



Instituto Superior de Engenharia de Coimbra

Engenharia Informática

Programação Orientada a Objetos Trabalho Prático 2021/22

Relatório Final

Hugo Gabriel Carvalho Ferreira, 2020128305

Rafaela Oliveira Carvalho, 2019127935

Índice

Introdução	3
Interface	3
Classes	3
Classe Jogo	3
.....	4
Classe Ilha	4
Classe Zona	4
Classe Trabalhador	5
Classe Edifício	6
Classe SaveLoad	6
Conclusão	7

Introdução

O trabalho prático da disciplina de Programação Orientada a Objetos pretende que se construa em C++ um jogo do tipo single-player de construção e desenvolvimento.

Para a realização do mesmo, foram interpretados diferentes conceitos e entidades que foram traduzidas em classes no projeto.

Também será justificado a organização do programa e o progresso do mesmo ao longo deste relatório.

Interface

A interface é constituída por 9 funções que são responsáveis por apresentar o jogo ao utilizador:

```
class Interface{
public:
    static void comeceInterface();
    static void menuJogo();
    static void mostraComandos();
    static void mostraIlha(Jogo &jogo);
    static void list(Jogo &jogo, int l, int c);
    static bool verificaLinhaColuna(Jogo &jogo, int l, int c);
    static void validaComando(Jogo &jogo, SaveLoad &jogos, istringstream &recebe);
    static void mostraRecursos(Jogo &jogo);
    static bool leFicheiro(Jogo &jogo, SaveLoad &jogos, string ficheiro);
    static bool config(Jogo &jogo, string ficheiro);
};
```

Classes

Classe Jogo

A classe Jogo é a classe principal. Esta classe permite consultar tudo relativo ao jogo como, total de trabalhadores, quantidade de dinheiro existente, quantidade de recursos, o tipo de edifício em cada zona, entre outros.

Sendo que esta é a classe responsável pela lógica do jogo ela permite construir edifícios, contratar trabalhadores, vender recursos e vender edifícios.

A classe Jogo tem apenas uma colaboração, um ponteiro da classe Ilha com o nome i.

[illegible]

Classe Ilha

Através da classe Ilha é nos permitido consultar todas as informações relacionadas com as zonas da ilha.

Nesta classe podemos procurar uma zona específica, um zona do tipo pasto e permite nos mover trabalhadores.

A classe Ilha contém as zonas da ilha.

```

class Itha {
private:
    Zona **Itha;

    int ITHMAS;
    int COLUMNAS;

    int totalTrabalhadores;

    int precoMNF;
    int precoMNC;
    int precoCEN;
    int precoBAT;
    int precoFUN;
    int precoSER;

    int precoOPER;

    int precoLIN;
    int precoLINER;

public:
    Zona**getItha() const{return Itha;}

    int getPrecoOPER() const{return precoOPER;}
    int getPrecoLEN() const{return precoLEN;}
    int getPrecoLINER() const{return precoLINER;}
    int getIthmas() const{return ITHMAS;}
    int getColunas() const{return COLUMNAS;}
    int getTotalTrabalhadores() const{return totalTrabalhadores;}

    int getPrecoMNF() const{return precoMNF;}
    int getPrecoMNC() const{return precoMNC;}
    int getPrecoCEN() const{return precoCEN;}
    int getPrecoBAT() const{return precoBAT;}
    int getPrecoFUN() const{return precoFUN;}
    int getPrecoSER() const{return precoSER;}

    void setPreco(Istringstream &recebe);

    void setPrecoTraba(Istringstream &recebe);

    Itha(int l=0, int c=0);

    Itha(const Itha& aux);

    ~Itha(){int i=Ithas;}

    int micoracao(const string ed, int l, int c);

    bool presencaPasta(string traba, int d);

    int moveTrabalhador(string id, int l, int c);

    void retirarTotalTrabalhadores(int conta){totalTrabalhadores = totalTrabalhadores - conta;}

    void decrementaTotalTrabalhadores(){totalTrabalhadores--;}

    void salvarArquivo();

    void imprimirArquivo();
}

```

Classe Zona

A classe Zona aloca o tipo de zona, o tipo de edifícios e o número total de trabalhadores.

