Instituto Superior de Engenharia de Coimbra

Engenharia Informática

Programação Orientada a Objetos
Trabalho Prático 2020/21

Meta 1

Diogo Miguel Pinto Pascoal, 2018019825, Lab.2

Nuno Gabriel Tavares Honório, 2019126457, Lab.6

Índice

Introdução

O trabalho prático da disciplina de Programação Orientada a Objetos pretende que se construir em C++ um jogo do tipo single-player sobre conquista e expansão territorial.

Para a realização do mesmo, foram interpretados diferentes conceitos e entidades que foram traduzidas em classes no projeto, cujo mesmos irão ser descritos na próxima secção. Também será justificado a organização do programa e o progresso do mesmo ao longo deste relatório.

Classes – Conceitos da Primeira versão (1ª Meta)

Serão de seguida apresentados os diferentes conceitos do enunciado que foram traduzidos em classe nesta primeira meta. Todas as classes têm apenas get's e set's que se mostraram necessárias à execução do programa, sendo que não existem get's e set's não utilizados. Também são apresentadas as funcionalidades / funções pela qual a classe é responsável.

Classe Mundo

A primeira classe evidente ao enunciado do trabalho prático foi a criação de uma classe Mundo, que representa o conjunto de territórios presentes no mundo e, consequentemente, do jogo. Assim que o programa é iniciado, é gerado um objeto Mundo que irá ser acessível a todas as funções exteriores que necessitem de o aceder ao conjunto de territórios pertencente ao mundo. Este objeto é iniciado na função main aquando da execução do programa.

Deste modo a classe mundo apenas tem 1 propriedade: um vetor territórios.

```
class Mundo{
public:
    vector<Territorio*> territorios;
```

Figura 1 - Propriedades da classe Mundo

O objeto mundo é responsável pela criação dos territórios, integrando-os automaticamente no seu vetor. Também tem uma função de que devolve um ponteiro para um território através do seu nome, assim como a função lista que se encontra overloaded, uma para listar todos os territórios do mundo e outra para listar a informação de um território em específico.

```
void criaTerritorioInicial();
void criaTerritorio(const string& tipo,int n);

string lista(const string& nomeTerritorio);
string lista();

Territorio* devolvePonteiroTerritorio(const string& nomeTerritorio);
```

Figura 2 - Funções da classe Mundo

Classe Territorio

Outra classe clara ao enunciado é a classe Território, que representa então cada um dos territórios constituintes do jogo e do mundo. São estes o objetivo de conquista por parte do jogador.

Sendo a classe Território a classe principal do programa, esta é também a que apresenta mais propriedades:

- nome: Indica o nome dado ao território pelo sistema (de notar que o utilizador nunca escolhe o nome do território e o sistema gera o mesmo automaticamente aquando da criação do objeto).
- tipo: Apesar de nesta meta ainda não ser usada, o Tipo do território, como o nome indica, representa o tipo de território que o mesmo é, sendo as opções disponíveis as seguintes: da categoria de Continente podemos ter Planície, Montanha, Fortaleza, Mina, Duna e Castelo; da categoria de Ilhas podemos ter ainda Refúgio dos Piratas e Pescaria. De notar que ainda não são implementados estes conceitos nesta primeira meta e que o conceito de categoria é, por enquanto, apenas virtual e não se encontra definido no projeto.
- resistencia: Representa a força necessária de ultrapassar quando o jogador quiser conquistar o território. Se o território tiver sido conquistado, este valor representa a força com que luta contra os invasores.
- produtosPorTurno: Representa a quantidade de produtos que o território gera para o império por turno, caso este seja conquistado.
- ouroPorTurno: Assim como o produtosPorTurno, representa a quantidade de ouro que o território gera para o império por turno caso o mesmo esteja conquistado.
- pontos: Indica a quantidade de Pontos de vitória com que contribui para o império.
- conquistado: Indica se o território em questão já foi conquistado

Nesta meta existe apenas 1 tipo de território simbólico de nome “Territorio Generico”, visto que a implementação dos diferentes tipos de território ainda não é realizada nesta meta. Deste modo, existe 1 variável *int static* nTerritorioGenerico que indica ao programa quantos territórios deste tipo já foram criados. Esta variável estática é crucial à geração do nome para o território.

