UNIVERSIDADE FEDERAL DO CARIRI

DungeonPy

Fique forte, explore masmorras

2026

INTEGRANTES

ALAN MENDES VIEIRA - @alan-mendes-ufca

CICERO JESUS DA SILVA GOMES - @cicero-jesus

LEÔNCIO FERREIRA FLORES NETO - @LeoncioFerreira

PAULO GABRIEL LEITE LANDIM - @LandimPG

SALOMÃO RODRIGUES SILVA - @salomaosilvaa

DungeonPy

Fique forte, explore masmorras

Trabalho apresentado a Jayr e demais colegas do curso de Engenharia de software referente ao segundo projeto da disciplina de POO.

1. DESCRIÇÃO DO DOMÍNIO

Introdução

O projeto consiste no desenvolvimento de um jogo de exploração de masmorras (Dungeon Crawler) utilizando o paradigma de orientação a objetos. O jogador controla um herói que entra em uma masmorra composta por várias salas conectadas sequencialmente. Em cada sala, o herói deve gerenciar recursos limitados para enfrentar inimigos e sobreviver.

O sistema modela:

- **Entidades:** O personagem do jogador (Herói) e os inimigos (Monstros). Ambos compartilham comportamentos vitais, mas diferem em suas especializações. As Entidades podem sofrer mudanças de estado (envenenamento, atordoamento, congelamento) dependendo dos ataques recebidos.
- **Itens:** Equipamentos que definem o poder ofensivo (Armas) ou recursos de sobrevivência (Consumíveis).
- **Combate:** Mecânica tática baseada em turnos, onde a escolha da arma e o aproveitamento das fraquezas elementais determinam a vitória.
- **Salas:** Representam os segmentos da dungeon, atuando não apenas como cenário, mas como um contexto ativo que influencia as regras de combate através de fatores ambientais.

1.1 Entidades

O conceito de “Entidades” refere-se aos seres vivos dispostos ao longo da aventura.

Monstros: Personagens não-jogáveis controlados pelo sistema. Possuem um elemento intrínseco (fraqueza) e, ao serem derrotados, deixam recompensas (loot). O objetivo dos monstros é reduzir a vida do herói a zero.

Herói: O protagonista é controlado pelo jogador. Possui atributos vitais (vida atual, vida máxima, ataque, defesa e velocidade) e gerencia um inventário. O Herói pertence a uma de três classes especializadas:

1. **Arqueiro:** Especialista em combate à distância.
    - *Recurso:* Depende de **Munição** no inventário.
    - *Atributo Exclusivo:* **Pontaria** (aumentam a eficácia do disparo).
    - *Habilidade:* **Mirar** (Sacrifica o turno atual para garantir um acerto no próximo ataque).

2. **Guerreiro:** Especialista em combate físico e sobrevivência.
    - *Recurso:* Utiliza força bruta e equipamentos defensivos.
    - *Atributo Exclusivo:* **Armadura/Escudo** (reduz dano recebido) e **Bloqueio**.
    - *Habilidade:* **Fúria** (Sacrifica a defesa no turno atual para dobrar o dano no próximo ataque).

3. **Mago:** Especialista em dano elemental e controle.
    - *Recurso:* Depende de **Mana**.
    - *Atributo Exclusivo:* **Aprimoramento de Varinha** (potencializa o dano mágico).
    - *Habilidade:* **Magia Suprema** (Gasta grande quantidade de mana e um turno para causar dano massivo).

1.2 Itens

Os itens são fundamentais para a mecânica de jogo e dividem-se em duas categorias na hierarquia:

Itens Consumíveis: Equipamentos de uso único (ex: Poções). Ao serem utilizados, aplicam um efeito imediato (como restaurar vida) e são removidos do inventário. O uso consome o turno do jogador.

Armas (Itens Equipáveis): São os itens que definem o potencial ofensivo. Toda arma possui um **Dano Base** e um **Elemento**. Elas se subdividem em estratégias de uso:

- *Arma à Distância (Arcos):* Exigem munição para funcionar.
- *Grimórios (Armas Mágicas):* Exigem Mana para funcionar e permitem ao Mago lançar feitiços de diferentes elementos.
- *Armas Comuns:* Aplicam dano direto baseado nos atributos do portador.

1.3 Combate

O combate opera sob o modelo de **“Custo de Oportunidade”** em turnos rígidos. A cada rodada, o jogador deve decidir entre atacar (arriscando sofrer dano) ou usar um item (perdendo a chance de causar dano).

