Universidade Estadual de Campinas
Instituto de Computação

Laboratório 1

RPG - Jogo Narrativo

MC322 - Programação Orientada a Objetos

Professor: Marcelo da Silva Reis
PEDs: Athyrson Machado Ribeiro / Giorgio de Moraes Rossa

---

1. Descrição Geral

No universo dos jogos digitais, os RPGs (Role-Playing Games) representam um dos gêneros mais influentes e complexos. Desde épicos de fantasia medieval a aventuras de ficção científica, esses jogos cativam milhões de jogadores ao combinar narrativas ricas com sistemas profundos de progressão de personagens e mundos interativos. A complexidade por trás de suas regras, interações e eventos os torna um campo de estudo ideal para a engenharia de software.

Inspirados por este universo, durante a disciplina de MC322, construiremos um projeto de Jogo de RPG com o objetivo de introduzir os alunos à lógica de programação e modelagem de sistemas orientados a objetos.

Ao longo deste projeto, desenvolveremos um sistema incremental, aplicando conceitos fundamentais da programação orientada a objetos, como encapsulamento, herança e polimorfismo, enquanto damos vida a heróis, monstros e aventuras em um mundo virtual interativo.

1.1. O que é um RPG?

RPG, ou Role-Playing Game, é um gênero de jogo no qual os participantes assumem os papéis de personagens dentro de um universo ficcional. O cerne da experiência é a imersão e a narrativa, onde cada jogador controla as ações de seu personagem, toma decisões que afetam a história e interage com o mundo e seus habitantes, que podem ser controlados por outros jogadores ou por um mestre de jogo (em jogos de mesa) ou pelo próprio sistema (em videogames). Cada personagem é único, definido por uma série de atributos (como força, inteligência, agilidade), habilidades, uma história de fundo e uma classe (como guerreiro, mago ou ladino), permitindo que os jogadores explorem a fantasia de ser outra pessoa.

O progresso é um pilar fundamental da mecânica de um RPG. Conforme os personagens superam desafios, como completar missões, resolver quebra-cabeças ou derrotar inimigos em combate, eles ganham pontos de experiência (XP). Acumular XP suficiente permite que o personagem "suba de nível", tornando-se mais poderoso ao melhorar seus atributos, aprender novas habilidades e ganhar acesso a equipamentos melhores. Este ciclo contínuo de enfrentar desafios, ser recompensado e fortalecer o personagem cria um senso de crescimento e desenvolvimento que motiva o jogador a explorar o mundo e avançar na história principal.

1.2. Principais Componentes de um Jogo de RPG

- **Sistema de Personagens:** Representação do avatar do jogador e de outros personagens (NPCs). Inclui seus atributos (força, agilidade, inteligência), classes (guerreiro, mago), habilidades, inventário e o sistema de progressão (níveis e pontos de experiência).
- **Mundo Virtual:** Define o espaço onde a aventura acontece (cidades, masmorras, florestas, etc.). Inclui NPCs com quem interagir, missões (quests) para completar, itens, obstáculos e a própria narrativa que ambienta o jogo.
- **Sistema de Navegação e Interação:** Define como os personagens se movem pelo mundo (baseado em grades/tiles, livre, ponto e clique) e como interagem com objetos e NPCs. Gerencia a exploração e o acionamento de eventos no mapa.
- **Interface do Jogador (UI) e Estado do Jogo:** Elementos visuais que comunicam o estado atual do jogo e do personagem ao jogador. Inclui barras de vida e mana, minimapa, janelas de diálogo, tela de inventário e diário de missões. Permite que o jogador "perceba" o mundo e gerencie seu personagem.
- **Motor de Regras e Mecânicas de Jogo:** O cérebro do RPG. Algoritmos que definem como as ações são resolvidas com base nos atributos e nas regras do jogo. Governa o cálculo de dano, testes de habilidade, o ganho de experiência e as consequências das escolhas do jogador.

