Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática: 2º Ano – 1º Semestre

Programação Orientada a Objetos

Trabalho Prático – Final

Trabalho realizado por

Pedro Ramos (2019130867)

Rúben Almeida (2019130955)

Índice

Introdução

Este trabalho é constituído por um programa em C++ em que consiste num jogo do tipo “single-player” sobre conquista e expansão territorial. Neste jogo podemos melhorar o nosso império, conquistar novos territórios bem como melhorar recursos (elementos do jogo) necessários às ações do jogador e que este deverá gerir da melhor forma de modo a obter os meios que lhe permitam ter sucesso na expansão do seu império podendo comprar novas tecnologias nas quais o jogador tem de ter recursos (ouro) para as obter.

O jogo decorre em turnos. Cada um é composto por várias fases que ocorrem sequencialmente, e que correspondem a determinadas ações que o jogador pode fazer.

O objetivo do jogo é obter a maior pontuação possível.

Classes

As classes usadas no nosso trabalho foram as seguintes:

- Registo
- Imperio
- Território
- Settings
- Tecnologias

- Grava

Código

Função “Main”:

```
int main(int argc, char** argv) {
    int pontos_finais = 0;
    srand(time(NULL));
    Registo r; //registo de novos territorios
    vector<settings *> def;
    //Inserir as definiçoes iniciais
    inserir_settings(def);
    //Mostra as pessoas no vetor settings
    /*for(auto x: def){
        x->getAsString();
    }*/
    string inst_inicial,inst_sec;
    cout << "Bem-vindo ao jogo" << endl;
    cout << "Introduza as instrucoes iniciais de jogo (Comandos: cria [ou] carrega)" << endl;
    getline(cin,inst_inicial); //Ler instrucoes cria <ter> <n> ou carrega <ficheiro>
    inst_inicial = to_min(inst_inicial); //Colocar para minusculo instrucao inicial

    //Criação do territorio Inicial
    r.registaImperio(new territorio("Territorio Inicial","Territorio Inicial", 9));

    //Fase Inicial - Leitura de Comandos
    r.leitura_instinicial(def, inst_inicial, r);//Fase inicial

    //Inicio do jogo
    r.inicio_jogo(inst_sec,r , pontos_finais);

    //Mostra os dados todos do vetor territorios
    //r.mostraT("todos"); //Mostra todos os territorios
    r.deleteReg(); //Apaga os dados
    //Liberta os vetores restantes
    for(auto a:def){
        delete(a);
    }
    return 0;
}
```

Figura 1-Função Main

Na função “main” do nosso trabalho, começamos por inicializar os pontos do jogo a 0 que durante o jogo serão aumentados seguindo as normas indicadas, é atribuído o vetor com os dados da classe “settings” que guarda as indicações de todos os territórios que podem ser adicionados no jogo, e na função “inserir_settings” é inserido no vetor da classe todos os territórios disponíveis que estão colocados num ficheiro de texto que é lido pelo programa e atribui o tipo do território, o nome e a sua resistência.

Depois disso é lido a instrução inicial do jogo (comando cria ou carrega) e é guardado numa “string” que envia para a função na classe “registo” que irá iniciar o processo de atribuição do território à classe território. Antes de fazer o processo de atribuição é feito a criação do território inicial do jogador que é inserido no vetor dos territórios e na classe imperio pertencendo ao império do jogador. Na função “leitura_instinicial” é feito o processo descrito em cima da inserção de territórios.

Em seguida, quando acaba o processo das configurações de todos os territórios é iniciado o jogo e todas as suas fases.