Mecânica Elemental: O sistema processa validações contextuais baseadas em um ciclo de vantagens estrito:

Veneno > Raio > Gelo > Fogo > Veneno

- **Natureza da Arma:** O elemento do ataque (Ex: Um Grimório de Fogo).

- **Natureza do Monstro:** A fraqueza da entidade (Ex: Um Monstro de Gelo sofre dano crítico de Raio).

Estados (Efeitos de Status): Além do dano direto, o combate aplica condições temporárias baseadas nos elementos:

- *Veneno:* Aplica o estado **Envenenado** (Dano contínuo por turno).
- *Fogo:* Aplica o estado **Queimado** (Reduz o poder de ataque).
- *Gelo:* Aplica o estado **Congelado** (Reduz a velocidade/defesa).
- *Raio:* Aplica o estado **Atordoado** (Impede a ação no turno).

1.4 Progressão

Diferente de RPGs tradicionais baseados em níveis infinitos, a progressão em DungeonPy é baseada na **aquisição de equipamento (Loot)**.

Como os inimigos das salas finais são progressivamente mais fortes (possuem mais vida e defesa), o jogador precisa explorar a masmorra para encontrar:

- **Armas Melhores:** Espadas, Arcos ou Grimórios com maior dano base.

- **Itens de Aprimoramento:** Itens que aumentam permanentemente a Vida Máxima ou a Mana Máxima.

A sensação de evolução vem da capacidade de derrotar monstros que antes eram impossíveis, graças ao uso estratégico de novas armas elementais encontradas.

1.5 Salas

As salas representam a unidade estrutural da masmorra. No domínio do sistema, uma sala não é apenas um local de passagem, mas um **Objeto de Contexto** que agrupa inimigos e tesouros.

Cada sala possui uma **Descrição Narrativa** (para imersão) e um **Tipo de Ambiente** (Elemento). O ambiente influencia diretamente o combate através da "Sinergia Ambiental": uma sala com ambiente *Vulcânico (Fogo)* pode potencializar ataques de fogo ou enfraquecer monstros de gelo. O jogador deve analisar a descrição da sala para preparar sua estratégia antes de iniciar o combate.

2. JUSTIFICATIVA DE COMPLEXIDADE DO SISTEMA

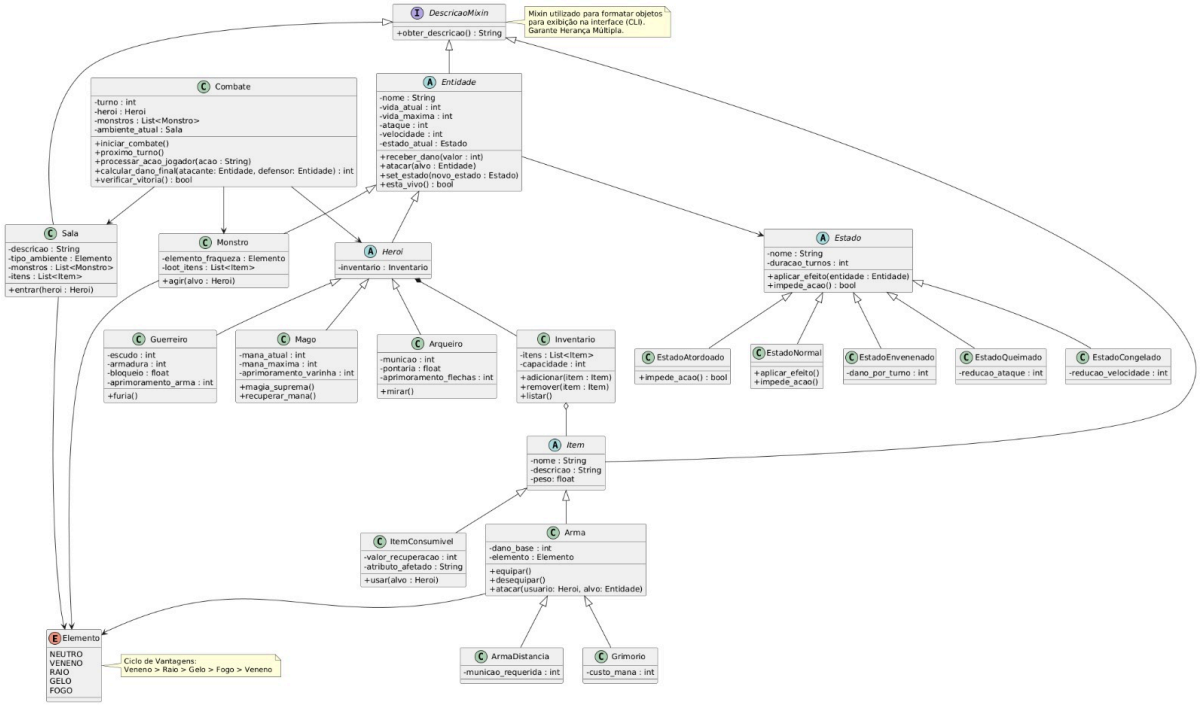
O projeto DungeonPy transcende a implementação de um CRUD simples, caracterizando-se como um sistema de **Domínio Não Trivial** devido à necessidade de gerenciar estados mutáveis complexos e interações polimórficas em tempo real. A complexidade do sistema sustenta-se em três pilares técnicos principais:

