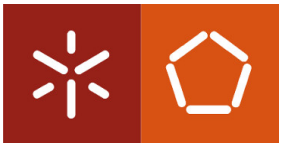
Desenvolvimento de Sistemas de Software

Racing Manager

LEI - 2022/2023

Grupo 19

Hugo Novais - a96267
Telmo Oliveira - a97018
Afonso Bessa - a95225
Martim Ribeiro - a96113



Universidade do Minho

Índice

List of Figures

1 Introdução

No âmbito da Unidade Curricular de Desenvolvimento de Sistemas de *Software* foi-nos proposto a implementação de um sistema capaz de simular Campeonatos de Automobilismo.

A primeira fase consistiu na análise dos cenários apresentados, através dos quais definimos o Modelo de Domínio e os Modelos de Use Cases.

Nesta segunda fase, para garantir a coerência global do procedimento, foi necessário fazer uma reavaliação de todo o trabalho realizado na primeira fase. Para além disso, construiu-se e definiu-se os subsistemas e os seguintes diagramas:

- Diagrama de Componentes;
- Diagrama de *Packages*;
- Diagramas de Atividades;
- Diagramas de Classes;
- Diagramas de Sequência.

O presente relatório tem por objetivo demonstrar as alterações feitas, bem como, apresentar todos os Diagramas anteriormente mencionados.

2 Descrição da Abordagem Utilizada

Em primeiro lugar, foi feita uma análise detalhada do trabalho realizado na fase anterior, de moda a corrigir situações que não tinham sido previstas ou às quais não tinha sido atribuído igual foco e atenção. Posteriormente foi necessário, tendo em conta os Use Cases definidos previamente para o Cenário 5 e o modelo de Domínio, fazer o levantamento das responsabilidades do sistema e a definição dos respetivos métodos (*API* da Lógica de Negócio), mostrando-se imprescindível a criação de vários subsistemas pelos quais estes métodos seriam distribuídos.

De seguida, foi levada a cabo a criação de um Diagrama de Classes, definindo atributos e classes, bem como, as relações entre essas mesmas classes. O facto deste diagrama ser navegável, facilitou o processo de desenvolvimento de Diagramas de Sequência que, por sua vez, representam as trocas de mensagens e a "interação" entre os vários objetos. Foi definido um Diagrama de Sequência para cada método, levantado para possibilitar a obtenção de uma visão das várias funcionalidades do programa.

Subsequentemente, criamos o Diagrama de Componentes no qual foram representados os vários subsistemas que o grupo considerou enquanto apropriados.

Por último, desenvolvemos o Diagrama de Packages e os Diagrama de Atividades.

Nota:

Todos estes Diagramas foram desenvolvidos usando a ferramenta *Visual Paradigm* lecionada nas aulas.

3 Alterações à 1ª Fase

3.1 Modelo de Domínio - Fase1

Com o fornecimento do *ZIP* com código na Linguagem Java e após a observação dos aspetos positivos e negativos apresentados pelo Docente nas aulas teóricas, pareceu-nos clara a necessidade de alterar o Modelo de Domínio entregue na primeira fase de maneira a aprimorar e aperfeiçoar o nosso projeto e corrigir alguns defeitos e inconsistências. O nosso modelo de domínio sofreu então bastante alterações em relação à primeira fase, uma vez que com a realização dos diferentes diagramas de classe e de sequência tornou-se bastante perceptível os defeitos do modelo inicial.

Modelo Antigo:

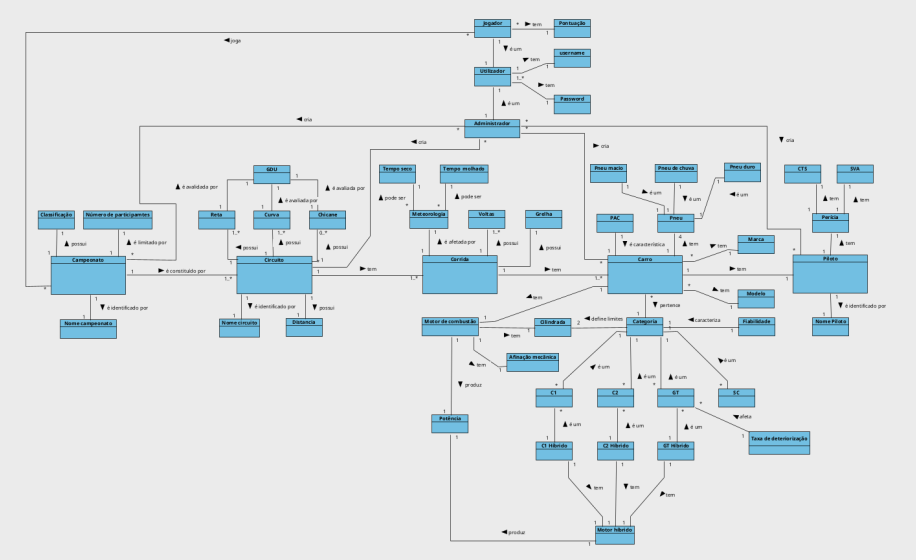


Figura 1: *Modelo de domínio da primeira fase do projeto.*

Inicialmente, acrescentamos o Atributo DNF que demonstra se um Carro acabou ou não uma Corrida. Dessa maneira, numa Corrida podem existir zero ou mais carros que não acabaram e um Carro pode ou não ter acabado a Corrida.

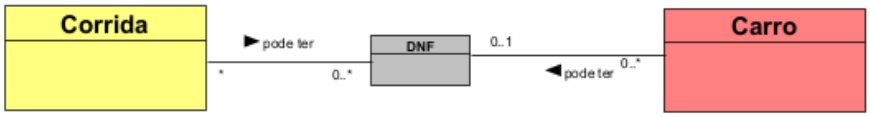


Figura 2: *Atributo DNF.*

Posteriormente, após bastante debate de ideias por parte dos elementos do grupo, decidimos alterar a Entidade Circuito e Corrida. Na primeira entrega ficou definido que um Campeonato era composto por um ou mais Circuitos, que um Circuito possuía uma ou mais Corridas e uma Corrida era composta por 2 ou mais Carros. Após as alterações efetuadas, o resultado final foi um Campeonato que tem uma ou mais Corridas, uma Corrida é feita num Circuito e uma Corrida é composta por 2 ou mais Carros.

Acreditamos que esta nova ideologia de Carro e Corrida retrata melhor o que se pretende no Projeto.

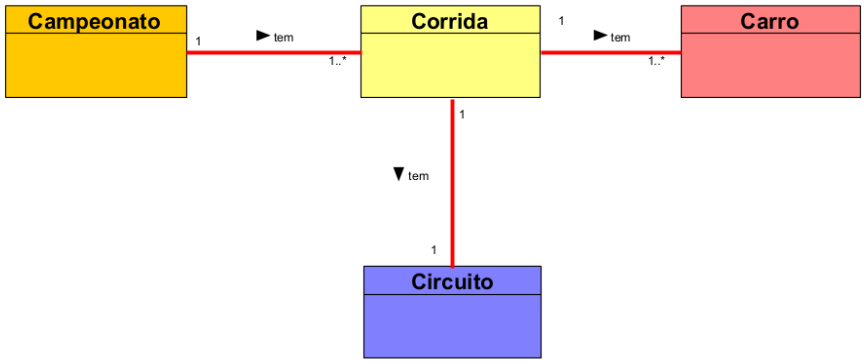


Figura 3: *Representação atual da relação Campeonato, Corrida, Carro e Circuito.*

A Entidade Carro foi a que sofreu uma maior alteração. Inicialmente, idealizamos que um Carro pertenceria a uma Categoria e essa mesma teria os Carros C1/C2/SC/GT, Cilindrada, Fiabilidade, Motor de Combustão, entre outros. Ao averiguarmos novamente com maior atenção, constatamos que tínhamos demasiada repetição de atributos, como, por exemplo, Motor de Combustão.

