**Documentação do Backend do Projeto**

**Nota:** Este documento ainda está em construção e é atualizado conforme a lógica do projeto é implementada.

**1. Conceitos Básicos**

* **Cena Tela:** Uma cena que funciona como uma tela do jogo, representando um estágio da gameplay, como a tela inicial, tela de batalha, tela de exploração, tela de diálogo, etc. O nó principal das cenas de tela deve, preferencialmente, ser um nó sem script.
* **Cena Entidade:** Um conjunto de nós que se organizam em um microssistema para representar um elemento visual concreto e independente. Não é uma tela de estágio do jogo, mas sim uma entidade que integra uma tela.

**2. Estrutura das Cenas Telas e seus Elementos**

**Cena Tela de Batalha (Battle Scene)**: Tela de batalha, contendo a principal dinâmica do jogo.

* **Interface**: Nó que contempla elementos puramente visuais.
  + **UserUI**: Sprite2D responsável por exibir o elemento visual estático na parte inferior da cena de batalha. Não possui script próprio, pois é sempre a mesma imagem estática.
  + **BackGround**: Sprite2D que representa o cenário. Possui um script para gerenciar o cenário a ser exibido (ver classe *StageManager* - ainda não implementado).
* **BattleManager**: Node2D que gerencia a batalha (ordem de turnos, fases, ciclo de vida da aplicação, etc.). Contém todos os nós que participam da batalha. (Classe *BattleManager* - ainda não implementado).
  + **Enemy**: Node2D representando o inimigo. Possui script do tipo *Enemy* (que herda de *Tamer*). Gerencia o comportamento do Digimon Inimigo.
    - **Digimon (Enemy)**: Node2D que representa a entidade visual do Digimon inimigo.
  + **Player**: Node2D com script do tipo *Player* (também herda de *Tamer*). Gerencia a interface do usuário e a interação com os botões.
    - **Digimon (Player)**: Node2D idêntico ao Digimon (Enemy). A lógica determina o comportamento com base no *Tamer* (Player ou Enemy).
    - **ButtonPanel**: Node2D que gerencia todos os botões da cena.

**3. Estrutura das Cenas Entidades**

**Digimon**: Node2D com script da classe *Digimon*.

* **DigimonSprite**: Sprite2D responsável pela textura do Digimon.
* **Animator**: AnimationPlayer para gerenciar as animações do Digimon.
* **SkillSpawner**: Node2D responsável por mostrar efeitos visuais das habilidades usadas pelo Digimon, posicionado em (0, 0).
  + **SkillSprite**: Sprite2D que recebe a textura da habilidade.
  + **Animator**: AnimationPlayer que executa a animação da habilidade.

**4. Modelagem de Dados: Classes, Funções e Atributos**

**Classes Globais e de Armazenamento de Dados**

* **DigimonDB**: Armazena todos os recursos que representam os Digimons.
  + **Atributos**:
    - digimons (dicionário): Relação chave-valor com o ID do Digimon e seu endereço dentro do projeto.
  + **Funções**:
    - getDigimonData(index: int): Retorna um valor do tipo *DigimonData* baseado no ID do Digimon.
* **EnumDB**: Centraliza todos os enums da aplicação. Todos os atributos são enums.
  + **Atributos**:
    - Type, Tier, Element, DamageType
* **Utility**: Classe global com funções variadas.
  + **Funções**:
    - random(minn: int, maxn: int): Retorna um valor aleatório entre dois valores.
    - chance(nchance: float): Retorna um booleano baseado em uma chance.
    - cap(value: float): Retorna um valor flutuante com duas casas decimais.
    - damageAbilityBuilder(damageSkill: DamageSkill): Cria um *DamageData* a partir de uma *DamageSkill*.

**Classes Entidade**

* **Digimon**: Representa os Digimons, principal elemento do jogo.
  + **Atributos**: digimonID, digimonName, digimonIcon, digimonDescription, element, digimonTier, digimonType, atributos base e bônus (baseSTR, bonusSTR...), atributos dinâmicos (totalDamage, criticalChance...), entre outros.
  + **Funções**:
    - setBehave(): Determina a posição e manipula o flip da textura do Digimon.
    - setStats(stats: DigimonData): Atualiza os status do Digimon com base em um *DigimonData*.
    - getAttribute(att: String): Retorna a soma de um atributo base e seu bônus.
    - Outras funções para cálculos de dano, chance crítica, precisão e processamento de habilidades.
* **Tamer**: Classe que gerencia o Digimon.
  + **Funções**:
    - summonDigimon(index: int): Invoca um Digimon com base em seu ID.
* **Player**: Herda *Tamer*. Gerencia o painel de botões e outras interações do usuário.
  + **Funções**:
    - \_ready(): Invoca um Digimon aleatório e atualiza o painel de botões.
* **Enemy**: Herda *Tamer*. No futuro, será programada para simular uma inteligência artificial.

**Classes para Construtores de Interface**

* **ButtonPanel**: Gerencia os botões em tela e suas interações.
  + **Funções**:
    - setButtons(): Associa cada botão à habilidade do Digimon.
    - updateButtons(): Atualiza os botões com base no cooldown das habilidades.
* **SkillSpawner**: Cria efeitos visuais das habilidades.
  + **Funções**:
    - \_ready(): Flipa a sprite com base no *Tamer*.
    - spawnSkill(skill: Skill): Executa a animação da habilidade.

**Classes Abstratas para Recursos**

* **DigimonData**: Armazena dados estáticos sobre os Digimons.
* **DamageData**: Recurso dinâmico para processar dano em batalha.
* **DamageSkillData**: Armazena habilidades de dano no banco de dados.

**Classes de Habilidades**

* **Skill**: Classe básica de habilidades.
  + **Funções**:
    - learn(), unlearn(), applyCooldown(), countCooldown()
* **DamageSkill**: Herda *Skill*. Representa habilidades de dano com efeitos secundários.
  + **Funções**:
    - setStats(), setValues(), applyStatus(), skillSingularity()