RELATÓRIO

Trabalho Prático

Programação Orientada a Objetos

Licenciatura em Engenharia Informática

Criação de um simulador/jogo (single-player) de construção e desenvolvimento na linguagem de programação C++.

Trabalho realizado por:

Tomás Gomes Silva - 2020143845

Tomás da Cunha Pinto - 2020144067

Índice

Índice **2**

Introdução **3**

Implementação **4**

Estruturas de Dados 4

Classes 5

Comandos 6

cons 6

cont 7

Funções 8

initIlha 8

mostraIlha 9

isNumber 9

validaComando 10

Conclusão **11**

Introdução

O trabalho prático de Programação Orientada a Objetos consiste na criação de um jogo/simulação na linguagem de programação C++ em modo de consola.
O jogador escolhe o tamanho da ilha e vai industrializando a mesma colocando edifícios e trabalhadores nas várias células da ilha.
Existem várias maneiras do jogador interagir com o jogo de forma a ganhar dinheiro para mais tarde investir noutras zonas e existem também várias jogadas que este pode fazer, tais como: mover um trabalhador, vender edifícios, listar a informação atual do jogo de forma detalhada, etc...

Implementação

Estruturas de Dados

A estrutura de dados que foi utilizada para guardar os objetos da classe Zona foi um vetor bidimensional.
Para criar um vetor bidimensional utilizámos um vetor dentro de um vetor para podermos aceder aos objetos utilizando a notação vetor[i][j].

```
vector< vector<Zona> > matriz;
```

Foi necessário criar um vetor temporário para armazenar objetos da mesma coluna nesse vetor para posteriormente adicionarmos o vetor temporário ao vetor principal.

```
for(int i = 0; i < linhas; i++){
    for(int j = 0; j < colunas; j++){
        tmp.push_back(z);
    }
    matriz.push_back(tmp);
}
```

A função responsável por criar este vetor é a função initIlha. Esta função recebe como parâmetro um vetor bidimensional como referência e retorna o mesmo vetor só que já com os objetos todos inseridos.

Classes

De momento temos uma classe, a classe Zona. Os objetos desta classe correspondem às diferentes zonas da ilha e são compostos por várias variáveis e funções:

- **Variáveis:**
  - zona: guarda o tipo de zona
  - edifício: guarda o tipo de edifício
  - trabalhadores: guarda a representação dos trabalhadores
  - nrTrabalhadores: guarda o número de trabalhadores
  - linha, coluna: guarda a linha e a coluna dessa zona

- **Funções:**
  - setLinha, getLine: define e obtém, respetivamente, a linha da zona
  - setColuna, getColuna: define e obtém, respetivamente, a coluna da zona
  - setZona, getZona: define e obtém, respetivamente, o tipo de zona para esse objeto
  - getEdificio, setEdificio: define e obtém, respetivamente, o edifício para esse objeto
  - getTrabalhadores: obtém o número de trabalhadores na zona
  - setNrTrabalhadores, getNrTrabalhadores: define e obtém, respetivamente, o número de trabalhadores
  - getInfo: obtém toda a informação sobre essa zona em forma de string

As funções cons e cont são as funções que constroem edifícios e contratam trabalhadores, respetivamente.

```
class Zona {

    private:
        string zona;
        string edificio;
        string trabalhadores;
        int nrTrabalhadores;
        int linha, coluna;

    public:
        void setLinha(int l);
        void setColuna(int c);
        int getLinha() const;
        int getColuna() const;
        string getZona() const;
        void setZona(string z);
        string getEdificio() const;
        void setEdificio(string e);
        string getTrabalhadores() const;
        string getInfo() const;
        void setNrTrabalhadores(int n);
        int getNrTrabalhadores() const;
        void cons(string tipo, int linhaX, int colunaX);
        void cont(string tipo);

};
```

Comandos

Os dois comandos que implementámos até agora foram o comando cons, que constrói um edifício numa dada zona, e o comando cont, que contrata um trabalhador para uma zona de pasto. Antes de as funções serem chamadas no objeto a função verificaComando trata de verificar se todos os parâmetros do comando são válidos, só depois é realmente executada a função.

