Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Iniciando no desenvolvimento de emuladores com CHIP-8

Diego Pereira Alvarez

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Orientador: Prof. Dr. Siang Wun Song

São Paulo
Março de 2021

Resumo

Diego Pereira Alvarez. **Iniciando no desenvolvimento de emuladores com CHIP-8**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

Emulação é a principal técnica utilizada para estudo e preservação de plataformas de hardware antigas e não mais disponíveis. É uma área desafiadora, que envolve conceitos de diversas áreas de computação e engenharia elétrica. O objetivo deste trabalho é ser uma introdução a área de emulação. São estudados os conceitos e técnicas fundamentais da área, assim como a arquitetura de um emulador.

Como aplicação prática desses conceitos é desenvolvido um emulador de CHIP-8, uma plataforma simples voltada ao desenvolvimento de jogos. São documentados o processo de escolha da plataforma e coleta de documentação técnica, assim como sua arquitetura, técnicas de emulação empregadas e principais decisões de projeto.

Palavras-chave: Emulador. CHIP-8. Python. Jogos.

Abstract

Diego Pereira Alvarez. **Iniciando no desenvolvimento de emuladores com CHIP-8**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2020.

Emulation is the main technique used in the study and preservation of old hardware platforms that are no longer available. It is a challenging area, involving concepts from many different areas of computing and electrical engineering. The goal of this monograph is to be an introduction to the area of emulation. Basic concepts and techniques of the area are studied, as well as the architecture of an emulator.

As a practical application of those concepts a CHIP-8 emulator is developed, which is a simple platform focused on game development. The platform selection and documentation collection process are documented, as well at its architecture, utilized emulation techniques and main project decisions.

Keywords: Emulator. CHIP-8. Python. Games.

Sumário

Capítulo 1

Introdução a emulação

1.1 Definição e aplicações

É difícil definir o termo "emulação" precisamente pois seu significado exato varia entre diferentes áreas e depende do contexto onde é aplicado. Durante este trabalho o termo "emulador" será utilizado para se referir a uma recriação digital e funcional de um sistema de hardware específico, criado com o objetivo de imitar, da forma mais fiel possível, todos os comportamentos deste sistema, incluindo bugs e limitações presentes no sistema original.

Em termos práticos, um emulador toma a forma de um software que possibilita a execução de programas desenvolvidos para uma plataforma de hardware em outra plataforma diferente da original. Existem diversas aplicações práticas para emulação. Por exemplo, é comum criar emuladores como ferramenta de desenvolvimento de software para um sistema específico. Isso elimina a necessidade de interagir com o hardware manualmente, possibilita depuração mais flexível e até mesmo o desenvolvimento de software sem ter acesso ao hardware em questão.

Emuladores são particularmente importantes como forma de preservação de sistemas históricos. Existem diversos sistemas que não são mais produzidos, muitos dos quais a documentação do projeto original foi perdida ou é incompleta. Eventualmente o hardware original se deteriorará e emulação se torna a única alternativa viável para utilizar o sistema em questão (Dan *et al.*, 2009).

Atualmente, grande parte dos projetos de emulação são projetos open-source, sem fins comerciais e voltados para a preservação de sistemas de videogame antigos. Isso acontece pois vários desses sistemas ainda são bastante populares entre desenvolvedores de software, que tem tanto interesse quanto habilidade técnica para o desenvolvimento desses emuladores.

Este trabalho não tem como foco exclusivo emulação de sistemas de videogame mas, devido a popularidade de tais sistemas como projeto de emulação, será dada mais atenção a tais sistemas.

1.2 Conceitos básicos

Antes de nos aprofundar mais no tema de emulação é necessário introduzir a terminologia, conceitos e definições básicas que são usadas na área. Os primeiros e mais importantes conceitos são os de sistema target e host. Sistema target se refere ao sistema que desejamos emular enquanto sistema host é o sistema onde o emulador será executado.

Ao falar sobre a velocidade de um emulador estamos nos referindo ao quão rápido o sistema host consegue executar o emulador quando comparado ao sistema original. Um requisito considerado fundamental em um emulador é que consiga executar, no mínimo, na mesma velocidade do sistema target.

Um conceito fundamental é a precisão, também chamada de acurácia, de um emulador. Esses termos se referem ao quão bem o emulador imita o comportamento do hardware original, sendo um emulador perfeitamente preciso indistinguível do hardware original. Uma maior precisão sempre é desejada, mas isso tem custos na velocidade de execução e complexidade de desenvolvimento e é considerado extremadamente difícil desenvolver um emulador perfeitamente preciso.

O termo ROM é usado para se referir a qualquer software escrito para a plataforma target. Esse termo é usado pois, em sistemas antigos, era muito comum software ser vendido em cartuchos que continham um chip de memória somente leitura (read-only memory). Quando esse software é parte integral do sistema original e requisito fundamental para seu funcionamento, é conhecido como firmware ou BIOS.

É importante também manter em mente a diferença entre circuitos analógicos e digitais. Circuitos analógicos incluem componentes sensíveis à tensão e corrente e dependem de fenômenos tais como indução magnética, campos elétricos, etc. Circuitos digitais são aqueles implementados por meio de componentes lógicos, onde, em condições ideais, o nível exato de tensão e corrente não é relevante.

Tipicamente circuitos analógicos são mais desafiadores de emular precisamente quando comparados à circuitos digitais. Note que, para fins de emulação, não podemos simplificar circuitos digitais a simples equações lógicas: Detalhes tais como atrasos de transmissão, ativação em bordas altas ou baixas, etc., ainda devem ser implementados em um emulador perfeitamente preciso.

1.3 Arquitetura de um emulador

Considerando que emuladores tem como objetivo imitar o sistema target, é comum que a arquitetura interna do código seja paralela ao projeto do hardware do sistema original. Isso significa que diferentes emuladores podem ter arquiteturas radicalmente diferentes dependendo do sistema target. Apesar disso, existem componentes e técnicas presentes na grande maioria dos sistemas e esse será o foco de estudo (Doruk, 2017).

Normalmente a CPU é o principal componente de um emulador e é em torno dela que todo sistema é construído. A sua principal função é repetir o ciclo de ler um endereço de memória, decodificar a instrução correspondente e executar a função apropriada. Esse

ciclo é conhecido como fetch-decode-execute e as instruções individuais são chamadas de opcodes (operation code). As instruções que a CPU implementa consistem em sua maioria de operações lógico-matemáticas, de controle de fluxo e manipulação de memória.

A conexão entre a CPU e os demais componentes do sistema, incluindo a memória principal, normalmente se dá através dos barramentos de dados e endereço. Isso significa que as mesmas instruções que manipulam a memória também controlam os demais componentes do sistema, dependendo do endereço acessado. Essa técnica é conhecida como memory-mapped I/O.

Emuladores normalmente implementam a CPU e todas suas funções como uma classe discreta. O barramento onde são conectados os componentes tipicamente é implementado por um componente conhecido como MMU (memory mapping unit). Esse componente é responsável pela conexão entre a CPU virtual e os demais componentes do emulador, lendo o endereço que está sendo acessado no barramento e o redirecionando para o componente apropriado. Esse mapeamento entre endereço de barramento e componente é chamado de memory map.

