RELATÓRIO META Nº2

TRABALHO PRÁTICO C++

Licenciatura em Engenharia Informática Pós-Laboral
Programação Orientada a Objetos
2021-2022

Trabalho realizado por:
João Rui Pires Morais - 2019134282
Sandra Maria Gonçalves Perdigão - 2019102697

Índice

1. Introdução

O trabalho prático da disciplina de Programação Orientada a Objetos tem por objetivo construir em C++ um jogo, do tipo single-player, sobre construção e desenvolvimento de uma ilha. A concessão da ilha é atribuída ao jogador, que a deve desenvolver e industrializar, construindo assim todo um complexo fabril.

2. Classes

Nesta secção iremos explicar as classes que foram usadas na segunda meta do projeto.

2.1 Classe Interação

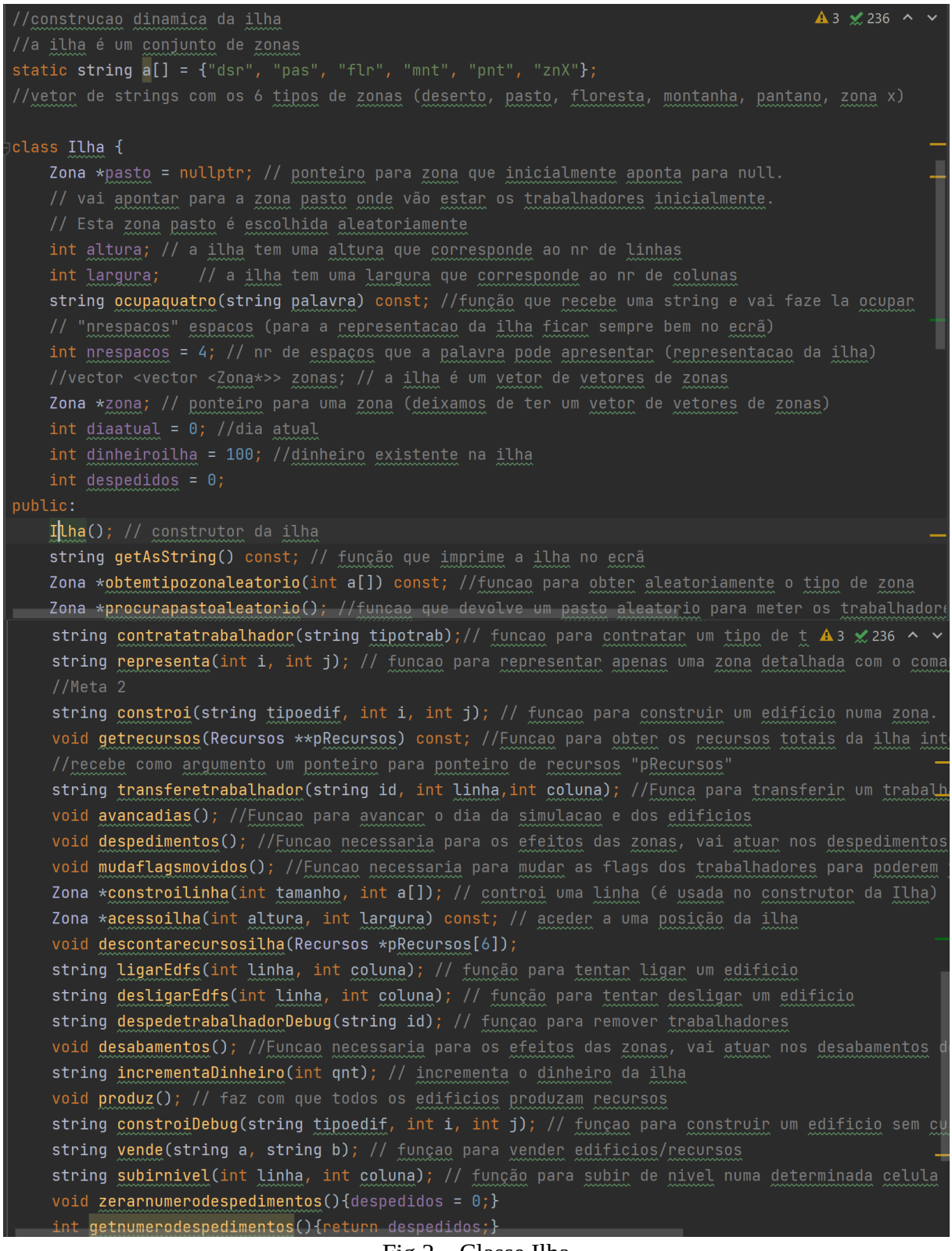
Como principais funções desta classe encontram-se a interpretação e validação de todos os comandos, seja por teclado, ou por leitura do ficheiro de configuração.
É também responsável por chamar as funções gerais, seja criar a ilha, chamar as funções para provocarem os efeitos nas zonas (incrementar dias no jogo, desabamento de edifícios, despedimento de trabalhadores, etc.), funções de recolha de recursos (função de produção de recursos), entre outras. Esta classe tem um objetivo bastante focado, de gerir o jogo e comunicar com as diferentes partes da aplicação para se desenvolver e progredir o jogo, gerindo a comunicação com o utilizador.

```
class Interacao {
    Ilha ilha= Ilha(); //construtor da ilha
    string trim(string str); //funcao para remover os espaços em branco
    bool terminar = false;
public:
    Ilha getilha(){return ilha;} //obter a ilha
    void lecomandos(istream& in); //ler comandos
    void executaficheiro(string nomeficheiro); //ler ficheiro txt
    bool getflag(){return terminar;} //para o main saber quando sair
    void efeitoszonas();
    void recolharecursos();
};
```

