The Assembly Gauntlet

Manuella dos Santos Araujo
Luca Heringer Megiorin
Eduardo Pereira de Sousa

University of Brasília, Dept. of Computer Science, Brazil

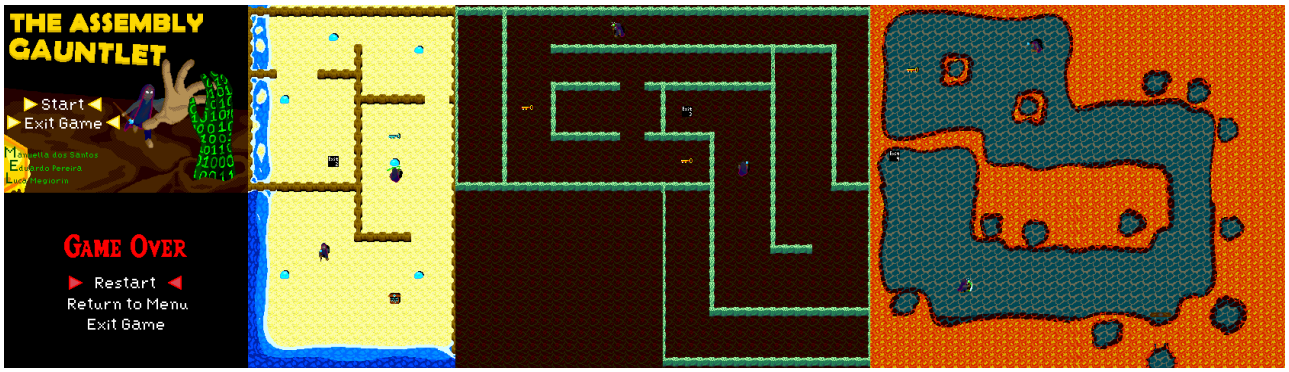

Figura 1: Menu principal, menu de morte e mapas do jogo

Resumo

Neste artigo será detalhado o processo de criação de “The Assembly Gauntlet”, que é uma releitura do jogo Gauntlet (1985 - Atari Games), sendo desenvolvido em Assembly com a ISA RISC-V 32 no simulador RARS por Manuella dos Santos Araujo, Luca Heringer Megiorin e Eduardo Pereira de Sousa. Este projeto tem como objetivo implementar os conhecimentos adquiridos ao longo do semestre através da disciplina Introdução aos Sistemas Computacionais (ISC), ministrada pelo professor Marcus Vinícius Lamar na Universidade de Brasília.
Palavras-chave: Assembly, Gauntlet, Jogo, RARS, RISC-V.

1 Introdução

Gauntlet é um jogo eletrônico desenvolvido pela Atari Games e lançado em 1985. O jogo é um arcade no estilo Hack and Slash, e foi um dos pioneiros nos jogos com a opção de multiplayer em seu estilo. O jogo possui diversos elementos ligados ao RPG (Role-Playing Game), e consiste em diversas fases que se passam em um labirinto contendo quatro personagens jogáveis, cada personagem tendo habilidades e mecânicas únicas para os diferentes tipos de ataque e de inimigos. O Gauntlet ganhou diversos títulos subsequentes que inovaram saindo do 2D para o 3D.

The Assembly Gauntlet sendo uma releitura teve como base o Gauntlet na jogabilidade e no visual do jogo e a equipe tinha como objetivo implementar essas mecânicas, mas com algumas mudanças, tais como colocar um personagem original, mapas estáticos, porém contendo uma sequência de continuidade, e com essas mecânicas implementar junto com as sprites feitas do zero uma ambientação baseada nos elementos, sendo eles: água, terra e fogo.