Classe “Registo”:

```
class Registo{
    vector<territorio *> territorios; //Territorios nao conquistados
    //vector<imperio *> imp; //Imperio do jogador
    imperio imp;
public:
    void proc_recolha(int count_turno, int &count_mont);
    void registaTerritorio(territorio *p);
    void registaImperio(territorio *p);
    void mostraT(string nome_ter);
    void deleteReg();
    void inicio_jogo(string inst_sec,Registo &r, int &pontos_finais);
    void leitura_instinicial(vector<settings *> def,string inst_inicial, Registo &r);
    void proc_conquista(string nome_ter, Registo &r, int &count_mont);
    void aumenta_forcamil();
    void adq_tec(string tech);
    void fase_conquista(Registo &r, int count_turno, int &count_mont);
    void fase_compra(int count_turno,Registo &r, int &count_mont);
    void fase_eventos(int &count_turno, int &count_mont, int forca);
    void fase_verificacoes(int count_turno,string inst_sec,Registo &r, int &count_mont);
    void count_pontos(int &pontos_finais);
};
```

Figura 2- Registo

A função “registo” é a função principal do nosso jogo e é ela que determinar como o jogo irá proceder, a classe contem um vetor da classe território que irá guardar todos os territórios disponíveis no mundo, que contem o seu nome, o seu tipo e a sua resistência. Contem ainda uma variável da classe império que é responsável por controlar o império do jogador. A função “registaTerritorio” é usada para registar um novo território no mundo) e a função “registaImperio” é usada para adicionar um território já existente que agora irá pertencer ao império que mais tarde pode vir a ser conquistado por um império inimigo.

A função “mostraT” é utilizada quando é utilizado o comando lista ou lista <nome> e mostra informações do jogo.

A função “inicio_jogo” é responsável pela chamada das funções necessárias por cada fase e pela contagem de turnos do jogo.

As funções “fase_conquista” e “proc_conquista” são responsáveis pelo processo de conquista de novos territórios que irão pertencer ao império do jogador e serão feitas todas as verificações e atribuições necessárias no processo.

A função “proc_recolha” é usada na fase de recolha de produtos e ouro para o armazém e o cofre do império é feita todas as verificações necessárias para a recolha por vários tipos de territórios no império devido ao facto dos territórios produzirem diferentes quantidades e diferentes tipos.

A função “fase_compra” é utilizada na fase de compra que é necessária para adquirir novas tecnologias para o império (função “adq_tec”) ou então aumentar a força militar do império (função “aumenta_forcamil”).

A função “fase_eventos” atribui qual será o evento que acontecerá na fase que decorre, através de um fator sorte (1-4) é atribuído um evento com as suas consequências para o império do jogador.

A função “fase_verificacoes” é usada em todas as funções de cada fase para se for necessário inserir algum comando lista, comandos de DEBUG ou outro tipo de comando para verificação.

A função “count_pontos” faz a contagem dos pontos e é usada durante cada fase.

Função “ Inserir_settings “:

```
void inserir_settings(vector<settings *> &def){
    string texto;
    string tipo_global,nome;
    int res, p_prod, p_ouro;
    ifstream ter_settings("settings.txt");
    if(ter_settings.is_open()){
        while(getline(ter_settings, texto)){
            //Ler linha a linha
            //cout << texto << endl;
            istringstream buffer(texto);
            if(buffer >> tipo_global && buffer >> nome && buffer >> res &&  buffer >> p_prod
                    && buffer >> p_ouro){ //Ler os dados no ficheiro settings.txt
                def.push_back(new settings(tipo_global, nome, res, p_prod, p_ouro)); //Introduzir as settings dos territorios
            }
        }
    }else{
        cout << "Não foi encontrado o ficheiro com as definições dos territorios\n";
    }
}
```

Figura 3- Inserir Settings

Na função “inserir_settings” do nosso trabalho, o mesmo lê o ficheiro ".txt" que está localizado na pasta do projeto no qual tem o modelo " Tipo de Terreno - Território - Resistência - Produção - Ouro", na qual foi usada na primeira meta. Após ler, ele guarda num ”buffer” linha a linha e adiciona ao vetor da classe “settings” se cumprir todos os requisitos.

Classe “settings”:

```
class settings {
    string tipo_global,nome;
    int res, p_prod, p_ouro;
public:
    settings(string tipo_global,string nome, int res, int p_prod, int p_ouro);
    ~settings();
    string getAsString() const;
    void setP_ouro(int p_ouro);
    int getP_ouro() const;
    void setP_prod(int p_prod);
    int getP_prod() const;
    void setRes(int res);
    int getRes() const;
    void setNome(string nome);
    string getNome() const;
    void setTipo_global(string tipo_global);
    string getTipo_global() const;
    //tolower
};
```

Figura 4- Settings

A classe “settings” é guardado como foi dito anteriormente todos os territórios que podem ser adicionados ao jogo e as suas definições e esta classe serve para garantir que todos os dados dos territórios sejam verificados antes de inseridos no jogo.

