Universidade do Minho
Escola de Engenharia

Relatório SO - MIEI 2020/21

Grupo 42

Junho 2021

(a) João Pedro Fontes Delgado a93240

(b) Bruno Filipe Miranda Pereira a93298

(c) Francisco José Martinho Toldy a93226

Figure 1: Autores do trabalho

Contents

1 Introdução

Este projeto foi desenvolvido no âmbito da UC de **Programação Orientada a Objetos**. Este tinha como finalidade criar um programa de gestão de equipas e simulação de jogos de futebol em Java.

Desenvolvimento do software/abordagem do problema:

O projeto foi realizado em fases. Para além das duas fases de entrega definidas pelos docentes da UC, o grupo também teve em conta os 3 patamares de classificação definidos no enunciado e planeou o trabalho de acordo com isso. Foi decidido pelo grupo juntar o patamar 16 e 20 num só uma vez que métodos do patamar 16 teriam que ser refeitos parcialmente para poder enquadrar com os planos para o patamar 20. Começamos por definir que os dados dos Jogadores seriam guardados numa classe abstracta ***Jogador*** e nas suas respectivas subclasses (referentes a cada posição). Esses Jogadores seriam guardados numa ***Equipa***, e as Equipas estariam agrupadas numa ***Liga***.

Usámos uma arquitetura **Model View Controller** em que

- **View**: *FMView*
- **Model**: *Liga, Equipa, Jogador*
- **Controller**: *FMApp*

2 Classes

Nesta secção iremos abordar as principais classes referentes a entidades do projeto

2.1 Jogador

Classe abstrata usada para representar um jogador, contendo a informação comum a cada jogador independentemente da posição a que joga, ou seja:

- **private String** *nome*
- **private int** *numero*
- Estatísticas (velocidade, resistencia, destreza, impulsao, jogoDeCabeca, remate, passe, habilidade) no tipo **int**
- **private String** *posicao*
- **private List** *historial*

Além dos métodos adequados à classe (getters, setters, equals e toString), a classe possui também o método addHistorial para adicionar o nome de uma equipa à List historial.

Esta classe é estendida pelas Classes representativas de cada posição (**Avancado Defesa GuardaRedes Lateral e Medio**).

2.1.1 Subclasses da Classe Jogador

Estendem a definição de Jogador , contendo assim as variáveis todas mencionadas acima e além disso característica específica à posição a que a subclasse se refere:

- **private int** *elasticidade* no caso dos guarda-redes
- **private int** *cruzamento* no caso do lateral
- **private int** *recBola* no caso do médio
- **private int** *forca* no caso do defesa
- **private int** *agilidade* e
- **int** *finalizacao* no caso do avançao

Cada subclasse define o método de parse e também o de cálculo de habilidade usando uma fórmula que atribui importância diferente a cada atributo consoante a posição do jogador (o *jogoDeCabeca* tem um peso diferente no *guardaRedes* que no *avancado*).

2.2 Equipa

Usada para agrupar jogadores pertencentes à mesma equipa para além de definir os métodos de manipulação de equipas. Cada Equipa guarda o seu nome; guarda também um Map de todos os jogadores em que a Key é o seu número e um Map dos jogadores titulares, armazenados em listas relativas às suas posições.

A decisão sobre que estruturas de dados utilizar para guardar esta informação foi tomada e alterada com o desenvolver do projeto à medida que o grupo se apercebeu que certos métodos iriam requerer as mesmas.

Esta classe tem as variáveis de instância

- **private String** *nome*
- **private Map <Integer, Jogador>** *jogadores*
- **private Map <String , List<Jogador>>** *titulares*
- **private int** *habilidade*

Esta classe define também o método de inserção de um Jogador numa equipa *(insereJogador)* e o método de inserção de um jogador como resultado de transferência *(insereJogadorTransferência)* bem como métodos de procura *(procuraJogador)* e remoção *(removeJogador)*

2.3 Liga

Usada para agrupar e contar as equipas e armazenar os jogos bem como definir métodos relativos às ações entre equipas.

À semelhança do sucedido na criação da classe Equipa, também foi escolhido um Map para armazenar as equipas, usando como Key o nome de cada uma. Por outro lado, os jogos são armazenados numa List de objetos da classe Jogos (próxima secção). Assim, as variáveis de instância são :

- **private Map <String,Equipa>** *equipas*;
- **private List <Jogo>** *jogos*;
- **private int** *n_equipas*;

A classe define também os métodos para criação de equipas *(criarEquipa)* , adicionar um jogador a uma equipa *(addJogador)* , adicionar uma equipa ao Map de equipas *(adicionaEquipa)*, bem como os métodos de transferência e consulta de jogador *(transferencia* e *consultaJogador*, respectivamente)

2.4 Jogo

A classe mais complexa de todo o projeto, quer em número de variáveis de instância, quer em número e complexidade de métodos definidos para a mesma.

