UNIVERSIDADE FEDERAL DE OURO PRETO
CAMPUS MORRO DO CRUZEIRO – OURO PRETO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Samara Paloma Lopes Augusto Ribeiro

Trabalho Prático 1 – Protótipo de Jogo de Simulação de Estratégia

OURO PRETO

2025

Instruções de compilação e execução do programa utilizando terminal ou prompt de comando = Comando ‘make run’ para compilação e execução

OBS = Comando ‘make clean’ para desfazer a compilação e execução

---

SUMÁRIO

3. CONCLUSÃO

---

1. INTRODUÇÃO

Este trabalho acadêmico tem como objetivo a implementação de um sistema de simulação de batalhas entre exércitos. O foco principal é a **estruturação de classes em C++** que representam as entidades do sistema, **utilizando conceitos fundamentais da programação orientada a objetos, como herança, polimorfismo, abstração e coesão.**

A estrutura proposta segue os princípios de alta coesão e baixo acoplamento, visando a criação de um código modular e eficiente. Cada classe foi projetada para ter uma responsabilidade única e bem definida, além do mais, o uso de herança garante que o sistema seja flexível e reutilizável, possibilitando a implementação de diferentes tipos de unidades de combate e a simulação de diferentes cenários de batalha.

O trabalho, portanto, oferece uma solução para a criação de um protótipo de jogo de simulação estratégica, com base nos conceitos de programação orientada a objetos, e segue a estrutura proposta no enunciado, visando a implementação de uma solução eficiente e de fácil entendimento.

1.1. Objetivo Geral

O objetivo geral deste trabalho acadêmico é desenvolver um sistema de simulação de batalhas entre exércitos compostos por unidades de Veículos, Aeronaves e Infantarias. O sistema irá simular batalhas de ida e volta, calculando o poder de ataque das unidades de forma aleatória, registrando vitórias, derrotas e empates. Além disso, o projeto permitirá o gerenciamento de exércitos, com a adição e monitoramento de unidades, e fornecerá relatórios sobre o desempenho das batalhas e das campanhas.

1.2. Objetivos Específicos

- Desenvolver as classes que representam as unidades: **Veículo**, **Infantaria** e **Aeronave**, garantindo que cada uma possua atributos específicos relacionados às suas funções e métodos para calcular o poder de ataque individual, utilizando fórmulas que considerem suas características principais e um elemento de aleatoriedade para maior dinamismo.
- Criar a **classe Exército** que representa um conjunto de unidades organizadas para batalhas. Cada exército possui um nome, um registro de resultados das batalhas (vitórias, empates e derrotas) e uma lista de unidades. Ao ser criado, o exército inicializará com cinco unidades: **duas de Infantaria**, **duas de Veículo e uma de Aeronave**. A classe então incluirá métodos para adicionar unidades, exibir informações sobre elas, registrar os resultados das batalhas, zerar o histórico e retornar o nome e a lista de unidades do exército. Essa classe é projetada para gerenciar e monitorar o desempenho das unidades em combate.
- Implementar a **classe Batalha** que organiza combates entre dois exércitos e é responsável por calcular os resultados das batalhas. Ela determinará o poder de ataque total de cada exército, somando o poder de ataque de todas as suas unidades. Além disso, registrará os resultados (vitória, derrota ou empate) para cada exército, contabilizando destruições em unidades no caso de derrota. A classe também permitirá consultar o exército vencedor, o resultado da batalha em formato de placar e os detalhes dos exércitos envolvidos.
- Criar a **classe Campanha** que é responsável por organizar e executar batalhas entre dois exércitos. Ela registrará os resultados de cada batalha (ida e volta), armazena os dados relevantes, como a data da campanha, e fornece relatórios detalhados sobre o desempenho de cada exército. Através da classe, é possível determinar o vencedor de uma campanha, gerar tabelas de posições, e identificar o exército mais destrutivo. Essa implementação é essencial para realizar simulações estratégicas e análises dos confrontos.
- Criar a **classe `Date`** que representa uma data, com funcionalidades para validar, avançar dias, randomizar e formatar datas.
- Garantir que a estrutura de dados seja eficiente e que a simulação seja fluida.

1.3. Justificativa

A necessidade de simulações realistas e eficazes em jogos de estratégia é considerável. Neste trabalho, a escolha pela programação orientada a objetos visa garantir a flexibilidade, escalabilidade e manutenção do sistema, permitindo a criação de um protótipo de jogo estratégico. A utilização de conceitos como herança, polimorfismo e abstração proporciona um design modular, onde as entidades são facilmente manipuladas e expandidas. Esse modelo viabiliza a simulação das batalhas, tornando a experiência mais dinâmica e realista. Assim, este trabalho contribui para a construção de uma base sólida no desenvolvimento de jogos de estratégia e simulações de combate, aplicando os conceitos de POO para garantir a eficiência e a reutilização do código.

