Universidade do Minho

Programação Orientada aos Objetos
Relatório do Trabalho Prático

José Gonçalves - a93204

Sofia Gomes - a93314

Miguel Santa Cruz - a93194

Universidade do Minho

Junho 2021

Índice

1. Introdução

Este projeto surgiu no âmbito da cadeira de Programação Orientada aos Objetos, lecionada no 2º semestre do 2º ano do Mestrado Integrado em Engenharia Informática, e tem como objetivo consolidar os conhecimentos adquiridos neste paradigma de programação, assim como os conhecimentos da linguagem Java.

O seu objetivo passa por criar um programa com base no famoso jogo Football Manager, assim como um sistema de gestão e simulação de equipas, contando com várias equipas, em que cada equipa tem jogadores titulares e suplentes. A nossa implementação permite a cada jogador uma posição (avançado, defesa, guarda-redes, etc...) e uma classificação de habilidade calculada com base nos aspetos mais importantes de um jogador para a sua posição.

Para cumprir o objetivo proposto, cumprindo as estratégias de encapsulamento, abstração e capacidade de evolução controlada, foram criadas várias classes, pelo que passamos a introduzir cada uma delas, assim como referir a função de cada uma e as estratégias utilizadas.

2. Descrição da Arquitetura

2.1. Atleta

Esta foi a primeira classe a ser desenvolvida, e é a mais geral de todas, sendo que várias outras descendem desta.
Esta classe conta com as seguintes variáveis de instância:

- ***idAtleta***, que tal como o próprio nome diz, corresponde a um id único para cada atleta;
- ***nomeAtleta***, que corresponde ao nome do atleta em questão;
- ***idade***, que é a idade do atleta.

Relativamente aos métodos implementados nesta classe, apenas foram necessários os habituais, tais como o clone, toString, getters & setters e equals.

2.2. Jogador

Na implementação desta classe, optamos por utilizar o fenómeno de **herança** de Java, sendo esta subclasse de Atleta, o que não só nos permite contemplar o encapsulamento, assim como a capacidade de evolução controlada do projeto (por exemplo, se quiséssemos alterar para outro desporto, um nadador seria também um atleta).

O Jogador tem 7 características base, sendo estas a *velocidade*, *resistencia*, *destreza*, *impulsao*, *jogo_de_cabeça*, *remate* e *capacidade_de_passe*.
Para além destas variáveis de instância (que os próprios nomes explicam o que faz cada uma), a classe conta também com as seguintes:

- ***habilidade*** - a classificação do jogador;
- ***numeroJogador*** - o número da camisola do jogador;
- ***equipas*** - uma lista de todas as equipas;
- ***posicao*** - um inteiro para cada posição, em que 0 corresponde ao guarda-redes, 1 ao defesa, 2 ao lateral, 3 ao médio e 4 ao avançado.
- ***titular*** - booleano que dita se um jogador foi colocado a titular ou não;
- ***suplente*** - booleano que dita se um jogador está no banco ou se está a jogar.

Uma vez mais, os métodos implementados foram apenas os habituais, como *clone*, *toString*, *equals* e *getters & setters*.

2.3. Avançado, Defesa, Lateral e Guarda-redes

Estas três classes são bastante equivalentes, sendo que todas são subclasse de Jogador (herdam, portanto, as caracteristicas lá descritas), mas apenas diferem nas variáveis de instância (cada classe tem como variáveis de instância as características que mais se adequam a cada posição), sendo que os seus nomes são bastante "*self explanatory*".

Para cada uma das classes foi também desenvolvido um método que calcula a habilidade do *Jogador* com base numa avaliação por pesos das características mais importantes para essa posição(valorizando umas em prol de outras).