Esta classe consegue encontrar mineiros e operários, adicionar e remover trabalhadores, libertar edifícios, entre outras funções.

A classe contém um ponteiro tipoZona, Edifício e um vetor de trabalhadores ponteiro que servem para realizar as heranças e o polimorfismo.

```
class Zona{
private:
    tipoZona *tp;
    Edificio *e;
    vector<Trabalhador*> trabalhadores;
    int totalTrabalhadores;
    static int p, d, pa, f, m, z;
public:
    tipoZona* getTipoZona() const{return tp;}
    Edificio* getEdificio() const{return e;}
    int getTotalTrabalhadores() const{return totalTrabalhadores;}
    Zona();
    Zona(const Zona& aux);
    Zona& operator=(const Zona& aux)
    {
        if(this == &aux) return *this;
        delete tp;
        tp = nullptr;
        delete e;
        e = nullptr;




        auto it = trabalhadores.begin();
        while(it != trabalhadores.end()){
            delete *it;
            ++it;
        }
        trabalhadores.clear();

        tp = aux.tp->duplica();
        e = aux.e->duplica();
        for(int i=0, i<aux.trabalhadores.size(); i++)
            trabalhadores[i] = aux.trabalhadores[i]->duplica();
        totalTrabalhadores = aux.totalTrabalhadores;
        return *this;
    }

    ~Zona(){
        for(int i=0; i<trabalhadores.size(); i++) delete trabalhadores[i]; delete e;
    }
    bool alocaTrabalhador(string traba, string id);
    bool alocaEdificio(string ed);
    bool verificaEdificio();
    void listaTrabalhadores();
    bool encontraMineiro();
    bool encontraOperario();
    Trabalhador* encontraTrabalhador(string id);
    int contaLenhadores();
    int contaTrabalhadores();
    void removeTrabalhador(Trabalhador *t);
    bool adicionaTrabalhador(Trabalhador* t);
    void aumentaTotalTrabalhadores ()(totalTrabalhadores++);
    void diminuiTotalTrabalhadores()(totalTrabalhadores--);
    void subtraiTotalTrabalhadores(int conta)(totalTrabalhadores = totalTrabalhadores - conta);
    bool despedeTrabalhador();
    void libertaEdificio();
    void limpaTrabalhadores();
    void podeMover();
    Zona* duplicaZona(){return new Zona(*this);}
};
```

Classe Trabalhador

A classe Trabalhador tem como herança os tipos de trabalhadores existentes, sendo estes:

-  Operário;
-  Lenhador;
-  Mineiro.

```
class Trabalhador{
    string tipo;
    string id;
    int precoContrat;
    int diasContrato;
    int controle;
    int estado;
protected:
    int probabilidade;
public:
    string getTipo() const{return tipo;}
    string getId() const{return id;}
    int getPrecoContrat() const{return precoContrat;}
    int getDiasContrato() const{return diasContrato;}
    int getControle() const{return controle;}
    Trabalhador(string t, string i, int p, int pre, int c=0, int dc=0): tipo(t), precoContrat(p), probabilidade(pre), controle(c), diasContrato(dc), estado(0){}
    virtual ~Trabalhador(){}
    void naoPodeMover(){controle = 1;}
    void podeMover(){controle = 0;}
    virtual Trabalhador* duplica() const=0;
    void incrementaDias();
    void aumentaProbabilidade();
    bool seFaltou();
};







class Mineiro : public Trabalhador{
public:
    Mineiro(string tipo, string id, int p, int pre): Trabalhador(t, id, p, pre){}
    Trabalhador* duplica() const override { return new Mineiro(*this); }
};

class Operario : public Trabalhador{
public:
    Operario(string tipo, string id, int p, int pre): Trabalhador(t, id, p, pre){}
    Trabalhador* duplica() const override { return new Operario(*this); }
};

class Lenhador : public Trabalhador{
public:
    Lenhador(string tipo, string id, int p, int pre): Trabalhador(t, id, p, pre){}
    Trabalhador* duplica() const override { return new Lenhador(*this); }
};
```