Para simplificarmos e tornar o nosso pensamento o mais objetivo possível, foi retirada a Entidade Categoria, bem como, o Atributos Motor de Combustão, significando que se um Carro é C1/C2/SC/GT este tem subentendido um Motor de Combustão. Tendo sido adicionada diretamente à Entidade Carro Cilindrada, Fiabilidade, Potência e Afinação Mecânica. Foi possível continuar a manter os Motor Híbrido, uma vez que esses podem ou não existir, sendo relevante a sua presença.

Por fim nesta Entidade acrescentamos também os Modos do Motor: Normal, Agressivo ou Conversador.

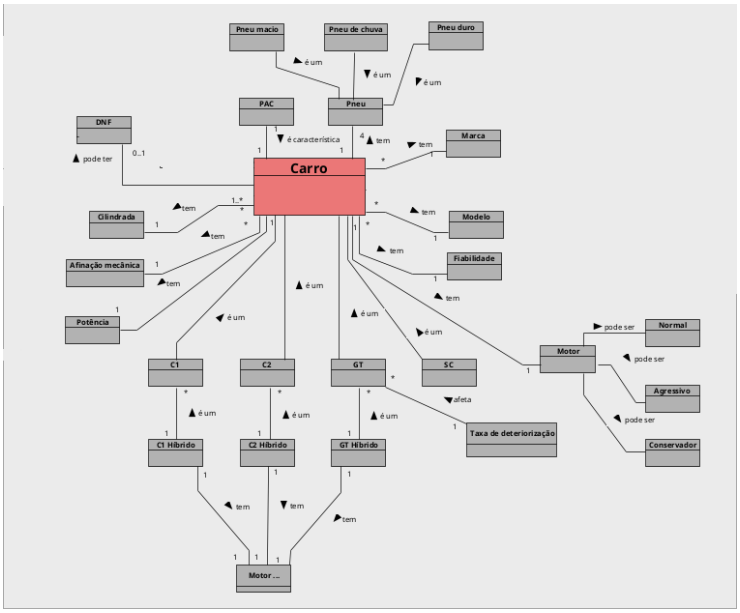


Figura 4: *Entidade final do carro.*

Por fim, acrescentamos alguns Atributos ao Modelo de Domínio de modo a obtermos informações e estatísticas sobre os resultados obtidos: - O Utilizador sabe qual é a sua posição no Ranking;

- O Campeonato possui a tabela de pontos a atribuir no final de cada Campeonato, bem como, a Classificação Final. Dispõem também um identificador para saber qual a Corrida que está a acontecer durante o Campeonato;

- Em cada Corrida é possível saber a melhor volta.

3.2 Modelo de Domínio - Fase 2

De modo a tornar o Modelo de Domínio mais legível e de fácil compreensão utilizamos cores identificatórias para as Entidades mais revelantes, como por exemplo, a cor vermelha para a Entidade Carro ou a cor rosa para o Jogador.

Modelo Atual:

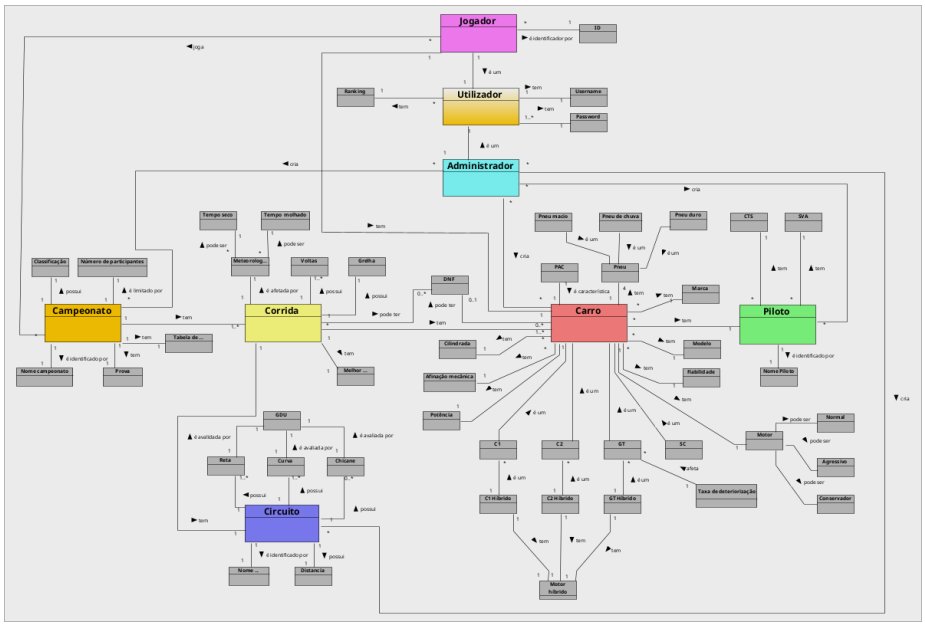


Figura 5: *Modelo de Domínio atual.*

4 Use Cases

Quanto aos Use Cases integrados na Fase 1, estes também foram sujeitos a pequenas alterações, à medida que melhor foi sendo analisado o pretendido, acabando por serem corrigidos alguns lapsos cometidos. Para a sua complementação foram elaboradas as seguintes tabelas, cujo papel é crucial para uma implementação simplificada, uma vez que nelas já se incluem todos os métodos a serem implementados em cada Use Case, assim como, a sua ordem, facilitando assim a criação dos restantes diagramas.

4.1 Começar Campeonato

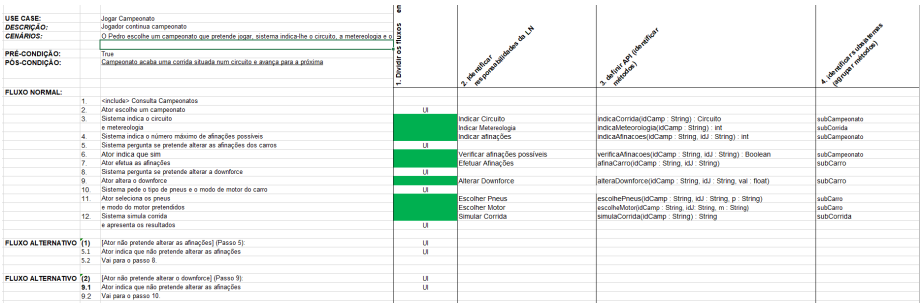
Este Use Case trata de Começar um Campeonato. Primeiramente, o ator escolhe o Campeonato que pretende começar e, logo de seguida, indica o número e nome dos Jogadores que irão participar no mesmo. Escolhe, da mesma maneira, o piloto e o carro de cada Jogador. Finalmente, após efetuadas as confirmações, um novo Campeonato é iniciado.