Em seguida, encontram-se as assinaturas das classes e as suas variáveis de instância.

```
public class Lateral extends Jogador{
    private int cruzamentos;
```

Figura 1. Assinatura da classe Lateral

```
public class Guarda_Redes extends Jogador {
    private int elasticidade;
```

Figura 2. Assinatura da classe Lateral

```
public class Defesa extends Jogador {
    private int intersecao;
    private int drible;
```

Figura 3. Assinatura da classe Defesa

```
public class Avancado extends Jogador {
    private int finalizacao;
    private int sprint;
```

Figura 4. Assinatura da classe Avançado

```
public class Medio extends Jogador{
    private int recuperacao_bolas;
```

Figura 5. Assinatura da classe Médio

2.4. Equipa

Esta classe forma uma equipa, sendo que contém todas as informações relativas da mesma como variáveis de instância:

- ***plantel*** - *HashMap* que tem todos os jogadores e que tem como key o número da camisola do jogador e como value o próprio Jogador;
- ***nome*** - O nome da equipa;
- ***dataDeFundação*** - A data de fundação da equipa, em formato *LocalDate*;
- ***jogosAgendados*** - A lista dos jogos agendados para a equipa;
- ***habilidadeGlobal*** - A média da habilidade dos jogadores que estão em jogo;

Esta classe tem os habituais métodos "básicos" (*clone*, *toString*, etc...), um método para calcular a habilidade do plantel que está em jogo, métodos para adicionar e remover um jogador da equipa e um parser que recebe as informações de uma equipa no formato de *String* e converte para uma *Equipa* (através do construtor parametrizado).

2.5. Jogo

A classe Jogo está relacionada (como o próprio nome diz) com tudo o que acontece durante um jogo, quer seja a nível de jogadores, golos ou controlo das substituições.
Deste modo, passamos a apresentar sucintamente as variáveis de instância, sendo de notar que os que é feito para a equipa da casa (tem "casa" no nome), é feito de igual forma para a visitante:

- ***golosCasa*** - nº de golos da equipa da casa;
- ***date*** - data do jogo;
- ***eqCasa*** - Equipa da casa;
- ***titularesCasa*** - Lista com os jogadores titulares da equipa da casa;
- ***emJogoCasa*** - Lista dos jogadores da equipa da casa que estão em jogo.
- ***entraSaiCasa*** - Map das substituições que tem como key o nº da camisola do jogador que entra e value o nº do jogador que sai;
- ***posicaoBola*** - inteiro correspondente à posição da bola no determinado instante do jogo, podendo ser 0 caso esteja no meio-campo, 1 na baliza da equipa da casa (posse do guarda-redes), 2 na baliza da equipa visitante (posse do guarda-redes), 3 na área da equipa da casa, 4 na área da equipa visitante, 5 é canto para a equipa da casa, 6 é canto para a equipa visitante, 7 é golo para a equipa da casa e 8 é golo para a equipa visitante.

Para esta classe, mais uma vez, foram apenas definidos os métodos habituais.

2.6. Parser

Esta classe é a responsável pela leitura dos ficheiros. Para tal, tem como variável de instância o caminho para o ficheiro (*path*).

Para além dos métodos habituais, implementamos um método que analisa a posição do jogador e o classifica de acordo com a sua posição, construindo-o com todos os seus atributos (*parse*).

Também foram implementadas as funcionalidades de ler e escrever em binário (*readBin* e *guardaBin*) - recorrendo a *FileIntputStream/FileOutputStream* - assim como um método que lê para uma lista todas as linhas do ficheiro de entrada.

2.7. GameResult

Esta classe contém o método que permite inferir o resultado de um jogo. Para tal, o método *calculaJogada* recorre a números aleatórios que são utilizados com um "fator de sorte" para poder decidir a próxima posição da bola (dentro das possibilidades definidas na classe Jogo) e aos seus métodos auxiliares (*bolaCanto*, *bolaArea*, *bolaBaliza* e *bolaMeioCampo*).
Deste modo, através da posição atual da bola, deliberamos a probabilidade de a bola ir para cada uma das possíveis posições e, consoante o número aleatório, é escolhida a próxima posição da bola.
A probabilidade de uma equipa efetuar uma ação é calculada tendo em conta a diferença de habilidade global das equipas.
Assim, duas equipas com a mesma habilidade global tem 50% de hipóteses de ganhar um jogo. Caso a diferença entre as equipas seja máxima (ou seja uma equipa tem habilidade zero e outra cem) então a equipa com maior habilidade global ganha 100% das vezes.

2.8. Interpretador

O interpretador é responsável por tratar toda a informação dos pedidos do utilizador, operando todas as funções do jogo e fazendo com que estas executem.
A classe *Interpretador* atua como controlador de todo o programa.
Nesta classe estão presentes os métodos para transferir jogador entre equipas, procurar um jogador na lista das equipas, procurar uma equipa, adicionar jogadores e jogar.

