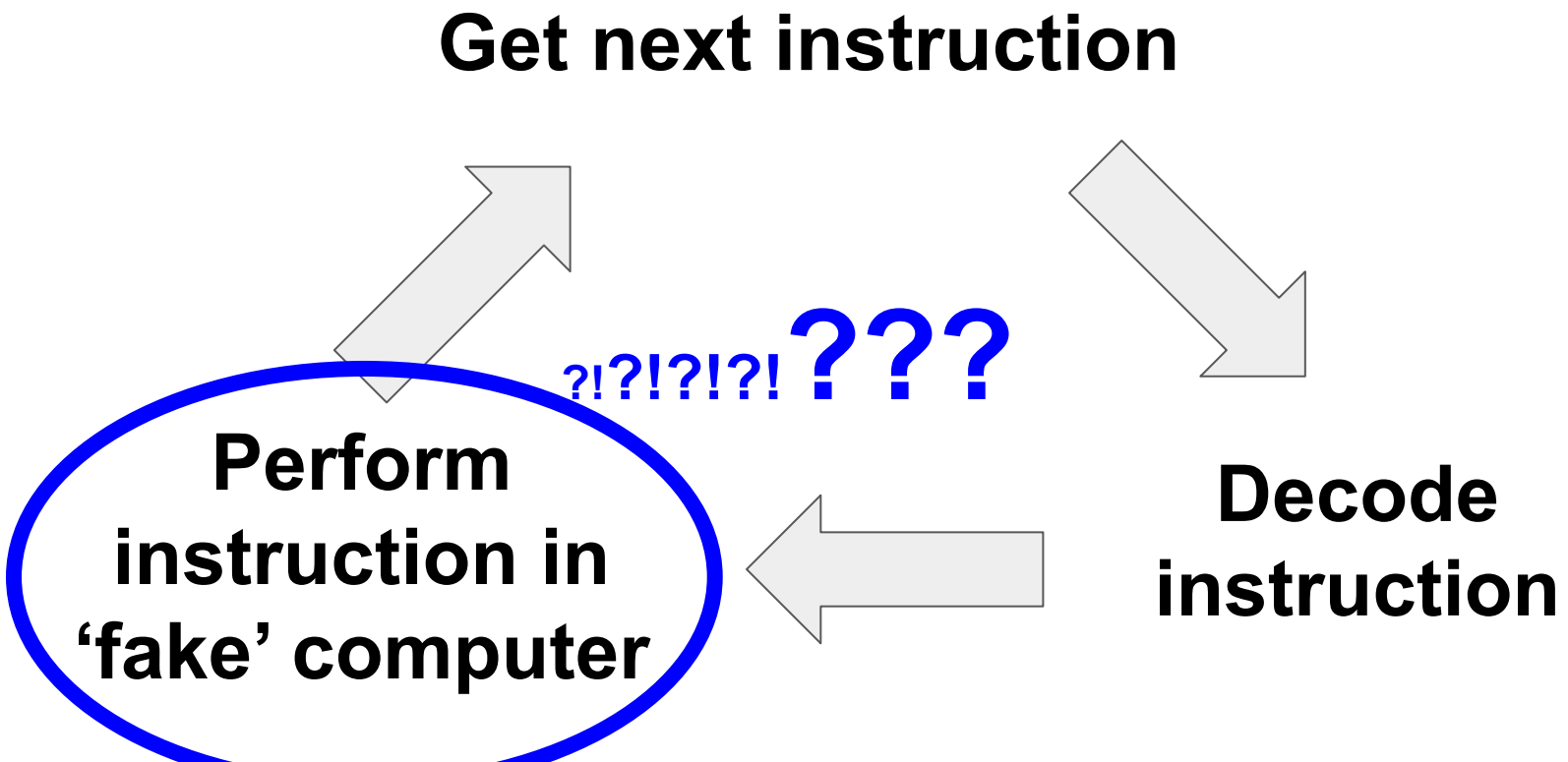
“...an emulator is hardware or software that enables one computer system (called the host) to behave like another computer system (called the guest). An emulator typically enables the host system to run software or use peripheral devices designed for the guest system.”