2 Metodologia

O projeto foi simulado no RARS na ISA RISC-V 32, sendo programado em Assembly, com o objetivo de obter aprendizado e conhecimento de uma linguagem de programação de baixo nível. Foram utilizadas as ferramentas disponibilizadas pelo RARS, como o Keyboard MMIO que serviu para reconhecer inputs do teclado e o Bitmap Display, que serviu para realizar a renderização do jogo e a seguir será relatado o processo passo a passo do desenvolvimento do jogo.

2.1 Personagem

O primeiro passo para começar a desenvolver o jogo foi a criação do personagem. O design do personagem foi de um mago com seu cajado, sendo feito no aplicativo Krita. Logo após sua criação o segundo passo é feito com o auxílio do teclado de memória (Keyboard MMIO), onde o personagem se move conforme a sprite renderizada.

Figura 2: Sprites do personagem principal

2.2 Animação

A animação foi implementada utilizando uma técnica de sprites, onde uma série de imagens representando diferentes estados e direções do jogador são combinadas em um único arquivo. Cada sprite do jogador possui um tamanho de 24x24 pixels, e ao ser renderizada, a animação avança pulando 24 linhas ou 48 linhas, dependendo da sprite específica que deve ser exibida. Essas sprites são organizadas de acordo com um índice que indica a continuidade da animação.

A movimentação e a animação são controladas por meio de arquivos contendo diversas sprites, associadas a um contador e um índice. Ao modificar esses valores, diferentes partes da imagem são selecionadas para a renderização, criando a sensação de animação do jogador em movimento.

Essa abordagem é amplamente utilizada em desenvolvimento de jogos e outras aplicações gráficas para criar animações fluidas e realistas de personagens em movimento. O uso de sprites e o gerenciamento adequado de índices e contadores permitem uma transição suave entre as diferentes imagens, criando a ilusão de movimento contínuo.

2.3 Colisão

A colisão estática é implementada utilizando uma imagem paralela do mapa, onde as diferentes cores indicam as interações do jogador com o ambiente. Por exemplo, a cor verde permite que o jogador passe livremente, enquanto a cor laranja indica que o jogador não pode passar livremente, mas seus projéteis podem atravessar. A cor azul representa paredes intransponíveis, onde nada pode passar, e a cor rosa indica que o jogador pode passar, mas será transferido para outro nível do jogo.

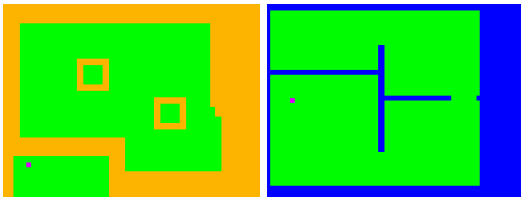

Figura 3: Mapa do nível 3 e 1 no formato de hitbox

Por outro lado, a colisão dinâmica é aplicada a elementos não fixos no mapa, como portas, chaves e inimigos. Nesse caso, é utilizada a lógica de dois retângulos que se sobrepõem. Quando ocorre uma interseção entre esses retângulos, significa que o jogador ou outro objeto está colidindo com algo. Por exemplo, se as coordenadas do lado esquerdo de um retângulo são maiores do que as coordenadas do canto direito do outro retângulo, não há colisão. Além disso, se o retângulo estiver abaixo da coordenada mais baixa do outro no eixo Y, também não há colisão.

Uma vez detectada a colisão, são acionados diversos processos dependendo das entidades envolvidas. Por exemplo, no caso de uma colisão entre o jogador e um inimigo, o jogador sofre dano. No entanto, se for a colisão entre o projétil do jogador e o inimigo, o inimigo é eliminado.

Essa abordagem de colisão estática e dinâmica é comum em jogos e aplicativos que envolvem movimentação e interação entre personagens e elementos do ambiente.

