Universidade de Brasília - Introdução aos sistemas computacionais - 2023.1

Gauntlet – Arcade

ISA RISC-V

Emerson Luiz Cruz Junior – 231003531

João Henrique Jácomo Lemes – 231018893

José Neto Souza de Lima - 231018955

Resumo:

Este artigo apresenta o processo utilizado pela equipe para atender aos requisitos e solucionar problemas de um projeto de aplicativo. O projeto consiste em implementar uma versão do modo arcade do Gauntlet afim de aplicar os conhecimentos adquiridos acerca da linguagem Assembly – RISC-V.

Introdução:

O propósito desse artigo é relatar as dificuldades e soluções encontradas para a implementação de um jogo utilizando as mecânicas do modo arcade do jogo Gauntlet, desenvolvido em 1985 pela Atari Games. Neste modo o player tem o objetivo de acumular a maior quantidade de pontos em um período de tempo (vida), passando de nível ao encontrar a chave para a sala onde se encontra a saída. A equipe atribuiu uma temática futurística para a versão do Gauntlet.

Para a implementação do modo arcade do jogo Gauntlet de 1985 usou-se como ferramenta o **instrucion set architecture RISC-V**. Para o uso do RISC-V foi necessário a utilização da IDE RARS (1).

Figura 1. Capa do jogo *Gauntlet*.

Figura 2. imagem do jogo *Gauntlet*.

Metodologia:

Para inserir o jogo no contexto da matéria a equipe alterou os personagens de modo que os mesmos são personalidades que fizeram parte do semestre da matéria Introdução aos Sistemas Computacionais. Para iniciarem os avanços no projeto a equipe procurou seguir o roteiro das especificações:

Movimentação e colisão:

Para seguir os requisitos do projeto foi necessário iniciar com a movimentação do personagem, para isso foi utilizado o vídeo disponibilizado pelos monitores passados (2) explicando acerca do funcionamento do bitmap display, com isso ficou mais fácil fazer com que “ecos” deixados pelo personagem no mapa fossem apagados. A movimentação consiste em todas as vezes em que se pressiona as teclas de movimentação (‘W’, ‘A’, ‘S’, ‘D’), guarda se a posição atual do personagem, avança a posição do mesmo na direção desejada em 16x16 pixeis (o tamanho do personagem) e após isso coloca-se a imagem do chão na posição guardada, porém para avançar com os ataques do player era preciso desenvolver as dinâmicas de colisão.

Para a colisão foi atribuída a abordagem de fazer uma hitbox, inspirando-se no projeto do jogo Celeste de OAC (3), inicialmente a hitbox se encontrava apenas na memória, e as comparações consistiam em checar qual a cor na posição do movimento desejado, se era a cor de uma parede (0xffffffC8) o código não continuaria ao processo de atualizar a posição, porém essa abordagem apresentava algumas futuras limitações para os ataques e os inimigos que se movimentavam. Portanto, uma alteração fez-se necessária. Agora somente um dos frames é mostrado na tela, enquanto o outro fica encarregado de fazer o papel da hitbox dinâmica contando agora com as mesmas atualizações feitas para a movimentação.

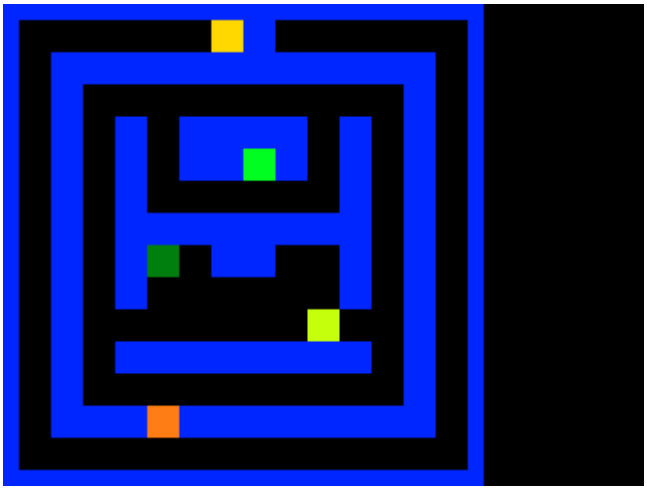

Figura 3. Hitbox da fase 1 do projeto.