Why write an emulator?

Out of curiosity, for fun, to learn..

How does an emulator work?

How does an emulator work?



Emulator - your ‘fake’ computer...

- Byte array for RAM / ROM / Memory
- Variables for CPU registers
- Switch/IF/Case etc to decode instructions and run them.
- I/O functions that map to your host computers I/O for keyboard, sound, graphics etc

Cycle Accuracy

“..the most dominant *[SNES]* emulators are Nestopia and Nintendulator, requiring 800MHz and 1.6GHz, respectively, to attain full speed. The need for speed isn't because the emulators aren't well optimized: it's because they are a far more faithful recreation of the original NES hardware in software.” - *Accuracy takes power: one man’s 3GHz quest to build a perfect SNES emulator**

*https://arstechnica.com/gaming/2011/08/accuracy-takes-power-one-mans-3ghz-quest-to-build-a-perfect-snes-emulator/

SNES Specification

- CPU: 16-bit 65816 (**3.58MHz**)
- RAM: 128KB (1Mb), 64KB (0.5Mb) Video RAM.
- Graphics: Dedicated graphics processor.
- Colors: 32768 (256 on screen)
- Sprites: 128.
- Sprite Size: 64x64 pixels.
- Resolution: 512x448 pixels.
- Sound: 8-channel 8-bit Sony SPC700 digitized sound.

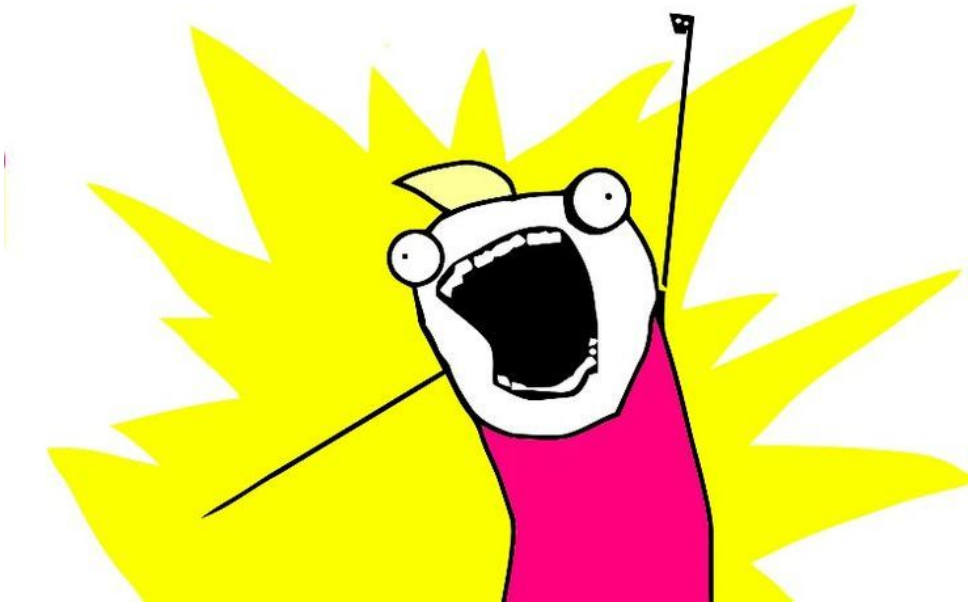
1.6GHz to emulate 3.58MHz ???

SNES Specification

0.279329608938547 5 ms to perform 1 clock cycle

- CPU: 16-bit 65816 (**3.58MHz**)
- RAM: 128KB (1Mb), 64KB (0.5
- Graphics: Dedicated graphics processor.
- Colors: 32768 (256 on screen)
- Sprites: 128.
- Sprite Size: 64x64 pixels.
- Resolution: 512x448 pixels.
- Sound: 8-channel 8-bit Sony SPC700 digitized sound.

What do you need to know about the device to write an emulator?



EVERYTHING!!

What do you need to know about the device to write an emulator?

- You need a thorough understanding of the device you want to emulate.
- You need to know everything about all the components in the system.
- How they’re connected together.
- How they work both the documented and undocumented features.

You need to emulate the bugs.

Even if it means it can’t do division properly.

