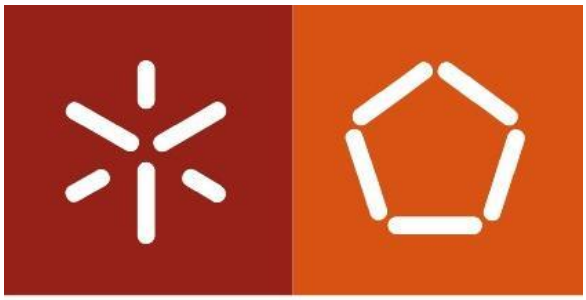

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Programação Orientada aos Objetos - Trabalho Prático

Grupo 40

Duarte Lucas - A89526 Francisco Reis - A93241 Paulo Oliveira - A93222

Ano Letivo - 2020/2021

Índice

1. Introdução ao Projeto

Este projeto teve como objetivo o desenvolvimento de uma aplicação de futebol, onde seria possível a gestão e manipulação de equipas e simulações de jogos na linguagem de programação Java, respeitando a programação orientada aos objetos, no âmbito da disciplina de Programação Orientada aos Objetos.

Neste projeto, de forma a ir ao encontro do desafio proposto e respeitando o paradigma de programação orientada aos objetos tivemos de primeiramente planear a relação que os objetos teriam entre si, se seria uma relação de composição ou agregação, bem como escolher qual a melhor arquitetura que se adequava ao projeto.

Posteriormente à planificação do projeto, tivemos a criação das várias classes de objetos, cada uma com as suas características e funcionalidades próprias de forma que, através da cooperação entre as mesmas, conseguíssemos obter o resultado final. A criação de cada classe teve antecipadamente um planeamento e divisão do projeto em "partes", isto é a divisão dos vários componentes que iriam incorporar o projeto. Esta divisão facilitou a realização do projeto uma vez que conseguimos repartir o trabalho e implementar cada classe de forma independente. É de salientar que a divisão do projeto nas várias classes teve por base o raciocínio: "O que é que irá constituir o nosso projeto?". Este raciocínio permitiu-nos salientar as várias classes que incorporam o projeto. Assim temos as várias respostas à nossa pergunta: "Ora se é um projeto sobre futebol e gestão de equipas, teremos jogadores, equipas de futebol, registos de jogos que ocorreram e jogos entre equipas de futebol". Ora deste modo temos classes como *Equipa*, *Jogador* e *Jogo* de entre várias que constituem o programa e que serão posteriormente detalhadas.

É também de salientar, que além dos vários componentes supramencionados também tivemos em conta que teríamos um cliente a interagir com o nosso programa no qual tivemos de fazer a escolha da arquitetura do projeto que supramencionámos, de forma a haver interação entre o programa e o cliente. Assim a escolha passou por ser o padrão arquitetural Model View Controller que será mais à frente detalhado.

2. Classes

2.1 Main

Esta classe é a classe principal do projeto uma vez que é através dela que se inicia o programa. Esta é responsável por chamar o método *run* da classe *Controller*, que por sua vez inicializa o programa e estabelece a ligação entre cliente e programa.

2.2 Parser

A classe *Parser* é uma classe disponibilizada pelos docentes da disciplina, que faz o tratamento dos ficheiros recebidos, nos quais estão, em modo texto, equipas, jogadores e jogos (que foram posteriormente realizados) já prontos e que têm de ser carregados para o programa. Este carregamento passa por chamar os construtores das classes necessárias tais como da classe *Equipa*, *Jogador* e *Registos*. Assim, os jogadores criados são associados às equipas correspondentes que são guardadas na estrutura de dados *Map* cuja chama é o nome da equipa e o valor é a equipa em si. Os registos de jogos são guardados na estrutura de dados *List* de *Registos*. Esta classe é chamada pela classe *Model* no arranque do programa, carregando todas as equipas do ficheiro e disponibilizando as equipas e registos através das estruturas supramencionadas.

