Badly Programmed Ice Cream

Arthur Araújo - 232000472 Arthur Silva - 232000490 João Carlos - 232009511

University of Brasília, Dept. of Computer Science, Brazil

Abstract

O projeto ”Badly Programmed Ice Cream” foi criado como um remake do jogo Bad Ice Cream feito em Assembly RISC-V utilizando o RARS como ferramenta de desenvolvimento. O projeto foi desenvolvido durante as aulas da disciplina de Introdução a Sistemas Computacionais. RISC-V é uma arquitetura de conjunto de instruções (ISA) baseada em princípios de design de processadores de redução de conjunto de instruções (RISC). Ela é caracterizada por ser livre e aberta, o que significa que seu design está disponível publicamente e pode ser usado livremente por qualquer pessoa ou organização. Foi desenvolvida na Universidade da Califórnia, em Berkeley, e é amplamente reconhecida por sua flexibilidade e eficiência.

1 Introdução

”Bad Ice-Cream” é um jogo de arcade desenvolvido pela Nitrome, lançado em 10 de Dezembro de 2010. No jogo, os jogadores controlam um personagem que deve coletar frutas em um labirinto enquanto evita inimigos. O objetivo é coletar todas as frutas para completar o nível. O jogo é conhecido por seu estilo de arte pixelizada e jogabilidade simples, mas desafiadora. No jogo original, o jogador deve escolher um dos sorvetinhos para controlar, sendo capaz de criar blocos de gelo para bloquear inimigos ou abrir caminho através do labirinto, mas também devem evitar que os inimigos quebrem os blocos de gelo ou os atinjam diretamente. Novos desafios e elementos são introduzidos ao longo das fases, tornando cada nível mais complexo e exigindo estratégia e habilidade para serem concluídos.

Figure 1: Jogo original Bad Ice Cream

Na seção 2 será apresentada a metodologia utilizada. A seção 3 apresenta os resultados obtidos. A seção 4 conclui este trabalho.

2 Metodologia

Inicialmente, o projeto foi desenvolvido no RARS, apresentado pelo professor Lamar. Contudo, durante o semestre, atualizações foram feitas e a testagem passou a ser feita com a ferramenta FPGRARS criada pelo ex-aluno Leonardo Riether. A ferramenta facilita a execução e garante uma velocidade superior em comparação ao montador original RARS.

De maneira análoga aos projetos de alunos anteriores, foi implementado um sistema baseado em matrizes, as quais determinam o comportamento posicional do jogador, como também dos inimigos e dos coletáveis.

2.1 Planejamento

Toda comunicação durante o desenvolvimento do projeto foi realizada via Discord e Whatsapp. O projeto foi iniciado no dia 12/11 e finalizado no dia 21/12. Ao longo de cada semana, houveram reuniões para organização e definição de tarefas semanais. Para o versionamento de código, foi utilizado o GitHub, o qual possibilitou uma melhora significativa na organização e implementação de cada etapa do jogo.

Cada sprite utilizada para representação visual no jogo foi feita em conjunto, caracterizada pelas artes do aluno Arthur Araújo.

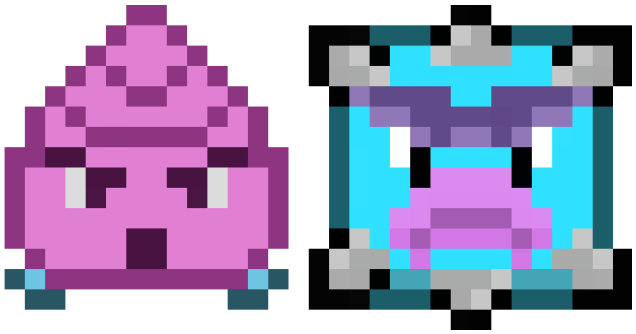

Figure 2: Sprites do player e do inimigo

2.2 Renderização

Para renderizar uma imagem no bitmap display do RARS/FPGRARS, foi necessário entender como os dados são interpretados. O bitmap consiste em dois grandes vetores de bytes, cada um representando uma camada de cor na tela. Cada byte do vetor codifica uma cor específica, e a tela tem uma resolução de 320x240 pixels (resolução 4:3). A interpretação dos dados do bitmap é feita de forma sequencial, começando pelo primeiro byte (pixel) na quina superior esquerda da tela e avançando ordenadamente até atingir a quina inferior direita, pulando para a próxima linha a cada 320 pixels. Nesse viés, com base no entendimento do funcionamento do bitmap display, e no tutorial feito pelo ex-aluno Davi Paturi, foi possível implementar o sistema inicial de renderização do mapa e do player.