			1. Divisão dos fluxos em	2. Identificar responsabilidades da LN	3. Definir API (identificar métodos)	4. Atribuir responsabilidades (definição métodos)
USE CASE:		Começar Campeonato				
DESCRIÇÃO:		Jogador começa novo campeonato				
CENÁRIOS:		O Pedro decide começar um campeonato. Indica o número de jogadores e				
PRÉ-CONDIÇÃO:		Ator que começa campeonato está autenticado				
PÓS-CONDIÇÃO:		Sistema inicia um campeonato e os jogadores estão registados				
FLUXO NORMAL:						
	1.	<include> Consulta Campeonatos				
	2.	Ator escolhe campeonato	UI			
	3.	Sistema pede número e nome de jogadores	UI			
	4.	Ator fornece número		Receber Numero	numeroJogadores(n : int)	subCampeonato
		e nome de jogadores		Receber Nome	nomeJogadores() : List<String>)	subCampeonato
	5.	Sistema pede piloto e carros dos jogadores	UI			
	6.	Jogadores escolhem piloto e carro		Receber Piloto	escolhePiloto(idJ : String, idCamp : String, p : Piloto)	subCampeonato
		e carro		Receber Carro	escolheCarro(idJ : String, idCamp : String, c : Carro)	subCampeonato
	7.	Sistema exibe as escolhas feitas e pede confirmação para iniciar	UI			
	8.	Jogador responde que sim e novo campeonato é iniciado	UI			
	9.	Sistema inicia um campeonato e os jogadores estão registados		Iniciar Campeonato	novoCampeonato(newCamp : Campeonato)	subCampeonato
FLUXO ALTERNATIVO	1)	[Ator responde que não] (passo 9)	UI			
	9.1	Sistema verifica que a resposta do Ator foi não	UI			
	9.2	Retorna a 2				

Figura 6: *Excel do Use Case Começar Campeonato.*

4.2 Jogar Campeonato

Este Use Case trata de Jogar um Campeonato. Numa primeira fase, o ator escolhe um Campeonato, de seguida é apresentado o circuito, a meteorologia e o número de afinações disponíveis. O ator tem então a possibilidade de afinar o carro e de alterar o *downforce*. Por último, seleciona os pneus e o modo do motor pretendidos e ao confirmar a corrida é simulada e os resultados são apresentados.



			1. Dividir os fluxos em	2. Identificar responsabilidades da LN	3. Definir API (identificar métodos)	4. Identificar subsistemas (agrupar métodos)
USE CASE:		Jogar Campeonato				
DESCRIÇÃO:		Jogador continua campeonato				
CENÁRIOS:		O Pedro escolhe um campeonato que pretende jogar, sistema indica-lhe o circuito, a metereologia e o				
PRÉ-CONDIÇÃO:		True				
PÓS-CONDIÇÃO:		Campeonato acaba uma corrida situada num circuito e avança para a próxima				
FLUXO NORMAL:						
	1.	<include> Consulta Campeonatos				
	2.	Ator escolhe um campeonato	UI			
	3.	Sistema indica o circuito		Indicar Circuito	indicaCorrida(idCamp : String) : Circuito	subCampeonato
		e metereologia		Indicar Metereologia	indicaMeteorologia(idCamp : String) : int	subCorrida
	4.	Sistema indica o número máximo de afinações possíveis		Indicar afinações	indicaAfinacoes(idCamp : String, idJ : String) : int	subCampeonato
	5.	Sistema pergunta se pretende alterar as afinações dos carros	UI			
	6.	Ator indica que sim		Verificar afinações possíveis	verificaAfinacoes(idCamp : String, idJ : String) : Boolean	subCampeonato
	7.	Ator efetua as afinações		Efetuar Afinações	afinaCarro(idCamp : String, idJ : String)	subCarro
	8.	Sistema pergunta se pretende alterar a downforce	UI			
	9.	Ator altera o downforce		Alterar Downforce	alteraDownforce(idCamp : String, idJ : String, val : float)	subCarro
	10.	Sistema pede o tipo de pneus e o modo de motor do carro	UI			
	11.	Ator seleciona os pneus		Escolher Pneus	escolhePneus(idCamp : String, idJ : String, p : String)	subCarro
		e modo do motor pretendidos		Escolher Motor	escolheMotor(idCamp : String, idJ: String, m : String)	subCarro
	12.	Sistema simula corrida		Simular Corrida	simulaCorrida(idCamp : String) : String	subCorrida
		e apresenta os resultados	UI			
FLUXO ALTERNATIVO	**(1)**	[Ator não pretende alterar as afinações] (Passo 5):	UI			
	5.1	Ator indica que não pretende alterar as afinações	UI			
	5.2	Vai para o passo 8.				
FLUXO ALTERNATIVO	**(2)**	[Ator não pretende alterar o downforce] (Passo 9):	UI			
	9.1	Ator indica que não pretende alterar as afinações	UI			
	9.2	Vai para o passo 10.				

Figura 7: *Excel do Use Case Jogar Campeonato.*

5 Diagrama de Componentes

O Diagrama de Componentes permite-nos identificar, em cada nível, o que se afigura necessário para construir o sistema e auxilia o reconhecimento das dependências entre as diferentes componentes. Além do mais, permite, em geral, uma maior organização do código, contribuindo para o encapsulamento e a legibilidade do código.

Para a realização deste projeto, identificamos três subsistemas principais, são eles: **SubCampeonato**, **SubCorrida** e **SubCarro**.

Sendo assim, construímos o seguinte Diagrama de Componentes:

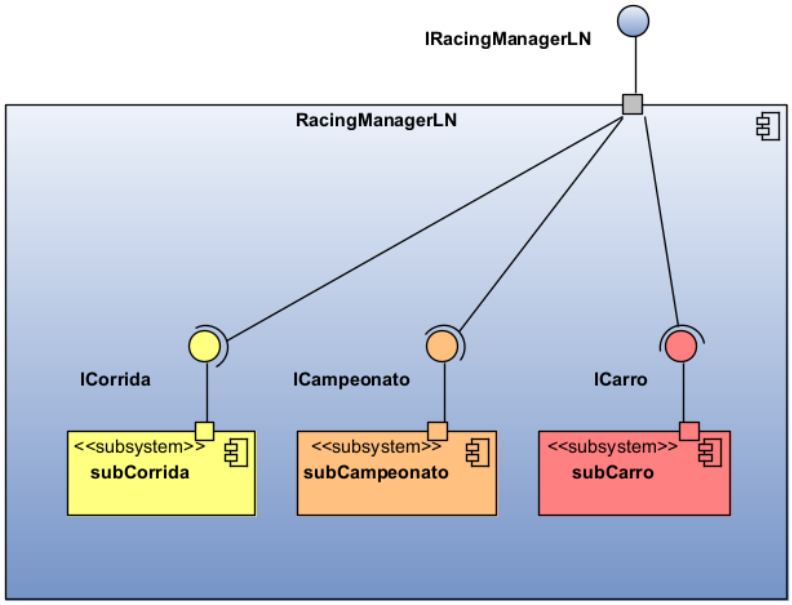


Figura 8: *Diagrama de Componentes.*

6 Diagrama de Packages

O Diagrama de Packages vai ao encontro das funcionalidades que o Diagrama de Componentes sugere, possibilitando a percepção de como subdividir o código quer a nível de classes, quer a nível de interfaces e também de métodos.

