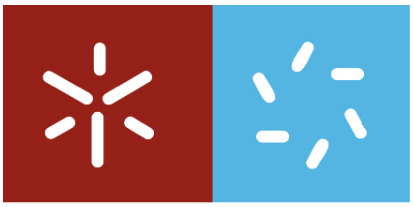

Universidade do Minho

Universidade do Minho

Licenciatura em Ciências da Computação

POO - Trabalho Prático

Grupo nº13

Ivo Miguel Gomes Lima
(A90214)

Miguel Ângelo Alves de Freitas
(A91635)

Tiago dos Santos Silva Peixoto Carriço
(A91695)

12 de junho de 2021

Conteúdo

Capítulo 1

Introdução e principais desafios

Este projeto consistiu no desenvolvimento de uma aplicação semelhante ao Football Manager na linguagem de programação Java, de forma a pôr em prática os conhecimentos adquiridos ao longo do semestre.

Consideramos que o maior desafio seja o impedimento da comunicação entre a simulação de jogo e a *view*.

Capítulo 2

Classes

2.1 Jogador

```
private String nome; // Nome do jogador
private String equipa; // Nome da equipa
private int id; // Identificador do jogador (valor unico)
private ArrayList<String> historico; // Historico de equipas
private Map<String,Integer> atributos; // Atributos do jogador
```

Começamos por fazer uma classe mais geral, onde temos os identificadores gerais representados a cima. Com isto podemos proceder à criação de um jogador mais genérico onde identificamos os jogadores, os seus atributos, nomes, equipas a que pertence, número da camisola e o seu histórico.

2.2 Guarda-Redes

```
private int elasticidade;
```

"Guarda-Redes" é uma classe mais específica da classe "Jogador", ou seja, é uma subclasse onde vamos atribuir peso específico como o acima assinalado, pois no nosso ponto de vista é uma especialização importante dos Guarda-Redes num jogo de futebol.

2.3 Lateral

```
private int cruzamento;
private int drible;
```

"Lateral" é uma classe mais específica da classe "Jogador", ou seja, é uma subclasse onde vamos atribuir pesos específicos como os acima assinalados, pois no nosso ponto de vista são as especializações dos laterais num jogo de futebol.

2.4 Defesa

```
private int corte;
private int intersecao;
```

"Defesa" é uma classe mais específica da classe "Jogador", ou seja, é uma subclasse onde vamos atribuir pesos específicos como os acima assinalados, pois no nosso ponto de vista são as especializações dos defesas num jogo de futebol.

2.5 Médio

```
private int intersecao;
private int visao;
```

"Médio" é uma classe mais específica da classe "Jogador", ou seja, é uma subclasse onde vamos atribuir pesos específicos como os acima assinalados, pois no nosso ponto de vista são as especializações mais imporantes dos médios num jogo de futebol.

2.6 Avançado

```
private int finalizacao;
private int compostura;
```

"Avançado" é uma classe mais específica da classe "Jogador", ou seja, é uma subclasse onde vamos atribuir pesos especificos como os acima assinalados, pois no nosso ponto de vista são as especializações dos avançados num jogo de futebol.

2.7 Equipa

```
private String nome;
private Map<Integer,Jogador> jogadores;
private int[] titulares;
```

"Equipa" é uma classe abrangente onde vamos adicionar os jogadores, removê-los, definir se o jogador é titular, calcular o overall da equipa e/ou dos jogadores titulares. Também nesta classe temos métodos para conseguirmos obter a equipa completa e mudar conforme as devidas substituições em jogo. Podemos modular as equipas fazendo tranferências de jogadores e possibilitando ainda a chamada da classe *parse* para a leitura do ficheiro.

2.8 Jogo

```
private String equipaCasa;
private String equipaFora;
private int golosCasa;
private int golosFora;
private LocalDate date;
private List<Integer> jogadoresCasa;
private List<Integer> jogadoresFora;
private Map<Integer, Integer> substituicoesCasa; // sai -> entra
private Map<Integer, Integer> substituicoesFora; // sai -> entra
```

Nesta classe são guardados todas as informações de um jogo, nomeadamente os nomes dos jogadores, nomes das equipas, número de golos, a data do evento, jogadores titulares assim como as substituições feitas.

2.9 Parser

"Parser" é uma classe feita pela equipa de docente da disciplina, mas tendo algumas alterações da nossa parte para adaptar ao nosso código. Esta classe vai permitir ler todo o texto de um ficheiro, organizando as equipas, as posições e os jogos já realizados.

2.10 SaveGame

"SaveGame" é uma classe onde enviamos todas as informações relevantes dos jogos para serem guardadas, ou seja, vamos guardar o estado atual das equipas, e todos realizados para que possam ser posteriormente guardados num ficheiro .txt que será interpretado pela classe *parser*.

2.11 SimulacaoJogo

```
private static final int CASA = -1;
private static final int MEIO = 0;
private static final int FORA = 1;

private Jogo jogo;
private Equipa equipaCasa;
private Equipa equipaFora;
private int overallCasa;
private int overallFora;
private int posseBola;
private final int posseIntervalo;
private int posicaoBola;
private int substituicoesCasa;
private int substituicoesFora;
```

Esta classe vai representar a simulação de um jogo de futebol real entre duas equipas, a de casa e a de fora, necessitando das informações dos constituintes de ambas as equipas assim como os seus *overalls*. O algoritmo que utilizamos para esta simulação baseia-se em:

- O campo está divido em 3 partes(área da equipa de casa, meio campo, área da equipa de fora).
- Em cada jogada, a equipa atacante pode suceder e avançar para o terço seguinte, se estiver no último terço marca golo, em caso de insucesso a posse de bola muda para a equipa adversária e dá-se um contra-ataque.
- Ao longo das jogadas, pode dar-se um momento de substituição, onde o utilizador fará a escolha.

