isec Engenharia

LICENCIATURA ENGENHARIA INFORMÁTICA - CURSO EUROPEU

RELATÓRIO POO

DAVID OLIVEIRA - 20221470717
DAVID ARAÚJO - 20232128810

ANO LETIVO 2025/2026

Índice

1. Introdução e Organização do Projeto

A Meta 1 do projeto focou-se na construção da arquitetura do simulador, dando especial atenção à robustez e à interface de comandos.

O projeto foi planeado para ter uma separação clara entre a interface e a implementação, usando ficheiros .h e .cpp, distribuídos nas pastas include e src. Esta organização facilita a clareza do código e a gestão das dependências entre os módulos.

A classe Settings é fundamental nesta arquitetura, centralizando todas as constantes e valores numéricos do projeto, como agua_min e a capacidade do regador.

A hierarquia de classes do simulador seguiu as especificações. O Jardim agrega o Jardineiro, numa relação de composição. A Celula é composta por Planta* e Ferramenta*, permitindo que contenha várias plantas e ferramentas, representadas por ponteiros. As classes Planta e Ferramenta possibilitam a criação de diferentes tipos de plantas (como cacto, flor) e ferramentas (como regador, pá) com uma interface comum, mas implementações específicas.

Todos os objetos no simulador são criados dinamicamente e destruídos de forma controlada.

A validação de comandos é processada no método Comando::processar(). Este método organiza os comandos em três grupos com base no número de *tokens*. O primeiro grupo inclui comandos de 1 *token*; o segundo grupo abrange comandos de 2 *tokens*; o terceiro grupo compreende comandos de 3 *tokens*.

2. Estrutura de Dados e Implementação Base

2.1. Classe `Jardim` e Restrições de Memória

A classe `Jardim` é o contentor central da simulação.

- **Estrutura da Grelha:** Em cumprimento à restrição de não utilizar coleções da biblioteca standard para o armazenamento do solo, a área do jardim (`grelha`) foi implementada como uma matriz de ponteiros para `Celula` (`Celula** grelha`). Esta alocação dinâmica garante que apenas a memória estritamente necessária é utilizada.
- **Gestão de Memória:** O construtor `Jardim::Jardim` aloca a grelha e o objeto `Jardineiro` (Composição), enquanto o destrutor `Jardim::~Jardim` liberta corretamente todos os blocos de memória alocados dinamicamente (`delete[] grelha[i]`, `delete[] grelha` e `delete jardineiro`), prevenindo *memory leaks*.
- **Interface:** Foram implementados *getters* (`getLinhas()`, `getColunas()`) em `Jardim.h` para que outras classes (como `Comando`) possam consultar as dimensões da grelha, essencial para a validação semântica das posições.

2.2. Classe `Celula`

A classe `Celula` representa um bocado de solo e gere o seu estado (água e nutrientes) e o conteúdo da posição.

- **Conteúdo:** A `Celula` utiliza ponteiros para `Planta` e `Ferramenta` (`Planta* planta`, `Ferramenta* ferramenta`) para implementar a regra de que uma célula pode conter 0 ou 1 planta e 0 ou 1 ferramenta.
- **Encapsulamento e Composição:** A `Celula` é responsável por libertar a memória dos objetos `Planta` e `Ferramenta` que contém (no seu destrutor e nos métodos `removerPlanta`/`removerFerramenta`), estabelecendo uma forte relação de Composição com o seu conteúdo.

3. Planeamento de Classes

3.1. Hierarquias (Herança e Polimorfismo)

A classe base abstrata Planta define os atributos internos (agua_interna, idade) e a função virtual pura atualizar. As quatro classes derivadas (Cacto, Roseira, ErvaDaninha, PlantaExotica) herdam e implementam este comportamento específico de simulação. De igual modo, a classe base Ferramenta define a interface polimórfica usar para as suas quatro derivadas (Regador, Adubo, TesouraPoda, FerramentaZ).

3.2. Relacionamentos (Agregação e Composição)

O Jardim estabelece uma relação de Composição com o objeto Jardineiro, sendo responsável pela sua criação e libertação de memória. Por sua vez, a Celula mantém ponteiros para os objetos que contém, e o Jardineiro gere as suas ferramentas num contentor. Estas relações demonstram a correta aplicação dos conceitos de agregação e composição.

4. Processamento e Validação de Comandos

O processamento de comandos é centralizado no método Comando::processar(), que utiliza uma abordagem procedural de validação.O método Comando::processar() centraliza o processamento de comandos, empregando uma abordagem procedural para a validação.

4.1. Estratégia de Validação

A funcionalidade de Comando emprega um método de tokenização via std::stringstream para processar a entrada. A verificação rigorosa do tipo de parâmetro (<n>) é efetuada através de blocos try-catch com std::stoi, garantindo a conversão e validação dos valores como inteiros, em conformidade com a sintaxe predefinida.

4.2. Utilitário de Posição

A função stringParaPosicao é responsável por converter a entrada alfanumérica do utilizador (por exemplo, "ej") em coordenadas de grelha (4, 9). Esta conversão é vital para garantir que as posições introduzidas não são apenas sintaticamente corretas, mas também se encontram dentro dos limites válidos do jardim, conforme determinado pela função Jardim::posicaoValida.

4.3. Comandos Validados

A validação é estruturada de acordo com o número de tokens:

- **Comandos de 1 token:** Incluem "sair", "avanca" e "larea". A validação verifica apenas a ausência de argumentos adicionais, garantindo um processamento rápido e seguro.
- **Comandos de 2 tokens:** Abrangem "entra aa", "colhe bb" e "lplanta cc". O segundo token é convertido para uma posição utilizando a função stringParaPosicao() e validado com Jardim::posicaoValida().
- **Comandos de 3 tokens:** Exemplos são "jardim 5 5", "planta aa cacto" e "ferramenta bb regador". Além da validação da posição, o terceiro token é comparado com uma lista de tipos permitidos.

5. Conclusão

A Meta 1 do projeto foi concluída com sucesso, estabelecendo as fundações robustas da arquitetura do simulador. As principais estruturas de dados (`Jardim` e `Celula`), as hierarquias polimórficas (`Planta`, `Ferramenta`) e a gestão de memória (construtores/destrutores) foram implementadas conforme planeado e em conformidade com as restrições do enunciado [cite: `POO - 2526 - Enunciado Trabalho Pratico.pdf`].

O sistema de validação de comandos, centralizado na classe `Comando`, está totalmente funcional e processa de forma robusta a sintaxe de todos os comandos especificados. A arquitetura está preparada para a próxima fase.

O foco da Meta 2 será a implementação da lógica de simulação, ativando os comportamentos polimórficos (como `atualizar` e `usar`) através do comando "avanca", e implementando as ações e o inventário do jardineiro.