Universidade de Brasília

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Disciplina: CIC0003 - Introdução a sistemas computacionais - Turma 01 - 2023/2
Professor: Marcus Vinicius Lamar
Nome: Rafael Dias Ghiorzi
Nome: Bernardo Vilar Cunha Lima Oliveira Macedo
Nome: Guilherme Lima Barbosa

Projeto aplicativo
- Bad Santa -

Resumo

O trabalho a seguir é uma releitura do jogo Bad Ice Cream, desenvolvido pela empresa Nitrome em Dezembro de 2010. O jogo foi programado na linguagem Assembly Risc-V R32 com auxílio do simulador RARS (RISC-V Assembler and Runtime Simulator). Neste documento, serão abordados o conceito do jogo original, metodologias de programação, dificuldades, e o resultado final do projeto.

1 Introdução

O jogo Bad Ice Cream(1) é um jogo criado em flash no ano de 2010 e distribuido em 2011 como o primeiro jogo da empresa a ser emulado em HTML5. Nele, o jogador controla um sorvete que tem como objetivo coletar todas as frutas presentes em cada nível, enquanto tenta não ser alcançado pelos inimigos presentes na fase. Ele também conta com a habilidade de criar e quebrar paredes de gelo dentro do mapa para se defender de ditos inimigos. O conceito é simples, porém a simplicidade do jogo é o que torna ele interessante e amigável para qualquer um que tiver a curiosidade de jogá-lo.

Imagem 1: fase do jogo original

2 Desenvolvimento do projeto

imagem 2: foto de menu do jogo

O projeto foi desenvolvido usando o RARS como simulador RISC-V e ambiente para rodar o jogo, o FPGRARS(5) foi usado para tentar rodar o jogo de forma suave, porém sua velocidade milhares de vezes mais rápido que o RARS acabou trazendo alguns problemas inesperados. Um repositório no Github(6) também foi usado como para deixar os arquivos do projeto atualizados entre os integrantes do grupo. Além do conhecimento dado em sala de aula pelo professor, também foi de grande ajuda os vídeos tutoriais desenvolvidos por alunos veteranos e ex-alunos da disciplina.

Como a data de entrega do projeto ficou extremamente perto da época do Natal, decidimos que o tema do nosso jogo seria natalino. Dessa forma, nomeamos o jogo de Bad Santa e fomos criando a temática, onde o personagem principal seria o papai Noel, as paredes a serem criadas, presentes e os inimigos e colecionáveis relacionados também a essa temática

2.1 Metodologia

O programa foi baseado num sistema“híbrido” de matriz. No início, o objetivo era fazer todo o jogo pegar informações de uma única matriz, no entanto, isso se tornou mais uma dificuldade do que um forma de tornar nossas vidas mais fáceis. Assim, decidimos passar para o seguinte método:

A fase é impressa no início do programa, e abaixo disso, existe uma matriz que define certos sprites como números (0 como espaço livre, 1 como parede, 2 como coletáveis, 5 como inimigo estático). Dessa forma. a colisão com qualquer tipo de entidade no jogo se tornou mais fácil.

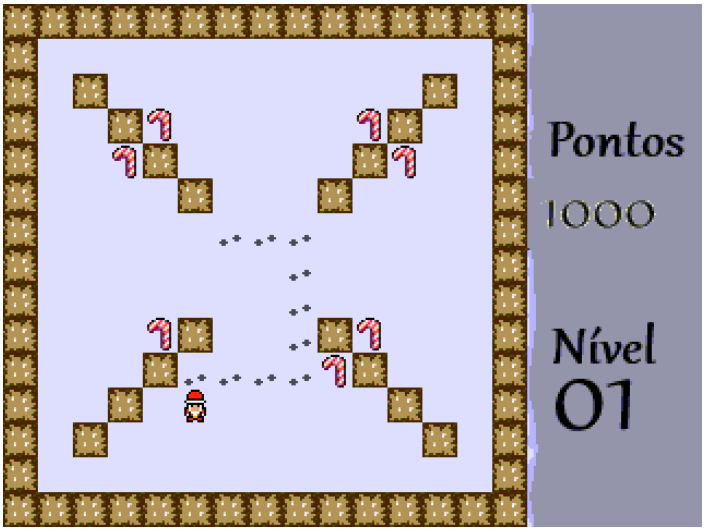

imagem 3: print do jogo em desenvolvimento

Originalmente a colisão do inimigo (Grinch) dinâmico com o jogador, foi feita se baseando no mesmo cálculo da posição relativa da matriz, porém sem usar a matriz. Eram salvos os respectivos X e Y de ambos em registradores S toda vez que eles se moviam, e em todo loop de movimentação de ambos, era checado se esses X e Y eram iguais, como condição de derrota.

Entretanto, para facilitar a implementação do ataque do personagem, trocamos a metodologia de colisão para utilizar a matriz que é utilizada na fase. Para fazer isso, fizemos com que o boneco assumisse um valor dentro da matriz e esse valor se deslocasse com ele quando ele andasse pelo mapa. O inimigo também recebeu um código similar para mover seu número designado pela matriz.

