The Legend of Zelda: A Link to the Future

João Lucas de Andrade, Isabela Honda, Michele Aiko, Mariana Souza

Departamento de Ciência da Computação - Universidade de Brasília (UNB)

lucas987557@gmail.com, Micheleaikomukaihata@gmail.com,
marianafarias.oficial1@gmail.com, isabela.honda.ferreira@gmail.com

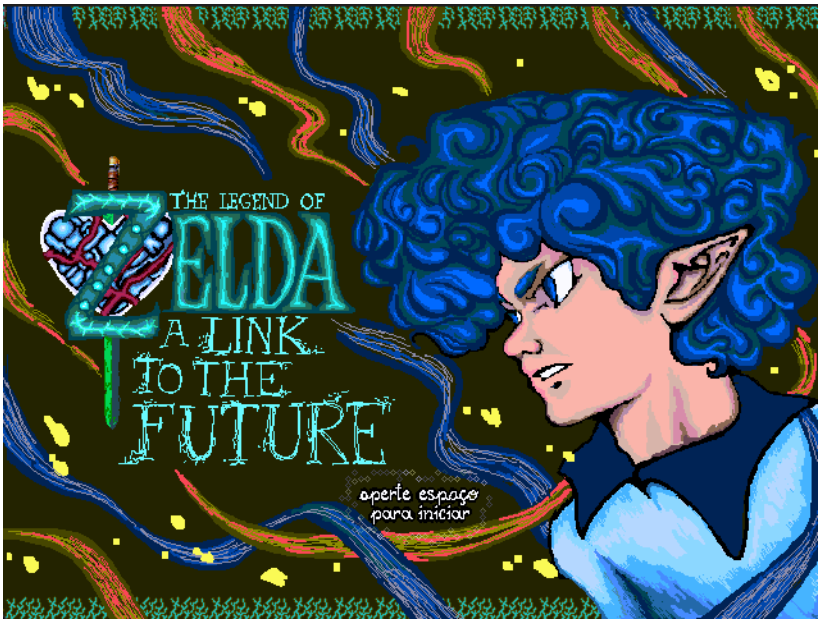

Figura 1.Tela inicial

Resumo. O *objetivo deste artigo é abordar o desenvolvimento de um jogo do gênero RPG/Aventura utilizando a linguagem de programação Assembly na arquitetura RISC-V. O projeto, inspirado na franquia "The Legend of Zelda", implementa mecânicas de exploração, combate, economia (loja) e resolução de quebra-cabeças. O trabalho visa explorar técnicas de programação de baixo nível, manipulação de memória de vídeo (Bitmap), interrupções e lógica de jogos em tempo real*

1. Introdução

O jogo desenvolvido é uma aventura top-down onde o jogador controla um herói que deve atravessar fases, derrotar inimigos e resolver um puzzle para alcançar a vitória. O projeto foi inspirado em "The Legend of Zelda", jogo lançado em 1986 pela Nintendo que se tornou um clássico. Nosso objetivo foi recriar, em um estilo novo, esse ícone da cultura de games oferecendo uma abordagem própria.

2. Metodologia

2.1. Ferramentas utilizadas

As principais ferramentas usadas foram o RARS (RISC-V Assembly Runtime Simulator) e o FPGRARS. O simulador foi essencial para a visualização da memória e depuração via Bitmap Display e Keyboard MMIO. Para a criação e conversão dos sprites e mapas, foram utilizados, pela artista Isabela Honda, os aplicativos Paint.NET, Pixel Studio e um script de conversão de imagem compatível com o simulador.

2.2. Personagem e Mecânicas

O protagonista possui sprites animados para quatro direções (Cima, Baixo, Esquerda, Direita). A movimentação é baseada em grid, garantindo alinhamento perfeito com o cenário. As mecânicas incluem:

Movimentação: Controle por WASD, com verificação de limites de tela e colisão.

Combate: Sistema de ataque com espada (Espaço) e defesa com escudo (comprável).

Inventário: Coleta de moedas, compra de itens na loja (Corações, Escudo) e animação especial ao obter itens.

Vida: Sistema de 3 a 4 corações, com frames de invencibilidade e feedback visual (piscada) ao receber dano

Figura 2. Personagem Principal

2.3. Inimigos

Foram implementados dois tipos de inimigos com comportamentos distintos:

Drone: Se move em padrões predefinidos, causando dano por contato.

Torre (Turret): Um inimigo estático que dispara projéteis periodicamente em direção ao jogador.

A lógica dos inimigos utiliza verificação de coordenadas relativas ao jogador para detectar colisões e aplicar dano.

Figura 3. Inimigo Drone

Figura 4. Inimigo Turret

2.4. Mapas, Colisões e Puzzle

O jogo conta com um sistema de múltiplos níveis (Fase 1, Fase 2, Loja, Puzzle Final).

