TG1: Sistema de RPG

Programação Orientada à Dados

Prof. Me. Otávio Parraga

Objetivo:

Durante a disciplina discutimos sobre a importância de abstrair o mundo real em objetos, e como isso pode facilitar a implementação de sistemas complexos. O objetivo dessa tarefa é aplicar os conceitos discutidos em aula para implementar um sistema de RPG, onde o aluno deve criar uma série de classes que representem os conceitos de um jogo siples de RPG.

Descrição:

O grupo deve implementar um sistema de RPG, onde o jogador pode criar um personagem, escolher uma classe e realizar ações como atacar e defender. O sistema deve ser capaz de calcular o dano causado por um ataque, levando em consideração a defesa do oponente e as habilidades do personagem.

O sistema deve conter um arquivo principal `main.py` que deverá selecionar uma opção de menu e executar a ação correspondente.

- O sistema deve conter um menu com as seguintes opções:
  - Combate entre dois personagens
  - Combate entre múltiplos personagens (*FreeFor All*)

O sistema deve carregar todos os personagens, habilidades e classes de um arquivo de configuração, que deve estar no formato `markdown`. O arquivo de configuração deve conter as informações necessárias para poder instanciar os objetos de personagens. Durante o processo de leitura, deverão ser tratados erros, como por exemplo, quando um personagem carregado do arquivo possui mais habilidades que o permitido pela classe. O sistema deve ser capaz de lidar com esses erros e registrar as informações em um arquivo de log.

Ao final da execução, dois arquivos deverão ter sido gerados, um arquivo de relatório de combates, e um arquivo contendo os erros.

O desenvolvimento será em grupos de até 2 pessoas!

Classes Obrigatórias:

- `Dado`: Classe abstrata que representa um dado de RPG.
  - `lados`: Número de lados do dado.
  - `jogar()`: Método que simula o lançamento do dado e retorna um número aleatório entre 1 e o número de lados do dado.
- `Classe`: Classe abstrata que representa uma classe de personagem. Deve conter os seguintes atributos:
  - `nome`: Nome da classe.
  - `pontos_vida`: quanta vida o personagem deverá ter.
  - `dado_de_ataque`: Dado de ataque da classe, um objeto do tipo `Dado`.
  - `pontos_ataque`: Pontos de ataque da classe.
  - `pontos_defesa`: Pontos de defesa da classe.
  - `limite_habilidades`: Limite de habilidades que o personagem pode ter.
- `Personagem`: Classe que representa um personagem do jogo.
  - `qntd_instancias`: Atributo que representa a quantidade de objetos instanciados.
  - `nome`: Nome do personagem.
  - `classe`: Classe do personagem, um atributo do tipo `Classe`.
  - `inventario`: Inventário do personagem, uma lista de objetos do tipo `Habilidade`. Após usar uma habilidade, ela deve ser removida do inventário.
  - `pontos_vida`: Pontos de vida do personagem, definidos pela classe escolhida.
  - `dado_de_ataque`: Pontos de ataque do personagem, definidos pela classe escolhida.
  - `pontos_ataque`: Pontos de ataque do personagem, definidos pela classe escolhida.
  - `pontos_defesa`: Pontos de defesa do personagem, definidos pela classe escolhida.
  - `atacar(alvo : Personagem)`: Método que simula um ataque do personagem, retornando o dano causado.
    - Ao atacar, o personagem deve, antes de jogar o dado de ataque, verificar se não utilizará uma habilidade.
    - Enquanto houver habilidades no inventário, o personagem deve ter uma chance de 50% de usar uma habilidade.
    - O dano padrão de qualquer personagem é realizado com o dado de ataque da classe.
  - `usar_habilidade(alvo : Personagem)`: Método que simula o uso de uma habilidade, retornando o dano causado.
- `Habilidade`: Classe que representa uma habilidade do personagem.
  - `nome`: Nome da habilidade.
  - `descricao`: Descrição da habilidade.
  - `pontos_ataque`: Pontos de ataque da habilidade.
  - `usar()`: Método que simula o uso da habilidade.

- `Arena`: Classe que representa a arena de combate.
  - `personagens`: Lista de personagens que estão na arena.
  - `adicionar_personagem()`: Método que adiciona um personagem à arena.
  - `remover_personagem()`: Método que remove um personagem da arena.
  - `combate()`: Método que simula o combate entre os personagens da arena, retornando o vencedor.
    - As regras do combate serão as seguintes:
      - O combate será realizado em turnos, onde cada personagem pode atacar um oponente aleatório (em combates com dois jogadores, será sempre o mesmo).
      - O atacante rodará um D20 (um dado de 20 lados) e somará o resultado ao seu ataque.
      - Se o valor final de ataque for maior que o valor de defesa do oponente, o ataque será bem sucedido.