Sendo assim, construímos o seguinte Diagrama de Packages:

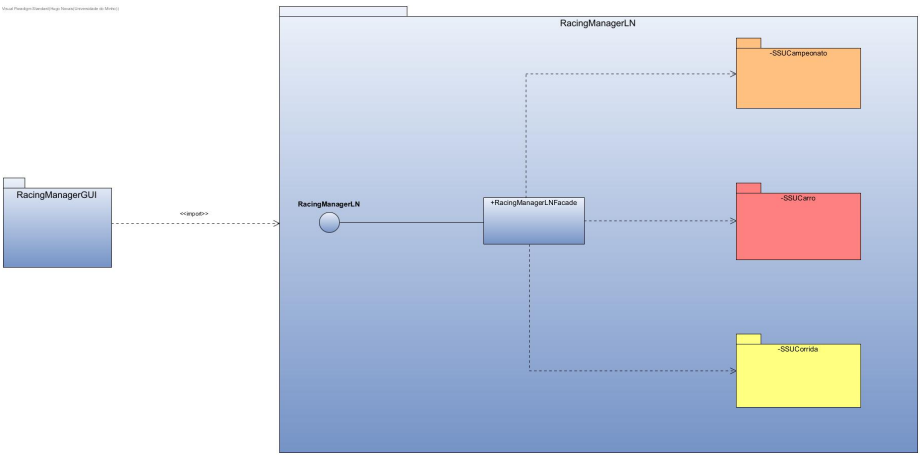


Figura 9: *Diagrama de Packages.*

7 Diagrama de Atividades

Os Diagramas de Atividade surgem para melhorar a compreensão do Enunciado e das funcionalidades que o nosso Programa prevê para o Cenário 5. Dessa maneira, foram elaborados três Diagramas de Atividades.

O primeiro corresponde ao processo de Desenvolvimento do Projeto, que demonstra de forma simples, as diversas fases/modelos elaborados pelo grupo de trabalho. Os restantes dois, especificam o comportamento no caso do Jogador JogarCampeonato e/ou ComeçarCampeonato.

Sendo assim, construímos os seguintes Diagramas de Packages:

7.1 Modelação do Projeto

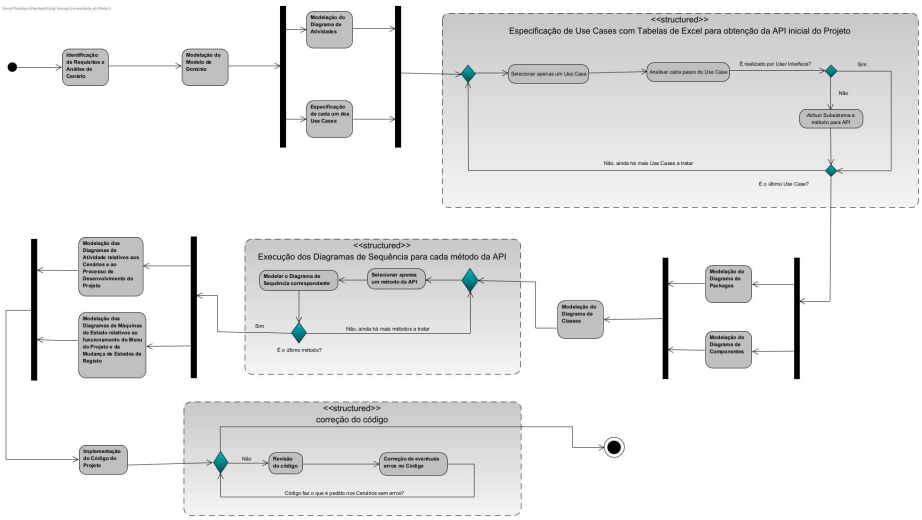


Figura 10: *Diagrama de Atividades da Modelação.*

7.2 JogarCampeonato

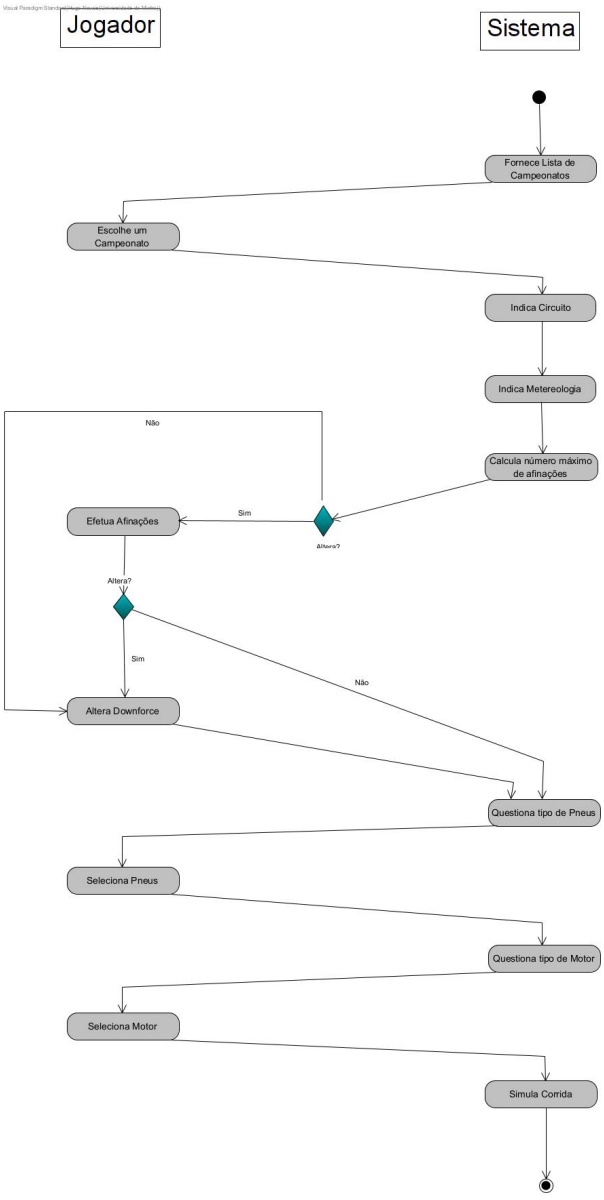


Figura 11: *Diagrama de Atividades do Use Case Jogar Campeonato.*

7.3 ComeçarCampeonato

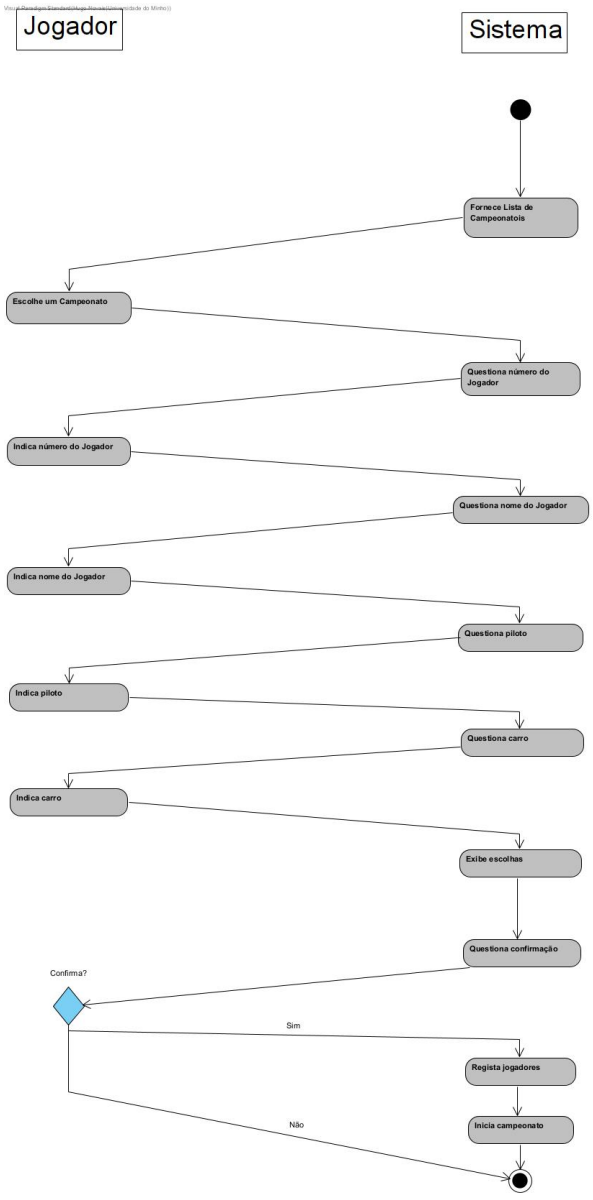


Figura 12: *Diagrama de Atividades do Use Case Começar Campeonato.*

8 Diagrama de Classes

O Diagrama de Classe mapeia de forma clara a estrutura de um determinado sistema ao modelar suas classes, seus atributos, operações e relações entre objetos. Desse modo, ajuda a entender melhor a visão geral dos esquemas de uma aplicação, fornecer uma descrição independente de implementação de tipos utilizados em um sistema e passados posteriormente entre seus componentes, entre outros.

8.1 *RacingManager*

Como seria de esperar, o Diagrama é bastante semelhante ao Modelo de Domínio, no entanto, realça-se a presença das *interfaces* que se encontram dentro da lógica de negócio, e que vêm das conclusões tiradas através dos Diagramas de Componentes e *Packages*.

Similarmente, os métodos utilizados são provenientes de uma análise dos Use Cases.

Existem também várias relações de composição e agregação, sendo que o *RacingManagerFacade* contém *Maps*/*Lists* de todos as entidades do sistema.

A utilização de *Facades* tem como objetivo servir de intermediário entre os diversos subsistemas dentro do sistema, bem como comunicar para fora dele.

Deste diagrama, sai uma conversão direta para Java, por ser uma linguagem orientada a objetos, o que orientará muito a fase de implementação.

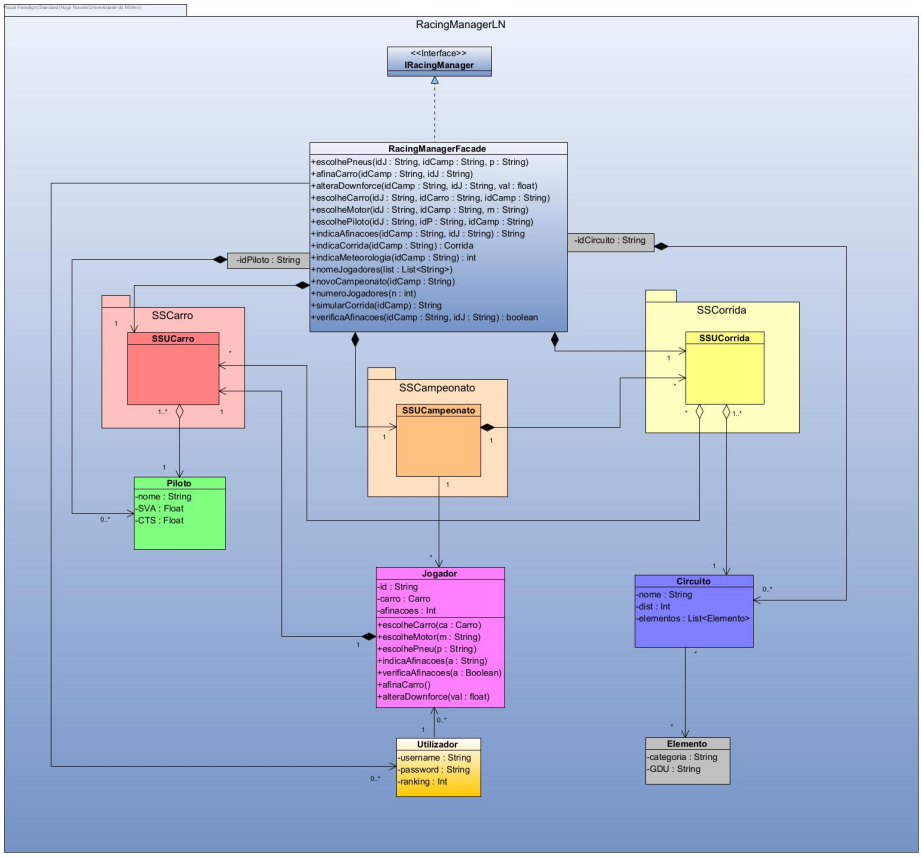


Figura 13: *Diagrama de Classes do RacingManagerLN.*

8.2 SSCampeonato

O Subsistema Campeonato apresenta uma enorme variedade de métodos relacionados com a base do jogo, desde as diferentes alterações possíveis de se fazer ao longo das Corridas do campeonato, como também o método *simularCorrida()*, que tal como o nome indica trata-se do método responsável pelas simulações das corridas entre os Jogadores.

De notar que um Campeonato é identificado por um *idCamp* e o mesmo está ligado a Corridas e a Jogadores uma vez que os seus atributos requerem estas Entidades.