Outro componente extremadamente importante é o sistema de processamento de vídeo, normalmente chamado de GPU ou PPU (pixel processing unit). Seu funcionamento exato varia muito entre cada plataforma mas em sistemas de videogame pode ser considerado análogo de um motor de jogo, porém implementado em hardware. Como processamento de vídeo era um grande desafio em sistemas antigos, é comum esse chip ser mais complexo que a CPU principal e normalmente é o componente mais difícil de emular precisamente. Quando esse chip possui memória reservada para o seu uso ela é conhecida como VRAM (video RAM).

A implementação de demais componentes, tais como chips de áudio, entradas de teclado, portas de comunicação, etc., varia bastante entre diferentes plataformas.

É importante notar que a maior complexidade em um projeto de emulação normalmente não é o comportamento e implementação dos componentes individuais, mas sim modelar precisamente a interação entre eles. Como os sistemas de hardware originais consistem em diversos sistemas trabalhando independentemente, o principal desafio em emuladores precisos é garantir a sincronização exata entre todos seus componentes.

1.4 Técnicas de emulação

Ao estudar a arquitetura de um emulador foram vistos os seus principais componentes e a relação entre eles, mas não a forma que esses componentes são implementados. As técnicas usadas na implementação de emuladores são ainda mais variadas que suas arquiteturas, então só é possível estudar e categorizar essa técnicas em alto nível.

Em projetos de emulação, a primeira e mais importante decisão de projeto é o nível de precisão que se deseja alcançar. A complexidade do desenvolvimento e requisitos técnicos de plataforma host aumentam exponencialmente ao aumentar a precisão, e trocar o nível de precisão durante o desenvolvimento nem sempre é factível. Emuladores de alta precisão são projetos extremadamente complexos e de alto custo computacional, normalmente desenvolvidos em equipe.

As duas principais abordagens ao implementar um emulador são conhecidas como LLE (low-level emulation) e HLE (high-level emulation). LLE consiste em imitar o comportamento da plataforma target em nível de hardware, enquanto HLE implementa o funcionamento do sistema em um nível mais alto, por exemplo implementando a especificação de uma API intermediária sem necessariamente imitar o hardware original. LLE é a técnica mais antiga e geralmente mais precisa, enquanto HLE foi desenvolvida devido à necessidade de emuladores mais rápidos, ao custo de precisão.

A separação entre HLE e LLE não é clara nem definitiva, e dois emuladores que implementam a mesma técnica podem ter diferenças radicais em velocidade e precisão. Similarmente, a complexidade do desenvolvimento usando técnicas LLE ou HLE depende da plataforma e escolha da técnica varia de acordo com a plataforma e as necessidades de performance, precisão e complexidade de implementação. Sistemas mais antigos tem a especificação tão simples que a forma mais óbvia de implementar o emulador é imitando o hardware, de modo que HLE só é usado em sistemas com uma certa complexidade mínima.

Independente da abordagem HLE ou LLE, há diversas formas de implementar o emulador. O mais simples e direto é implementar um interpretador, que lê as instruções da ROM original uma por uma e reproduz o seu comportamento em uma representação virtual do hardware original. Essa abordagem é a mais comum e de menor complexidade, porém a mais lenta.

A medida que se faz necessário maior performance outras técnicas são utilizadas, sendo a mais comum JIT (just-in-time compilation). JIT consiste em analisar o código da ROM em tempo de execução e identificar segmentos que podem ser traduzidos diretamente para código nativo da plataforma host, sem necessidade de interpretar cada opcode individualmente. Essa técnica pode aumentar a velocidade do emulador tremendamente mas é considerada difícil de implementar, especialmente em projetos de emulação.

Capítulo 2

A Plataforma Target

2.1 Objetivos

Estudados os conceitos básicos, arquitetura e principais técnicas de emulação, será dado início ao desenvolvimento de uma implementação própria de um emulador simples. O objetivo dessa implementação é colocar em prática o processo e os conceitos estudados anteriormente, consolidando o aprendizado em um projeto concreto.

O primeiro passo do desenvolvimento é pesquisar e definir a plataforma target que será emulada, respeitando os objetivos do projeto. Será feita uma pesquisa mais aprofundada na história e aspectos técnicos desse sistema e, finalmente, coletados as referências e ferramentas necessárias para a criação de um emulador desse sistema.

Considerando o objetivo educacional deste emulador, o principal foco será em sua arquitetura e decisões de projeto. Assim, buscamos um sistema target simples pois uma plataforma complexa demandaria muita atenção em detalhes de implementação em detrimento de arquitetura e estrutura do projeto.

2.2 Decidindo a plataforma

2.2.1 Pesquisa inicial

Um dos critérios que será usado ao decidir a plataforma target é a qualidade da documentação publicamente disponível, pois não é o objetivo documentar uma plataforma pouco conhecida. Similarmente, como buscamos uma plataforma simples, será dado mais foco à plataformas mais antigas. Existe uma variedade imensa de plataformas bem documentadas, grande parte sendo videogames, mas também existem computadores de uso geral que serão considerados.

Entre os primeiros computadores de uso geral, existem poucas opções adequadas. Embora a variedade de sistemas seja grande não existe tanto interesse em sua emulação e, por conta disso, a documentação desses sistemas é esparsa.

Dois sistemas mais antigos que fogem a essa regra são o ZX Spectrum e o CHIP-8. O

ZX Spectrum era uma implementação simples de um computador baseado na popular CPU Z80, produzido com o objetivo de ser o mais barato possível. Por ter sido extremamente popular e com hardware simples, é um dos poucos computadores pessoais adequado para este projeto.

O CHIP-8, diferente dos demais sistemas discutidos, é uma "máquina virtual". É uma especificação de uma CPU e suas instruções, com o objetivo de facilitar a programação de jogos em sistemas mais simples. O interpretador rodava em apenas 512 bytes de RAM e várias das instruções da CPU virtual eram passadas diretamente para o hardware. Surgiu com o COSMAC VIP e após sua criação continuou sendo usado em outras plataformas por diversos anos. Hoje é bastante popular como sistema target para projetos de emulação.

Os primeiros videogames produzidos em escala foram sistemas de arcade. Vários desses sistemas eram produzidos com circuítos lógicos discretos (não tinham CPU) e a maior parte rodava apenas um jogo. Por serem sistemas sofisticados para a época, tinham vários componentes customizados e partes analógicas desafiadoras de emular.

Assim, a maior parte desses sistemas arcade não é adequada para o este projeto. Uma exceção é o arcade *Space Invaders*, que era baseado no popular microprocessador Intel 8080. Por usar um processador bastante conhecido e documentado, e ter poucos componentes customizados, é um sistema relativamente simples (Cantrell, 2019a). Combinado à sua popularidade, é uma plataforma adequada para este projeto.

Os próximos sistemas a serem considerados são os videogames domésticos. As especificações técnicas de vários deles são aparentemente simples, usando CPUs populares e jogos simples. Porém, essa simplicidade é apenas aparente pois eram extremamente dependentes de minúcias do hardware em que rodavam, e mesmo a mais mínima diferença de timing faz com que não executem corretamente.