Subclasses:

- `Guerreiro`: Subclasse de `Classe` que representa um guerreiro.
  - `pontos_vida`: 10 + (`pontos_defesa` * 5)
  - `dado_de_ataque`: D12.
  - `pontos_ataque`: 6
  - `pontos_defesa`: 8
  - `limite_habilidades`: 2
- `Mago`: Subclasse de `Classe` que representa um mago.
  - `pontos_vida`: 8 + (`pontos_defesa` * 2)
  - `dado_de_ataque`: D6.
  - `pontos_ataque`: 10
  - `pontos_defesa`: 3
  - `limite_habilidades`: 5
- `Ladino`: Subclasse de `Classe` que representa um ladino.
  - `pontos_vida`: 6 + (`pontos_defesa` * 3)
  - `dado_de_ataque`: D8.
  - `pontos_ataque`: 8
  - `pontos_defesa`: 5
  - `limite_habilidades`: 3
- `BolaDeFogo`: Subclasse de `Habilidade` que representa uma bola de fogo.
  - `descricao`: "Uma bola de fogo que causa dano em área."
  - `usar()`: Método que simula o uso da habilidade, causando 10 dano.

- `Cura`: Subclasse de `Habilidade` que representa uma cura.
  - `descricao`: "Uma cura que recupera 10 pontos de vida."
  - `usar()`: Método que simula o uso da habilidade, recuperando 10 pontos de vida.
- `Tiro de Arco`: Subclasse de `Habilidade` que representa um tiro de arco.
  - `descricao`: "Um tiro de arco que causa dano em área."
  - `usar()`: Método que simula o uso da habilidade, causando 6 dano.
- Subclasses para cada tipo principal de Dado:
  - `D4`: Subclasse de `Dado` que representa um dado de 4 lados.
  - `D6`: Subclasse de `Dado` que representa um dado de 6 lados.
  - `D8`: Subclasse de `Dado` que representa um dado de 8 lados.
  - `D10`: Subclasse de `Dado` que representa um dado de 10 lados.
  - `D12`: Subclasse de `Dado` que representa um dado de 12 lados.
  - `D20`: Subclasse de `Dado` que representa um dado de 20 lados.

Restrições:

- Toda classe deve implementar métodos `__str__` e `__repr__` para facilitar a visualização dos objetos.
- Implemente um método `__eq__` para comparar dois personagens, considerando o nome e a classe.
- Implemente todos os métodos necessários para garantir que a classe `Dado` possa ser comparada com outros dados (`D20` > `D12`, `D4` <= `D10`, ...).
- O sistema deve ser capaz de lidar com erros, como por exemplo, quando um personagem carregado do arquivo possui mais habilidades que o permitido pela classe.
- Os erros devem ser registrados em um arquivo de log, que deve conter uma explicação sobre o erro.
- O sistema deve gerar um relatório de combate, que deve conter o nome dos personagens, o dano causado por cada ataque e o vencedor do combate.
- Todas as classes e métodos devem ser documentados com docstrings.
- Para usar o sistema, você deverá implementá-lo como um pacote. Ou seja, deverá criar um diretório chamado `RPG` que conterá os arquivos de classes. O script `main.py` deverá estar fora desse diretório e chamar as classes do pacote `RPG`.

Ponto por Criatividade:

- O grupo deverá implementar alguma funcionalidade nova no jogo, que não foi discutida em aula. Essa funcionalidade deve ser explicada e demonstrada durante a apresentação.
- Após as apresentações, os colegas irão votar pela funcionalidade mais interessante, e o grupo que receber mais votos ganhará um ponto extra o trabalho (lembrando que o

trabalho totalizará sempre 10 pontos).

Requisitos:

- Os únicos módulos que podem ser utilizados livremente são:
  - `random`
  - `math`
  - `datetime`
  - `sys`
  - `abc`
  - `os`
- Caso outro módulo seja utilizado, o aluno deve justificar o uso do mesmo em um comentário no código.
- O sistema deve ser implementado em Python 3.8 ou superior.
- Todas as configurações de entrada deverão estar contidas em um arquivo de configuração no formato `markdown`.

Critérios de Avaliação:

- Implementação correta das classes, métodos obrigatórios e organização (4 pontos)
- Implementação correta das subclasses (1 pontos)
- Implementação correta do sistema de combate (1 pontos)
- Implementação correta do sistema de log e relatório (1 ponto)
- Inovação e criatividade (1 ponto)
- Apresentação (1 ponto)
- Clareza, documentação e organização do código (1 ponto)

Entrega:

- A entrega deverá ser feita através do GitHub.
- O repositório deverá conter todo o código-fonte, incluindo os arquivos de configuração.
- O repositório deverá conter um arquivo `README.md` com as instruções de execução do sistema e nome e matrícula dos alunos do grupo.
- Apresentação em sala de aula, com duração de 10 minutos detalhando o processo de implementação, as funcionalidades do sistema e a inovação implementada.
- **O Trabalho deverá ser entregue até o dia 19/05/2025.**