State of Retro Gaming in Emacs

Vasilij Schneidermann

November 2019

Outline

Abschnitt 1

Intro

Sprecher

- Vasilij Schneidermann, 27
- Cyber Security Consultant bei msg
- mail@vasilij.de
- https://github.com/wasamasa
- http://brause.cc/
- http://emacsninja.com/

Motivation

- Emacs ist ein großartiger Zeitvertreib
- Viele kuriose Demonstrationen:
  - Salat online bestellen
  - Window-Manager
  - IRC-Bot
  - Text-Browser
  - Einfache Spiele
  - 3D-Labyrinth
  - Z-Machine-Emulator
  - Audio-/Video-Editor
  - Sex-Toy-Controller
- Ist es möglich Spiele performant zu emulieren?

Kontext

- Voriger Vortrag auf FrOSCon/Quasiconf: Audiovisuelle Demos
- NES-Emulatoren sollen leicht sein
- Zufälliger Mensch im Internet™ kam mir zuvor
- Empfohlenes Emulations-Projekt für Anfänger: CHIP-8
- Alternative: Space Invaders auf dem Intel 8080 oder CP/M
- Anderer Mensch™ veröffentlichte einen GB-Emulator...

Abschnitt 2

Interaktive Demos

NES

- https://github.com/gongo/emacs-nes
- Sehr langsam (100x), schafft nicht mehr als den Startbildschirm
- Meiste Zeit wird beim Rendering verbraten
- Könnte mit viel Frameskip funktionieren...

GB

- https://github.com/vreeze/eboy
- WIP, wurde in Eile nach meinem Projekt veröffentlicht
- Fast spielbar dank viel Frameskip
- Nur Tetris funktioniert
- Populärstes Projekt von allen

CHIP-8

- https://github.com/wasamasa/chip8.el
- Abgeschlossen, <1000SLOC
- Unterstützt die Super CHIP-8-Erweiterungen
- Läuft auf voller Geschwindigkeit ohne Frameskip
- Vollständige Kompatibilität

Abschnitt 3

Fun Facts über `chip8.el`

Was zur Hölle ist ein CHIP-8?

- Es ist eine VM, keine Spielekonsole
- Design auf einfache Portierbarkeit von Spielen ausgelegt
- Nicht besonders erfolgreich
- Kleine Gruppe von Enthusiasten welche Spiele dafür entwickeln
- Es existieren sogar Demos!

Systemspezifikationen

- CPU: 8-Bit, 16 universelle Register, 36 Instruktionen, fixe Instruktionsgröße
- RAM: 4KB
- Stack: 16 Rücksprungadressen
- Auflösung: 64 x 32 schwarz-weiße Pixel
- Rendering: Sprites werden im XOR-Modus gezeichnet
- Sound: Monotoner Piepser
- Input: Hexadezimales Keypad

Ziele

- Namen ausdenken
- ROM-Pack und Doku besorgen
- Die Plattform verstehen
- Bau von Reverse-Engineering-Tools
- Rendering
- Sound
- Möglichst viele Spiele ans Laufen bekommen
- Kein Debugger

Wie funktioniert es?

- Ausführungsgeschwindigkeit: Nicht spezifiziert
- Sound- und Delay-Timer zählen mit 60FPS auf 0 herunter
- Spiel wird in den RAM bei #x200 geladen
- Program Counter wird auf #x200 gesetzt
- Instruktion dekodieren, ausführen, wiederholen

Probleme mit der Game-Loop

- Typischer Ansatz: Tue Dinge™, warte ab, wiederholen
- Funktioniert nicht gut in Emacs, da interaktives Programm
- Nicht blockierendes Warten: Unvorhersehbares Verhalten
- Blockierendes Warten: Editor hängt sich auf
- Timer: Kontrolle wird dem Editor überlassen, Benutzereingabe ist möglich
- Es wird eine Funktion mit 60Hz aufgerufen, diese darf nicht zu viel tun:
  - CPU-Zyklen ausführen
  - Sound-/Delay-Register dekrementieren
  - Rendering

Abbilden der VM auf Emacs Lisp

- Schlussendlich sind es nur Zahlen (und Arrays von Zahlen)
- RAM, Register, Stack, Tastatur-State, Bildschirm, etc.
- Alles in globalen Variablen gespeichert
- Es werden keine Listen verwendet
- Seiteneffekt: Vorhersehbares Laufzeitverhalten, keine GC-Pausen
- Register werden als Array mithilfe eines `enum`-Makros abgebildet
- Seiteneffekt: Viel einfacheres Dekodieren von Instruktionen

Eingebaute Sprites

- Nicht spezifiziert
- Jeder klaut diese von der Referenzimplementierung
- Super CHIP-8 hat doppelt so große Sprites
- Hochskaliert mit einer unschönen Zeile Ruby-Code
- Lektion: Manchmal ist es nicht den Hirnschmalz wert

Dekodieren von Instruktionen

- Jede Instruktion ist zwei Bytes groß
- Argumente werden in diesen Bytes enkodiert
- `JP nnn` entspricht zum Beispiel `#x1nnn`
- Typ: `#xF000` als Bitmaske, gefolgt von Shift um 12 Bits
- Argument: `#x0FFF` als Bitmaske (kein Shift nötig)
- Typisches Muster: Adressen sind immer die letzten drei Nibbles
- Großes `cond` stellt den Typ fest und führt Seiteneffekte aus
- Typischer Seiteneffekt: Program Counter um 2 Bytes erhöhen

Interaktives Testen

- Erster Ansatz: ROM ausführen bis der Benutzer die Schleife abbricht
- Debug-Command um den Bildschirminhalt anzuzeigen
- Maze: Kleines ROM, wenige Instruktionen
- Viele ROMs welche nur einen statischen Bildschirm anzeigen
- Ich habe alle getestet und fehlende Instruktionen implementiert

