CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE GUARULHOS

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

RICARDO ALVES GOMES

TIAGO DA SILVA SIQUEIRA

DESENVOLVIMENTO DE UM PROTÓTIPO DE EMULADOR DE CHIP-8 COMO FORMA DE ESTUDO DE PRÁTICAS DE PROGRAMAÇÃO ANTIGAS

GUARULHOS

2022

RICARDO ALVES GOMES
TIAGO DA SILVA SIQUEIRA

DESENVOLVIMENTO DE UM PROTÓTIPO DE EMULADOR DE CHIP-8 COMO FORMA DE ESTUDO DE PRÁTICAS DE PROGRAMAÇÃO ANTIGAS

Trabalho de Conclusão de Curso apresentado ao Curso de Análise de Desenvolvimento de Sistemas como requisito parcial para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Me. Rodrigo Vieira Campos

GUARULHOS
2022

RICARDO ALVES GOMES
TIAGO DA SILVA SIQUEIRA

DESENVOLVIMENTO DE UM PROTÓTIPO DE EMULADOR DE CHIP-8 COMO FORMA DE ESTUDO DE PRÁTICAS DE PROGRAMAÇÃO ANTIGAS

Trabalho de Conclusão de Curso apresentado ao Curso de Análise e Desenvolvimento de Sistemas como requisito parcial para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Banca Examinadora

Orientador: _____
Me. Rodrigo Vieira Campos
Fatec Guarulhos

Banca**:** _____
Me. Milton Francisco de Brito
Fatec Guarulhos

Banca: _____
Me. Renato Alves Ferreira
Fatec Guarulhos

Guarulhos, 04/12/2022

RESUMO

GOMES, Ricardo Alves; SIQUEIRA, Tiago da Silva. **DESENVOLVIMENTO DE UM PROTÓTIPO DE EMULADOR DE CHIP-8 COMO FORMA DE ESTUDO DE PRÁTICAS DE PROGRAMAÇÃO ANTIGAS.** 2022. 18p. Trabalho de Conclusão de Curso – Faculdade de Tecnologia de Guarulhos, Guarulhos.

O presente trabalho visou desenvolver um emulador de CHIP-8 para a *web*, como forma de estudo de técnicas de programação de jogos antigos que fizeram parte da história dos *videogames* e agora não podem ser adquiridos legalmente. O objetivo do projeto é disponibilizar um emulador acessível como alternativa à pirataria e com ferramentas que possam ser usadas no estudo de técnicas antigas de desenvolvimento de jogos. Utilizou-se técnicas de engenharia reversa para o desenvolvimento, compilando as mais antigas formas de manipulação de memória e operações *bitwise* com as ferramentas mais modernas de desenvolvimento *web*.

Palavras-chave: emulação, conservação, engenharia reversa.

ABSTRACT

Describes the process of development of a web-based CHIP-8 emulator as a means for the study of programming techniques of old games that have been part of the history of videogames and now cannot be legally acquired. This project's objective is to provide an accessible emulator as an alternative to piracy and with tools capable of being used in the study of old game development techniques. Reverse engineering techniques were employed for the developme compiling the older forms of memory manipulation and bitwise operations with more modern tools of web development.

Keywords: emulation, conservation, reverse engineering.

LISTA DE ABREVIATURAS

ABNT Associação Brasileira de Normas Técnicas

NES Nintendo Entertainment System

NPM Node Package Manager

ROMs Read Only Memory

VS Code Visual Studio Code

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

O desenvolvimento de emuladores de programas antigos é um tema controverso, que não pode ser tratado sem abordar questões como direitos autorais e pirataria. Apesar dos temas acima serem abordados neste artigo, seu foco é descrever o processo de criação de um emulador de CHIP-8 com ferramentas de desenvolvimento *web*.

CHIP-8 é uma linguagem de programação interpretada, ou seja, executada diretamente pelo sistema, usada principalmente no processador dos microcomputadores Telmac 1800 e COSMAC VIP, criada por volta dos anos 1970 por Joseph Weisbecker, com o intuito de permitir que jogos de *videogame* fossem mais facilmente programados em computadores. A linguagem não é mais utilizada em processadores modernos, mas até hoje é usada para o estudo da emulação.

O projeto nasce com o desejo de se aprender como funciona a lógica executada dentro de um processador, durante uma aula da disciplina de arquitetura e organização de computadores, do 1º semestre do curso superior de Análise e Desenvolvimento de Sistemas. Primeiramente tentando reproduzir a estrutura de um *videogame* moderno, objetivo que depois foi simplificado à sua forma mais rudimentar, escolhendo-se o CHIP-8 por ser uma das linguagens mais simples para processadores, e já ter tido seu código disponibilizado para outros desenvolvedores por seus criadores.

Muitos emuladores de CHIP-8 já existem na internet, mas a maioria em linguagens compiladas, como C e C++. Então, visando contribuir de alguma forma com a comunidade de desenvolvimento de emuladores, enquanto ao mesmo tempo contribui-se para a acessibilidade e preservação dos jogos estudados, decidiu-se por disponibilizar este emulador *online*, utilizando-se linguagens e ferramentas modernas, como Javascript e React para componentização e interação com o usuário.

Segundo Vanius Zapalowski (2011), a escolha de uma linguagem interpretada acarreta severos problemas de tempo de execução de código, que serão abordados ao longo do projeto. Mas a discussão destes problemas inerentes às linguagens de alto nível, como o Javascript, assim como a exploração dos processos de interpretação de instruções e ponteiros de memória característicos de linguagens de baixo nível, como C e Assembly, são o que torna este projeto distinto de outros e digno de ser explorado.

