Documentação Final do Projeto: "X-Men - Academia de Mutantes"

1. Visão Geral do Projeto

Objetivo

O objetivo do projeto "X-Men - Academia de Mutantes" é criar um sistema de gerenciamento de mutantes dentro de uma academia fictícia, onde eles podem ser treinados, enviados em missões e batalhas. O sistema utilizará **Programação Orientada a Objetos (POO)** com **herança**, **polimorfismo**, **encapsulamento** e **abstração**, além de persistência de dados em arquivos **JSON**. O projeto visa simular um ambiente dinâmico de combate, evolução e estratégias de mutantes como Wolverine, Jean Grey, Magneto, entre outros.

Tecnologias Utilizadas

- **Python 3.x**: Linguagem de programação.
- **JSON**: Para persistência dos dados (salvamento e carregamento de mutantes e suas interações).
- **POO**: Paradigma de programação com classes e objetos, aplicando conceitos como herança, polimorfismo e encapsulamento.

---

2. Funcionalidades do Sistema

O sistema possui as seguintes funcionalidades principais:

Cadastro de Mutantes

- O usuário pode cadastrar mutantes, atribuindo a eles **nome**, **poder** principal e **habilidades especiais**.
- O sistema armazena informações como **vida inicial**, **poderes** e **habilidades**.

Treinamento dos Mutantes

- O sistema permite que os mutantes passem por sessões de **treinamento**, que aumentam seus **atributos** de poder ou resistência.

- Cada tipo de mutante tem suas habilidades exclusivas que influenciam no tipo de treinamento.

Batalhas e Missões

- O usuário pode enviar os mutantes para **batalhas** contra inimigos ou entre si.

- Cada batalha causa **dano** à vida dos mutantes, e a batalha é decidida com base nos **poderes** e **habilidades** de cada mutante.

Persistência de Dados

- Todos os dados dos mutantes, incluindo progresso em missões, treinamento e batalhas, são salvos em um arquivo **JSON**. Isso garante que o progresso seja mantido entre as execuções do programa.

Menu Interativo

- O menu do sistema permite ao usuário interagir com as opções de cadastro, batalha, treinamento, visualização de estatísticas e carregamento/salvamento de dados.

---

3. Estrutura do Sistema

O sistema é estruturado utilizando **classes**, onde cada classe tem um papel específico dentro da lógica do sistema. A seguir, a **estrutura de classes**:

Classes do Sistema

1. **Classe `Mutante` (Classe Base)**
    - Representa um mutante genérico.
    - Atributos:
        - `nome`: Nome do mutante.
        - `vida`: Vida inicial do mutante.

        - `poder`: Poder principal do mutante.
        - `habilidades`: Lista de habilidades especiais.
    - Métodos:
        - `usar_poder()`: Método polimórfico que será sobrescrito nas subclasses para definir o efeito específico de cada poder.

2. **Subclasses de `Mutante`**
    - **`MutanteFisico`**: Representa mutantes com poderes físicos, como Wolverine.
        - Exemplo de poder: Garras, Força Aumentada.
    - **`MutantePsiquico`**: Representa mutantes com poderes psíquicos, como Jean Grey.
        - Exemplo de poder: Telecinese, Leitura de Mentes.

3. **Classe `Missao`**
    - Representa uma missão em que o mutante pode ser enviado.
    - Atributos:
        - `descricao`: Descrição da missão ou inimigo enfrentado.
        - `dificuldade`: Nível de dificuldade da missão.
        - `premios`: Premiação em XP após a missão.
    - Métodos:
        - `iniciar_missao()`: Inicia a missão, com cálculo de recompensa e consequências para os mutantes.

4. **Classe `Batalha`**
    - Representa uma batalha entre mutantes ou contra vilões.
    - Atributos:

        - `mutante1`: O primeiro mutante na batalha.
        - `mutante2`: O segundo mutante ou vilão.
    - Métodos:
        - `iniciar_batalha()`: Controla a batalha entre dois mutantes, aplicando o cálculo de dano e vitória.

5. **Classe `ArquivoJSON`**
    - Responsável pela leitura e escrita de dados dos mutantes em **JSON**.
    - Métodos:
        - `salvar_dados()`: Salva as informações dos mutantes no arquivo JSON.
        - `carregar_dados()`: Carrega os dados salvos para o sistema.