Debugging

- Debugger sind relativ nutzlos in diesem Szenario
- Aus diesem Grund: Logging!
- In besonders schwierigen Fällen: Vergleich von Logs meines Emulators mit einem anderen präparierten Emulator
- Erster Unterschied in den Logs: Quelle des Bugs
- Projektidee: CHIP-8 Debugger, Spiele-Entwicklungs-Umgebung
- Inspiration:
  - https://massung.github.io/CHIP-8/
  - http://johnearnest.github.io/Octo/

Analyse

- Es ist einfach, aber nervig einen Disassembler zu schreiben
- Analyse-Funktionalität hinzuzufügen ist nicht trivial
- Idee: radare2 nutzen, Analyse/Disassembler-Plugin schreiben
- Anfangs in Python geschrieben, später stellte ich fest, dass das Projekt schon Plugins in C hat. . .
- Diese wurden auf das gleiche Niveau wie die Python-Versionen verbessert
- Grafische Demo von Analyse-Output

Unit Tests

- Ziel: Komplette Abdeckung aller Instruktionen und Seiteneffekte
- Dient als Sicherheitsnetz, deckt nicht sämtliche Fehlerquellen ab
- Eingebaute ERT-Bibliothek ist nicht besonders toll
- https://github.com/jorgenschaefer/emacs-buttercup ist besser
- Jeder Test initialisiert die VM, lädt Maschinencode, führt einen CPU-Zyklus aus und prüft auf Seiteneffekte
- Andere Idee: Ausführbare Spezifikation einer CPU (siehe ARM)

Rendering

- Deutlich schwieriger als alles andere
- Ich habe mich bewusst gegen eine fertige Bibliothek entschieden
- SVGs erzeugen: Dauert zu lange
- Erzeugen/Mutieren von Strings: Dauert zu lange, zu komplex
- SVG-Tiles: Lücken zwischen Bildschirmzeilen
- XPM-Bild mit Bool-Vector-Repräsentation: Caching ruiniert alles
- Text mit Hintergrundfarbe: Ideale Lösung

Rendering-Optimierungen

- Anfangs: Buffer-Inhalt löschen, Text einfügen
- Optimierung 1: Navigation im Text, geänderte Teile löschen und neue einfügen
- Optimierung 2: Dirty Frame Tracking
- Geänderte Teile: Unterschiede zwischen zwei Framebuffer finden
- Optimierung 3: Text löschen ist langsam, Texthintergrundfarbe ändern ist deutlich schneller
- Zukunftsmusik: C-Modul für ein schnelles Canvas-Objekt schreiben

Garbage Collection

- Problem: Gelegentliches Stottern
- Problemquelle: Code welcher Arrays dupliziert
- Lösung: Funktion à la `memcpy` schreiben
- Problem: Alle paar Tests gibt es eine konstante Verzögerung
- Lösung: Funktion à la `memset` nutzen statt neue Arrays zu erzeugen
- Es gibt keine guten Werkzeuge um solche Probleme zu debuggen

Sound

- Emulation ist nicht schwierig, da man nur konstant piepsen muss
- Abspielen des Tons ist schwierig, da Emacs nur synchrone Wiedergabe unterstützt
- Emacs unterstützt asynchrone Prozesse
- `mpv` kann mit einer FIFO Befehle akzeptieren
- Proof of Concept:
  - `mpv` im Loop-Mode pausiert starten, mit einer FIFO
  - Senden eines Pause-/Wiedergabe-Befehls an die FIFO

Benutzereingabe (nicht blockierend)

- Abfrage des Status einer Taste: Nicht unterstützt
- Lösung: Globaler Key-Handler welcher letzte Tastendruck und Zeitstempel speichert
- Vergleich der aktuellen Zeit mit letzter Zeit gegen einen Timeout
- Liegt man unter dem Timeout, zählt die Taste als noch gedrückt
- Fühlt sich ohne weitere Anpassungen nicht besonders flüssig an

Benutzereingabe (blockierend)

- Schwierig aufgrund der ungewöhlichen Game-Loop
- Die bestehende State-Machine (Abspielen/Pause) musste umgeschrieben werden
- Die Instruktion wechselt den Emulator in einen Warten-Zustand
- Globaler Key-Handler prüft auf diesen Zustand und wechselt zum normalen Abspiel-Zustand

Super CHIP-8

- Unterstützt interessantere Spiele
- Ordentliches Scrolling erfordert Trickserei
- Verdoppelte Auflösung erfordert Rendering-Optimierung
- Man kann zwischen beiden Auflösungen wechseln, mögliche Ansätze sind:
  - Immer in hoher Auflösung rendern, bei Bedarf herunterskalieren
  - Alternative: Es wird zu einem von zwei Bildschirmen je nach aktueller Auflösung gerendert
  - Ich habe mich für letzteres entschieden...

Weitere Anmerkungen

- Manchmal weichen Spiele von der Spezifikation ab, dies führt zu Konflikten
- Manchmal ist es unklar ob es sich lohnt obskure Funktionalität zu unterstützen
- Ich bin kein guter Spieler, mir macht es nicht besonders viel Spaß Spiele zu spielen
- Dennoch: Man versteht deutlich besser wie ein Computer funktioniert

Abschnitt 4

Outro

Mögliche nächste Schritte

- Ein Intel 8080 Emulator welcher das CP/M-Betriebssystem ausführt (Betriebssystem im Betriebssystem)
- Experimente mit schnellem Rendering
- Schwierigere Emulationsprojekte in einer tauglicheren Programmiersprache

Fragen?