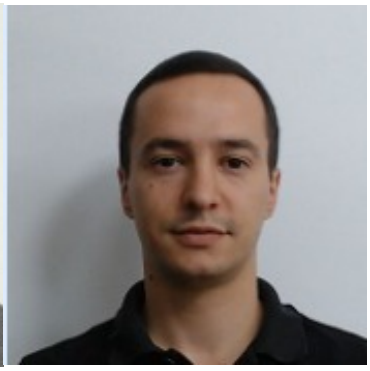
Programação Orientada aos Objetos
Relatório de Projeto - Grupo 21

A93324 Daniel Torres Azevedo A70573 Mário Rui Sendas Rocha
A42865 Mário Rui Freitas Coelho


A93324 Daniel


A70573 Mário


A42865 Mário Coelho

1 Introdução

Este trabalho realizado no âmbito da unidade curricular de Programação Orientada aos Objetos tem o objetivo de desenvolver uma aplicação para criar um sistema de gestão e simulação de equipas. Embora se pretenda que o sistema seja versátil e adaptável a qualquer desporto, no presente relatório apresenta-se o caso particular do futebol a titulo ilustrativo.

Esta aplicação permite simular a existência de um conjunto de jogadores, agrupados em equipas. É possível também simular a realização de jogos de futebol.

2 Arquitetura da Aplicação

Esta aplicação segue o modelo MVC (Model View Control), onde uma classe esta encarregada da comunicação com o usuário, a View, outra esta encarregada do processamento da informação, o Model, e uma terceira, o Control, que faz a "ponte" entre estas duas, isto é, recebe a informação do usuário, envia-a para o model e envia os resultados do model para o View para serem mostrados.

Este padrão de arquitetura foi implementado com recurso à utilização de *Interfaces* de modo a permitir ocultar a implementação de cada componente. Assim, no futuro será possível por exemplo:

1. alterar a view, por exemplo substituindo a interface textual por uma interface gráfica, bastando para isso implementar uma nova classe (com a referida interface gráfica) que implemente a interface *IView* e, por consequência, os respetivos métodos;

2. alterar o model, por exemplo substituindo a atual lógica de negócio centrada no futebol, por uma outra lógica de negócio relacionada com um outro desporto. Para tal, bastará desenvolver uma nova classe que implemente a interface *IModel*;

3. alterar o control, em função de alguma (ou ambas) das alterações referidas acima, de modo a conseguir responder às novas funcionalidades que venham a ser implementadas. Neste caso, a interface que o novo control teria de implementar seria, naturalmente, a interface *IControl*.

Todas as três interfaces foram implementadas seguindo a seguinte estrutura de código:

- métodos abstratos: cuja implementação ficará a cargo das classes que implementem a interface;
- constantes: que passariam a estar disponíveis em qualquer classe que implemente a respetiva interface;
- métodos default: cuja implementação seria fornecida pela própria interface;
- métodos static: semelhantes aos anteriores, mas disponíveis a partir da própria interface;

2.1 Model

A classe Model contém a lógica de negócio. Assim, é dentro desta classe que reside o coração da aplicação e da funcionalidade que desta se espera.

É esta classe que chama depois todas as classes detalhadas na secção seguinte e que traduzem os conceitos que a aplicação pretendia modelar.

Em particular, as classes *Jogador*, e as suas várias especializações (tipos de jogador), bem como a classe *Equipa*

2.2 Control

Como dito anteriormente, o control é a classe "intermediária" da arquitetura MVC, este contem um objeto da classe View e outro da class Model, e faz a ligação entre os mesmos.

Esta classe é também responsável por gerir as tarefas de input e output, tendo para tal um conjunto de métodos que permitem receber do utilizador informação e devolver ao utilizador as respostas adequadas.

2.3 View

Por último, na classe view são incluídos todo um conjunto de conteúdos gráficos sob a forma de menus. De facto, sendo a aplicação baseada numa interface textual, os elementos gráficos a mostrar ao utilizador final são essencialmente linhas de texto.

No entanto, de modo a dar algum aspeto dinâmico à aplicação, procurou-se que os menus fossem variando e se fossem adaptando a cada momento de interação com o utilizador.

3 Classes

Nesta secção detalha-se um pouco as principais classes que constituem a aplicação desenvolvida.

