Metroid OAC

Erick Taira Ithalo Medeiros Vitor Lemos Ryan Reis Gustavo de Sousa Santos

Universidade de Brasília, 27 de agosto de 2024

Figura 1: *Título Original do Jogo*

RESUMO

Este relatório detalha o desenvolvimento do jogo Metroid em Assembly para a arquitetura RISC-V, como parte da disciplina de Organização e Arquitetura de Computadores, ministrada pelo professor Lamar na Universidade de Brasília. O código do jogo foi implementado utilizando a ferramenta RARS e foi otimizado para garantir a execução eficiente na placa FPGA DE1-SoC, com ajustes específicos para adequação às limitações do hardware.

Palavras-chave: Metroid · Assembly RISC-V · Matriz

1 INTRODUÇÃO

Metroid, lançado em 1986 pela Nintendo para o Nintendo Entertainment System (NES), é um jogo de ação e aventura que combina elementos de plataforma e exploração. O jogador assume o papel de Samus Aran, uma caçadora de recompensas que deve infiltrar-se no planeta Zebes para derrotar os Piratas Espaciais e destruir os Metroids, criaturas perigosas que os Piratas pretendem usar como armas biológicas.

O jogo é conhecido por sua atmosfera sombria, jogabilidade não-linear, e exploração, onde os jogadores adquirem novos itens e habilidades para acessar áreas previamente inacessíveis. Metroid foi inovador por apresentar uma protagonista feminina forte e por seu design de mundo aberto, que incentivava a exploração e a descoberta.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Devido à recente introdução da linguagem Assembly RISC-V, há uma escassez de materiais de fácil compreensão disponíveis na internet. Nesse contexto, os vídeos disponibilizados no YouTube por antigos monitores da disciplina, juntamente com as monitorias presenciais e online conduzidas pelo monitor Pedro Avilla, foram essenciais para o progresso bem-sucedido do projeto. Além disso, a ferramenta FPGRARS, desenvolvida por LeoRiether no GitHub, proporcionou uma simulação eficaz do projeto em software.

2.1 BITMAP DISPLAY E KDMMIO

O Bitmap Display é uma ferramenta do RARS que permite renderizar imagens em uma tela de 320x240, enquanto o KDMMIO possibilita a utilização do teclado para entrada de dados. Ambas as ferramentas são acessíveis por meio de syscalls e oferecem suporte para a placa FPGA DE1-SoC. Para aprender a utilizá-las, recorremos especialmente ao vídeo "RISC-V RARS - Renderização dinâmica no Bitmap Display", disponível no canal Davi Paturi no YouTube. Nesse vídeo, adquirimos conhecimento sobre como renderizar imagens na tela, mover o personagem e estrutura do loop principal do jogo

2.2 PROJETOS ANTIGOS

No Aprender3 há vários projetos antigos disponíveis, mas devido à complexidade desses trabalhos, a compreensão do código se mostrou desafiadora e, portanto, pouco útil para o nosso projeto. No entanto, as conversas com os criadores desses projetos foram extremamente valiosas, oferecendo dicas sobre como começar o desenvolvimento e fornecendo ideias gerais para a implementação de diversas funcionalidades.

3 METODOLOGIA

3.1 ORGANIZAÇÃO

O projeto foi desenvolvido em etapas progressivas de complexidade. Começamos pelas tarefas mais simples e, à medida que ganhávamos experiência com a linguagem, avançamos para as partes mais desafiadoras. Iniciamos com a renderização de imagens, seguidos pela movimentação do personagem, mapa deslizante, integração de itens, adição de músicas, implementação da gravidade, integração ao tamanho final do jogo, desenvolvimento da arma do personagem e, por fim, a criação da inteligência artificial dos inimigos.

3.2 MATRIZ DE BLOCOS E RENDERIZAÇÃO DE IMAGEM

O primeiro passo para dar início ao jogo foi criar um sistema em que cada número em uma matriz seria um bloco, isso é necessário, devido as limitações da placa FPGA DE1-Soc que possuí apenas 4kiB de dados.