2. EMBASAMENTO TEÓRICO

2.1 Metodologia de desenvolvimento

A metodologia utilizada para o desenvolvimento do programa foi a de análise bibliográfica, aliada à engenharia reversa, analisando as partes do programa e pesquisando a solução desenvolvida por outros desenvolvedores para lidar com o problema. O processo de desenvolvimento do emulador é baseado nos métodos da engenharia reversa, com o objetivo de descobrir as partes constituintes do sistema, ou seja, usa-se o processador do Telmac 1800 e os códigos de operação (*opcodes*), com o objetivo de desenvolver um interpretador que, quando alimentado, produza os mesmos resultados do sistema original.

Como instruções de entrada, foram utilizadas as *ROMs* de dois jogos clássicos que podiam ser jogados no Telmac: PONG e *Breakout*. Segundo Pavlo Liasota (2021) *ROMs*, ou *Read Only Memory*, são registros de dados que só podem ser lidos, não escritos, e no contexto de jogos e emulação são representações digitais dos "cartuchos", ou os jogos físicos, que no caso do processador estudado são arquivos binários de até 3800 *bytes* de dados. Nenhum jogo é disponibilizado com o emulador, por motivos descritos na seção sobre emulação e pirataria, portanto, fica sob responsabilidade do usuário do emulador a aquisição de uma *ROM* própria.

2.2 Componentes do CHIP-8

A comunidade de desenvolvimento de emuladores é muito ativa, sendo os emuladores de CHIP-8 considerados os mais simples e, portanto, recomendados para os programadores iniciantes. Isto significa que existem diversos guias de criação para esta linguagem e até projetos prontos nas principais linguagens de programação atuais.

Apesar de ter evitado os emuladores já existentes, o sistema descrito no artigo baseia-se em dois dos principais guias de desenvolvimento para o CHIP-8, sendo eles um artigo científico da Universidade de Colúmbia (KLING et al, 2016) e o blog do desenvolvedor norueguês Tobias V. Langhoff (2020). E numa segunda etapa do desenvolvimento, implementação de soluções desenvolvidas no artigo de Eric Grandt (2020). Segundo estes autores, o desenvolvimento de um emulador deve ser dividido nas seguintes partes: entrada de dados e *disassembler*, controle de memória e ponteiro, variáveis, teclado, *display* e *timers*.

O *disassembler* pode ser considerado o elemento principal do emulador, é ele que vai interpretar qual instrução está sendo enviada ao processador e decidir como ela será

interpretada, no Apêndice A pode-se observar como foi desenvolvido o disassembler deste projeto, que interpreta cada instrução e a envia para o método que irá realizar a função desejada.

A entrada de dados, conforme citado previamente, é composta de arquivos binários de até 3800 *bytes* (cada *byte* é composto por 8 *bits*). Dentro do processador os *bytes* são separados em duas partes, formando valores hexadecimais. Por exemplo: uma entrada de 2 *bytes* seria dividida em quatro valores hexadecimais. Os valores hexadecimais são então agrupados de quatro em quatro, formando uma instrução, que deve ser interpretada por uma estrutura de *switch* chamada de *disassembler* e executada pelo processador. Um exemplo de instrução seria 6A0C, onde 6 representa qual operação deve ser feita com a instrução, A representa a variável a ser alterada e 0C o valor que a referida variável terá.

A memória é composta por 4KB de dados, ou 16x16x16, sendo os primeiros 512 *bytes* reservados para a fonte tipográfica do emulador e outras configurações, de 0x0 até 0x200 (hexadecimal) e o resto, de 0x201 até 0xFFF, pode ser ocupado pelas instruções do cartucho. A fonte tipográfica do emulador tem a função de desenhar na tela valores como pontuação do jogador e percentual de compleição, não tem lugar determinado na memória, mas deve poder ser encontrada caso a instrução FX29 seja chamada (X sendo substituído por algum valor hexadecimal). Na figura 1 pode se verificar como é feita a divisão da memória no emulador.

Figura 1- Espaços de memória do CHIP-8

Fonte: Universidade de Columbia, 2016 e adaptado pelos autores.

Ainda segundo o artigo de Kling, o ponteiro da memória, chamado de 'I' na literatura especializada, é usado para guardar a posição de uma instrução que será repetida, ou de um padrão que será usado para desenhar na tela do computador, este valor pode ser manipulado por instruções como ANNN (define o valor do ponteiro) e FX1E (adiciona um valor ao ponteiro).

Os registradores, ou as variáveis que uma *ROM* é capaz de manipular durante sua execução são 16 posições fixas da memória, que são capazes de armazenar dois valores hexadecimais. Os registradores são chamados VX (de V0 até VF), sendo que VF não é alterado diretamente pelo cartucho, mas funciona como um controle de operações, armazenando restos de operações e verificações de estado. Como por exemplo, mudar o valor de VF de 0 para 1 caso a tela seja alterada de certa forma, com o objetivo de registrar a alteração. As instruções 6XNN (mudar o valor de um registrador), 7XNN (adicionar um valor ao registrador) e 8XYZ (trocar um valor por um outro já registrado), são geralmente as responsáveis por manipular as variáveis.

Muitas das instruções que manipulam os registradores, fazem-no por meio de operações *bit* a *bit*, que são operações realizadas não nos valores hexadecimais de um número, mas em seu valor binário, um certo operador lógico (*AND*, *OR*, *XOR*) é aplicado em cada *bit* de dois valores hexadecimais distintos e seus resultados são agrupados, formando um novo valor, ou no caso das operações de deslocamento de *bits*, os valores binários de um número são movidos para a direita ou esquerda e o restante é armazenado na variável VF.

O teclado é hexadecimal, assim como os outros aspectos do CHIP-8, sendo composto de 16 teclas, de 0 a F, que são usadas de formas diferentes por jogos diferentes, múltiplas teclas podem ser acionadas ao mesmo tempo.

