

[UNIVERSIDADE DO MINHO](https://www.uminho.pt/PT)

[Mestrado Integrado em Engenharia Informática](https://miei.di.uminho.pt/)

Programação Orientada Aos Objetos

Grupo 3

Carlos Daniel da Silva Santos (a85617)

Ana Paula Oliveira Henriques (a93268)

Ana Filipa Ribeiro Murta (a93284)

Ano Letivo 2020/2021

Índice

1. Introdução e principais desafios 3
2. Arquitetura de classes 4
   1. Teste . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   2. Jogador . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   3. Subclasses de Jogador . . . . . . . .. . . . . . . . . . . . . . . . . . . . 4
   4. CriarJogador . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   5. Equipa . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
   6. CriarEquipa . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   7. EstadoJogo. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   8. Jogo . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   9. JogoRegisto . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   10. MainMenu . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   11. GerirMenu . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   12. JogMenu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   13. ParserMod . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   14. Saver . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
   15. Data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   16. Exceptions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
3. Aplicação desenvolvida 10
4. Conclusão 10
5. Introdução e principais desafios

Este projeto consistiu no desenvolvimento de um jogo de futebol, semelhante ao conhecido jogo *Football Manager*, na linguagem de programação Java de forma a pôr em prática os conhecimentos aprendidos ao longo do semestre.

O programa desenvolvido consiste, portanto, numa partida entre duas equipas, uma identificada pela equipa que joga em casa e a outra pela equipa que joga fora, que foram escolhidas a partir de um ficheiro fornecido pelos docentes ou, então, criada pelo usuário.

Um jogo tem duração de 90 minutos em que, aos 45 minutos, é feito um intervalo para se efetuar substituições. Ao longo do jogo, serão feitas jogadas que serão determinadas consoante os números aleatórios gerados pela classe Random do pacote java.util.

O maior desafio enfrentado é capaz de ter sido as restrições dos jogadores consoante as suas posições ...................................

**IMPORTANTE**: todas as classes estão associadas por composição.

1. Arquitetura de classes
   1. Main

Esta classe é responsável por arrancar o programa, começando por imprimir no terminal o menu principal e, a partir daí, o usuário escolherá o que pretende fazer a seguir tendo em conta as opções que lhe são apresentadas.

* 1. Jogador

A cada tipo de jogador é associado um inteiro de forma a podermos pegar diretamente num objeto de tipo Jogador e saber se é um avançado, um médio, um lateral, um defesa ou um guarda-redes:

**private** static final int AVANCADO = 1;

**private** static final int MEDIO = 2;

**private** static final int LATERAL = 3;

**private** static final int DEFESA = 4;

**private** static final int GR = 5;

Um jogador tem um conjunto de características-base (cujos valores estão entre 0 a 100):

**private** String nome;

**private** int nr\_camisola;

**private** int velocidade;

**private** int resistencia;

**private** int destreza;

**private** int impulsao;

**private** int jogoCabeca;

**private** int remate;

**private** int capPasse;

**private** List<String> historico;

**private** int tipoJogador;

* 1. Subclasses de Jogador

A superclasse Jogador possui as seguintes subclasses:

Avancado:

**private** int drible;

Medio:

**private** int capRecuperacao;

**private** int dominioBola; -- característica extra

Lateral:

**private** int capCruzamento;

Defesa:

**private** int desarme;

GuardaRedes:

**private** int elasticidade;

**private** int lancamento; -- característica extra

Para além das características-base que nos foram fornecidas, foi-nos, também, proposto acrescentar características extra consoante o tipo do jogador. As únicas características extra que adicionámos foram: o lançamento de bola do guarda redes (com o pé ou com a mão); o domínio da bola do médio, referente à sua capacidade de controlar a bola; o desarme do defesa, que consiste em desarmar uma equipa; e o drible do avançado. A estas características extras foram gerados números aleatórios entre 0 a 100 pela classe Random do pacote java.util.

Cada jogador possui um valor de habilidade. Ora, os métodos que calculam a habilidade dum jogador encontram-se em cada uma das superclasses visto que este cálculo vai depender das características específicas de cada tipo de jogador. Quanto mais elevado for este valor, melhor é o desempenho esperado do jogador.

* 1. CriarJogador

Esta classe destina-se a criar um jogador com a intervenção do usuário, ou seja, é o usuário que escolhe que tipo de jogador é que pretende criar e lhe atribui um nome e valores às suas características. Depois disto, é perguntado ao utilizador a que equipa é que quer associar este novo jogador, inserindo-o nessa equipa caso esta exista.