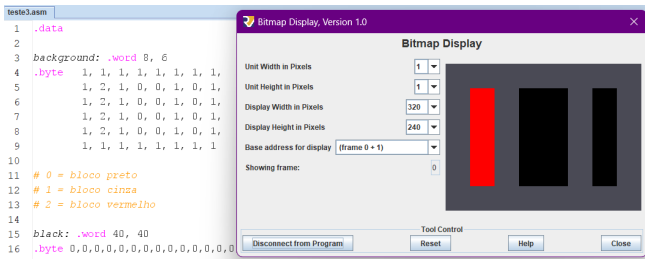

Figura 2: *Exemplo de estágios iniciais do projeto, cada número é um bloco 40x40.*

Listing 1: *parte do código de como percorrer a matriz e como renderizar os blocos da imagem acima.*

```
.text
PERCORRE_MATRIZ:la t0, background #carrega o
    endereco do background
    lw t1, 0(t0)     #carrega a largura
    addi t1, t1, 1
    lw t2, 4(t0)     #carrega a altura
    addi t0, t0, 8     #pula a largura e a
        altura
    li t4, 1     #t4 = 1
    li t5, 1     #t5 = 1
    mv s0, zero
    mv a1, zero    #a1 = 0
    mv a2, zero    #a2 = 0
    mv a3, zero    #a3 = 0

WHILE:     beq t4, t1, CASO_1   #t4 = t1 ? caso
    sim pule para caso_1
    j ITERACOES    #caso n o pule para iteracoes

CASO_1:    beq t5, t2, END    #t5 = t2 ? caso sim
     pule para end
    addi t5, t5, 1     #t5 += 1
    li t4, 1     #t4 = 1
    addi a2, a2, 39    #a2 += 39
    j ITERACOES

ITERACOES:  lb t3, 0(t0)     #guarda os bytes de
     background em t3
    li t6, 0     #t6 = 0
    bne t3, t6, CASE2 #se t6 != t3
    la a0, black     #carrega em a0 o endereco
        de black
    jal SALVAMENTO
    addi a1, a1, 40    #a1 += 40
    addi t0, t0, 1     #t0 += 1
    addi t4, t4, 1     #t4 += 1
    addi s0, s0 ,1
    j WHILE

CASE2:     li t6, 1
    bne t3, t6, CASE3 #se t6 != t3
    la a0, gray
    jal SALVAMENTO
    addi a1, a1, 40    #a1 += 40
    addi t0, t0, 1     #t0 += 1
    addi t4, t4, 1     #t4 += 1
    addi s0, s0 ,1
    j WHILE

        ...
        #CASE3 e analogo ao CASE2.
```

3.3 Movimentação do Personagem e colisão

Para a movimentação do personagem, optamos por realizá-la diretamente na matriz de blocos, o que simplificou significativamente a implementação da colisão. Dessa forma, a detecção de colisões foi feita de maneira simples: bastava verificar o bloco à frente do movimento e, em caso de colisão, retornar ao game loop. Para criar a impressão de movimento, a tela foi programada para se atualizar constantemente dentro do game loop.

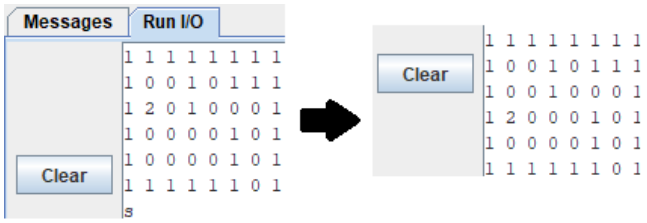

Figura 3: *Movimentação do personagem direto na matriz.*

Na matriz percebe-se, diferentes números que representam elementos distintos: o número 1 corresponde às paredes, o 0 indica espaço vazio, e o 2 representa o personagem.

Listing 2: *Como foi realizado a movimentação, nesse estágio a entrada de dados ainda não havi sido realizada no KDMMIO.*

```
        GAME_LOOP:
    li a7, 12
    ecall
    mv s0, a0
    la a0, pular
    li a7, 4
    ecall
    la t0, background #endereco do
        mini_background
    addi t0, t0, 8     #pula a largura e a altura

    #MOVIMENTACAO
    li t1, 'w'
    bne s0, t1, BAIXO
    #CIMA
    la t2, CHAR_POS    #endereco de CHAR_POS
    li t3, 8     #carrega 4 em t3
    lh t4, 0(t2)     #carrega o y em t4
    mul t4, t4, t3     #em qual linha o CHAR esta
    lh t3, 2(t2)     #carrega o x em t3
    add t4, t4, t3     #em qual coluna o CHAR esta
          #agora t4 contem a posicao do CHAR_POS
              em relacao a matriz
    #procedimento que averigua colisao
    add t0, t0, t4     #endereco do CHAR_POS na
        matriz
    addi t0, t0, -8    #novo endereco do CHAR_POS
        na matriz
    lb t3, 0(t0)     #carrega o valor de CHAR_POS
        da matriz
    li t5, 1     #t5 = 1
    addi t0, t0, 8
    beq t3, t5, GAME_LOOP #va para GAME_LOOP
    li t3, 0     #carrega 0 em t3
    sb t3, 0(t0)     #colocando 0 no endereco
    li t3, 2     #carrega 2 em t3
    addi t0, t0, -8    #novo endereco do CHAR_POS
        na matriz
    sb t3, 0(t0)     #colocando 2 no endereco
    lh t3, 0(t2)     #carrega o y em t3
    addi t3, t3, -1    #decrementa um em t3
```

```
sh t3, 0(t2)    #novo CHAR_POS
jal PRINT_MATRIZ
j GAME_LOOP
```

