Trabalho final de Programação Orientada a Objetos – ECOP13A

UNIFEI - Universidade Federal de Itajubá
Prof. João Paulo R. R. Leite (joaopaulo@unifei.edu.br)

O objetivo deste trabalho é aplicar os conceitos de Programação Orientada a Objetos (POO) em C++, desenvolvendo um jogo de RPG baseado em turnos. Você deverá utilizar todo o conteúdo aprendido durante a disciplina (**Classes, Herança, Polimorfismo, Tratamento de Exceções, Templates, STL etc.**).

---

1. Descrição do Jogo

Neste trabalho, você terá que desenvolver um jogo de RPG baseado em turnos, utilizando os principais conceitos de Programação Orientada a Objetos em C++. O jogo será inteiramente em ambiente textual (terminal) e deverá conter os elementos essenciais de um RPG.

A temática do jogo é livre: você pode criar um RPG medieval, espacial, *steampunk*, *cyberpunk*, entre outros. A originalidade e criatividade na construção do universo do jogo serão valorizadas.

Apesar da liberdade de ambientação e narrativa, o jogo deverá obrigatoriamente implementar as mecânicas descritas na seção 1.1. Além disso, você deverá escolher e implementar pelo menos duas mecânicas intermediárias e pelo menos uma mecânica avançada, conforme listadas nas seções seguintes.

O jogo deverá possuir um fluxo bem definido, com **início, desenvolvimento e fim**, representando uma narrativa **coerente e jogável**. O jogador deve ser capaz de explorar o mundo, enfrentar desafios, adquirir recompensas e, eventualmente, completar sua missão principal ou vencer o desafio final proposto.

Todos os sistemas devem ser implementados com o uso de **classes**, promovendo o uso de **encapsulamento, herança, polimorfismo e abstração**. Seu código deve ser modular, organizado e orientado a objetos.

1.1. Mecânicas Principais (obrigatórias)

Sistema de Inventário

- **Descrição**: Gerencia os objetos coletados pelo jogador durante a aventura. Permite adicionar, remover e visualizar objetos, como armas, armaduras, itens, poções e outros objetos.
- **Exemplos de classes**: `Inventario`, `Item`, `Pocao`, `Armamento`, `Arma`, `Armadura`.

Sistema de Batalha por Turnos

- **Descrição**: Controla as batalhas em que jogadores e inimigos se alternam realizando ações (ataques, habilidades, defesas). Cada participante age em seu turno.
- **Exemplos de classes**: `Batalha`, `Turno`, `Habilidade`.

Progressão de Personagem

- **Descrição**: Permite que o jogador evolua ao longo do jogo, aumentando atributos (força, defesa, vida), subindo de nível e desbloqueando novas habilidades.
- **Exemplos de classes**: `Personagem`, `Jogador`, `ClassePersonagem`.

Sistema de Inimigos e Chefes

- **Descrição**: Define os oponentes do jogador, variando em força, comportamento e habilidades. Chefes são inimigos mais fortes, geralmente relacionados a momentos importantes da história. **É recomendado que se tenha diferentes tipos de inimigos, como por exemplo Vampiro, Lobisomem, Caveira, etc.**
- **Exemplos de classes**: `Inimigo`, `Chefe`.

Sistema de Missões

- **Descrição**: Controla a história e a progressão principal do jogo através de missões (objetivos a serem cumpridos). Pode incluir missões principais e secundárias, com recompensas ao serem concluídas. **Lembre-se de desenvolver uma história com começo, meio e fim.**
- **Exemplos de classes**: `Missao`, `MissaoPrincipal`, `Recompensa`.

1.2. Mecânicas Intermediárias (escolha pelo menos duas)

Exploração de Cenários

- **Descrição**: Permite que o jogador se mova entre diferentes áreas do jogo (cidades, masmorras, florestas), encontrando eventos, inimigos ou recompensas.
- **Exemplos de classes**: `Cenario`, `Mapa`, `Masmorra`, `Cidade`.

Sistema de Habilidades

- **Descrição**: Permite que personagens usem habilidades especiais além dos ataques comuns, como magias, golpes poderosos ou *buffs*.
- **Exemplos de classes**: `Habilidade`, `TipoHabilidade`.

Loja e Economia

- **Descrição**: Estabelecimentos onde o jogador pode comprar ou vender itens usando alguma forma de moeda ou recurso.
- **Exemplos de classes**: `Loja`, `Transacao`, `Carteira`.

Diálogo com NPCs

- **Descrição**: Implementa interações entre o jogador e personagens não-jogáveis (NPCs), podendo influenciar o progresso do jogo ou fornecer informações e missões.
- **Exemplos de classes**: `NPC`, `Dialogo`, `OpcaoDialogo`.

Sistema de Status Temporário

- **Descrição**: Aplica efeitos positivos (*buffs*) ou negativos (*debuffs*) aos personagens por tempo limitado durante as batalhas (ex.: envenenamento, aumento de ataque, lentidão).
- **Exemplos de classes**: `Status`, `Efeito`, `Condicao`.

Sistema de Raridade

- **Descrição**: Classifica itens e inimigos em diferentes níveis de raridade (comum, raro, épico, lendário), afetando seus atributos e a probabilidade de encontrá-los.
- **Exemplos de classes**: `Raridade`, `TabelaDrop`.

1.3. Mecânicas Avançadas (escolha pelo menos uma)

Sistema de Crafting

- **Descrição**: Permite combinar materiais e itens para criar novos equipamentos, poções ou objetos especiais.
- **Exemplos de classes**: `Receita`, `SistemaDeCriacao`.