Este algoritmo é efetuado duas vezes, pois existe divisão de partes.

2.12 View

```
private Controller controller;
private Scanner input;
```

A *view* é a Vista do programa. Comunica apenas com o Controller. Possui vários métodos, pricipalmente métodos para mostrar algum tipo de informação ao utilizador ou mensagens de erro. Mantendo a identididade da arquitetura (Model-View-Controller).

2.13 Menu

O jogo FM incializa com este menu e uma mensagem de texto. Possui as seguintes opções:

"0.Sair" - Sair do Jogo FM

"1.Carregar ficheiro" - Carrega uma lista de equipas e jogos através um ficheiro .txt, seguindo a estrutura é a indicada pela equipa docente

"2.Gravar estado em ficheiro" - Grava o estado atual do jogo e tudo que nele se encontra num ficheiro .txt.

"3.Carregar ficheiro de objetos" - Faz o mesmo que o anterior mas carrega em ficheiro objeto em vez de .txt.

"4.Gravar estado em ficheiro de objetos" - Faz o mesmo que o anterior mas grava em ficheiro objeto em vez de .txt.

"5.Criar Equipa" - Permite ao utilizador criar uma nova equipa vazia para que possam ser adicionados jogadores.

"6.Adicionar Jogador" - Permite ao utilizador criar um jogador onde e adicionar a uma equipa à sua escolha. Caso não existam equipas só é possível adicionar à equipa "jogadores sem equipa".

"7.Transferir Jogador" - Permite ao utilizador transferir qualquer jogador entre equipas guardando no histórico do jogador essa transferência.

"8.Remover Equipa" - Permite ao utilizador remover uma equipa e, por defeito, todos esses jogadores são atribuidos à equipa "jogadores sem equipa".

"9.Remover Jogador" - Permite ao utilizador remover definitivamente um jogador de qualquer equipa.

"10.Simular Jogo" - Permite a realização de um jogo entre duas equipas definidas pelo utilizador.

"11.Informação" - Nesta opção podemos encontrar todas as informações extra-Jogo, ou seja, consultar mais dados referente à equipa e informações referentes aos jogadores.

2.14 Controller

```
private static final int CASA = -1;
private static final int FORA = 1;
private Map<String,Equipa> equipas;
private List<Jogo> jogos;
private SimulacaoJogo simulacao;
```

Esta classe, juntamente com outras, direciona o fluxo da aplicação mapeando e direcionado as ações recebida (request) pela camada da apresentação para os respectivos serviços da aplicação. Esta classe vai ser responsável pela gestão do menu inicial, além de interagir com outras classes controladores que gerem os menus da view, fazendo todo o trabalho "lógico"da view.

2.15 Interface

Esta classe é a classe principal que inicializa as componentes necessárias para o funcionamento do Modelo MVC.

Capítulo 3

Exceptions

3.1 EquipaNaoDefinidaException

Esta exception foi criada no âmbito de quando a equipa não é definida ou quando é definida incorretamente, alertando o utilizador para este problema.

3.2 EquipaJaExisteException

Esta exception foi criada no âmbito de quando a equipa já está definida e impede a sobreposição de comandos sobre a mesma, alertando o utilizador para este problema.

3.3 NumeroInvalidoException

Esta exception foi criada no âmbito de quando um dos jogadores não é definido ou quando é definido incorretamente, alertando o utilizador para este problema.

3.4 JogadorNaoExisteException

Esta exception foi criada no âmbito de quando um dos jogadores não está definido ou foi definido de forma incorretamente, alertando o utilizador para este problema.

3.5 JogadorJaExisteException

Esta exception foi criada no âmbito de quando um dos jogadores é definido e já existe na equipa, alertando o utilizador para este problema.

3.6 LinhaIncorretaException

Esta exception foi criada no âmbito de quando uma das linhas está de forma incorreta, alertando o utilizador para este problema.

3.7 SubstituicaoErradaException

Esta exception foi criada no âmbito de quando na fase do jogo a substituição não é feita corretamente, alertando o utilizador para este problema.

3.8 TitularesNaoDefinidosException

Esta exception foi criada no âmbito de quando os titulares não foram definidos corretamente, na inicialização do jogo ou durante, alertando o utilizador para este problema.

3.9 SimulaçãoImpossivelException

Esta exception foi criada no âmbito de caso exista algum erro numa definição ou alguma atribuição do utilizador a simulação não se realizar, alertando o utilizador para este problema.

Capítulo 4

Estrutura do projeto

O nosso projeto segue a estrutura *Model View Controller* (MVC), estando por isso organizado em três camadas:

- A camada de dados (o modelo) é composta pelas Classes Jogador, Guarda-Redes, Lateral, Defesa, Médio, Avançado, Equipa, Jogo e Simulacao pelas interfaces.
- A camada de interação com o utilizador (a vista, ou apresentação) é composta unicamente pela classe View.
- A camada de controlo do fluxo do programa (o controlador) é composta pela classe Controller.

Como foi referido anteriormente, todo o projeto baseia-se na ideia de encapsulamento.

Capítulo 5

Diagrama de Classes

Figura 5.1: Diagrama de classes do programa, gerado pelo *IntelliJ*

Capítulo 6

Conclusão

A nível geral, e tendo em conta o que foi explicado nos capítulos anteriores, como grupo achamos que todos os objetivos foram cumpridos e apesar das dificuldades que fomos encontrando o grupo conseguiu superar de uma forma muito boa, sempre com um olhar crítico e a pensar no próximo passo. Acreditamos que respondemos de forma correta ao problema apresentado pela equipa docente da disciplina.