2.3 Registos

A classe *Registos* guarda todos os eventos ocorridos num jogo que já foi realizado (carregando a informação guardada) ou que acabou de se realizar. Os vários eventos guardados são substituições da equipa da casa e de fora, os golos marcados pela equipa da casa e de fora, os jogadores integrantes da equipa da casa e de fora e também a data na qual foi realizado o jogo. Através do método *toString* da classe, é possível o cliente obter a informação sobre os vários eventos ocorridos no jogo que serão impressos no ecrã.

2.4 Jogo

A classe *Jogo* é das classes mais importante e é a que concilia várias classes tais como *Equipa*, *Jogador* e *Registos* de forma a realizar um jogo entre duas equipas. A realização do jogo entre duas equipas válidas passa por calcular o resultado do jogo que por sua vez é dividido em possíveis eventos ocorridos durante o jogo. Desta forma, existe a partição do jogo disputado entre duas equipas em eventos no qual, cada evento determina que equipa ataca e se o ataque resulta em golo ou oportunidade desperdiçada. Além dos vários eventos, existem fatores no decorrer de um jogo, como o cansaço dos jogadores que é crescente com o tempo, as substituições de ambas as equipas e a mudança de posição de um determinado jogador. No final de um jogo, é criado um registo associado a esse jogo de forma a ser guardado no historial de ambas as equipas. Além disso, é feito um “reset” a ambas as equipas de forma a recompor as alterações causadas pelos fatores acima mencionados.

2.5 Equipa

A classe *Equipa* gere e manipula uma equipa.

Esta classe é fundamental pois é das classes que constituem a base do projeto, uma vez que é nas equipas que são inseridos os vários jogadores e é entre as equipas que os jogos são disputados. Como foi acima mencionado, esta classe permite gerir e manipular uma equipa disponibilizando vários métodos adequados à classe. A gestão da equipa passa por organizar a formação, aceder ao plantel, aos titulares e aos suplentes bem como aceder a um determinado jogador, saber o nome da equipa e ver o histórico de jogos realizados pela equipa.

A manipulação passa por permitir fazer substituições de forma coerente e respeitando as regras de futebol (no máximo só se pode fazer 3 substituições por jogo e não se pode voltar a fazer entrar um jogador que já saiu), permitir a transferência de um jogador da equipa eliminando-o da mesma, alterar a formação da equipa consoante desejado e com isto alterar o *overall* da equipa e promover um jogador a suplente/titular. Esta última gestão tem em conta se a equipa titular está cheia sendo necessária ser feita uma troca entre o suplente a promover a titular e um titular que será promovido a suplente, caso a equipa titular não esteja cheia, é realizada a promoção sem trocas.

Nesta classe temos 2 tipos de construtores, um que permite a criação de uma equipa ao utilizador dando-lhe um nome e ficando inicialmente vazia (o cliente tem de criar/ transferir jogadores de forma a ter a equipa válida) e outro é responsável por receber a equipa proveniente de um ficheiro e criá-la consoante as características recebidas.

Os jogadores que fazem parte dos titulares são inseridos na estrutura de dados *List* de *Jogador* assim como os suplentes e os jogadores todos (titulares e suplentes) por uma dada ordem (esta ordem da lista de todos os jogadores é importante para no final do jogo dar "reset" às equipas devido a possíveis substituições). O plantel é guardado num *Map* cuja chave é o número da camisola do jogador e o valor é o jogador em si, sendo esta estrutura importante na verificação da existência de um dado jogador.

É de salientar a importância de um método, *equipaValida*, que verifica se uma dada equipa é válida para jogo, isto é, se tem onze titulares e um deles é guarda-redes. Desta forma, existe o controlo das equipas que podem ir a jogo e disputá-lo.