Também existe 1 território especial de nome "TerritorioInicial" que é gerado aquando da iniciação do objeto mundo, que representa o ponto de partida para o jogador. Este território tem características específicas, nomeadamente:

```
nome = "TerritorioInicial";
tipo = "Territorio Inicial";
resistencia = 9;
produtosPorTurno= 1;
ouroPorTurno = 1;
pontos = 0;
conquistado = false;
```

Figura 3 – Caraterísticas do Território Inicial

O jogador **não** tem permissão para criar outro território deste tipo.

Esta classe apenas tem os seus get's e set's para além do construtor e destruídor.

Classe Imperio

A classe Imperio representa o domínio do jogador sobre o mundo, sendo que, assim como o mundo, existe apenas um objeto Imperio que é instanciado no início do programa dentro do main, logo após a criação do mundo. O território inicial que é criado junto com o mundo é também automaticamente conquistado para o império.

Deste modo, as propriedades do Império são as seguintes:

- Vetor Territorio*: Vetor de ponteiros para os territórios que o império conquistou (não guarda o objeto em si, apenas um ponteiro para o objeto território guardado no objeto mundo).
- armazemProdutos: Indica a quantidade de produtos que o Império tem.
- cofreOuro: Indica a quantidade de ouro que o Império tem.
- tamArmazem e tamCofre: Valor máximo do armazém e do cofre, respetivamente.
- forcaMilitar e maxForcaMilitar: Representa a força militar do Império, assim como o seu limite, respetivamente.

Todas estas propriedades são privadas.

```
private:
    vector<Territorio*> territoriosConquistados;
    int armazemProdutos;
    int cofreOuro;
    int tamArmazem,tamCofre;
    int forcaMilitar,maxForcaMilitar;
```

Figura 4 – Propriedades do Império

A classe é responsável por conquistar os territórios para si, obter o ouro e produtos dos territórios por si conquistados e comprar unidades militares. Também dispõe de uma função listai() que apresenta a informação acerca do império e 2 get's, um para obter o ouro do império e outro para obter os produtos.

```
public:
    Imperio(Mundo& mundo);
    int ConquistaTerritorio(Mundo& mundo,const string& nomeTerritorio);
    void processaOuroProdutos();
    void compraUnidadeMilitar();
    int getOuroImperio() const;
    int getProdutosImperio() const;
```

Figura 5 – Funções do Império

Classe Interface

Por último temos a classe interface que não representa um conceito do jogo em si, mas sim trata de gerir o jogo e interagir com o utilizador.

Tem 3 propriedades: o inteiro stage, inteiro turno e inteiro fase, sendo que cada um destes representa uma componente mecânica do processo de jogo. O stage representa em que estado se encontra o jogo, sendo que pode ter um destes 3 valores:

0 – Configuração
1 – In-Game
2 – Endgame

Em configuração, o jogador vai carregar e criar os territórios para o jogo decorrer. Todos os momentos de turnos, conquistar territórios, obter ouro e produtos, etc. decorrem neste 2º stage In-Game, que representa o jogar o jogo em si. Por último, em Endgame o jogo terminou e é apresentada informação final acerca do jogo.

Quando está em jogo, as outras duas variáveis entram em ação: fase indica em que fase do jogo nos encontramos:

0 – Conquistar Território ou Passar

1 – Recolher produtos e ouro

2 – Comprar unidade militar / Obter tecnologia[1]

3 – Eventos[1]

O inteiro turno indica em que turno se encontra o jogo.

A interface é arrancada através da função run que é chamada no início do programa no main. Esta classe tem todas as funções para interpretar comandos, ler ficheiros e prosseguir com o jogo.

