

Castlevania

Gustavo Pierre Starling

Felipe Dantas Borges

Regina Emy Da Nóbrega Kamada

Universidade de Brasília, Departamento de ciência da computação, Brasil



Figure 1: Castlevania, OAC Adventure

ABSTRACT

O projeto constitui uma spin-off do jogo Castlevania, implementada por meio do programa RARS RISC-V Simulator, com o auxílio da ferramenta FPGRARS.

A narrativa é formada por caixas de diálogo e consiste em impedir que o Drácula, o vilão clássico do jogo, seja ressuscitado. Nesse sentido, o personagem principal Ritcher com a ajuda do Lamarzinho, terá que enfrentar diversos inimigos e obstáculos em busca dos 3 cálices de sangue, entidades necessárias para o ritual do renascimento do Drácula.

O jogo utiliza tela não estática, sendo formada principalmente por 8 setores, os quais possuem diversas saídas, sendo possível ir e voltar em cada setor.

Em relação à movimentação do Ritcher (personagem principal) é apresentado 4 teclas (w, s, a, d) associadas aos movimentos verticais e horizontais, respectivamente. Além disso, foi implementado um buffer do teclado, responsável por armazenar e executar os dados enviados pelo input a cada 8 milissegundos. Nesse sentido, faz-se possível movimentar em qualquer direção, incluindo o movimento na diagonal ao pressionar "w" e "a" ou "w" e "d" simultaneamente. Desse modo, o buffer do teclado, atrelado com o momento aplicado (no caso, o módulo da força resultante com a direção) foi crucial para o funcionamento.

No jogo, observa-se uma barra vermelha que constitui a vida do personagem, e números ao lado, o qual corresponde à sua quantidade de mana atual (entidade utilizada ao conjurar determinada habilidade ou arma). Faz-se possível aumentar ambos (quantidade de vida e mana atual) ao coletar corações que podem ser encontrados com uma chance de 50 por cento ao derrotar monstros.

O Richter possui um chicote (arma corpo-a-corpo de alcance médio), além de conseguir lançar shurikens (arma de longo alcance)

com custo de mana. Ademais, faz-se possível utilizar o poder de "Flash", também com um custo de mana, que corresponde à um teletransporte de alcance médio.

Os inimigos possuem sua individualidade, assim como pontos fortes e fracos. Nesse contexto, o âmbito dos inimigos condicionais é formado por fantasmas, zumbis e cavaleiros. Já os inimigos incondicionais são formados por esqueletos, slimes e espinhos. Além disso, no final do jogo é apresentado o boss final que possui mecânicas complexas e únicas.

Apresenta-se diversas interações com o mapa, como os 3 cálices de sangue, corações e até mesmo diálogo com o Lamarzinho (Personagem visível e interativo).

Palavra-chave: RISC-V, RARS, desenvolvimento de jogos, Castlevania

1 INTRODUÇÃO

De acordo com REFERENCES [4], "Castlevania: Symphony of the Night é um jogo de ação-adventura 2D desenvolvido e distribuído pela Konami em 1997. Ele é o 13º título da série Castlevania, sendo o primeiro a ser lançado para o console PlayStation e a sequência cronológica de Castlevania: Rondo of Blood."

Nesse viés, tendo em vista que RISC-V é uma arquitetura de conjunto de instruções de padrão aberto (ISA) baseada em princípios estabelecidos de um computador de conjunto de instruções reduzido (RISC). Nesse contexto, por meio do RARS (simulador do RISC-V), fez-se possível implementar em assembly (conhecido como linguagem de montagem que corresponde à uma notação legível por humanos para o código de máquina) uma spin-off do jogo Castlevania Symphony of the Night com um enredo no universo do Castlevania, uma jogabilidade única, personagens novos e personagens clássicos da franquia.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Apriori, foi implementado a entrada do teclado por polling, ou seja, é testado periodicamente se o dispositivo está pronto para realizar a transferência de dados. No entanto, tal técnica exige um tempo considerável do processador, porém não ocorreu perda de qualidade no programa.

Nesse contexto, de acordo com o conceito de buffer, REFERENCES [1] (região de memória física, utilizada para armazenar temporariamente dados, enquanto eles estão sendo movidos de um lugar para o outro), foi realizado um buffer para o teclado. Dessa forma, foi possível executar movimentos diagonais ao apertar as teclas de movimento vertical e horizontal simultaneamente.

Ademais, com o conhecimento adquirido sobre pilha, foi implementada uma pilha que armazena a próxima ação dos inimigos e um auxiliar que controla sua movimentação. Desse modo, foi possível a realização de múltiplos inimigos na tela.

Em relação aos trabalhos relacionados foi utilizado o procedimento de printar uma figura de um arquivo .bin na tela e de executar uma música do REFERENCES [2].