2.6 Jogador

A classe *Jogador* é uma classe abstrata pois um jogador pode ser um dos seguintes tipos de dados:

- Guarda-Redes;
- Defesa;
- Lateral;
- Médio;
- Avanço;

Assim temos a pormenorização do que é um jogador consoante a sua posição e é estabelecida um relação de herança entre a superclasse *Jogador* e as subclasses supramencionadas.

Além disso, o *overall* do jogador (característica fundamental para saber o quão forte é uma equipa em relação a outra) é afetado consoante a sua posição (basicamente consoante o seu tipo de dados). Também cada jogador tem uma zona de campo associado à sua posição, o *role*, que permite perceber se uma dada mudança de posição é válida. É de salientar que um jogador tem uma "partição" entre as variáveis que são afetadas durante o decorrer de um jogo e fora do mesmo. Com isto, temos variáveis de instância que não são afetadas pelo decorrer de um jogo e variáveis de instância que são usadas e alteradas no decorrer de um jogo. Isto permite-nos fazer alterações ao jogador durante um jogo e no final dar "reset" às características alteradas durante o jogo. Estas variáveis de instância próprias de um jogo são: *role_jogo*, *posiçao_jogo* e *overall_jogo*, que permitem fazer as seguintes alterações durante um jogo:

- substituir um jogador, ou seja, um jogador que entra pode ir para qualquer zona de campo, afetando o seu *role_jogo* e *overall_jogo*;
- mudar a posição de um jogador, ou seja, uma mudança de posição do jogador apenas é válida se for para a mesma zona de campo, afetando a *posiçao_jogo* e *overall_jogo;*

Fora de um jogo, as alterações feitas a um jogador, como mudar a posição, promovê-lo a suplente/titular, transferi-lo e consequências de alterar a formação da equipa onde se insere são permanentes e afetam as variáveis de instância de jogo e as originais.

É de salientar que quando um jogador é substituído, o estado do jogador que sai é alterado dizendo que foi substituído e fica com o *role* vazio e todos os suplentes têm o *role* vazio uma vez que, tal como foi dito, um suplente que entra em jogo pode ir para qualquer zona de campo.

Um jogador também tem guardado o histórico de equipas pelas quais passou, sendo este histórico atualizado aquando de uma transferência.

O cálculo do *overall* do jogador é feito nas subclasses e posteriormente enviado à superclasse, tendo em consideração as estatísticas apropriadas conforme a posição que ocupa.

Por fim, esta classe permite a criação de um jogador cujas características são provenientes de um ficheiro e/ou especificadas pelo cliente.

- ## Guarda-Redes

Um guarda-redes tem como zona de campo a baliza e este, no decorrer de um jogo, apenas pode ser substituído por um outro guarda-redes. Um guarda-redes tem como habilidade especial a elasticidade que é calculada tendo em consideração as estatísticas apropriadas.

- ## Defesa

Um defesa tem como zona de campo o centro sendo referido por defesa central e este, no decorrer de um jogo pode ser substituído por qualquer outro jogador, sendo que o jogador (suplente) que entra fica a jogar na zona central do campo (o *role* é alterado para

“centro”). Um defesa tem como habilidade especial o desarme que é calculado tendo em consideração as estatísticas apropriadas.

- **Lateral**

Um lateral tem como zona de campo o lado e este, no decorrer de um jogo pode ser substituído por qualquer outro jogador, sendo que o jogador (suplente) que entra fica a jogar na zona lateral do campo (o *role* é alterado para “lado”). Um lateral tem como habilidade especial o cruzamento que é calculado tendo em consideração as estatísticas apropriadas.

- **Médio**

Um médio tem como zonas de campo disponíveis o centro e a lateral sendo referido por médio centro ou médio ala respetivamente e este, no decorrer de um jogo pode ser substituído por qualquer outro jogador, sendo que o jogador (suplente) que entra fica a jogar na mesma zona de campo que o médio que saiu (o *role* é alterado para “centro” ou “lado”). Um médio tem como habilidade especial a recuperação que é calculada tendo em consideração as estatísticas apropriadas.