Os primeiros sistemas mais simples de emular são aqueles um pouco mais tardios que, devido ao seu hardware mais capaz, não obrigava os programadores a dependerem de detalhes tão específicos do hardware. Esses sistemas mais tardios são contemporâneos dos primeiros portáteis e tem hardware similar, então também serão incluídos nesta pesquisa.

Os principais sistemas nessa categorias são o Master System, Game Gear, Genesis, NES e Game Boy. Entre eles, o Game Boy é considerado o mais adequado como introdução à emulação devido a excelente documentação e seu software não depender de minúcias de timing.

Alguns sistemas mais avançados, tais como SNES, Game Boy Advance e Saturn, também podem ser adequados mas são consideravelmente mais complexos. Após esses sistemas a indústria de videogames cresce bastante e sistemas mais modernos, tais como Nintendo 64, Dreamcast, Playstation, etc., se tornam bastante desafiadores para emular.

2.2.2 Analisando os sistemas

A partir da pesquisa anterior, temos 4 sistemas candidatos para plataforma target do nosso emulador. Eles são: ZX Spectrum, Space Invaders, CHIP-8 e Game Boy. Vamos

elaborar suas especificações e decidir o mais adequado entre eles.

O ZX Spectrum tem como CPU um Zilog Z80A rodando a 3.5MHz, que por sua vez é baseada no Intel 8080. Tem 16KB de ROM e um máximo de 48KB de RAM. É conectado em uma TV e tem uma resolução de 32×24 caracteres em modo texto ou 256×192 em modo gráfico, em até 15 cores. Tem também saída de som de 1-bit e 10 oitavos, controlada diretamente pela CPU (*ZX Spectrum Specifications* 2003).

O Space Invaders é baseado em uma CPU Intel 8080 rodando a 2MHz. Tem 8KB de ROM, 1KB de RAM e 7KB de VRAM. Sua saída de vídeo é um monitor padrão em resolução de 256×224, em preto e branco. Tem saída de som controlada por um chip AY-3-8910 combinado com circuítos analógicos (Cantrell, 2019b).

O Game Boy é baseado em uma CPU LR35902 rodando a 4.19MHz, variante do Z80. Tem 8KB de RAM e 8KB de VRAM, com ROM no cartucho variando entre 32KB e 8MB. Sua saída de vídeo é um LCD de 160×144 pixels em preto e branco, controlado por uma sofisticada unidade de processamento de vídeo baseada em sprites. Tem 4 canais de saída de som, controlado por hardware próprio (*Pan Docs* s.d.).

O CHIP-8 é mais difícil de definir por ser uma "máquina virtual". Sua CPU é própria e em sua implementação original rodava a aproximadamente 500 instruções/segundo. Sua saída de vídeo tem 64×32 pixels em preto e branco, controlada diretamente pela CPU. Tem uma saída de áudio de 1 canal monótono, também controlado pela CPU (Mikolay, 2017).

Para comparar a disponibilidade de documentação, vamos fazer uma pesquisa por emuladores desses sistemas no GitHub. Pressupõe-se que sistemas com mais emuladores implementados tem mais documentação disponível. A partir dessa pesquisa vemos que CHIP-8 é, com uma margem considerável o, sistema mais popular com 2300 projetos. Game Boy fica em segundo lugar com 1700 projetos e tanto ZX Spectrum quanto Space Invaders estão bem distantes com 192 e 151 projetos, respectivamente.

Isso é corroborado pesquisando documentação sobre os sistemas. CHIP-8 foi implementado em diversas plataformas ao longo dos anos e é popular muito como projeto de emulação. Gameboy foi imensamente popular e ainda é o terceiro sistema com mais unidades vendidas, mesmo 32 anos após seu lançamento. Essas características tornam esses dois sistemas extremamente bem estudados e documentados.

ZX Spectrum e Space Invaders não tem a mesma popularidade e embora exista documentação desses sistemas, não é tão completa e variada quanto CHIP-8 e Game Boy. Por esse motivo vamos desconsiderar esses sistemas como target para este projeto.

CHIP-8 e Game Boy ambos satisfazem os requisitos estabelecidos. Comparando ambos vemos que o CHIP-8 é consideravelmente menos complexo por ter uma CPU mais simples, não ter unidade de processamento de vídeo e ter menos componentes auxiliares. Assim, o sistema escolhido será o CHIP-8 devido a sua maior popularidade e devido a ser um sistema mais simples que o Game Boy, que foram os critérios estabelecidos.

2.3 O CHIP-8

CHIP-8 é uma especificação de máquina virtual, criada com objetivo de executar jogos. Isso significa que, diferente da maior parte dos outros sistemas, não é um hardware e sim uma especificação. Apesar disso, essa especificação é bastante similar a forma de funcionamento de sistemas de hardware antigos.

CHIP-8 foi criado em 1976 por Joseph Weisbecker como parte do sistema COSMAC VIP, integrado em uma ROM no computador. Era uma das duas linguagens de programação disponíveis, a outra sendo assembly puro. Devido a complexidade de escrever programas diretamente em assembly, combinado ao interesse em videogames, CHIP-8 foi criado com o propósito de facilitar a criação de jogos (*COSMAC VIP* s.d.).

O design de CHIP-8 foi feito para ser bastante minimalista, utilizando diretamente as funções de hardware quando possível. O interpretador ocupava apenas 512 bytes em memória e implementava 34 instruções diferentes. Essa instruções implementavam operações lógico-matemáticas, controle de fluxo do programa, saída de áudio e vídeo, interação com teclado, entre outros. O design de CHIP-8 limitava os programas a 4KB de RAM, dos quais 512 bytes eram reservados pelo interpretador.

É possível implementar clones de vários jogos populares, tais como *Pong*, *Breakout*, *Tank*, etc., de forma bastante simples em CHIP-8, o que tornou a linguagem bastante popular. Foi implementada em outros sistemas contemporâneos ao COSMAC VIP, tal como o Telmac 1800 e existiam revistas de computação com colunas dedicadas ao CHIP-8, compartilhando jogos e programas feitos para essa plataforma.

Com o aumento da popularidade e melhores sistemas de hardware disponíveis, começaram a surgir extensões do CHIP-8. A maior parte das extensões mantinha a compatibilidade com a especificação original e adicionava recursos tais como cores, maior resolução para gráficos, etc.

Algumas das variantes mais significativas são o CHIP-48, criado em 1990 por Andreas Gustafsson para a linha de calculadoras HP-48. Computadores pessoais contemporâneos já eram muito mais potentes e CHIP-8 não era mais um linguagem relevante para PCs, mas era bastante apropriada para o hardware de calculadoras. Em 1991 Erik Bryntse criou uma variante de CHIP-48 chamada de SUPER-CHIP, abreviado como SCHIP. Era bastante similar ao CHIP-48 mas adicionava um modo gráfico de alta resolução e instruções para rolagem de tela, e rapidamente substituiu CHIP-48.