Sistema de chaves e abertura de portas:

O próximo passo foi a inserção de um sistema de chaves e portas, para isso adicionou-se à hitbox as cores da chave (0x00000037) e da porta (0x0000001f). Além disso, adicionou-se uma comparação para a coleta da chave e para a porta, reaproveitando-se da comparação das paredes, porém, se for a chave, prossegue com a movimentação e se altera um registrador para marcar que o player coletou a chave, dessa forma na comparação da porta adiciona-se uma condicional para determinar que se o player estiver com a chave ele deverá prosseguir na direção e se não, não prosseguir.

Ataques do player:

Com isso, o próximo avanço foi as três opções de ataques para os quatro personagens. A primeira a ser desenvolvida foi a dos projéteis que ao pressionar a tecla “espaço”, designada como o botão de ataque,

marca-se em um registrador que o projétil já foi lançado e de acordo com o posicionamento do seu sprite (cima, baixo, esquerda, direita) no momento do pressionamento da tecla, é definida a direção do projétil.

A partir de uma comparação logo no próximo lugar que o projétil ocuparia para checar se era uma parede ou uma porta que não estivesse aberta (ou seja, ainda estivesse com a cor 0x0000001f), o projétil era apagado ao invés de somente não prosseguir o caminho, diferente do jogador. Além disso, utiliza-se a chamada de sistema sleep (32) para poder controlar a velocidade do mesmo. Nesse processo, devido a posição em que essa linha de código se encontrava alguns erros aconteceram, pois o imediato do branch estava tendo seu limite de 12 bits ultrapassado, sendo necessário, como sugerido pelo professor, o uso de uma instrução jal auxiliar pois seu imediato pode contar com 20 bits.

O ataque seguinte a ser desenvolvido foi o melee, o processo para saber a direção do ataque dos projéteis foi reaproveitado, porém somente se compara 16x16 pixels naquela direção, também se checa a parede para saber se é possível fazer o ataque nos próximos 16x16 pixels.

A última opção de ataque não está presente no Gauntlet original, foi inserida como uma opção extra pela equipe. A mecânica presente nela é de um ataque em área, logo aproveita-se as comparações do melee, porém as checagens são feitas em um retângulo de 2 quadrados de 16x16 por 3 quadrados 16x16, houve uma tentativa de fazer essas checagens em um loop e somente definir a direção com a mesma checagem utilizada no melee e no projétil, mas isso apresentou erros, então as checagens foram feitas diretamente na linha de código.

Além disso, a equipe adicionou uma mecânica de cooldown onde inicialmente assume valor igual a zero e a partir do momento em que o player aperta a tecla “espaço”, assume o valor do respectivo cooldown de ataque do personagem que a partir daí decrementa junto com o contador de tempo.

Inimigos:

Com os ataques prontos, o próximo tópico são os inimigos, suas movimentações e ataques. A equipe desenvolveu três inimigos que, assim como as opções de personagens para o player, fizeram parte do semestre de ISC. O primeiro tipo de inimigo é a “moto”, que para seu movimento a equipe reaproveitou os conceitos usados para a movimentação do personagem, porém para definir o caminho implementaram uma checagem e um espaço na memória para definir a direção (1 – cima, 2 – baixo, 3 – esquerda, 4 – direita). A checagem consiste em ver se é possível seguir na direção atual, ou seja, se o próximo espaço de 16x16 é uma parede, porta fechada ou objeto de colisão ou não. Caso não, se altera a direção indo do 1 para o 4, do 4 para o 2, do 2 para o 3 e do 3 para o 1 ou caso seja o player ele segue na mesma posição causando dano ao player e seguindo seu caminho normalmente.

