Projeto Pulsar

Grupo 08

Guilherme Fornari Leonel
221017023

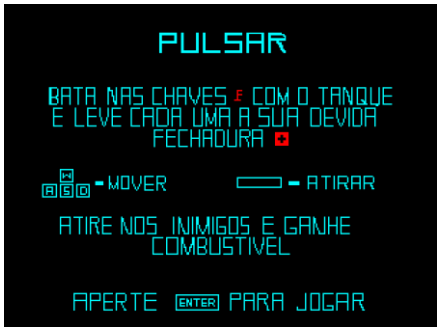
Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Brasil

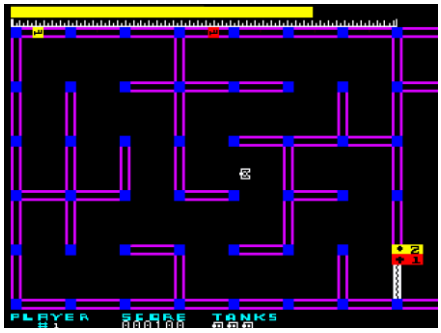
Pedro Avila Beneveli
221001972

Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Brasil

Vinícius da Silva Araujo
221001981

Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Brasil





***Resumo*—Este artigo relata o processo de criação de um jogo inspirado em Pulsar, arcade lançado em 1981 pela empresa SEGA, muito conhecida por sua participação futura na "Guerra dos Consoles" contra a Nintendo. O projeto visa a contemplação do conteúdo visto nas aulas de Introdução aos Sistemas Computacionais, ministrada pelo professor Marcus Vinícius Lamar, na Universidade de Brasília, por meio da utilização da ISA RISC-V 32 bits. Este artigo, além de mostrar o resultado obtido, relata ainda as dificuldades encontradas além das técnicas utilizadas.**

1. INTRODUÇÃO

1.1 - O Jogo Original

Pulsar é um jogo eletrônico, lançado em 1981 pela SEGA/Gremlin, em que o jogador controla um tanque em um labirinto. O objetivo do jogo é pegar as chaves espalhadas no mapa e inseri-las nas portas de cores correspondentes. No caminho, haverá inimigos que tentarão impedir o jogador de realizar tal feito – seja por meio de tiros direcionados ao player, seja por obstrução de passagem –, além de haver uma preocupação com a quantidade de gasolina que o jogador possui, uma vez que, com o tempo, ela vai acabando e, se acabar, o jogador perde.

1.2 - Objetivos

O objetivo era criar um jogo, com a mesma identidade visual do jogo original, mostrada na Figura 1, mas com algumas poucas mudanças pontuais nas mecânicas da gameplay, de forma a trazer uma divergência com o modelo. No mais, a ideia é mais uma apreciação da obra, através de uma releitura fidedigna do material original.

1.3 - Requisitos

Para o projeto, eram requisitadas na implementação do jogo as seguintes questões: duas fases com layouts diferentes, animação e movimentação do jogador e seus ataques, colisão com as paredes, condição de vitória por coleta de chaves e de derrota por falta de combustível ou de vida, dois tipos de inimigos que se movimentam e interagem com o player, um menu com informações de vida, combustível e pontuação e, por fim, música e efeitos sonoros. As sessões deste artigo irão descrever todo o processo de elaboração do jogo pelos integrantes do grupo.

Fig. 1. Pulsar (1981 – SEGA).

2. METODOLOGIA

Para fim de auxiliar na execução do programa e permitir a boa performance do jogo em vários tipos de computadores, foi utilizado o programa FPGRARS (*Fast Pretty Good RISC-V Assembly Rendering System*). E para a edição do código, foi utilizado também o RARS e o Neovim.