Função “inicio_jogo”:

```
void Registo::inicio_jogo(string inst_sec,Registo &r, int &pontos_finais){
    imp.inicializa_tec();
    int count_turno = 0;
    int count_mont = 0;
    do{

        cout << "Inicio do turno: " << (count_turno+1) << endl;
        cout << "Fase de conquista e recolha de produtos e ouro:" << endl;
        fase_conquista(r, count_turno,count_mont);
        //Fase de recolha de produtos e ouro
        r.proc_recolha(count_turno, count_mont);
        cout << "Fase de Compra:" << endl;
        fase_compra(count_turno,r, count_mont);
        cout << "Fase de eventos:" << endl;
        fase_eventos(count_turno,count_mont,0);
        //na ultima fase de cada turno aumentar o count_turno
        count_turno++;
        count_pontos(pontos_finais); //Funcao usada para calcular quanto pontos tem atualmente
        //cout << "Pontos atuais: " << pontos_finais << endl;
    }while(count_turno < 12);
    cout << "- Jogo terminado -" << endl;
    r.mostraT("todos"); //Mostra todos os territorios
    cout << "Terminou o jogo com " << pontos_finais << " pontos." << endl;
}
```

Figura 5-Inicio Jogo

A função “inicio_jogo” está responsável pelo funcionamento de todas as fases do jogo, é nela que é chamada cada função atribuída a cada fase do jogo, inicializa o vetor com todas as tecnologias disponíveis para serem adquiridas, inicializa a variável para contar os turnos e para

contar os turnos desde que o território “montanha” é adquirido pelo facto de só produzir no 2º turno após ser conquistado.

Classe “Tecnologias”:

```
class tecnologias{
    string nome, objetivo;
    int preco;
public:
    tecnologias(string nome_tec, string obj, int p);
    ~tecnologias();
    int GetPreco() const;
    string GetObjetivo() const;
    string GetNome() const;
    string getAsString() const;

};
```

Figura 6-Tecnologias

A classe “tecnologias” é utilizada para identificar os vários tipos de tecnologias disponíveis, indicando o nome, o seu objetivo e o seu preço.

Classe “Territorio”:

```
class territorio {
    static int contador; //Para adicionar ao nome de territorio
    string tipo_global,nome;
    int res;
public:
    territorio(string tipo,string nome_t, int res_t);
    ~territorio();
    virtual string getAsString() const;
    void SetRes(int res);
    int GetRes() const;
    void SetNome(string nome);
    string GetNome() const;
    void SetTipo_global(string tipo_global);
    string GetTipo_global() const;
};
```

Figura 7-Territorio

A classe “território” é usada para identificar os territórios adicionados pelo jogador que estão disponíveis no mundo do jogo, identifica o território pelo seu nome, o seu tipo que irá influenciar as ações do jogo e sua resistência que será usada para calcular se o jogador conseguirá conquistar os vários territórios disponíveis.

Classe “Imperio”:

```
class imperio{
    vector<territorio *> ter;
    bool drones, misseis, defesas, bolsa, banco_central;
    int forca_mil, cofre, armazem, forca_max, cofre_max, armazem_max;
    string evento_passado;
    int ultimo_fator;
    vector<tecnologias *> tec;
public:

    imperio();
    ~imperio();
    std::string getAsString() const;
    void setForca_mil(int forca_mil);
    int getForca_mil() const;
    void setBanco_central(bool banco_central);
    bool isBanco_central() const;
    void setBolsa(bool bolsa);
    bool isBolsa() const;
    void setDefesas(bool defesas);
    bool isDefesas() const;
    void setMisseis(bool misseis);
    bool isMisseis() const;
    void setDrones(bool drones);
    bool isDrones() const;
    string getDados() const;
    string mostraTec() const;
    void setArmazem_max(int armazem_max);
    int getArmazem_max() const;
    void setCofre_max(int cofre_max);
    int getCofre_max() const;
    void setForca_max(int forca_max);
    int getForca_max() const;
    void setArmazem(int armazem);
    int getArmazem() const;
    void setCofre(int cofre);
    int getCofre() const;
```