1. **Gestão Dinâmica de Estados e Comportamento (State Pattern):** Diferente de sistemas onde atributos são estáticos, as Entidades no DungeonPy sofrem mutações comportamentais temporárias. A implementação de efeitos como *Envenenamento* (dano progressivo) ou *Atordoamento* (interrupção de fluxo) exige o uso de encapsulamento avançado e delegação, onde o objeto não apenas altera seus dados, mas modifica sua lógica de execução durante o ciclo de vida do combate.

2. **Algoritmos de Validação Contextual e Sinergia (Context Object):** O sistema de combate não opera sob uma lógica linear simples. O cálculo de dano exige a orquestração de múltiplas variáveis independentes: a natureza da Arma (Strategy), a vulnerabilidade do Monstro e, crucialmente, o contexto do Ambiente (Sala). Essa integração demanda uma arquitetura que evite o alto acoplamento (muitos if/else aninhados) através do uso extensivo de Polimorfismo e Enumeradores para validação de regras de negócio cruzadas (Ciclo de Vantagens Elementais).

3. **Persistência e Gerenciamento de Recursos (Composition):** O sistema de Inventário e Progressão impõe desafios de gerenciamento de memória e agregação. O herói não é uma entidade isolada, mas um compositor de objetos (Itens, Armas, Habilidades) que alteram seus atributos base. A lógica de "Custo de Oportunidade" no combate força o sistema a tratar cada turno como uma transação que deve validar pré-condições (ex: existência de munição ou mana) antes de executar uma ação, exigindo um tratamento de exceções e fluxo robusto.

3. DIAGRAMA UML



O diagrama apresenta a estrutura estática do sistema DungeonPy, evidenciando as relações de herança, composição e dependência que sustentam as regras de negócio. A arquitetura foi organizada em quatro núcleos principais:

- **Núcleo de Entidades (Polimorfismo):** A classe abstrata Entidade centraliza os atributos vitais e a gestão de estados. Dela derivam as classes concretas Heroi (e suas especializações Guerreiro, Mago, Arqueiro) e Monstro. Essa estrutura permite que o sistema de combate trate personagens e inimigos de forma polimórfica durante o cálculo de dano.

- **Núcleo de Itens (Padrão Strategy):** A hierarquia de itens utiliza o padrão **Strategy** na classe Arma. As subclasses ArmaDistancia e Grimorio encapsulam algoritmos de ataque distintos (consumo de munição e mana, respectivamente), permitindo que o comportamento de ataque do herói mude dinamicamente conforme o item equipado, sem alteração na classe do personagem.

- **Núcleo de Estados (Padrão State):** Para gerenciar efeitos temporários (Veneno, Atordoamento), utilizou-se o padrão **State**. A classe abstrata Estado define o contrato para aplicação de efeitos por turno, enquanto subclasses como EstadoEnvenenado ou EstadoNormal implementam as lógicas específicas, eliminando cadeias condicionais complexas no fluxo principal.

- **Núcleo de Contexto e Ambiente:** A classe Sala atua como agregadora de Monstros e Itens. O Combate orquestra a interação entre todas as entidades, utilizando o Enum Elemento para validar o ciclo de vantagens (Sinergia Elemental) entre a arma do atacante, a fraqueza do defensor e o ambiente da sala.

4. HIERARQUIAS DE CLASSES

4. HIERARQUIAS DE CLASSES

O projeto utiliza o pilar da Herança para promover a reutilização de código e estabelecer contratos de comportamento através do Polimorfismo. O sistema é estruturado sobre árvores de hierarquia que definem os seres vivos, os objetos e os estados do jogo.