Fig.1 – Classe Interação

2.2 Classe Ilha

Enquanto que na Meta 1 a ilha era um vetor de vetores de Zonas, na Meta 2 a ilha é construída por lista ligada de Zonas, em que cada zona tem um ponteiro para a primeira zona da próxima linha e um ponteiro para a próxima zona. Assim, temos uma ilha com zonas ao longo da largura e da altura da ilha, sendo estes parâmetros fornecidos pelo jogador no início do jogo.
Como principais funções desta classe temos a construção da ilha e a representação visual da ilha, com os tipos de zona gerados de forma aleatória. Outras responsabilidades da ilha são construir edifício numa determinada zona, contratar trabalhador que vai para uma zona de pasto escolhida de forma aleatória, incrementar dinheiro da ilha, obter os recursos totais da ilha, etc.



```
//construcao dinamica da ilha                                    ⚠ 3  ✓ 236
//a ilha é um conjunto de zonas
static string a[] = {"dsr", "pas", "flr", "mnt", "pnt", "znX"};
//vetor de strings com os 6 tipos de zonas (deserto, pasto, floresta, montanha, pantano, zona x)

class Ilha {
    Zona *pasto = nullptr; // ponteiro para zona que inicialmente aponta para null.
    // vai apontar para a zona pasto onde vão estar os trabalhadores inicialmente.
    // Esta zona pasto é escolhida aleatoriamente
    int altura; // a ilha tem uma altura que corresponde ao nr de linhas
    int largura;    // a ilha tem uma largura que corresponde ao nr de colunas
    string ocupaquatro(string palavra) const; //função que recebe uma string e vai faze la ocupar
    // "nrespacos" espacos (para a representacao da ilha ficar sempre bem no ecrã)
    int nrespacos = 4; // nr de espaços que a palavra pode apresentar (representacao da ilha)
    //vector <vector <Zona*>> zonas; // a ilha é um vetor de vetores de zonas
    Zona *zona; // ponteiro para uma zona (deixamos de ter um vetor de vetores de zonas)
    int diaatual = 0; //dia atual
    int dinheiroilha = 100; //dinheiro existente na ilha
    int despedidos = 0;
public:
    Ilha(); // construtor da ilha
    string getAsString() const; // função que imprime a ilha no ecrã
    Zona *obtemtipozonaleatorio(int a[]) const; //funcao para obter aleatoriamente o tipo de zona
    Zona *procurapastoaleatorio(); //funcao que devolve um pasto aleatorio para meter os trabalhador
    string contratatrabalhador(string tipotrab);// funcao para contratar um tipo de t ⚠ 3  ✓ 236
    string representa(int i, int j); // funcao para representar apenas uma zona detalhada com o coma
    //Meta 2
    string constroi(string tipoedif, int i, int j); // funcao para construir um edifício numa zona.
    void getrecursos(Recursos **pRecursos) const; //Funcao para obter os recursos totais da ilha int
    //recebe como argumento um ponteiro para ponteiro de recursos "pRecursos"
    string transferetrabalhador(string id, int linha,int coluna); //Funca para transferir um trabalh
    void avancadias(); //Funcao para avancar o dia da simulacao e dos edificios
    void despedimentos(); //Funcao necessaria para os efeitos das zonas, vai atuar nos despedimentos
    void mudaflagsmovidos(); //Funcao necessaria para mudar as flags dos trabalhadores para poderem
    Zona *constroilinha(int tamanho, int a[]); // controi uma linha (é usada no construtor da Ilha)
    Zona *acessoilha(int altura, int largura) const; // aceder a uma posição da ilha
    void descontarecursosilha(Recursos *pRecursos[6]);
    string ligarEdfs(int linha, int coluna); // função para tentar ligar um edificio
    string desligarEdfs(int linha, int coluna); // função para tentar desligar um edificio
    string despedetrabalhadorDebug(string id); // funçao para remover trabalhadores
    void desabamentos(); //Funcao necessaria para os efeitos das zonas, vai atuar nos desabamentos d
    string incrementaDinheiro(int qnt); // incrementa o dinheiro da ilha
    void produz(); // faz com que todos os edificios produzam recursos
    string constroiDebug(string tipoedif, int i, int j); // funcao para construir um edificio sem cu
    string vende(string a, string b); // funçao para vender edificios/recursos
    string subirnivel(int linha, int coluna); // função para subir de nivel numa determinada celula
    void zerarnumerodespedimentos(){despedidos = 0;}
    int getnumerodespedimentos(){return despedidos;}
```