Figura 8 - Classe império

Na classe “imperio” é a classe que controla todos os dados que representam o império do jogador, guardar num vetor da classe “território” todos os territórios que estão no império do jogador no momento, as variáveis booleanas indicam quais as tecnologias que o jogador já adquiriu ou não. Tendo acesso ao vetor com todas as tecnologias disponíveis da classe “tecnologias”.

Tem informação dos valores que o jogador tem no seu cofre, no seu armazém e os valores máximos que estes podem acumular. Indica qual é a força militar que o império dispõe e o seu valor máximo.

Guarda ainda o nome do último evento realizada na fase dos eventos do jogo e o ultimo fator sorte no jogo e estes valores todos podem ser consultados através do comando “lista”.

```
//Açoes no imperio
void add_cofre(int valor);
void add_armazem(int valor);
void add_forcamil(int valor);
void atr_evento(string evento);
void add_territorio(territorio* t);
int verifica_ter(territorio *t);
int getCountTer();
void proc_recolha(int count_turno, int &count_mont);
void evento_invasao(int count_turno, int &count_mont);
void toma_tec(string nome);
void toma_ter(string nome, int &count_mont);
void inicializa_tec();
void setUltimo_fator(int ultimo_fator);
int getUltimo_fator() const;
```

Figura 9 - Classe império / Funções

Na mesma classe, tem algumas funções necessárias para controlar os valores do império e executar alguns passos das diferentes fases do jogo, as funções “add_cofre”, “add_armazem”, “add_forcamil”, “atr_evento” servem para atribuir valores aos componentes do império.

A função “add_territorio” é utilizada para adicionar um ponteiro do vetor “território” que guarda todos os territórios disponíveis e adiciona ao império se todas as verificações forem bem-sucedidas.

A função “proc_recolha” é usada para concluir a fase da recolha do jogo e a função “evento_invasao” é chamada quando na fase de eventos for sorteado o evento invasão e o ultimo território adicionado fica em risco de sair do império.

As funções “toma_tec” e “toma_ter” são usadas para executar os comandos de DEBUG pedidos.

Função “evento_invasao”:

```
if(count_turno < 6){
    forca_total = r + 2;
    rest = ter.back()->GetRes();
    if(defesas == true){
        if(forca_total > (rest+1)){
            if(to_min(nome_ter) == "territorio inicial"){
                cout << "O seu imperio apenas tem o territorio inicial e foi invadido. Perdeu o jogo." << endl;
                count_turno = 12;
            }else{
                //Territorio do imperio é perdido
                if(nome_ter == "montanha"){
                        count_mont = 0;
                }
                cout << "O territorio " << nome_ter << " que pertencia ao seu imperio foi conquistado durante o evento da invasao." << endl;
                ter.pop_back();
            }
        }else{
            //O seu territorio do imperio não conseguiu ser invadido
            cout << "Tentativa de invasao falhada, o territorio <" << nome_ter << "> permanece no imperio." << endl;
        }
    }else{
        if(forca_total > rest){
            if(to_min(nome_ter) == "territorio inicial"){
                cout << "O seu imperio apenas tem o territorio inicial e foi invadido. Perdeu o jogo." << endl;
                count_turno = 12;
            }else{
                //Territorio do imperio é perdido
                if(nome_ter == "montanha"){
                        count_mont = 0;
                }
                cout << "O territorio " << nome_ter << " que pertencia ao seu imperio foi conquistado durante o evento da invasao." << endl;
                ter.pop_back();
            }
        }else{
            //O seu territorio do imperio não conseguiu ser invadido
            cout << "Tentativa de invasao falhada, o territorio <" << nome_ter << "> permanece no imperio." << endl;
        }
    }
```