Eventos Aleatórios

- **Descrição**: Introduz situações inesperadas que podem acontecer durante a exploração ou batalhas (ex.: armadilhas, tesouros escondidos, inimigos surpresa).
- **Exemplos de classes**: `EventoAleatorio`, `Chance`, `TabelaEvento`.

Sistema de Dia e Noite

- **Descrição**: Simula a passagem do tempo no mundo do jogo, alterando cenários, comportamento de NPCs, surgimento de inimigos ou eventos especiais.
- **Exemplos de classes**: `Tempo`, `Relogio`, `FaseDoDia`.

Armadilhas e Quebra-Cabeças

- **Descrição**: Introduz desafios além do combate, como armadilhas em masmorras ou enigmas que o jogador deve resolver para prosseguir.
- **Exemplos de classes**: `Armadilha`, `Desafio`, `Sala`.

Companheiros/NPC aliados

- **Descrição**: Permite que personagens controlados pelo jogador ou computador acompanhem o jogador e o auxiliem em batalhas ou eventos importantes.
- **Exemplos de classes**: `Aliado`.

2. Requisitos Técnicos

Para garantir a aplicação adequada dos conceitos de Programação Orientada a Objetos (POO) em C++, o jogo deve atender aos seguintes requisitos técnicos:

2.1. Estrutura e Organização

- O projeto deve estar **organizado em múltiplos arquivos `.cpp` e `.h`**.
- O código deve ser **modular e bem documentado**, com comentários claros explicando o propósito das classes e funções.

2.2. Orientação a Objetos

- **Todos os sistemas do jogo devem ser implementados com o uso de classes.**
- É obrigatório implementar **no mínimo 15 classes distintas**. Estas devem cobrir os principais sistemas do jogo e as mecânicas escolhidas (por exemplo: `Jogador`, `Guerreiro`, `Inventário`, `Arma`, `Poção`, `Cenário`, `Floresta`, `Chefe`, `Loja`, `Missão`, `Status` etc.).
- A estrutura deve contemplar os **principais pilares da POO** (alguns exemplos estão mostrados abaixo):
  - **Herança**: ex.: `Jogador`, `Inimigo` e `Chefe` herdando de `Personagem`; `Floresta` e `Loja` herdando de `Cenario`.
  - **Polimorfismo**: utilização de métodos virtuais sobrescritos, como `usarHabilidade()` ou `atacar()`.
  - **Encapsulamento**: atributos privados com métodos de acesso (`get`, `set`) quando necessário.

- **Abstração**: definição de classes base que representam conceitos genéricos, como `Item`, `Personagem`, `Cenario`.

2.3. Templates e STL

- O jogo deve utilizar **pelo menos um template próprio**, ou seja, uma classe ou função genérica utilizando `template<typename T>`.
- Um exemplo de aplicação prática é criar o **sistema de inventário** como um template. Isso permite que o jogador possua um inventário para itens (Acessório, Consumível, Poção, etc) e outro para armamento (Arma, Espada, Armadura etc.).
- Utilizar **estruturas da STL** (Standard Template Library), como `vector`, `map`, `set`, `queue` etc., para organizar listas, inventários, filas de inimigos, registros de eventos etc.

2.4. Tratamento de Exceções

- Usar `try`/`catch` para tratar erros comuns (alguns exemplos estão mostrados abaixo):
  - Entrada de dados inválida pelo usuário.
  - Acesso fora dos limites de vetores.
  - Erros de carregamento de arquivos (se aplicável).

2.5. Mecânicas e Funcionalidades

- O jogo deve implementar **todas as mecânicas principais** apresentadas na descrição do jogo.
- Além disso, deve conter **pelo menos duas mecânicas intermediárias** e **uma avançada**, conforme a lista fornecida na descrição.
- Cada uma dessas mecânicas deve ser implementada por meio de **classes e relacionamentos bem definidos**.

3. Entrega

A entrega deve ser um arquivo zip contendo:

- **Os arquivos `.cpp` e `.h`**, organizados em uma estrutura clara.
- **Um documento em pdf descrevendo o jogo desenvolvido** (temática, narrativa). Além disso, liste as classes usadas para implementar o jogo, incluindo uma breve descrição da função de cada uma dentro do programa. **Incluir os nomes e números de matrícula no documento.**
- O trabalho valerá **50% da nota 2** de ECOP13A.

4. Critérios de avaliação

Critério	Descrição	Pontuação
1. Estrutura e Organização do Código	Código modular, bem comentado, com nomes significativos. Separação correta entre arquivos `.h` e `.cpp`. Arquitetura clara e compreensível.	**20 pts**
2. Aplicação dos Conceitos de POO	Uso adequado de classes, herança, polimorfismo, encapsulamento, construtores e métodos. Implementação de **no mínimo 15 classes distintas**.	**25 pts**
3. Funcionalidade e Jogabilidade	Jogo funcional com início, meio e fim. Implementação completa das mecânicas principais: batalhas por turnos, progressão do personagem, sistema de inventário, etc.	**25 pts**
4.Mecânicas Intermediárias e Avançadas	Implementação de pelo menos **duas mecânicas intermediárias** e **uma avançada**, escolhidas da lista proposta.	**20 pts**
5. Tratamento de Exceções	Uso correto de `try/catch` para capturar exceções relevantes (ex.: entrada inválida, ponteiro nulo, acesso fora do vetor).	**5 pts**
6. Templates e Uso de STL	Uso de pelo menos **um template próprio** e **estruturas da STL** (`std::vector`, `std::map`, etc).	**5 pts**