Fig.2 – Classe Ilha

2.3 Classe Zona

Como principais funcionalidades desta classe temos as funções para obter o tipo de trabalhadores da zona e o número, obter o tipo de edifício da zona, representar os detalhes da zona, entre outros.
Todas as classes derivadas da Zona (Deserto, Floresta, Montanha, Pântano, Pasto, ZonaX) têm a sua implementação das funções virtuais.

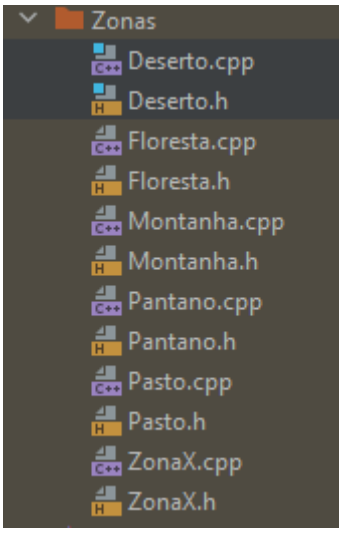


Fig.3 – Classes derivadas de Zona

```
class Zona {
    Edificio *edificio = nullptr; //a zona contem ponteiro do tipo Edificio chamado "edificio",
    // que aponta para null caso a zona nao tenha edificio
    Zona *proximazona = nullptr; // ponteiro para a próxima zona
    Zona *primeiraproxzona = nullptr; // ponteiro para a primeira posição da próxima linha
protected:
//Meta 2
    vector<Trabalhadores *> trabalhadores;
//a zona contem um vetor de ponteiros do tipo trabalhadores.
//a zona contem recursos
//a classe zona tem um ponteiro "tiposrecursos" que aponta para um vetor de recursos
//cada uma das ó celulas do vetor aponta para um tipo de recurso (ferro, aço, carvão, madeira, vigas de madeira, eletricidade)
    Recursos *tiposrecursos[6] = {new Ferro(), new Aco(), new Carvao(), new Madeira(), new VigasMadeira(),
                                  new Eletricidade()};
public:
    Zona(); //construtor da zona
    static int count; //variavel estatica para contar as zonas. Nao esta a ser usada
    string teste; // para imprimir a ilha no ecra. Nao esta a ser usada
    //get
    string gettipoedificio() const; //obter o tipo de edificio existente na zona
    //funcoes
    string contratatrabalhador(string tipo, int dia); // funcao para contratar trabalhador.
    //tem como argumentos o tipo de trabalhador(operario, mineiro,lenhador)
    //e o dia do jogo,  que sera necessario para o id do trabalhador
    int verificacondicoescontratacao(string tipo, int dia, int dinheiroilha);
    int obtemnumerotrabalhadores() const { return trabalhadores.size(); } //funcao para obter o numero de trabalhadores. corres
    string getasstring(); //funcao para representar o detalhe de cada zona
    string obtemtipotrabalhadores() const; //funcao para obter a inicial do tipo de trabalhadores
    string constroiedificio(string tipoedif); //funcao para construir edificio na zona
    Trabalhadores *retornatrabalhador(string id); //funcao para devolver um ponteiro para o trabalhador com determinado ID
    virtual string gettipozona() const = 0; // esta funcao existe em todos os "filhos" de zona: deserto, pastagem, etc. Devolve
    void getrecursoszonaapenas(Recursos **a) const; // devolve os recursos apenas da zona
    void getrecursos(Recursos **a) const; //esta funcao obtem os recursos existentes numa zona
    //os recursos de uma zona são o resultado da soma dos recursos existentes nessa zona
    //com os recursos do edificio da zona, se aplicavel
    //esta funcao recebe um ponteiro de ponteiros de recursos,
    // semelhante ao ponteiro *tiposrecursos, que provém do edificio
    string getrecursosss() const; // esta funcao vai chamar a funcao acima
//vai permitir obter a quantidade total de recursos da zona e passa-los para uma string
string gettrabalhadores() const; //funcao para representar os trabalhadores de uma zona
void adicionatrabalhador(Trabalhadores *pTrabalhadores); //funcao para adicionar um trabalhador a uma zona (acrescenta ao vetor de trabalhadores dessa zona)
void apagatrabalhador(Trabalhadores *pTrabalhadores); //funcao para apagar um trabalhador de uma zona (apaga do vetor de trabalhadores da zona)
int despedimentos(int diaatual); //funcao para despedir trabalhadores, baseando se na probabilidade. Precisa do dia atual
void mudaflags(); // as flags dos trabalhadores passam todas a FALSE
Zona *getproximazona() { return proximazona; } // devolve a proxima zona
Zona *getprimeriaproxzona() { return primeiraproxzona; } // devolve a primeira proxima zona
void setproximazona(Zona *prox) { this->proximazona = prox; } // altera a proxima zona
void setprimeiraproxzona(Zona *prox) { this->primeiraproxzona = prox; } // altera a primeira proxima zona
void descontarecursoszona(Recursos **pRecursos);
Edificio *getponteiroedificio() { return edificio; } //devolve o ponteiro de edificio da zona
int desabamentos(string tipozona); //funcao para desabar edificios, baseando se na probabilidade
string getEstadoEdificio(); // função que recebe o estado do edifício e devolve uma string com ligado ou desligado
bool despedimentosDebug(string id); // função para despedir um trabalhador em especifico
virtual void produzir(int i) {}; // produzir recursos numa zona
virtual void produzirnosEdfs(vector<string> zonasdailha, Edificio *bateria); // produzir recursos num edificio
bool existeEdificio(); // retorna tru/false dependendo do ponteiro para edificio
virtual int getarvores() const { return getarvores(); }; // retorna o numero de arvores da floresta
int venderEdf(Edificio* tipoEdf); // função para vender um edificio
void setEdfNull(){this->edificio=nullptr;} // coloca o edificio novamente a nullptr (usada quando se vende um edf)
int venderRec(string tipoRec, int * qnt); // função para vender recursos
Recursos** getRecursosZona(){return tiposrecursos;}
void incrementadiasedificio();
```