O *display* é composto de 2048 *pixels* (64 por 32 *pixels*), e a principal instrução que lida com o desenho de elementos na tela usa linhas de 8 pixels de largura por X de altura, chamadas de sprites, que são ou não desenhadas de acordo com uma operação *bitwise XOR*. Ou seja: um *pixel* só é redesenhado se houver alteração em seu estado. A instrução DXYN é a principal responsável por desenhar na tela do computador.

Os *timers* são dois, um *delay* de execução, que pode ser usado para esperar um certo tempo e um *timer* sonoro, que toca por X tempo o som principal do *hardware*. Estes *timers* são definidos com o valor de um dos registradores, de 0x0 a 0xFF e se decrementam até chegar em 0 a uma velocidade de 60Hz, independente da velocidade dos ciclos da *CPU*.

2.3 Emulação e Pirataria

A emulação de sistemas de jogos é uma ferramenta usada principalmente para que jogadores sejam capazes de ler jogos exclusivos de certos consoles de *videogame* em seus computadores, geralmente de forma ilegal, mas também é usada pelas próprias empresas detentoras dos direitos de jogos, seja para fazer com que seus sistemas mais recentes sejam compatíveis com jogos de gerações passadas, seja para relançar algum produto em uma plataforma diferente para os computadores atuais.

Vender ou distribuir produtos digitais sem a expressa autorização dos proprietários de uma marca ou produto é crime e pode ser punido com três meses a um ano de reclusão e/ou multa, de acordo com o Art. 184 da Lei nº 10.695, de 1º de julho de 2003 (BRASIL, 2003). A lei brasileira (e da maior parte dos países) só considera crime a distribuição de jogos, não a criação de emuladores que rodem estes jogos. Destaca-se neste sentido uma decisão da nona corte de apelos dos Estados Unidos entre *Sony Entertainment Network* e *Connectix Corp.* (2000), na decisão, o juiz decidiu contra a *Sony*, empresa proprietária de grande parte dos jogos digitais atuais, citando que a emulação e a engenharia reversa de jogos eram ferramentas importantes para incentivar o estudo tecnológico e a livre competição do mercado.

Certas empresas reconhecem a importância da emulação e disponibilizam seus próprios sistemas de emulação, como é o caso da Sega, que disponibiliza o *Sega Mega Drive Classics*, um emulador próprio de seus jogos de Mega Drive (console de *videogames* lançado pela empresa em 1988). Além de preservar o legado da Sega, o sistema de emulação adiciona algumas tecnologias atuais, como realidade virtual e modos de multijogador aos jogos antigos. O emulador tem avaliações muito positivas e, segundo matéria do site *Game Developer* (KERR, 2016), estimulou a venda de aproximadamente 350.000 jogos antigos no mês seguinte ao seu lançamento.

No lado oposto da Sega está a Nintendo, que tem uma posição muito firme quanto aos emuladores, sempre considerando-os uma grande ameaça para suas propriedades intelectuais. A Nintendo ficou conhecida por processar qualquer pessoa ou entidade que tente usar suas marcas ou produtos com fins lucrativos ou até acadêmicos. Por outro lado, a própria empresa utilizou um programa que simula jogos antigos para o lançamento da coletânea "*Super Mario 3D All-Stars*".

Em um artigo para o site *Bitniks* (2021), a jornalista Giovanna Breve reuniu a opinião de alguns advogados especializados em direito intelectual sobre a questão. Todos concordam que a Nintendo tem pleno direito de processar quem lucra indevidamente com sua marca, e da

forma que a lei está posta, tem o dever de tomar ação legal em todos os casos para não abrir precedentes legais. Essa necessidade de constante vigilância pode acabar prejudicando aqueles cujo objetivo na emulação é puramente educacional ou recreativo.

3. DESENVOLVIMENTO DA TEMÁTICA

3.1 Desenvolvimento da aplicação *web*

Para alcançar o objetivo de disponibilizar o emulador como uma aplicação *web*, alguns serviços se demostraram fundamentais. As ferramentas utilizadas para o desenvolvimento deste trabalho foram escolhidas com base em dois fatores principais: sua popularidade e a quantidade de documentação disponível na internet sobre processos similares, principalmente em fóruns como o *Stack Overflow*.

O primeiro serviço necessário é o de hospedagem de sites com um domínio, onde o site possa existir e ser atualizado conforme as necessidades intrínsecas ao projeto. Felizmente, diversas plataformas de desenvolvimento contínuo existem na internet, estas são plataformas de *deployment*, ou implantação de código, que compilam e disponibilizam o site em um servidor a partir de um sistema de controle de versões. Para versionamento do código foi usado o Github, por ser o serviço mais popular atualmente, para hospedagem optou-se pela plataforma Vercel, que proporciona os serviços descritos acima de forma gratuita. Para a execução da aplicação foram usados o *Microsoft Edge* e o *Google Chrome,* ambos navegadores de *internet,* sendo o primeiro oferecido pela Microsoft de forma conjunta com o Windows 10 e o segundo oferecido pela Google.

