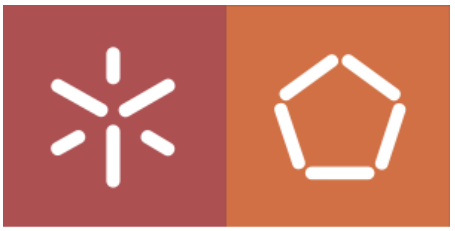

Universidade Do Minho
Departamento de Informática

SPORTS MANAGER

Trabalho Prático de Programação Orientada aos Objetos
Grupo nº 71

Francisco Neves
(A93202)

Leonardo Freitas
(A93281)

Miguel Martins
(A89584)

12 de junho de 2021

Índice

1. Introdução

O projeto realizado tem como objetivo a construção de um programa semelhante ao conhecido jogo *Football Manager*, tendo sido, para isso, criado um sistema de gestão e simulação de equipas para um determinado desporto, sendo que, o foco da criação do programa foi futebol.

De forma a ser possível criar um programa sofisticado, procuramos atingir os diversos objetivos do Paradigma Orientado aos Objetos e da linguagem de programação Java, nomeadamente, a modularidade e reutilização do código, a abstração e a hierarquia de classes.

Além disso, desde a génese do projeto, o mesmo foi desenvolvido numa arquitetura MVC de forma a obtermos um projeto mais organizado, permitindo assim, um desenvolvimento rápido e escalável, sendo que, através desta arquitetura tornou-se mais simples a inserção de novas funcionalidades do projeto, visto que, a inserção de alterações no *Model* não necessitam de ser imediatamente implementadas no *Controller* e no *View*.

2. Arquitetura Utilizada

Desde a sua génese, a aplicação foi concebida tendo em mente o modelo de desenvolvimento MVC (Model-View-Controller). Assim sendo, podemos dividir o nosso projeto em três grandes bases, sendo elas:

- *Model* - Agrega todas as classes que são utilizadas no armazenamento de estruturas de dados e a parte algorítmica do programa;
- *View* - Agrega as classes que permitem oferecer resposta visual aos pedidos do utilizador;
- *Controller* - Controla o funcionamento do programa, funcionando, muitas vezes, como intercomunicador entre o *Model* e a *View*.

Relativamente ao *Model*, tendo em vista a hierarquia de classes, decidimos começar por definir classes abstratas que nos ajudaram na criação de diferentes classes que partilhavam algumas variáveis de instância. Dessa forma, decidimos, em primeiro lugar, criar uma classe abstrata que daria para qualquer jogador (independentemente da sua posição) à qual chamamos *Player*. Em seguida, tendo em conta as necessidades dos guarda-redes, consideramos benéfico que estes tivessem os seus próprios atributos que os distinguiria dos restantes jogadores, assim sendo, criou-se duas novas classes que herdam a classe *Player*, a classe *GK*, que é a classe de um guarda-redes e a classe abstrata *FieldPlayer* que trata de todos os outros jogadores. Por fim, dividimos esta última nas diversas subclasses que representam as restantes posições, *DF* (para os defesas), *WG* (para os laterais), *MD* (para os médios) e, por fim, *FW* (para os avançados).

No que toca ao *Controller*, decidiu-se dividir o *Controller* em várias classes mais pequenas partilhando todas o mesmo *Model*, sendo que, desta forma, quando um *Controller* afeta o *Model*, todos os outros têm também o *Model* atualizado.

Por fim, quanto à *View*, tal como o *Controller* é subdividida em diferentes *Views* responsáveis, cada uma, por uma parte específica do programa, por exemplo, a classe *ViewPlayer* está encarregue de todos os *outputs* relacionados com jogadores. Além disso, esta tem uma classe *Outputs* que contém métodos gerais capazes de imprimir menus e tabelas que, em seguida, serão chamados pelas várias classes da *View*. Consideramos importante salientar que todas as classes da *View* são finais com métodos *static*, o que possibilita que os seus métodos sejam chamados sem recurso à criação de um Objeto *View* poupando, desta forma, espaço em memória.

3. Algumas Classes

- **Model**
  - *GK, DF, WG, MD e FW*
    - Classes utilizadas para o tratamento de cada jogador, sendo que cada uma delas é representativa da posição com o seu nome.
  - *Team*
    - Classe que permite tratar dos dados de uma equipa como o seu nome, os seus jogadores e o seu histórico de jogos.
  - *Game*
    - Classe que permite tratar dos dados de um jogo, como por exemplo, as equipas pelas quais o jogo está a ser disputado, os jogadores (representados pelo seu número) que estão em campo, o resultado, o tempo de jogo e as substituições realizadas.
  - *HistoryGame*
    - Classe que permite simplificar os dados de um jogo, para que apenas estes sejam guardados no histórico de jogos de uma equipa.
    - Surgiu devido à impossibilidade de manter todos os dados em memória de forma eficiente se fossem utilizados os jogos criados pela classe *Game*.
  - *Utils*
    - Classe que possui diversos métodos úteis para diversas outras classes.
  - *Meteorology e Position*
    - *Enums* auxiliares que permitem a definição de constantes.
  - *Model*
    - Classe que trata os dados da base de dados atual para resposta aos diversos requisitos do *Controller*.
- **Controller**
  - *PlayerController*
    - Responsável por realizar todas as ações relacionadas com um jogador, por exemplo, a sua criação, a visualização dos seus dados ou a atualização dos seus atributos.
  - *TeamController*
    - Responsável por realizar todas as ações relacionadas com uma Equipa, por exemplo, a sua criação, a visualização dos seus dados ou a visualização do seu histórico de jogos.