Figura 10 - Função evento_invasao

Nesta primeira parte da função “evento_invasao” verificamos em qual turno nos encontramos no jogo e pelo fator sorte adicionamos apenas 2 unidades a força total e tentamos invadir o último território adicionado ao império e se a sua resistência for inferior a força total gerada o território é retirado do império do jogador e deixa de contribuir para o império e se o jogador apenas tiver o território inicial no seu império e este for invadido perde o jogo.

```
}else{
    forca_total = r + 3;
    rest = ter.back()->GetRes();
    if(defesas == true){
        if(forca_total > (rest+1)){
            if(to_min(nome_ter) == "territorio inicial"){
                cout << "O seu imperio apenas tem o territorio inicial e foi invadido. Perdeu o jogo." << endl;
                count_turno = 12;
            }else{
                //Territorio do imperio é perdido
                if(nome_ter == "montanha"){
                        count_mont = 0;
                }
                cout << "O territorio " << nome_ter << " que pertencia ao seu imperio foi conquistado durante o evento da invasao." << endl;
                ter.pop_back();
            }
        }else{
            //O seu territorio do imperio não conseguiu ser invadido
            cout << "Tentativa de invasao falhada, o territorio <" << nome_ter << "> permanece no imperio." << endl;
        }
    }else{
        if(forca_total > rest){
            if(to_min(nome_ter) == "territorio inicial"){
                cout << "O seu imperio apenas tem o territorio inicial e foi invadido. Perdeu o jogo." << endl;
                count_turno = 12;
            }else{
                //Territorio do imperio é perdido
                if(nome_ter == "montanha"){
                        count_mont = 0;
                }
                cout << "O territorio " << nome_ter << " que pertencia ao seu imperio foi conquistado durante o evento da invasao." << endl;
                ter.pop_back();
            }
        }else{
            //O seu territorio do imperio não conseguiu ser invadido
            cout << "Tentativa de invasao falhada, o territorio <" << nome_ter << "> permanece no imperio." << endl;
        }
    }
}
```

Figura 11 - Função evento_invasao 2

Se estiver no segundo ano do jogo adiciona ao fator sorte 3 unidades para a tentativa de invasão e as verificações funcionam da mesma maneira.

Questões

1. Quais foram as classes consideradas na primeira versão da aplicação que foi testada?

Nesta primeira versão, criámos as classes “território”, “registo” e “settings”, “imperio” e “tecnologias”, “grava”.

2. Quais os conceitos/classe que identificou ao ler o enunciado?

Ao ler o enunciado identificamos que obrigatoriamente tínhamos de criar uma classe para guardar os dados iniciais do jogo, classe “settings” que contêm todos os territórios disponíveis e outra classe “território” para armazenar todos os territórios inseridos pelo utilizador, uma classe para controlar os dados do jogador, a classe “imperio”.

3. Relativamente a duas das principais classes da aplicação, identifique em que classes ou partes do programa são criados, armazenados e destruídos os seus objetos.

As duas principais classes da aplicação são as classes “território”, “settings” e “imperio”. A classe “settings” é criada e inicializada no início da aplicação e destruído os seus objetos no final do programa. A classe “território” é criada num vetor, dentro da classe registo, e inicializada antes do processo de adição de territórios sendo destruída no final do programa através da classe “registo”. A classe “imperio” controla os dados necessários para o jogador tem todos os dados necessários ao imperio.

4. Indique um exemplo de uma responsabilidade atribuída a uma classe que esteja de acordo com a orientação dada acerca de Encapsulamento.

Um exemplo de uma responsabilidade atribuída a uma classe que esteja de acordo com a orientação dada acerca de encapsulamento é a classe “território” e “imperio” dentro da classe “Registo”.

5. De entre as classes que fez, escolha duas e justifique por que considera que são classes com objetivo focado, coeso e sem dispersão.

As classes com objetivo focado, coeso e sem dispersão das classes criadas são a “território” e a “settings” e a classe “imperio”. A classe “território” é usada para guardar todas as características de cada território, adicionado pelo utilizador e a partir dela podem se fazer todas as alterações aos territórios.

A classe “settings” guarda as características de todos os territórios disponíveis ao utilizador e serve como guião para a classe “território”. A classe “imperio” controla os dados necessários para o jogador tem todos os dados necessários ao imperio.