2.1. Sprites e Mapa

As imagens criadas para este jogo necessitavam a transformação de seu formato original *bmp* para o formato *mif*, em que as cores de cada pixel são expressas no formato RGB 8 bits, com 2 bits expressando a intensidade da cor azul, 3 bits a intensidade da cor verde e mais 3 bits expressando a intensidade da cor vermelha. Para isso, foi utilizado o código de conversão disponibilizado pelo professor Marcus Vinícius Lamar, que gera um arquivo em formato *mif* e um em formato *bin* a partir de uma imagem em formato *bmp*.

Para a criação dessas imagens em formato *bmp* foi utilizado o software *Photoshop CS6*, utilizando-se como referência as imagens do jogo original para se obter um visual semelhante ao jogo lançado pela SEGA. No total, foram criadas 2 fases e 16 sprites.

No código em Assembly, foram utilizadas 3 chamadas de sistema, uma para obter o descritor do arquivo, outra para guardar um número especificado de bytes do arquivo na memória e a última para fechar o arquivo. Com isso em mente, foram criados dois métodos, um para exibir o mapa e outro para guardar os bytes representando a cor em um espaço reservado na memória de dados. No caso do mapa, os bytes são guardados diretamente na memória de vídeo VGA, exibindo o mapa totalmente após a *ecall*.

Além disso, para garantir que todos os elementos no mapa serão guardados na memória de vídeo antes de serem exibidos ao jogador, também utilizamos os frames 0 e 1 disponibilizados na ferramenta bitmap do RARS e ficamos alternando qual é exibida, e qual prepara os próximos elementos.

2.2. Colisão

A colisão do jogo foi realizada a partir de uma matriz de números, apresentada na Figura 2, em que cada número simboliza um elemento presente no mapa, e as situações de colisão serão comparações entre esses números. Os números da matriz representam:

- 0 = Espaço vazio
- 1 = Parede
- 2 = Jogador
- 3 = Chave vermelha
- 4 = Chave amarela
- 5 = Porta vermelha
- 6 = Porta amarela
- 7 = Tiro
- 8 = Inimigo

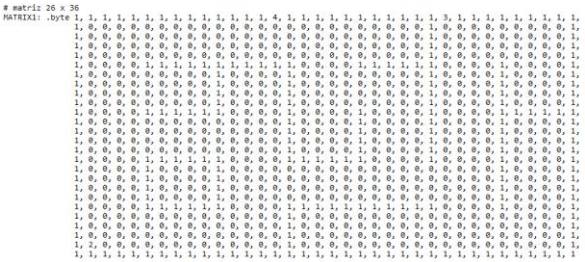


Fig. 2. Matriz de Colisão do estágio 1

2.3. Movimentação do Jogador

A movimentação do jogador depende, antes de tudo, do input recebido pelo jogador para que ele se movimente na direção desejada. Para isso, utilizamos a ferramenta disponibilizada pelo RARS chamada *Keyboard and Display MMIO Simulator*, que permite que nós coletemos o código ASCII da tecla pressionada pelo jogador apenas quando o teclado for pressionado.

Com isso em mente, utilizamos a instrução do Assembly RISC-V que permite que ocorra uma ramificação do código quando dois valores são iguais. Por fim, decidimos utilizar as teclas ‘w’ para se movimentar para frente, ‘a’ para se movimentar para a esquerda, ‘s’ para se movimentar para trás e ‘d’ para se movimentar para a direita.

Após a direção ser selecionada pelo jogador, uma verificação ocorre na matriz para garantir que o espaço que ele quer ocupar esteja vazio, ou se deve ocorrer alguma mudança no mapa e/ou estado do tanque. Depois, o código altera variáveis guardadas na memória que determinam parametros utilizados no método *PRINT*, que são: a orientação do tanque, a posição do tanque em que ele será exibido e a posição

antiga do tanque em que será pintada de preto a imagem antiga do tanque.

