



**Departamento de Engenharia Informática e de Sistemas**

## **Programação Orientada a Objetos**

### **Trabalho Prático**

### ***Relatório Meta 1***

Filipe Fernandes Gomes: 2022144673

João Pedro Mendes Redondo: 2022118679

02 de novembro de 2025

Ano Letivo 2025/2026 | 1º Semestre

## ÍNDICE

1.	INTRODUÇÃO.....	3
2.	ESTRUTURA E OPÇÕES TOMADAS.....	3
2.1.	Representação do Jardim.....	3
2.2.	Modelação de Plantas e Ferramentas .....	4
2.2.1.	Plantas .....	4
2.2.2.	Ferramentas .....	5
2.3.	Jardineiro .....	6
2.4.	Comandos e Validação .....	6
2.5.	Organização de Ficheiros.....	6
2.6.	Relações entre Classes e Estruturas.....	7
3.	Conclusão.....	7

## 1. INTRODUÇÃO

O presente relatório descreve o trabalho realizado na **Meta 1** do projeto “*Simulador de um Jardim*”, realizado no âmbito da unidade curricular de Programação Orientada a Objetos (POO).

O objetivo principal desta primeira meta é a definição da estrutura base do trabalho, aplicando os conceitos de encapsulamento, herança e agregação. Pretende-se também estabelecer a organização das classes e as respetivas relações entre elas.

Nesta fase inicial foram desenvolvidas as classes fundamentais para representar **plantas**, **ferramentas** e o ator principal, o **jardineiro**, bem como a estrutura de dados responsável por armazenar e apresentar visualmente o estado do **jardim**.

Foram também implementados os **comandos base** do programa, que permitem criar o jardim, ler instruções de um ficheiro, entre outros. Estes comandos são já validados quanto à sua sintaxe e parâmetros, garantindo a correta criação e inicialização do jardim.

O sistema implementado nesta meta já é capaz de criar um jardim com dimensões definidas pelo utilizador, gerar três plantas iniciais em posições aleatórias e apresentar a disposição do jardim no ecrã.

## 2. ESTRUTURA E OPÇÕES TOMADAS

Nesta primeira Meta, privilegiou-se o planeamento da arquitetura e a criação das bases estáveis do projeto, com foco na clara separação de responsabilidades.

### 2.1. Representação do Jardim

Para a representação da grelha, e em estrito cumprimento da restrição do enunciado de **não utilização de coleções da biblioteca standard** (como std::vector), optou-se por uma abordagem de gestão de memória manual.

O jardim é representado por uma grelha dinâmica através das estruturas **Retangulo** e **bloco**.

- O **Retangulo** armazena as dimensões do jardim (dimLin, dimCol) e um ponteiro (**bloco\*** solo) para um vetor alocado dinamicamente. Esta abordagem permite que o tamanho do jardim seja definido em tempo de execução.

- O **bloco** representa cada célula do solo e contém os seus atributos (água, nutrientes, temJardineiro) e um ponteiro para **Planta\***.

Esta abordagem, embora procedural na sua gestão de memória, serve como um “contentor” que armazena um ecossistema puramente orientado a objetos. O ponteiro **Planta\*** permite o uso de **polimorfismo**, onde cada bloco pode conter qualquer classe derivada de **Planta** (Cacto, Roseira, etc.).

Durante a criação do jardim, validando o funcionamento da alocação e representação, são geradas três instâncias da classe **Roseira**, posicionadas aleatoriamente na grelha.

## 2.2. Modelação de Plantas e Ferramentas

### 2.2.1. Plantas

As plantas são modeladas por uma classe base abstrata **Planta**, que agrupa o comportamento comum (atributos de **água**, **nutrientes** e **beleza**) e define funções virtuais puras como **absorve()**, **morre()** e **simbolo()**. Cada espécie deriva desta classe e redefine os comportamentos conforme as suas regras específicas.

Além das classes **Roseira**, **Cacto** e **ErvaDaninha**, foi adicionada a espécie **Dália Solar**, representada pela classe **Exotica** e que corresponde à solicitação de criação de uma **planta exótica**.

A **Dália Solar** é uma planta de beleza “bonita”, com a previsão de implementação dos seguintes comportamentos:

Inicialização:

- Inicia com **20 unidades de água** e **20 unidades de nutrientes** acumulados.

Comportamento a cada instante:

- Absorve **8 unidades de água** do solo (caso existam disponíveis).
- Absorve **2 unidades de nutrientes** do solo (caso existam disponíveis).
- Liberta **3 unidades de nutrientes** no solo, enriquecendo-o ativamente (simbolizando o seu “pólen fértil”).

Condições de morte:

- A planta **morre** se a sua reserva interna de **água chegar a 0**.
- Também morre se a quantidade de **água no solo for 0** durante **3 instantes consecutivos** (morre de sede por não conseguir absorver).
- Ao morrer, **deixa no solo 10 unidades de nutrientes** que tinha acumulado internamente (a sua água evapora-se).

Multiplicação:

- A planta **multiplica-se** para uma posição vizinha **vazia** se a sua quantidade interna de **água for superior a 40 unidades** e a de **nutrientes for superior a 30 unidades**.

- A nova planta (clone) **nasce com 10 unidades de água e 10 unidades de nutrientes**, enquanto a planta original perde essas mesmas quantidades no processo.

Representação visual:

- Nos comandos e na grelha do jardim, a **Dália Solar é representada pelo caractere 'x'**.

