Universidade de Brasília
Introdução a Sistemas Computacionais - 2022.1

Releitura do jogo Pulsar utilizando assembly RISC-V

Luiz Eduardo Reis – 221017088
Mateus Oliveira Santos - 221029150

Resumo

O projeto apresentado teve por finalidade programar, utilizando instruções da ISA RISC-V, um jogo para computador inspirado no jogo *Pulsar*, para demonstrar e consolidar os conhecimentos obtidos durante o semestre na matéria Introdução a Sistemas Computacionais (ISC). Abordaremos, neste artigo, o processo de desenvolvimento do jogo, problemas enfrentados, bem como as soluções encontradas para os problemas que surgiram.

Introdução

O jogo *Pulsar,* lançado pela *Sega Corporation*, em 1980, é um jogo de plataforma, no qual o jogador controla um tanque de guerra 2D e precisa percorrer um labirinto, enfrentar inimigos e coletar chaves, para que consiga sair do labirinto e vencer a fase. Tudo isso sem perder vidas e sem deixar o combustível acabar.

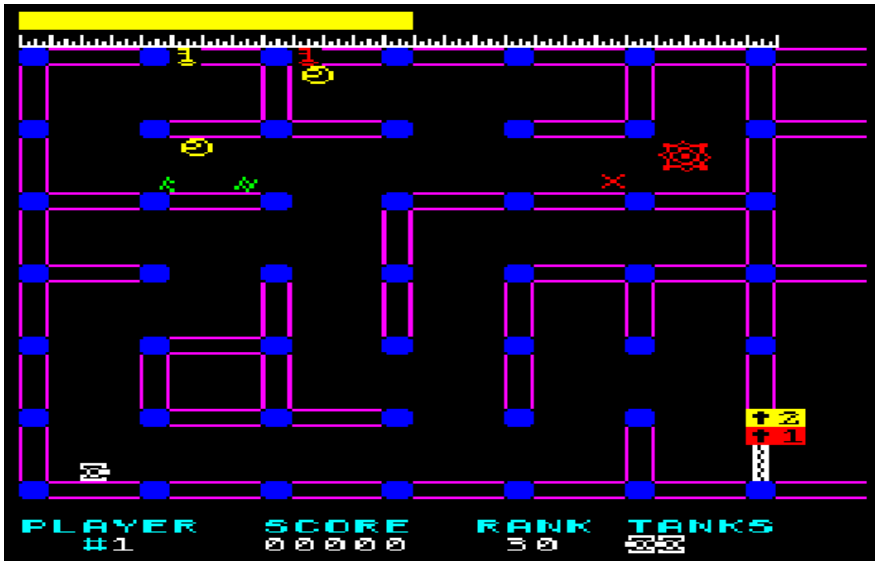

Figura 1. Fase do jogo *Pulsar.*

Nosso grupo buscou criar uma releitura do jogo que, ao mesmo tempo, mantivesse recursos característicos do jogo, mas também modificasse alguns recursos da forma que considerássemos mais interessante.

Os recursos implementados foram principalmente aqueles requeridos nas especificações do projeto: 1) interface gráfica (*Bitmap Display*, 320x240, 8 *bits/pixel*); 2) interface com teclado (*Keyboard and Display MMIO simulator);* 3) interface de áudio MIDI (ecalls 31, 32, 33); 4) Duas fases com *layouts* diferentes; 5) animação e movimentação do jogador e seus ataques; 6) colisão com as paredes; 7) Implementação da condição de vitória por coleta de chaves; 8) implementação da condição de derrota (combustível e vida); 9) dois tipos de inimigos diferentes que se movimentam e interagem com o jogador (tiros, colisão); 10) menu com informações de vida, combustível e pontuação; 11) música e efeitos sonoros.

Para a construção do código e para executar o jogo, inicialmente utilizamos o programa RARS – que simula um processador RISC-V de 32 bits. Posteriormente, o RARS começou a ser um problema, pois, devido ao tamanho e complexidade crescentes do código, o *software* começou a travar muito.

Metodologia

O primeiro passo para o desenvolvimento do jogo foi tentar mostrar uma imagem na tela e tentar fazê-la se mover. Nesse primeiro momento, as monitorias e tutoriais em vídeo feitos pelos colegas monitores ajudaram muito. Por meio dos vídeos, adaptamos os processos ensinados e implementamos, no nosso jogo, um procedimento que mostra imagens na tela. Também por meio das monitorias, aprendemos a converter imagens em arquivos ".data", que são utilizados para mostrar imagens no *Bitmap Display* do RARS. Dessa maneira, aprendemos a imprimir qualquer imagem na tela. Outro processo ensinado pelos monitores e que foi de grande ajuda no projeto foi o de converter músicas em código para serem tocadas no RARS.

Após conseguirmos implementar a renderização dinâmica, implementamos o loop principal do jogo, no qual primeiro são feitos todos os devidos cálculos e procedimentos inerentes ao andamento do jogo e, então, todas as informações são renderizadas na tela.

Seguimos o desenvolvimento pela parte de colisões. Criamos, na memória, uma variável que armazena a posição de cada obstáculo e inimigo das fases do jogo e, a cada movimentação que o jogador faz, um procedimento avalia se há ou não colisão, e retorna de forma correspondente. Dessa forma, o personagem só se movimenta o quanto possível, caso haja uma parede ou um inimigo na frente.

Começamos, então, a desenvolver o sistema ataque do personagem. O jogador consegue disparar um projétil com alcance de 40 pixels para qualquer lado desejado. Esse recurso foi implementado criando uma variável na memória que guarda a posição do projétil. Inicialmente, o projétil é armazenado fora do mapa. Quando o jogador aperta um dos botões de tiro, a posição do projétil é definida para a posição atual do personagem e, daí, entra em ação um procedimento que movimenta o projétil pela distância definida.