2.4. Ataque do Jogador

O jogador realiza um ataque ao pressionar a tecla de espaço no teclado. Ao pressionar essa tecla, o programa determina a direção do ataque com base na direção do jogador ao atacar. Com isso, o jogo determina algumas variáveis:

- Próxima posição da matriz de colisão a ser ocupada pelo tiro;
- Próxima posição em que o tiro será exibido;
- Presença do tiro, para evitar que o jogador crie muitos tiros de uma vez;
- Orientação do tiro.

Após criar o tiro, um método no loop de jogo chamado *MOVE_BULLET* verifica se o tiro deve se mover. Essa determinação depende de 2 variáveis, a quantidade de iterações do código para evitar que o ataque se movesse em uma velocidade maior que algo perceptível, e o número de espaços da matriz que o tiro percorreu. Além disso, determinou-se que o ataque também deveria sumir ao encontrar algum obstáculo. Uma particularidade da parte de impressão do código que pode ser uma causa de erros é que primeiro ocorre a impressão do ataque e a atualização de sua posição na matriz, e depois que há mudanças na posição, deixando a posição na tela e a posição na matriz prontas para sua próxima impressão, contrário do que foi feito com o jogador.

2.4. Música e efeitos sonoros

A música e o efeito sonoro foram implementados utilizando principalmente a chamada de sistema 31, a chamada *MidiOut*, em que o sistema operacional toca a nota solicitada, mas faz com que o programa continue rodando normalmente. Como o efeito sonoro é composto de uma única nota, essa implementação foi simples: apenas realizar uma *Syscall* (chamada de sistema) na situação necessária, no caso a morte do jogador.

Já no caso da música tocar durante o jogo, foi necessária uma implementação mais complexa, com o uso da *Syscall* 30, de tempo, que fornece nos registradores a0 e a1 o tempo em milissegundos a partir do dia 1° de janeiro de 1970. Com isso, conseguimos comparar o tempo obtido no momento em que a nota foi tocada, e fazer com que a próxima nota toque apenas quando a diferença entre o tempo atual e o tempo no qual a última nota foi tocada for maior ou igual a duração da nota em milissegundos, informação guardada na memória de dados. Por fim, a música escolhida para o jogo foi "*Facing a Huge Reaction*" do jogo Metroid Fusion (2002 – Nintendo), ilustração mostrada na Figura 3.

Fig. 3. Metroid Fusion OST 09 Facing a Huge Reaction, https://www.youtube.com/watch?v=V7UoIvtsXZg acessada em 06/10/22.

2.5. Condição de vitória

A vitória deve ocorrer quando o jogador coletar e entregar todas as chaves, assim possibilitando a progressão para o próximo estágio, ou o reinício do jogo. Isso é feito por meio da mudança de variáveis de estado, que vão determinar se certos trechos de código serão lidos pelo programa.

A primeira variável determina se o jogador está carregando a chave ou não. Ela vai determinar a cor do tanque, para ficar a mesma da chave, vai atualizar a matriz para substituir a antiga posição da chave por uma parede e para fazer a porta daquela chave aparecer na matriz para o jogador interagir com ela. Além disso, uma outra variável que faz com que a fase seja renderizada novamente é atualizada, fazendo com que o mapa seja desenhado de novo para deixar o portão visível ao jogador. Outro ponto importante é que existe outra variável de estado que impede que o jogador colete uma segunda chave ao estar carregando uma simultaneamente, evitando um erro no jogo.

A segunda variável determina se o jogador entregou uma chave, assim completando parte da condição de vitória. Isso irá retirar a porta da matriz da tela, e se tiver sido a última chave a ser entregue, irá fazer com que haja uma pequena animação de vitória para criar uma transição entre as fases.

3. RESULTADOS OBTIDOS

De uma maneira geral, o projeto foi satisfatório. Inicialmente, o grupo tentou utilizar o RARS para a execução dos códigos do jogo, mas com o tempo, foi se tornando inviável e a mudança para o FPGRARS foi certeira, resultando em uma melhora abrupta de desempenho nos testes.