[1] Ainda não implementado

```
class Interface{
private:
    int stage;
    int turno;
    int fase;

public:
    explicit Interface(Mundo&);
    ~Interface();
    void run(Mundo& mundo,Imperio& imperio);

private:
    void processaFicheiro(const string& nomeFicheiro,Mundo& mundo);
    void apresentaListaComandos() const;

    int processaComando(Mundo& mundo,Imperio& imperio);
    void processaComandoDoFicheiro(istringstream &iss, Mundo &mundo);
    int processaComandoJogo(Mundo& mundo,Imperio& imperio);
    void checkIfEndgame();
    void passaTurno();
};
```

Figura 6 – Classe Interface

Acerca de: Classes Importantes

Consideramos que a classe Imperio e Territorio são duas classes com objetivo bem focado, coeso e sem dispersão: O Império representa o estado do jogador em relação ao jogo, sendo que todos os seus campos e funções associadas são diretamente associadas ao próprio sem alterar conceitos e valores exteriores (excluindo o campo conquistado nos Territórios). Do mesmo modo, a classe Território representa bem explicitamente cada território no jogo, com as suas propriedades bem estabelecidas e claras, assim como funções apenas acerca de si e para direcionar informação sobre si para outras funções que possam necessitar do mesmo.

Como referido aqui, a classe Interface é a responsável por fazer a ligação entre o utilizador e a lógica do jogo. Logo de seguida a informação recebida pelo jogador é processada na mesma classe através das diferentes funções “processa *”. Como é possível concluir, as diferentes funcionalidades são todas geridas nesta classe que por sua vez irá aceder a informação e funções necessárias para concluir as suas tarefas indicadas pelo utilizador.

Classes – Conceitos da Versão Final (2ª Meta)

Nesta secção iremos explicar as classes que foram alteradas e/ou acrescentadas ao projeto.

Classes dos Continentes e das Ilhas

Nesta versão final para a segunda meta, abandonou-se o conceito de território genérico, implementando-se os diferentes tipos de territórios dos Continentes e das Ilhas.

Apesar de não existir uma classe Continente ou uma classe Ilha, as classes do tipo Continente encontram-se num ficheiro (Continente.cpp) e as do tipo Ilha encontram-se noutro (Ilhas.cpp).

Todos os territórios derivam da classe base Territorio:

```
class Territorio {
protected:
    string nome;
    int resistencia=0;
    bool conquistado=0;
public:


    string getNome();

    int getResistencia() const;

    virtual int getProdutos() =0;

    virtual Territorio* clone() const = 0;

    virtual int getOuro() =0;

    virtual int getPontos() =0;
    ~Territorio();

    bool getConquistado() const;

    void setConquistado(bool newState);

};
```

Os vetores de Territórios do Mundo e do Império são à base desta classe baste abstrata.

Todas as classes derivadas deste território (Planicie, Montanha, Fortaleza, Mina, Duna e Castelo nos Continentes e Refugio e Pescaria nas Ilhas) têm a sua implementação das funções virtuais.

Classe Interface

Esta classe é a classe que desempenha o papel principal. Apresenta as funções e é responsável por chamar as funções gerais. Seja criar territórios, conquistar, etc...

Apresenta também as funções de leituras de ficheiros e comandos.