### 2.2.2. Ferramentas

As ferramentas seguem o mesmo princípio de herança. Existe uma classe base abstrata **Ferramenta**, responsável por definir atributos genéricos, como, o número de série, o caractere de identificação (tipo), e o método virtual **aplicaEfeito()**, que é redefinido por cada ferramenta concreta.

As subclasses **Regador**, **Adubo**, **Tesoura** e **Conversor Hídrico (FerramentaZ)** especializam este comportamento.

O **Conversor Hídrico** é a nossa ferramenta inovadora, representada visualmente pelo caractere 'z', que permite **converter nutrientes em água** dentro de uma célula do jardim. O seu funcionamento é o seguinte:

Comportamento:

- Quando aplicado numa célula de solo, **se existirem pelo menos 15 unidades de nutrientes**, a ferramenta **consome essas 15 unidades e adiciona 25 unidades de água** ao solo.

- Se o solo tiver **menos de 15 unidades de nutrientes**, a tentativa de utilização **falha e não tem qualquer efeito**.

Capacidade e funcionamento interno:

- Possui uma **capacidade inicial de 50 "cargas"**.
- Gasta **5 cargas por cada utilização bem-sucedida**.
- **Não gasta cargas** se a utilização falhar (nutrientes insuficientes).
- Quando as cargas chegam a **0**, o conversor torna-se **inutilizável** e é automaticamente **descartado**.

### 2.3. Jardineiro

Em linha com o planeamento da Meta 1, a classe **Jardineiro** foi definida, embora a sua implementação esteja reservada para a Meta 2. O **Jardineiro** será a classe central de interação do utilizador com o jardim e as suas responsabilidades planeadas são:

- Posicionamento: Manter a sua posição atual (linha, coluna) ou o estado de “fora do jardim”.

- Inventário: Gerir um inventário de ferramentas (provavelmente um `std::vector<Ferramenta*>`), permitindo ao utilizador apanhar, largar e comprar ferramentas.

- Ferramenta Ativa: Manter um ponteiro para a **Ferramenta** que está “na mão” e que será ativada a cada instante.

- Gestão de Ações: Controlar os limites de ações por turno (movimentos, plantações, colheitas) lidos a partir da classe **Settings**.

### 2.4. Comandos e Validação

Todos os comandos do programa encontram-se centralizados na **namespace cmd**, definida como `constexpr std::string_view`, o que permite evitar o uso de “strings mágicas” e otimizar comparações.

O ficheiro **Comandos.cpp** trata da leitura e validação de todas as instruções. A função **Executa\_Commandos** recebe o **istream** e uma referência para o **Retangulo**, permitindo-lhe processar comandos quer do **std::cin** quer de ficheiros.

Funções auxiliares como **strParaInt()**, **strParaCoords()** e **verificaLixo()** garantem a validação rigorosa dos dados. Esta validação é aplicada a **todos** os comandos do enunciado (incluindo a verificação de parâmetros extra), mesmo aqueles cuja funcionalidade só será implementada na Meta 2, cumprindo integralmente os requisitos.

### 2.5. Organização de Ficheiros

Um dos objetivos primários da Meta 1 era a correta organização do projeto. A estrutura de pastas adotada separa o código por responsabilidades (domínio), facilitando a manutenção e o desenvolvimento:

```
[Nome do Projeto]/  
|---CMakeLists.txt  
|---main.cpp      (Ponto de entrada, contém o loop principal)  
|---Settings.h    (Ficheiro de constantes fornecido)  
|--- Comandos/  
|   |---Comandos.h (Definição do namespace 'cmd' e protótipos)  
|   |---Comandos.cpp (Implementação da validação e execução)  
|--- Jardim/  
|   |---Jardim.h    (Definição das structs 'Retangulo' e 'bloco')  
|   |---Jardim.cpp   (Funções de gestão da grelha: CriaJardim, etc.)  
|--- Jardineiro/  
|---Jardineiro.h   (Definição da classe Jardineiro – Planeamento)  
|--- Plantas/  
|   |---Planta.h     (Classe base abstrata)  
|   |---Cacto.h, Roseira.h, ErvaDaninha.h, Exotica.h (Classes derivadas)  
|   | ...           (Ficheiros .cpp correspondentes)  
|--- Ferramentas/  
|   |---Ferramenta.h (Classe base abstrata)  
|   |---Regador.h, Adubo.h, Tesoura.h, FerramentaZ.h (Classes derivadas)  
|   | ...           (Ficheiros .cpp correspondentes)
```

## 2.6. Relações entre Classes e Estruturas

O **jardim (Retângulo)** mantém uma relação de **composição** com os **blocos** que o constituem – quando o jardim (**Retangulo**) é destruído (através do **delete[ ] x.solo**), os **blocos** também são libertados.

Cada bloco contém um ponteiro para uma planta, o que configura uma relação de **Agregação**, pois as plantas (objetos) existem no *heap* e o seu ciclo de vida não está diretamente preso ao **bloco** que as aponta. De forma similar, a classe **Jardineiro** (a ser implementada) terá uma relação de **Agregação** com **Ferramenta**, armazenando ponteiros para as ferramentas no seu inventário.

## 3. Conclusão

Os objetivos da Meta 1 foram cumpridos e a arquitetura do projeto está bem definida. O foco seguinte será a implementação da lógica da simulação a cada instante.