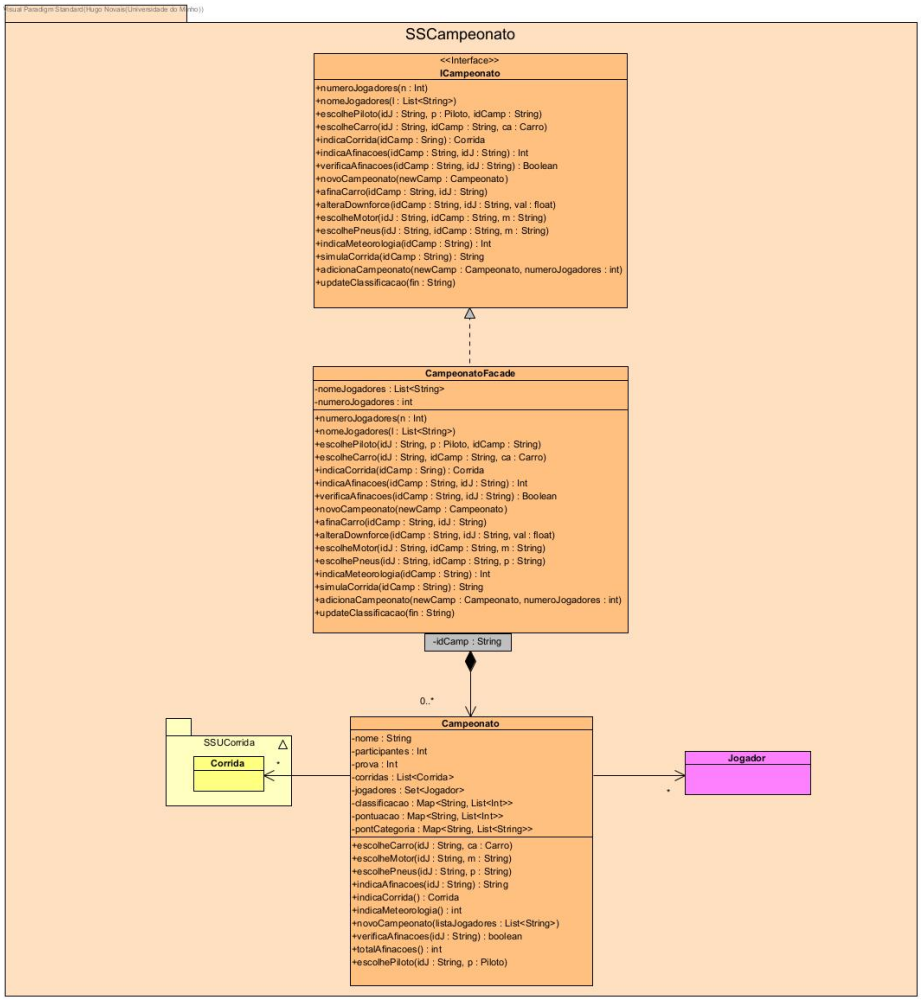


Figura 14: *Diagrama de Classes do Subsistema Campeonato.*

8.3 SSCorrida

O Subsistema Corrida representa as diferentes Corridas que um Campeonato possui, nela são guardados desde as voltas da mesma, até aos tempos respetivos de cada jogador.

A Corrida está ligado a Carros e a Circuitos uma vez que os seus atributos requerem estas Entidades.

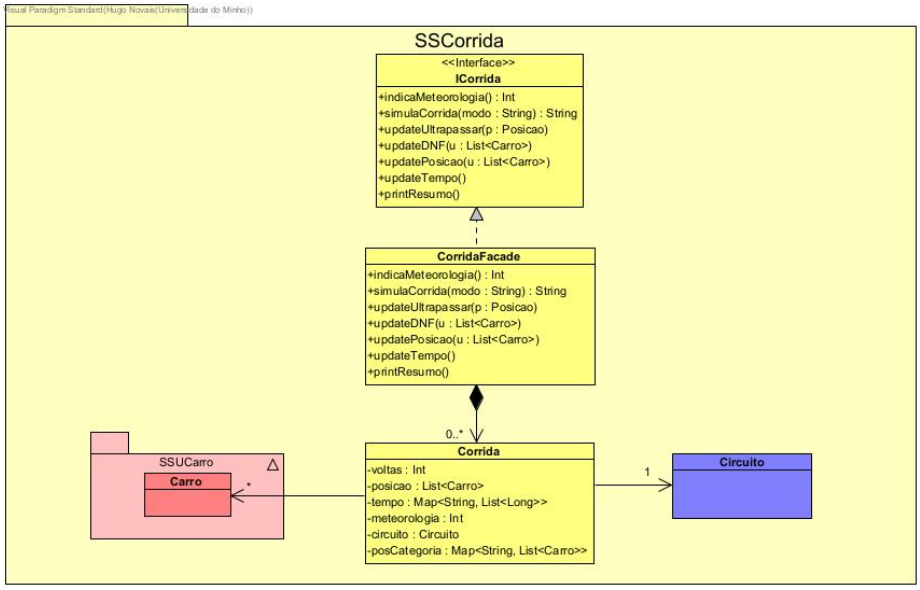


Figura 15: *Diagrama de Classes do Subsistema Corrida.*

8.4 SSCarro

Por último, o Subsistema Carro, este pode ser modificado nos diferentes Campeonatos, podendo assumir diferentes tipos de motor e pneus, contribuindo assim para uma maior diversidade de Corridas e de resultados.

O Carro está ligado a um Piloto os seus atributos requerem estas Entidades.

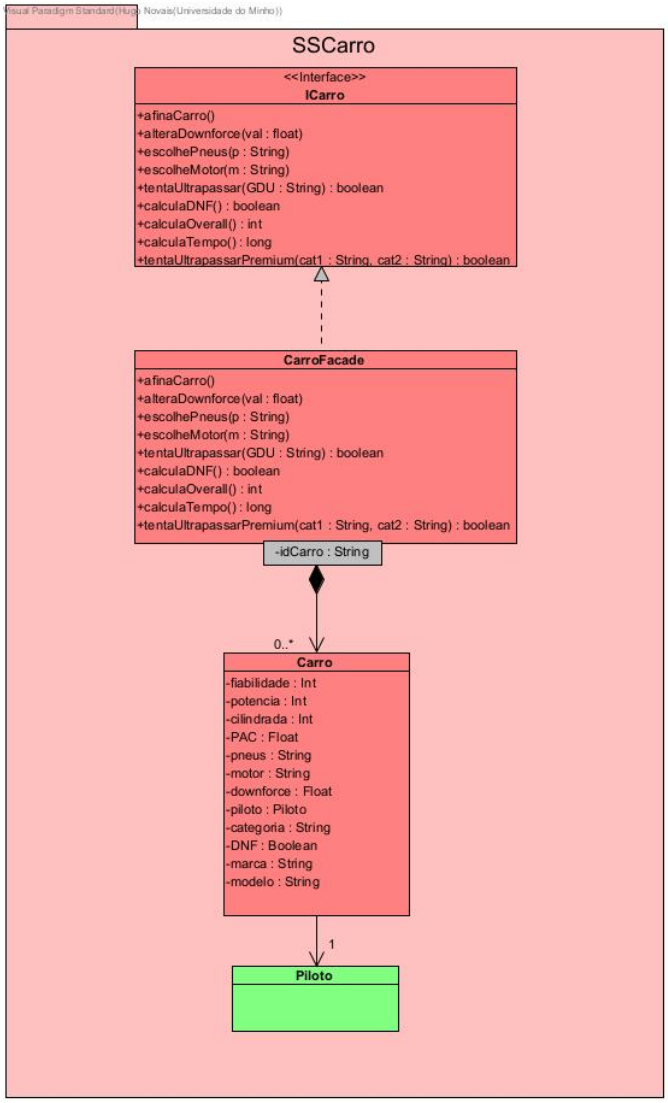


Figura 16: *Diagrama de Classes do Subsistema Carro.*

9 Diagramas de Sequência

Os Diagramas de Sequência representam as interações entre objetos respeitando uma certa ordem. Estes permitem-nos analisar a distribuição de "responsabilidades" pelas diferentes entidades do sistema, assim contribuindo para um melhor entendimento de como é efetuado todo o processo, seguindo uma ordem temporal.

Tendo como base os Use Cases apresentados anteriormente relativos ao Cenário 5, foram criados os seguintes Diagramas de Sequência das funções que constituem os mesmos.