2.9. Input

Uma vez que o programa trabalha com I/O para escolher equipas, jogadores, entre outras opções, a classe Input trabalha com a classe Scanner do Java, para que apenas esteja aberto um *Scanner* e o seu uso seja facilitado.
Esta classe faz a verificação do tipo do input, pedindo repetidamente input ao utilizador até que este introduza um valor válido.

2.10. Model

Esta classe tem como função receber a informação das equipas e jogadores e tratar toda a informação recebida.
O *Model* faz parte da arquitetura *MVC*, sendo responsável por guardar a equipa (de modo implícito, os jogadores também) e jogos, ou seja, tudo o que é lido e trabalhado no programa.

2.11. View

A view, que também pertence à arquitetura *MVC*, trabalha em paralelo com o interpretador, sendo que trata de mostrar todas as mensagens dos comandos do nosso programa no ecrã.

2.12. Main

Esta classe é a primeira a ser chamada, sendo que na execução do programa é a primeira a executar e a mandar o interpretador correr

2.13. Menu

Esta classe apresenta a lista das opções de cada vez que é escolhida uma opção e aparece um novo menu para as seguintes. Para tal, tem como variáveis de instância o título do menu e a lista das opções do menu.
Nesta classe, para além dos métodos habituais (construtores, *getters e setters*, etc...) temos também um método que apresenta o menu, bem como as suas opções.

2.14. Exceptions & Interfaces

Ao longo do desenvolvimento deste projeto, recorremos ao uso de exceptions e interfaces como meio de atingir os objetivos propostos, assim como permitir uma maior facilidade a nível de tratamento de erros.
A utilização de interfaces por todo o código permite que seja possível no futuro implementar de outras maneiras as classes do projeto com mudanças mínimas no código.

3. Funcionalidades do Programa

Ao inicializar o programa é apresentado um menu de 7 opções, as quais se apresentam em seguida:

1. **Consultar equipas:** Ao digitar o número 1 no terminal, passamos a consultar a lista das equipas, na qual podemos consultar as informações da totalidade das equipas ou pesquisar por uma determinada equipa.
2. **Consultar jogadores:** De forma análoga às equipas, podemos consultar a lista dos jogadores através da pesquisa da totalidade dos jogadores de uma equipa ou as estatísticas de um único jogador (exibidas com maior detalhe).
3. **Consultar jogos:** Ao selecionar esta função, é-nos permitido consultar o histórico de jogos, contendo as equipas, resultados e data do jogo.
4. **Adicionar Equipa:** Esta opção permite adicionar uma nova equipa ao jogo.
5. **Adicionar jogador:** Tal como a anterior, esta opção permite adicionar um novo jogador à base de dados do programa.
6. **Transferir Jogador:** Esta opção traz a possibilidade de trocar um jogador de equipa, quer seja para equilibrar as equipas ou por preferência.
7. **Jogar:** Nesta opção temos o cerne do programa, permitindo escolher as equipas e, de seguida, simular um jogo entre elas.
8. **Carregar estado:** Ao escolher esta opção, temos a possibilidade de carregar um estado anterior do nosso jogo, para que possamos continuar com essa sequência.
9. **Guardar estado:** Para que a opção anterior seja possível, temos que, em primeiro lugar, guardar o nosso progresso no jogo. Para tal, utilizamos esta opção que nos permite guardar o estado atual do nosso jogo.
10. **Remover jogador:** Dado um nome de um jogador, esta opção permite apagar os seus dados do jogo.
11. **Remover equipa:** Tal como a opção anterior, dado o nome de uma equipa, esta opção permite apagá-la do jogo.
12. **Sair:** Permite ao utilizador sair do programa.

Figura 6. Menu principal de interação com o programa

4. Conclusão

Em suma, pensa-se que este projeto cumpre com todos os objetivos propostos, desde o uso dos conceitos de encapsulamento, abstração e capacidade de evolução controlada, até a apresentação de um programa que cumpre as funções propostas de um modo eficaz e agradável.

Uma das principais dificuldades no desenvolvimento deste programa foi ao nível do algoritmo que decide os eventos de cada jogo (golos e posicionamento da bola), sendo que se investiu bastante tempo nesta parte, mas, no final, pensa-se que se obteve um resultado bastante positivo e, de certa forma, assertivo.