É feita uma gestão de menus e fases através de switch case, switch(menu) e switch(fase) respetivamente.

```
void Interface::apresentaListaComandos(int menu) {  // apresentaListaComandos(menu);
    switch (menu) {
        case 0:
            cout << "********************" << endl;
            cout << "Comandos disponiveis:" << endl;
            cout << " - cria <tipo de territorio> <numero de territorios>" << endl;
            cout << " - carrega <nomeFicheiro>" << endl;
            cout << " - lista " << endl; // mostra todos os detalhes
            cout << " - lista <nomeTerritorio> " << endl; // mostra os detalhes de 1 território
            cout << " - listaimperio " << endl;
            cout << " - listatecnologias" << endl;
            cout << " - avanca" << endl;
            cout << " - sair" << endl;
            cout << "********************" << endl;
            cout << "Comando: ";
            break;
        case 1:
            cout << "********************" << endl;
            cout << "Comandos disponiveis:" << endl;
            cout << " - conquista <nomeTerritorio>" << endl;
            cout << " - passa" << endl;
            cout << " - lista " << endl;
            cout << " - lista <nomeTerritorio>" << endl;
            cout << " - listaimperio " << endl;
            cout << " - listatecnologias" << endl;
            //new
            cout << " - toma <terr/tec> <nome>" << endl;
            cout << " - modifica <ouro/prod> <quantidade>" << endl;
            cout << " - fevento <nome>" << endl;
            cout << " - grava <nome>" << endl;
            cout << " - ativa <nome>" << endl;
            cout << " - apaga <nome>" << endl;
            //
```

Figura 7 – Switch case Menu

```
int Interface::processaComandoJogo(Mundo &mundo, Imperio &imperio) {
    switch (fase) {
        case 1: { // fase de conquista

            string comando;
            do {
                apresentaListaComandos(menu);
                getline(cin, comando);
                stringstream ss(comando);
                ss >> comando;

                if (comando == "conquista") {
                    if (ss.good()) {
                        string nomeTerritorio;
                        ss >> nomeTerritorio;
                        if (imperio.ConquistaTerritorio(mundo, nomeTerritorio) == 1) {
                            fase = 2;
                            break;
                        }
                    }
                    else {
                        cout << "Argumentos em falta no conquista! Sintaxe: conquista <nomeTerritorio>" << endl;
                    }
                }
                else if (comando == "toma") {
                    string tipo;
                    ss >> tipo;

                    if (tipo == "terr") {
                        string nome;
                        ss >> nome;
                        imperio.tomaTerr(mundo, nome);
```

Figura 8 – Switch case Fases

Além disso, também faz a gestão de quando o jogo chega ao fim, seja pelo meio de invasão e derrota do nosso território ou pela chegada do final do turno 12.

Também apresenta uma função para passar o turno e apresentar a mensagem de mudança do mesmo. Esta função além de passar o turno faz a conversão para o 2º ano quando alcança o turno 7.

```
void Interface::passaTurno(Imperio& imperio) {

    cout.flush();
    cout << "********************" << endl
         << "--FIM DO TURNO " << turno << " DO ANO " << ano << "--"<< endl
         << "********************" << endl;
    fase = 1;
    stage = 1;
    menu = 1;

    Interface::turno++;
    for (auto& e : imperio.territoriosConquistados) {
        if (e->getNome().find("Montanha") != std::string::npos) {
            e->atualizaRondas();
        }
    }
    if (Interface::turno == 7 && Interface::ano == 1) {
        Interface::turno = 1;
        Interface::ano = 2;
    }
    else
        if (Interface::ano == 2 && Interface::turno == 7) {
            Interface::turno = 13;
        }


    if (Interface::turno != 13) {
        cout << "********************" << endl
            << "--INICIO DO TURNO " << turno << " DO ANO " << ano << "--" << endl
            << "********************" << endl;
    }
}
```

Figura 9 – Função passa turno

Classe Save

A Classe Save é a classe responsável por guardar e carregar estados do jogo. Esta classe é constituída por um nome, uma cópia do Império e uma cópia do Mundo, assim como de algumas variáveis da interface.

```
class Save {
private:
    string nomeSave;
    Imperio imperioSave;
    Mundo backupMundo;

    int ano;
    int turno;
    int stage, menu,fase;

public:
    string getNomeSave();
    void saveMundo(Imperio& oldImperio,Mundo& mundo,int ano,int turno,int stage,int menu,int fase);
    void setNomeSave(string);
    void load(Imperio& IGimperio, Mundo& IGmundo, Interface& interface);
    void removeSave();
};
```

A lista de cópias é guardada na interface através de um vetor deste mesmo tipo.