- **Sistema de Combate:** O conjunto de regras que governa os confrontos. Pode ser baseado em turnos ou em tempo real, e define como ataques, defesas, magias e itens são usados em batalha. Inclui a inteligência artificial (IA) que controla o comportamento dos inimigos.
- **Gerenciador de Narrativa e Missões (Quest System):** Controla o fluxo da história principal e das missões secundárias. Rastreia o progresso do jogador, gerencia os diálogos, libera novos eventos e garante que a jornada do personagem tenha uma progressão coesa e interessante.

2. Objetivos

Os principais objetivos deste laboratório são:

- Familiarizar-se com a linguagem Java (Versão 21) e a IDE escolhida (VSCode);
- Compreender os conceitos fundamentais de Programação Orientada a Objetos: classes, atributos e métodos;
- Aplicar o conceito de Herança para criar especializações, modelando uma hierarquia de personagens;
- Utilizar Classes Abstratas para definir conceitos e contratos que devem ser implementados por subclasses;
- Implementar um pequeno sistema funcional que demonstre a interação entre objetos de diferentes classes.

3. Classes

Esta seção descreve a implementação principal do laboratório. As classes a seguir representam a estrutura mínima esperada, mas sinta-se encorajado(a) a expandi-la.

As seguintes diretrizes gerais devem ser observadas:

- **Liberdade Criativa:** Fique à vontade para criar classes auxiliares, atributos e métodos extras que julgar necessários para enriquecer seu projeto. O tema do RPG também é de sua livre escolha (ex: fantasia medieval, Pokémon, dinossauros, ficção científica, etc.).
- **Autonomia Técnica:** Este enunciado é intencionalmente vago em alguns aspectos, pois parte da atividade é a tomada de decisão de design. É esperado que você:
  - Identifique os tipos de variáveis mais adequados para cada atributo (ex: `int`, `String`, `double`).
  - Determine quais parâmetros são os mais adequados para cada método, com base na sua função.

3.1. Classe Abstrata `Personagem`

Esta classe representará a base para qualquer entidade viva no mundo do jogo. Sendo abstrata, não pode ser instanciada diretamente.

- **Atributos:**
  - `nome` - Nome do personagem;
  - `pontosDeVida` - A saúde atual do personagem;
  - `forca` - O atributo de força, usado em combate.
- **Métodos:**
  - Construtor para inicializar os atributos;
  - `receberDano()` - Reduz os pontos de vida com base no dano recebido;
  - `exibirStatus()` - Imprime as informações do personagem (nome, vida, etc.).
  - `atacar(Personagem alvo)` **(Método Abstrato)** - Um contrato que obriga todas as classes filhas a implementar sua própria lógica de ataque.

3.2. Classe Abstrata `Heroi`

Esta classe herda de `Personagem` e serve como base para todas as classes jogáveis.

- **Atributos Adicionais:**
  - `nivel` - O nível atual do herói;
  - `experiencia` - Pontos de experiência acumulados.
- **Métodos Adicionais:**
  - Construtor que utiliza o `super()` para inicializar os atributos da classe `Personagem`;
  - `ganharExperiencia()` - Adiciona pontos de experiência ao héroi;
  - `exibirStatus()` - Sobrescreve (*override*) o método da classe pai para incluir o nível e a experiência.
  - `usarHabilidadeEspecial(Personagem alvo)` **(Método Abstrato)** - Contrato que obriga as classes concretas de heróis a terem uma habilidade única.

3.3. Classes Concretas de Heróis

Estas são as classes jogáveis que herdam de `Heroi`. Exemplos de classes comuns em RPGs: Guerreiro, Arqueiro, Mago, etc[1].

- Implemente duas classes concretas de héroi.
- Adicione um atributo único para cada uma (ex: `furia` para um Guerreiro, `precisao` para um Arqueiro).
- Forneça a implementação concreta para os métodos abstratos `atacar()` e `usarHabilidadeEspecial()`. Nesse lab a implementação pode ser um simples `System.out.println` descrevendo a ação.

3.4. Classe Abstrata `Monstro`

Esta classe herda de `Personagem` e serve como base para todos os inimigos do jogo.

