Desenvolvimento do Jogo Helltaker como Meio para Aprendizagem de Linguagem Assembly no Contexto da Introdução aos Sistemas Computacionais

Felippe D. M. Carvalho Henrique de O. Ramos

Universidade de Brasília, Departamento de Ciências da Computação, Brasil

***Abstract*—Este artigo apresenta a descrição da implementação de um jogo eletrônico, utilizando linguagem assembly da ISA RISC-V. São apresentadas as ferramentas necessárias para realização do projeto, sendo o RARS a principal delas, seguido do detalhamento das metodologias e algoritmos usados para criação de interfaces gráficas, de input de teclado e de áudio, bem como para todas as mecânicas de apresentação, animação, colisões, penalizações que fornecem a jogabilidade do *game*. Também são apresentados e discutidos os principais desafios enfrentados na tarefa, levando a conclusão de que o objetivo de aprendizado e sedimentação de conhecimentos foram bem alcançados com a realização deste projeto.**

***Keywords*-helltaker, risc-v, assembly, jogos, sistemas computacionaishelltaker, risc-v, assembly, jogos, sistemas computacionais,**

I. Introdução

O desenvolvimento de jogos têm sido um dos grandes tópicos a atrair o interesse daqueles que desejam iniciar sua jornada de aprendizado em computação. Vários dos problemas elementares em programação podem ser abordados e internalizados através da estruturação de um jogo eletrônico. Quando se alia esse projeto a uma linguagem de programação de baixo nível, é possível ir além do mero entendimento de algoritmos clássicos e alcançar uma maior compreensão das bases de funcionamento de um sistema computacional.

É neste contexto que surge um projeto no âmbito da matéria Introdução aos Sistemas Computacionais do curso de Ciência da Computação na Universidade de Brasília. O objetivo é fazer os alunos que estão entrando no curso de graduação pensarem e desenvolverem um programa utilizando linguagem assembly para a ISA modular RISC-V [1]. Esta arquitetura, ainda recém-criada por assim dizer, foi pensada para possibilitar uma forma mais didática de ensino, mas vem cada vez mais se firmando como uma das grandes promessas para aplicações comerciais, dadas as suas vantagens perante outras arquiteturas mais popularmente utilizadas.

Figure 1. Helltaker - jogo eletrônico indie independente.

Neste projeto, foi desenvolvido uma versão do jogo Helltaker (Figura 1), desenvolvido independentemente por Łukasz Piskorz, para um processador de 32 bits, utilizando o módulo RV32I. Esse é o módulo base da RISC-V, contendo as instruções primordiais para manipulação de números inteiros. A ideia era fazer a programação do zero, tendo como base os conhecimentos adquiridos em sala da aula e o simulador RARS.

Será apresentada a metodologia e as ferramentas utilizadas durante o projeto na Seção II, seguida dos principais resultados obtidos de dos desafios enfrentados ao longo da construção do jogo na Seção III. Ao final, uma breve discussão das conclusões e aprimoramntos para trabalhos futuros.

II. Metodologia

Nesta seção são apresentados, em detalhes, as metodologias, os métodos e os algoritmos utilizados no desenvolvimento do jogo, bem como as ferramentas e programas que serviram de suporte para implementação.

A. Ferramentas

A principal ferramenta utilizada no desenvolvimento foi o programa RARS (RISC-V Assembler, Simulator, and Runtime)[2] em sua versão 1.5 Custom 1. Como o próprio nome indica, ele permite a simulação completa de um processador de arquitetura RISC-V. Ele possui dois ambientes principais: um de inserção e edição do código, o outro de execução do código, permitindo a avaliação precisa dos dados armazenados nos registradores e na memória, além de um debugger durante o processo de assemble do código, facilitando a depuração do código. Além dessas, para permitir a entrada e saída de dados, também foram utilizadas duas ferramentas complementares já imbutidas no RARS: o Bitmap Display e o Keyboard and Display MMIO Simulator. Com o Bitmap Display é possível escrever pixels em regiões específicas da memória que serão apresentadas em um tela de 320x240. E com o Keyboard MMIO, o programa implementa a interface com o teclado, permitindo a leitura de teclas pressionadas para controles do jogo.

Para criação das artes e sprites utilizados no jogo, foi utilizado o GIMP (GNU Image Manipulation Program), um software livre de edição de imagens. Os sprites foram retirados de alguns sites [3] [4] ou extraídos diretamente do jogo. Nele foram editadas, redimensionadas, e ajustadas a imagens e posteriormente exportadas em formato .bmp. Para conversão dessas imagens bitmap para um formato compatível com a implementação no RARS, foram utilizadas ferramentas fornecidas pelo professor: um script em Python e um executável bmp2isc.exe. Estas entregavam arquivos .data com informações do tamanho da imagem e das cores de cada um dos pixels.