O próximo tipo de inimigo desenvolvido é o “projetor”. Antes de fazer a movimentação o “projetor” checa se logo a frente contém o player ou alguma parede, se tiver um player ele permanece ali causando danos. Caso seja uma parede, ele avança para a dinâmica de movimentação e caso não seja uma parede nem o jogador, ele lança seu projétil que segue a dinâmica do projétil do player, após isso ele segue para a dinâmica de movimento. Para o movimento do “projetor” a equipe reaproveitou novamente o código de movimento, porém para definir a direção que o inimigo deve seguir utiliza-se um espaço na memória (1 – esquerda, 2 – direita) que só se altera se a checagem da dinâmica de movimento apresentar que é impossível seguir naquela direção.

O último tipo de inimigo é o capacitor que assim como o dano em área sua dinâmica não aparece no Gauntlet original. Sua movimentação consiste em guardar o endereço atual da posição do jogador e de 1 em 1 segundo ir para ela, seguindo o jogador que, caso esteja lá no momento que o capacitor se teletransporta, recebe dano.

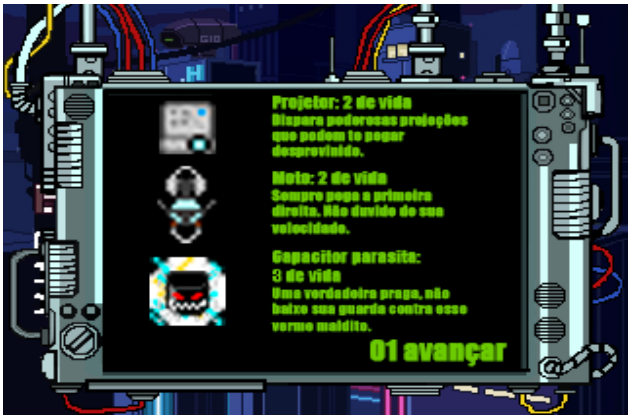

Figura 4. Tela de instruções sobre os tipos de inimigos.

Vida e Score:

A próxima mecânica a ser desenvolvida é a da vida e a da pontuação (score) do player. Como uma das mecânicas do jogo é o fato da vida ser atrelada ao tempo, a equipe assumiu a abordagem de fazer um contador de tempo em que a cada segundo se decrementa um da vida do personagem. Outras ocasiões em que a vida do personagem se altera é o dano recebido dos inimigos, nesse caso se decrementa 10 da vida, além disso, uma das mecânicas do jogo é ter alguns itens que recuperam a vida do personagem, logo a equipe inseriu a cor desse item na hitbox dinâmica do jogo. Por fim, caso a vida do player zere ou fique abaixo disso lhe é apresentado uma tela de *game over* onde ele tem a opção de retornar ao jogo ou sair. Se ele quiser retornar ao jogo, o programa retorna os valores da memória e dos registradores necessários para seus estados iniciais.

Além da vida, outra dinâmica presente no jogo é o score. Eliminar inimigos e pegar certos itens aumentam o score. Inicialmente, o endereço de memória responsável por guardar o score armazena o valor zero e ao completar as condições passíveis de pontuação, o valor do endereço é incrementado de acordo.

Mudança de fases, mapas e o menu:

Com isso finalizado, os próximos tópicos são os sistemas de mudança de fases e mapas e o menu. Como o jogo tem o objetivo de conseguir a maior quantidade de pontos, a equipe optou por repetir as fases iniciais, logo toda vez que o contador de fases chegava a 3 (0 – mapa 1; 1 – mapa 2; 2 – mapa 3) os mapas recomeçavam. Além disso, para passar de fase é necessário que na checagem da movimentação o próximo ponto seja de cor (0x00000038).

Enquanto para o menu, a equipe decidiu que para a seleção das opções o player deveria inserir algum valor numérico, representado em binário na tela devido a estética escolhida. Para a comparação desses valores foi utilizado o valor em ASCII uma vez que o KDMMIO só recebe valores ASCII. Além disso, o menu apresenta informações sobre os controles. Ao selecionar a opção “jogar” algumas informações sobre o jogo são mostradas na tela, juntamente com a tela de seleção de personagem.