Outra variante mais moderna é XO-CHIP, criado em 2014 por John Earnest. Foi criada juntamente a um emulador de CHIP-8 chamado de Octo e implementa diversas novas funções tais como sistema de áudio mais complexo, suporte a 64KB de memória, múltiplos planos gráficos, etc (Mikolay, 2019).

Ainda existem competições para programação de jogos em CHIP-8, produzindo softwares surpreendentemente complexos considerando as limitações do sistema. Apesar disso, devido a sua idade e limitações, CHIP-8 e suas variantes são considerados sistemas "mortos" e estudados apenas por seu valor histórico e educacional.

Considerando que para os objetivos deste projeto as funcionalidades que as extensões

fornecem não são relevantes, neste projeto será implementada apenas a especificação original de CHIP-8.

2.4 Especificações técnicas

2.4.1 Overview

A especificação do CHIP-8 define uma CPU com 34 instruções e 16 registradores de uso geral, nomeados v0-vF. Todos esses registradores tem capacidade de 8 bits e o último deles, vF, guarda as flags de operação e é modificado automaticamente por instruções lógico-matemáticas. Nenhum dos demais registradores tem significado especial.

Como o CHIP-8 opera em 4096 bytes de memória e os registradores de uso geral tem apenas 8 bits (portanto podem endereçar apenas 256 bytes), existem outros dois registradores de 16 bits: PC e I. PC aponta para o endereço de memória que contém a próxima instrução, como na maioria das CPUs, e I aponta para um endereço de memória arbitrário, sendo usado em diversas instruções.

As instruções do CHIP-8 tem 2 bytes de comprimento e normalmente são representadas por 4 caracteres hexadecimais, `80A1` por exemplo. Os parâmetros das instruções fazem parte desses bytes, no exemplo `80A1` a instrução é realizar um OR lógico entre os registradores indicados nos bits 4-7 e 8-11, v0 e vA nesse exemplo.

Entre os 4096 bytes de memória disponíveis para um programa em CHIP-8, os primeiros 512 bytes são reservados pelo interpretador e não há nenhum outro endereço de memória com significado especial..Toda a interação com os sistemas de áudio, vídeo, leitura de teclado, etc são feitas através de instruções específicas, o que simplifica bastante os programas pois não há dispositivos mapeados em memória.

2.4.2 Sistema de vídeo

A saída de vídeo de CHIP-8 é em preto e branco, com uma resolução de 64×32 pixels. Existe apenas uma instrução para desenhar na tela, `DXYN`, o que torna o sistema de vídeo bastante simples.

O sistema de vídeo do CHIP-8 é baseado em sprites, que são uma sequência de bytes. Cada um desses bytes representa uma linha de uma figura, onde cada bit representa a cor a ser desenhada, preto (bit em 0) ou branco (bit em 1). Cada sprite pode ter até 15 bytes, limitando os sprites individuais a uma resolução de, no máximo, 8×15 pixels.

Para desenhar um sprite na tela, ele primeiro deve ser carregado em uma localização da memória. Então, usando as instruções apropriadas, modificamos o registrador I para apontar para o sprite que desejamos desenhar e executamos a instrução `DXYN` sendo X e Y as coordenadas onde começar o desenho e N o número de linhas do sprite. Os sprites são desenhados em modo XOR: Para cada pixel a ser desenhado, o resultado na tela é um XOR entre o pixel da sprite e o pixel que já está na tela. Isso traz a propriedade que para apagar um sprite, basta desenhá-lo novamente na mesma posição.

A instrução `DXYN` tem outra função importante: Ao desenhar um sprite, o registrador vF é modificado para 1 caso algum pixel na tela tenha sido modificado de branco para preto e 0 caso contrário. Isso é uma forma bastante eficiente de fazer detecção de colisão em jogos pois acontece automaticamente do ponto de vista do programador.

Existem também instruções para facilitar o desenho de números e sprites básicas embutidas que fazem parte da especificação, mas sua descrição detalhada está além do escopo.

2.4.3 Áudio, timers e teclado

O sistema de áudio do CHIP-8 é extremamente simples e desenhado para gerar feedback sonoro simples, não para efeitos de áudio complexos. Existe apenas uma instrução para controlar o áudio: `FX18`. Essa instrução carrega o conteúdo do registrador vX no registrador de áudio, que é decrementado automaticamente a uma velocidade de 60 hertz. Enquanto o registrador de áudio for maior que zero, um som de timbre de frequências não definidos toca.

O sistema de timer do CHIP-8 é similarmente simples. Existe apenas um timer e é controlado por duas instruções, uma para ler e outra para escrever o seu valor. Quando um valor é escrito nele, é decrementado a uma velocidade de 60 hertz. A intenção é que o timer seja usado para controlar a velocidade nos jogos, mas alguns programadores optam por depender da velocidade de execução das instruções ao invés de usar o timer. Isso gera problemas que vamos ver mais a frente.

O teclado em CHIP-8 consiste em 16 teclas, numeradas de 0 até F. Existem três instruções para interagir com o teclado. A primeira delas pausa a execução do programa até uma tecla ser pressionada, e seu valor é guardado em um registrador. As outras duas funcionam controlando o fluxo do programa: Pulam a execução de uma instrução se uma tecla estiver ou não pressionada.

2.5 Problemas de especificação

Como já dissemos, o CHIP-8 não é um sistema de hardware e sim uma especificação de máquina virtual. Isso gera alguns problemas pois a especificação não é completa e alguns casos são ambíguos, fazendo com que diferentes implementações de CHIP-8 tenham comportamentos sutilmente diferentes.

Um dos maiores problemas é a velocidade de execução da CPU virtual. Como não é definido um número de instruções por segundo que a CPU deve executar, historicamente as implementações de CHIP-8 rodavam na velocidade máxima do hardware e cada implementação tinha uma velocidade de execução diferente. Na teoria isso não deveria ser um problema se todo programa usasse o timer para controlar a sua velocidade de execução mas, na prática, é muito comum dependerem da velocidade de execução do interpretador para o qual foram programados.

A medida em que o hardware se tornou mais potente a velocidade dos interpretadores de CHIP-8 aumentou. Isso fez com que os softwares mais novos dependessem dessa maior

velocidade e os antigos rodassem muito rápido. Como hardware moderno é capaz de executar programas em CHIP-8 extremamente rápido, todos os emuladores incluem algum tipo de limitador de velocidade de execução. O problema disso é que muitas vezes não é possível a velocidade de execução no qual o software foi originalmente programado, então é necessário ajustar manualmente a velocidade de execução para cada programa até "parecer correto". Alguns jogos, principalmente os modernos, utilizam corretamente o timer e não é necessário limitar a velocidade do interpretador.

Com o surgimento de extensões para o CHIP-8 algumas delas modificaram o comportamento de certas instruções, algumas vezes propositalmente e outras não. O interpretador original de CHIP-8 contém alguns bugs e alguns programas não executam corretamente se esses bugs não estiverem implementados.