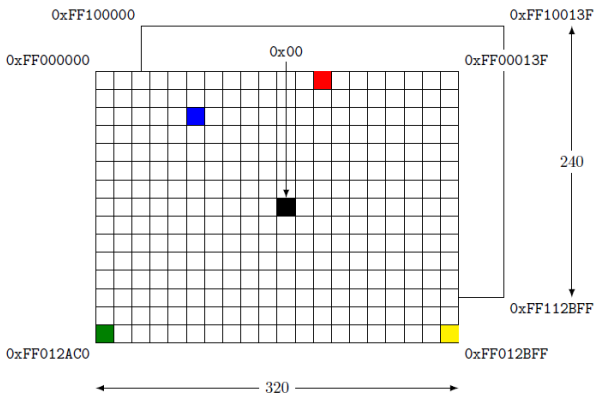

Figure 3: Funcionamento do bitmap display

2.3 Movimentação

Para a movimentação, usamos a chamada do teclado para definir a direção, sendo: a = esquerda, d = direita, s = baixo e w = cima. Desse modo, utilizamos endereços que vão verificar a colisão e calcular a nova posição do personagem, armazenando em uma .half, enquanto a posição antiga será sobreposta pelo tile de chão.

```
############################
#   SISTEMA DE MOVIMENTAÇÃO  #
############################

CHAR_ESQ:   la t1, playerstate
        li t2, 2
        lw t3, (t1)
        bne t3,t2, FIRST_ESQ
        sw t2, (t1)         #PLAYERSTATE 2 = ESQUERDA

        la t0,CHAR_POS          # carrega em t0 o endereco de CHAR_POS
        la t1,OLD_CHAR_POS      # carrega em t1 o endereco de OLD_CHAR_POS
        lw t2,0(t0)
        sw t2,0(t1)         # salva a posicao atual do personagem em OLD_CHAR_POS

        lh t1,0(t0)         # carrega o x atual do personagem
        addi t1,t1,-16          # decrementa 16 pixeis

        j Colisao_horizontal

FIRST_ESQ:  la t1, playerstate
        li t2, 2
        sw t2, (t1)
        j NAO_MOVE_HOR

CHAR_DIR:   la t1, playerstate
        li t2, 3
        lw t3, (t1)
        bne t3,t2, FIRST_DIR
        sw t2, (t1)         #PLAYERSTATE 3 = DIREITA

        la t0,CHAR_POS          # carrega em t0 o endereco de CHAR_POS
        la t1,OLD_CHAR_POS      # carrega em t1 o endereco de OLD_CHAR_POS
        lw t2,0(t0)
```

Figure 4: Trecho de código do sistema de movimentação

2.4 Colisão

No sistema de colisão, desenvolvemos uma matriz (320x240). Na matriz, os índices 0 e 1 representam respectivamente o caminho do chão e dos blocos de parede. Os coletáveis são representados por 2 e o gelo pelo número 3. O inimgo estático da fase inicial é representado pelo 4. Nesse viés, se a posição pretendida pelo usuário na movimentação do personagem corresponder ao número 1 na matriz, a posição não é alterada. Caso contrário, o personagem é movimentado na direção desejada usando a lógica das teclas pressionadas, garantindo a colisão com inimigos e coletáveis no bitmap display.

