SIMULADOR DE HABITAÇÃO INTELIGENTE

Programação Orientada a Objetos

Ano Letivo 2023/2024

Pedro Paiva - 2021134625
Tomás Laranjeira - 2021135060
Licenciatura de Engenharia Informática

ÍNDICE

CONTENTS

ÍNDICE DE FIGURAS

INTRODUÇÃO

Este relatório aborda o desenvolvimento de um simulador em C++ para uma habitação controlada por domótica. O simulador engloba várias zonas da habitação, cada uma com propriedades como temperatura e luz, monitorizadas por sensores. Estes sensores fornecem dados a processadores de regras que, por sua vez, determinam ações com base nas leituras obtidas. Além disso, as zonas possuem aparelhos que afetam as suas propriedades e respondem a comandos dos processadores de regras.

Apesar da aparente simplicidade dos comandos, o simulador é, essencialmente, um conjunto de objetos interligados com uma interface de utilizador. Destaca-se, igualmente, a forma como o simulador aborda a passagem do tempo, utilizando unidades abstratas denominadas "instantes", que avançam conforme os comandos emitidos pelo utilizador.

Este relatório fornecerá uma visão abrangente da arquitetura, implementação e funcionamento deste simulador domótico em C++.

ORGANIZAÇÃO DO CÓDIGO

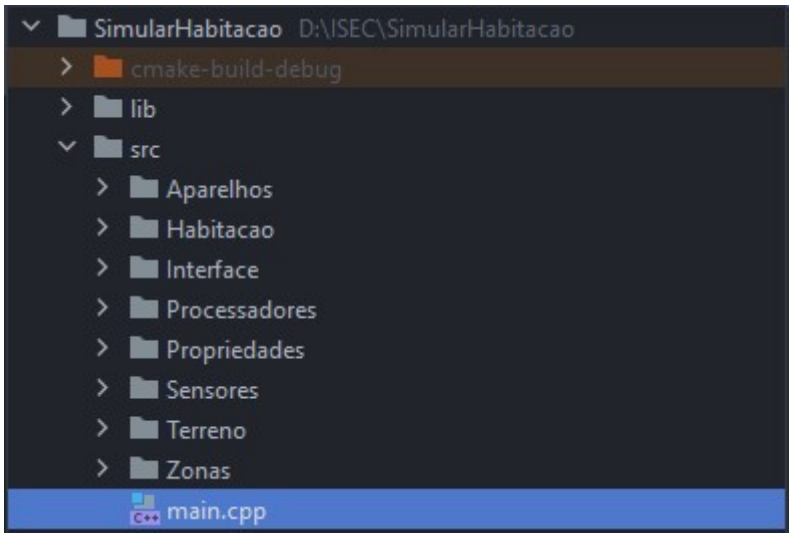

Figura 1 - Organização do projeto

\lib: Neste diretoria, agrupámos todos os ficheiros relacionados à biblioteca fornecida pelo professor.

\src: Esta é a principal pasta do projeto, onde segmentámos o código em subdiretorias específicas.

- **main.cpp**: Funciona como o ponto de entrada central do projeto.
- **\Terreno**: Dedicada à implementação da classe "Terreno".
- **\Habitacao**: Contém os ficheiros relacionados com a classe "Habitacao".
- **\Interface**: Reservada para os ficheiros da classe "Interface".
- **\Zonas**: Estrutura os elementos da classe "Zona".
- **\Aparelhos**: Contém os ficheiros da classe principal "Aparelho" e as suas subclasses (Aquecedor,Refrigerador, Lampada, ...).
- **\Processadores**: Designada para os ficheiros das classes principais "Processador" e "Regra" e as suas subclasses (EntreRegra, ForaRegra , ...).
- **\Propriedades**: Agrupa os ficheiros relacionados à classe principal "Propriedade" e as suas subclasses Temperatura, Som, ...

- **\Sensores**: Reservada para a classe "Sensor" e todas as suas subclasses, tais como Temperatura, Movimento, Som, Fumo, entre outras.

Esta organização visa otimizar a gestão do código, facilitando a colaboração entre membros da equipa e contribuindo para a manutenção e expansão eficientes do projeto.

CLASSE TERRENO

Figura 2- Classe Terreno

A classe "Terreno" representa uma parcela de terreno no contexto do projeto. Nela, é armazenado um ponteiro para um objeto da classe Habitacao. A sua função principal é permitir a criação de uma habitação neste "terreno", indicando o número de linhas e colunas desejado. Em resumo, esta classe atua como uma entidade que encapsula e gere a habitação associada, facilitando a manipulação e interação com essa habitação no âmbito do sistema.