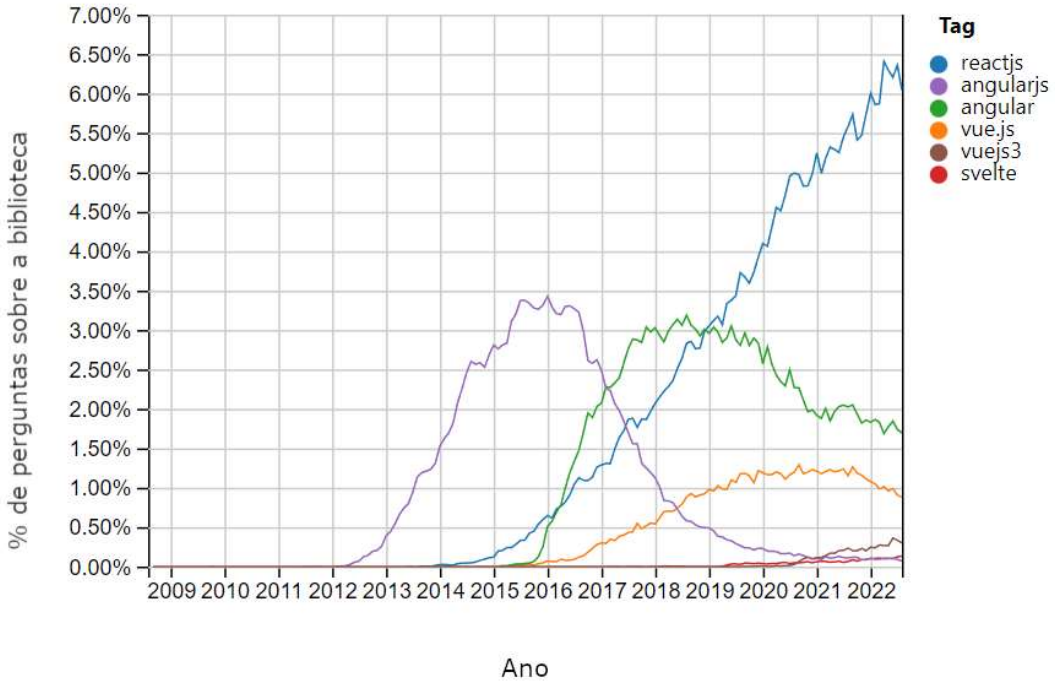
No desenvolvimento do código e leitura dos arquivos binários, a plataforma utilizada foi o *Visual Studio Code*, editor de códigos fonte leve, que possui uma vasta biblioteca de extensões que podem ser utilizadas para melhor visualizar arquivos binários. A extensão utilizada para este fim foi a *"hexdump for VSCode"*, disponível gratuitamente na loja do *VS Code*.

Para compilação da aplicação e testes do sistema também foi necessária uma plataforma que possibilitasse a interpretação do código Javascript, servidor local de testes e roteamento de páginas. Nesta área não existem muitas opções viáveis de escolha, apenas o Node.js e Deno.js, que apesar de terem o mesmo criador, são bastante diferentes, como por exemplo a dependência do Node.js de seu gerenciador de pacotes, o NPM (*Node Package Manager*) e uma maior segurança do Deno no seu gerenciamento de pacotes. Optou-se pelo uso do Node, por ter maior suporte e literatura especializada, apesar de apresentar alguns problemas de segurança, com

módulos menos confiáveis, tais falhas não devem ser um problema neste projeto, já que ele não utiliza nenhum módulo com falhas conhecidas de segurança.

Quanto ao desenvolvimento da aplicação *web* em si, havia um grande leque de escolhas possíveis, mas as três principais são React, Vue e Angular, plataformas de desenvolvimento de *Single Page Applications*, aplicações de página única capazes de interação direta e dinâmica com o usuário. Decidiu-se pelo React, por ser a biblioteca mais usada atualmente, com longa literatura de apoio para facilitar o desenvolvimento. Segundo levantamento do site *Stack Overflow*, principal fórum de desenvolvimento mundial, usando como parâmetro a quantidade de perguntas feitas sobre diversas tecnologias e linguagens de desenvolvimento de sistemas, pode-se dizer que a biblioteca mais usada no desenvolvimento para internet é o React. Na Figura 2, é possível conferir a evolução da quantidade de perguntas feitas sobre cada uma das principais bibliotecas de programação e sua comparação.

Figura 2- Perguntas no Stack Overflow sobre as principais bibliotecas web.

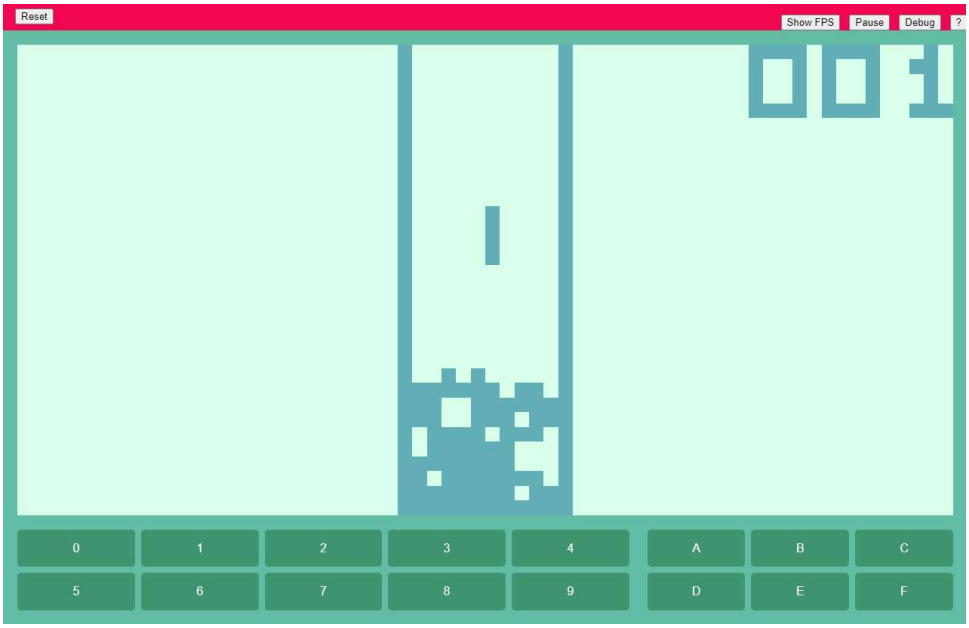


Fonte: Stack Overflow, 2022.