Além disso, foi utilizado o código para capturar e disponibilizar as notas da música do castlevania no REFERENCES [3]

3 METODOLOGIA

A paralaxe consiste em um aparente deslocamento de um objeto observado, que é causado por uma mudança no posicionamento do observador. Nesse contexto, foi implementado um paralaxe, representado de rosa na Figura 2 por meio do deslocamento de um outra imagem com velocidade diferente do deslocamento da tela em comparação à movimentação do personagem.

Com o intuito de se implementar a colisão, foi realizado um segundo mapa do jogo (Figura 3), implementado a partir do mapa original, possuindo o mesmo tamanho, porém cada pixel de colisão foi pintado de preto e cada pixel de livre movimentação foi pintado de branco. Além disso, as entradas/saídas de um setor para o outro foi pintada cada de uma cor. Assim, ao transformar em .bin teremos uma matriz de colisão.

A colisão é realizada pela verificação da matriz de colisão. Assim, caso há colisão, o personagem será empurrado até no sentido contrário.

Já no âmbito de hitbox com os inimigos, quando o personagem atacar, ocorrerá uma verificação pela pilha para ver se a área do ataque do personagem está na área da hitbox dos inimigos, logo o inimigo sofrerá dano até que a condição seja falsa, ou seja, o inimigo sofrerá dano baseado no período que a hitbox do ataque do personagem esteja na área de hitbox do inimigo. Desse modo, ocorrerá uma verificação de vida do inimigo, caso sua vida seja igual ou menor que 0, o inimigo irá explodir como mostrado na figura 5, porém caso contrário, o inimigo será empurrado e apenas sofrerá dano como mostrado na figura 6.

A animação do personagem foi realizada pela mudança de sprites em cada situação (como por exemplo, ao se movimentar para uma determinada direção e sentido) com uma duração específica, a fim de deixar a movimentação fluída.

4 ANALISE DE RESULTADOS

Em relação à tela, obteve-se um empecilho em relação ao tamanho dos setores, pois observa-se que alguns setores possuem largura menor de 320 e altura menor que 240, sendo assim, aparecia um pedaço de outros setores na tela. Contudo, foi facilmente corrigido a partir da construção de uma borda preta para deixar todos os setores com no mínimo 320x240 (Figura 7), assim como nas suas hitbox (Figura 8).

Além disso, também foi implementado um setor com 4 entradas/saídas, o qual funciona como esperado, mostrado na figura 9 e 10.

Percebe-se a fluidez da animação do personagem, em decorrência do uso de todos os sprites disponíveis no REFERENCES [5].

Em relação aos inimigos, observa-se 3 inimigos incondicionais, no caso, a slime (Figura 11) que é imóvel e causa dano ao contato, o esqueleto (Figura 12) que desloca uma distância pré programada, somente na horizontal e após finalizar o movimento, volta à posição inicial, constituindo uma ideia de patrulhamento, e o espinho (Figura 13) que é imortal, imóvel e causa dano ao contato.

Além disso, também observa-se 3 inimigos condicionais, no caso, o zumbi (Figura 14) que surge quando o personagem está próximo dele, perseguindo-o, o fantasma(Figura 15), o qual atravessa parede e persegue o personagem, independente da distância, desde de que esteja no mesmo setor, e o cavaleiro(Figura 16) que consegue bater no chão, lançando onda de energia sobre o chão na direção do personagem ou andar e bater corpo-a-corpo dependendo da distância entre ele e o personagem.

Ademais, verifica-se que em certos pontos do mapa, infelizmente a colisão não obteve a perfeição, pois dependendo da velocidade do personagem e da espessura da parede, faz-se possível ultrapassar área de colisão e o jogo fechar de forma inesperada.

5 CONCLUSÕES E TRABALHOS FUTUROS

Em suma, obteve-se sucesso nas implementações. Nesse sentido, percebe-se a expressiva importância do planejamento, pois faz-se notável a facilidade no início do projeto quando de fato ocorreu um maior nível de organização e planejamento, comparado ao final do semestre, o qual ocorreu um maior nível de desordem devido ao prazo do projeto.

Além disso, com o intuito de melhorar o projeto, tem-se como planejado aperfeiçoar a colisão, assim evitando que o aplicativo feche de forma inesperada.