CLASSE HABITACAO

```
class Habitacao {
private:
    vector<Zona*> zonas;
    int habLinhas;
    int habColunas;

    int instancia = 1;

    map<string, ProcessadorRegras*> procGuardados;
```

Figura 3- Classe habitcao

A classe Habitacao é fundamental no projeto, sendo responsável por gerir uma habitação, que por sua vez é composta por diferentes zonas. Aqui estão os principais pontos:

Zonas: A classe mantém um vetor de zonas, cada uma representa uma área específica na habitação. Essas zonas podem ser manipuladas e geridas pela instância da "Habitacao".

Processadores de Regras Guardados: A classe mantém um mapa associando nomes a processadores de regras. Este mapa permite guardar e recuperar processadores de regras específicos associados à habitação.

Operações com Zonas: Fornece métodos para adicionar e remover zonas, bem como obter informações sobre as zonas existentes na habitação.

Resumidamente, a classe "Habitacao" atua como uma entidade central para gerir a configuração, as zonas e os processadores de regras associados.

CLASSE INTERFACE

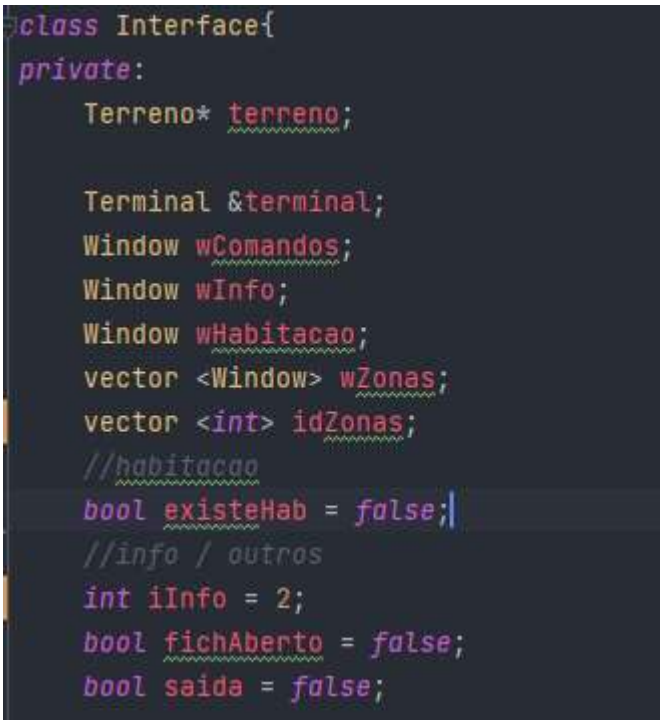

Figura 4- Classe Interface

A classe "Interface"trata da interação do utilizador com o sistema. Alguns pontos principais:

- Terreno: Mantém uma ligação com a classe "Terreno", permitindo interação com a habitação e consequentemente com as zonas.
- Janelas (Windows): Servem para exibir informações, comandos, detalhes da habitação e zonas.
- Processamento de Comandos: Oferece métodos para processar e executar comandos fornecidos pelo utilizador.
- Construção e Manipulação da Habitação e Zonas: Inclui métodos para construir a habitação, avançar no tempo, criar zonas, limpar zonas e realizar outras operações relacionadas à habitação.
- Verificações: Realiza várias verificações para garantir o melhor funcionamento possível.

Esta classe atua como uma interface entre o utilizador e o sistema, permitindo-lhes interagir e controlar as diferentes funcionalidades do projeto.

CLASSE ZONA

```
class Zona {
private:
    int idZona;
    static int nextIdZona;
    int linhaZona;
    int colunaZona;
    int posZona;

    vector<Sensor*> sensores;
    vector<ProcessadorRegras*> processadores;
    vector<Aparelho*> aparelhos;
    map<Propriedade*,int> propriedades;
```

Figura 5- Classe Zona

A classe "Zona" é responsável por representar uma área específica dentro do contexto do projeto. Cada instância desta classe corresponde a uma zona na habitação.

- Componentes: Mantém listas de sensores, aparelhos e processadores de regras, fundamentais para a monitorização e controlo da zona.
- Propriedades: Utiliza um map para armazenar e gerir as propriedades da zona, como temperatura , humidade , fumo , entre outras.
- Adição e Remoção de Componentes: Fornece métodos para adicionar e remover sensores, aparelhos e processadores de regras.
- Consulta de Componentes e Propriedades: Permite a obtenção de várias informações sobre os componentes e propriedades associadas à zona.