* 1. Equipa

Esta classe está encarregue de trabalhar com tudo relacionado com as equipas.

**private** int nr\_equipa;

**private** int nr\_tatica;

**private** String nome;

**private** Map<Integer, Jogador> jogadores;

**private** Map<Integer, Integer> titulares;

**private** Map<Integer, Integer> suplentes;

No map jogadores, cada objeto jogador (value) está associado ao número da sua camisola de cada jogador (key). Já no map titulares, cada posição no jogo do titular (value) está associado ao número da sua camisola (key). O mesmo pensamento foi aplicado noo map suplentes, em que cada posição no jogo do suplente (value) está associado ao número da sua camisola (key).

Estes foram os inteiros associados a cada posição no jogo:

**1** guarda redes

**2** defesa central

**3** defesa lateral

**4** médio / médio centro

**5** extremos

**6** avançado / avançado centro

**7** não existe no jogo, é suplente

O maior desafio desta classe foi, de facto, fazer as substituições de titulares por suplentes dentro de uma equipa consoante a sua posição.........

**FILIPAAAAAAAAAAAAAA**

* 1. CriarEquipa

Esta classe destina-se a criar uma equipa com a intervenção do usuário: é lhe pergunta o nome da nova equipa e depois é criado um objeto de tipo Equipa com todos os seus campos vazios, exceto o campo referente ao nome. É-lhe, ainda, dada a opção de criar os 20 jogadores para essa equipa.

* 1. EstadoJogo

**private** LocalDate data;

**private** Equipa equipaCasa;

**private** Equipa equipaFora;

**private** int scoreCasa;

**private** int scoreFora;

**private** List<Integer> jogadoresCasa;

**private** List<Integer> jogadoresFora;

**private** Map<Integer,Integer> substituicoesCasa;

**private** Map<Integer,Integer> substituicoesFora;

**private** int nrSubstituicoesCasa;

**private** int nrSubstituicoesFora;

Nesta classe, tal como o seu nome sugere, trabalha-se o estado do jogo, desde quem joga em casa a quem joga fora, o resultado de quem joga em casa ao resultado de quem joga fora, por aí em diante. Por outras palavras, trabalha-se tudo relacionado com as duas equipas que estão a jogar em campo numa determinada partida.

* 1. Jogo

private String equipaAtual;

private int gameProgress;

private EstadoJogo estado;

Nesta classe, cria-se o dito jogo. Temos, então, um método que verifica se um jogo ainda não começou, iniciando-o se sim e outro que verifica se a partida já terminou, apresentando, se sim, os dados referentes à pontuação final e a que equipa ganhou.

Para iniciar e construir as jogadas, distingue-se três métodos em que, em todos, a estratégia utilizada foi gerar números aleatórios através da classe Random do pacote java.util:

O iniciaJogada() é responsável por começar uma jogada a partir de um estado. Se o número aleatório gerado estiver entre 6 a 9 (inclusive), começa-se uma jogada; caso contrário, não acontece nada. No entanto, dentro daquele intervalo, são feitas duas divisões consoante o valor da habilidade de cada equipa. Para fazer estas divisões, tem-se de calcular as hipóteses de cada equipa de ganhar o jogo. Aquela que possuir maior hipótese é a que tomará posse da bola jogada iniciada.

O constroiJogada() está encarregue de construir uma jogada a partir de um estado. Se o número aleatório gerado estiver entre 0 a 2 (inclusive), a equipa com posse de bola perde a bola para a equipa adversária; se estiver entre 3 a 6 (inclusive), um jogador da equipa com posse de bola passa a bola para outro jogador; se estiver entre 7 a 8 (inclusive), a equipa adversária pressiona os adversários; e, por último, se for 9, a equipa com a bola remata.

O remate() efetua um remate a partir de um estado. Se o número aleatório gerado estiver entre 0 a 3 (inclusive), a equipa com a bola remata e marca golo; se estiver entre 4 e 6 (inclusive), esta remata, mas o guarda-redes adversário defende; e se estiver entre 7 a 9 (inclusive), esta remata, mas falha a baliza.

* 1. JogoRegisto

Esta classe do código, delimita-se a moldar um objeto capaz de guardar todas

as informações necessárias de um jogo que já efetuado anteriormente, daí o nome de registo.