---

2. DESENVOLVIMENTO

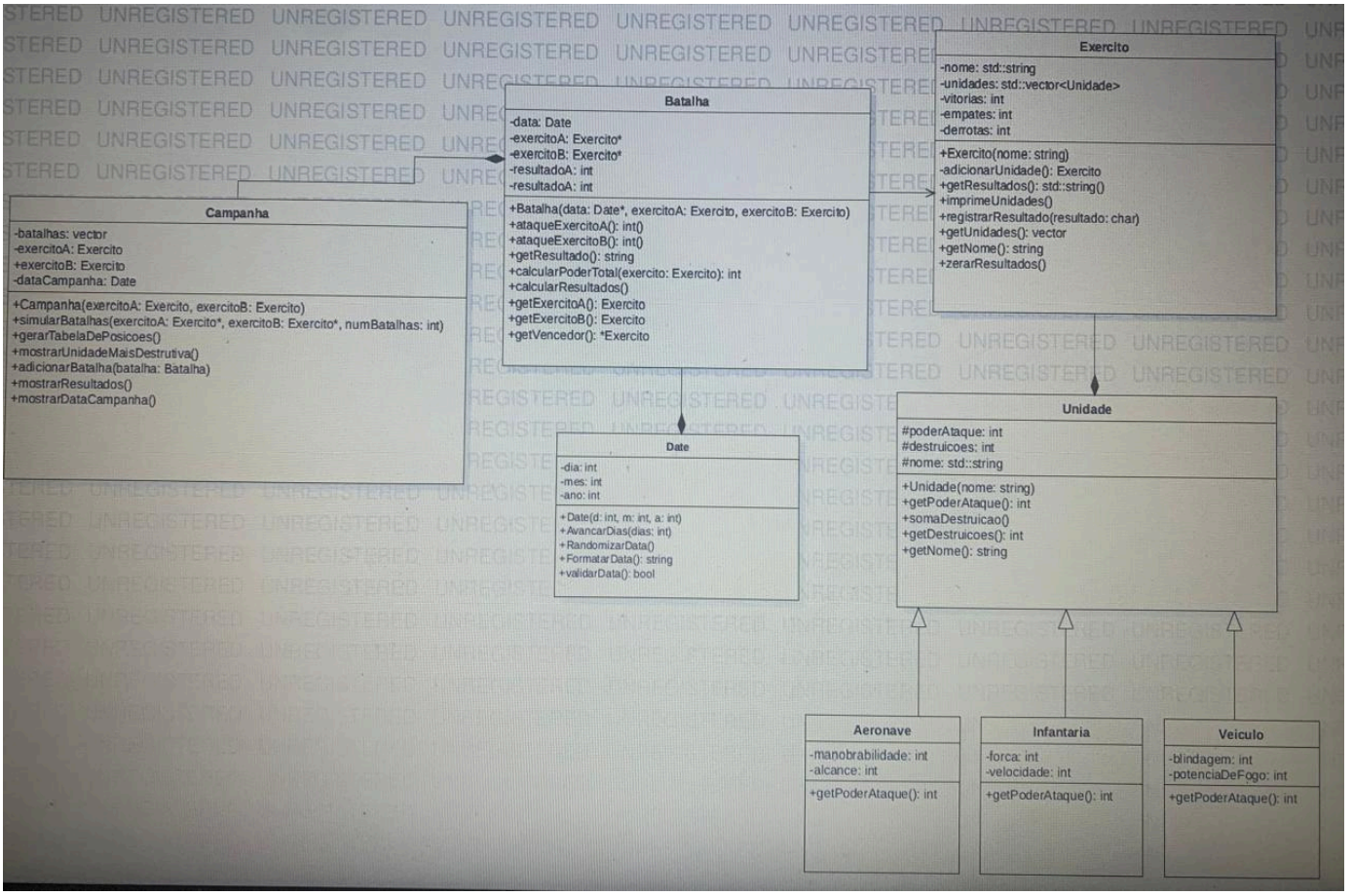
Neste desenvolvimento, o sistema de simulação de batalhas entre exércitos foi projetado utilizando conceitos de Programação Orientada a Objetos, garantindo um código modular, flexível e de fácil manutenção, conforme os requisitos. O código principal, localizado no arquivo `main`, gerencia a execução do sistema e a interação entre as diferentes classes: `Exercito`, `Batalha`, `Date`, `Campanha`, e as unidades de combate (Veículos, Aeronaves e Infantarias).