O emulador foi idealizado como uma aplicação responsiva de página única, seu espaço é utilizado completamente pelos elementos funcionais do sistema (tela, teclado e espaço para entrada do cartucho). Também foi elaborado um espaço sobre a tela, chamado aqui de *menu* de *debug*, ou depuração, que fornecerá ao usuário a descrição do estado atual da aplicação, para auxiliar no estudo CHIP-8, esta tela mostrará muitos dos elementos descritos anteriormente como a instrução sendo executada atualmente, valor dos registradores e uma descrição em

detalhes caso o usuário passe o mouse sobre algum dos itens. Para que possa utilizar o *menu* de *debug*, o usuário terá alguns controles sobre o emulador que não estavam presentes no sistema original, como o poder de pausar o jogo a qualquer instante e um comando que avança as instruções uma por vez, para que se possa analisar o estado atual do jogo com mais calma. A figura 3 demonstra como os elementos do emulador se apresentação na aplicação.

Figura 3- Aplicação executando o jogo Tetris.



Fonte: Desenvolvido pelos autores.

Com o objetivo de instruir o usuário no uso do site, um *menu* de instruções contendo informações de navegação é exibido quando o endereço é acessado e permanece em foco até que o usuário interaja com a aplicação. Este *menu* pode ser acessado novamente através do teclado ou por um botão na parte superior da tela.

3.2 Refatoração do código

Quanto à linguagem utilizada para o desenvolvimento do emulador, utilizou-se um projeto análogo para aferir os resultados, segundo Carvalho e Junior (2016), cujo projeto foi a criação de um emulador de *Nintendo Entertainment System* (NES) baseado na linguagem Python, também interpretada, o emulador resultante, apesar de funcional, era mais lento do que o esperado. Sendo assim, já era esperado que o emulador não performasse em uma velocidade tão alta quanto desejável, mas desde o desenvolvimento do projeto estudado, novas ferramentas de depuração e compilação de código foram desenvolvidas, o que torna interessante o retorno ao problema.

Quando se terminou o desenvolvimento, o emulador executava uma média de 10 instruções por segundo, onde o original executaria 20. Pôde-se notar, utilizando ferramentas de depuração de código fornecidas pelo próprio navegador *web* (*Google Chrome*), que o principal responsável pela lentidão era a atualização dos componentes através do React. Quando uma variável é atualizada com um novo valor, o React chama de forma assíncrona um "*hook*", ou gancho de estado, para dizer ao componente que ele deve ser redesenhado. Este processo é demorado e faz com que instruções como desenhar na tela levem mais tempo do que o desejado para serem executadas.

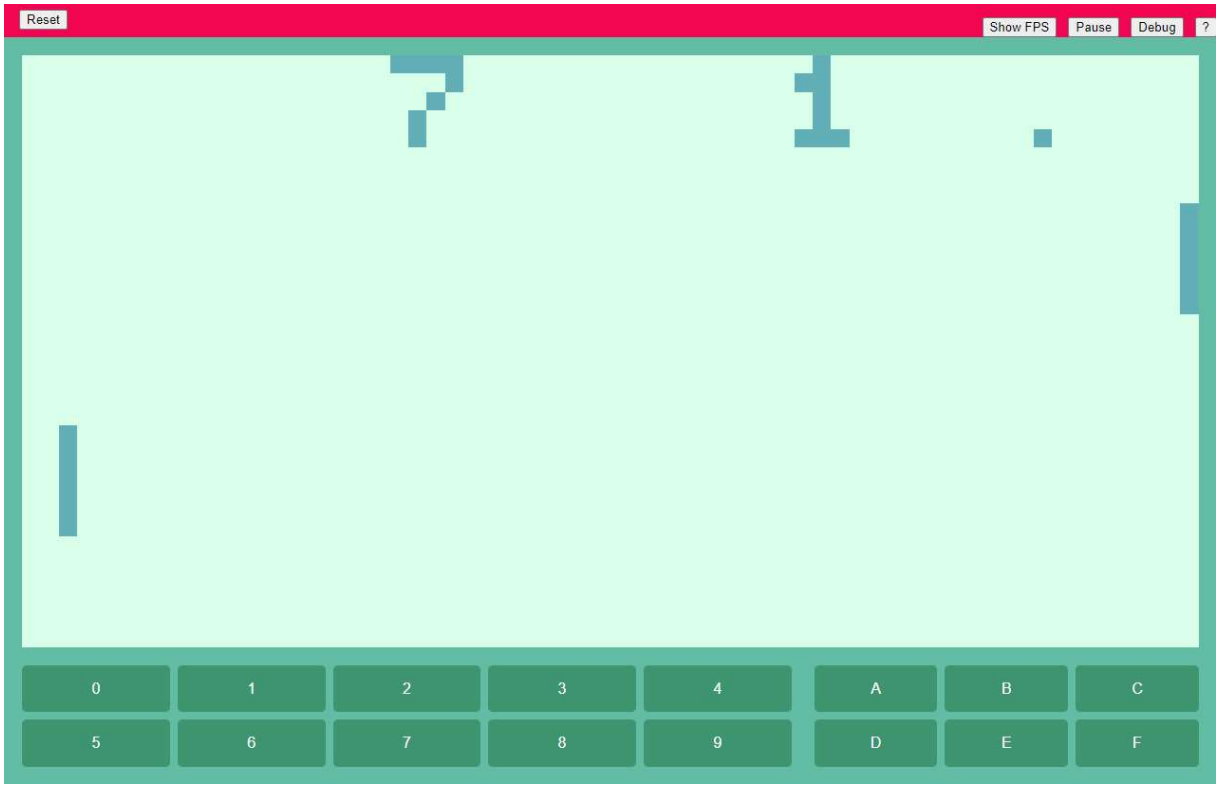
Para contornar a lentidão, foram agrupadas estas atualizações, para que fossem executadas o menor número possível de vezes. Seguiu-se também algumas dicas do guia de Eric Grandt, (2020) para diminuir o tempo gasto com cálculos utilizando um tipo de dado chamado UInt8, que só armazena valores de 0 a 255, eliminando a necessidade de checar se os valores ultrapassam a capacidade do emulador.