9.1 *afinaCarro*

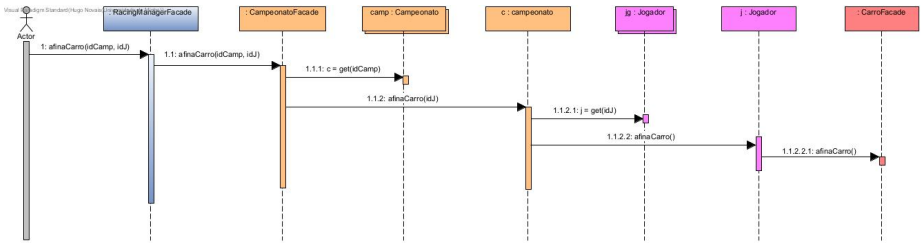


Figura 17: *Diagrama de Sequência da função afinaCarro().*

9.2 *alteraDownforce*

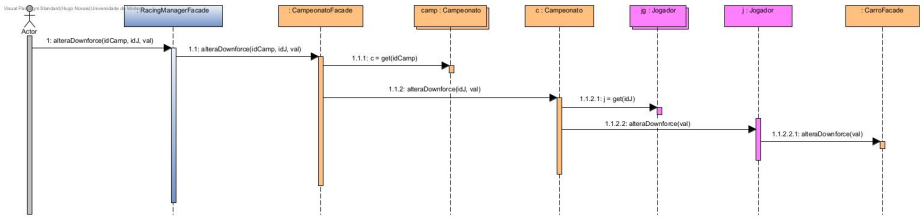


Figura 18: *Diagrama de Sequência da função alteraDownforce().*

9.3 *escolheCarro*

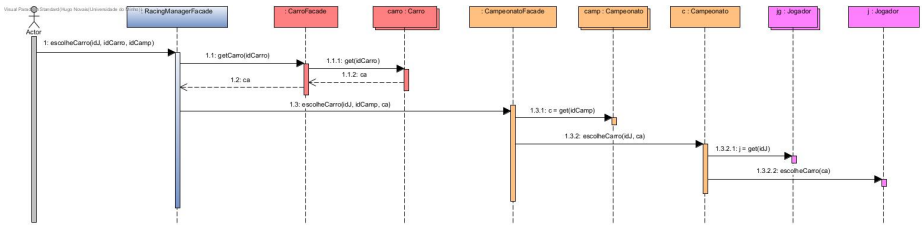


Figura 19: *Diagrama de Sequência da função escolheCarro().*

9.4 *escolheMotor*

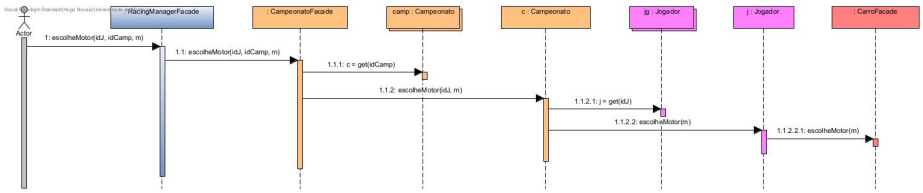


Figura 20: *Diagrama de Sequência da função escolheMotor().*

9.5 *escolhePiloto*

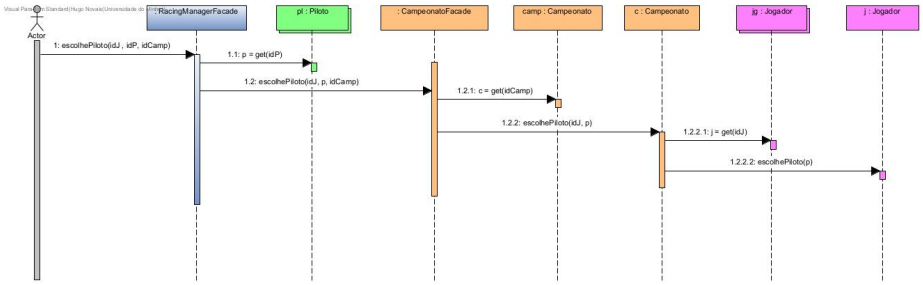


Figura 21: *Diagrama de Sequência da função escolhePiloto().*

9.6 *escolhePneus*

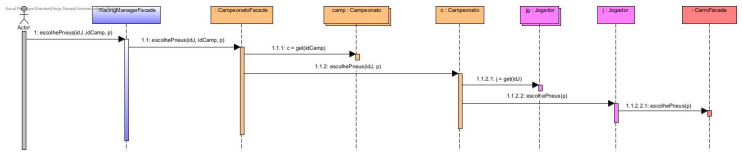


Figura 22: *Diagrama de Sequência da função escolhePneus().*

9.7 *indicaAfinacoes*

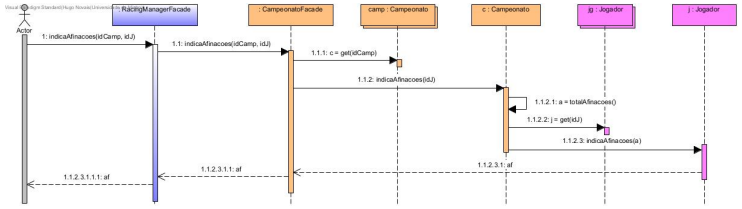


Figura 23: *Diagrama de Sequência da função indicaAfinacoes().*

9.8 *indicaCorrida*

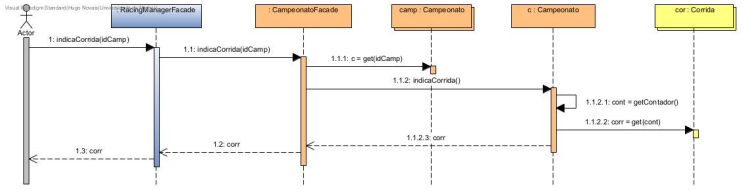


Figura 24: *Diagrama de Sequência da função indicaCorrida().*

9.9 *indicaMeteorologia*

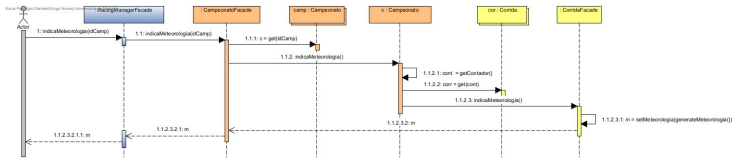


Figura 25: *Diagrama de Sequência da função indicaMeteorologia().*

9.10 *nomeJogadores*

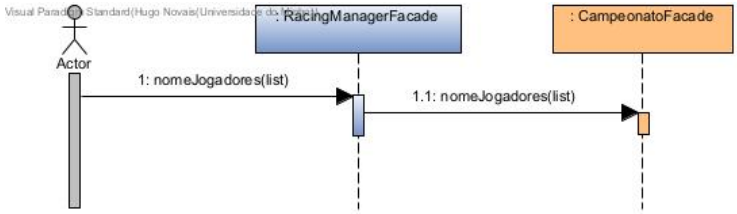


Figura 26: *Diagrama de Sequência da função nomeJogadores().*

9.11 *novoCampeonato*

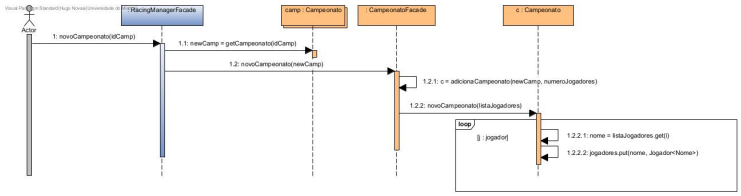


Figura 27: *Diagrama de Sequência da função novoCampeonato().*

9.12 *numeroJogadores*

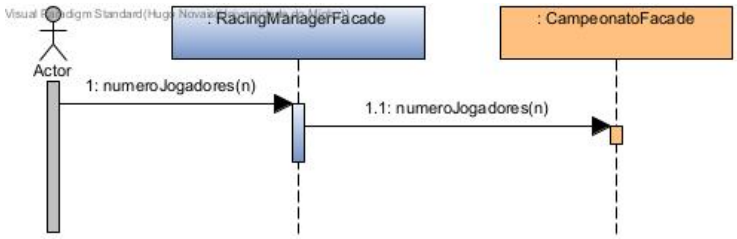


Figura 28: *Diagrama de Sequência da função numeroJogadores().*

9.13 *verificaAfinacoes*

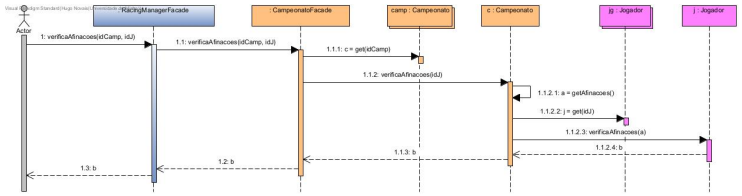


Figura 29: *Diagrama de Sequência da função verificaAfinacoes().*

9.14 *simulaCorrida*

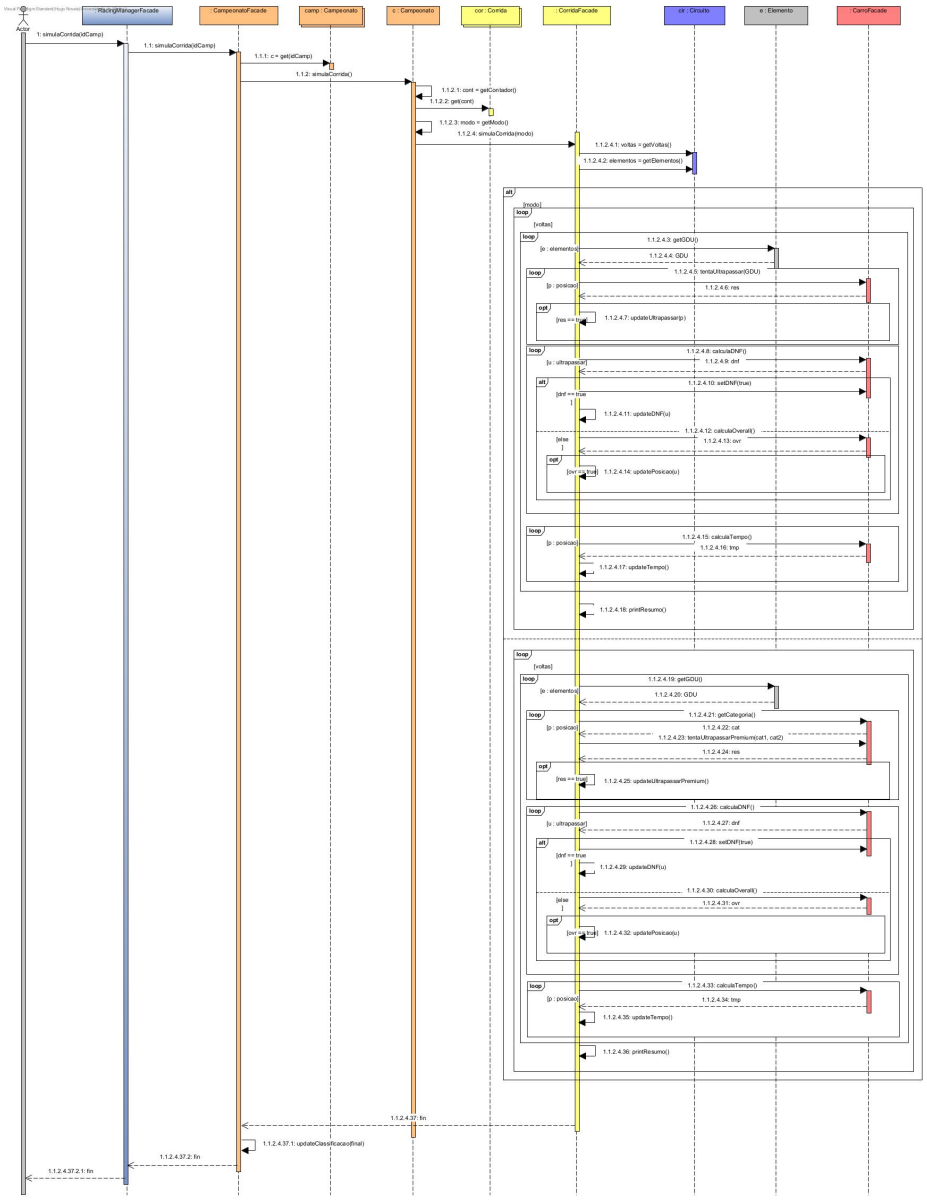


Figura 30: *Diagrama de Sequência da função simulaCorrida().*