3.4 Mapa deslizante

Para o nosso projeto era necessário que o background fosse móvel, para isso foi criado uma flag que informa em que posição deve-se iniciar a renderização das tiles da matriz de blocos, nos estágios iniciais essa movimentação era feita de forma manual nas teclas 'o' e 'p'.

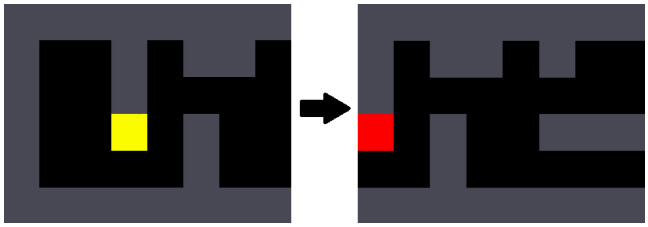

Figura 4: *Movimentação da tela.*

Listing 3: *Como foi realizado a movimentação da tela, nesse estágio a entrada de dados já está sendo realizada no KDMMIO.*

```
    .data
        POS_MAPA: .half 0
    .text
        ...
    KEY2:   li t1,0xFF200000    # carrega o
        endereco de controle do KDMMIO
    lw t0,0(t1)     # Le bit de Controle
        Teclado
    andi t0,t0,0x0001   # mascara o bit menos
        significativo
      beq t0,zero,FIM         # Se nao ha tecla
           pressionada entao vai para FIM
      lw t2,4(t1)       # le o valor da tecla
          tecla

    li t0,'w'
    beq t2,t0,CIMA      # se tecla pressionada
        for 'w', chama CHAR_CIMA

    li t0,'a'
    beq t2,t0,ESQUERDA    # se tecla
        pressionada for 'a', chama CHAR_CIMA

    li t0,'s'
    beq t2,t0,BAIXO     # se tecla pressionada
        for 's', chama CHAR_CIMA

    li t0,'d'
    beq t2,t0,DIREITA   # se tecla pressionada
        for 'd', chama CHAR_CIMA

    li t0,'o'
    beq t2, t0,MOV_ESQ    # se tecla
        pressionada for 'o', chama MOV_ESQ

    li t0,'p'
    beq t2,t0,MOV_DIR   # se tecla pressionada
        for 'p', chama MOV_DIR

    li t0,'u'
    beq t2,t0,EXIT

        ...

        MOV_ESQ:  la t0, POS_MAPA   #endereco
            da posicao do mapa
    lh t1, 0(t0)    #valor do endereco
    li t2, 0    #t2 = 0
    beq t1, t2, GAME_LOOP #se t1 == t2   porque
         est  na borda e o mapa n o pode ir
        mais pra esquerda
    addi t1, t1, -1   #caso contr rio
    sh t1, 0(t0)    #salvando no endereco da
        posicao do mapa
    ret

MOV_DIR:  la t0, POS_MAPA   #endereco da
    posicao do mapa
    lh t1, 0(t0)    #valor do endereco
    addi t1, t1, 8    #
    li t2, 16   #t2 = 0
    beq t1, t2, GAME_LOOP #se t1 == t2   porque
         est  na borda e o mapa n o pode ir
        mais pra esquerda
    addi t1, t1, -8
    addi t1, t1, 1    #caso contr rio
    sh t1, 0(t0)    #salvando no endereco da
        posicao do mapa
    ret
```

3.5 Integração de Itens

Nessa etapa foi criado o primeiro item que consistia na diminuição voluntária do personagem ao clicar a tecla 'h' e para retornar ao tamanho normal 'g', dessa forma também foi necessário que o personagem tivesse dois valores, parte superior e inferior, na matriz de blocos.

Figura 5: *O item é o bloco azul.*

3.6 Adição de música

A adição de música foi uma das etapas mais significativas do projeto, não tanto pela sua importância em si, mas porque nos permitiu aprender a manipular o tempo do jogo. Com esse conhecimento, a implementação de outras funcionalidades, como a gravidade e a temporização dos inimigos, tornou-se muito mais intuitiva.