cons

A sintaxe do comando cons é a seguinte: **cons <edificio> <linha> <coluna>**.
O utilizador pode escolher entre construir uma mina de ferro, uma mina de carvão, uma central elétrica, uma bateria ou uma fundição. Para além disso os outros dois parâmetros do comando indicam onde é que o edifício vai ser construído. Se faltar algum parâmetro, ou se algum parâmetro for inválido é apresentada uma mensagem de erro. No caso de todos os parâmetros tiverem sido introduzidos corretamente, a função cons é chamada definindo assim o tipo de edifício dessa zona e dando output a uma mensagem de sucesso.

```
else if (args[0] == "cons" && args.size() == 4){
    int linhaX, colunaX;
    if(isNumber(args[2])) linhaX = stoi(args[2]);
    else{
        cout << "[ERRO] Introduza um numero para a linha" << endl;
        return false;
    }
    if(isNumber(args[3])) colunaX = stoi(args[3]);
    else{
        cout << "[ERRO] Introduza um numero para a coluna" << endl;
        return false;
    }

    if(args[1] == "minaf" || args[1] == "minac" || args[1] == "central" || args[1] == "bat" || args[1] == "fund" || args[1] == "edx"){
        if(linhaX < linhasTab && linhaX >= 0){
            if(colunaX < colunasTab && colunaX >= 0){
                matriz[linhaX][colunaX].cons(args[1], linhaX, colunaX);
                return true;
            } else cout << "[ERRO] Coluna invalida" << endl;
        } else cout << "[ERRO] Linha invalida" << endl;
    } else cout << "[ERRO] Tipo de edificio invalido" << endl;
}
```

```
void Zona::cons(string tipo, int linhaX, int colunaX){
    this->edificio = tipo;
    cout << "Edificio do tipo " << edificio << " CONSTRUIDO na posicao (" << linhaX << "," << colunaX << ")!" << endl;
}
```

cont

A sintaxe do comando cont é a seguinte: **cont <operário>**.

O utilizador pode escolher contratar um mineiro, um operador ou um lenhador. Quando um operário é contratado é colocado numa zona de pasto. A função cont só é executada quando o tipo de trabalhador é válido.

Foi criado um algoritmo para escolher aleatoriamente uma zona de pasto para a qual o trabalhador vai. São adicionados a um vetor temporário todos os objetos cujo o tipo de zona é pasto.

```
else if (args[0] == "cont" && args.size() == 2){
    if(args[1] == "oper" || args[1] == "len" || args[1] == "miner"){
        vector<Zona*> pastos;
        for(int i = 0; i < linhasTab; i++){
            for(int j = 0; j < colunasTab; j++){
                if(matriz[i][j].getZona() == "pas"){
                    pastos.push_back(&matriz[i][j]);
                }
            }
        }
        if(pastos.size()){
            int randomIntger = rand()%(pastos.size() - 0) + 0;
            (*pastos[randomIntger]).cont(args[1]);
            return true;
        } else {
            cout << "[ERRO] Nao existem zonas de pasto disponiveis" << endl;
            return false;
        }
    } else {
        cout << "[ERRO] Esse tipo de operario nao existe" << endl;
        return false;
    }
}
```

```
void Zona::cont(string tipo){
    if(tipo == "oper") trabalhadores.append("O");
    else if(tipo == "miner") trabalhadores.append("M");
    else if(tipo == "len") trabalhadores.append("L");
    else return;
    nrTrabalhadores++;
    cout << "Opearario do tipo " << tipo << " foi CONTRATADO e colocado na zona de pasto (" << linha << ", " << coluna << ")!" << endl;
}
```

Funções

Foram criadas várias funções de suporte tais como: iniciar a ilha, mostrar a ilha, verificar se um número é realmente um número, validar os comandos, etc...
As funções de suporte desenvolvidas até ao momento são as seguintes:

- menu: apresenta o menu inicial
- initIlha: cria objetos, atribui valores iniciais e zonas aleatórias
- mostraIlha: mostra a ilha em formato visual
- mostraInfoTotal: mostra a informação de todas as zonas
- mostraASCII: mostra uma arte em forma de texto a dizer "Ilhéu"
- isNumber: verifica se uma string é um número
- validaComando: validação de um comando inserido pelo utilizador

initIlha

Para inicializar a ilha temos uma função que recebe como parâmetros: a referência para um vetor bidimensional e o número de linhas e de colunas da ilha.
Com os parâmetros todos na posse da função fazemos um ciclo para adicionar os objetos ao vetor (como foi explicado no tópico Estruturas de Dados).
De seguida, é necessário atribuir tipos de zonas aos objetos de forma aleatória e de maneira a que as zonas sejam distribuídas aproximadamente com a mesma frequência. Para isto criamos um vetor de suporte para armazenar as zonas que vão ser atribuídas posteriormente. O primeiro ciclo ignora o facto de haverem objetos aos quais não vai ser atribuído nenhum tipo de zona, no segundo ciclo isto vai ser endereçado.
Utilizamos a função random_suffle para misturar todos os elementos dentro do vetor.

```
vector<string> zonas;
string zonasExistentes[6] = {"mnt", "dsr", "pas", "flr", "pnt", "znZ"};
int celulas = linhas * colunas;
int repeticoes = celulas / 6;
int randIndex;

for(int i = 0; i < 6; i++){
    for(int j = 0; j < repeticoes; j++){
        zonas.push_back(zonasExistentes[i]);
    }
}
random_shuffle(zonas.begin(), zonas.end());
```

```
int count = 0;
for(int i = 0; i < linhas; i++){
    for(int j = 0; j < colunas; j++){
        matriz[i][j].setLinha(i);
        matriz[i][j].setColuna(j);
        matriz[i][j].setNrTrabalhadores(0);
        if(count < (6 * repeticoes)) matriz[i][j].setZona(zonas[count]);
        else{
            randIndex = rand()%(6);
            matriz[i][j].setZona(zonas[randIndex]);
        }
        count++;
    }
}
```

Na imagem da direita estão os ciclos for que tratam de atribuir zonas aos objetos contidos no vetor com os objetos que representam as zonas. São definidos os valores iniciais para cada zona e no if...else... as zonas que ainda não têm um tipo de zona definido vão finalmente ganhar uma zona aleatória escolhida do vetor que armazena as zonas possíveis utilizando a função rand.