- **Avançado**

Um avançado tem como zonas de campo disponíveis o centro e a lateral sendo referido por avançado centro ou extremo respetivamente e este, no decorrer de um jogo pode ser substituído por qualquer outro jogador, sendo que o jogador (suplente) que entra fica a jogar na mesma zona de campo que o avançado que saiu (o *role* é alterado para “centro” ou “lado”). Um avançado tem como habilidade especial a finalização que é calculada tendo em consideração as estatísticas apropriadas.

Funcionalidades

Tal como foi supramencionado, de forma a termos um programa mais interativo e fácil de manusear, o cliente tem acesso a uma lista de comando com várias funcionalidades.

Estas funcionalidades são disponibilizadas ao cliente no arranque do programa, através da impressão do menu de ajuda no qual constam as várias funcionalidades e as suas descrições.

Deste modo o cliente consegue interagir com o programa através da seguinte lista de funcionalidades:

Menu de Ajuda

- **Help** -> Menu de ajuda
- **Quit** -> Sair do programa;
- **Clear** -> Limpa a tela do programa;

Lista de comandos outGame

- **AllTeams** -> Lista todas as equipas do programa;
- **ShowTeam** <NomeEquipa> -> Mostra a equipa selecionada;
- **ShowGames** <NomeEquipa> -> Mostra o histórico de jogos da equipa selecionada;
- **ShowPlayer** <NomeEquipa> -> Mostra o jogador selecionado da equipa respetiva;
- **CreatePlayer** -> Cria um jogador com as características e inseri-o numa equipa valida;
- **RemoveEleven** <NomeEquipa> -> Remove um jogador dos titulares e inseri-o nos suplentes;
- **CreateTeam** <NomeEquipa> -> Cria uma equipa previamente sem jogadores;
- **Transfer** <NomeEquipaAtual> -> Transferência de um dado jogador para outra equipa;
- **ChangePos** <NomeEquipa> -> Muda a posição de um dado jogador;
- **Formation** <NomeEquipa>=<Defesas Medios Avancados> -> Altera a formação da equipa;
- **AddEleven** <NomeEquipa> -> Promove um suplente a titular;
- **LoadProgram** <NomeFicheiro> -> Carrega um ficheiro binário (programa) previamente guardado;
- **SaveProgram** <NomeFicheiro> -> Guarda em ficheiro binário o programa;

Lista de comandos inGame

- **Play** -> Começa um jogo entre as duas equipas selecionadas;
- **SubHome** -> Substituição entre os jogadores selecionados da equipa da casa;
- **SubAway** -> Substituição entre os jogadores selecionados da equipa da casa;
- **ShowTeamHome** -> Mostra a equipa da casa;
- **ShowTeamAway** -> Mostra a equipa de fora;
- **ShowPlayerHome** -> Mostra o jogador da casa a ser previamente selecionado;
- **ShowPlayerAway** -> Mostra o jogador de fora a ser previamente selecionado;
- **ChangePos** <NomeEquipa> -> Muda a posição de um dado jogador (não altera a zona de campo!);
- **Next** -> Prossegue com o decorrer do jogo;

4. Arquitetura do projeto

Tal como foi referido na introdução, no início do projeto tivemos que planear a estrutura do programa, sendo que a planificação teve várias considerações como estabelecer interação com o cliente e que tipo de relação teriam os objetos entre si. Primeiramente decidimos escolher que as relações entre os objetos seriam de agregação no qual se guarda internamente o apontador dos objetos passados como parâmetro e devolve-se sempre o apontador dos objetos e, caso seja solicitado, uma cópia da estrutura de dados que os guarda. É de extrema importância salientar que ao longo de todo o projeto tivemos em conta a um dos pilares da programação orientada aos objetos que é o **encapsulamento** no qual não se acede diretamente a variáveis de instâncias fora da classe e tudo é encapsulado.

