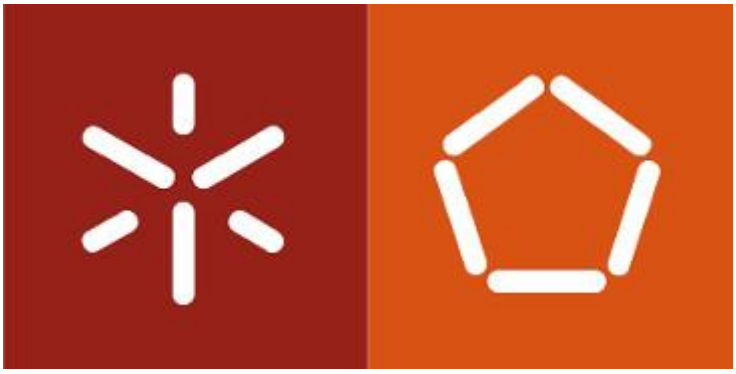

Universidade do Minho

Mestrado Integrado em Engenharia Informática

POO – TRABALHO PRÁTICO

GRUPO 100

Lucas Verdelho (A93247) Nelson Ribeiro (A93188)
Tiago Sousa (A93248)

Conteúdo

1 Introdução e principais desafios

Este projeto consistiu no desenvolvimento de um programa semelhante ao conhecido jogo Football Manager, de forma a pôr em prática os conhecimentos adquiridos ao longo do semestre.

Consideramos que o maior desafio tenha sido efetuar a simulação de jogo, uma vez que o futebol é um jogo complexo em termos de estatísticas, táticas, posicionamento dos jogadores, etc, criando assim um maior desafio na realização deste projeto. O último desafio com que nos deparamos foi a realização de faltas, atribuindo cartões amarelos e vermelhos aos jogadores.

2 Classes

2.1 Jogador

```
private String nome;
private int numero;
private int velocidade;
private int resistencia;
private int destreza;
private int impulsao;
private int jogo_de_cabeca;
private int remate;
private int passe;
private List<String> historico;
```

A classe Jogador possui todas as características bases de um jogador de futebol juntamente com o histórico de equipas por onde um jogador já passou.

Trata-se de uma **Classe Abstrata** que depois é expandida para os diversos tipos de jogadores (GuardaRedes, Defesa, Lateral, Médio, Avançado).

Contém o método abstrato Overall, que é posteriormente desenvolvido nas Subclasses que implementam a Classe Jogador, uma vez que os diversos tipos de Jogadores têm atributos diferentes para o cálculo do seu Overall.

```
public abstract int overall();
```

2.2 GuardaRedes

```
private int elasticidade;
```

Classe que estende a Classe Jogador, contendo uma nova característica, a *elasticidade*. O método Overall atribui pesos diferentes às diferentes características, dando mais importância à *elasticidade* e de seguida à *destreza* e *impulsão*.

2.3 Defesa

Classe que estende Classe Jogador. O método Overall dá prioridade ao *jogo de cabeça* e atribui pesos de forma mais uniforme às restantes características.

2.4 Lateral

```
private int cruzamento;
```

Classe que estende a Classe Jogador, contendo uma nova característica, o *cruzamento*. O método Overall atribui pesos diferentes às diferentes características, dando mais importância ao *cruzamento* e *resistência*.

2.5 Médio

```
private int recuperacao_bola;
```

Classe que estende a Classe Jogador, contendo uma nova característica, a *recuperação de bola*. O método Overall atribui pesos diferentes às diferentes características, dando mais importância ao *passe* e *recuperação de bola*.

2.6 Avançado

Classe que estende a Classe Jogador. O método Overall atribui pesos diferentes às diferentes características, dando mais importância ao *remate*.

2.7 Equipa

```
private String nomeE;
private Map<Integer, Jogador> titulares;
private Map<Integer, Jogador> suplentes;
```

Classe responsável por formar uma equipa de futebol. Para tal, escolhemos a estrutura Map, criando um Map de jogadores titulares e um Map de jogadores suplentes.

