💻 Documento de Design Técnico (TDD) – Lucy & Nero

1. Visão Geral e Motor

- **Motor:** Godot Engine 4.5 (ou superior), utilizando o renderizador Forward+ para aproveitar a iluminação dinâmica e os efeitos visuais.
- **Linguagem:** GDScript para a maior parte da lógica de jogo, devido à sua integração com o motor e prototipagem rápida. C# pode ser considerado para sistemas de alta performance se necessário, mas não é o plano inicial.
- **Versionamento:** Git, com um repositório central (ex: GitHub, GitLab). O arquivo `.gitignore` já está configurado para ignorar a pasta `.godot` .

2. Arquitetura Principal

- **Estrutura de Cenas:** O jogo será modular. `Player.tscn` (contendo Lucy e instanciando Nero), `Enemy.tscn` , e `Level.tscn` serão as cenas base.
- **Singleton (Autoload) - Event Bus:** Um script global `EventBus.gd` será usado para comunicação desacoplada entre partes distantes do jogo (ex: um inimigo morre e emite um sinal `enemy_died` , que a UI escuta para atualizar um contador). Isso evita referências diretas e complexas na árvore de cenas.
- **Recursos Customizados:** A maior parte dos dados do jogo (armas, runas, habilidades, status de inimigos) será definida através de Recursos (`.tres` arquivos). Isso permite que designers de jogo ajustem o balanceamento sem tocar no código.
    - `WeaponData.tres`
    - `RuneData.tres`
    - `NeroAbility.tres`

3. Sistemas de Gameplay

Controle do Jogador (Lucy)

- **Nó Principal:** `CharacterBody3D` para gerenciar movimento e colisões.
- **Input:** O `Input Map` do Godot será usado para mapear ações (Ataque, Esquiva, Habilidade, etc.), permitindo fácil remapeamento de teclas/controles.
- **Máquina de Estados (State Machine):** Uma FSM simples irá gerenciar o estado de Lucy (`Idle` , `Attacking` , `Dodging` , `UsingAbility`) para evitar lógicas conflitantes.

Inteligência Artificial (Nero)

- **Nó Principal:** `CharacterBody3D` .
- **Máquina de Estados (FSM):** Uma FSM mais complexa gerenciará o comportamento de Nero. Os estados serão baseados nos comandos do jogador e no contexto do combate:
  - `STATE_FOLLOW` : Segue Lucy de perto.
  - `STATE_ATTACK_TARGET` : Foca no alvo designado por Lucy.
  - `STATE_DEFENSIVE` : Permanece perto de Lucy e ataca inimigos que se aproximam.
  - `STATE_FREE_ROAM` : Ataca alvos de oportunidade.
- **Navegação:** `NavigationServer3D` será usado para o pathfinding de Nero nos níveis.

Sistema de Combate

- **Detecção de Acerto:** `Area3D` será usado para hitboxes (quem ataca) e hurtboxes (quem pode ser atingido). Sinais de `body_entered` ou `area_entered` detectarão as colisões.
- **Cálculo de Dano:** Uma função central, possivelmente no `EventBus` ou em outro Singleton, calculará o dano final com base nos status da arma, runas, e resistências do alvo.

4. Mundo e Níveis

- **Construção:** Os níveis serão cenas Godot construídas manualmente para garantir um design de alta qualidade.
- **Otimização:** `Occluder3D` e `VisibilityNotifier` serão usados para fazer o culling de objetos que não estão na visão da câmera, garantindo performance.
- **Elementos Aleatórios:** A aleatoriedade (loot, posição de inimigos) será controlada por um script de nível que usa `seed` para garantir que as "runs" possam ser replicadas se necessário.

5. Interface (UI)

- **Estrutura:** As cenas de UI (`.tscn`) serão separadas da lógica de jogo. Uma cena principal de HUD será adicionada à tela e atualizará suas informações escutando os sinais do `EventBus` (ex: `player_health_changed`).