- *GameMenuController*
  - Responsável pela criação de jogos e a sua simulação.
- *LoadFile*
  - Responsável pela leitura dos logs.
- *LoadSaveController*
  - Responsável por guardar e por ler o estado do programa num ficheiro com a extensão .sm.
- *Inputs*
  - Classe *final* responsável por pedir *inputs* ao utilizador de maneira "segura".

- **View**
  - *MainMenu*
    - Responsável por apresentar o menu principal.
  - *ViewPlayer*
    - Responsável por apresentar todas as opções ao utilizador relativas a um jogador.
  - *ViewTeam*
    - Responsável por apresentar todas as opções ao utilizador relativas a uma Equipa.
  - *ViewGame*
    - Responsável por apresentar todas as opções ao utilizador relativas a um jogo.
  - *LoadSaveMenu*
    - Responsável por apresentar o menu com todas as opções relativas a guardar ou ler ficheiros de estado.
  - *Outputs*
    - Classe *final* responsável por apresentar tabelas e menus em funções genéricas.

4. Funcionalidades

- **Menu Principal**

Ao ser inicializada a aplicação é apresentado ao utilizador o menu principal, proporcionando-lhe a oportunidade de escolher uma das opções disponíveis de forma a interagir com o programa.

Fig. 1 - Menu Principal

- **Menu do Jogador**

  Ao escrever “1” o utilizador é encaminhado para um menu de jogador.

Fig. 2 - Menu do Jogador

Dentro deste menu são possíveis as seguintes opções:

1. *Criação de um novo jogador* - deverá ser fornecido o nome do jogador, o nome da equipa, o número do jogador e a sua posição. Além disso, é possível escolher a opção da criação de atributos aleatórios ou de gerar os atributos manualmente.
2. *Ver todos os jogadores* - apresenta uma tabela com todos os jogadores da base de dados.
3. *Procurar a informação de um jogador através do seu nome* - deverá ser fornecido o nome do jogador desejado e será devolvida a informação associada a ele.
4. *Procurar o histórico de equipas de um jogador através do seu nome* - deverá ser fornecido o nome do jogador desejado e será devolvido o histórico de equipas do jogador.

- **Menu da Equipa**

Ao escrever “2” no menu principal, o utilizador é encaminhado para um menu de equipa.

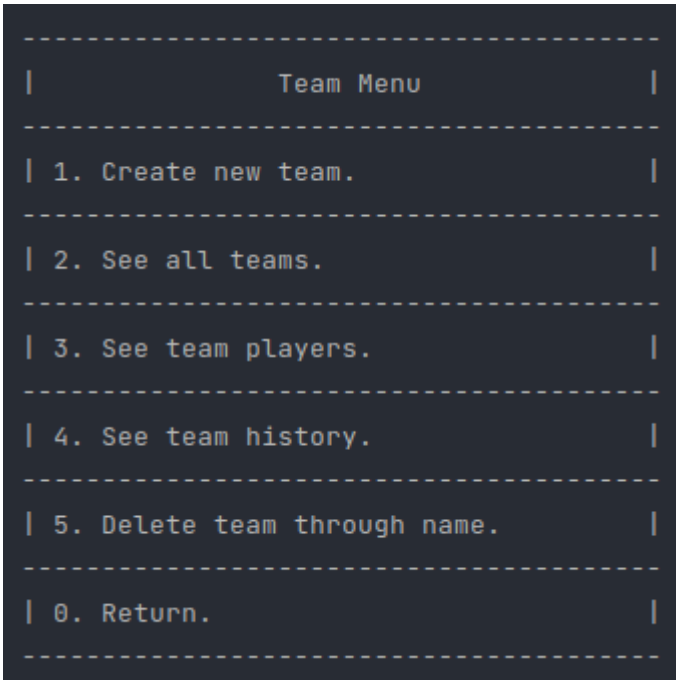

Fig. 3. - Menu da Equipa

Dentro deste menu são possíveis as seguintes opções:

1. *Criar uma nova equipa* - deverá ser fornecido o nome da equipa e a equipa será criada vazia.
2. *Ver todas as equipas* - apresenta uma tabela com todas as equipas da base de dados.
3. *Ver os jogadores de uma equipa* - deverá ser fornecido o nome da equipa e será apresentada uma tabela com todos os jogadores da equipa.
4. *Ver o histórico de jogos de uma equipa* - deverá ser fornecido o nome da equipa e será apresentado o seu histórico de jogos.