Esses problemas de especificação faz com que seja comum um programa não executar corretamente pois dependia de alguma sutileza ou bug presente na implementação em que foi programado. A única forma de resolver esse problema é encontrar e listar manualmente os bugs e sutilezas nas implementações mais comuns de CHIP-8 e ajustar manualmente o comportamento do emulador até que execute o software corretamente.

Neste projeto não iremos nos concentrar nessas sutilezas e não é a prioridade emular exatamente todos os bugs e detalhes de alguma implementação específica. Sempre que possível iremos implementar o comportamento que seja compatível com o maior número de programas, embora vão existir alguns casos que não executem corretamente.

2.6 Ferramentas e referências

O primeiro recurso que será usado durante o desenvolvimento é um emulador de CHIP-8 já maduro, que permitirá ver o comportamento do sistema na prática antes de implementá-lo e será usado como base comparativa em questões de acurácia. O emulador de CHIP-8 mais maduro atualmente é chamado *Octo* e implementa diversas funcionalidade úteis além de emular CHIP-8. Tem uma linguagem própria para facilitar o desenvolvimento de programas para CHIP-8, compiladores e descompiladores para essa linguagem, editor de sprites, entre outros. Está disponível em uma interface web em https://johnearnest.github.io/Octo/ (Earnest, 2021b).

A descrição sobre o CHIP-8 na seção acima é suficiente para entender a sua arquitetura e complexidade, porém não é uma descrição técnica completa e portanto não é suficiente para implementar um emulador. Existe bastante documentação sobre CHIP-8 e foram escolhidos dois documentos para serem usados como referência técnica primária durante o desenvolvimento.

O primeiro deles é *Mastering CHIP-8* (Mikolay, 2017), escrito em 2017 por Matthew Mikolay. Esse documento descreve com algum detalhe todos os sistemas descritos na especificação de CHIP-8 e o comportamento de todas suas instruções. Esta será a principal referência usada.

O segundo documento é *Cowgod's Chip-8 Technical Reference v1.0* (Greene, 1997), escrito em 1997 por Thomas P. Greene. Foi criado para ser usado como referência para a implementação de um emulador de CHIP-8 e também descreve todos os sistemas e suas

instruções. Será usada como referência secundária, em caso de dúvidas ou ambiguidade no documento anterior.

Também é necessário uma variedade de software escrito para CHIP-8, que será usado durante o desenvolvimento do emulador para testar e validar o comportamento da implementação. O emulador *Octo* tem uma pequena biblioteca com demos e jogos simples que serão usados no início do desenvolvimento. Uma biblioteca mais completa de jogos antigos foi compilada por Frédéric Devernay obtida em http://devernay.free.fr/hacks/chip8/ (Devernay, s.d.). Para testar os aspectos finais do emulador serão usados jogos mais sofisticados, desenvolvidos a partir de 2014 em competição de programação para CHIP-8 e variantes, obtidos em https://johnearnest.github.io/chip8Archive/ (Earnest, 2021a).

Capítulo 3

Implementação do Emulador

3.1 Objetivo

Definida a plataforma target e coletados todos os recursos necessários para o desenvolvimento, será dado início a implementação do emulador. Considerando os objetivos educacional deste projeto a performance do emulador não é prioridade, e, conforme já discutido, será dada prioridade a compatibilidade de software em detrimento a seguir exatamente a especificação original.

A técnica de emulação usada será LLE pois, devido a simplicidade do CHIP-8, não existem camadas de abstração intermediárias que poderiam ser emuladas. O emulador será do tipo interpretado e durante o desenvolvimento será priorizada uma arquitetura modular e extensível, mesmo quando isso impactar outros aspectos tais como performance final.

Como a maior parte dos projetos de emulação tem velocidade de execução como prioridade, as linguagens de programação mais usadas são de baixo nível, tipicamente C ou C++. Considerando os objetivos deste projeto uma linguagem de alto nível é mais adequada, embora incomum em projetos de emulação. Com essas considerações a linguagem escolhida foi Python, devido a familiaridade com a mesma.

A biblioteca usada para gerenciamento de janelas, desenho dos gráficos e leitura de entrada do usuário foi `Pygame`, que consiste em uma fina camada de abstração no topo da conhecida biblioteca SDL, permitindo utilizá-la em Python.

3.2 Estrutura da implementação

O projeto foi estruturado em módulos e submódulos Python, de forma a separar as principais funções do emulador em componentes modulares. Pode-se observar a estrutura do projeto na seguinte figura:

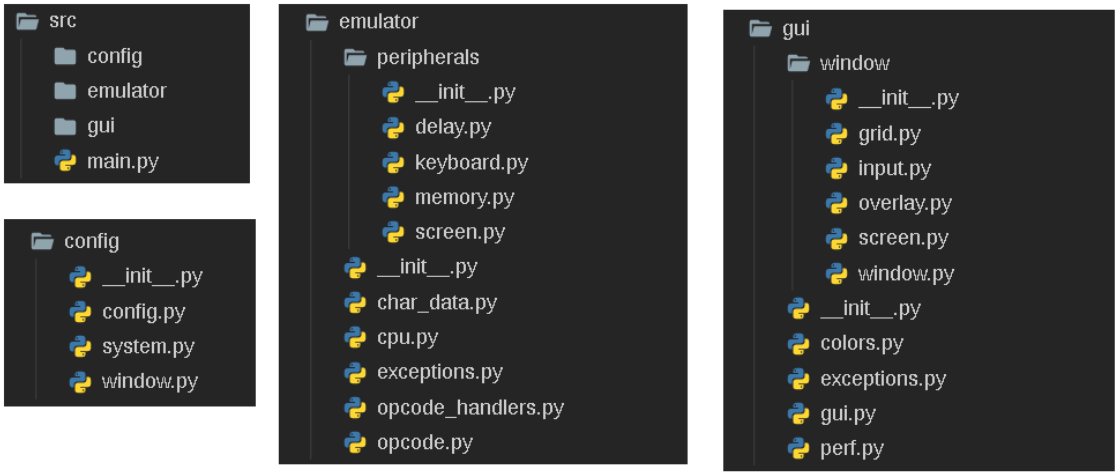

Figura 3.1: *Estrutura geral do projeto.*

O primeiro módulo, `config`, contém apenas as configurações que serão utilizadas ao instanciar os demais componentes do programa.

O segundo módulo, `emulator`, implementa o núcleo de emulação de forma independente dos demais componentes. O emulador de CHIP-8 propriamente dito está contido inteiramente neste módulo, incluindo implementações da memória, CPU, etc. Todos os componentes que o CHIP-8 utiliza para comunicação (tais como tela, teclado e timers) são implementados com interfaces simples para fácil integração com os demais módulos.

O terceiro módulo, `gui`, implementa todas as funções relacionadas a interação com o usuário e sistema operacional. Funções tais como criação e gerenciamento da janela, desenho da saída do emulador, leitura de teclado, etc estão todos contidos neste módulo.

Existe apenas um arquivo, `main.py`, que não faz parte de nenhum módulo. Esse arquivo é responsável por inicializar todos os componentes e implementar o loop principal de emulação.