Depois destas medidas de refatoração de código e da eliminação de componentes que não estavam sendo utilizados e variáveis supérfluas, o emulador passou a executar as instruções em uma velocidade aceitável, entre 18 e 22 instruções por segundo, dependendo do dispositivo que está executando.

4. RESULTADOS E DISCUSSÃO

O emulador resultante deste projeto é capaz de reproduzir fielmente qualquer jogo do computador Telmac 1800. Apesar de ter sido desenvolvido com base em apenas dois jogos, graças a reprodução da lógica interna do processador, qualquer instrução recebida por ele apresenta resultado fiel ao sistema real. Na figura 4 é possível conferir o estado atual do emulador através de sua interface atual. Como se pode perceber o emulador está executando um port do jogo *Pong (1972)*.

Figura 4- Interface atual do emulador.



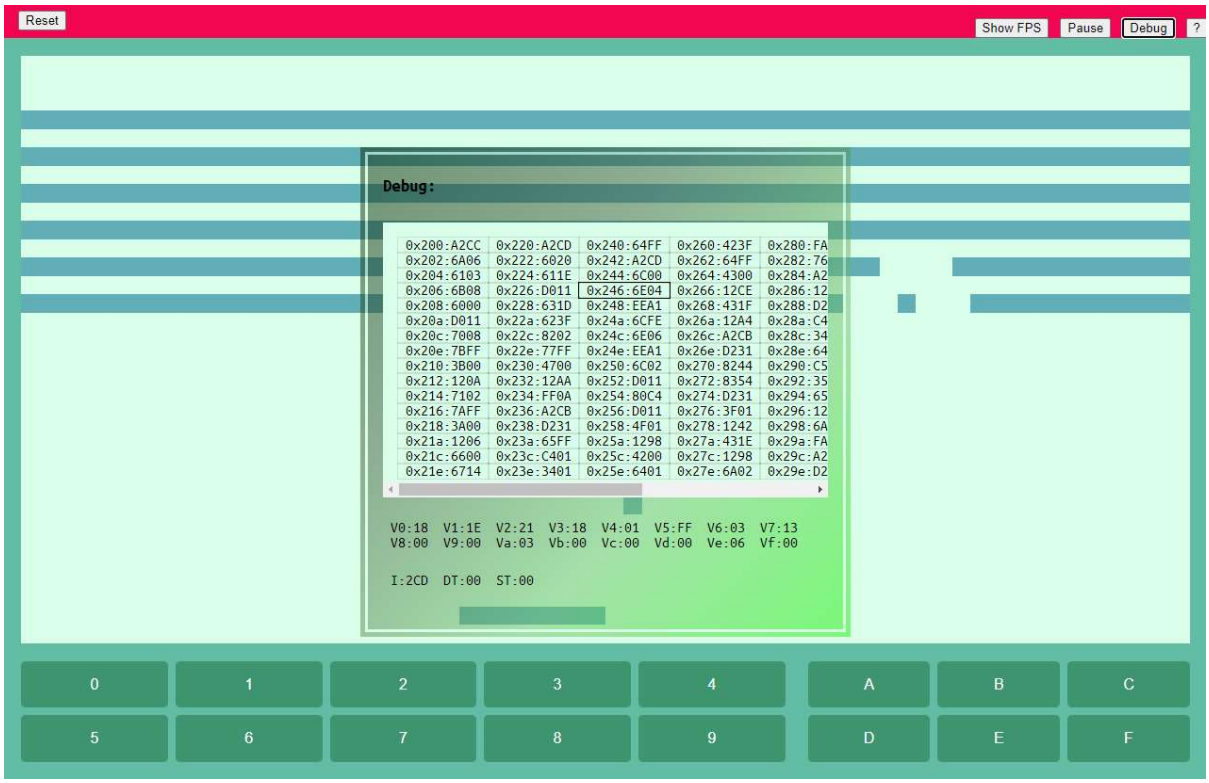
Fonte: Desenvolvido pelos autores.

Existem, porém diferenças entre o original e o emulador: o som executado pelo emulador é de natureza digital, então não pode ser considerado uma reprodução fiel, assim como o *delay*, que idealmente teria decrescimento variável entre as instruções do sistema, mas foi decidido por usar um valor fixo para fins de simplificação do projeto.

A aplicação está hospedada no serviço Vercel (https://chip-oito.vercel.app/) e o projeto pode ser atualizado a qualquer momento utilizando o repositório do projeto no Github. A interface dá preferência à tela do emulador e possui um teclado virtual que pode ser utilizado em tablets ou celulares. A tela é responsiva e o leitor de arquivos permite entrada de qualquer arquivo binário.

Apesar de o emulador ser executado em uma velocidade comparável ao sistema original, ainda pode ser melhorada em sua consistência. No caso de um computador *desktop* atual, com bom poder de processamento, o emulador executa com a mesma velocidade de um sistema real, mas no caso de um celular ou computador com baixo processamento, seja um *tablet* ou *desktop* mais antigo, a velocidade ainda fica um pouco menor que o original, em uma média de 20 *frames* por segundo (FPS). O número ideal de *frames* por segundo para que um jogo pareça fluido seria por volta de 30 FPS.

Figura 5- *Menu* de depuração ativo.



Fonte: Desenvolvido pelos autores.