Respostas a questões para o relatório

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.

A interface e o mundo são facilmente duas das principais classes da aplicação, sendo que ambos os seus objetos são criados no main, passados como argumento para a Interface poder funcionar e são destruídos quando a função run da interface termina, sendo que cada um dos elementos do mundo é destruído e de seguida o mundo em si. O objeto da interface e do mundo também são destruídos logo de seguida.

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

A responsabilidade de guardar os diferentes estados de jogo está claramente atribuída à classe Save, visto que é esta quem tem o conjunto das diferentes informações acerca do jogo. As suas funções saveMundo e load são às que aplicam esta responsabilidade dentro da aplicação.

5/6 - De entre as classes que fez, escolha duas e justifique por que considera que são classes com objetivo focado, coeso e sem dispersão. / Relativamente à aplicação entregue, quais as classes que considera com responsabilidades de interface com o utilizador e quais as que representam a lógica?

Ainda consideramos, assim como na 1ª meta, a classe Imperio como uma das mais importantes, pelas mesmas razões antes referidas; Pelo contrário, a classe Territorio perdeu alguma importância devido à sua responsabilidade ser distribuídas pelas diferentes classes de cada tipo.

Deste modo, em vez de Territorio, consideramos a classe **Interface** bastante mais importante, com um objetivo igualmente mais focado. Esta classe é a responsável por gerir o jogo e comunicar com as diferentes partes da aplicação para se desenvolver o jogo e progredir na aplicação.

Esta classe é também resposta à questão 6, visto ser esta a classe que gere a comunicação com o utilizador.

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

Em geral, o primeiro objeto a ser usado após comunicação com o utilizador tende a ser um Territorio após efetuar alguma ação, seja conquistar o mesmo, obter produtos e ouro dos territorios, etc.

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 interface não processa muita lógica por si, sendo que esta chama as funções necessárias às diferentes ações às respetivas classes: p.e, se quiser conquistar um territorio, a interface apenas vai procurar qual territorio e que se quer conquistar e chamar a função do territorio que trata dessa lógica para efetuar os seus passos.

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 Função getProdutos() é uma função abstrata da classe Territorio, sendo que cada tipo de territorio devolve uma quantidade diferente de produtos e também pode alterar conforme o estado do jogo (p.e Montanha e Castelo)

```
class Territorio {
protected:
    string nome;
    int resistencia=0;
    bool conquistado=0;
    int nRondasConquistado = 0;
public:
    virtual int getProdutos() =0;
```

```
int Montanha::getProdutos() {
    if (this->nRondasConquistado >= 2)
        return 1;
    else
        return 0;
}
```

```
int Castelo::getProdutos() {
    if (Interface::turno < 3)
        return 3;
    else
        return 0;
}
```

Figura 9 – Função getProdutos

10 – Apresente as principais classes da aplicação

As principais classes da nossa aplicação são a classe Mundo, Imperio, SaveLoad e Interface, sendo elas as principais responsáveis e detentoras das funções que regulam as ações e atividade do jogo.

A classe Mundo é a responsável pela criação de todos os territórios e criação de ponteiros para os mesmos, além da listagem.

A classe Império apresenta todas as funções que realizam ações no império do jogador, tanto o conquista, como o ganhar Tecnologia e as restantes que envolvem alterações na mesma.

A classe SaveLoad responsável pelos comandos grava e ativa. Tem como principal objetivo permitir ao jogador guardar estados de jogos seguros.

Por fim, a classe Interface descrita como a organizadora do jogo, responsável pela passagem das fases, turnos e anos.

Funcionalidades Implementadas

Componente do Trabalho	Realizado	Realizado Parcialmente	Não Realizado
Mundo	X		
Império do Jogador	X		
Territórios	X		
Tecnologias	X		
Criação do Mundo	X		
Realização do Jogo	X		
Fim do jogo	X		
Comandos e DEBUG	X		