3.3 Loop principal

O ponto de entrada do programa é o arquivo `main.py`, que é responsável pela inicialização de todos os componentes do emulador e implementa o loop principal do programa. Devido a natureza modular dos componentes este arquivo é relativamente simples e legível, então seções do código serão usadas para explicar, em alto nível, o funcionamento e estrutura do emulador como um todo.

Note que os seguintes fragmentos de código foram editador por clareza. O código original do arquivo `main.py` está no apêndice A.

```
1   import time
2   import random
3   import sys
4   from config import *
5   from emulator import *
6   from gui import *
7
8   global config, now
9   sysconfig = SystemConfig(speed=speed)
10  winconfig = WindowConfig(sysconfig=sysconfig)
11  config = Config(system=sysconfig, window=winconfig, romfile=romfile)
12  now = 0
13  last_update = 0
```

A primeira seção do código simplesmente importa todos os componentes de todos os módulos, deixando-os disponíveis para uso. Qualquer código de inicialização básico é executado automaticamente pelos módulos no momento em que são importados, sem necessidade de inicialização manual.

Em seguida são instanciados objetos com as configurações padrão para todos os componentes. A maior parte dos parâmetros não precisam ser modificados pois tem defaults que funcionam bem para a maioria dos casos. Os único parâmetros modificados são a velocidade de execução, já que a mesma tem que ser ajustada para cada programa individualmente, e a ROM a ser emulada.

```
1   global cpu, mem, delay, screen, keyboard, random
2   mem = Memory(config.system.ramsize)
3   delay = Delay(config.system.delay)
4   screen = Screen(config.system.screen_size)
5   keyboard = Keyboard(range(len(config.keymap)))
6   random = random.Random(config.system.seed)
7   cpu = Cpu(mem, delay, screen, keyboard, random)
```

Em seguida são criados os objetos necessários para o núcleo de emulação funcionar, configurados de acordo com os objetos de configuração instanciados anteriormente.

Os principais componentes de um sistema de CHIP-8 são a memória principal, sua saída de vídeo e entrada via teclado. Também são necessários objetos que possibilitem a implementação de timers e geração de números aleatórios pois essas funções dependem de estado externo e não podem ser implementadas diretamente dentro do núcleo de emulação.

Finalmente, com todos os componentes instanciados conforme as especificações do CHIP-8 a CPU principal é criada, com referências a todos os objetos externos necessários para o seu funcionamento.

```
1    with open(config.romfile, 'rb') as f:
2      f.readinto(cpu.mem[cpu.ip:])
3
4    init(config, screen, keyboard)
```

O próximo passo é carregar o programa que será emulado na memória do CHIP-8. Isso é tão simples quanto abrir um arquivo binário e copiar seu conteúdo, bit-a-bit, no primeiro endereço de memória que a CPU virtual irá executar, apontado pelo registrador PC da CPU virtual. A partir deste ponto o núcleo de emulação está pronto para iniciar a execução.

Por último é inicializada a interface gráfica, o que cria a janela principal do programa. Note que a interface gráfica recebe como parâmetros apenas referências para a saída de vídeo e entrada de teclado do emulador e não depende de nenhum outro estado para o seu funcionamento.

```
1    try:
2      while True:
3        while now-last_update < config.system.speed:
4          now = time.perf_counter()
5        last_update = now
6
7        try:
8          delay.tick(now)
9          cpu.tick()
10       except EmulationError as e:
11         handle_emulation_error(e)
12
13       perf.n_updates += 1
14       perf.update(now)
15       window.update(now)
16
17   except WindowClose:
18     pass
```

Finalmente é iniciado o loop principal de emulação. A limitação de velocidade de execução do emulador é implementada aqui e é bem simples, consistindo apenas em verificar constantemente se já se passou tempo suficiente desde que a última iteração do emulador foi executada.

Ao executar uma iteração de emulador, existem três sistemas que devem ser executados: O núcleo de emulação, a interface de usuário e o sistema responsável por monitorar a performance do emulador.

Primeiramente é executado o núcleo de emulação. O primeiro passo é atualizar o timer do sistema com o horário atual, que irá atualizar o próprio estado conforme as especificações do CHIP-8. Em seguida instruímos a CPU a executar um ciclo de fetch-decode-execute, que faz com que a CPU virtual execute exatamente uma instrução. Caso qualquer erro seja encontrado durante a execução da CPU virtual uma exceção é lançada e

tratada apropriadamente.

Em seguida, é atualizada a janela e toda a interface de usuário. Isso inclui a leitura da saída de vídeo do emulador e desenho da mesma na interface de usuário, leitura do teclado do computador e atualização do teclado virtual do emulador, desenho de estatísticas de performance, etc. O último passo é atualizar os contadores de performance.

Caso a janela seja fechada uma exceção é lançada pelo módulo de interface de usuário e o programa sai silenciosamente. Caso contrário, o loop reinicia e a próxima iteração do emulador é executada.

3.4 Módulo de configuração

O módulo `config` é o mais simples do projeto, tendo como função apenas guardar constantes de configuração de forma estruturada. Essas configurações estão separadas em três tipos: Configurações de interface gráfica, configurações do núcleo de emulação e configurações gerais.

As configurações de interface gráfica definem, por exemplo, o tamanho e layout da janela, frequência de atualização da tela, cores e fontes usadas na interface, etc. Configurações gerais são apenas o mapeamento entre as teclas reais e o teclado do CHIP-8 e a frequência de atualização dos contadores de performance.

As configurações do núcleo de emulação definem o tamanho da memória virtual, resolução do display, tamanho do teclado, frequência de atualização dos timers, entre outros. A maior parte dessas configurações estão definidas na especificação do CHIP-8 e mudar qualquer uma delas tornaria o emulador incompatível com a especificação. O motivo de estarem presentes em um módulo separado é para evitar constantes “mágicas” no resto do código e facilitar uma eventual implementação de extensões de CHIP-8.

3.5 Núcleo de emulação

3.5.1 Periféricos

O módulo `emulator` implementa toda a lógica de emulação de CHIP-8 de forma independente da lógica de desenho da tela, leitura de entrada e qualquer biblioteca ou estado externo. O propósito de estruturar o núcleo de emulação dessa forma é fazer com que seja facilmente adaptável a outros propósitos. Isso pode variar desde outra interface gráfica ou até mesmo um sistema de testes automatizados, sendo possível controlar facilmente o estado do emulador.

Para isto foi necessário separar todas as interfaces do CHIP-8 que sejam responsáveis por entrada e saída, ou tenham qualquer interação com estado externo ao emulador, e as expor através de uma interface simples para que outros módulos possam lidar com o núcleo de emulação facilmente. Isso é feito no submódulo `peripherals` que contém interfaces para a memória, saída de vídeo, teclado, timer e estado aleatório.

As interfaces com o núcleo de emulação são bastante simples. A interface para a saída

de vídeo, por exemplo, pode ser acessada como um array binário indicando quais pixels estão ativos. Similarmente, a memória é como um array de bytes e o teclado como um dicionário indicando quais teclas estão pressionadas. Isso torna bastante simples escrever outros módulos que precisem interagir com o núcleo de emulação, sem expor detalhes do sistema CHIP-8.

