State of Retro Gaming in Emacs

Vasilij Schneidermann

April 2019

Outline

Section 1

Intro

About

- Vasilij Schneidermann, 26
- Software developer at bevuta IT, Cologne
- mail@vasilij.de
- https://github.com/wasamasa
- http://emacshorrors.com/
- http://emacsninja.com/

Motivation

- Emacs is the ultimate procrastination machine
- Many fun demonstrations:
  - Order salad online
  - Window manager
  - IRC bot
  - Textual web browser
  - Basic games
  - 3D maze
  - Z-Machine emulator
  - Audio/video editor
  - Sex toy controller
- Can we emulate retro games at 60 FPS?

Context

- Prior art at FrOSCon/Quasiconf: Audiovisual demonstrations
- NES emulators are supposed to be simple
- Random Japanese guy beat me to the punch
- Recommended emulation project: CHIP-8
- Alternative: Intel 8080 running Space Invaders or CP/M
- Then someone else released a GB emulator...

Section 2

Interactive demonstrations

NES

- https://github.com/gongo/emacs-nes
- Super slow (100x), doesn't go beyond initial game screen
- Most time spent in rendering
- Could maybe be made to work at acceptable speed with lots of frameskip?

GB

- https://github.com/vreeze/eboy
- WIP, released in a hurry after I released mine
- Almost playable thanks to lots of frameskip
- Only Tetris works
- The most popular

CHIP-8

- https://github.com/wasamasa/chip8.el
- Pretty much finished, <1000SLOC
- Supports Super CHIP-8 extensions
- Runs at full speed, games behave OK

Section 3

Fun facts about `chip8.el`

What the hell is a CHIP-8 anyway?

- It's a VM, not a console
- Designed for easy porting of home computer games
- Not terribly successful
- Small community of enthusiasts writing games for it
- There are even a few demos!

System specs

- CPU: 8-Bit, 16 general-purpose registers, 36 instructions, each two bytes large
- RAM: 4KB
- Stack: 16 return addresses
- Resolution: 64 x 32 black/white pixels
- Rendering: Sprites are drawn in XOR mode
- Sound: Monotone buzzer
- Input: Hexadecimal keypad

Goals

- Coming up with a name
- Obtaining a ROM pack
- Understanding the system
- Basic RE tools
- Rendering
- Beeps
- Make as many games run as possible
- No debugger

How does it work?

- Runs at an unspecified speed
- Sound and delay timer count down at 60FPS
- Game is loaded up at `#x200` into RAM
- Program counter is set to `#x200`
- Decode instruction, execute, loop

Game loop woes

- Game approach: Do stuff, wait, repeat
- Doesn't work terribly well in Emacs due to user input
- Interruptible sleep: Unpredictable
- Un-interruptable sleep: Freezes
- Timers: Inversion of control, allows user input to happen
- Call a timer function at 60FPS, don't do too much in it:
  - Execute CPU cycle(s)
  - Decrement sound/delay registers
  - Repaint

Mapping the system to Emacs Lisp

- It's all integers and vectors (of integers)
- RAM, registers, return stack, key state, screen, etc.
- Stored in global variables
- No lists are used at all
- Side effect: No consing happens, no GC pauses
- Registers are mapped to a vector with an enum macro
- Side effect: Much easier decoding

Built-in sprites

- Unspecified
- Everyone steals them from the canonical implementation
- Super CHIP-8 has bigger sprites
- I upscaled the small ones using a terrible Ruby oneliner
- Lesson here: Sometimes it's not worth being clever

Decoding instructions

- All instructions are two bytes
- Arguments are encoded inside them
- `JP nnn` for example maps to `#x1nnn`
- Type extracted by masking with `#xF000`, then shifting by 12 bits
- Argument by masking with `#x0FFF` (no shift needed)
- Common patterns emerge, like addresses being the last three nibbles
- Big `cond` dispatching on the type and executing side effects
- Common side effect: Bumping program counter by two

Interactive Testing

- Initially: Execute ROM until user interrupt
- Use a debug command to render screen to a buffer
- Maze: Small ROM, few instructions
- There are many more ROMs that just display a static screen
- I went through them all and added instructions as needed

Debugging

- My usual approach of using edebug was ineffective
- Therefore: Logging it is
- I compared my log output with an instrumented version of evhan's chick-8 emulator
- If the logs diverge, that's where the bug lies
- Future project idea: A CHIP-8 debugger, game development environment
- Inspirations:
  - https://massung.github.io/CHIP-8/
  - http://johnearnest.github.io/Octo/

Analysis

- Writing a disassembler is simple, but tedious
- Adding analysis functionality is particularly tricky
- Idea: Reuse radare2 framework, add analysis/disasm plugin
- I wrote one in Python, then discovered there is one in core...
- I then improved that one to the same level

Unit testing

- Goal: Coverage of all instructions and what they do
- More of a safety net, doesn't catch everything
- Built-in ERT library isn't terribly good
- https://github.com/jorgenschaefer/emacs-buttercup is better
- Each test initializes the VM, loads up code, executes the `chip8-cycle` function, checks for side effects

Rendering

- By far the trickiest part
- I intentionally decided against using a library
- Creating SVGs: Too expensive
- Creating/mutating strings: Too expensive or complicated
- Changing SVG tiles: Gaps between lines
- Bool vector backed XPM: Caching effects ruin everything
- Plain text with background color: Perfect

Rendering optimization

- Initially: Clear buffer, insert text
- Better: Move across text, delete and insert changed parts
- Optimization: Track dirty frame
- Changed parts: Diff two framebuffers
- Final optimization: Erasing text was slow, changing background text property was way faster
- Future optimization: Make a C module with a fast canvas

Garbage collection

- Occasionally there was a small stutter
- This turned out to be code duplicating vectors
- Solution: Writing a `memcpy`-style function
- Delays after every few tests
- Solution: Using a `memset`-like function instead of recreating vectors
- Hard to profile and spot, may require a custom package

Sound

- You only need a beep, so no difficulties emulating it
- Playing it is hard because Emacs only supports synchronous playback...
- Emacs processes are asynchronous, so controlling one works
- `mplayer` has a slave mode, `mpv` supports listening on a FIFO for commands
- Proof of concept:
  - Start paused `mpv` with a FIFO in loop mode
  - Send pause/unpause command to the FIFO

User input (non-blocking)

- Checking for key press state: Unsupported
- Solution: Global key handler stores key press timestamp
- Compare the timestamp with current time against timeout
- Key considered pressed if less than timeout
- Requires tweaking to feel "natural"

User input (blocking)

- Tricky due to inversion of control
- Required me to do a state machine rewrite
- The command transfers the emulator into a waiting state
- The global key handler checks for that state and transfers to the playing state

Super CHIP-8

- Supports more interesting games
- Proper scrolling support requires tricks to do in-place
- Doubled resolution required an extra rendering optimization
- It's possible to switch between both modes, making it tricky to implement:
  - You could always work in high-res and downscale if needed
  - Alternatively: Switch between low-res and high-res screen to render to
  - I went for the latter

Other stuff

- Sometimes games deviate from the reference, conflicting with it
- Sometimes it's unclear whether it's worth it to support an obscure feature
- I'm not good at games and didn't enjoy playing them
- However: You gain great insight how the machine works

Section 4

Outro

What next?

- Maybe an Intel 8080 emulator running CP/M
- Maybe experimentation with faster rendering
- More serious stuff in CHICKEN, like NES or GB emulator

Questions?