Listing 4: *Como foi realizado a movimentação da tela, nesse estágio a entrada de dados já está sendo realizada no KDMMIO.*

```
    .data
NUM: .half 39          # N mero de notas a
    tocar
          # Lista de notas (tom, dura  o, tom,
              dura  o, ...)
NOTAS: .half
    84,147,84,147,84,147,84,441,82,441,...

.text
START:
    # Inicializa  o
    la s10, NOTAS           # Carrega o
        endere o da lista de notas em s10
    csrr s11, time          # L  o tempo
        atual do registrador de tempo
```

```
    li t4, 0                # Inicializa o
        contador de notas
    li s9, 0

MAIN:
    jal MUSIC               # Chama a fun   o
        MUSIC
    j MAIN                  # Loop principal

MUSIC:
    csrr t0, time           # L   o tempo atual
         do registrador de tempo
    sub t0, t0, s11          # Calcula ?T (
        tempo decorrido)
    lh t2, -2(s10)
    bge t0, t2, PROXIMA_NOTA # Se ?T >=
        dura   o da nota, vai para
        PROXIMA_NOTA
    ret                     # Retorna para o
        chamador

PROXIMA_NOTA:
    la t0, NUM
    lh t0, 0(t0)
    lh a0, 0(s10)            # Carrega a nota
        atual em a0
    lh a1, 2(s10)            # Carrega a
        dura   o da nota em a1
    li a2, 0                # Define o
        instrumento
    li a3, 127              # Define o volume
    li a7, 31               # Define a syscall
        para tocar a nota
    ecall                   # Toca a nota

    addi s9, s9, 1
    addi s10, s10, 4          # Avan a para a
        pr xima nota (cada entrada    4 bytes)
    csrr s11, time           # L   o tempo
        atual do registrador de tempo
    beq s9, t0, END_MUSIC
    ret
END_MUSIC:
    la s10, NOTAS            # Carrega o
        endere o da lista de notas em s10
    csrr s11, time           # L   o tempo
        atual do registrador de tempo
    li t4, 0                # Inicializa o
        contador de notas
    li s9, 0
    ret
```

3.7 GRAVIDADE

A implementação da gravidade do personagem foi realizada através da função de movimentação vinculada à função KEY.2. Dentro dessa função, o caso default fazia com que o personagem caísse. No entanto, para evitar que o personagem ficasse "preso"ao chão, foi necessário temporizar o tempo de queda e ativar a gravidade apenas quando o personagem estivesse no ar. Para isso, foram criadas várias flags na memória que indicavam se o personagem estava voando, além de um temporizador para garantir que a gravidade não fosse aplicada de forma instantânea..

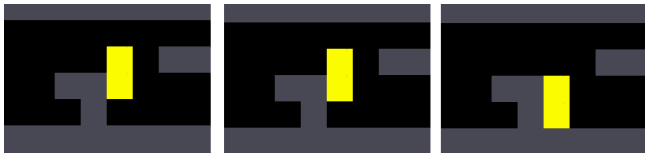

Figura 6: *Gravidade.*

3.8 INTEGRAÇÃO AO TAMANHO FINAL DO JOGO

Após implementar todas as etapas anteriores, consideramos que estávamos prontos para escalar o jogo para sua versão completa. Como todas as funcionalidades já estavam implementadas, foi necessário apenas ajustar alguns parâmetros no código para adaptá-lo à escala real do jogo.

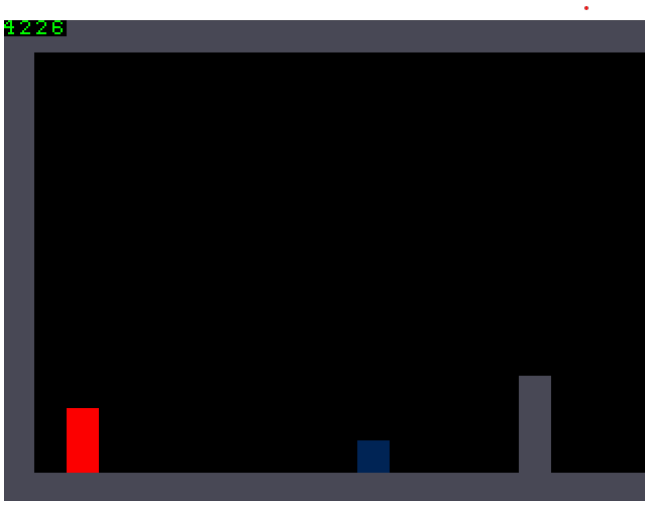

Figura 7: *Escala real do projeto, porém com sprites temporárias.*

3.9 COMBATE

No jogo original, há diversas armas, mas optamos por implementar apenas a arma mais básica. Inicialmente, planejamos um projétil que percorreria seu alcance máximo sem deixar rastro. No entanto, por motivos não identificados, essa funcionalidade funcionava corretamente na FPGRARS, mas não na placa FPGA. Por isso, decidimos simplificar a arma para um laser. Além disso, nessa etapa, também adicionamos as sprites finais do jogo e o segundo item que aumentaria o range e o dano da arma.