O ataque do personagem foi implementado utilizando a colisão por matriz para fazer uma verificação dos espaços a frente do personagem, o ataque verificava se o espaço a seguir era um espaço vazio caso fosse transformava o numero de espaço vazio para o número que foi designado para o bloco criado pelo ataque, e em seguida chamava a função para imprimir o bloco. A quebra usava a mesma lógica apenas substituindo o fato de que se o bloco a seguir fosse um bloco criado pelo jogador ele teria seu número da matriz trocado por zero (espaço vazio) e era chamada a função de impressão de um bloco vazio.

O placar de fase e pontuação também funcionava de uma forma simples. No início da fase era impresso no display a fase com o número da fase e carregado sua matriz na memória. A pontuação era apenas um macro que checava a pontuação guardada no registrador s2 e imprimia a pontuação correspondente no bitmap.

O sistema de impressão no bitmap display foi implementada com uma função PRINT genérica que servia para qualquer sprite de qualquer tamanho. Essa função foi inicialmente criada com a ajuda de um vídeo tutorial(2), entretanto, para que o código funcionasse junto com a técnica da matriz, foram necessárias algumas mudanças, não bastando apenas 4 parâmetros variáveis para a impressão.

A música implementada no menu e os efeitos sonoros foram implementados usando novamente a ajuda de scripts(4) e tutoriais(3) de ex-alunos, usando as syscalls padrão do sistema (31,32 e 33). Essa implementação foi direto ao ponto, sem problemas nenhum.

As artes foram todas feitas à mão a partir do aplicativo Paint, Photoshop e um site utilizado para fazer pixel arts(8). A imagem do menu foi gerada com a inteligência artificial GPT-4 e DALL-E a partir do Bing AI(7) e estendida com esses mesmo aplicativos.

3 Dificuldades

O processo de programar o jogo em uma linguagem que não é familiar à maioria é normalmente dificil, e a linguagem Assembly não foi diferente. Por ser uma linguagem de baixo nível, foi complicado seguir a lógica que normalmente usamos quando programamos em linguagens mais elegantes, como python. A seguir, serão listados algumas dificuldades principais que surgiram durante o projeto.

3.1 Alocação de memória e registradores

A alocação de memória e registradores disponíveis em pequena quantidade para fazer as rotinas do código foi uma tarefa ardilosa, visto que, como recém estudantes de programação, não é de costume programar enquanto presta atenção nos valores que nós mesmo colocamos dentro da memória e dos registradores da CPU. Para solucionar esse problema, foi necessário muita tentativa e erro e comentários dentro do código para conseguir seguir uma lógica correta, reconhecendo quais registradores eram utilizados para quais funções, a fim de resolver os conflitos existentes dentro do programa.

3.3 Ret

Por algum motivo, a função ret no nosso código simplesmente se recusava a voltar para a instância que o chamou. Dessa forma, tivemos que fazer um caminho muito mais longo para resolver alguns problemas que surgiram no meio do processo de criação do jogo.

3.4 O branch inalcançável

Durante a implementação do ataque ocorreu uma situação onde um erro ocorria que impedia a compilação do código. Este erro apontava que o branch estava fora do alcance estando a mais de 12 bits (do imediato) de distância, por este motivo, tivemos que implementar o retorno da função de criar e destruir blocos de uma forma alternativa.

"Error in D:Projeto_ISC/JOGO/src/print_pontos.s line 60: Branch target word address beyond 12-bit range"

4 Resultados obtidos e conclusão

No final, conseguimos implementar perfeitamente 100% do que foi pedido. Apesar de não ser um jogo complexo e visualmente deslumbrante, o resultado foi mais que satisfatório, visto que, antes de começarmos a produção, não estávamos com altas esperanças para o polimento desse projeto.

imagem 4: tela de fim de jogo

Concluindo, com todos os erros inesperados, a inexperiência com uma linguagem rígida como Assembly e o tempo disponível para programar o projeto, o resultado final trouxe muito conhecimento para o futuro, horas sem sono, mas também divertimento, visto que cada passo na escritura do código era uma pequena vitória.
medidos esforços para finalizar o que foi possível até a data final.

Assim, a conclusão do projeto foi um jogo com duas fase, com menu e música implementada, movimentação fluida e colisão correta com os sprites presentes na tela. Apesar das dificuldades, a equipe conseguiu produzir sua versão "beta" do jogo Bad Santa, baseado no jogo Bad Ice Cream.

Bibliografia

(1) NITROME WIKI - https://nitrome.fandom.com/wiki/Bad_Ice-Cream

(2) DAVI PATURI - Renderização dinâmica no Bitmap Display, YouTube, 12 maio 2021.

(3) VICTOR LISBOA - Como importar músicas do Hooktheory para tocar no RARS, YouTube, 4 maio 2021.

(4) DAVI PATURI - Repositório com o código para extrair midi do site Hooktheory, 19 outubro 2020.

(5) LEO RIETHER - Release v2.3 · LeoRiether/FPGRARS, GitHub, 18 setembro 2021.

(6) REPOSITÓRIO - Jogo-ISC-Assembly-Risc-V, GitHub, 10 dezembro 2023

(7) BING AI - https://www.bing.com/images/create, Microsoft, 2023

(8) PIXILART - https://www.pixilart.com.