Também foram utilizados o sistema Git e o repositório online GitHub para auxiliar na gestão de versões do programa e no compartilhamento de arquivos entre o desenvolvedores. E o Google Docs para o momento inicial de registro de ideias e decisões estratégicas, e confecção do roteiro a ser utilizado na narrativa do jogo.

B. Métodos

A seguir são apresentadas as formas com que as principais funcionalidades, como interfaces gráfica, input de comandos do teclado, interface de áudio, e sistema de detecção de colisões do jogo foram implementadas.

:

1) Interface Gráfica: **O primeiro desenvolvimento do programa foi uma rotina que permitisse "desenhar" no Bitmap Display. O método utilizado foi baseado no fornecido por exemplos de aula. A lógica básica por trás de passo consiste em ler a cor de cada pixel do arquivo .data, como mencionado na Subseção II-A e escrevê-lo no endereço de memória correspondente ao vídeo. No entanto, este método inicial apenas era válido para imagens que correspondiam ao tamanho total da tela, portanto era necessário modificá-lo para que fosse possível localizar o início da imagem onde quer fosse necessário. Para tanto, a rotina primeiramente realiza o cálculo do endereço de memória para o início da imagem, através da multiplicação da largura de tela pela linha que se deseja começar, adicionada da coluna desejada. Este valor é por fim adicionado ao endereço inicial global da memória VGA.**

Para melhor desempenho do programa, a rotina de desenho de imagens no display realizava o loop em steps de múltiplos pixels. Ou seja, carregava quatro pixels da imagem e os escrevia na memória VGA para cada iteração. Portanto, para garantir que não haja desalinhamento com os endereços de memória, uma subrotina verificava se o pixel inicial da imagem estava alinhado. Caso não estivesse, ela alterava para o endereçamento alinhado. Além disso, para garantir que os personagens e elementos fossem desenhos sobre o mapa e o plano de fundo, a rotina também detectava se o pixel continha a cor magenta, como mostrado na Imagem . Caso verdadeiro, ela não colocava esse pixel no display, continuando o loop com o próximo.

Figure 2. Sprite de um dos elementos do jogo, o esqueleto, com a cor magenta de fundo, indicando quais pixel não devem ser desenhados.

:

2) Interface do Teclado: **Dois tipos de interação com o teclado foram implementadas utilizando o Keyboard MMIO, ambas utilizado o método polling. A diferença entre elas, foi que uma se comporta de forma blocante, ou seja, para a execução do programa até que uma tecla seja pressionada. Este foi utilizada principalmente no menu inicial e nas cenas de diálogo ao longo do jogo. A outra apenas checava se alguma tecla estava pressionada, caso não seguia com a execução do código. Este segunda forma foi mais amplamente utilizada dentro do loop de cada fase, quando o interesse era a movimentação**

do personagem ou a colisão com algum inimigo ou obstáculo.

:

3) Interface de Áudio: **A implementação de músicas e efeitos sonoros no jogo, foi feita utilizando System Calls de MIDI que já são integradas no RARS. O algoritmo consiste em especificar a duração, o volume e o instrumento de cada nota, e em seguida utilizar uma das ecalls disponíveis. Assim, como na interface com o teclado, existiam duas opções: uma que esperava a nota ser executada para prosseguir com a execução e outra que não o fazia. Efeitos sonoros foram utilizados para momentos em que o personagem chutava um obstáculo ou inimigo, para quando um inimigo morria e para o momento em que o personagem sofre o dano de espinhos.**

:

4) Movimentação e Posicionamento: **Para posicionamento do personagem e dos elementos, foi concebida uma malha cartesiana para o mapa de cada fase. Desse modo, o labirinto foi divido em células de 20x20 que eram indicadas por índices numéricos como mostrado na Figura 3 abaixo. Desse modo, todas**

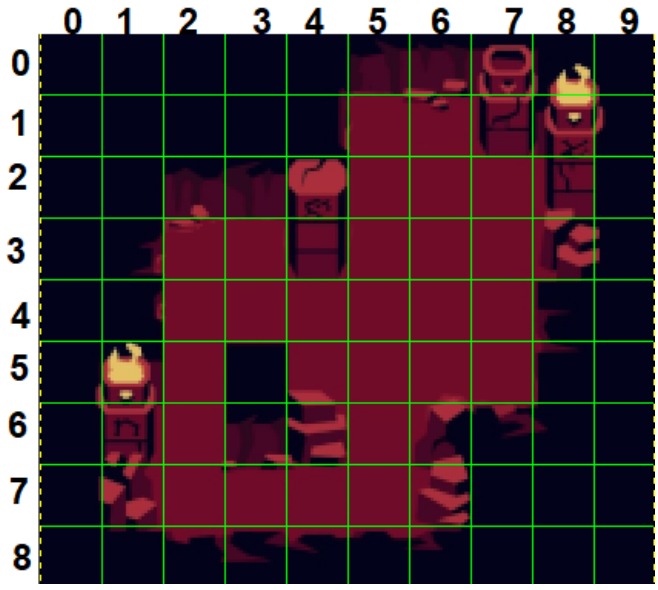

Figure 3. Esquema da malha cartesiana utilizada como guia de referência posicional em cada fase.