---

4. Fluxo de Execução

O programa começa com o **menu principal**, onde o usuário pode selecionar as ações desejadas. O fluxo de execução do sistema segue as etapas abaixo:

1. **Menu Principal**:
    - **Cadastrar Mutante**: Permite ao usuário adicionar um novo mutante, fornecendo seu nome, poder e habilidades.
    - **Treinar Mutante**: O usuário pode treinar um mutante, aumentando suas habilidades.
    - **Enviar para Missão**: O usuário envia um mutante para uma missão, onde ele poderá ganhar XP e melhorar.
    - **Iniciar Batalha**: O usuário desafia outro mutante ou vilão para uma batalha.
    - **Visualizar Estatísticas**: O sistema exibe as informações sobre o progresso dos mutantes cadastrados.
    - **Salvar e Sair**: Salva o estado atual dos mutantes e encerra o programa.

2. **Batalhas e Dano**:
    - **A cada batalha**, os mutantes podem perder **vida** com base no **dano causado** pelos ataques.
    - O dano é calculado de acordo com o tipo de poder de cada mutante (físico ou psíquico) e a **resistência** do oponente.
    - Durante a batalha, a vida do mutante será reduzida, e o mutante poderá usar **habilidades especiais** para se defender ou causar mais dano.
    - **Exemplo de cálculo de dano**:
        - **Mutante A (Wolverine)**: Vida 100, Poder 20.
        - **Mutante B (Magneto)**: Vida 120, Poder 25.
        - Wolverine ataca e causa 20 de dano a Magneto (Magneto fica com 100 de vida).
        - Magneto ataca e causa 10 de dano a Wolverine (Wolverine fica com 90 de vida).
        - A batalha continua até que um dos dois perca toda a vida.
3. **Após a Batalha**:
    - O sistema verifica se o mutante venceu ou perdeu.
    - **Experiência** (XP) é atribuída ao vencedor, e o mutante perde **vida** caso tenha sofrido dano.
    - Caso o mutante vença a batalha, ele pode ganhar **habilidades especiais** ou **itens de cura**.

---

5. Persistência de Dados (JSON)

A persistência de dados é fundamental para garantir que o progresso dos mutantes seja mantido. O programa salvará os dados em arquivos **JSON** para que as informações possam ser recuperadas entre as execuções.

Exemplo de Estrutura de Arquivo JSON:

```
{
  "mutantes": [
    {
      "nome": "Wolverine",
      "vida": 100,
      "poder": "Garras",
      "habilidades": ["Cura Rápida", "Força Aumentada"],
      "missões_completadas": 3
    },
    {
      "nome": "Jean Grey",
      "vida": 80,
      "poder": "Telecinese",
      "habilidades": ["Leitura de Mentes", "Voo"],
      "missões_completadas": 5
    }
  ]
}
```

- **Função `carregar_dados()`**: Ao iniciar o programa, os dados dos mutantes são carregados do arquivo JSON.
- **Função `salvar_dados()`**: Após qualquer alteração no status dos mutantes (batalhas, missões, etc.), o sistema salva automaticamente as informações em JSON.

---

6. Critérios de Avaliação

A avaliação do projeto será feita com base nos seguintes critérios:

1. **Aplicabilidade dos Conceitos de Orientação a Objetos (30%)**:
    - Uso correto de **herança**, **polimorfismo** e **encapsulamento**.
    - Organize corretamente as classes e seus métodos.
2. **Organização e Estrutura do Código (15%)**:

    - Código bem organizado em módulos e classes.
    - Boa separação de responsabilidades entre as classes.
3. **Persistência de Dados (20%)**:
    - Salvar e carregar dados dos mutantes de maneira eficiente e funcional utilizando **JSON**.
4. **Interação e Funcionalidade do Menu (20%)**:
    - O menu e as opções devem ser claras e de fácil navegação.
5. **Criatividade (15%)**:
    - Implementação de recursos criativos, como habilidades únicas para mutantes, missões desafiadoras e batalhas com resultados variáveis.

---

7. Conclusão

O projeto "X-Men - Academia de Mutantes" oferece uma aplicação prática dos conceitos de **orientação a objetos**, proporcionando uma experiência divertida e dinâmica com batalhas e missões. Ele utiliza **herança**, **polimorfismo** e **persistência de dados**, atendendo aos requisitos do professor e oferecendo um aprendizado profundo sobre programação orientada a objetos.