10 Conclusão

Esta fase do projeto consistiu essencialmente na aplicação das metodologias lecionadas nas aulas teóricas de Desenvolvimento de Sistemas de Software. Isto permitiu, não apenas consolidar todo o conhecimento adquirido, mas também permitir ao grupo a aquisição de uma nova perspetiva de programação, não começando logo a criar código puro, mas antes a planear o trabalho faseadamente, possibilitando uma visão mais geral e concreta.

Assim, consideramos que o desenvolvimento faseado do projeto é bastante útil, ou seja, a definição de Use Cases, dos subsistemas, dos Diagramas de Classes e outros, permitem desenvolver progressivamente uma solução e analisar o trabalho de uma perspetiva distinta.

Durante a realização desta fase, encontramos algumas adversidades. Inicialmente, foram encontradas algumas falhas no Modelo de domínio da 1ª Fase, as quais foram corrigidas de imediato. Posteriormente, houve alguma dificuldade em definir bem os subsistemas, dado ser um Modelo de Domínio bastante complexo. Os Diagramas de Sequências também ocuparam uma boa parte do nosso tempo por necessitarem de muita atenção e trabalho, como é exemplo disso o *simulaCorrida*. Como é a primeira vez que todos os elementos do grupo frequentam a Unidade Curricular, houve também um período de adaptação ao *Visual Paradigm*.

Assim, o grupo considera que fez um excelente trabalho, uma vez que lhe foi dedicado muita atenção, trabalho e esforço, havendo um especial cuidado de manter a coerência e seguir uma linha realista. Por fim, podemos dizer que o planeamento e a modelação de um sistema de software é bastante importante para o seu desenvolvimento, uma vez que nos permite organizar e planear a aplicação antes do seu desenvolvimento concreto.