Para todas as classes, seguindo as boas práticas aprendidas ao longo do semestre, procurou-se o mais possível seguir a seguinte estruturação do código:

- variáveis de instância: disponíveis a qualquer instância da respetiva classe;
- variáveis de classe: que passariam a estar disponíveis diretamente a partir da classe;

- construtores: pelo menos os seguintes três foram considerados havendo, no entanto, margem para a consideração de outros mais específicos caso necessário.

  vazio

  paramétrico

  cópia

- getters e setters: responsáveis por ler ou escrever as/nas variáveis de instância. Deste modo garantia-se desde logo o encapsulamento de cada classe;

- métodos override: a implementar sempre que se pretenda substituir algum método cuja implementação exista por defeito. Nesta categoria inserem-se, por exemplo, os métodos *toString*, *equals* e *clone*;

- métodos abstratos: a implementar obrigatoriamente caso se esteja a implementar alguma interface;

- métodos específicos: cuja implementação só faz sentido no contexto de instâncias da respetiva classe;

- métodos static: semelhantes aos anteriores, mas disponíveis a partir da própria classe e não de instâncias suas;

3.1 Jogador(es)

Uma vez que os vários tipos de jogadores apenas diferem em algumas variáveis de instância, optou-se por recorrer a uma superclasse *Jogador* onde tudo o que fosse comum aos demais jogadores ficaria agrupado.

No entanto, pelo facto de que cada tipo de jogador tem uma forma distinta de quantificar a sua habilidade, a superclasse *Jogador* foi tornada abstrata. O seu principal método abstrato era precisamente o método *habilidade*, que seria calculado de forma distinta para cada tipo de jogador.

Na Tabela 1 apresentam-se os pesos relativos que foram considerados para calcular a habilidade de cada tipo de jogador. Destacam-se as três características especificas dos jogadores do tipo Guarda-redes, Médio e Lateral. Uma vez que estas classes traduziam funcionalidades especificas destes tipos de jogador, procurou-se que o peso relativo destas fosse substancialmente superior às demais.

Nos restantes jogadores, procurou-se dar mais importância às características mais relevantes para cada posição.

Table 1: Pesos adotados na quantificação da habilidade de cada tipo de jogador.

Jogador	Guarda-redes	Defesa	Médio	Avançado	Lateral
Velocidade	0.1	0.2	0.05	0.05	0.1
Resistência	0.1	0.2	0.05	0.05	0.1
Destreza	0.2	0.05	0.05	0.05	0.15
Impulsão	0.2	0.2	0.1	0.2	0.1
Cabeça	0	0.2	0.1	0.2	0.1
Remate	0	0.05	0.2	0.4	0.1
Passe	0.1	0.1	0.05	0.05	0.1
Elasticidade	0.3	-	-	-	-
Cruzamento	-	-	-	-	0.25
Recuperação	-	-	0.4	-	-

Em termos de implementação, para além das variáveis de instância visíveis na Tabela 1, foi ainda considerada uma variável para guardar o estado possível de cada jogador ao longo do jogo (*SUPLENTE, TITULAR, SAINDO, ENTRANDO*) e uma outra, do tipo lista, com o historial de equipas por onde o jogador passou.

Por último, foi também considerada uma variável de classe que guardaria um contador dos jogadores existentes no sistema. Deste modo garante-se que não existem dois jogadores iguais no sistema

3.2 Equipa e Jogo

As restantes duas principais classes da aplicação eram a Equipa e o Jogo. Contrariamente à classe Jogador, nestas não foi considerado o uso de um contador global uma vez que se pressupõe que não existem duas equipas com o mesmo nome, no caso da classe Equipa, nem dois jogos exatamente iguais, no caso da classe Jogo.

Para além das necessárias variáveis de instância, e respetivos métodos getters e setters, estas classes implementaram algumas das funcionalidades especificas de cada uma.

No caso da Equipa, foi implementado um método *habilidade* semelhante ao existente para os Jogadores. Na verdade, é um método que se limita a chamar o método homónimo em cada jogador da equipa e soma o total da habilidade de todos os jogadores.

3.3 Exceções

Tendo em conta as funcionalidades implementadas na aplicação, foram consideradas essencialmente dois tipos de exceções. Tanto para o caso dos jogadores como para as equipas, essas exceções correspondem aos cenários em que ou a instância (de jogador/equipa) que está a ser solicitada pelo utilizador não existe no sistema ou já existe uma cópia da mesma pelo que não é possível criar uma nova instância igual.

4 Conclusão

O desenvolvimento do presente trabalho revelou-se um desafio interessante para o grupo. Foi possível experienciar alguns dos principais problemas que surgirão, no futuro profissional, num projeto semelhante. A aplicação das técnicas e metodologias aprendidas ao longo do semestre revelaram que, efetivamente, ter hábitos de boas práticas de programação e dividir o problema maior em pequenos problemas mais pequenos permitem abordar com sucesso o desenvolvimento de aplicações.