3.5.2 CPU

O principal componente do núcleo de emulação é a CPU, pois como o CHIP-8 não possui coprocessadores auxiliares toda a lógica da plataforma target é implementada diretamente na CPU. Nesta implementação o núcleo da CPU foi implementado separadamente da lógica dos opcodes individuais, a fim de tornar o código mais legível.

A classe CPU implementa todas as estruturas da CPU de CHIP-8 descritas nas especificações técnicas tais como os registradores de uso geral, stack, registradores de endereço e programa, etc. Também é responsável por inicializar os periféricos e deixar o sistema virtual pronto para execução.

Sua principal função é `tick`, que completa um ciclo fetch-decode-execute. A implementação desse processo foi feita de uma forma bastante conveniente e fácil de modificar, porém computacionalmente cara: Para cada opcode é feita uma busca em uma lista de expressões regulares até achar uma que corresponda. A função associada a essa expressão regular, que corresponde a implementação do opcode dado, é então executada.

Como em CHIP-8 os parâmetros para as instruções são codificadas nos bits intermediários dos próprios opcodes, foi implementada uma classe `Opcode` que facilita a leitura de nibbles intermediários como inteiros padrão.

3.5.3 Implementação dos opcodes

A implementação da lógica de todos os opcodes está em `opcode_handlers.py`, juntamente a uma tabela que associa uma expressão regular, que corresponde a cada opcode, a função que o implementa.

A maior parte dos opcodes tem função bastante simples e sua implementação é imediata, como por exemplo os opcodes responsáveis por operações lógico-matemáticas. Alguns são bem mais complexos, como por exemplo o opcode responsável pelo desenho de sprites e detecção de colisões.

Apesar de simples é necessário bastante atenção na implementação dos opcodes a fim de replicar exatamente a especificação de CHIP-8 e seu comportamento em casos extremos, tais como overflow ou underflow. Durante o desenvolvimento aconteceram diversas instâncias em que, por exemplo, algum resultado intermediário não foi devidamente verificado por overflow e a instrução falhava apenas em alguns casos específicos, criando bugs difíceis de localizar.

Algumas instruções não tem seu comportamento totalmente definido na documentação e formas diferentes de implementá-las podem causar bugs bastante estranhos em softwares que dependam dessas minúcias. Por exemplo, ao usar operações lógico-matemáticas onde

o registrador de destino é vF, que é responsável pelas flags, não é definido se vF deve guardar o resultado da operação ou as flags. Isso não causa problemas com a maior parte dos jogos para CHIP-8 exceto os mais modernos, que utilizam todos os recursos possíveis da máquina e dependem de tais detalhes.

3.6 Interface gráfica

O módulo `gui` implementa toda a interface de usuário e é o único módulo que utiliza as bibliotecas Pygame e Numpy, a únicas dependências externas ao projeto.

Este módulo implementa também um monitor de performance, usado para acompanhar a velocidade de execução do emulador. Sua implementação é bastante simples e consiste de contadores que monitoram a taxa de atualização da tela e o número de instruções por segundo que o emulador está executando. Embora simples sua funcionalidade foi essencial para acompanhar o impacto de diferentes funcionalidades na velocidade de emulação e fazer ajustes de acordo.

O submódulo `window` contém a maior parte da lógica de interface de usuário e é responsável por definir o layout dos componentes na janela, leitura de teclado, desenho da saída de vídeo, entre outros. De forma geral sua função é fazer a ponte entre o sistema real, acessado a partir da biblioteca Pygame, e o núcleo de emulação, utilizando as interfaces definidas no submódulo `peripherals`.

A função de desenho de tela em particular, embora curta, foi a mais intricada de implementar. A primeira implementação dessa função simplesmente iterava a saída de vídeo do emulador pixel-por-pixel, desenhando cada pixel individualmente. Essa implementação era funcional e capaz de rodar a 60 frames por segundo (velocidade de atualização de tela do CHIP-8), mas bastante ineficiente e não escalaria para sistemas com resolução de tela maior.

Para garantir que o módulo `gui` seja adequado para reutilização futura em outros projetos de emulação, foram feitas diversas tentativas de implementar o desenho de tela de forma mais eficiente. A solução final envolve a conversão das superfícies de display em arrays da conhecida biblioteca Numpy, o que permite sua manipulação direto por funções C que são bem mais eficientes. Essa solução, além de aproximadamente 12x mais rápida que a solução anterior, escala para sistemas com resolução de vídeo muito maior de forma eficiente.

Capítulo 4

Conclusão

Durante este trabalho foram estudados os principais componentes, decisões de projeto, arquitetura e técnicas envolvidas em um projeto de emulação. Isso possibilitou a implementação de um emulador de uma forma organizada, analisando de forma metódica o processo de escolha de plataforma target, coleta de documentação, considerações de design inicial e escopo de projeto.

O processo de implementação de um emulador de CHIP-8 forneceu uma oportunidade de implementar na prática os conceitos estudados e observar o fluxo de trabalho, problemas de documentação e compatibilidade, impacto das decisões de design e ferramentas escolhidas.

A implementação final do emulador foi satisfatória, tendo boa compatibilidade com software escrito para CHIP-8 e tem performance suficiente para executar todas as ROMs testadas satisfatoriamente. Isso fornece uma boa base para o início de outros projetos de emulação mais complexos, tanto em relação aos diversos aprendizados durante as etapas de pesquisa e implementação, quanto da estrutura e componentes do código que podem ser reaproveitados em projetos futuros.