Fig.4 – Classe Zona

2.4 Classe Trabalhadores

A Classe Trabalhadores guarda as informações *default* dadas pelo enunciado acerca de um trabalhador, ou seja, o seu preço de contratação, a sua probabilidade de se despedir, após quantos dias se pode despedir, o seu ID e um booleano que representa se já foi movido naquele dia.

Esta classe permite construir um Trabalhador, com um id no formato n.d, despedir um trabalhador, representar um trabalhador, entre outros.

```
static int contador = 0; //contador de trabalhadores

class Trabalhadores{
    int preco; //preco de contratar
    float prob_despedir; //probabilidade de se despedir
    int dia_despedir; // dia em que comeca a probabilidade de se despedir
    int id [2]; //id do trabalhador n.d -> n é o nr de trabalhador e d é o dia que foi contratado
    bool movido = false; //flag para saber se o trabalhador ja foi movido nesse dia
public:
    Trabalhadores(int preco, float prob_despedir, int dia_despedir, int dia); //construtor
    Trabalhadores(){}
    virtual string gettipotrabalhador() const = 0; //funcao necessaria para os "filhos" de Trabalhador: Operario, Mineiro, Lenha
    virtual string getinicial() const = 0; //funcao necessaria para os "filhos" de Trabalhador: Operario, Mineiro, Lenhador
    virtual int getprecotrabalhador() const = 0; //funcao necessaria para os "filhos" de Trabalhador: Operario, Mineiro, Lenhado
    string getid() const{
        ostringstream oss;
        oss<<(id[0])<<"."<<(id[1]);
        return oss.str();
    } //funcao para obter o id dos trabalhadores no formato n.d -> n é o nr de trabalhador e d é o dia que foi contratado
    string getasstring() const {return gettipotrabalhador()+" "+getinicial()+ " ID: "+ getid();} //funcao para representar como
    bool getmovido() const {return movido;} //condicao para ser movido apenas uma vez por dia
    void alteramovido(){movido = true;} //condicao para ser movido apenas uma vez por dia
    bool despedetrabalhador(int diaatual, string tipozona); //funcao para despedir o trabalhador
    void alteraflagamanhecer() {movido = false;} //funcao para alterar a flag no trabalhador ao amanhecer. Assim pode ser movido
```

Fig.5 – Classe Trabalhadores

Todas as classes derivadas de Trabalhadores (Lenhador, Mineiro, Operário) têm a sua implementação das funções virtuais.

