Relatório Técnico: Angel vs Sins

Desenvolvedores: Alan, João Faé e Ronaldo

Linguagem: Python

1. Descrição do Jogo e Regras

O jogo "Angel vs Sins" é um shooter visto de cima. A premissa é controlar um anjo que desceu ao inferno para combater os Sete Pecados Capitais. O visual é em pixel art com efeitos de partículas e iluminação dinâmica simples.

Regras e Mecânicas:

- **Objetivo:** Sobreviver às hordas de inimigos para acumular pontos. Ao atingir uma pontuação específica (500 pontos), o chefe final (Lúcifer) é invocado. O jogo termina ao derrotar o chefe (Vitória) ou se os pontos de vida do jogador chegarem a zero (Derrota).
- **Controles:**

  WASD: Movimentação.

  Mouse: Mira e Tiro (Botão Esquerdo).

  1, 2, 3: Troca de armas (Pistola, Metralhadora, Shotgun).

  R: Recarregar.

  ESC: Pausa/Menu.

- **Inimigos:** Existem 7 tipos de inimigos comuns, cada um representando um pecado com comportamento único (ex: a *Preguiça* é lenta mas fica furiosa se atacada; a *Gula* cospe outros inimigos).
- **Progresso:** O jogo conta com um sistema de pontuação local, onde o jogador insere seu nome ao final da partida.

2. Estrutura e Diagrama de Classes

O projeto foi estruturado utilizando Orientação a Objetos para facilitar a manutenção e a escalabilidade. As classes se relacionam da seguinte forma:

- **Game (Main):** É a classe "mãe". Ela gerencia o loop principal, inicializa o Pygame, controla os estados do jogo (Menu, Playing, Victory, Leaderboard) e instancia o mapa.
- **Sprite (Classe Base do Pygame):** Quase tudo no jogo herda dessa classe nativa.

    - **Player:** Herda de Sprite. Gerencia input, armas, munição e vida.
    - **Enemy (Classe Pai):** Controla a IA básica de perseguição e colisão.
        - **Boss (Herda de Enemy):** Uma versão especializada com muito mais vida, ataques especiais (tiro em círculo, dash) e máquina de estados para fases de combate.
    - **Projectile:** Classe genérica para balas (tanto do player quanto dos inimigos).
    - **Tile/Lava:** Blocos estáticos que compõem o cenário (paredes e chão perigoso).
- **Sistemas Auxiliares:**
    - **MapGenerator:** Classe responsável pela geração procedural do mapa usando "Random Walk" e autômatos celulares para a lava.
    - **Database:** Classe isolada para gerenciar a conexão SQLite.
    - **Menu/HUD:** Classes responsáveis apenas pela interface visual (UI).

3. Banco de Dados e Estrutura do Ranking

Para o sistema de liderança (Leaderboard), optamos por usar o **SQLite**. A escolha se deu por ser um banco de dados leve, que não exige instalação de servidor externo (o arquivo fica na pasta do jogo) e já vem nativo no Python.

Estrutura da Tabela: Criamos uma tabela única chamada scores com a seguinte estrutura:

- `id`: Chave primária (autoincremento).
- `name`: Texto (String) para armazenar o nome do jogador (limitado a 10 chars no input).
- `score`: Inteiro (Int) para a pontuação final.

Lógica de Funcionamento: Sempre que o jogador termina a partida (seja morrendo ou vencendo), o jogo abre um input de texto. Ao confirmar, o script database.py executa um INSERT. Para exibir o ranking, faço uma query SELECT ordenando pelo score de forma decrescente (DESC) e limitando aos top 10 resultados.

4. Dificuldades Encontradas e Soluções Adotadas

Durante o desenvolvimento, enfrentamos alguns desafios técnicos interessantes:

1. **Controle de Volume em Tempo Real:**
    - *Problema:* Inicialmente, ao mudar o volume no menu, o som dos tiros não mudava. Isso acontecia porque a variável de configuração era importada de forma estática (from settings import *).
    - *Solução:* Mudamos a importação para o módulo inteiro (import settings) e passei a acessar settings.SFX_VOLUME. Assim, qualquer alteração no menu reflete instantaneamente no jogo sem precisar reiniciar.
2. **Gerenciamento de Erros (Logs):**
    - *Problema:* Era difícil debugar erros que aconteciam com usuários testando (ex: arquivo de áudio faltando) sem ter o console aberto.

    - Solução: Implementamos o módulo logging do Python. Agora, qualquer Exception crítica ou falha de carregamento de assets é gravada num arquivo error.log com data e hora, facilitando muito a manutenção.
3. **Lógica do Chefe (Lúcifer):**
    - *Problema:* Fazer o chefe não ser apenas um "inimigo grande", mas ter comportamento inteligente.
    - *Solução:* Criamos uma máquina de estados simples dentro da classe Boss. Ele alterna entre perseguir, atirar em leque, invocar minions e dar um dash furioso dependendo de timers e da quantidade de vida (fase 1 e fase 2).