Cada imagem utilizada (figura do jogador, inimigos, paredes) foi criada no *software* "paint.net" e desenhada manualmente. O jogador é uma nave em formato de losango (ao invés de um tanque, como na versão original do jogo) e os inimigos têm o formato inspirado em um obstáculo do jogo *Mario Kart* e o rosto inspirado num *creeper* - do jogo *Minecraft.*

Para a movimentação do personagem, utilizamos um procedimento que verifica a tecla pressionada e atualiza a posição do jogador para a posição correta. Já para a movimentação dos inimigos, utilizamos um contador de iterações no loop principal do jogo. Dessa forma, definimos um certo número de iterações para que, a cada ciclo desse determinado tamanho, o inimigo se movesse um pouco. Além disso, a colisão com qualquer inimigo causa dano.

Implementamos dois tipos de inimigos: o primeiro apenas se movimenta para atrapalhar a movimentação do jogador pelo mapa, e colidir com esse inimigo causa 1 de dano. O segundo tipo de inimigo atira projéteis que são fatais (ser acertado tira toda a vida do jogador). Os tiros do inimigo foram implementados da seguinte maneira: o mesmo contador de iterações que é utilizado na movimentação dos inimigos é avaliado por um procedimento. A cada número definido de iterações, um disparo é realizado.

As principais dificuldades enfrentadas decorreram de dificuldades em lidar com o RARS e utilizar assembly, pois não tínhamos quase nenhuma experiência com programação em linguagens de baixo nível. Também, por vezes o RARS apresentava erros de compilação e travamentos sem solução aparente. Nesse momento, começamos a usar o programa FPGRARS e todos os problemas de desempenho cessaram. Portanto, para terminar o jogo, utilizamos o RARS para editar o código e o FPGRARS para executar o jogo e fazer testes.

Por fim, criamos a tela inicial e implementamos efeitos sonoros no jogo. Colocamos uma música ao início e ao final. Chegou a ser implementado um efeito sonoro nos tiros do inimigo, porém a frequência de tiro é alta e ficamos apenas com um ruído repetitivo, portanto resolvemos remover esse recurso.

Para ganhar, o jogador deve, com uma quantidade limitada de combustível e de pontos de vida, derrotar os inimigos, coletar todas as chaves e entrar no portal que se abre quando as chaves são todas coletadas. O jogador deve fazer isso nas duas fases do jogo. Para monitorar o seu combustível, vida, chaves e pontos, o jogador tem disponível um menu na parte de baixo da tela, com todas essas informações. Para esse menu, foram utilizados os macros criados por colegas, que imprimem inteiros e *strings* no *Bitmap Display*.

Resultados obtidos

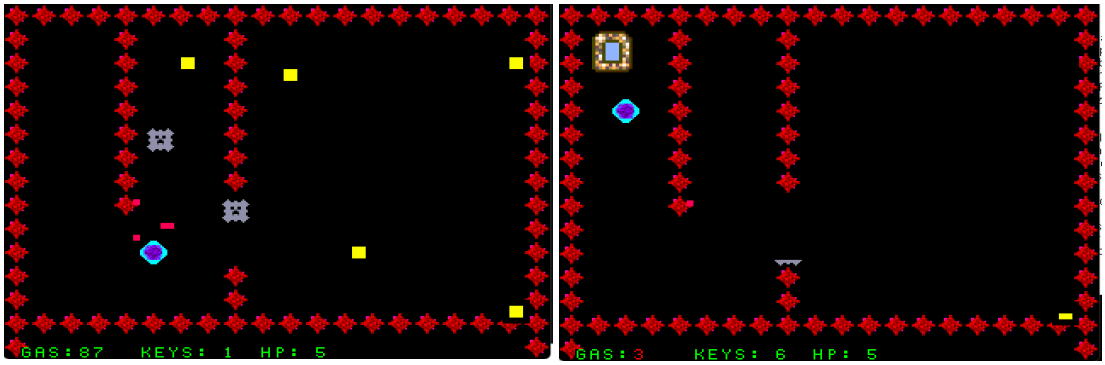

Figura 2. Tela do jogo desenvolvido no início da fase (à esquerda) e ao final da fase, com o portal já aberto (à direita)

Ao final do projeto, o grupo conseguiu implementar todos os recursos requeridos, mesmo que com certa dificuldade ao lidar com o RARS e com a linguagem assembly RISC-V.

Conclusão

O desenvolvimento do jogo foi difícil e trabalhoso por causa da falta de experiência do grupo com linguagens de programação de baixo nível, além do pouco conteúdo disponível na internet sobre o assunto. Dessa forma, ao contornar as adversidades, ganhamos muita experiência de desenvolvimento em equipe e de funcionamento de *software* e *hardware.*

Referências Bibliográficas

LISBOA, VICTOR. **Como importar músicas do Hooktheory para tocar no RARS**. Youtube, 4 maio 2021. Disponível em: https://www.youtube.com/watch?v=mBmOkyqejHU Acesso em: 15 set. 2022

LISBOA, VICTOR. **Convertendo imagens pra usar no RARS**. Youtube, 5 maio 2021. Disponível em: https://www.youtube.com/watch?v=tx9t2hGWWko Acesso em: 15 set. 2022.

PATURI, DAVI. **RISC-V RARS: renderização dinâmica no Bitmap Display**. Youtube, 12 maio 2021. Disponível em: https://www.youtube.com/watch?v=2BBPNgLP6_s Acesso em: 15 set. 2022.