O jogo manteve sua identidade visual característica, clássica do jogo original de 1981; o jogador consegue movimentar-se livremente no mapa por intermédio do tanque, para o qual não é permitido atravessar paredes, logo, o sistema de colisão implementado teve grande êxito; o jogador, ao colecionar uma chave, aciona automaticamente a porta de cor correspondente, e, ao carregá-la até tal porta, destrava-a e libera a chave;

quando o jogador leva as duas chaves para as portas, observa uma animação do tanque se dirigindo para a porta aberta e avança de fase; o tanque consegue atirar; há um efeito sonoro ao morrer; existem inimigos pelo mapa e eles possuem movimentação própria; o jogo apresenta um menu com as especificações necessitadas em *1.3 - Requisitos*; há música durante todo o jogo, o que foi bem desafiador para os membros, tendo em vista a problemática de interromper o andamento do jogo ou atrapalhá-lo de alguma forma, o que foi solucionado após muito esforço.

Infelizmente, houve pontos em que o grupo não obteve o mesmo êxito. O ataque do jogador deixou a desejar, apresentando incosistência em seu alcance; há apenas um tipo de inimigo, o qual não morre e não ataca. No mais, o inimigo acabou se tornando um obstáculo móvel. A implementação dos inimigos acabou ficando para o final pelo esforço gasto nos outros setores do jogo, e foi imensamente prejudicada por isso, visto que algo feito em um curto espaço de tempo não possui a mesma qualidade de algo realizado com tempo e primazia.

4. CONCLUSÃO

Portanto, observa-se que o grupo conseguiu atender à maioria dos requisitos com eficiência, apesar das dificuldades encontradas ao longo de todo o processo de desenvolvimento do jogo. Tivemos problemas em alguns momentos, mas todos de certa forma serviram para engrandecer o trabalho apresentado e enriquecer nossos conhecimentos. Trabalhar com o Assembly RISC-V 32 bits sem dúvidas foi um grande desafio para todos, devido ao extremo baixo nível da linguagem. Enfim, é importante salientar o apoio e ajuda dos monitores, em especial do monitor Marcos Alexandre, o qual todas as sextas-feiras esteve presente, sempre atencioso e prestativo com os integrantes do grupo. Além disso, o server no Discord de ISC também apresentou alguns materiais auxiliares que ajudaram no desenvolvimento do projeto.

Outro ponto positivo desse projeto foi o aprendizado de recursos básicos do site GitHub, que auxiliou muito na reta final do projeto para facilitar o trabalho de todos em partes diferentes do código ao mesmo tempo. Com isso, conseguimos economizar bastante tempo na junção dos códigos dos partipantes do grupo para criar o produto final.

Logo, mesmo diante de todas as adversidades, concluímos que o saldo final foi bastante positivo, tanto pelo resultado final, quanto pelo processo de aprendizagem obtido.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FPGRARS from LeoRiether. Disponível em: https://github.com/LeoRiether/FPGRARS
[2] Convertendo imagens pra usar no RARS from Victor Lisboa. Disponível em: https://youtu.be/tx9t2hGWWko
[3] RISC-V RARS - Renderização dinâmica no Bitmap display from Davi Paturi. Disponível em: https://youtu.be/2BBPNgLP6_s
[4] Pulsar : Sega : Free Borrow & Streaming : Internet Archive. Disponível em: https://archive.org/details/arcade_pulsar
[5] Arcade Game: Pulsar (1981 Sega). Disponível em: https://youtu.be/BTbygumOaWU
[6] Pulsar – Sega Retro from SagaRetro.org. Disponível em: https://segaretro.org/Pulsar
[7] Hooktheory: Create amazing music. Disponível em: https://www.hooktheory.com/
[8] Como importar músicas do Hooktheory para tocar no RARS from Victor Lisboa. Disponível em: https://youtu.be/mBmOkyqejHU