Esta classe é responsável por toda a informação relativa aos jogos e à sua realização. Possui v.i.'s para os nomes das Equipas em jogo, os seus respetivos golos, a data, a lista de jogadores em campo para ambas as equipas e Maps para armazenar as substituições de cada equipa e, por último, uma String bola (mais sobre isso nas secções seguintes) . Assim as variáveis são:

- **private String** *nomeEquipaCasa*;
- **private String** *nomeEquipaFora*;
- **private int** *golosCasa*;
- **private int** *golosFora*;
- **private LocalDate** *date*;
- **private List<Integer>** *jogadoresCasa*;
- **private List<Integer>** *jogadoresFora*;
- **private Equipa** *equipaCasa*;
- **private Equipa** *equipaFora*;
- **Map <Integer, Integer>** *substituicoesCasa*;
- **Map <Integer, Integer>** *substituicoesFora*;
- **private String** *bola*;

A classe é completa com métodos relativos ao funcionamento de um Jogo (ver secção 6) bem como os habituais getters, setters, clone e a função de parse.

Nota sobre o Parser A classe Parser foi alterada o mínimo possível, aproveitando ao máximo o código fornecido pelos docentes, sendo assim, não terá a sua subsecção dedicada.

3 FMView e FMApp

3.1 FMView

Usada apenas para realizar operações de IO com o utilizador , não conhece o modelo de dados nem sabe como está implementado , tem apenas definidos métodos que recebem input do utilizador ou que imprimem output dos pedidos do utilizador no ecrã. Tudo isso é controlado pelo FMApp , quando o utilizador pretende executar algo , como por exemplo consultar um jogador, o FMApp vai executar um método da View para recolher o nome do jogador a consultar e em seguida passa esse nome de jogador para outros métodos do modelo de dados , funcionando assim como a ponte entre o modelo de dados e a view .

3.2 FMApp

Classe responsável por executar o programa, funciona como um controller da aplicação , gerindo aquilo que o utilizador pede e o que é mostrado no ecrã . Além disso faz a comunicação entre a view e o modelo de dados do programa.

4 Excepções e Comparações

- **CamposInvalidos** -
- **EquipaNaoExisteException** - para quando o nome de uma equipa usado como argumento não existe no Map de equipas
- **InsufficientPlayers** - para quando a tática fornecida pelo utilizador não prefaz um total de 10 jogadores (o guarda-redes não é considerado).
- **JogadorNaoExiste** - para quando o nome de um jogador usado como argumento não existe no Map de jogadores numa Equipa
- **NumeroNaoDisponivel** - para quando o número do jogador que será inserido já está a ser utilizado por outro jogador na equipa
- **LinhaIncorretaException**
- **SubstituicaoInvalida**

5 Métodos sobre equipas

Esta secção explicará melhor alguns dos métodos mais complexos sobre Equipas

5.1 criaInicial

De acordo com a táctica fornecida pelo utilizador, cria um 11 inicial automaticamente escolhendo os melhores jogadores possíveis para cada posição. Este método foi desenvolvido com o intuito de não ser necessário obrigar o utilizador a definir sempre o onze inicial de uma equipa antes de jogar e também para o calculo da habilidade (que é feito apenas tendo em conta os 11 jogadores que estão em campo num dado momento). Caso não haja jogadores suficientes de uma certa posição, serão escolhidos os jogadores disponíveis com maior habilidade (que será diminuída em 30 por cento por não se encontrarem na sua posição base).

5.2 substituiEquipa

Realiza uma substituição, verificando a presença nos titulares do jogador especificado como o de saída, procedendo à sua remoção do Map dos titulares. Esta função também tem em conta uma verificação sobre se a posição para o qual o jogador que entra (a mesma posição que o jogador retirado ocupava) corresponde à sua posição-base. Se essa correspondência não se verificar, acontece o mesmo que acontece quando é criado o 11 inicial - há um decremento de 30 por cento na sua habilidade.

6 O Jogo

Esta secção irá abordar todo o funcionamento dos métodos da classe Jogo bem como o seu impacto no projeto completo.

6.1 Raciocínio

Como foi referido na primeira secção, o desenvolvimento do jogo era abordado nos requisitos do patamar 16 e 20, tratando-se da parte mais desafiante do trabalho.