Quanto ao objetivo de auxiliar no estudo de técnicas de programação, também podem-se fazer melhorias nesta direção, o *menu* de *debug* desenvolvido exibe todas as instruções, mas não deixa clara a ordem em que elas serão executadas, isto pode tirar um pouco da perspectiva do usuário sobre o todo do jogo sendo estudado, algumas melhorias simples nesta tela podem tornar mais fácil o estudo das *ROMs* e utilização do aplicativo. O *menu* de instruções iniciais também pode ser incorporado à tela, por exemplo, desenhando diretamente na tela do emulador as instruções quando o site é acessado pela primeira vez. Na figura 5 apresenta-se a tela de *debug*, onde o usuário pode examinar o estado atual do jogo executado.

Um exemplo de aprendizado com técnicas antigas de desenvolvimento seria o modo como jogos checavam colisão entre as partes, em programas atuais de desenvolvimento de

jogos, os controles de colisão ocupam parte considerável do processamento, enquanto o jogo PONG simplesmente checava se tentou redesenhar a bola por cima de um pixel já preenchido nas áreas onde as raquetes podem estar. O uso deste tipo de técnicas pode diminuir muito o uso de memória de jogos atuais.

5. CONSIDERAÇÕES FINAIS

Este trabalho pretendeu demonstrar a importância da emulação no estudo e preservação de práticas de programação em jogos virtuais e consoles antigos, a fim de usar a emulação para fins de preservação de jogos abandonados, a partir do desenvolvimento de um emulador. Além disso, havia uma premissa voltada para apontar técnicas de desenvolvimento utilizadas pela indústria de jogos digitais e de expor práticas que ajudem a entender melhor os processos de desenvolvimento de programas empregados pela indústria dos jogos.

Ao longo de um período de aproximadamente sete meses, foi desenvolvido um protótipo de emulador de CHIP-8, com base em uma grande documentação já existente. Foram utilizadas a linguagem de programação Javascript, a biblioteca React.js e a plataforma de hospedagem de código-fonte Github juntamente com o serviço de implantação de código da Vercel. Tais tecnologias se atualizam frequentemente, ou seja, para que este projeto se mantenha atualizado, ele deve acompanhar qualquer grande atualização nestes serviços.

O principal desafio do trabalho foi o estudo do hardware a nível de arquitetura de sistema e como simular este sistema para executar jogos e aplicações desenvolvidos por terceiros. Ao longo do projeto um problema recorrente foi a lentidão apresentada na execução contínua das instruções, advinda da natureza interpretada da linguagem Javascript e da característica assíncrona dos componentes da biblioteca React.js. Apesar de esta situação ter sido parcialmente resolvida, o emulador ainda apresenta oscilação grande de desempenho em equipamentos diferentes, o ideal seria que o projeto pudesse ser executado do mesmo modo nos dispositivos a fim de aumentar sua acessibilidade.

A lentidão pôde ser estudada mais profundamente com ferramentas de análise de performance e superada com refatoração do código e melhor entendimento das ferramentas utilizadas, o que acabou por enriquecer o projeto.

Sendo assim, espera-se ter atraído novos interesses sobre técnicas fundamentais de desenvolvimento, como manipulação de variáveis e operações bit a bit, demonstrado também que é completamente possível o desenvolvimento de um emulador para preservação histórica

de jogos digitais, sem necessidade do uso de pirataria, usando ferramentas modernas de desenvolvimento, como ganchos de estado e componentização. Ferramentas estas que tornaram possível publicar o emulador como um site da *web* ao invés de um programa executável, o que isso auxilia na acessibilidade do emulador, principalmente entre pessoas que não possuem um conhecimento técnico do funcionamento do projeto.

Embora o resultado não tenha um desempenho competitivo com outros emuladores de CHIP-8, há espaço para evolução do projeto, em pesquisas futuras, pode-se tentar contornar o redesenho constante dos componentes através do React, otimizando as atualizações e acelerando a execução do emulador. Acreditamos que o projeto servirá para o fomento do desenvolvimento de emuladores, preservação de código e principalmente para o estudo de técnicas de desenvolvimento de baixo nível para solucionar problemas no desenvolvimento com linguagens de alto nível. O repositório do projeto no Github pode ser acessado em https://github.com/sleiph/CHIP-OITO.

REFERÊNCIAS

BRASIL. Lei n° 10.695, de 1° de julho de 2003. **Sobre direitos do autor e proprietário de direitos autorais.** Disponível em: https://planalto.gov.br/ccivil_03/leis/2003/l10.695.htm Acesso em: 23 out. 2022.

BREVE, G. **Pirataria ou preservação? Como a Nintendo radicaliza o debate sobre propriedade intelectual**, 2021. Disponível em: https://www.bitniks.com.br/pirataria-ou-preservacao-como-a-nintendo-radicaliza-o-debate-sobre-propriedade-intelectual. Acesso em: 14 set. 2022.

CARVALHO, José Rafael F. P. de; JUNIOR, Brivaldo A. S. **Emulador de Nintendo implementado em Python**, 2016**.** Disponível em: https://facom.ufms.br/~brivaldo/pdfs/tcc/pynes.pdf. Acesso em: 2 nov. 2022.

CHIKOFSKY, E.J.; CROSS, J.H. ***Reverse engineering and design recovery: a taxonomy.*** IEEE Software, 7 ed., p.13-17, 1990.

ESTADOS UNIDOS. Nona corte de apelos. *Sony Computer Entertainment, Inc. v. Connectix Corp.* CANBY, J., 10 de fevereiro de 2000. Disponível em: http://digital-law-online.info/cases/53PQ2D1705.htm. Acesso em: 23 out. 2022.

GRANDT, Eric. ***How to Create Your Very Own Chip-8 Emulator***, 2020. Disponível em: https://www.freecodecamp.org/news/creating-your-very-own-chip-8-emulator/. Acesso em: 2 out. 2022.