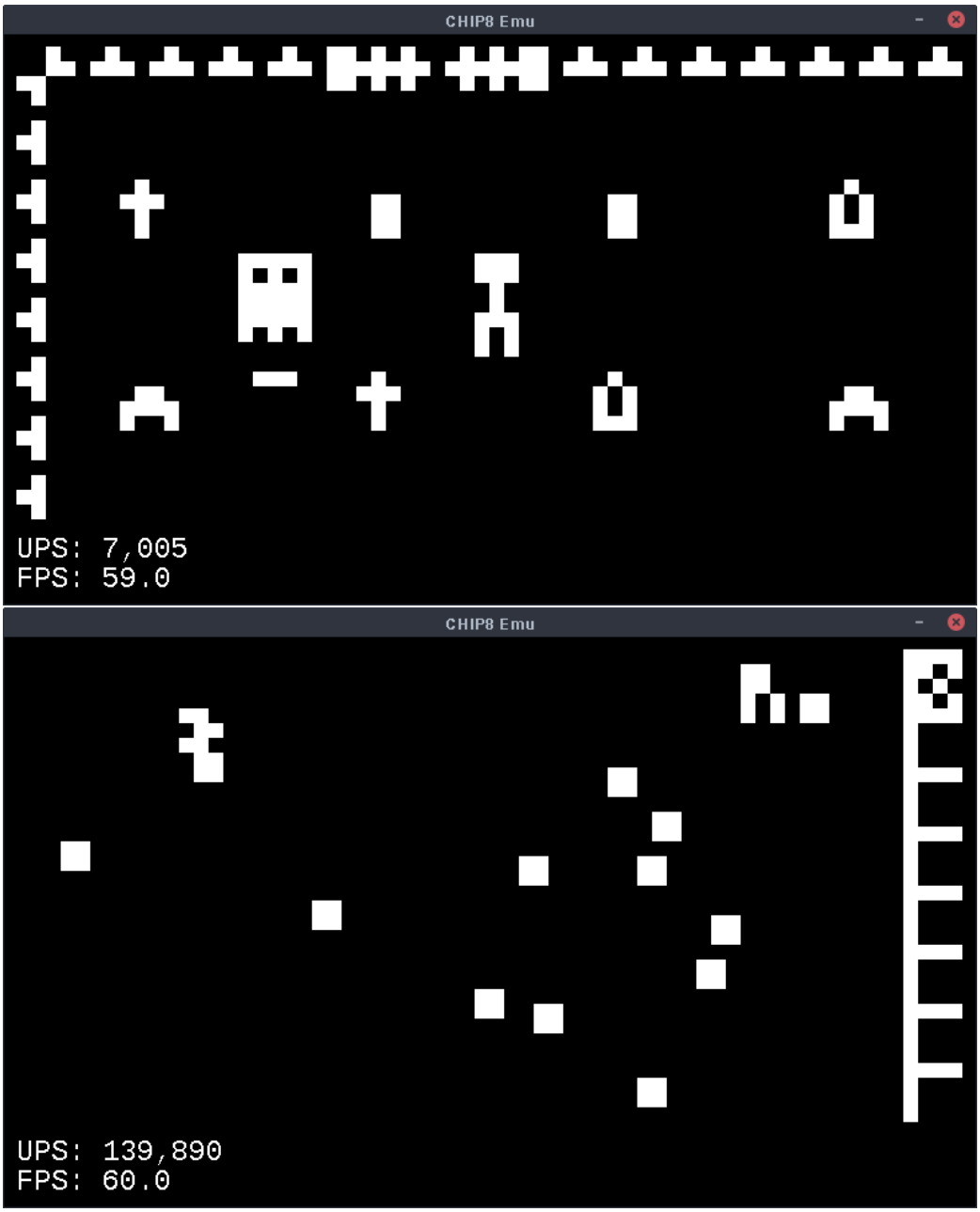

Figura 4.1: *Emulador funcionando com duas ROMs diferentes.*

Apêndice A

Código-Fonte

```
import time
import random
import sys
from config import *
from emulator import *
from gui import *


def tick_emulator():
  try:
    delay.tick(now)
    cpu.tick()
  except EmulationError as e:
    handle_emulation_error(e)


def handle_emulation_error(e):
  print(f'{type(e).__name__}: {e}')
  print(cpu)
  cmd = input()
  perf.skip = True

  if cmd == "q":
    raise WindowClose

  elif cmd == "c":
    cpu.ip += cpu.opc.size

  elif cmd == "r":
    raise e

  else:
    raise e


def main(romfile, speed):
  global config, now
  sysconfig = SystemConfig(speed=speed)
```

```
39      winconfig = WindowConfig(sysconfig=sysconfig)
40      config = Config(system=sysconfig, window=winconfig, romfile=romfile)
41      now = 0
42      last_update = 0
43
44      global cpu, mem, delay, screen, keyboard, random
45      mem = Memory(config.system.ramsize)
46      delay = Delay(config.system.delay)
47      screen = Screen(config.system.screen_size)
48      keyboard = Keyboard(range(len(config.keymap)))
49      random = random.Random(config.system.seed)
50      cpu = Cpu(mem, delay, screen, keyboard, random)
51
52      with open(config.romfile, 'rb') as f:
53        f.readinto(cpu.mem[cpu.ip:])
54
55      init(config, screen, keyboard)
56
57      try:
58        while True:
59          while now-last_update < config.system.speed:
60            now = time.perf_counter()
61          last_update = now
62          perf.n_updates += 1
63          tick_emulator()
64          perf.update(now)
65          window.update(now)
66
67      except WindowClose:
68        print(f"Avg UPS: {perf.get_average():,.0f}")
69
70      finally:
71        gui_quit()
72
73
74    if __name__ == "__main__":
75      if len(sys.argv) < 2:
76        print("Usage: chip8_emu <rom file> <emulation speed>")
77        sys.exit(1)
78      else:
79        romfile = sys.argv[1]
80        speed = 1/(1000000)
81        if len(sys.argv) >= 3:
82          speed = 1/int(sys.argv[2])
83
84      main(romfile, speed)
```

Referências

[Cantrell 2019a] Christopher Cantrell. *Space Invaders*. 2019. URL: https://www.computerarcheology.com/Arcade/SpaceInvaders/ (citado na pg. 6).

[Cantrell 2019b] Christopher Cantrell. *Space Invaders Hardware*. 2019. URL: http://computerarcheology.com/Arcade/SpaceInvaders/Hardware.html (citado na pg. 7).

[*COSMAC VIP* s.d.] *COSMAC VIP*. URL: https://oldcomputermuseum.com/cosmac_vip.html (citado na pg. 8).

[Dan *et al.* 2009] Pinchbeck Dan *et al.* "Emulation as a strategy for the preservation of games: the keep project". Em: *DiGRA &#3909 - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Brunel University, set. de 2009. ISBN: ISSN 2342-9666. URL: http://www.digra.org/wp-content/uploads/digital-library/09287.31196.pdf (citado na pg. 1).

[Devernay s.d.] Frédéric Devernay. *SVision-8: CHIP-8 and SCHIP emulator*. URL: http://devernay.free.fr/hacks/chip8/ (citado na pg. 12).

[Doruk 2017] Alpay Doruk. "The design of an emulator". Em: out. de 2017, pgs. 1064–1067. DOI: 10.1109/UBMK.2017.8093468 (citado na pg. 2).

[Earnest 2021a] John Earnest. *CHIP-8 Archive*. 2021. URL: https://johnearnest.github.io/chip8Archive/ (citado na pg. 12).

[Earnest 2021b] John Earnest. *Octo*. 2021. URL: https://johnearnest.github.io/Octo/ (citado na pg. 11).

[Greene 1997] Thomas P. Greene. *Chip-8 Technical Reference*. 1997. URL: http://devernay.free.fr/hacks/chip8/C8TECH10.HTM (citado na pg. 11).

[Mikolay 2017] Matthew Mikolay. *Mastering CHIP-8*. 2017. URL: http://mattmik.com/files/chip8/mastering/chip8.html (citado nas pgs. 7, 11).

[Mikolay 2019] Matthew Mikolay. *CHIP-8 Extensions Reference*. 2019. URL: https://github.com/mattmikolay/chip-8/wiki/CHIP%E2%80%908-Extensions-Reference (citado na pg. 8).

[*Pan Docs* s.d.] *Pan Docs*. URL: https://gbdev.io/pandocs/ (citado na pg. 7).

[*ZX Spectrum Specifications* 2003] *ZX Spectrum Specifications*. 2003. URL: https://rk.nvg.ntnu.no/sinclair/computers/zxspectrum/spec_specifications.htm (citado na pg. 7).