Efeitos sonoros e MIDI:

Para a produção dos efeitos sonoros a equipe, inicialmente, teve que encontrar uma maneira para gerar os arquivos MIDI a serem usados no programa. Para isso encontraram um repositório no GitHub (5) e um vídeo (4) que utilizam do site *Hook Theory*(4) para gerar os arquivos MIDI. Quanto a implementação dos efeitos sonoros, o arquivo MIDI se apresenta no formato de pares (1 – nota, 2 - duração), logo a equipe guarda o primeiro valor em um registrador e o segundo em outro, para o primeiro executa o syscall 31, já

para o segundo aplicaram o uso do syscall de sleep, assim como para a animação do projetil, evitando que as notas tocassem desordenadamente.

Resultados obtidos:

A equipe alcançou o objetivo de atender aos requisitos estabelecidos, fazendo assim uma versão futurista do modo arcade de Gauntlet. Porém algumas coisas presentes no Gauntlet não foram inseridas na versão final do jogo, dentre elas a mensagem que aparece quando o player sofre seu primeiro dano para cada tipo de inimigo, pois essa causava problemas no momento de retornar o mapa para seu estado anterior. Além dessa mecânica, outra que está ausente no jogo é a música contínua, pois sua implementação é desconhecida para os membros da equipe.

Figura 5. Tela do menu do projeto finalizado.

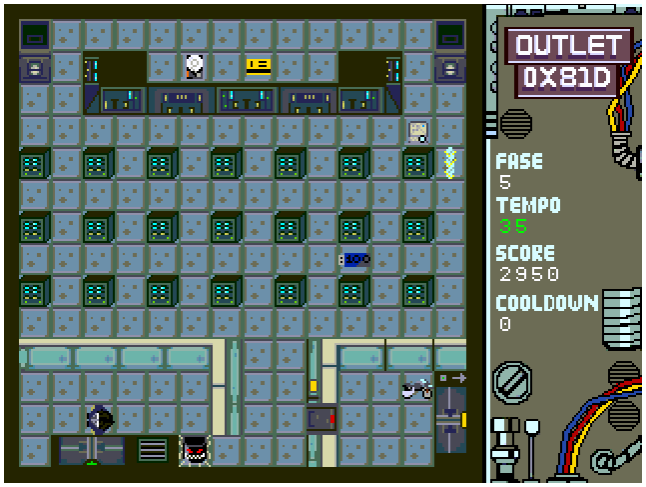

Figura 6. Imagem da jogabilidade do projeto finalizado.

Conclusão:

Por fim conclui-se que apesar da inexperiência e dos problemas enfrentados durante o processo de produção do jogo a equipe obteve êxito e produziu sua versão de Gauntlet no modo arcade usando o RARS.

Além disso, durante o projeto tornou-se evidente as dificuldades e limitações ocasionadas pelo uso de uma linguagem rígida, uma vez que necessita uma grande proficiência na mesma para projetos maiores. Tornou-se evidente também a inabilidade do RARS em executar o jogo sendo necessário que a equipe recorresse ao FPGRARS (6) para os testes das mecânicas do jogo.

Referências bibliográficas:

(1) SANDERSON, PETE; VOLLMAR, KENNETH. RARS: RISC-V assembler and runtime simulator. Versão 1.6 Custom 1. 2023.
(2) PATURI, DAVI. RISC-V RARS - Renderização dinâmica no Bitmap Display - YouTube, 12 maio 2021.
(3) MANUEL, VICTOR; PEREIRA, NATHÁLIA. https://github.com/tilnoene/celeste-assembly, 3 novembro 2021.
(4) LISBOA, VICTOR. Como importar músicas do Hooktheory para tocar no RARS - YouTube, 4 maio 2021.
(5) PATURI, DAVI. https://gist.github.com/davipatury/cc32ee5b5929cb6d1b682d21cd89ae83, 19 outubro 2020.
(6) RIETHER, LEO. Release v1.13.1 · LeoRiether/FPGRARS · GitHub, 18 setembro 2021.