6. Relativamente à aplicação entregue, quais as classes que considera com responsabilidades de interface com o utilizador e quais as que representam a lógica?

Relativamente à aplicação, as classes que consideramos como responsabilidades de interface com o utilizador e que representam a lógica é a classe “registo”.

7. Identifique o primeiro objeto para além da camada de interação com o utilizador que recebe e coordena uma funcionalidade de natureza lógica?

O primeiro objeto para além da camada de interação com o utilizador que recebe e coordena uma funcionalidade de natureza lógica é a criação do território inicial do utilizador sendo criado na inicialização do jogo.

8. A classe que representa a envolvente de toda a lógica executa em pormenor muitas funcionalidades, ou delega noutras classes? Indique um exemplo em que esta classe delega uma funcionalidade noutra classe.

A classe “território” delega na classe “registo” algumas funcionalidades como a verificação de alguns campos e de outras funcionalidades adicionais para o bom funcionamento das iterações do jogo. A classe “imperio” delega na classe “registo” é chamada algumas funções internas da classe “imperio” através de funções da classe “registo”.

9. Dê um exemplo de uma funcionalidade que varia conforme o tipo do objeto que a invoca. Indique em que classes e métodos está implementada esta funcionalidade.

A classe para concluir o comando grava apesar de não estar completa nem a funcionar, conseguimos concluir que teríamos que associar a classe grava à classe “imperio” através de herança entre classes. Para quando o comando grava fosse executado, era chamada a função duplica na classe “grava” que iria duplicar o conteúdo que estaria na classe “imperio” guardando na classe “grava” para se depois se o utilizador quisesse voltaria a chamar esta fase gravada e passaria a atual do jogo.

```
class grava : public imperio{

public:
    grava();
    ~grava();
//    grava *duplica() const;
};
```

Figura 12 - Criação classe grava

```
//virtual imperio* duplica() const = 0;
```

Figura 13 - Função duplica na classe imperio

```
//grava * duplica() const override{
////     return new grava(*this);
//}
```

Figura 14 - Protótipo da função duplica

10. Apresente as principais classes da aplicação através da seguinte informação:

Classe: settings

Responsabilidades:

- Permite consultar todos os dados necessários relativamente aos territórios disponíveis a serem adicionados.

- Permite apenas fazer a consulta dos territórios.

Colaborações: colabora com a classe “território” e “registo” mutuamente.

Classe: territorio

Responsabilidades:

- Permite consultar todos os dados dos territórios atuais adicionados no jogo.

- Permite fazer alterações aos campos a cada território de forma a avançar com o processo do jogo seguindo várias restrições.

Colaborações: colabora com a classe “settings”, “registo” e “imperio”.

Classe: imperio

Responsabilidades:

- Permite consultar todos os dados necessários ao império do jogador

- Guarda num vetor de classe do território com os territórios atuais no império do jogador

- Manter organizado e seguro os dados do jogador

Colaborações: colabora com a classe “registo”, “território” e “tecnologias”

Classe: registo

Responsabilidades:

- Controlar todos os passos do jogo e chamar as funções e classes necessárias ao jogo atual

- Controlar o jogo e os pontos obtidos pelo jogador e o seu percurso no jogo

Colaborações: colabora com a classe “território”, “imperio” e “settings”

Conclusão

Neste trabalho abordamos a matéria que foi lecionada durante este semestre a Programação Orientada a Objetos na criação deste jogo de conquista de territórios utilizando objetos/classes para comunicarem entre si para a evolução do jogo.

Cumprimos a maioria dos objetivos pedidos no enunciado, houve alguns pontos do trabalhos que demoraram mais tempo a analisar e tentar adaptar da melhor maneira para um funcionamento mais robusto das classes usadas e etapas do jogo. Contudo pensamos que o produto final apresentado representa o nosso trabalho e vai de encontro as metas pedidas.

Este trabalho foi importante para uma melhor compreensão da matéria dada e em certa forma ajudou a aprofundar e aperfeiçoar os nossos conhecimentos sobre a mesma com as pesquisas e novos recursos descobertos.