**private** String equipaCasa;

**private** String equipaFora;

**private** int golosCasa;

**private** int golosFora;

**private** LocalDate date;

**private** List<Integer> jogadoresCasa;

**private** List<Integer> jogadoresFora;

Map<Integer, Integer> substituicoesCasa;

Map<Integer, Integer> substituicoesFora;

Primeiramente, esta classe será usada logo de inicio no jogo, para guardar o registo de Jogos existente no ficheiro logs.txt fornecido, Jogos estes, que depois de passarem pelo parse(String input), vão pode ser consultados no Menu principal através do método apresentarJogo(), que apresenta, a data em que foi efetuado o jogo, as equipas e os seus respetivos resultados.

Apesar das suas semelhanças nas variáveis com a classe EstadoJogo, apenas se remete a guardar o nome de cada equipa, pois as táticas e características já foram usadas e podem ser descartadas em prol do Jogo ser guardado de forma menos pesada no objeto Data, e eventualmente guardado num ficheiro com a função saver(PrintWriter print).

* 1. MainMenu

A função interativa do jogo inicia-se nesta classe, ou seja, tem o objetivo de

chamar objetos e métodos de modo a decidir o rumo do jogo, de acordo com as

escolhas do utilizador.

**private** int option;

**private** Data dados;

No construtor desta classe, são apresentadas as opções disponíveis ao utilizador,

sendo estas a opção de criar uma partida verificando sempre a validez das equipas que forem escolhidas, criando um JogMenu caso os parâmetros estejam corretos. Também existe a possibilidade de gerir o jogo e os seus dados, como por exemplo, criar equipas ou transferir jogadores.

Outra opção será a visualização do registo de jogos já ocorridos em forma de páginas, com a opção de verificar os jogos de acordo com a sua data.

Adicionando as opções têm também métodos que carregam um ficheiro, ou simplesmente guardam os dados da variável instância num ficheiro chamado dados.txt.

* 1. GerirMenu

Tal como o MainMenu esta classe oferece um conjunto de opções para o

utilizador escolher com o propósito de manipular os dados do jogo.

**private** int option;

**private** Data dados;

O construtor inicia por apresentar as diferentes escolhas, e de acordo com a opção do utilizador, o menu tem a habilidade de criar uma equipa, (com a opção de criar 20 jogadores para a respetiva equipa) tal como criar um jogador, e as suas respetivas características, e adicionar na base de dados. É também possível a escolha de apenas inspecionar as equipas, os seus planteis e os jogadores representados, na base de dados

Por fim o Gerir Menu dá também a escolha de transferir um jogador de equipa para equipa na função transferirJog(), que após perguntar a equipa origem, a equipa destino e o jogador em causa, vai mover o jogador, aumentando o histórico deste com a sua antiga equipa.

* 1. JogMenu

O terceiro Menu do jogo vai ser apresentado apenas na preparação de uma

partida planeada pelo jogador no estado.

**private** int option;

**private** Jogo jogo;

**private** EstadoJogo estado;

**private** Data dados;

Através do estado de jogo, depois de receber a escolha do utilizador, o JogMenu vai conseguir escolher as substituições a serem feitas no jogo com antecipação, verificando todos os parâmetros tal como a existência de jogadores e a possibilidade do seu tipo (Avancado, Médio, etc.) pode jogar nas posições mais específicas (Defesa Centro, Defesa Lateral, etc.) .

É possível também fazer a escolha da tática a ser usada por uma equipa através da função escTatica()o que faz com que os titulares, sejam automaticamente atribuídos de acordo com as suas novas posições.

* 1. ParserMod

Nesta classe, é feita o parse dos dados sobre os jogadores, as equipas e os jogos a partir do conteúdo do ficheiro input. Os métodos desta classe foram fornecidos pelos docentes, nós apenas o adaptámos e melhorámos para aquilo que era preciso.

* 1. Saver

Nesta classe, encontra-se o método estático save(Data dados), que ao inverso do Parser vai guardar toda a informação do jogo carregada e modificada pelo utilizador, no ficheiro dados.txt para ser carregada no futuro.

* 1. Data

Esta classe é onde será criado um objeto que guarda toda a informação do

Jogo, ou seja, as equipas e o registo de Jogos.

**private** Map<String, Equipa> equipas;

**private** List<JogoRegisto> jogos;

Não só o objeto é utilizado para ter sempre o jogo atualizado, como também usa métodos próprios para apresentar a informação guardada em detalhe, juntamente com a habilidade de chamar métodos para dividir o plantel de equipa em titulares e suplentes, de acordo com a tática.

* 1. Exceptions

**FILIPAAAAAAAAAA**