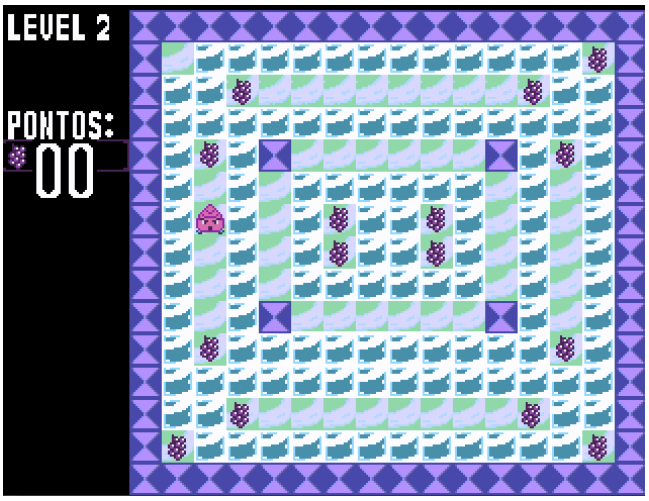

Figure 5: Fase 2 em jogo

```
.byte
    0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    0, 0, 0, 0, 1, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1,
    0, 0, 0, 0, 1, 3, 3, 2, 0, 0, 0, 0, 0, 0, 0, 2, 3, 3, 1,
    0, 0, 0, 0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1,
    0, 0, 0, 0, 1, 3, 2, 3, 1, 0, 0, 0, 0, 0, 0, 1, 3, 2, 3, 1,
    0, 0, 0, 0, 1, 3, 0, 3, 0, 3, 3, 3, 3, 3, 3, 0, 3, 0, 3, 1,
    0, 0, 0, 0, 1, 3, 0, 3, 0, 3, 2, 3, 3, 2, 3, 0, 3, 0, 3, 1,
    0, 0, 0, 0, 1, 3, 0, 3, 0, 3, 2, 3, 3, 2, 3, 0, 3, 0, 3, 1,
    0, 0, 0, 0, 1, 3, 0, 3, 0, 3, 3, 3, 3, 3, 3, 0, 3, 0, 3, 1,
    0, 0, 0, 0, 1, 3, 0, 3, 1, 0, 0, 0, 0, 0, 0, 1, 3, 0, 3, 1,
    0, 0, 0, 0, 1, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 1,
    0, 0, 0, 0, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1,
    0, 0, 0, 0, 1, 3, 3, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 3, 3, 1,
    0, 0, 0, 0, 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 1,
    0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

Figure 6: Matriz de representação da fase 2

2.5 Ataque

A mecânica de ataque funciona de maneira semelhante à movimentação e também está localizada no mesmo script ”movement.s”. A cada frame, o programa verifica se o jogador pressionou a tecla ”J” e, caso afirmativo, direciona o programa para uma subrotina que realiza os cálculos para o movimento ”especial” do jogador. Primeiramente, é verificada a variável ”playerstate”, que define para qual lado o jogador está olhando, e redireciona para o código correspondente que imprime os blocos na direção correta. Em seguida, ele usa a variável CHAR-POS e calcula a coordenada exata do bloco respectivo em frente ao jogador. Se o caminho para o ataque estiver livre, o botão ”J” cria uma fileira de blocos até encontrar um tile diferente de ”0” (livre) na matriz da fase, atualizando toda a fileira na matriz com o tile ”3” (gelo). Se o caminho tiver gelo, o ataque quebra a fileira inteira de gelo (3) e reposiciona o caminho livre (0) na matriz. Por fim, se o tile ao lado do jogador for uma parede comum (1), o ataque não é realizado.

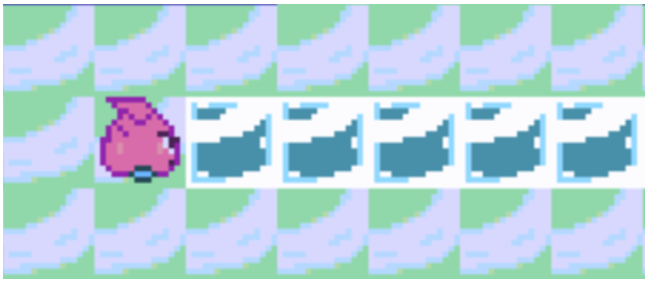

Figure 7: Player realizando o ataque

2.6 Música

Para a transposição da música de abertura para a forma interpretada em assembly, utilizamos o Hooktheory. Nesse contexto, organizamos as notas musicais e suas durações nessa ordem, representadas por números conforme a interpretação da interface. Os efeitos sonoros foram criados como parte dos testes, nos quais experimentamos os instrumentos disponíveis no próprio RARS em diferentes alturas. Esses testes nos permitiram identificar quais sons poderiam ser associados aos efeitos sonoros desejados. Adicionalmente, incluímos uma música de encerramento para quando a vitória é alcançada.

```
###     Música Menu     ###
TAMANHO: .word 80
NOTAS: 36,202,36,202,40,202,36,202,47,202,36,101,45,202,36,101,36,303,36,101,36

###     Música Win      ###
TAMANHO_WIN: .word 78
NOTAS_WIN: 69,185,60,92,71,92,72,462,71,92,69,92,71,92,72,185,76,185,74,185,60,
```

Figure 8: Código da música de abertura

3 Resultados Obtidos

Apesar de algumas dificuldades enfrentadas durante o processo de desenvolvimento, é válido afirmar que obtivemos um resultado final geral satisfatório em relação ao projeto. De uma forma simples, a jogabilidade foi aplicada com todas as ferramentas disponibilizadas pelo Assembly RISC-V. Seguem abaixo alguns problemas e desafios que enfrentamos:

- A dificuldade inicial foi entender como começar, qual o primeiro passo? A partir dos vídeos de ex-alunos e algumas monitorias, essa etapa foi vencida.
- Logo em seguida, nossa dificuldade era implementar os coletáveis e impedir que o jogador ”quebrasse” o jogo invadindo regiões que não devia, problema que foi solucionado com a criação da matriz de representação.
- Implementar um inimigo inteligente no RARS foi um desafio que infelizmente não foi possível concluir devido ao tempo insuficiente conforme a entrega final se aproximava.
- A mecânica do gelo foi um grande desafio, visto que gerava muitos bugs, alguns deles ainda ocultos.

4 Conclusão

Desse modo, é visível que o projeto não foi uma tarefa fácil para alunos do primeiro semestre, e, apesar das dificuldades, o resultado foi muito gratificante. Desde o início, sabiamos que desenvolver um jogo em uma linguagem de baixo nível seria uma tarefa árdua. Além disso, o aprendizado foi coletivo e guiado por todos os conteúdos e materiais disponibilizados em inúmeras fontes, como repositórios antigos do github, slides do professor e vídeos escondidos no youtube.

Figure 9: Tela de Game Over

Referências

https://github.com/victorlisboa/LAMAR
https://github.com/LeoRiether/FPGRARS
https://www.hooktheory.com/theorytab/genres
Aulas do professor Marcus Vinicius Lamar
Computer Organization and Design RISC-V Edition, por David A. Patterson e John LeRoy Hennessy