REFERENCES

- [1] Conceito de buffer. [https://pt.wikipedia.org/wiki/Buffer_\(ci%C3%Aancia_da_computa%C3%A7%C3%A3o\)](https://pt.wikipedia.org/wiki/Buffer_(ci%C3%Aancia_da_computa%C3%A7%C3%A3o)), 2022.
- [2] Github do projeto oac 2021. <https://github.com/Acucar-tempo-e-tudo-que-ha-de-bom/ProjetoOAC-2021>, 2022.
- [3] Github sobre midi. <https://gist.github.com/davipatury/cc32ee5b5929cb6d1b682d21cd89> 2022.
- [4] Historia do castlevania. https://pt.wikipedia.org/wiki/Castlevania:_Symphony_of_Heaven, 2022.
- [5] Sprites do castlevania. <https://www.sprites-resource.com/playstation/cvsotn/>, 2022.

6 IMAGENS



Figure 2: Setor 1



Figure 3: Hitbox do setor 1



Figure 6: Inimigo sofrendo dano

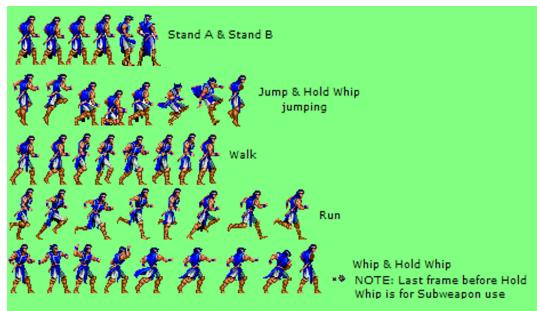


Figure 4: Sprite do Richter



Figure 7: Setores com borda preta



Figure 5: Inimigo morrendo

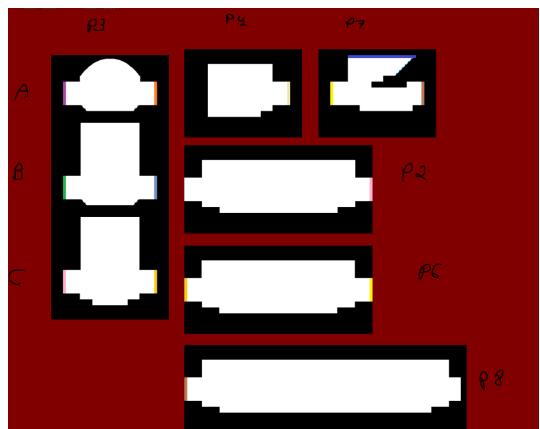


Figure 8: Hitbox dos setores com borda preta



Figure 9: Setor com 4 entradas/saídas

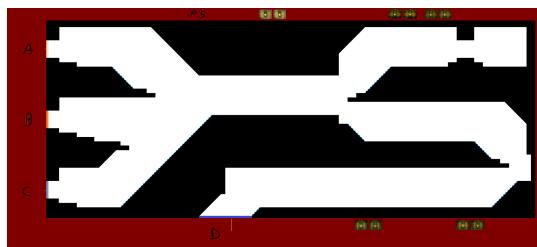


Figure 10: Hitbox do setor com 4 entradas/saídas

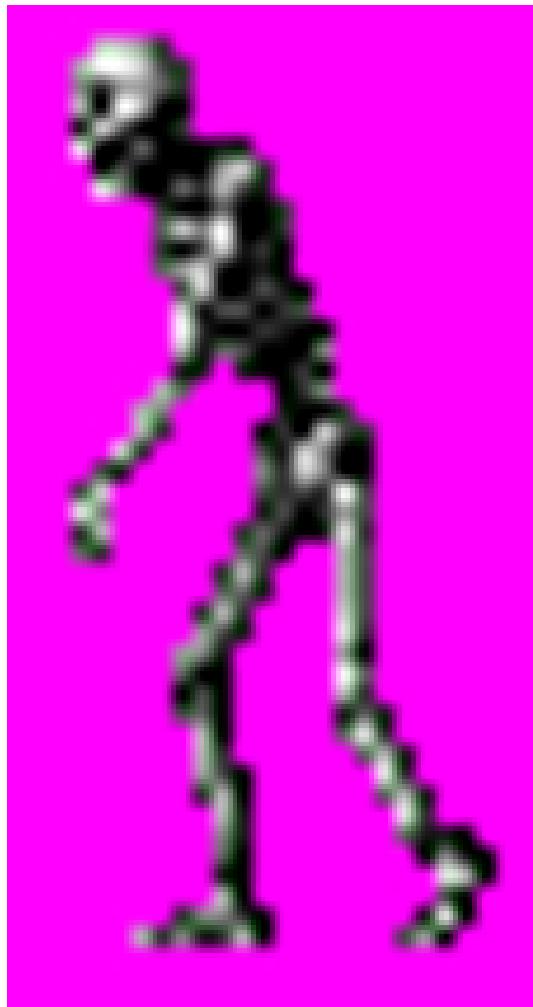


Figure 12: Esqueleto



Figure 11: Slime



Figure 13: Espinho



Figure 14: Espinho



Figure 16: Espinho



Figure 15: Espinho