O primeiro desafio foi decidir como a simulação do jogo iria funcionar. A resposta para esse problema foi a seguinte: Dividir o período de jogo em várias subsecções (neste caso 18 , uma a cada 5 minutos) em que aconteceria um "evento". Esse evento seria uma mudança de posicionamento da bola. A bola é uma variável de instância doo tipo String que indica a posição da bola em campo numa das seguintes situações/posicionamentos :

- **Pontapé de baliza**
- **Área**
- **Canto**
- **Zona de defesa (entre a área e o meio campo)**
- **Meio Campo**

Sendo cada uma destas situações possíveis para ambas as equipas.

A forma como a bola é movida assenta num conjunto de métodos associados a cada situação de jogo que, através de atribuição de probabilidades de um resultado e de um número aleatório, escolhem a posição seguinte da bola. Cada mudança de posicionamento de bola fornece um resultado para um método do FMView que irá transmitir uma mensagem (escolhida aleatoriamente de um conjunto de 3 mensagens possíveis, para adicionar variedade) a simular o relato do jogo.

6.2 O impacto das habilidades

Uma versão inicial do sistema de jogo descrito anteriormente não tinha em conta o valor de habilidade das equipas em campo. Assim, os métodos foram alterados de forma a considerar a diferença de habilidade. Essa ideia foi implementada através de uma alteração das probabilidades de cada resultado com base no diferencial de habilidade entre as equipas.

Por exemplo : Dadas duas equipas, uma com mais 3 de habilidade do que a segunda, todos os resultados que favoreciam a equipa A terão mais 3 por cento de probabilidade de acontecer.

6.3 As substituições

Por último, foi decidido que a cada 2 períodos de jogo seria oferecida ao utilizador a possibilidade de realizar uma substituição. A decisão de o fazer de 2 em 2 períodos foi feita com a experiência de utilizador em mente: invés de estar a obrigar o utilizador a ordenar que o jogo prosseguisse a cada jogada, seria oferecida a possibilidade de 2 em 2 períodos como compromisso.

7 Funcionalidades da app

A aplicação possui diversos comandos organizados em menus. Optámos por um sistema de menus em que as opções estavam associadas a números. Cada Menu teria um switch que seleccionaria os métodos relativos à operação que o utilizador pretende realizar

7.1 Menu de Gestão

- **Adicionar Jogador** : pede ao utilizador para inserir as várias características do Jogador que pretende criar e adiciona à Equipa cujo nome foi indicado pelo mesmo
- **Adicionar Equipa** : Adiciona à liga uma equipa cujos jogadores foram indicados pelo utilzador
- **Mudar jogador de Equipa** : opção a selecionar quando se pretende transferir um jogador de uma equipa para outra, é pedido ao utilizador ,por métodos do FMView, os dados importantes para a operação

7.2 Menu de Consulta

- **Consultar Equipa**: permite ao utilizador consultar a Equipa indicada pelo mesmo.
- **Consultar Jogador** : permite ao utilizador consultar um Jogador indicado pelo mesmo.
- **Calcular Habilidade Jogador** : indica ao utilizador o valor de habilidade do jogador inidicado pelo utilizador
- **Calcula Habilidade Equipa** : gera um onze inicial default da equipa cujo nome foi indicado

7.3 Menu de Jogo

- **Escolher Equipas** : é pedido ao utilizador para indicar a equipa de casa e a equipa de fora para o jogo que será simulado
- **Escolher Tática de Casa**: à semelhança do que pode ser pedido no comando de Cálculo de Habilidade de Equipa, é pedido ao utilizador a tática do onze inicial da equipa da casa
- **EscolherTática Fora** : executa os mesmos métodos que o comando anterior mas desta vez para a equipa de fora
- **Jogar** : Inicia a simulação de jogo
- **Desforra** : No final do jogo o utilizador pode optar por fazer outra simulação do jogo entre as mesmas equipas
- **Novo Jogo** : o utilizador recomeça o jogo tendo que voltar a indicar as equipas escolhidas e o respectivo alinhamento táctico utilizando as opções referidas anteriormente.

8 Conclusão e Reflexão crítica

Neste projeto foram alcançados os objetivos propostos pelos docentes da Unidade Curricular de Programação Orientada a Objetos e respeitados também os princípios nucleares a este paradigma como o encapsulamento.

Alguns aspetos que gostaríamos de ter implementado seriam, por exemplo, adicionar mais complexidade a situações de jogo e explorar as possibilidades de uma interface gráfica inves do atual modelo de funcionamento baseado no terminal.