A função retorna o vetor bidimensional que foi passado como parâmetro para a função só que já com os objetos todos inseridos com os valores iniciais definidos e com as zonas aleatoriamente escolhidas.

mostraIlha

A função mostraIlha recebe como parâmetros o vetor bidimensional matriz que contém os objetos/zonas da ilha e o número de linhas e colunas total da ilha.
É construído um tabuleiro com caracteres e este é preenchido com a informação de cada objeto.
Utilizámos funções importadas como a setfill e setw que tratavam de garantir que o tamanho da informação apresentada era constante de forma a que a representação do tabuleiro não ficasse desconfigurada.

```
for(int i = 0; i < linhas; i++){
    for(int j = 0; j < colunas; j++){
        cout << "*-------";
        if(j == (colunas - 1)){
            cout <<  "*";
        }
    }
    cout << endl;
```

```
for(int j = 0; j < colunas; j++){
    cout << "*-------";
    if(j == (colunas - 1)){
        cout <<  "*";
    }
}
```

```
cout << endl;
cout << "DIA N/A" << endl;
cout << "Quantidade de Recursos: N/A" << endl;
int nTrab = 0;
for(int i = 0; i < linhas; i++){
    for(int j = 0; j < colunas; j++){
        nTrab += matriz[i][j].getNrTrabalhadores();
    }
}
cout << "Numero de Trabalhadores: " << nTrab << endl;
cout << endl << endl;
```

```
else if(z == 2) cout << "|" << left << setfill(' ') << setw(7) << (matriz[i][k].getTrabalhadores()).substr(0, 6);
else if(z == 3) cout << "|" << left << setfill(' ') << setw(7) << matriz[i][k].getNrTrabalhadores();
else cout << "|       ";
```

isNumber

A função isNumber é uma simples função que utiliza expressões regulares (regex) para verificar se uma string é realmente um número válido. Esta função é utilizada em vários sítios para confirmar o input do utilizador.

```
bool isNumber(string x){
    regex e ("-{0,1}\\d+");
    return (regex_match (x,e));
}
```

A expressão regular verifica se existe um menos ou não atrás de um caractere (feito com o -{0, 1}) que tem obrigatoriamente de ser um dígito positivo (graças ao *d+*). Se a string se enquadrar no formato da expressão regular a função regex_match retorna true e esse é o output da função isNumber.

validaComando

A função validaComando é das funções mais importantes do programa visto que é esta que verifica o input do utilizador e chama as funções para executar o que foi pedido no comando introduzido.
A função começa com um mecanismo para separar as palavras por espaços. As palavras vão sendo adicionadas a um vetor que servirá como um vetor de argumentos que podem ser vistos pelo resto da função de modo a haver comparações pertinentes.
Por exemplo, com o input: **cons minaf 2 2**, obteríamos um vetor deste género: **[“cons”, “minaf”, “2”, “2”]**.

```
vector<string> args;
while (iss){
    string subs;
    iss >> subs;
    args.push_back(subs);
}
args.pop_back();
```

```
if (args[0] == "exec" && args.size() == 2){…
else if (args[0] == "cons" && args.size() == 4){…
else if (args[0] == "liga" && args.size() == 3){…
else if (args[0] == "des" && args.size() == 3){…
else if (args[0] == "move" && args.size() == 4){…
else if (args[0] == "vende" && args.size() == 3){…
else if (args[0] == "cont" && args.size() == 2){…
else if (args[0] == "list" && args.size() >= 1){…
else if (args[0] == "next" && args.size() == 1){…
else if (args[0] == "save" && args.size() == 2){…
else if (args[0] == "load" && args.size() == 2){…
else if (args[0] == "apaga" && args.size() == 2){…
else if (args[0] == "config" && args.size() == 2){…
else if (args[0] == "debcash" && args.size() == 2){…
else if (args[0] == "debed" && args.size() == 4){…
else if (args[0] == "debkill" && args.size() == 2){…
else if (args[0] == "exit" && args.size() == 1){…
else return false;
return false;
```

```
if(isNumber(args[3])) colunaX = stoi(args[3]);
else{
    cout << "[ERRO] Introduza um numero para a coluna" << endl;
    return false;
}
```

Todas as verificações são efetuadas dentro desta função, por exemplo no comando cons, isto é uma das verificações que ocorrem.

Conclusão

O tópico Conclusão encontra-se em desenvolvimento. Aqui serão descritas as conclusões que tirámos ao longo do trabalho.