Pentium FPDIV Bug

The processor might return incorrect binary floating point results when dividing a number.

You need to emulate undocumented features.

MOS 6502 has only 151 of the 256 available opcodes defined.

Leaving 105 that trigger strange and occasionally hard-to-predict actions.

Choosing a programming language.

Your language matters.

- Performance matters
- System level lan
- Low-Level
- With mber of
  bly

Your language doesn’t actually matter.

Well It Depends™

- As long as you’re not emulating anything too complex/fancy it doesn’t matter.
- All you need is to be able to manipulate bytes at a bit level.

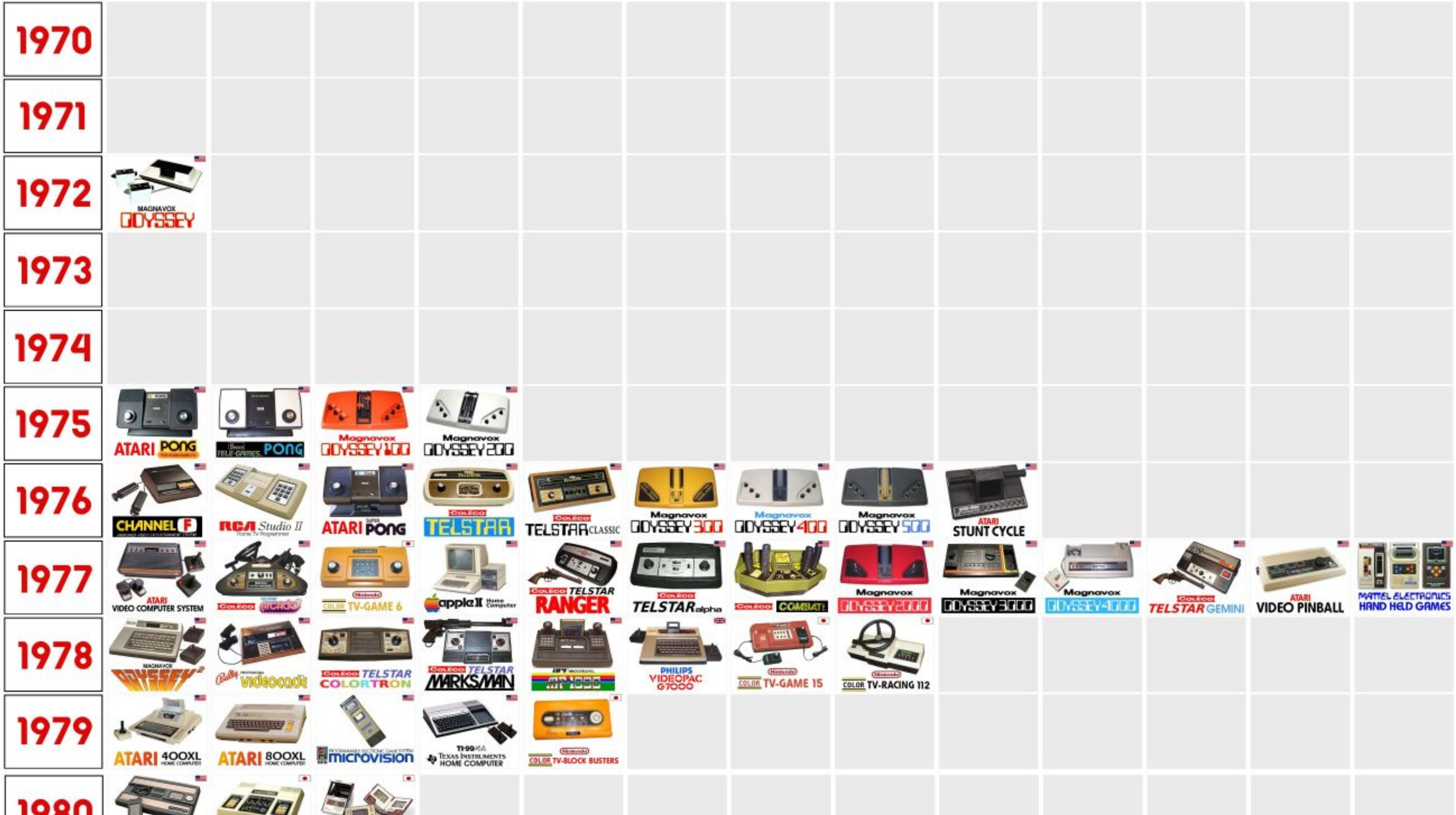
C# & .NET Core

- I know C# well
- It works on Linux, Windows and MacOS
- C# is close enough to C/C++ that code/structure can be easily replicated for porting to C/C++
- Good excuse to improve my .NET Core skills