- **Jogo**

Ao escrever “3” no menu principal, o utilizador é encaminhado para o menu de jogo. Lá ser-lhe-á oferecida opção de criar um novo jogo, sendo que, para isso, deverá fornecer o nome de cada uma das equipas, selecionar a tática de cada uma delas, os jogadores utilizados, tendo em conta, que todos a posição de guarda-redes e de lateral deve ser apenas ocupada por um jogador natural dessa posição, enquanto que, as outras posições podem ser ocupadas por jogadores de qualquer posição (exceto guarda-redes ou laterais), sendo que, o seu *overall* altera conforme a posição em que joga. Por fim, o utilizador deverá também escolher em quantas partes pretende dividir o jogo.

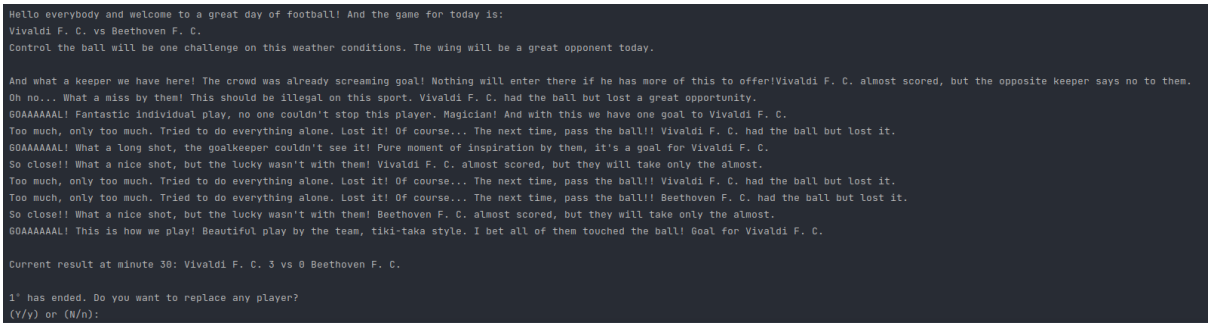

Fig. 4. - Jogo

Ao longo do jogo, sempre que se efetua uma paragem, o programa pergunta ao utilizador se este quer executar uma substituição após lhe fornecer um pequeno resumo dos lances do jogo até esse momento. Se o utilizador decidir efetuar uma substituição, ele deverá decidir em que equipa pretende fazer a substituição, em seguida, ser-lhe-á mostrado os jogadores que estão a jogar no momento para que ele indique (através do número) que jogador pretende substituir. Após isso, ser-lhe-ão fornecidos os jogadores que podem substituir esse mesmo jogador.

- **Leitor de Logs**

Ao escrever “4” no menu principal, serão lidos os logs que deverão ter como caminho de ficheiro “data/logs.txt”.

- **Gravar/Carregar**

Ao escrever “5” no menu principal, o utilizador será encaminhado para um menu onde deverá escolher gravar o estado atual num ficheiro com o nome que ele desejar, ou carregar um estado gravado dos ficheiros presentes na diretoria *saves*. Todos os ficheiros gravados serão gravados nessa mesma diretoria e com a extensão “*.sm*”, indicativa de *Sports Manager*.

- **Reset da base de dados**

Ao escrever “6” no menu principal, a base de dados será reiniciada.

5. Conclusão e Análise Crítica

Somos da opinião que o nosso projeto foi bem conseguido, obtendo resposta a todos os pedidos da equipa docente e acrescentando alguns itens que consideramos serem pertinentes, como por exemplo a adição de mais atributos aos jogadores que nos permitem ter um cálculo mais fiável da habilidade do jogador a efetuar uma determinada posição, ou a redução da habilidade "*Positioning"* dependendo da função que o jogador irá efetuar.

No entanto, estamos também cientes da existência de alguns pontos que poderiam ter sido melhor implementados, como por exemplo, a possibilidade de ter equipas ou jogadores com o mesmo nome originando um *ID* para cada entidade adicionada, além disso, podíamos também, em certas ocasiões, retirar melhor partido da reutilização de código.

Em suma, o nosso grupo foi capaz de implementar todos os requisitos pedidos usufruindo dos conceitos fundamentais do Paradigma Orientado aos Objetos, como o encapsulamento, a modularidade e a reutilização de código. Foi também utilizado o modelo *MVC* para a implementação do projeto. Somos, portanto, da opinião que o nosso projeto foi concluído com sucesso perante os pedidos da equipa docente desta unidade curricular.

6. Diagramas

- **Diagrama de Classes**