as movimentações do personagem e inicialização dos inimigos e elemntos do mapa foram baseadas em um rotina que calculava o pixel inicial que cada imagem deveria iniciar. Este cálculo, semelhante ao que foi feito na rotina de desenhar imagens, consistem em somar a coluna desejada ao produto entre linha desejada e largura total do mapa, considerando a informação de cada célula corresponde a vinte pixels.

Para a movimentação do personagem, o algoritmo implementado foi simplificado. Quando uma tecla fosse pressionada uma checagem de colisão é feita, como melhor especificado na Subseção II-B5, e se o caminho estiver livre desenha-se um tampão no local onde o personagem se encontrava anteriormente e em seguida o personagem era desenhado na posição final de movimentação.

:

5) Colisões e Contador de Passos: **Tomando-se como base a malha cartesiana mencionada anteriomente, o sistema de colisões foi implementado utilizando-se um arquivo externo .data. Esse arquivo contém basicamente uma matriz com um caractere específico em casa posição correspondente do mapa da fase. O que cada um deles representa pode ser melhor visto na Tabela I.**

Caractere	Significado
”X”	Barreira
”0”	Chão
”E”	Esqueleto
”P”	Pedra
”T”	Espinho

Table I
SIGNIFICADO DAS REPRESENTAÇÕES DO ARQUIVO DE COLISÕES.

Com este arquivo de colisões, primeiramente, no início de cada fase, uma rotina passa por cada célula da malha verificando se há algum esqueleto, pedra ou espinho. Caso haja correspondência com um desses caracteres, desenha o elemento na posição correspondente da malha cartesiana.

Além disso, todas as vezes em que o personagem se move dentro do labirinto são checados se a posição de destino é um obstáculo, um esqueleto, uma pedra ou um espinho, nessa ordem. Caso a posição esteja ocupada por um obstáculo, nada é feito e reinicia-se o loop de input do teclado. Caso a posição esteja ocupada por um esqueleto, este é movido uma célula na direção do movimento, é chamada a rotina de animação do chute do personagem, mas este fica imóvel onde estava. No entanto, também é verificado se o esqueleto encontra-se encurralado por uma parede ou uma pedra, caso esteja ele é retirado do mapa. Outro caso, é se a posição de destino está sendo ocupada por uma pedra, neste caso ele pode se mover como o esqueleto ou, se estiver encurralada, ficará imóvel. O último caso é para quando o personagem está se movendo para uma posição onde há espinhos. Neste caso, ele pode se mover, mas em seguida será chamada a rotina que anima o ferimento causado pelo espinho, com o personagem ficando temporariamente coberto de sangue.

Outra questão importante na implementação do sistema

de detecção de colisões é que quando o personagem interage com alguns elementos do mapa, estes devem ser trocados de posição na matriz de colisões. Como existe a possibilidade da fase ser reiniciada, é preciso garantir que a matriz de colisões originais sempre seja recarregada a cada novo início ou reinício de fase. Portanto, como procedimento inicial, as informações do arquivo .data são carregadas em um espaço de memória alocado no início do código, denominado com a label *colisao_temporaria*.

Por fim, todas as vezes que uma tecla é pressionada, subtrai-se um passo do personagem. Caso a quantidade de passos se esgote, a fase será reiniciada. Para acompanhamento da quantidade de passos, foi utilizada uma system call customizada disponível no arquivo "SYSTEMv21.s" que permite o visualização de números inteiros no Bitmap Display.

III. Resultados e Desafios

O projeto alcançou como principal resultado a criação de um jogo baseado em Helltaker, contendo menu inicial, cinco fases, diálogos com personagem, desenvolvendo uma história e aumentando o desafio dos *puzzles* a cada novo labirinto. As principais funcionalidades presentes no jogo são: a movimentação do personagem com as teclas W, A, S, D representando as direções cima, esquerda, baixo e direita respectivamente.

Figure 4. Menu inicial do jogo com escolha de opções.

O jogo se inicia com a apresentação do menu inicial. Após a escolha da opção "Novo Jogo", o jogador irá receber a primeira tela de diálogo de um dos personagens fazendo um pequena introdução da história. Ao pressionar enter, o jogo prossegue para a primeira fase, que possui um nível de dificuldade bem simples. A partir daí, o jogo segue aumentando a dificuldade em cada fase, onde o objetivo é atingir a demônia dentro no número máximo de passos estipulados. O código fonte com todos os recursos necessários para rodar o jogo podem ser encontrados no endereço *https://github.com/henriqueramosqs/Helltaker*.