As classes foram organizadas em arquivos de cabeçalho (`.h`) e implementação (`.cpp`), garantindo uma separação clara entre a definição das classes e seus métodos, o que facilita a manutenção e extensão do código. A estrutura do código segue boas práticas de modularidade e reutilização.

2.1. Especificação de Requisitos

O sistema de simulação de batalhas gerencia exércitos compostos por três tipos de unidades: veículos, aeronaves e infantarias. A **classe Unidade** é a base para as demais, definindo atributos e métodos gerais, **enquanto as classes Veículo, Aeronave e Infantaria** estendem essa funcionalidade, especificando atributos como blindagem, manobrabilidade e quantidade de soldados, e calculando o poder de ataque com fórmulas que incorporam fatores aleatórios. A **classe Exército** organiza as unidades, calcula o poder total e registra os resultados das batalhas, enquanto a **classe Batalha** simula confrontos entre dois exércitos, determinando o vencedor com base no poder de ataque de cada um. Já a **classe Campanha** permite a organização de uma sequência de batalhas, registrando o desempenho geral e gerando relatórios detalhados, armazenados em arquivos de texto. O sistema **também inclui a classe Date**, que gerencia o progresso temporal das campanhas. Sua arquitetura modular, com separação em arquivos de cabeçalho e implementação, facilita a manutenção e expansão. A imprevisibilidade dos resultados é garantida pelo uso de números aleatórios no cálculo do poder de ataque, tornando cada execução única.

2.2. Diagrama de Classes



O diagrama de classes representa a estrutura do sistema, com as seguintes interações entre as classes:

- O relacionamento entre as classes **`Unidade`, `Veiculo`, `Infantaria`** e **`Aeronave` segue o conceito de herança**. A classe `Unidade` é a classe base e as classes `Veiculo`, `Infantaria` e `Aeronave` são subclasses que herdam de `Unidade`. A seta no diagrama UML representa essa relação de herança entre elas, com a classe derivada apontando para a classe base.
- A classe **`Exercito`** é uma composição de unidades, **e tem uma relação direta com a classe `Unidade`.** Essa relação é de **composição**, onde um exército contém várias unidades de diferentes tipos, como `Infantaria`, `Veiculo` e `Aeronave`. A seta representando a composição é uma linha sólida com um losango preenchido na ponta, que vai de `Exercito` para `Unidade`. Isso mostra que o exército tem uma forte dependência das

unidades. As unidades não podem existir sem o exército, e quando o exército é destruído, as unidades também são destruídas automaticamente.

- A relação entre a **classe `Batalha` e a classe `Exercito`** pode ser descrita como **uma associação do tipo bidirecional**. A classe `Batalha` interage com dois objetos da classe `Exercito`, e essa relação pode ser representada no diagrama UML com uma linha simples (sem losango), indicando que a `Batalha` tem uma associação com os dois `Exercito`.
- A relação entre a **classe `Date` e a classe `Batalha` é de composição**, a seta representando a composição é uma linha sólida com um losango preenchido na ponta, que vai de `Batalha` para `Date`.
- A relação entre **a classe `Campanha` e a classe `Batalha` pode ser descrita como composição**, representada por uma seta preenchida apontando do lado da `Campanha` para o lado da `Batalha`.

---

3. CONCLUSÃO

O desenvolvimento deste sistema de simulação de batalhas permitiu a aplicação prática de conceitos fundamentais da programação orientada a objetos. A implementação das classes `Exercito`, `Batalha`, `Campanha` e `Date` demonstrou como a organização do código em classes bem estruturadas, com responsabilidades claras, pode resultar em um sistema modular e de fácil manutenção.

A separação do código em arquivos `.h` e `.cpp`, utilizando a técnica de encapsulamento, garantiu que as classes fossem responsáveis apenas por suas respectivas funcionalidades, respeitando o princípio da coesão. Além disso, o uso de herança e polimorfismo contribuiu para a flexibilidade do sistema, permitindo que diferentes tipos de unidades e batalhas fossem simuladas de forma eficiente.

Com esse trabalho, foi possível consolidar o entendimento sobre a importância da POO no desenvolvimento de sistemas complexos e na modularização do mundo

real através de código, como jogos de estratégia. Através da prática de dividir o sistema em entidades bem definidas e de baixo acoplamento, o projeto foi capaz de demonstrar como esses conceitos podem ser utilizados para criar soluções robustas e escaláveis, que podem ser facilmente modificadas e expandidas no futuro.