KERR, C. ***Sega mod hub launch sparks 350,000 in classic game sales***, 2016. Disponível em: https://www.gamedeveloper.com/business/sega-mod-hub-launch-sparks-350-000-in-classic-game-sales. Acesso em: 14 nov. 2021.

KLING, Ashley et al. ***CHIP-8 Design Specification.*** 4840 ed. *Columbia University - Embedded Systems*, 2016.

LANGHOFF, T. V. ***Guide to making a CHIP-8 emulator***, 2020. Disponível em: https://tobiasvl.github.io/blog/write-a-chip-8-emulator/#fx07-fx15-and-fx18-timers. Acesso em: 28 set. 2022.

LIASOTA, Pavlo. **CHIP-8 (and Emulators) In Plain English**, 2021. Disponível em: https://cgmathprog.home.blog/2021/05/20/chip-8-and-emulator-overview/. Acesso em: 5 dez. 2022.

Stack Overflow Trends. Disponível em: https://insights.stackoverflow.com/trends?tags=reactjs%2Cangularjs%2Cangular %2Cvue.js. Acesso em: 30 out. 2022.

VIPER. ***VIPER for RCA VIP Owners.*** *Intelligent Machines Journal,* California, 1 ed., p.09, 1978.

ZAPALOWSKI, V. **Análise quantitativa e comparativa de linguagens de programação**, 2011 Disponível em: https://lume.ufrgs.br/bitstream/handle/ 10183/31036/000782127.pdf. Acesso em: 30 out. 2022.

APÊNDICE A - Código-Fonte do interpretador de instruções

```
1     import Instrucoes from './Instrucoes';
2
3     function Disassembler(indice, inst1, inst2) {
4     let op =inst1 >> 4,   // primeira posicao da instrucao
5          x = inst1 & 0xf,  // segunda posicao
6          y = inst2 >> 4,   // terceira posicao
7          n = inst2 & 0xf;  // quarta posicao
8
9     switch(op) {
10         case 0:
11              if (y===14) {
12                   if (n===0) {    // OOEO
13                        return indice + Instrucoes.LimpaTela();
14                   } else if (n===14) {  // 00EE
15                        return Instrucoes.Retorna();
16                   }
17              }
18              else // 0NNN
19                   return indice + Instrucoes.Vazio();23
20         case 1: // 1NNN
21         return Instrucoes.StrHex(((inst1&0xf)<<8) + inst2);
22         case 2: // 2NNN
23         return Instrucoes.StrRot(((inst1&0xf)<<8)+inst2,indice);
24         case 3: // 3XNN
25         return indice + Instrucoes.skipXNNTrue(x, inst2);
26         case 4: // 4XNN
27         return indice + Instrucoes.skiptXNNFalse(x, inst2);
28         case 5: // 5XY0
29         return indice + Instrucoes.skipXYTrue(x, y);
30         case 6: // 6XNN
31         return indice + Instrucoes.setRegistrar(x, inst2);
32         case 7: // 7XNN
33         return indice + Instrucoes.setAdd(x, inst2);
34         case 8:
35              switch(n) {
36                   case 0:
37                   return indice + Instrucoes.setIgual(x, y);
38                   case 1:
39                   return indice + Instrucoes.setOR(x, y);
40                   case 2:
41                   return indice + Instrucoes.setAND(x, y);
42                   case 3:
43                   return indice + Instrucoes.setXOR(x, y);
44                   case 4:
45                   return indice + Instrucoes.setAddop(x, y);
46                   case 5:
47                   return indice + Instrucoes.setSubop(x, y);
48                   case 6:
49                   return indice + Instrucoes.setRightShift(x);
50                   case 7:
51                   return indice + Instrucoes.setRestop(x, y);
52                   case 14:
```

```
53                  return indice + Instrucoes.setLeftShift(x);
54                  default:
55                  throw new Error("8 e alguma coisa...");
56             }
57        case 9: // 9XY0
58             return indice + Instrucoes.skipXYFalse(x, y);
59        case 10: // ANNN
60    return indice+Instrucoes.setIndico(((inst1&0xf)<<8)+inst2);
61        case 11: // BNNN
62        return Instrucoes.pulaPraNNN(((inst1&0xf)<<8) + inst2);
63        case 12: // CXNN
64        return indice + Instrucoes.setRandom(x, inst2);
65        case 13: // DXYN
66        return indice + Instrucoes.Desenha(x, y, n);
67        case 14:
68             if (n===14) // EX9e
69                  return indice + Instrucoes.isApertando(x);
70             else if (n===1) //EXa1
71                  return indice + Instrucoes.isNotApertando(x);
72             throw new Error("E alguma coisa...");
73        case 15:
74             switch(n) {
75                  case 3: // fX33
76                  return indice + Instrucoes.setBCD(x);
77                  case 5:
78                       if (y === 1) // FX15
79                       return indice + Instrucoes.setTimer(x);
80                       else if (y === 5) // FX55
81                       return indice + Instrucoes.save(x);
82                       else if (y === 6) // FX65
83                       return indice + Instrucoes.load(x);
84        throw new Error("F alguma coisa alguma coisa 5...");
85                  case 7: // FX07
86                  return indice + Instrucoes.registraTimer(x);
87                  case 8: // FX18
88                  return indice + Instrucoes.setSound(x);
89                  case 9: //FX29
90                  return indice + Instrucoes.registraIndice(x);
91                  case 10: //FX0a
92                  return indice + Instrucoes.esperarTecla(x);
93                  case 14: // FX1E
94                  return indice + Instrucoes.setAddIndice(x);
95                  default:
96                  throw new Error("F total");
97             }
98             default:
99        throw new Error("instrução "+indice+" não entendida");
100       }
101  }
102  export default Disassembler;
```