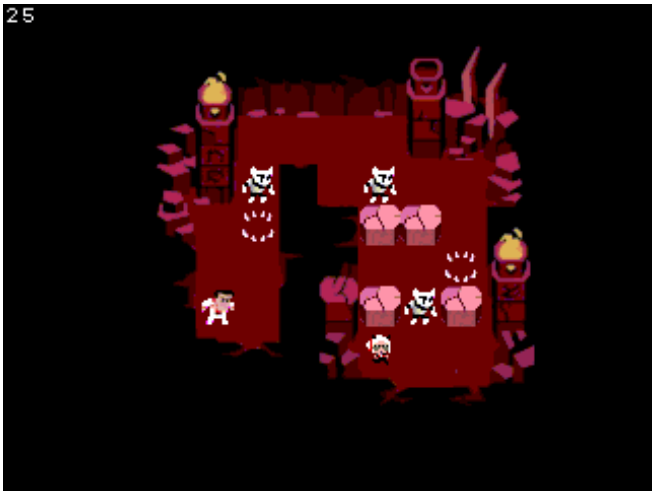

Figure 5. Demonstração da segunda fase do jogo com os elementos da mapa e o personagem principal.

Durante o desenvolvimento do projetos várias desafios, de variados níveis, foram enfrentados, e serão agora brevemente abordados como reflexão dos resultados obtidos. Primeiramente, pode-se citar a dificuldade de utilização e gestão dos registradores disponíveis no processador. Diante da familiridade com linguagens de programação de níveis mais altos, existe um costume de implementar algoritmos utilizando a lógica de variáveis, funções, recursões e estruturas de dados mais complexas. Uma vez se deparando com a linguagem assembly, várias vezes durante a escrita do código foi necessária a depuração porque um dos registradores utilizados estava sendo afetado por outra parte do código e causando inconscistências de resultados ou até mesmo erros de enderaçamento de memória.

Um segundo desafio que pode ser mencionado, é dificuldade de manter o código dentro de um tamanho mínimo. Ao ir implementando instruções, facilmente se atinge a casa das milhares de linhas de código, e muitas vezes é visível a repetição desnecessária de certas partes do código. Porém, transformar estas partes repetitivas em "funções" reutilizáveis constituiu um desafio.

Outro desafio encontrado, que pode ser explicitado aqui, é a relativa escassez de materiais e tutoriais sobre RISC-V ou até mesmo sobre linguagem assembly. Isso levou à

necessidade de implementar quase todas etapas partindo-se de pouca base, se não de nenhuma. Porém, se por um lado isso se constitui um desafio, por outro contribuiu para o atingimento do objetivo de trazer aprendizado para aqueles que realizam a atividade.

Por último, o maior desafio enfrentado, e até o fim não solucionado, foi a tentativa de utilização de macros. Logo no início da implementação, algumas macros foram criadas para facilitar mais a frente no desenvolvimento. Porém, logo mais a frente do projeto, vários erros começaram a surgir. A princípio não era compreensível qual a sua causa, mas posteriormente começou-se a desconfiar de que o problema estava nas macros e na distância dos *jumps* e *branches* ao longo do código. Esta questão ficou tão impeditiva que em um determinado momento, tomou-se a decisão de excluir de vez todas as macros presentes no código, e implementá-las através de subrotinas semelhantes à funções.

IV. Conclusão

Os objetivos deste trabalho foram, em grande, parte alcançados. O principal objetivo, que era a imersão e expansão do conhecimento em sistema computacionais, foi plenamente possibilitado pela prático do projeto, chegando mesmo até a suplantar a expectativa inicial do grupo de trabalho. As principais funcionalidades do programa foram implementadas, inclusive algumas das mais desafiadoras. Portanto, de modo geral, houve êxito nos resultados obtidos.

Para trabalhos futuros, alguns dos desafios mencionados na seção anterior podem ser retomados, aprimorados e vencidos. Também fica a reflexão de que novas funcionalidades podem ser adicionadas, como o controle de troca de frames, para melhor implementação das animações no jogo, e funcionalidades existentes aprimoradas, como uma melhor modularização do código, para torná-lo mais enxuto e de fácil compreensão, evitando repetição desnecessárias que ainda existe.

References

[1] D. Patterson, *Guia Prático RISC-V*, 2020.

[2] T. T. One. Rars wiki. [Online]. Available: ⟨https://github.com/TheThirdOne/rars/wiki⟩

[3] Steam. Helltaker (steam). [Online]. Available: ⟨https://store.steampowered.com/app/1289310/Helltaker/⟩

[4] S. Resources. Helltaker (sprites). [Online]. Available: ⟨https://www.spriters-resource.com/pc_computer/helltaker/⟩