É nesta Classe que determinamos a equipa titular a ser escolhida para um Jogo (*colocaTitulares (int Tatica)*). O algoritmo escolhido foi adicionar todos os jogadores ao Map de suplentes, determinar uma lista ordenada decrescentemente pelo seu Overall e, se seguida, de acordo com a tática utilizada, retirar os melhores jogadores para as suas respetivas posições.

2.8 Golo

```
private int minuto;
private String equipa;
private String jogador;
```

Classe que guarda a informação de um golo ocorrido num Jogo. Guarda o minuto em que o Jogador marcou, o seu nome e a sua Equipa.

2.9 Substituição

```
private int jogadorSai;
private int jogadorEntra;
```

Classe que guarda a informação duma substituição ocorrida num Jogo. Guarda o número do Jogador que saí e o número do Jogador que entra.

2.10 Jogo

```
private Equipa equipa1;
private Equipa equipa2;
private LocalDate data;

private int golos1;
private int golos2;
private int faltas1;
private int faltas2;
private int posse_de_bola1;
private int posse_de_bola2;
private int remates1;
private int remates2;
private List<Golo> golos;
private List<Integer> amarelos1;
private List<Integer> amarelos2;
private List<Integer> vermelhos1;
private List<Integer> vermelhos2;
private List<Substituicao> substituicoes1;
private List<Substituicao> substituicoes2;
```

Classe na qual é realizada a simulação do jogo e que guarda várias estatísticas em relação ao jogo principalmente os golos, os remates, as faltas e as substituições.

Na fase inicial da simulação de jogo é escolhida uma tática aleatória para cada equipa. Para simplificar a simulação e é feito previamente um cálculo do **overall** de cada equipa em cada posição e de seguida guardamos também o número de jogadores em cada posição tendo em conta a tática escolhida. Para eventuais expulsões dos jogadores resultantes de faltas criamos um **ArrayList** que guardava temporariamente os jogadores.

O corpo principal do método **simularJogo** consiste num ciclo que tem a duração da partida, calculada por uma variável chamada tempo que íamos atualizando ao fim de cada simulação. Antes de iniciar qualquer jogada era feito um cálculo de uma chance de uma das equipas ter apanhado falta e caso isso aconteça é determinado qual o jogador que apanhou a falta dependendo da posição dele, da zona do campo em que se encontrava a bola e da equipa que estava atualmente a atacar.

Quando um jogador recebe um cartão amarelo era simplesmente acionado ao **ArrayList** dos amarelos da equipa correspondente e, caso já estivesse presente o número do jogador nessa lista, era acionado aos vermelhos e expulso do jogo sendo removido dos titulares.

Para a simulação das jogadas dividimos o campo em zonas em que cada equipa era responsável pelo ataque ou pela defesa, sendo o sucesso de cada uma destas tentativas de ataque calculada num método auxiliar. Para simular a tentativa de golo optámos por escolher um jogador aleatório, entre aqueles que podiam estar envolvidos no ataque, tendo em conta a sua **capacidade de remate** e de seguida comparávamos com o **overall** do guarda-redes e era dada uma probabilidade ao golo de ocorrer.

Para realizar substituições tivemos várias ideias, contudo devido à nossa gestão de tempo tivemos que simplificar, optando por substituir um jogador de cada equipa ao fim da primeira parte e aos 65 minutos substituir outros 2 jogadores de cada equipa sendo este adicionados à lista de substituições das equipas correspondentes.

2.11 Manager

```
private Map<String, Equipa> equipas;
private List<Jogo> jogos;
```

Classe responsável por agregar todas as equipas e jogos. É composta por um Map que usa como key o nome da equipa e como value a respetiva equipa, e por uma lista de Jogo, que são todos os jogos executados.

Esta classe é também responsável pela leitura dos logs, escrita e leitura de ficheiros objeto.

3 Estrutura do Projeto

Para este projeto, baseamo-nos na ideia do encapsulamento, escolhendo assim a estratégia de composição.

4 Conclusão

Em suma, acreditamos que temos um projeto bem estruturado, tentamos ter sempre em conta o princípio do encapsulamento e implementar mais funcionalidades para tornar o jogo mais realista. Porém, havia ainda mais situações de jogo que tínhamos em mente para implementar neste projeto, como por exemplo, foras, cantos, cansaço dos jogadores e melhorar o nosso sistema de faltas, mas, por questões de tempo, não foi possível.