Classe Edifício

A classe Edifício tem como heranças os tipos de edifícios existentes no jogo, sendo estes:

-  Mina de Ferro
-  Mina de Carvão
-  Central elétrica
-  Bateria
-  Fundição
-  Serraria

Também é nesta classe que controlamos o nível em que cada tipo de edifício se encontra, se este se encontra ligado ou desligado e a quantidade de armazenamento de cada um.

```
class Edificio{
    string tipo;
    string estado;
protected:
    int nivel;
    int produtividade;
    int armazenamento;
    int probabilidade;
public:
    string getTipo() const{return tipo;}
    string getEstado() const{return estado;}
    int getProdutividade() const {return produtividade;}
    int getNivel() const{return nivel;}
    int getArmazenamento() const{return armazenamento;}
    Edificio(string t, int pnb, int proba, string e="desligado", int nci, int arma): tipo(t), produtividade(p), probabilidade(pro), estado(e), nivel(nci), armazenamento(arma){}
    virtual ~Edificio()=default;
    bool ligarEdificio();
    bool desligarEdificio();
    virtual bool aumentavel()=0;
    virtual void diminuiProdutividade()=0;
    virtual Edificio* duplica() const=0;
    void duplicaProdutividade(){produtividade = produtividade*2;}
    void aumentaArmazenamento();
    void diminuiArmazenamento();
    bool desabar();
};
```

```
class MinaFerro : public Edificio {
public:
    MinaFerro(string tipo) : Edificio("Mina de Ferro", 2, 100, 10){}
    bool aumentavel() override;
    void diminuiProdutividade() override;
    Edificio* duplica() const override { return new MinaFerro(*this); }
};

class MinaCarvão : public Edificio{
public:
    MinaCarvão(string tipo) : Edificio("Mina de Carvão", 2, 100, 10){}
    bool aumentavel() override;
    void diminuiProdutividade() override;
    Edificio* duplica() const override{return new MinaCarvão(*this);}
};

class CentralElétrica : public Edificio{
public:
    CentralElétrica(string tipo) : Edificio("Central Elétrica", 1, 100, 10){}
    bool aumentavel() override;
    void diminuiProdutividade() override();
    Edificio* duplica() const override{return new CentralElétrica(*this);}
};

class Bateria : public Edificio{
public:
    Bateria(string tipo) : Edificio("Bateria", 1, 100, 10){}
    bool aumentavel() override;
    void diminuiProdutividade() override();
    Edificio* duplica() const override{return new Bateria(*this);}
};

class Fundição : public Edificio{
public:
    Fundição(string tipo) : Edificio("Fundição", 1, 100, 10){}
    bool aumentavel() override;
    void diminuiProdutividade() override();
    Edificio* duplica() const override{return new Fundição(*this);}
};

class Serraria : public Edificio {
public:
    Serraria(string tipo) : Edificio("Serraria", 2, 100, 10){}
    bool aumentavel() override;
    void diminuiProdutividade() override();
    Edificio* duplica() const override { return new Serraria(*this); }
};
```

Classe SaveLoad

A classe SaveLoad guarda o snapshot do jogo atual num vetor de jogos. Esta classe cria cópia do jogo e guarda em memória que neste caso é um vetor de jogos.

```
class SaveLoad{
    vector<Jogo*> jogosGuardados;
public:
    void saveJogo(Jogo* aux);
    bool removeJogo(string nome);
    Jogo* encontraJogo(string nome);
};
```

Conclusão

Pensamos que para o projeto entregue conseguimos aplicar o que foi lecionado em aula, tendo em consideração os princípios da programação orientada a objetos, a nível de Polimorfismo, Herança e Encapsulamento e cumpre os requisitos de C++.

Na execução deparamo-nos com um problema. Apesar de conseguirmos guardar o jogo, a cópia da ilha falha, portanto, deste modo decidimos não deixar o load no trabalho. No entanto o save e o apagar estão a funcionar corretamente.

Para concluir pensamos que o trabalho se encontra bem-sucedido, e que cumpre os requisitos no enunciado do trabalho.