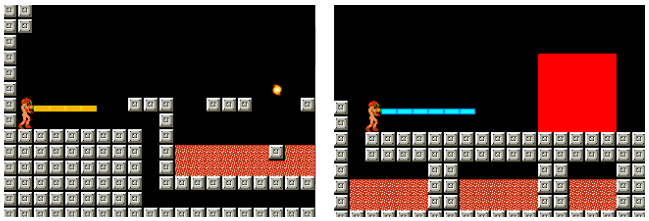

Figura 8: *Laser normal e laser aprimorado*

3.10 INIMIGOS

Finalmente, implementamos os inimigos do jogo, que consistem em lava, bola de fogo, laser e o BOSS. A lava é um bloco simples que causa dano ao jogador, sem características especiais. A bola

de fogo é mais interessante, subindo e descendo em um movimento contínuo, enquanto o laser é uma parede que aparece e desaparece em intervalos de tempo. Vale destacar que, se o jogador encostar em qualquer um desses inimigos, ele morre e retorna ao último checkpoint.

A implementação da movimentação da bola de fogo e do laser são semelhantes à do próprio jogador, porém são scriptadas.

O BOSS, por sua vez, é um inimigo pacífico, com muita vida, e sua sprite é propositalmente simples para homenagear os estágios iniciais do jogo, onde o personagem e o BOSS são visualmente idênticos.

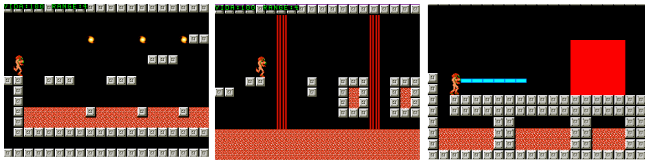

Figura 9: *Inimigos.*

4 Resultados Obtidos

4.1 Jogabilidade

A jogabilidade do jogo em sua forma final é bastante simplificada quando comparado ao jogo original, a mesma é afetada em grande parte devido à forma como as a movimentação do personagem foi lidada, onde na matriz de blocos o personagem movia de tile em tile, dificultando assim a movimentação fluída do personagem. Dito isso, conseguimos implementar ao final:

- Movimentação do personagem e colisão
- Gravidade
- Três mapas
- HUD
- Inimigos distintos
- Arma
- Música

O que não conseguimos implementar mas desejávamos:

- Movimentação mais flúida
- Inimigos mais inteligentes

Foi considerado criar um inimigo que analisava a posição do jogador e iá em direção a ele, porém dada a complexidade optamos por não implementar.

4.2 Problemas encontrados

Um dos maiores desafios enfrentados foi a execução do programa em diferentes plataformas: FPGRARS, RARS e na placa DE1-SoC. Muitas vezes, o código funcionava perfeitamente em uma plataforma, mas apresentava problemas em outra. Um exemplo disso é a impossibilidade de utilizar o comando .include na FPGRARS, o que resultava em um código volumoso e um tanto desorganizado.

5 Conclusões e Trabalhos futuros

Apesar das diversas dificuldades encontradas, consideramos que concluímos o projeto de maneira satisfatória. Durante o desenvolvimento, adquirimos um valioso aprendizado, especialmente no que diz respeito à programação em Assembly RISC-V. Além disso, foi extremamente gratificante observar o progresso que fizemos ao longo do curso de Ciências da Computação. Lembro-me de que, no início do curso, como calouros na disciplina de ISC, desenvolvemos um projeto semelhante, o PAC-Man, mas de forma muito mais simples. Agora, com mais experiência e conhecimento, conseguimos aplicar conceitos de outras disciplinas, o que elevou a qualidade deste projeto em comparação ao que realizamos em ISC.

6 Referências

Referências

[1] Davi Paturi e Gabriel B. Gomes Tradutor midi para RISC-V

[2] Davi Paturi Tutorial do bitmap display

[3] Victor Lisboa Introdução ao Rars

[4] Nes Metroid- The Spriters Resource Spriters Resource

[5] Aulas gravadas na pandemia do Professor Lamar