4.1. Hierarquia de Entidades (Seres Vivos) A classe base abstrata Entidade define os atributos vitais (vida, defesa, ataque, velocidade) e a gestão de estados (receber_dano, set_estado).

- **Entidade (Abstract):** Generalização máxima de qualquer ser vivo na masmorra, garantindo que Heróis e Monstros possam interagir no combate sob as mesmas regras.
    - **Heroi (Abstract):** Especialização que representa o avatar do jogador. Implementa a lógica de gerenciamento de Inventário e atributos exclusivos de progressão.
        - **Subclasses Concretas:** Guerreiro, Mago e Arqueiro. Cada uma implementa sua própria versão dos métodos de habilidade (furia, magia_suprema, mirar) e gerencia recursos exclusivos (Mana, Munição, Armadura).
    - **Monstro (Concrete):** Especialização controlada pelo sistema (NPC). Possui o atributo elemento_fraqueza para o cálculo de sinergia e define a lista de recompensas (loot_itens) geradas ao ser derrotado.

4.2. Hierarquia de Itens (Inventário e Combate) A classe base abstrata Item permite que o sistema trate objetos de naturezas distintas como elementos armazenáveis no mesmo inventário, compartilhando atributos como nome, peso e descrição.

- **Item (Abstract):** Contrato básico para objetos interativos.

    - **ItemConsumivel:** Representa objetos de uso único que aplicam efeitos imediatos (cura, remoção de status) e são removidos do inventário após o uso.

    - **Arma (Concrete/Base):** Classe que define o dano base e o elemento do ataque. **Por padrão, representa armas de combate corpo a corpo** (como Espadas e Machados) que não exigem recursos extras.
        - **ArmaDistancia:** Especialização para Arqueiros. Sobrescreve o método de ataque para exigir e consumir munição do inventário.
        - **Grimorio:** Especialização para Magos (Strategy). Sobrescreve o método de ataque para exigir e consumir mana do usuário, convertendo o ataque físico em mágico.

4.3. Hierarquia de Estados (Status) Para evitar condicionais complexas, o jogo utiliza uma hierarquia para representar as condições temporárias dos personagens.

- **Estado (Abstract):** Define o contrato para efeitos que duram múltiplos turnos.
    - **EstadoNormal:** Representa a ausência de efeitos nocivos (Padrão Null Object).
    - **Subclasses de Efeito:** EstadoEnvenenado (dano por turno), EstadoCongelado (redução de velocidade), EstadoQueimado (redução de ataque) e EstadoAtordoado (bloqueio de ação).

5. PADRÕES DE PROJETO

Para solucionar a complexidade inerente ao domínio não trivial do sistema, foram aplicados padrões de projeto (Design Patterns) clássicos. A utilização desses padrões visa garantir a extensibilidade do código, a redução do acoplamento e a eliminação de estruturas condicionais complexas.

5.1. Strategy (Estratégia)

- **Aplicação:** Hierarquia de Armas e Combate.

- **Justificativa:** O sistema de combate exige algoritmos de ataque distintos dependendo do item equipado (consumo de munição para arcos, consumo de mana para grimórios ou dano físico direto).
- **Implementação:** Utilizado para encapsular os diferentes algoritmos de cálculo de dano. A classe Heroi delega a execução do ataque para o objeto equipado (interface polimórfica Arma), permitindo que subclasses como Grimorio ou ArmaDistancia injetem suas próprias regras de negócio (custo de recurso e cálculo elemental) sem que a classe do personagem precise ser alterada. Isso favorece o princípio Aberto/Fechado (OCP).

5.2. State (Estado)

- **Aplicação:** Gestão de Efeitos de Status (Envenenado, Congelado, Atordoado).
- **Justificativa:** Entidades sofrem mutações de comportamento temporárias que afetam atributos e a permissão de agir no turno. Controlar isso com *flags* booleanas geraria uma cadeia insustentável de condicionais.
- **Implementação:** Cada condição é representada por uma classe concreta (ex: EstadoEnvenenado, EstadoAtordoado) que herda de Estado. A classe Entidade delega o comportamento do turno para seu estado_atual. Inclui-se também a aplicação do padrão **Null Object** através da classe EstadoNormal, garantindo que a entidade sempre possua um estado válido e eliminando verificações de nulidade.