Choosing a machine to emulate.

Choosing a machine to emulate.

- Needs to be a simple architecture
  - Low clock speed
  - No fancy graphics
  - No fancy sound
- Well documented


1970
1971
1972
MAGNAVOX ODYSSEY
1973
1974
1975
ATARI PONG
TELE-GAMES PONG
Magnavox ODYSSEY 100
Magnavox ODYSSEY 200
1976
CHANNEL F
RCA Studio II
ATARI SUPER PONG
COLECO TELSTAR
COLECO TELSTAR CLASSIC
Magnavox ODYSSEY 300
Magnavox ODYSSEY 400
Magnavox ODYSSEY 500
ATARI STUNT CYCLE
1977
ATARI VIDEO COMPUTER SYSTEM
COLECO
COLOR TV-GAME 6
apple II Home Computer
COLECO TELSTAR RANGER
COLECO TELSTAR alpha
COLECO COMBAT!
Magnavox ODYSSEY 2000
Magnavox ODYSSEY 3000
Magnavox ODYSSEY 4000
COLECO TELSTAR GEMINI
ATARI VIDEO PINBALL
MATTEL ELECTRONICS HAND HELD GAMES
1978
MAGNAVOX ODYSSEY²
Bally videocade
COLECO TELSTAR COLORTRON
COLECO TELSTAR MARKSMAN
PHILIPS VIDEOPAC G7000
COLOR TV-GAME 15
COLOR TV-RACING 112
1979
ATARI 400XL HOME COMPUTER
ATARI 800XL HOME COMPUTER
microvision
TI-99/4A Texas Instruments HOME COMPUTER
COLOR TV-BLOCK BUSTERS
1980

1977

Telmac 1800

COSMAC VIP

Photo 1: The COSMAC VIP system demonstrating its graphics abilities. The kit version of this product includes the board alone; the user must supply a standard video monitor (or modified TV) and inexpensive tape recorder.

COSMAC VIP, the RCA Fun Machine

Telmac 1800	COSMAC VIP
RCA 1802 CPU @ 1.76MHz	RCA 1802 CPU @ 1.76MHz
2 kB RAM, expandable to 4 kB	2 KB RAM (Expandable to 4 KB on board and 32 KB via an expansion slot)
Cassette tape interface	Cassette tape interface
64×32 pixels display resolution	CDP1861/CDP1864 video display chip - 64x32 pixel display resolution

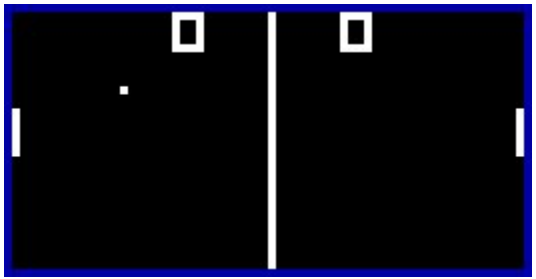
What do they have in common?

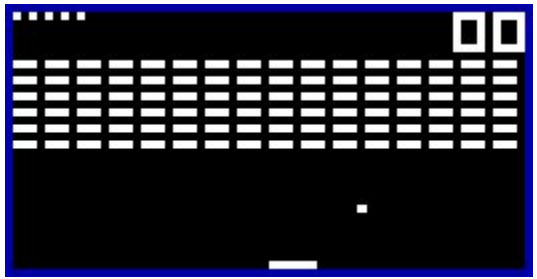
CHIP8

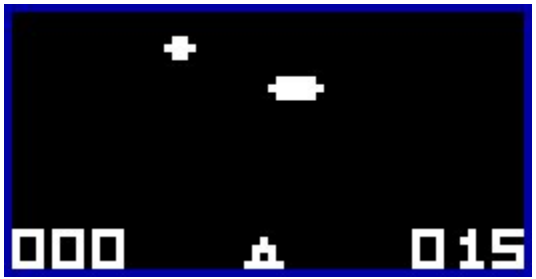
A gaming platform you've probably never heard of.