Chegados a esta fase das conclusões do trabalho, e tendo a clara noção de que muito poderia ser melhorado, detalham-se de seguida alguns dos aspetos a melhorar numa próxima revisão da aplicação:

- desenvolver uma sistema de base de dados que, à medida que a aplicação fosse utilizada, permitisse ir guardando os dados de utilização da mesma. Assim, seria possível convergir para uma aplicação mais semelhante à que serviu de inspiração ao presente trabalho;
- conforme o trabalho foi idealizado, existem muitos parâmetros que tiveram de ser arbitrados. Isso poderia ser revisto numa próxima versão através do fornecimento de um ficheiro de logs mais completo que o atualmente existente;
- apesar de se terem previsto e tratado algumas das possíveis exceções, existem muitas outras que não foram abordadas que seguramente beneficiariam a aplicação caso o fossem no futuro;

Anexo - Diagrama de classes

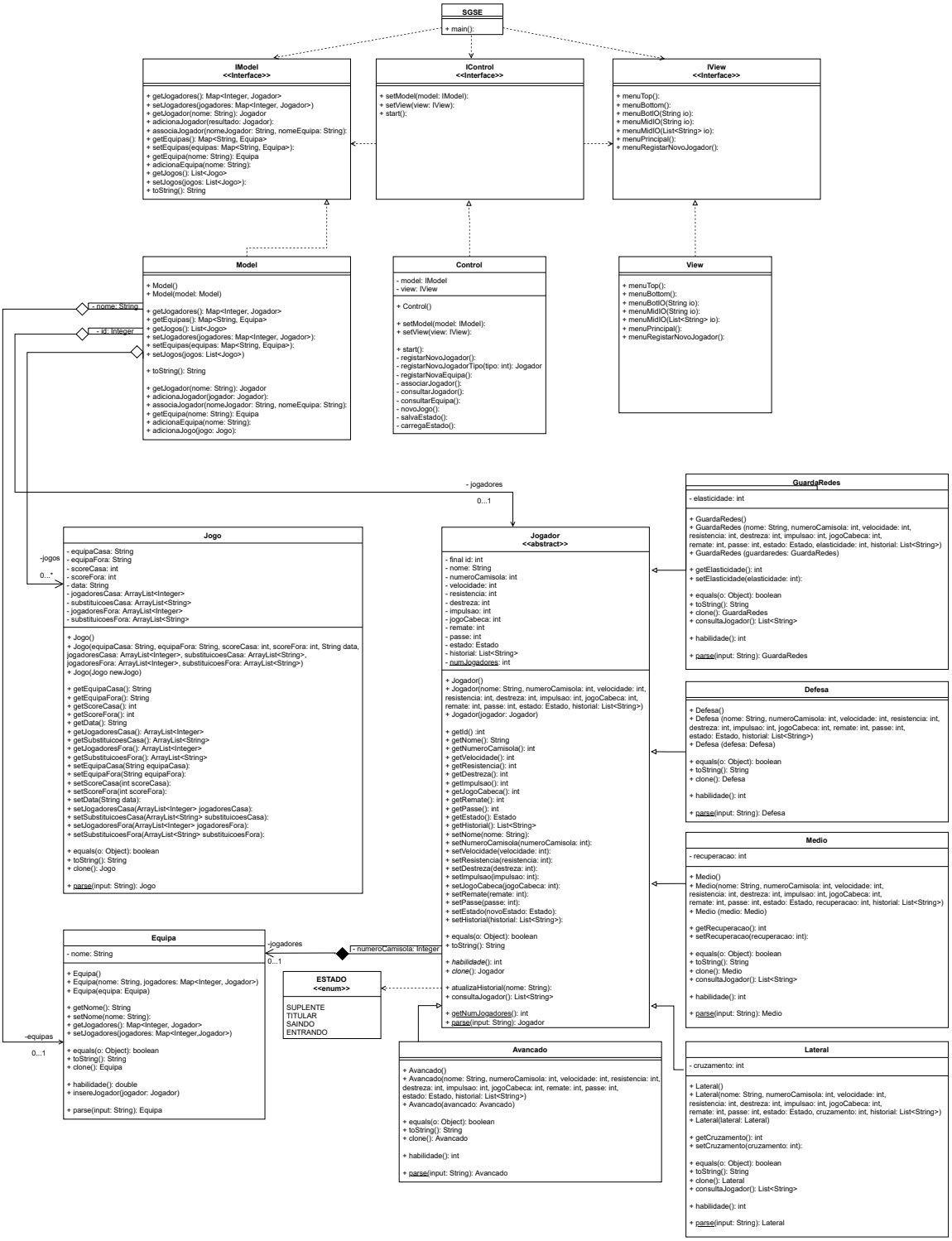


Figure 2: Diagrama de classes do projeto.