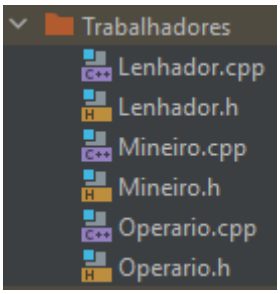


Fig.6 – Classes derivadas de Zona

2.5 Classe Edifício

Na Meta 2, as principais funcionalidades da classe Edifício são a de construir um edifício numa determinada zona, obter o tipo de edifício de uma zona, ligar ou desligar um edifício, subir o edifício de nível e contar os dias de um edifício.

```
class Edificio {
    float prob_desabar;
    bool ligado = false; // flag para saber se um edificio está ligado ou desligado
    int nivel_atual; // nivel atual
    int nivel_maximo; // nivel maximo
    int novo_nivel; // novo nivel
    int preco_nivel; // preco para melhorar de nivel
    int qnt_vigas; // quantidade em vigas para aumentar de nivel
    int cap_max_recursos; // capacidade maxima de armazenamento
    int cap_adicional_nivel; // capacidade adicional de armazendamento por melhoria de nivel
    int contadiasedificio; //contador de dias de existencia do edificio, necessario para o pantano
protected:
//o edificio contem recursos
//a classe edificio tem um ponteiro "tiposrecursos" que aponta para um vetor de recursos
//cada uma das 6 celulas do vetor aponta para um tipo de recurso (ferro, aço, carvão, madeira, vigas de madeira, eletrici
    Recursos *tiposrecursos[6] = {new Ferro(), new Aco(), new Carvao(), new Madeira(), new VigasMadeira(),
                                  new Eletricidade()};
public:
    Edificio(float prob_desabar, int nivel_atual, int nivel_maximo, int novo_nivel, int preco_nivel, int preco_vigas,
             int cap_max_recursos, int cap_adicional_nivel, int contadiasedificio); //construtor
    virtual string gettipoedificio() const = 0; //esta funcao existe em todos os "filhos" de edificio (bateria, centralel
    Recursos ** getrecursos() { return tiposrecursos; } // a funcao getrecursos devolve o endereço do vetor "tiposrecurso
    virtual int podeconstroir(Recursos *pRecursos[6], int dinheiro, string tipo) = 0;
    int decrementa(int qtdadecrementar, int i);
    bool getLigado() { return ligado; } // devolve a flag de um edificio
    void setLigado(bool estado) { ligado = estado; } // liga/desliga um edificio
    bool desabaedificio(); //Funcao para desabar o edificio
    virtual void produzEdf(vector<Trabalhadores *> trabalhadores, vector<string> zonasdailha, Recursos **recursosdazona,
                           Edificio *bateria) {}; // produzir recursos no edificio
    virtual void sobeNivel(int *dinheiroilha, Recursos **recursosdazona,bool *resultado){}
    int getPreco() const { return preco_nivel; }
    int getQntvigas() const { return qnt_vigas; }
    int getNivelatual() const { return nivel_atual; }
    void setNivelatual() {nivel_atual++;}
    int getNivelmax() const { return nivel_maximo; }
    int getCapmax() const { return cap_max_recursos; }
    void setCapmax(){cap_max_recursos += getCapadicional();}
    int getCapadicional() const { return cap_adicional_nivel; }
    void incrementadiasedificio(){contadiasedificio++;}
    int getdiasedificio(){return contadiasedificio;}
```

Fig.7 – Classe Edifício

Todas as classes derivadas de Edifício (Bateria, Central Elétrica, Edifício X, Fundição, Mina Carvão, Mina Ferro) têm a sua implementação das funções virtuais.