5.3. Template Method

- **Aplicação:** Método atacar() na classe base Arma.
- **Justificativa:** Todo ataque segue um fluxo lógico comum (verificar condições -> calcular acerto -> calcular dano -> aplicar efeito), variando apenas nos detalhes de consumo de recurso.
- **Implementação:** A classe abstrata Arma define o "esqueleto" do algoritmo de ataque em seu método principal. As subclasses (ArmaDistancia, Grimorio) sobrescrevem apenas os passos específicos (hooks), como a verificação de munição ou mana, reaproveitando a lógica central de cálculo de dano e aplicação de efeitos elementais definida na superclasse.

5.4. Factory Method (Fábrica)

- **Aplicação:** Geração procedural de Monstros e Salas.
- **Justificativa:** O jogo necessita criar inimigos com configurações variadas (nível, elemento, tipo) de forma dinâmica conforme o jogador avança nas salas.
- **Implementação:** O padrão isola a complexidade da criação de objetos da lógica principal do jogo. Uma estrutura de fábrica centraliza a instanciação de monstros, garantindo que as entidades criadas (ex: um *Monstro de Fogo*

Nível 5) sejam instanciadas com os atributos corretos e balanceados para o ambiente da sala, sem expor as classes concretas diretamente ao controlador do jogo.

6. PRINCÍPIOS SOLID APLICADOS

A arquitetura do DungeonPy foi concebida para respeitar os princípios SOLID, garantindo manutenibilidade e extensibilidade ao código.

6.1. SRP (Single Responsibility Principle): Garantimos que cada classe tenha uma única responsabilidade no domínio:

- **Classe Combate:** Responsável exclusivamente pela orquestração do fluxo de turnos e cálculo de dano final. Ela não gerencia o inventário nem decide a IA do monstro.
- **Classe Inventario:** Foca apenas na gestão de armazenamento (adicionar/remover/buscar), sem se preocupar com a lógica de como os itens são usados.
- **Classe Estado:** Responsável isolada pela aplicação de efeitos temporários, retirando essa lógica de dentro da classe Entidade.
- **Classe Sala:** Atua apenas como contexto/container de inimigos e ambiente, sem processar regras de batalha.

6.2. OCP (Open/Closed Principle): O sistema está aberto para extensão, mas fechado para modificação:

- **Sistema de Armas (Strategy):** Graças ao polimorfismo, podemos criar novas subclasses de Arma (como fizemos com Grimorio) sem alterar uma única linha de código na classe Heroi ou na lógica de Combate.
- **Novos Estados:** É possível implementar um EstadoPetrificado apenas criando uma nova classe filha de Estado, sem precisar editar as validações existentes na classe Entidade.

6.3. LSP (Liskov Substitution Principle) As subclasses garantem a intercambialidade sem quebrar o sistema:

- **Polimorfismo de Entidades:** A classe Combate opera sobre a abstração Entidade. Isso significa que um Monstro, um Guerreiro ou um Mago podem ser passados para o método calcular_dano_final() e o sistema funcionará corretamente, pois todos respeitam o contrato base.
- **Substituição de Armas:** Um Grimorio substitui uma Arma comum transparentemente. Embora internamente ele consuma Mana em vez de Munição, para o Heroi que o utiliza, o método de chamada atacar() permanece idêntico.

6.4. ISP (Interface Segregation Principle): Evitamos interfaces (ou classes base) "gordas" que forçam subclasses a implementar métodos inúteis:

- **Segregação de Itens:** Em vez de uma classe Item genérica contendo métodos como equipar() e usar(), separamos a hierarquia. A classe Arma possui equipar(), enquanto ItemConsumivel possui usar(). Dessa forma, uma Poção não é obrigada a implementar lógica de equipamento, e uma Espada não precisa implementar lógica de consumo imediato.
- **Especialização de Armas:** A classe ArmaDistancia implementa gestão de munição, atributo que não existe e não é forçado na classe ArmaBase.

6.5. DIP (Dependency Inversion Principle): O sistema depende de abstrações e não de implementações concretas:

- **Injeção de Dependência no Combate:** A classe Combate não depende das classes concretas Orc ou Guerreiro, mas sim de listas genéricas de Monstro e da abstração Heroi.
- **Abstração de Estado:** A Entidade depende da classe abstrata Estado, e não de EstadoEnvenenado. Isso permite que os efeitos sejam trocados dinamicamente em tempo de execução sem acoplamento rígido.