Colisão: Utiliza-se um "Mapa de Colisão"(array de bytes) paralelo ao mapa visual. O algoritmo verifica se a coordenada futura do personagem corresponde a um bloco sólido (parede, pilar) ou interativo (porta, baú).

Puzzle: Na fase final, foi implementado um quebra-cabeça lógico onde o jogador deve pressionar botões no chão em uma sequência específica (baseada em uma dica visual) para liberar a porta da vitória. Errar a sequência reinicia o puzzle e toca um som de erro.

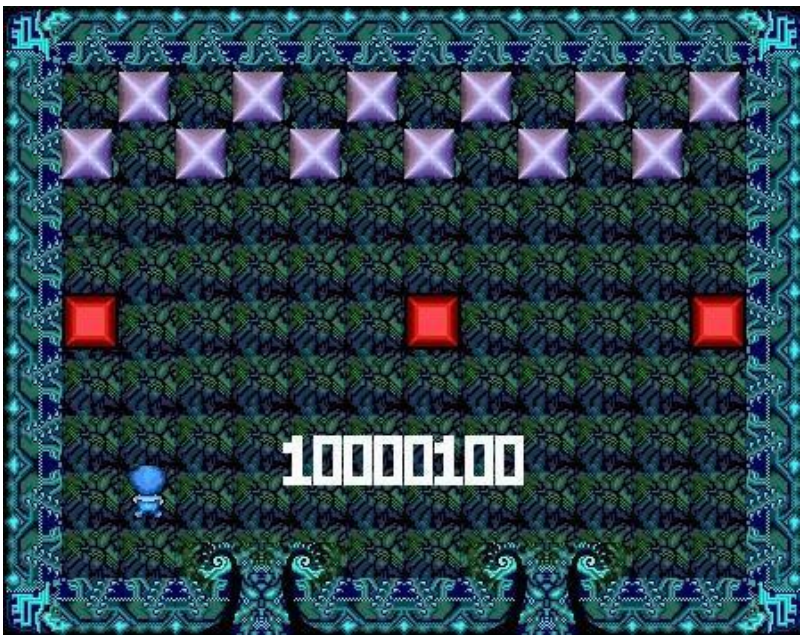

Figura 5. Mapa 4 (Puzzle)

2.5. Música e Áudio

A implementação sonora é híbrida:

Música de Fundo: Utiliza um player avançado ("audioplayerPT.s") que lê uma estrutura de dados de notas e durações, permitindo tocar a trilha sonora sem travar o processamento do jogo.

Efeitos Sonoros: Para efeitos imediatos (ataque, coleta de moeda, dano), utiliza-se a ecall 31 (MIDI out) diretamente, fornecendo feedback instantâneo ao jogador.

2.6. Game Loop

O ciclo principal do jogo gerencia os estados da aplicação:

Input: Leitura do teclado via MMIO.

Update: Atualização da lógica de inimigos, projéteis, animação do personagem e verificação de condições de vitória/derrota.

Render: Desenho do background (otimizado por nível), sprites dinâmicos e HUD (interface de vida e moedas).

Estado: Gerenciamento de transições (telas pretas entre fases, animação de item coletado, telas de Game Over e Vitória)

3. Resultados Obtidos

O projeto resultou em um jogo completo e funcional. Um dos maiores desafios enfrentados foi o gerenciamento da memória de vídeo e o alinhamento dos sprites. Inicialmente, o jogo apresentava travamentos (crashes) quando o personagem tentava sair das bordas da tela, o que foi corrigido com verificações de limite antes do cálculo de movimento.

Outro desafio foi a animação do personagem, que exigiu uma tabela de ponteiros organizada para alternar corretamente entre os quadros de movimento dependendo da

direção. A implementação do áudio junto com o jogo também exigiu otimização para não causar lentidão.

4. Conclusão

O desenvolvimento deste projeto permitiu a aplicação prática dos conceitos aprendidos em sala de aula como os de registradores, pilha, chamadas de sistema e manipulação de memória. A complexidade de criar um jogo em Assembly RISC-V evidenciou a importância da organização do código e do entendimento profundo de como o software interage com o hardware.

5. Referências

-Github com material de apoio da matéria de ISC

https://github.com/victorlisboa/LAMAR?tab=readme-ov-file

-Playlist de videoaulas do professor Marcus Vinícius Lamar

https://www.youtube.com/playlist?list=PLALgy7vIdqUb5INNKsUwkw6Avpo-h_qpm

-Playlist do ex-aluno Thales Menezes

https://www.youtube.com/playlist?list=PLL0Kob75DU32afhLBN5nY2KzOJ5k6lw-Q

-Jogo The Legend of Zelda

https://www.retrogames.cc/nes-games/the-legend-of-zelda-font-mod-retranslation.html

-Site usado para a música

https://www.ninsheetmusic.org/browse/series/TheLegendofZelda