A "Zona" desempenha um papel central na modelagem da habitação, encapsulando e organizando os componentes que a compõem.

CLASSE PROPRIEDADE

```
class Propriedade {
private:
    string nome;
    string unidade;
    int min;
    int max;
```

Figura 7- classe Propriedade

```
class Humidade: public Propriedade {
public:
    Humidade();
    ~Humidade();
};
```

Figura 6- Classe Humidade

A classe "Propriedade" é responsável por gerir as informações associadas a uma propriedade do ambiente. Cada propriedade tem um nome descritivo (por exemplo, "temperatura", "luz"), uma unidade de medida correspondente (por exemplo, "graus Celsius", "lumens"), e valores mínimo e máximo admissíveis.

Utilizámos o conceito de herança, onde classes específicas, como "TemperaturaSensor", "LuminosidadeSensor" e outras, podem herdar características e comportamentos básicos da classe "Propriedade". Isto permite a fácil incorporação de novos tipos de sensores ou propriedades, mantendo uma estrutura coesa e flexível.

CLASSE SENSOR

```
class Sensor {
private:
    string idSensor;
    static int nextIdSensor;
    string tipo;
    int valor;
```

Figura 8- Classe Sensor

A classe "Sensor" é responsável pela criação de sensores. Estes são componentes que têm como objetivo averiguar o valor de uma propriedade em específico, na zona em que o sensor se encontra. Esse valor será então transmitido a elementos da classe "Regra". Para a formulação desta classe optámos por utilizar o método da herança, criando assim uma class Sensor de cada propriedade (por exemplo, TemperaturaSensor).

CLASSE APARELHO

```
class Aparelho {
private:
    string idAparelho;
    static int nextIdAparelho;
    string tipo;
    string comando;
    bool ligado;
    Zona* zona;
    int counterLigado = 0;
```

Figura 9- Classe Aparelho

A classe "Aparelho" é encarregada da criação de aparelhos. Aparelhos são componentes que se irão ligar e desligar consoante os comandos enviados pelos processadores de regras. Estes aparelhos podem causar vários efeitos que irão alterar os valores das propriedades. Na implementação desta classe optámos pelo uso de herança, pois achámos que seria o método mais vantajoso nesta situação.

CLASSE PROCESSADORREGRAS

Figura 10- Classe ProcessadorRegras

A classe "ProcessadorRegras" representa um dispositivo do sistema responsável por gerir regras específicas relacionadas aos sensores e, consequentemente, aos aparelhos associados.

- Comando: O processador tem um comando de saída configurável.
- Associação a Zona: O processador está associado a uma zona específica na habitação.
- Regras e Sensores: Gere um conjunto de regras, cada uma vinculada a sensores específicos. A ativação do comando do processador depende do cumprimento de todas as regras.
- Associação a Aparelhos: Pode comunicar o comando configurado a um ou mais aparelhos conectados à sua saída.

CLASSE REGRA

```
class Regra {
private:
    string idRegra;
    static int nextIdRegra;
    string nome;
    Sensor* sensorAssoc;
```

Figura 11- Classe Regra

Por fim temos a classe "Regra", está é responsável por receber um valor de um sensor e verificar se esse valor se verifica com a regra em questão (por exemplo valor igual a 3). De seguida a regra avisa um processador de regras se ela se confirma ou não. Para a concretização desta classe optámos novamente pela utilização de herança.

DIFICULDADES ENCONTRADAS

Inicialmente sentimos um nivel de dificuldade elevado na compreensão e utilização da biblioteca fornecida pelo professor, eventualmente conseguimos alcançar o nivel de compreensão necessário para a realização do trabalho, no entanto considerámos que o tempo usado para compreender como utilizar a biblioteca não foi proporcional aos resultados obtidos.

Durante o nosso projeto, enfrentamos um desafio intrigante: inclusões circulares no código. Uma experiência inesperada que nos consumiu tempo até identificar e corrigir o problema. Essa descoberta foi uma valiosa aprendizagem, destacando a importância da vigilância constante na estrutura do projeto e na interação das dependências entre diferentes partes

Enfrentamos desafios consideráveis ao testar o programa devido à complexidade das centenas de combinações possíveis com os vários comandos disponíveis. Muitas vezes, a visualização do que estava

realmente a acontecer tornou-se um obstáculo, exigindo esforços adicionais na depuração e compreensão do comportamento do programa.