Trata-se dum projeto que, num futuro próximo, com tempo disponível, será devidamente terminado, uma vez que gostamos bastante do tema em questão e houve sempre ânimo na realização do mesmo.

5 Diagrama de Classes

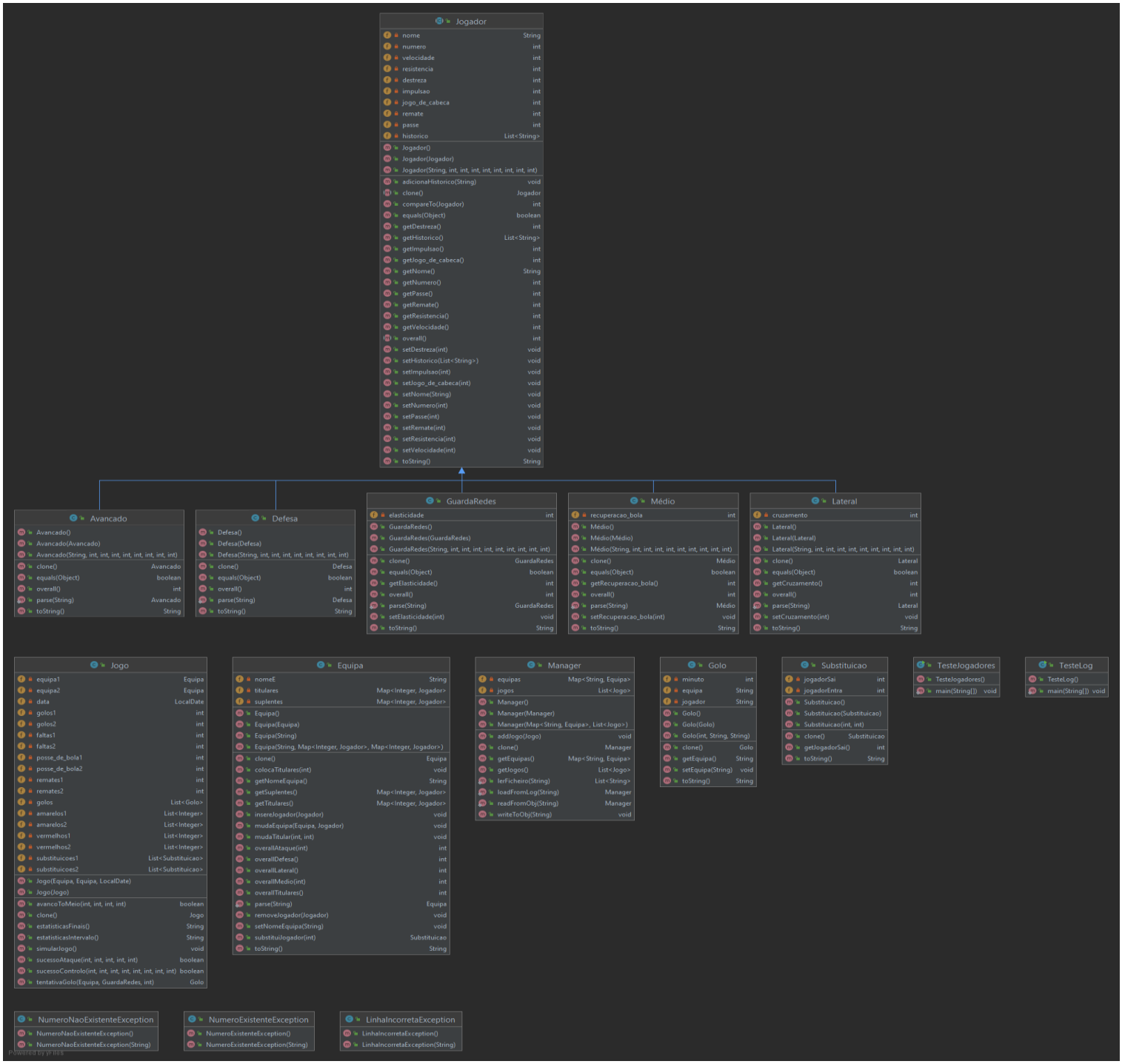

Diagrama de Classes gerado pelo *IntelliJ*.