2.4 Música

O site Hooktheory foi utilizado para a formação da música do jogo, já que nele são mostradas todas as notas das músicas, aparecendo sua duração e afinação. Após a música ser escolhida, a outra parte do processo era feita por um programa em Javascript feito por Davi Patury, onde o programa já traduz automaticamente as notas para Assembly, tornando mais fácil e rápido o processo ao invés de ser feito manualmente nota por nota. Depois desses processos serem concluídos, era apenas preciso implementar as notas no código do jogo, assim rodando no RARS.

2.5 Inimigos

Os inimigos são controlados por dois parâmetros principais: sua posição no ambiente do jogo e sua quantidade de vida. Além disso, há um terceiro parâmetro que determina se eles devem ser renderizados na tela ou não.

A movimentação dos inimigos que andam segue um padrão pré-definido, que consiste em alternar entre a velocidade e o número de passos que eles devem dar na direção horizontal (eixo X) e a velocidade e o número de passos na direção vertical (eixo Y). Esses padrões são definidos em uma lista, permitindo a criação de inimigos com diferentes trajetórias de movimentação.

A frequência com que esses padrões são executados é determinada pela música do jogo. Através de análises musicais, é possível identificar a quantidade de vezes que um padrão deve ser repetido. Dessa forma, os inimigos seguem esses padrões de movimentação em sincronia com a música, criando uma experiência de jogo dinâmica e coreografada.

Por outro lado, os inimigos atiradores possuem um funcionamento semelhante, mas, em vez de se moverem pelo cenário, eles disparam projéteis em uma determinada direção e quantidade predefinida. Esses inimigos também seguem padrões de ataque em consonância com a música, aumentando o desafio do jogo e criando momentos de interação mais intensos com o jogador.

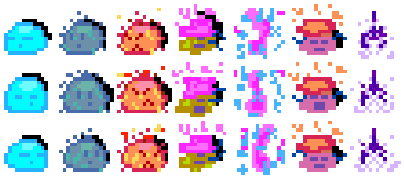

Figura 4: Inimigos e seus projéteis

3 Resultados Obtidos

O desenvolvimento do jogo foi um processo longo e trabalhoso, e houve algumas dificuldades notáveis. A maior de todas e a que mais dificultou foi o fato de que a depuração em Assembly é muito trabalhosa, pois a linguagem não é muito prática para essa função, visto que para isso é necessário desacelerar o programa e ir executando linha por linha até encontrar o problema. Outro obstáculo que atrapalhou o desenvolvimento foi a dificuldade em si da linguagem de programação Assembly. Por possuir um nível extremamente baixo, o Assembly é de alta complexidade de compreensão e de utilização, além de que as instruções são muito mais extensas do que seriam em outra linguagem. Um dos exemplos é o fato de os branches apenas realizarem saltos de 12 bits, o que é limitante em um código longo (aproximadamente 5359 linhas).

Apesar de todas as limitações fornecidas e das dificuldades percorridas, foi possível realizar o projeto em pouco mais de um mês, com resultados mais que satisfatórios. O jogo contém música, contador de pontos, de vida e de fases, inimigos com comportamento fácil de modificar, ataques do jogador e de inimigos e ambientação personalizada.

4 Conclusão

Neste artigo foi abordado sobre a criação da releitura do jogo Gauntlet na linguagem de programação Assembly na ISA RISC-V 32 e conclui-se que esse trabalho serviu de grande aprendizado para a matéria de ISC (Introdução a Sistemas Computacionais) e rendeu experiência em linguagem de baixo nível. Foram cumpridos todos os objetivos requisitados.

5 Referências

https://youtu.be/2BBPNgLP6_s - Referente a renderização.
https://github.com/tilnoene/celeste-assembly - Referente a colisão.
https://www.geeksforgeeks.org/find-two-rectangles-overlap/ - Referente a colisão.
https://www.hooktheory.com/theorytab - Referente a música.
https://gist.github.com/davipatury/cc32ee5b5929cb6d1b682d21cd89ae83 - Referente a música.
Aula de laboratório 1 ministrada por Dr. Marcus Vinícius Lamar.