Por fim, arquitetura do nosso projeto segue a estrutura Model View Controller (MVC), estando por isso organizado em três camadas:

- **Model** (Modelo)

  A camada de dados é composta pelas classes mais importantes como por exemplo a Classe Equipa, Classe Jogador e Classe Jogo, no qual pedidos do controlador são recebidos e processados chamando os métodos necessários nas classes que implementam a base do programa e que posteriormente lhe devolve o resultado;

- **View** (Vista)

A camada de interação com o utilizador é responsável pela leitura e escrita de dados no terminal, enviando os pedidos do cliente ao controlador e devolve ao cliente o resultado dos seus pedidos;

- **Controller** (Controlador)

A camada de controlo do fluxo do programa é responsável pela ligação da *View* com o *Model*. Recebe da *View* os pedidos do cliente, estes são carregados pelo *Controller* que irá mandar o pedido do cliente para serem processados pelo *Model* e consequentemente o *Model* processa o pedido e envia o resultado para o *Controller* que por sua vez envia o resultado para a *View* e dá ao utilizador o resultado do seu pedido.

5. Exceções

Ora, uma vez que há interação com o cliente este pode introduzir funcionalidades que o programa não reconhece sendo que a forma como o programa lida com isto é apenas dizer ao cliente que introduziu um comando inválido. Mas caso tenho introduzido um comando válido cujo os parâmetro são inválidos que acontece? A resposta a isto passa pelo tratamento de exceções no qual o controlador ao receber o comando vindo da vista, analisa e tem em conta se os argumentos são válidos como por exemplo, quando é pedido ao cliente o número do jogador e este introduz um caracter, no qual é o controlador que envia a mensagem à vista para posteriormente enviar ao cliente a mensagem de que um dos parâmetros é inválido. “E se o utilizador envia o comando correto, com os parâmetros corretos, mas é impossível realizá-lo porque o modelo não sabe como lidar com “segmentation faults”? É aqui que entram as exceções, através da classe *ComandoInvalidoException* o modelo consegue lançar exceções ao controlador que através do método polimórfico *showInfo* envia à vista a exceção para mostrar ao cliente o erro. Estas exceções são por exemplo uma equipa inexistente, caso o cliente introduza o comando *showTeam <NomeEquipa>* de forma válida, mas a equipa não existir. O modelo, vê se esta equipa existe no *Map* das equipas existentes no programa e caso não exista lança a exceção de *ComandoInvalidoException*.

6. Conclusão

De um modo geral, achamos que fizemos um trabalho bem conseguido. Tentámos cumprir com tudo o que nos foi proposto, procuramos uma solução eficiente para respeitar os vários condicionantes existentes, como por exemplo o *role* dos jogadores, a formação das equipas, o número de jogadores possíveis em casa posição. Do nosso ponto de vista, achamos mais desafiante a manipulação do jogador de cada equipa, mais precisamente a gestão da zona de campo e a posição de cada um que por sua vez implicaria um maior cuidado com as substituições em jogo, fora do jogo e trocas de posição entre os titulares. Achámos também interessante a manipulação do jogo, onde tentámos implementar um jogo, em que à medida que o jogo decorre 15 em 15 minutos é feito um comentário sobre um possível lance

ocorrido (jogadas de perigo, golos…). Nestes intervalos também é possível a intervenção do utilizador podendo fazer substituições, mostrar um jogador, uma equipa, entre outras intervenções.

Em suma, o projeto deu-nos a oportunidade de pôr em práticas os conceitos aprendidos na disciplina de Programação Orientada aos Objetos, respeitando os pilares do paradigma de programação orientada aos objetos, como o encapsulamento, polimorfismo e herança, bem como superar as dificuldades em relação a alguns conceitos do paradigma e por fim realizar um bom programa.

7. Diagrama de Classes