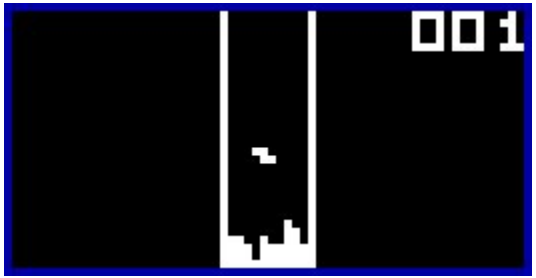
CHIP8 Games

There are a number of classic video games ported to CHIP-8, such as Pong, Arkanoid (breakout), Space Invaders and Tetris.









CHIP8

- Developed by Joseph Weisbecker in the mid 1970's
- Originally made to allow video games to be more easily programmed for the Telmac 1800 & COSMAC VIP computers.
- It's an interpreted programming language.
- CHIP-8 programs are run on a CHIP-8 virtual machine.

CHIP8 - Virtual Machine - Memory

CHIP8 was most commonly implemented on machines with 4KB of Memory.

- First **512 bytes** (0x00 - 0x200) was the actual interpreter.
- The uppermost **256 bytes** (0xF00-0xFFF) are reserved for display refresh
- The **96 bytes** below that (0xEA0-0xEFF) were reserved for call stack, internal use, and other variables
- **Leaving 3232 bytes for your game.**

CHIP8 - Virtual Machine - Registers

- CHIP-8 has 16 8-bit data registers named from V0 to VF
- VF doubles as a carry flag though and therefor should be avoided
- 16-bit wide address register called I and is often used with certain opcodes for memory operations

Stack

- The stack is only used to store return addresses when subroutines are called
- 48 bytes for up to 24 levels of nesting

CHIP8 - Virtual Machine - Timers

CHIP-8 has two timers. They both count down at 60 hertz, until they reach 0

- Delay timer: This timer is intended to be used for timing the events of games. Its value can be set and read.
- Sound timer: This timer is used for sound effects. When its value is nonzero, a beeping sound is made.

Input

- Input is done with a hex keyboard that has 16 keys which range from 0 to F. The '8', '4', '6', and '2' keys are typically used for directional input.

CHIP8 - Virtual Machine - Graphics

- CHIP-8 Display resolution is 64×32 pixels, and color is monochrome
- Graphics are drawn exclusively using sprites.
- Sprites are 8 pixels wide and 1-15 pixels high.
- Sprite pixels are XOR'd with corresponding screen pixels
- The carry flag (VF) is set to 1 if any screen pixels are flipped from set to unset when a sprite is drawn and set to 0 otherwise. (Can be used for collision detection)

CHIP8 - Virtual Machine - Opcodes

- CHIP-8 has 35 opcodes, which are all two bytes long

Example Opcode

8XY4 - Adds VY to VX. VF is set to 1 when there's a carry, and to 0 when there isn't.

80A4

Core CHIP8

My emulator in .NET Core

How legal is it?

IANAL

But it’s been over 40 years

How long did it take to write?

- I could only work 1-2 hours a night (small kids + wife).
- Approximately 8 hours to write emulator core + console UI.
- About 10 hours to debug and get it close to working.
- GTK# + Skia UI - 30 hours (I got carried away)
- 18 hours for a basic emulator.
- **Slides for this talk took significantly longer than writing the emulator (even including all the feature creep).**

What did I learn?

- I suck at bit-shifting and general bit manipulation (well I did when I started)
- .NET Core + VS Code is actually a nice dev environment.
- Console on Windows Suck compared to Linux (New Windows Terminal might fix that)
- Google Slides isn’t terrible (I hope)

DEMO TIME