2.6 Classe Recursos

Como principais funcionalidades desta Classe, ressaltam-se as funções para vender recursos, obter a quantidade de recursos e o tipo de recurso. No nosso trabalho, os recursos podem pertencer a uma zona ou a um edifício. As zonas começam com recursos gerados de forma aleatória, e os edifícios começam com os recursos a zero.
Todas as classes derivadas de Recursos (Aço, Carvão, Eletricidade, Ferro, Madeira, Vigas de Madeira) têm a sua implementação das funções virtuais.

```cpp
//Os recursos podem ser de 1 zona ou de 1 edificio
class Recursos {
    float preco;
    int quantidade = 0; // quantidade do recurso. Exemplo: quantidade de ferro, quantidade de carvao, etc
public:
    Recursos(float preco); //construtor
    virtual string gettipo() const = 0; //esta informacao é para os filhos dos Recursos (aço, carvao,etc)
    virtual double getprecounidade() = 0; //esta informacao é para os filhos dos Recursos (aço, carvao,etc)
    int getquantidade() { return quantidade; } //função comum para obter a quantidade dos recursos
    void incrementaquantidade(int qtd) {
        //funcao para incrementar a quantidade dos recursos
        //recebe como parametro a quantidade a incrementar (qtd) e acrescenta-a à "quantidade"
        if (qtd + quantidade < 0) quantidade = 0;
        else quantidade += qtd;
    }
    int venderecursos(int *qnt);

};
```

Fig.8 – Classe Recursos

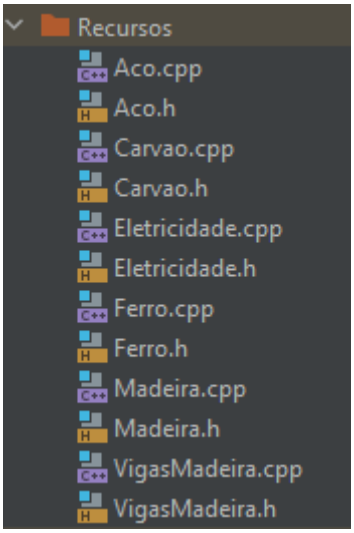


Fig.9 – Classes derivadas de Recursos

3. Funcionalidades Implementadas

3.1 Zonas da Ilha

A construção da ilha consiste numa lista ligada de zonas, em que cada zona tem um ponteiro para a próxima zona e um ponteiro para a primeira posição da próxima zona, sendo que este último pode apontar para uma próxima zona (no caso de ser o primeiro da linha), ou para null (no caso dos restantes).

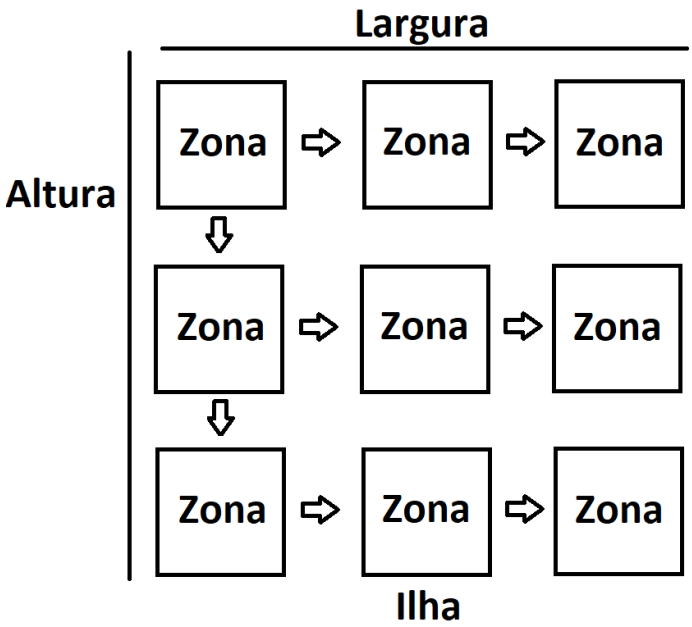


Fig.10– Construção da Ilha por lista ligada de Zonas

A ilha por nós construída tem o seguinte aspeto:

```
 Ilha
|----+----+----+----+----|
|pas |mnt |dsr |dsr |znX |
|    |    |    |    |    |
|    |    |    |    |    |
|0   |0   |0   |0   |0   |
|----+----+----+----+----|
|mnt |mnt |dsr |flr |pnt |
|    |    |    |    |    |
|    |    |    |    |    |
|0   |0   |0   |0   |0   |
|----+----+----+----+----|
|dsr |znX |pas |pnt |pas |
|    |    |    |    |    |
|    |    |    |    |    |
|0   |0   |0   |0   |0   |
|----+----+----+----+----|
|pas |znX |dsr |flr |pnt |
|    |    |    |    |    |
|    |    |    |    |    |
|0   |0   |0   |0   |0   |
|----+----+----+----+----|
|flr |pnt |flr |znX |mnt |
|    |    |    |    |    |
|    |    |    |    |    |
|0   |0   |0   |0   |0   |
|----+----+----+----+----|
```

Fig.11– Aspeto da Ilha construída

No enunciado, eram pedidas as seguintes funcionalidades, que implementámos:

Funcionalidade	Estado
Tamanho da ilha: mínimo 3x3; máximo 8x16	✓
Ocupar no ecrã 4x4 carateres por zona	✓
Pelo menos uma zona de cada tipo, com distribuição equilibrada	✓
Trabalhadores são inicialmente colocados numa zona pasto, definida de forma aleatória	✓

3.2 Recursos

Alguns dos recursos podem ser obtidos diretamente a partir da ilha, enquanto outros só podem ser obtidos a partir da transformação de outros recursos. Para efeitos deste trabalho, os recursos podem ser da zona, ou do edifício.
No enunciado eram pedidas as seguintes funcionalidades:

Funcionalidade	Estado
Ferro pode ser obtido do solo da ilha através de uma mina. Pode ser vendido por 1€ por Kg	✓
Barra de aço pode ser obtido através da transformação de ferro e carvão. Pode ser vendido por 2€ por Kg	✓
Carvão pode ser obtido a partir da ilha através de uma mina. Também pode ser obtido pela transformação de madeira em carvão. Pode ser vendido por 1€ por Kg	✓
Madeira é extraída das florestas da ilha. Pode ser vendida por 1€ por Kg de madeira	✓
Vigas de madeira podem ser obtidas por transformação de madeira numa serração. 2Kg de madeira dão origem a 1 viga. Pode ser vendida por 2€ por viga.	✓
Eletricidade obtida por queima de carvão. Pode ser obtida por 1.5€ por KWh. 1Kg de carvão dá origem a 1KWh de eletricidade.	✓

3.3 Edifícios

No enunciado, eram pedidas as seguintes funcionalidades, que implementámos:

Funcionalidade	Estado
O edifício só trabalha se estiver ligado, e assim que é construído está inicialmente desligado.	✓
Mina de ferro permite obter ferro. Pode ser construída gastando 10 vigas de madeira, em que cada viga pode ser substituída por 10€. Produz 2Kg de ferro por dia. Pode ser melhorada até ao nível 5 em que cada nível aumenta a produção em 1 kg de ferro por dia. Cada nível de melhoramento exige 15 € e 1 viga de madeira (não substituível por €). A mina apenas produz se existir um mineiro na zona em que a mina se encontra. Em cada dia a mina tem 15% de probabilidade de desabar. A mina de ferro armazena até 100 kg de ferro, mais 10 Kg por cada nível adicional. Depois disso, pára de produzir.	✓
Mina de carvão. Permite obter carvão. Pode ser construída gastando 10 vigas de madeira, em que cada viga pode ser substituída por 10 €. Produz 2 Kg de carvão por dia. Pode ser melhorada até ao nível 5 em que cada nível aumenta a produção em 1 kg de carvão por dia. Cada nível de melhoramento exige 10 € e 1 viga de madeira (não substituível por €). A mina apenas produz se existir um mineiro na zona em que a mina se encontra. Em cada dia a mina tem 10% de probabilidade de desabar. A mina de carvão armazena até 100 kg de carvão, mais 10 Kg por cada nível adicional. Depois disso, pára de produzir.	✓

Central elétrica de biomassa: Queima madeira, produzindo carvão e eletricidade. A central elétrica transforma 1kg de madeira em 1 kg de carvão mais 1 KWh de eletricidade por dia, desde que se encontre um operário na zona em que se encontra. A central pode armazenar o carvão produzido até 100 kg de carvão. A eletricidade ficará armazenada numa bateria que se encontre numa zona adjacente, caso contrário perde-se. Para funcionar, a central elétrica tem que estar numa zona adjacente a uma zona do tipo floresta, sendo a madeira obtida a partir da madeira que tenha sido cortada e depositada nessa zona de floresta. Custa 15 €.	✓
Bateria: é um edifício que consiste num enorme bloco de lítio que armazena eletricidade. Tem capacidade de 100 KWh. Custa 10 € e 10 vigas. Adquire automaticamente a energia produzida nas centrais elétricas que estejam colocadas em zonas adjacentes. Pode ser melhorado até ao nível 5 por mais 5 € cada nível	✓
Fundição. Permite obter aço a partir de ferro e carvão. Para funcionar é necessário que a zona em que se encontre seja adjacente a uma zona que tenha uma mina de ferro e a uma mina de carvão ou a uma central elétrica (por causa do carvão). Precisa também de ter um operário na sua zona. Custa 10 €.	✓
Edifício-X: Funciona como uma serração. Transforma 1 madeira em 4 vigas de madeira. Precisa de um operário para funcionar e de estar numa floresta. Custa 10€.	✓

3.4 Zonas

No enunciado, eram pedidas as seguintes funcionalidades, que implementámos:

Funcionalidade	Estado
Em cada zona pode haver zero ou 1 edifício, e zero, 1 ou muitos trabalhadores, independentemente do tipo de edifício ou de trabalhador.	✓
Deserto. São dunas de areia. Permite a construção de qualquer tipo de edifícios. No entanto, as minas sofrem uma redução de 50% de produção por causa dos aluimentos (areia solta).	✓
Pastagem. Aceita qualquer tipo de edifício. Os funcionários que se encontrem aqui nunca pedem a demissão dado que a paz da paisagem reduz o stress.	✓
Floresta. Tem um conjunto inicial de árvores, decidido aleatoriamente entre 20 e 40 árvores. A cada dois dias cresce uma nova árvore, até ao máximo de 100. A floresta produz 1 kg de madeira por cada lenhador que se encontra nessa zona. A madeira cortada fica armazenada ao ar livre sem limite de quantidade. Se forem construídos edifícios numa zona do tipo floresta, morrerá uma árvore por dia e não crescem novas árvores.	✓
Montanha. Aceita qualquer tipo de construção, no entanto o preço de construção é o dobro do normal. As minas aumentam a produção em 100%. As montanhas têm a caraterística interessante de produzirem espontaneamente ferro a 0.1 Kg por dia por cada funcionário que lá se encontre, independentemente do seu tipo. Este ferro extra fica armazenado na própria zona e não tem limite (fica ao ar livre). Dado que andam sempre vergados a apanhar pepitas de ferro, a probabilidade destes trabalhadores pedirem demissão aumenta em mais 5%, e até os lenhadores (se lá estiverem) são afetados.	✓
Pântano. Aceita qualquer tipo de edifício, que têm o custo normal e produzem normalmente. No entanto, passados 10 dias, o edifício afunda-se e desaparece. Os funcionários que lá se encontrem pedem a demissão passados esses mesmos 10 dias (se não for antes pelas suas próprias razões, conforme o seu tipo).	✓
Zona X: Aceita qualquer tipo de edifício, que têm metade do custo normal.	✓

4. Decurso do Jogo

No decurso do jogo o jogador não está posicionado em nenhuma parte da ilha, mas sim os seus edifícios e os seus trabalhadores é que estão posicionados em zonas.
No enunciado, eram pedidas as seguintes funcionalidades, que implementámos:

Funcionalidade	Estado
O Edifício não pode ser movimentado. O trabalhador pode ser movimentado para outra zona, num movimento instantâneo. Todos os trabalhadores estão inicialmente numa zona do tipo pasto	✓
O jogo decorre em dias	✓
Amanhecer: ocorrem os efeitos dos vários tipos de zona	✓
Meio dia: o jogador pode expressar as suas ordens. Cada trabalhador só pode ser movimentado uma vez por dia	✓
Anoitecer: Faz-se a recolha dos recursos obtidos.	✓

5. Interface com o utilizador

No enunciado, eram pedidas as seguintes funcionalidades, que implementámos:

Funcionalidade	Estado
Uso de Qt	×
As zonas são identificadas por linha, coluna. A primeira linha é a linha 1 e a primeira coluna é a coluna 1.	✓
A representação visual deve separar as zonas umas das outras (usando um carater qualquer a servir de separador)	✓
É necessário indicar o conteúdo das várias zonas: 1ª linha: tipo de zona; 2ªlinha: edifício (se houver); 3ªlinha: trabalhadores na zona; 4ªlinha: número total de trabalhadores na zona	✓
Para além da ilha em si, apresentar um sumário da situação do jogo (dia, quantidade de recursos, trabalhadores existentes, etc)	✓
É possível o utilizador solicitar mais informações sobre uma zona em particular	✓

5.1 Comandos

Funcionalidade	Estado
exec <nomeFicheiro>	✓
cons <tipo> <linha> <coluna>	✓
liga <linha> <coluna>	✓
des <linha> <coluna>	✓
move <id> <linha> <coluna>	✓
vende <tipo> <quanto>	✓
cont <tipo>	✓
list <linha> <coluna>	✓
vende <linha> <coluna>	✓
next	✓
save <nome>	×

load <nome>	✗
apaga <nome>	✗
config <ficheiro>	✗
debcash <valor>	✓
debed <tipo> <linha> <coluna>	✓
debkill <id>	✓
lvlup <linha> <coluna>	✓

6. Conclusão

Com este trabalho foi possível construir um programa em C++ que segue os princípios e as práticas de orientação a objetos, tendo sido concluídas com sucesso a maioria das funcionalidades do tema referidas no enunciado.