- **Atributos Adicionais:**
  - `xpConcedido` - Experiência que o monstro concede ao ser derrotado.
- **Métodos Adicionais:**
  - Construtor que utiliza o `super()` para inicializar os atributos da classe `Personagem`;
  - `exibirStatus()` - Sobrescreve o método para incluir o XP que será concedido.

3.5. Classe Concreta de Monstro

Um tipo de inimigo que herda de `Monstro`.

- Implemente pelo menos duas classes concretas de Monstro Ex: `Goblin, Esqueleto, Fantasma etc.` com atributos e comportamentos únicos entre si.
- Forneça a implementação concreta para o método abstrato `atacar()`.

3.6. Classe `Main` (Cenário de Sobrevivência)

A classe `Main` simulará um desafio de sobrevivência. O objetivo é criar um cenário onde um único herói deve enfrentar e sobreviver a três encontros consecutivos com monstros diferentes, um em cada turno.

- **Criação dos Personagens:**
  - Crie uma única instância de um herói de sua escolha (ex: um `Guerreiro`).
  - Crie três instâncias de monstros diferentes (ex: um `Goblin`, um `Esqueleto` e um `Orc`). Você pode organiza-los em um array do tipo `Monstro[]`.
- **Apresentação do Desafio:**

[1] Veja mais exemplos em https://abcdorpg.com/classes-do-rpg-medieval/

- Imprima uma mensagem inicial apresentando o desafio, como O HERÓI ENTRA NA MASMORRA PARA ENFRENTAR TRÊS DESAFIOS! – Use sua própria mensagem de introdução dando um resumo do cenário que vocês imaginaram.
- Exiba o status inicial do herói de uma forma elegante (usando o método exibirStatus()).

- **Simulação dos Turnos em Loop:**
  - Crie um laço de repetição (*loop*) que execute 3 vezes (um para cada turno/monstro).
  - Dentro do laço, a cada turno:
    - Anuncie a chegada do monstro daquele turno.
    - O herói ataca o monstro.
    - O monstro ataca o herói de volta.
    - Verifique se o herói sobreviveu. Se os pontos de vida do herói chegarem a zero ou menos, o laço deve ser interrompido e uma mensagem de "Game Over"deve ser exibida.
    - Exiba o status do herói e do monstro ao final de cada turno.

- **Conclusão do Desafio:**
  - Se o herói sobreviver aos três turnos, imprima uma mensagem de vitória no final.

4. Avaliação

Atenção para os seguintes pontos que serão considerados na avaliação:

- **Diferenciação e Criatividade:** A criatividade será observada na implementação das classes concretas. Heróis e monstros devem ter atributos e descrições de ações (ataques, habilidades) que os tornem únicos entre si.
- **Organização do Projeto:** Cada classe deve estar em seu próprio arquivo `.java`, seguindo as convenções da linguagem.
- **Clareza do Código:** Use comentários para descrever o propósito de cada classe e para explicar trechos de lógica que não sejam triviais.
- **Escopo do Laboratório:** Não utilize entrada de dados pelo teclado neste laboratório; todos os dados devem ser pré-definidos no código.
- **Apresentação do Resultado:** A clareza e a organização da saída de texto no console são fundamentais. Use cabeçalhos (ex: — INÍCIO DO TURNO —), espaçamento e mensagens descritivas para que a simulação seja fácil de ler e acompanhar.
- **Funcionalidade e Execução:** O projeto entregue deve compilar e executar sem erros no ambiente de correção especificado (Java 21). É responsabilidade do aluno garantir a compatibilidade e testar seu código antes da submissão. Projetos que não executarem devido a erros de código ou incompatibilidade de versão serão severamente penalizados.

5. Entrega

A entrega do laboratório deve ser feita via repositório no GitHub.

- **A entrega do Laboratório é realizada exclusivamente via Github**[2] Para a submissão no Github, gere um release (tag) com a identificação do laboratório.
- Utilize os horários de laboratório e atendimentos para tirar eventuais dúvidas de submissão e também relacionadas ao desenvolvimento do laboratório.

[2]Você deve criar um link e enviar no Google Classroom